



ActionScript 2.0 语言参考

8

## 商标

1 Step RoboPDF、ActiveEdit、ActiveTest、Authorware、Blue Sky Software、Blue Sky、Breeze、Breezo、Captivate、Central、ColdFusion、Contribute、Database Explorer、Director、Dreamweaver、Fireworks、Flash、FlashCast、FlashHelp、Flash Lite、FlashPaper、Flash Video Encoder、Flex、Flex Builder、Fontographer、FreeHand、Generator、HomeSite、JRun、MacRecorder、Macromedia、MXML、RoboEngine、RoboHelp、RoboInfo、RoboPDF、Roundtrip、Roundtrip HTML、Shockwave、SoundEdit、Studio MX、UltraDev 和 WebHelp 是 Macromedia, Inc. 的注册商标或商标，可能已经在美国或其它管辖区乃至世界范围内注册。本出版物中提到的其它产品名称、徽标、图案、标题、文字或短语可能是 Macromedia, Inc. 或其它实体的商标、服务标志或商品名称，并且可能已经在特定的管辖区甚至世界范围内注册。

## 第三方信息

本指南包含指向第三方 Web 站点的链接，这些站点不由 Macromedia 控制，Macromedia 不对所链接的任何站点的内容负责。如果您访问本指南中所涉及的第三方 Web 站点，您必须自己承担由此带来的风险。Macromedia 提供这些链接只是为您提供方便。包含这些链接并不意味着 Macromedia 为这些第三方站点的内容提供担保或承担责任。

语音压缩和解压缩技术得到了 Nellymoser, Inc. ([www.nellymoser.com](http://www.nellymoser.com)) 的许可。

Sorenson™ Spark™ 视频压缩和解压缩技术得到了 Sorenson Media, Inc. 的许可。

Opera® 浏览器版权所有 © 1995-2002 Opera Software ASA 和其提供商。保留所有权利。

Macromedia Flash 8 视频采用了 On2 TrueMotion 视频技术。© 1992-2005 On2 Technologies, Inc. 保留所有权利。<http://www.on2.com>。

Mitsubishi Electric Research Laboratory: 本产品包含了来自此公司的软件，版权所有 © 2005，Mitsubishi Electric Research Laboratory Inc. 保留所有权利。<http://www.merl.com>。

**版权所有 © 2004-2005 Macromedia, Inc. 保留所有权利。未经 Macromedia, Inc. 书面许可，本手册及其任何部分都不准许拷贝、影印、复制、翻译或转换成任何电子形式或机器可读的形式。尽管有以上规定，与本手册一起提供的软件有效副本的所有者或授权用户可以从本手册的电子版本打印一份副本，该副本只能供该所有者或授权用户学习使用该软件之用，禁止对本手册的任何部分进行打印、复制、分发、转售或传送以用于其它任何目的，包括（但不限于）商业目的，如销售本文档的副本或提供有偿支持服务。**

## 鸣谢

项目管理：JuLee Burdekin

主要撰写人员：Francis Cheng、Robert Dixon、Shimul Rahim

其他撰写人员：Jen deHaan、Thais Derich、Guy Haas、David Jacowitz、Jeff Swartz

范例开发人员：Luke Bayes、Francis Cheng、Robert Dixon、Ali Mills、Jeff Swartz

编辑：Linda Adler、Geta Carlson、Evelyn Eldridge、John Hammett、Noreen Maher、Mark Nigara、Lisa Stanziano、Anne Szabla、Jessie Wood

制作管理：Patrice ONeill

媒体设计和制作：Adam Barnett、John Francis、Brett Jarvis

特别感谢：Peter deHaan、Gary Grossman、Lee Thomason、Yi Tan 和 Flash Player 核心团队

第一版：2005 年 9 月

Macromedia, Inc.  
601 Townsend St.  
San Francisco, CA 94103

# 目 录

第1章：ActionScript 语言元素 .....	31
编译器指令 .....	31
#endinitclip 指令 .....	31
#include 指令 .....	32
#initclip 指令 .....	33
常数 .....	35
false 常数 .....	35
Infinity 常数 .....	36
-Infinity 常数 .....	36
NaN 常数 .....	36
newline 常数 .....	36
null 常数 .....	37
true 常数 .....	38
undefined 常数 .....	39
全局函数 .....	40
Array 函数 .....	45
asfunction 协议 .....	46
Boolean 函数 .....	47
call 函数 .....	49
chr 函数 .....	49
clearInterval 函数 .....	50
duplicateMovieClip 函数 .....	50
escape 函数 .....	51
eval 函数 .....	52
fscommand 函数 .....	53
getProperty 函数 .....	57
getTimer 函数 .....	57
getURL 函数 .....	58
getVersion 函数 .....	59
gotoAndPlay 函数 .....	60
gotoAndStop 函数 .....	61
ifFrameLoaded 函数 .....	62
int 函数 .....	62
isFinite 函数 .....	63
isNaN 函数 .....	63
length 函数 .....	64
loadMovie 函数 .....	65

loadMovieNum 函数	67
loadVariables 函数	69
loadVariablesNum 函数	71
mbchr 函数	72
mblength 函数	73
mbord 函数	73
mbsubstring 函数	74
MMEecute 函数	74
nextFrame 函数	75
nextScene 函数	76
Number 函数	77
Object 函数	78
on 处理函数	79
onClipEvent 处理函数	80
ord 函数	81
parseFloat 函数	82
parseInt 函数	82
play 函数	83
prevFrame 函数	84
prevScene 函数	84
print 函数	85
printAsBitmap 函数	86
printAsBitmapNum 函数	87
printNum 函数	88
random 函数	89
removeMovieClip 函数	90
setInterval 函数	91
setProperty 函数	95
showRedrawRegions 函数	96
startDrag 函数	97
stop 函数	98
stopAllSounds 函数	98
stopDrag 函数	99
String 函数	99
substring 函数	100
targetPath 函数	101
tellTarget 函数	102
toggleHighQuality 函数	103
trace 函数	104
unescape 函数	105
unloadMovie 函数	105
unloadMovieNum 函数	106
updateAfterEvent 函数	107
全局属性	108
_accProps 属性	109



_focusrect 属性 .....	112
_global 属性 .....	113
_highquality 属性 .....	114
_level 属性 .....	114
maxscroll 属性 .....	115
_parent 属性 .....	115
_quality 属性 .....	116
_root 属性 .....	117
scroll 属性 .....	118
_soundbuftime 属性 .....	118
this 属性 .....	119
运算符 .....	121
+ 加法运算符 .....	125
+= 加法赋值运算符 .....	126
[] 数组访问运算符 .....	127
= 赋值运算符 .....	129
& 按位 AND 运算符 .....	130
&= 按位 AND 赋值运算符 .....	131
<< 按位向左移位运算符 .....	132
<<= 按位向左移位并赋值运算符 .....	133
~ 按位 NOT 运算符 .....	134
按位 OR 运算符 .....	135
= 按位 OR 赋值运算符 .....	136
>> 按位向右移位运算符 .....	137
>>= 按位向右移位并赋值运算符 .....	138
>>> 按位无符号向右移位运算符 .....	139
>>>= 按位无符号向右移位并赋值运算符 .....	140
^ 按位 XOR 运算符 .....	140
^= 按位 XOR 赋值运算符 .....	141
/*..*/ 注释块分隔符运算符 .....	142
, 逗号运算符 .....	143
添加连接（字符串）运算符 .....	144
?: 条件运算符 .....	145
-- 递减运算符 .....	146
/ 除法运算符 .....	146
/= 除法赋值运算符 .....	147
. 点运算符 .....	148
== 等于运算符 .....	149
eq 等于（字符串）运算符 .....	150
> 大于运算符 .....	151
gt 大于（字符串）运算符 .....	151
>= 大于或等于运算符 .....	152
ge 大于或等于（字符串）运算符 .....	152
++ 递增运算符 .....	153
!= 不等于运算符 .....	155

◇ 不等于运算符.....	156
instanceof 运算符.....	157
< 小于运算符.....	158
lt 小于（字符串）运算符.....	158
<= 小于或等于运算符.....	159
le 小于或等于（字符串）运算符.....	160
// 注释行分隔符运算符.....	160
&& 逻辑 AND 运算符.....	161
and 逻辑 AND 运算符.....	162
! 逻辑 NOT 运算符.....	162
not 逻辑 NOT 运算符.....	163
逻辑 OR 运算符.....	164
or 逻辑 OR 运算符.....	165
% 模运算符.....	165
%= 模赋值运算符.....	166
* 乘法运算符.....	167
*= 乘法赋值运算符.....	167
new 运算符.....	168
ne 不等于（字符串）运算符.....	169
{ } 对象初始值设定项运算符.....	169
() 括号运算符.....	171
=== 全等运算符.....	172
!== 不全等运算符.....	174
" 字符串分隔符运算符.....	175
- 减法运算符.....	175
-= 减法赋值运算符.....	176
: type 运算符.....	177
typeof 运算符.....	178
void 运算符.....	179
语句.....	179
break 语句.....	181
case 语句.....	182
class 语句.....	183
continue 语句.....	185
default 语句.....	186
delete 语句.....	187
do..while 语句.....	189
dynamic 语句.....	190
else 语句.....	191
else if 语句.....	192
extends 语句.....	193
for 语句.....	195
for..in 语句.....	196
function 语句.....	197
get 语句.....	198

if 语句 .....	200
implements 语句 .....	201
import 语句 .....	201
interface 语句 .....	202
intrinsic 语句 .....	204
private 语句 .....	205
public 语句 .....	206
return 语句 .....	207
set 语句 .....	208
set variable 语句 .....	209
static 语句 .....	210
super 语句 .....	211
switch 语句 .....	212
throw 语句 .....	213
try..catch..finally 语句 .....	214
var 语句 .....	218
while 语句 .....	219
with 语句 .....	220
 <b>第 2 章：ActionScript 类 .....</b>	<b>223</b>
Accessibility .....	223
isActive (Accessibility.isActive 方法) .....	224
updateProperties (Accessibility.updateProperties 方法) .....	225
arguments .....	226
callee (arguments.callee 属性) .....	227
caller (arguments.caller 属性) .....	227
length (arguments.length 属性) .....	227
Array .....	227
Array 构造函数 .....	230
CASEINSENSITIVE (Array.CASEINSENSITIVE 属性) .....	231
concat (Array.concat 方法) .....	232
DESCENDING (Array.DECENDING 属性) .....	233
join (Array.join 方法) .....	233
length (Array.length 属性) .....	234
NUMERIC (Array.NUMERIC 属性) .....	235
pop (Array.pop 方法) .....	235
push (Array.push 方法) .....	236
RETURNINDEXEDARRAY .....	
(Array.RETURNINDEXEDARRAY 属性) .....	236
reverse (Array.reverse 方法) .....	237
shift (Array.shift 方法) .....	237
slice (Array.slice 方法) .....	238
sort (Array.sort 方法) .....	239
sortOn (Array.sortOn 方法) .....	241
splice (Array.splice 方法) .....	245

toString (Array.toString 方法)	246
UNIQUESORT (Array.UNIQUESORT 属性)	246
unshift (Array.unshift 方法)	247
AsBroadcaster	247
addListener (AsBroadcaster.addListener 方法)	249
broadcastMessage (AsBroadcaster.broadcastMessage 方法)	250
initialize (AsBroadcaster.initialize 方法)	251
_listeners (AsBroadcaster._listeners 属性)	252
removeListener (AsBroadcaster.removeListener 方法)	253
BevelFilter (flash.filters.BevelFilter)	254
angle (BevelFilter.angle 属性)	257
BevelFilter 构造函数	258
blurX (BevelFilter.blurX 属性)	260
blurY (BevelFilter.blurY 属性)	261
clone (BevelFilter.clone 方法)	262
distance (BevelFilter.distance 属性)	264
highlightAlpha (BevelFilter.highlightAlpha 属性)	265
highlightColor (BevelFilter.highlightColor 属性)	266
knockout (BevelFilter.knockout 属性)	267
quality (BevelFilter.quality 属性)	268
shadowAlpha (BevelFilter.shadowAlpha 属性)	269
shadowColor (BevelFilter.shadowColor 属性)	270
strength (BevelFilter.strength 属性)	271
type (BevelFilter.type 属性)	272
BitmapData (flash.display.BitmapData)	273
applyFilter (BitmapData.applyFilter 方法)	277
BitmapData 构造函数	279
clone (BitmapData.clone 方法)	280
colorTransform (BitmapData.colorTransform 方法)	282
copyChannel (BitmapData.copyChannel 方法)	283
copyPixels (BitmapData.copyPixels 方法)	284
dispose (BitmapData.dispose 方法)	286
draw (BitmapData.draw 方法)	286
fillRect (BitmapData.fillRect 方法)	288
floodFill (BitmapData.floodFill 方法)	289
generateFilterRect (BitmapData.generateFilterRect 方法)	290
getColorBoundsRect (BitmapData.getColorBoundsRect 方法)	291
getPixel (BitmapData.getPixel 方法)	292
getPixel32 (BitmapData.getPixel32 方法)	293
height (BitmapData.height 属性)	294
hitTest (BitmapData.hitTest 方法)	294
loadBitmap (BitmapData.loadBitmap 方法)	296
merge (BitmapData.merge 方法)	296
noise (BitmapData.noise 方法)	297
paletteMap (BitmapData.paletteMap 方法)	299

perlinNoise (BitmapData.perlinNoise 方法)	300
pixelDissolve (BitmapData.pixelDissolve 方法)	302
rectangle (BitmapData.rectangle 属性)	304
scroll (BitmapData.scroll 方法)	304
setPixel (BitmapData.setPixel 方法)	305
setPixel32 (BitmapData.setPixel32 方法)	306
threshold (BitmapData.threshold 方法)	307
transparent (BitmapData.transparent 属性)	309
width (BitmapData.width 属性)	309
BitmapFilter (flash.filters.BitmapFilter)	310
clone (BitmapFilter.clone 方法)	311
BlurFilter (flash.filters.BlurFilter)	311
BlurFilter 构造函数	313
blurX (BlurFilter.blurX 属性)	314
blurY (BlurFilter.blurY 属性)	315
clone (BlurFilter.clone 方法)	316
quality (BlurFilter.quality 属性)	317
Boolean	318
Boolean 构造函数	319
toString (Boolean.toString 方法)	320
valueOf (Boolean.valueOf 方法)	320
Button	321
_alpha (Button._alpha 属性)	324
blendMode (Button.blendMode 属性)	324
cacheAsBitmap (Button.cacheAsBitmap 属性)	328
enabled (Button.enabled 属性)	329
filters (Button.filters 属性)	330
_focusrect (Button._focusrect 属性)	332
getDepth (Button.getDepth 方法)	332
_height (Button._height 属性)	333
_highquality (Button._highquality 属性)	334
menu (Button.menu 属性)	334
_name (Button._name 属性)	335
onDragOut (Button.onDragOut 处理函数)	335
onDragOver (Button.onDragOver 处理函数)	336
onKeyDown (Button.onKeyDown 处理函数)	336
onKeyUp (Button.onKeyUp 处理函数)	337
onKillFocus (Button.onKillFocus 处理函数)	338
onPress (Button.onPress 处理函数)	339
onRelease (Button.onRelease 处理函数)	339
onReleaseOutside (Button.onReleaseOutside 处理函数)	339
onRollOut (Button.onRollOut 处理函数)	340
onRollOver (Button.onRollOver 处理函数)	340
onSetFocus (Button.onSetFocus 处理函数)	340
_parent (Button._parent 属性)	341

_quality (Button._quality 属性) .....	342
_rotation (Button._rotation 属性) .....	342
scale9Grid (Button.scale9Grid 属性) .....	343
_soundbuftime (Button._soundbuftime 属性) .....	344
tabEnabled (Button.tabEnabled 属性) .....	344
tabIndex (Button.tabIndex 属性) .....	345
_target (Button._target 属性) .....	346
trackAsMenu (Button.trackAsMenu 属性) .....	347
_url (Button._url 属性) .....	348
useHandCursor (Button.useHandCursor 属性) .....	348
_visible (Button._visible 属性) .....	349
_width (Button._width 属性) .....	350
_x (Button._x 属性) .....	350
_xmouse (Button._xmouse 属性) .....	351
_xscale (Button._xscale 属性) .....	351
_y (Button._y 属性) .....	352
_ymouse (Button._ymouse 属性) .....	352
_yscale (Button._yscale 属性) .....	353
Camera .....	354
activityLevel (Camera.activityLevel 属性) .....	356
bandwidth (Camera.bandwidth 属性) .....	357
currentFps (Camera.currentFps 属性) .....	358
fps (Camera.fps 属性) .....	359
get (Camera.get 方法) .....	360
height (Camera.height 属性) .....	362
index (Camera.index 属性) .....	362
motionLevel (Camera.motionLevel 属性) .....	363
motionTimeOut (Camera.motionTimeOut 属性) .....	365
muted (Camera.muted 属性) .....	366
name (Camera.name 属性) .....	367
names (Camera.names 属性) .....	367
onActivity (Camera.onActivity 处理函数) .....	368
onStatus (Camera.onStatus 处理函数) .....	369
quality (Camera.quality 属性) .....	370
setMode (Camera.setMode 方法) .....	371
setMotionLevel (Camera.setMotionLevel 方法) .....	372
setQuality (Camera.setQuality 方法) .....	373
width (Camera.width 属性) .....	375
capabilities (System.capabilities) .....	375
avHardwareDisable (capabilities.avHardwareDisable 属性) .....	378
hasAccessibility (capabilities.hasAccessibility 属性) .....	379
hasAudio (capabilities.hasAudio 属性) .....	379
hasAudioEncoder (capabilities.hasAudioEncoder 属性) .....	379
hasEmbeddedVideo (capabilities.hasEmbeddedVideo 属性) .....	380
hasIME (capabilities.hasIME 属性) .....	380

hasMP3 (capabilities.hasMP3 属性) .....	380
hasPrinting (capabilities.hasPrinting 属性) .....	381
hasScreenBroadcast (capabilities.hasScreenBroadcast 属性) ..	381
hasScreenPlayback (capabilities.hasScreenPlayback 属性) ....	381
hasStreamingAudio (capabilities.hasStreamingAudio 属性) ....	382
hasStreamingVideo (capabilities.hasStreamingVideo 属性) ....	382
hasVideoEncoder (capabilities.hasVideoEncoder 属性) .....	382
isDebugger (capabilities.isDebugger 属性) .....	383
language (capabilities.language 属性) .....	383
localFileReadDisable (capabilities.localFileReadDisable 属性) .	384
manufacturer (capabilities.manufacturer 属性) .....	385
os (capabilities.os 属性) .....	385
pixelAspectRatio (capabilities.pixelAspectRatio 属性) .....	385
playerType (capabilities.playerType 属性) .....	386
screenColor (capabilities.screenColor 属性) .....	386
screenDPI (capabilities.screenDPI 属性) .....	386
screenResolutionX (capabilities.screenResolutionX 属性) .....	387
screenResolutionY (capabilities.screenResolutionY 属性) .....	387
serverString (capabilities.serverString 属性) .....	387
version (capabilities.version 属性) .....	388
Color .....	388
Color 构造函数 .....	389
getRGB (Color.getRGB 方法) .....	390
getTransform (Color.getTransform 方法) .....	390
setRGB (Color.setRGB 方法) .....	391
setTransform (Color.setTransform 方法) .....	392
ColorMatrixFilter (flash.filters.ColorMatrixFilter) .....	393
clone (ColorMatrixFilter.clone 方法) .....	396
ColorMatrixFilter 构造函数 .....	397
matrix (ColorMatrixFilter.matrix 属性) .....	397
ColorTransform (flash.geom.ColorTransform) .....	398
alphaMultiplier (ColorTransform.alphaMultiplier 属性) .....	400
alphaOffset (ColorTransform.alphaOffset 属性) .....	401
blueMultiplier (ColorTransform.blueMultiplier 属性) .....	402
blueOffset (ColorTransform.blueOffset 属性) .....	403
ColorTransform 构造函数 .....	404
concat (ColorTransform.concat 方法) .....	405
greenMultiplier (ColorTransform.greenMultiplier 属性) .....	406
greenOffset (ColorTransform.greenOffset 属性) .....	407
redMultiplier (ColorTransform.redMultiplier 属性) .....	408
redOffset (ColorTransform.redOffset 属性) .....	409
rgb (ColorTransform.rgb 属性) .....	410
toString (ColorTransform.toString 方法) .....	411
ContextMenu .....	411
builtInItems (ContextMenu.builtInItems 属性) .....	413

ContextMenu 构造函数 .....	414
copy (ContextMenu.copy 方法) .....	415
customItems (ContextMenu.customItems 属性) .....	416
hideBuiltInItems (ContextMenu.hideBuiltInItems 方法) .....	417
onSelect (ContextMenu.onSelect 处理函数) .....	417
ContextMenuItem .....	418
caption (ContextMenuItem.caption 属性) .....	420
ContextMenuItem 构造函数 .....	420
copy (ContextMenuItem.copy 方法) .....	421
enabled (ContextMenuItem.enabled 属性) .....	422
onSelect (ContextMenuItem.onSelect 处理函数) .....	423
separatorBefore (ContextMenuItem.separatorBefore 属性) .....	424
visible (ContextMenuItem.visible 属性) .....	425
ConvolutionFilter (flash.filters.ConvolutionFilter) .....	426
alpha (ConvolutionFilter.alpha 属性) .....	428
bias (ConvolutionFilter.bias 属性) .....	429
clamp (ConvolutionFilter.clamp 属性) .....	429
clone (ConvolutionFilter.clone 方法) .....	430
color (ConvolutionFilter.color 属性) .....	432
ConvolutionFilter 构造函数 .....	432
divisor (ConvolutionFilter.divisor 属性) .....	434
matrix (ConvolutionFilter.matrix 属性) .....	434
matrixX (ConvolutionFilter.matrixX 属性) .....	435
matrixY (ConvolutionFilter.matrixY 属性) .....	436
preserveAlpha (ConvolutionFilter.preserveAlpha 属性) .....	436
CustomActions .....	437
get (CustomActions.get 方法) .....	438
install (CustomActions.install 方法) .....	439
list (CustomActions.list 方法) .....	440
uninstall (CustomActions.uninstall 方法) .....	441
Date .....	442
Date 构造函数 .....	446
getDate (Date.getDate 方法) .....	447
getDay (Date.getDay 方法) .....	448
getFullYear (Date.getFullYear 方法) .....	448
getHours (Date.getHours 方法) .....	449
getMilliseconds (Date.getMilliseconds 方法) .....	450
getMinutes (Date.getMinutes 方法) .....	450
getMonth (Date.getMonth 方法) .....	451
getSeconds (Date.getSeconds 方法) .....	451
getTime (Date.getTime 方法) .....	452
getTimezoneOffset (Date.getTimezoneOffset 方法) .....	452
getUTCDate (Date.getUTCDate 方法) .....	453
getUTCDay (Date.getUTCDay 方法) .....	453
getUTCFullYear (Date.getUTCFullYear 方法) .....	454



getUTCHours (Date.getUTCHours 方法)	454
getUTCMilliseconds (Date.getUTCMilliseconds 方法)	455
getUTCMinutes (Date.getUTCMinutes 方法)	455
getUTCMonth (Date.getUTCMonth 方法)	456
getUTCSeconds (Date.getUTCSeconds 方法)	456
getUTCYear (Date.getUTCYear 方法)	457
getYear (Date.getYear 方法)	457
setDate (Date.setDate 方法)	458
setFullYear (Date.setFullYear 方法)	458
setHours (Date.setHours 方法)	459
setMilliseconds (Date.setMilliseconds 方法)	460
setMinutes (Date.setMinutes 方法)	460
setMonth (Date.setMonth 方法)	461
setSeconds (Date.setSeconds 方法)	462
setTime (Date.setTime 方法)	462
setUTCDate (Date.setUTCDate 方法)	463
setUTCFullYear (Date.setUTCFullYear 方法)	464
setUTCHours (Date.setUTCHours 方法)	465
setUTCMilliseconds (Date.setUTCMilliseconds 方法)	466
setUTCMinutes (Date.setUTCMinutes 方法)	466
setUTCMonth (Date.setUTCMonth 方法)	467
setUTCSeconds (Date.setUTCSeconds 方法)	468
setYear (Date.setYear 方法)	468
toString (Date.toString 方法)	469
UTC (Date.UTC 方法)	470
valueOf (Date.valueOf 方法)	470
DisplacementMapFilter (flash.filters.DisplacementMapFilter)	471
alpha (DisplacementMapFilter.alpha 属性)	473
clone (DisplacementMapFilter.clone 方法)	475
color (DisplacementMapFilter.color 属性)	478
componentX (DisplacementMapFilter.componentX 属性)	479
componentY (DisplacementMapFilter.componentY 属性)	481
DisplacementMapFilter 构造函数	483
mapBitmap (DisplacementMapFilter.mapBitmap 属性)	485
mapPoint (DisplacementMapFilter.mapPoint 属性)	487
mode (DisplacementMapFilter.mode 属性)	488
scaleX (DisplacementMapFilter.scaleX 属性)	490
scaleY (DisplacementMapFilter.scaleY 属性)	492
DropShadowFilter (flash.filters.DropShadowFilter)	493
alpha (DropShadowFilter.alpha 属性)	496
angle (DropShadowFilter.angle 属性)	497
blurX (DropShadowFilter.blurX 属性)	498
blurY (DropShadowFilter.blurY 属性)	499
clone (DropShadowFilter.clone 方法)	500
color (DropShadowFilter.color 属性)	501

distance (DropShadowFilter.distance 属性)	502
DropShadowFilter 构造函数	503
hideObject (DropShadowFilter.hideObject 属性)	505
inner (DropShadowFilter.inner 属性)	506
knockout (DropShadowFilter.knockout 属性)	507
quality (DropShadowFilter.quality 属性)	508
strength (DropShadowFilter.strength 属性)	509
Error	510
Error 构造函数	511
message (Error.message 属性)	511
name (Error.name 属性)	512
toString (Error.toString 方法)	513
ExternalInterface (flash.external.ExternalInterface)	514
addCallback (ExternalInterface.addCallback 方法)	515
available (ExternalInterface.available 属性)	517
call (ExternalInterface.call 方法)	517
FileReference (flash.net.FileReference)	519
addListener (FileReference.addListener 方法)	523
browse (FileReference.browse 方法)	524
cancel (FileReference.cancel 方法)	526
creationDate (FileReference.creationDate 属性)	526
creator (FileReference.creator 属性)	527
download (FileReference.download 方法)	527
FileReference 构造函数	530
modificationDate (FileReference.modificationDate 属性)	530
name (FileReference.name 属性)	531
onCancel (FileReference.onCancel 事件侦听器)	532
onComplete (FileReference.onComplete 事件侦听器)	533
onHTTPError (FileReference.onHTTPError 事件侦听器)	533
onIOError (FileReference.onIOError 事件侦听器)	535
onOpen (FileReference.onOpen 事件侦听器)	536
onProgress (FileReference.onProgress 事件侦听器)	536
onSecurityError (FileReference.onSecurityError 事件侦听器)	537
onSelect (FileReference.onSelect 事件侦听器)	539
removeListener (FileReference.removeListener 方法)	540
size (FileReference.size 属性)	541
type (FileReference.type 属性)	541
upload (FileReference.upload 方法)	542
FileReferenceList (flash.net.FileReferenceList)	545
addListener (FileReferenceList.addListener 方法)	548
browse (FileReferenceList.browse 方法)	548
fileList (FileReferenceList.fileList 属性)	550
FileReferenceList 构造函数	551
onCancel (FileReferenceList.onCancel 事件侦听器)	552

onSelect (FileReferenceList.onSelect 事件侦听器) .....	552
removeListener (FileReferenceList.removeListener 方法) .....	553
Function .....	554
apply (Function.apply 方法) .....	555
call (Function.call 方法) .....	557
GlowFilter (flash.filters.GlowFilter) .....	558
alpha (GlowFilter.alpha 属性) .....	560
blurX (GlowFilter.blurX 属性) .....	561
blurY (GlowFilter.blurY 属性) .....	562
clone (GlowFilter.clone 方法) .....	563
color (GlowFilter.color 属性) .....	564
GlowFilter 构造函数 .....	565
inner (GlowFilter.inner 属性) .....	567
knockout (GlowFilter.knockout 属性) .....	568
quality (GlowFilter.quality 属性) .....	569
strength (GlowFilter.strength 属性) .....	570
GradientBevelFilter (flash.filters.GradientBevelFilter) .....	571
alphas (GradientBevelFilter.alphas 属性) .....	573
angle (GradientBevelFilter.angle 属性) .....	575
blurX (GradientBevelFilter.blurX 属性) .....	576
blurY (GradientBevelFilter.blurY 属性) .....	577
clone (GradientBevelFilter.clone 方法) .....	578
colors (GradientBevelFilter.colors 属性) .....	579
distance (GradientBevelFilter.distance 属性) .....	580
GradientBevelFilter 构造函数 .....	581
knockout (GradientBevelFilter.knockout 属性) .....	583
quality (GradientBevelFilter.quality 属性) .....	584
ratios (GradientBevelFilter.ratios 属性) .....	585
strength (GradientBevelFilter.strength 属性) .....	587
type (GradientBevelFilter.type 属性) .....	588
GradientGlowFilter (flash.filters.GradientGlowFilter) .....	589
alphas (GradientGlowFilter.alphas 属性) .....	592
angle (GradientGlowFilter.angle 属性) .....	593
blurX (GradientGlowFilter.blurX 属性) .....	594
blurY (GradientGlowFilter.blurY 属性) .....	595
clone (GradientGlowFilter.clone 方法) .....	596
colors (GradientGlowFilter.colors 属性) .....	598
distance (GradientGlowFilter.distance 属性) .....	599
GradientGlowFilter 构造函数 .....	600
knockout (GradientGlowFilter.knockout 属性) .....	602
quality (GradientGlowFilter.quality 属性) .....	603
ratios (GradientGlowFilter.ratios 属性) .....	604
strength (GradientGlowFilter.strength 属性) .....	606
type (GradientGlowFilter.type 属性) .....	607
IME (System.IME) .....	608

addListener (IME.addListener 方法)	612
ALPHANUMERIC_FULL (IME.ALPHANUMERIC_FULL 属性)	613
ALPHANUMERIC_HALF (IME.ALPHANUMERIC_HALF 属性)	613
CHINESE (IME.CHINESE 属性)	614
doConversion (IME.doConversion 方法)	614
getConversionMode (IME.getConversionMode 方法)	615
getEnabled (IME.getEnabled 方法)	616
JAPANESE_HIRAGANA	
(IME.JAPANESE_HIRAGANA 属性)	616
JAPANESE_KATAKANA_FULL	
(IME.JAPANESE_KATAKANA_FULL 属性)	617
JAPANESE_KATAKANA_HALF	
(IME.JAPANESE_KATAKANA_HALF 属性)	618
KOREAN (IME.KOREAN 属性)	618
onIMEComposition (IME.onIMEComposition 事件侦听器)	619
removeListener (IME.removeListener 方法)	620
setCompositionString (IME.setCompositionString 方法)	621
setConversionMode (IME.setConversionMode 方法)	622
setEnabled (IME.setEnabled 方法)	623
UNKNOWN (IME.UNKNOWN 属性)	624
Key	624
addListener (Key.addListener 方法)	626
BACKSPACE (Key.BACKSPACE 属性)	628
CAPSLOCK (Key.CAPSLOCK 属性)	628
CONTROL (Key.CONTROL 属性)	629
DELETEKEY (Key.DELETEKEY 属性)	629
DOWN (Key.DOWN 属性)	630
END (Key.END 属性)	631
ENTER (Key.ENTER 属性)	631
ESCAPE (Key.ESCAPE 属性)	632
getAscii (Key.getAscii 方法)	633
getCode (Key.getCode 方法)	634
HOME (Key.HOME 属性)	635
INSERT (Key.INSERT 属性)	636
isAccessible (Key.isAccessible 方法)	636
isDown (Key.isDown 方法)	637
isToggled (Key.isToggled 方法)	637
LEFT (Key.LEFT 属性)	639
_listeners (Key._listeners 属性)	640
onKeyDown (Key.onKeyDown 事件侦听器)	640
onKeyUp (Key.onKeyUp 事件侦听器)	641
PGDN (Key.PGDN 属性)	641
PGUP (Key.PGUP 属性)	642
removeListener (Key.removeListener 方法)	642
RIGHT (Key.RIGHT 属性)	643

SHIFT (Key.SHIFT 属性) .....	644
SPACE (Key.SPACE 属性) .....	644
TAB (Key.TAB 属性) .....	645
UP (Key.UP 属性) .....	646
LoadVars .....	647
addRequestHeader (LoadVars.addRequestHeader 方法) .....	649
contentType (LoadVars.contentType 属性) .....	650
decode (LoadVars.decode 方法) .....	650
getBytesLoaded (LoadVars.getBytesLoaded 方法) .....	651
getBytesTotal (LoadVars.getBytesTotal 方法) .....	652
load (LoadVars.load 方法) .....	653
loaded (LoadVars.loaded 属性) .....	655
LoadVars 构造函数 .....	656
onData (LoadVars.onData 处理函数) .....	656
onHTTPStatus (LoadVars.onHTTPStatus 处理函数) .....	657
onLoad (LoadVars.onLoad 处理函数) .....	659
send (LoadVars.send 方法) .....	660
sendAndLoad (LoadVars.sendAndLoad 方法) .....	662
toString (LoadVars.toString 方法) .....	664
LocalConnection .....	664
allowDomain (LocalConnection.allowDomain 处理函数) .....	666
allowInsecureDomain (LocalConnection.allowInsecureDomain 处理函数) .....	669
close (LocalConnection.close 方法) .....	671
connect (LocalConnection.connect 方法) .....	672
domain (LocalConnection.domain 方法) .....	675
LocalConnection 构造函数 .....	677
onStatus (LocalConnection.onStatus 处理函数) .....	678
send (LocalConnection.send 方法) .....	679
Locale (mx.lang.Locale) .....	681
addDelayedInstance (Locale.addDelayedInstance 方法) .....	683
addXMLPath (Locale.addXMLPath 方法) .....	684
autoReplace (Locale.autoReplace 属性) .....	684
checkXMLStatus (Locale.checkXMLStatus 方法) .....	685
getDefaultLang (Locale.getDefaultLang 方法) .....	686
initialize (Locale.initialize 方法) .....	687
languageCodeArray (Locale.languageCodeArray 属性) .....	687
loadLanguageXML (Locale.loadLanguageXML 方法) .....	688
loadString (Locale.loadString 方法) .....	689
loadStringEx (Locale.loadStringEx 方法) .....	690
setDefaultLang (Locale.setDefaultLang 方法) .....	691
setLoadCallback (Locale.setLoadCallback 方法) .....	692
setString (Locale.setString 方法) .....	692
stringIDArray (Locale.stringIDArray 属性) .....	693

Math .....	694
abs (Math.abs 方法) .....	696
acos (Math.acos 方法) .....	697
asin (Math.asin 方法) .....	697
atan (Math.atan 方法) .....	698
atan2 (Math.atan2 方法) .....	699
ceil (Math.ceil 方法) .....	699
cos (Math.cos 方法) .....	700
E (Math.E 属性) .....	701
exp (Math.exp 方法) .....	701
floor (Math.floor 方法) .....	702
LN10 (Math.LN10 属性) .....	702
LN2 (Math.LN2 属性) .....	703
log (Math.log 方法) .....	703
LOG10E (Math.LOG10E 属性) .....	703
LOG2E (Math.LOG2E 属性) .....	704
max (Math.max 方法) .....	704
min (Math.min 方法) .....	705
PI (Math.PI 属性) .....	705
pow (Math.pow 方法) .....	706
random (Math.random 方法) .....	707
round (Math.round 方法) .....	707
sin (Math.sin 方法) .....	708
sqrt (Math.sqrt 方法) .....	709
SQRT1_2 (Math.SQRT1_2 属性) .....	710
SQRT2 (Math.SQRT2 属性) .....	710
tan (Math.tan 方法) .....	710
Matrix (flash.geom.Matrix) .....	711
a (Matrix.a 属性) .....	715
b (Matrix.b 属性) .....	716
c (Matrix.c 属性) .....	716
clone (Matrix.clone 方法) .....	717
concat (Matrix.concat 方法) .....	717
createBox (Matrix.createBox 方法) .....	719
createGradientBox (Matrix.createGradientBox 方法) .....	720
d (Matrix.d 属性) .....	721
deltaTransformPoint (Matrix.deltaTransformPoint 方法) .....	721
identity (Matrix.identity 方法) .....	723
invert (Matrix.invert 方法) .....	724
Matrix 构造函数 .....	725
rotate (Matrix.rotate 方法) .....	726
scale (Matrix.scale 方法) .....	728
toString (Matrix.toString 方法) .....	729
transformPoint (Matrix.transformPoint 方法) .....	730
translate (Matrix.translate 方法) .....	731

tx (Matrix.tx 属性)	731
ty (Matrix.ty 属性)	732
Microphone	732
activityLevel (Microphone.activityLevel 属性)	734
gain (Microphone.gain 属性)	735
get (Microphone.get 方法)	736
index (Microphone.index property)	738
muted (Microphone.muted 属性)	739
name (Microphone.name 属性)	739
names (Microphone.names 属性)	740
onActivity (Microphone.onActivity 处理函数)	741
onStatus (Microphone.onStatus 处理函数)	742
rate (Microphone.rate 属性)	743
setGain (Microphone.setGain 方法)	744
setRate (Microphone.setRate 方法)	745
setSilenceLevel (Microphone.setSilenceLevel 方法)	746
setUseEchoSuppression	
(Microphone.setUseEchoSuppression 方法)	748
silenceLevel (Microphone.silenceLevel 属性)	749
silenceTimeOut (Microphone.silenceTimeOut 属性)	750
useEchoSuppression	
(Microphone.useEchoSuppression 属性)	752
Mouse	753
addListener (Mouse.addListener 方法)	754
hide (Mouse.hide 方法)	756
onMouseDown (Mouse.onMouseDown 事件侦听器)	757
onMouseMove (Mouse.onMouseMove 事件侦听器)	758
onMouseUp (Mouse.onMouseUp 事件侦听器)	759
onMouseWheel (Mouse.onMouseWheel 事件侦听器)	760
removeListener (Mouse.removeListener 方法)	762
show (Mouse.show 方法)	763
MovieClip	764
_alpha (MovieClip._alpha 属性)	772
attachAudio (MovieClip.attachAudio 方法)	773
attachBitmap (MovieClip.attachBitmap 方法)	774
attachMovie (MovieClip.attachMovie 方法)	775
beginBitmapFill (MovieClip.beginBitmapFill 方法)	776
beginFill (MovieClip.beginFill 方法)	778
beginGradientFill (MovieClip.beginGradientFill 方法)	779
blendMode (MovieClip.blendMode 属性)	785
cacheAsBitmap (MovieClip.cacheAsBitmap 属性)	790
clear (MovieClip.clear 方法)	791
createEmptyMovieClip	
(MovieClip.createEmptyMovieClip 方法)	792
createTextField (MovieClip.createTextField 方法)	793

_currentframe (MovieClip._currentframe 属性)	795
curveTo (MovieClip.curveTo 方法)	796
_droptarget (MovieClip._droptarget 属性)	798
duplicateMovieClip (MovieClip.duplicateMovieClip 方法)	799
enabled (MovieClip.enabled 属性)	801
endFill (MovieClip.endFill 方法)	801
filters (MovieClip.filters 属性)	802
focusEnabled (MovieClip.focusEnabled 属性)	803
_focusrect (MovieClip._focusrect 属性)	804
_framesloaded (MovieClip._framesloaded 属性)	805
getBounds (MovieClip.getBounds 方法)	806
getBytesLoaded (MovieClip.getBytesLoaded 方法)	807
getBytesTotal (MovieClip.getBytesTotal 方法)	808
getDepth (MovieClip.getDepth 方法)	809
getInstanceAtDepth (MovieClip.getInstanceAtDepth 方法)	810
getNextHighestDepth (MovieClip.getNextHighestDepth 方法)	811
getRect (MovieClip.getRect 方法)	812
getSWFVersion (MovieClip.getSWFVersion 方法)	813
getTextSnapshot (MovieClip.getTextSnapshot 方法)	814
getURL (MovieClip.getURL 方法)	815
globalToLocal (MovieClip.globalToLocal 方法)	817
gotoAndPlay (MovieClip.gotoAndPlay 方法)	819
gotoAndStop (MovieClip.gotoAndStop 方法)	820
_height (MovieClip._height 属性)	821
_highquality (MovieClip._highquality 属性)	821
hitArea (MovieClip.hitArea 属性)	822
hitTest (MovieClip.hitTest 方法)	823
lineGradientStyle (MovieClip.lineGradientStyle 方法)	824
lineStyle (MovieClip.lineStyle 方法)	828
lineTo (MovieClip.lineTo 方法)	830
loadMovie (MovieClip.loadMovie 方法)	831
loadVariables (MovieClip.loadVariables 方法)	834
localToGlobal (MovieClip.localToGlobal 方法)	835
_lockroot (MovieClip._lockroot 属性)	837
menu (MovieClip.menu 属性)	840
moveTo (MovieClip.moveTo 方法)	841
_name (MovieClip._name 属性)	841
nextFrame (MovieClip.nextFrame 方法)	842
onData (MovieClip.onData 处理函数)	843
onDragOut (MovieClip.onDragOut 处理函数)	844
onDragOver (MovieClip.onDragOver 处理函数)	844
onEnterFrame (MovieClip.onEnterFrame 处理函数)	845
onKeyDown (MovieClip.onKeyDown 处理函数)	845
onKeyUp (MovieClip.onKeyUp 处理函数)	846
onKillFocus (MovieClip.onKillFocus 处理函数)	847
onLoad (MovieClip.onLoad 处理函数)	848



onMouseDown (MovieClip.onMouseDown 处理函数) .....	849
onMouseMove (MovieClip.onMouseMove 处理函数) .....	849
onMouseUp (MovieClip.onMouseUp 处理函数) .....	849
onPress (MovieClip.onPress 处理函数) .....	850
onRelease (MovieClip.onRelease 处理函数) .....	850
onReleaseOutside (MovieClip.onReleaseOutside 处理函数) .....	851
onRollOut (MovieClip.onRollOut 处理函数) .....	851
onRollOver (MovieClip.onRollOver 处理函数) .....	852
onSetFocus (MovieClip.onSetFocus 处理函数) .....	852
onUnload (MovieClip.onUnload 处理函数) .....	853
opaqueBackground (MovieClip.opaqueBackground 属性) .....	853
_parent (MovieClip._parent 属性) .....	854
play (MovieClip.play 方法) .....	855
prevFrame (MovieClip.prevFrame 方法) .....	855
_quality (MovieClip._quality 属性) .....	856
removeMovieClip (MovieClip.removeMovieClip 方法) .....	858
_rotation (MovieClip._rotation 属性) .....	859
scale9Grid (MovieClip.scale9Grid 属性) .....	860
scrollRect (MovieClip.scrollRect 属性) .....	863
setMask (MovieClip.setMask 方法) .....	864
_soundbuftime (MovieClip._soundbuftime 属性) .....	865
startDrag (MovieClip.startDrag 方法) .....	865
stop (MovieClip.stop 方法) .....	866
stopDrag (MovieClip.stopDrag 方法) .....	867
swapDepths (MovieClip.swapDepths 方法) .....	868
tabChildren (MovieClip.tabChildren 属性) .....	869
tabEnabled (MovieClip.tabEnabled 属性) .....	870
tabIndex (MovieClip.tabIndex 属性) .....	870
_target (MovieClip._target 属性) .....	871
_totalframes (MovieClip._totalframes 属性) .....	872
trackAsMenu (MovieClip.trackAsMenu 属性) .....	872
transform (MovieClip.transform 属性) .....	873
unloadMovie (MovieClip.unloadMovie 方法) .....	875
_url (MovieClip._url 属性) .....	876
useHandCursor (MovieClip.useHandCursor 属性) .....	877
_visible (MovieClip._visible 属性) .....	877
_width (MovieClip._width 属性) .....	878
_x (MovieClip._x 属性) .....	879
_xmouse (MovieClip._xmouse 属性) .....	880
_xscale (MovieClip._xscale 属性) .....	881
_y (MovieClip._y 属性) .....	882
_ymouse (MovieClip._ymouse 属性) .....	883
_yscale (MovieClip._yscale 属性) .....	884
MovieClipLoader .....	885
addListener (MovieClipLoader.addListener 方法) .....	887

getProgress (MovieClipLoader.getProgress 方法) .....	888
loadClip (MovieClipLoader.loadClip 方法) .....	889
MovieClipLoader 构造函数 .....	891
onLoadComplete (MovieClipLoader.onLoadComplete 事件侦听器) .....	892
onLoadError (MovieClipLoader.onLoadError 事件侦听器) .....	894
onLoadInit (MovieClipLoader.onLoadInit 事件侦听器) .....	895
onLoadProgress (MovieClipLoader.onLoadProgress 事件侦听器) .....	897
onLoadStart (MovieClipLoader.onLoadStart 事件侦听器) .....	898
removeListener (MovieClipLoader.removeListener 方法) .....	899
unloadClip (MovieClipLoader.unloadClip 方法) .....	900
NetConnection .....	901
connect (NetConnection.connect 方法) .....	902
NetConnection 构造函数 .....	903
NetStream .....	904
bufferLength (NetStream.bufferLength 属性) .....	906
bufferTime (NetStream.bufferTime 属性) .....	907
bytesLoaded (NetStream.bytesLoaded 属性) .....	908
bytesTotal (NetStream.bytesTotal 属性) .....	909
close (NetStream.close 方法) .....	910
currentFps (NetStream.currentFps 属性) .....	911
NetStream 构造函数 .....	912
onMetaData (NetStream.onMetaData 处理函数) .....	912
onStatus (NetStream.onStatus 处理函数) .....	913
pause (NetStream.pause 方法) .....	915
play (NetStream.play 方法) .....	916
seek (NetStream.seek 方法) .....	917
setBufferTime (NetStream.setBufferTime 方法) .....	918
time (NetStream.time 属性) .....	919
Number .....	920
MAX_VALUE (Number.MAX_VALUE 属性) .....	921
MIN_VALUE (Number.MIN_VALUE 属性) .....	922
NaN (Number.NaN 属性) .....	922
NEGATIVE_INFINITY (Number.NEGATIVE_INFINITY 属性) .....	922
Number 构造函数 .....	923
POSITIVE_INFINITY (Number.POSITIVE_INFINITY 属性) .....	923
toString (Number.toString 方法) .....	924
valueOf (Number.valueOf 方法) .....	924
Object .....	925
addProperty (Object.addProperty 方法) .....	926
constructor (Object.constructor 属性) .....	929
hasOwnProperty (Object.hasOwnProperty 方法) .....	930
isPrototypeOf (Object.isPrototypeOf 方法) .....	930
isPrototypeOf (Object.isPrototypeOf 方法) .....	931

Object 构造函数	931
__proto__ (Object.__proto__ 属性)	932
prototype (Object.prototype 属性)	933
registerClass (Object.registerClass 方法)	934
__resolve (Object.__resolve 属性)	935
toString (Object.toString 方法)	938
unwatch (Object.unwatch 方法)	939
valueOf (Object.valueOf 方法)	940
watch (Object.watch 方法)	941
Point (flash.geom.Point)	943
add (Point.add 方法)	945
clone (Point.clone 方法)	945
distance (Point.distance 方法)	946
equals (Point.equals 方法)	946
interpolate (Point.interpolate 方法)	947
length (Point.length 属性)	948
normalize (Point.normalize 方法)	948
offset (Point.offset 方法)	949
Point 构造函数	949
polar (Point.polar 方法)	950
subtract (Point.subtract 方法)	951
toString (Point.toString 方法)	951
x (Point.x 属性)	952
y (Point.y 属性)	952
PrintJob	953
addPage (PrintJob.addPage 方法)	954
orientation (PrintJob.orientation 属性)	958
pageHeight (PrintJob.pageHeight 属性)	958
pageWidth (PrintJob.pageWidth 属性)	958
paperHeight (PrintJob.paperHeight 属性)	958
paperWidth (PrintJob.paperWidth 属性)	958
PrintJob 构造函数	959
send (PrintJob.send 方法)	960
start (PrintJob.start 方法)	961
Rectangle (flash.geom.Rectangle)	963
bottom (Rectangle.bottom 属性)	965
bottomRight (Rectangle.bottomRight 属性)	966
clone (Rectangle.clone 方法)	967
contains (Rectangle.contains 方法)	969
containsPoint (Rectangle.containsPoint 方法)	970
containsRectangle (Rectangle.containsRectangle 方法)	971
equals (Rectangle.equals 方法)	972
height (Rectangle.height 属性)	973
inflate (Rectangle.inflate 方法)	974
inflatePoint (Rectangle.inflatePoint 方法)	975

intersection (Rectangle.intersection 方法)	976
intersects (Rectangle.intersects 方法)	977
isEmpty (Rectangle.isEmpty 方法)	978
left (Rectangle.left 属性)	979
offset (Rectangle.offset 方法)	980
offsetPoint (Rectangle.offsetPoint 方法)	980
Rectangle 构造函数	981
right (Rectangle.right 属性)	982
setEmpty (Rectangle.setEmpty 方法)	983
size (Rectangle.size 属性)	983
top (Rectangle.top 属性)	984
topLeft (Rectangle.topLeft 属性)	985
toString (Rectangle.toString 方法)	986
union (Rectangle.union 方法)	986
width (Rectangle.width 属性)	987
x (Rectangle.x 属性)	988
y (Rectangle.y 属性)	989
security (System.security)	990
allowDomain (security.allowDomain 方法)	991
allowInsecureDomain (security.allowInsecureDomain 方法)	996
loadPolicyFile (security.loadPolicyFile 方法)	999
sandboxType (security.sandboxType 属性)	1001
Selection	1002
addListener (Selection.addListener 方法)	1004
getBeginIndex (Selection.getBeginIndex 方法)	1005
getCaretIndex (Selection.getCaretIndex 方法)	1006
getEndIndex (Selection.getEndIndex 方法)	1007
getFocus (Selection.getFocus 方法)	1008
onSetFocus (Selection.onSetFocus 事件侦听器)	1009
removeListener (Selection.removeListener 方法)	1010
setFocus (Selection.setFocus 方法)	1011
setSelection (Selection.setSelection 方法)	1013
SharedObject	1014
clear (SharedObject.clear 方法)	1016
data (SharedObject.data 属性)	1017
flush (SharedObject.flush 方法)	1019
getLocal (SharedObject.getLocal 方法)	1020
getSize (SharedObject.getSize 方法)	1024
onStatus (SharedObject.onStatus 处理函数)	1025
Sound	1027
attachSound (Sound.attachSound 方法)	1029
duration (Sound.duration 属性)	1029
getBytesLoaded (Sound.getBytesLoaded 方法)	1031
getBytesTotal (Sound.getBytesTotal 方法)	1033
getPan (Sound.getPan 方法)	1033

getTransform (Sound.getTransform 方法)	1035
getVolume (Sound.getVolume 方法)	1037
id3 (Sound.id3 属性)	1038
loadSound (Sound.loadSound 方法)	1041
onID3 (Sound.onID3 处理函数)	1042
onLoad (Sound.onLoad 处理函数)	1043
onSoundComplete (Sound.onSoundComplete 处理函数)	1044
position (Sound.position 属性)	1045
setPan (Sound.setPan 方法)	1045
setTransform (Sound.setTransform 方法)	1046
setVolume (Sound.setVolume 方法)	1048
Sound 构造函数	1048
start (Sound.start 方法)	1049
stop (Sound.stop 方法)	1050
Stage	1051
addListener (Stage.addListener 方法)	1052
align (Stage.align 属性)	1053
height (Stage.height 属性)	1054
onResize (Stage.onResize 事件侦听器)	1054
removeListener (Stage.removeListener 方法)	1055
scaleMode (Stage.scaleMode 属性)	1056
showMenu (Stage.showMenu 属性)	1057
width (Stage.width 属性)	1058
String	1059
charAt (String.charAt 方法)	1061
charCodeAt (String.charCodeAt 方法)	1062
concat (String.concat 方法)	1062
fromCharCode (String.fromCharCode 方法)	1063
indexOf (String.indexOf 方法)	1063
lastIndexOf (String.lastIndexOf 方法)	1064
length (String.length 属性)	1065
slice (String.slice 方法)	1066
split (String.split 方法)	1068
String 构造函数	1069
substr (String.substr 方法)	1069
substring (String.substring 方法)	1070
toLowerCase (String.toLowerCase 方法)	1071
toString (String.toString 方法)	1072
toUpperCase (String.toUpperCase 方法)	1073
valueOf (String.valueOf 方法)	1073
StyleSheet (TextField.StyleSheet)	1074
clear (StyleSheet.clear 方法)	1077
getStyle (StyleSheet.getStyle 方法)	1078
getStyleNames (StyleSheet.getStyleNames 方法)	1080
load (StyleSheet.load 方法)	1081

onLoad (StyleSheet.onLoad 处理函数)	1082
parseCSS (StyleSheet.parseCSS 方法)	1083
setStyle (StyleSheet.setStyle 方法)	1084
StyleSheet 构造函数	1085
transform (StyleSheet.transform 方法)	1086
System	1086
exactSettings (System.exactSettings 属性)	1088
onStatus (System.onStatus 处理函数)	1089
setClipboard (System.setClipboard 方法)	1090
showSettings (System.showSettings 方法)	1091
useCodepage (System.useCodepage 属性)	1092
TextField	1093
addListener (TextField.addListener 方法)	1098
_alpha (TextField._alpha 属性)	1099
antiAliasType (TextField.antiAliasType 属性)	1100
autoSize (TextField.autoSize 属性)	1101
background (TextField.background 属性)	1103
backgroundColor (TextField.backgroundColor 属性)	1103
border (TextField.border 属性)	1104
borderColor (TextField.borderColor 属性)	1104
bottomScroll (TextField.bottomScroll 属性)	1105
condenseWhite (TextField.condenseWhite 属性)	1106
embedFonts (TextField.embedFonts 属性)	1107
filters (TextField.filters 属性)	1108
getDepth (TextField.getDepth 方法)	1109
getFontList (TextField.getFontList 方法)	1110
getNewTextFormat (TextField.getNewTextFormat 方法)	1110
getTextFormat (TextField.getTextFormat 方法)	1111
gridFitType (TextField.gridFitType 属性)	1112
_height (TextField._height 属性)	1114
_highquality (TextField._highquality 属性)	1114
hscroll (TextField.hscroll 属性)	1114
html (TextField.html 属性)	1115
htmlText (TextField.htmlText 属性)	1116
length (TextField.length 属性)	1117
maxChars (TextField.maxChars 属性)	1117
maxhscroll (TextField.maxhscroll 属性)	1118
maxscroll (TextField.maxscroll 属性)	1118
menu (TextField.menu 属性)	1119
mouseWheelEnabled (TextField.mouseWheelEnabled 属性)	1120
multiline (TextField.multiline 属性)	1121
_name (TextField._name 属性)	1121
onChanged (TextField.onChanged 处理函数)	1122
onKillFocus (TextField.onKillFocus 处理函数)	1123
onScroller (TextField.onScroller 处理函数)	1124

onSetFocus (TextField.onSetFocus 处理函数)	1125
_parent (TextField._parent 属性)	1126
password (TextField.password 属性)	1127
_quality (TextField._quality 属性)	1128
removeListener (TextField.removeListener 方法)	1128
removeTextField (TextField.removeTextField 方法)	1129
replaceSel (TextField.replaceSel 方法)	1130
replaceText (TextField.replaceText 方法)	1131
restrict (TextField.restrict 属性)	1132
_rotation (TextField._rotation 属性)	1133
scroll (TextField.scroll 属性)	1134
selectable (TextField.selectable 属性)	1135
setNewTextFormat (TextField.setNewTextFormat 方法)	1136
setTextFormat (TextField.setTextFormat 方法)	1137
sharpness (TextField.sharpness 属性)	1139
_soundbuftime (TextField._soundbuftime 属性)	1140
styleSheet (TextField.styleSheet 属性)	1141
tabEnabled (TextField.tabEnabled 属性)	1143
tabIndex (TextField.tabIndex 属性)	1144
_target (TextField._target 属性)	1145
text (TextField.text 属性)	1145
textColor (TextField.textColor 属性)	1146
textHeight (TextField.textHeight 属性)	1147
textWidth (TextField.textWidth 属性)	1147
thickness (TextField.thickness 属性)	1148
type (TextField.type 属性)	1149
_url (TextField._url 属性)	1150
variable (TextField.variable 属性)	1150
_visible (TextField._visible 属性)	1151
_width (TextField._width 属性)	1151
wordWrap (TextField.wordWrap 属性)	1152
_x (TextField._x 属性)	1153
_xmouse (TextField._xmouse 属性)	1154
_xscale (TextField._xscale 属性)	1155
_y (TextField._y 属性)	1156
_ymouse (TextField._ymouse 属性)	1156
_yscale (TextField._yscale 属性)	1157
TextFormat	1157
align (TextFormat.align 属性)	1160
blockIndent (TextFormat.blockIndent 属性)	1161
bold (TextFormat.bold 属性)	1161
bullet (TextFormat.bullet 属性)	1162
color (TextFormat.color 属性)	1162
font (TextFormat.font 属性)	1163
getTextExtent (TextFormat.getTextExtent 方法)	1163

indent (TextFormat.indent 属性)	1166
italic (TextFormat.italic 属性)	1166
kerning (TextFormat.kerning 属性)	1167
leading (TextFormat.leading 属性)	1168
leftMargin (TextFormat.leftMargin 属性)	1168
letterSpacing (TextFormat.letterSpacing 属性)	1169
rightMargin (TextFormat.rightMargin 属性)	1170
size (TextFormat.size 属性)	1170
tabStops (TextFormat.tabStops 属性)	1171
target (TextFormat.target 属性)	1171
TextFormat 构造函数	1172
underline (TextFormat.underline 属性)	1174
url (TextFormat.url 属性)	1174
TextRenderer (flash.text.TextRenderer)	1175
maxLevel (TextRenderer.maxLevel 属性)	1176
setAdvancedAntialiasingTable (TextRenderer.setAdvancedAntialiasingTable 方法)	1177
TextSnapshot	1179
findText (TextSnapshot.findText 方法)	1181
getCount (TextSnapshot.getCount 方法)	1182
getSelected (TextSnapshot.getSelected 方法)	1182
getSelectedText (TextSnapshot.getSelectedText 方法)	1183
getText (TextSnapshot.getText 方法)	1184
getTextRunInfo (TextSnapshot.getTextRunInfo 方法)	1186
hitTestTextNearPos (TextSnapshot.hitTestTextNearPos 方法)	1188
setSelectColor (TextSnapshot.setSelectColor 方法)	1189
setSelected (TextSnapshot.setSelected 方法)	1190
Transform (flash.geom.Transform)	1191
colorTransform (Transform.colorTransform 属性)	1193
concatenatedColorTransform (Transform.concatenatedColorTransform 属性)	1194
concatenatedMatrix (Transform.concatenatedMatrix 属性)	1195
matrix (Transform.matrix 属性)	1196
pixelBounds (Transform.pixelBounds 属性)	1197
Transform 构造函数	1198
Video	1199
_alpha (Video._alpha 属性)	1202
attachVideo (Video.attachVideo 方法)	1202
clear (Video.clear 方法)	1203
deblocking (Video.deblocking 属性)	1204
_height (Video._height 属性)	1205
height (Video.height 属性)	1205
_name (Video._name 属性)	1206
_parent (Video._parent 属性)	1206
_rotation (Video._rotation 属性)	1206



smoothing (Video.smoothing 属性)	1207
_visible (Video._visible 属性)	1207
_width (Video._width 属性)	1208
width (Video.width 属性)	1208
_x (Video._x 属性)	1208
_xmouse (Video._xmouse 属性)	1209
_xscale (Video._xscale 属性)	1209
_y (Video._y 属性)	1209
_ymouse (Video._ymouse 属性)	1210
_yscale (Video._yscale 属性)	1210
XML	1210
addRequestHeader (XML.addRequestHeader 方法)	1214
contentType (XML.contentType 属性)	1215
createElement (XML.createElement 方法)	1215
createTextNode (XML.createTextNode 方法)	1216
docTypeDecl (XML.docTypeDecl 属性)	1218
getBytesLoaded (XML.getBytesLoaded 方法)	1218
getBytesTotal (XML.getBytesTotal 方法)	1219
idMap (XML.idMap 属性)	1220
ignoreWhite (XML.ignoreWhite 属性)	1222
load (XML.load 方法)	1224
loaded (XML.loaded 属性)	1225
onData (XML.onData 处理函数)	1226
onHTTPStatus (XML.onHTTPStatus 处理函数)	1227
onLoad (XML.onLoad 处理函数)	1229
parseXML (XML.parseXML 方法)	1230
send (XML.send 方法)	1230
sendAndLoad (XML.sendAndLoad 方法)	1232
status (XML.status 属性)	1234
XML 构造函数	1235
xmlDecl (XML.xmlDecl 属性)	1236
XMLNode	1237
appendChild (XMLNode.appendChild 方法)	1240
attributes (XMLNode.attributes 属性)	1241
childNodes (XMLNode.childNodes 属性)	1241
cloneNode (XMLNode.cloneNode 方法)	1243
firstChild (XMLNode.firstChild 属性)	1244
getNamespaceForPrefix	
(XMLNode.getNamespaceForPrefix 方法)	1246
getPrefixForNamespace	
(XMLNode.getPrefixForNamespace 方法)	1247
hasChildNodes (XMLNode.hasChildNodes 方法)	1248
insertBefore (XMLNode.insertBefore 方法)	1249
lastChild (XMLNode.lastChild 属性)	1249
localName (XMLNode.localName 属性)	1251

namespaceURI (XMLNode.namespaceURI 属性) .....	1252
nextSibling (XMLNode.nextSibling 属性) .....	1254
nodeName (XMLNode.nodeName 属性) .....	1254
nodeType (XMLNode.nodeType 属性) .....	1256
nodeValue (XMLNode.nodeValue 属性) .....	1257
parentNode (XMLNode.parentNode 属性) .....	1258
prefix (XMLNode.prefix 属性) .....	1259
previousSibling (XMLNode.previousSibling 属性) .....	1260
removeNode (XMLNode.removeNode 方法) .....	1261
toString (XMLNode.toString 方法) .....	1262
XMLNode 构造函数 .....	1263
XMLSocket .....	1264
close (XMLSocket.close 方法) .....	1266
connect (XMLSocket.connect 方法) .....	1267
onClose (XMLSocket.onClose 处理函数) .....	1269
onConnect (XMLSocket.onConnect 处理函数) .....	1269
onData (XMLSocket.onData 处理函数) .....	1270
onXML (XMLSocket.onXML 处理函数) .....	1271
send (XMLSocket.send 方法) .....	1272
XMLSocket 构造函数 .....	1272
XMLUI .....	1273
accept (XMLUI.accept 方法) .....	1274
cancel (XMLUI.cancel 方法) .....	1274
get (XMLUI.get 方法) .....	1274
set (XMLUI.set 方法) .....	1275
<b>第 3 章: ActionScript 不推荐使用的内容 .....</b>	<b>1277</b>
不推荐使用的类摘要 .....	1277
不推荐使用的函数摘要 .....	1277
不推荐使用的属性摘要 .....	1278
不推荐使用的运算符摘要 .....	1279
<b>索引 .....</b>	<b>1281</b>

# ActionScript 语言元素

本节提供关于下列各项的语法、用法信息和代码示例：全局函数和属性（即不属于 **ActionScript** 类的那些元素）；编译器指令；以及在 **ActionScript** 中使用的和在 **ECMAScript** (ECMA-262) 第 4 版草案语言规范中定义的常数、运算符、语句和关键字。

## 编译器指令

本节包含的指令需要包括在 **ActionScript** 文件中以指示编译器预处理某些指令。不要将分号 (;) 放在包含指令的行的末尾。

### 编译器指令摘要

指令	说明
<code>#endinitclip</code>	指示初始化动作块的结尾。
<code>#include</code>	包括指定文件的内容，就像该文件中的命令是调用脚本的一部分一样。
<code>#initclip</code>	指示初始化动作块的开始。

### `#endinitclip` 指令

`#endinitclip`

指示初始化动作块的结尾。

不要将分号 (;) 放在包含 `#endinitclip` 指令的行的末尾。

可用性：Flash Player 6.0；ActionScript 1.0

#### 示例

```
#initclip
...initialization actions go here...
#endinitclip
```

## #include 指令

`#include "[path]filename.as":String`

包括指定文件的内容, 就像该文件中的命令是调用脚本的一部分一样。#include 指令在编译时调用。因此, 如果对外部文件进行了任何更改, 则必须保存该文件, 并重新编译任何使用它的 FLA 文件。

如果对包含 #include 语句的脚本使用“语法检查”按钮, 则也会检查所包括文件的语法。

您可以在 FLA 文件和外部脚本文件中使用 #include, 但不能在 ActionScript 2.0 类文件中使用。

您可以对要包括的文件不指定路径、指定相对路径或指定绝对路径。如果您不指定路径, 则 AS 文件必须位于以下位置之一:

- 与 FLA 文件位于同一个目录。与包含 #include 语句的脚本位于同一个目录。
- 全局 Include 目录, 该目录为以下目录之一:
  - Windows 2000 或 Windows XP: C:\Documents and Settings\用户\Local Settings\Application Data\Macromedia\Flash 8\语言\Configuration\Include
  - Macintosh OS X: Hard Drive/Users/Library/Application Support/Macromedia/Flash 8/语言/Configuration/Include
- Flash 8 程序\语言\First Run\Include 目录: 如果您在此目录中保存一个文件, 则在下次启动 Flash 时, 会将此文件复制到全局 Include 目录中。

若要为 AS 文件指定相对路径, 请使用单个点 (.) 来指示当前目录, 使用两个点 (..) 来指示父级目录, 并使用正斜杠 (/) 来指示子目录。请参见以下示例部分。

若要为 AS 文件指定绝对路径, 请使用您的平台 (Macintosh 或 Windows) 所支持的格式。请参见以下示例部分。(不建议使用此方法, 因为此方法要求任一编译此脚本的计算机上均具备相同的目录结构。)

如果您将文件放在 First Run/Include 目录或全局 Include 目录中, 请备份这些文件。如果您需要卸载或重新安装 Flash, 可能会删除和覆盖这些目录。

不要将分号 (;) 放在包含 #include 指令的行的末尾。

可用性: Flash Player 4.0 ; ActionScript 1.0

### 参数

[path]filename.as:String - filename.as 要添加到“动作”面板或当前脚本中的脚本的文件名和可选路径; .as 是推荐使用的文件扩展名。

## 示例

下列示例说明指定要包含在脚本中的文件的路径的各种方法：

```
// Note that #include statements do not end with a semicolon (;)
// AS file is in same directory as FLA file or script
// or is in the global Include directory or the First Run/Include directory
#include "init_script.as"

// AS file is in a subdirectory of one of the above directories
// The subdirectory is named "FLA_includes"
#include "FLA_includes/init_script.as"
// AS file is in a subdirectory of the script file directory
// The subdirectory is named "SCRIPT_includes"
#include "SCRIPT_includes/init_script.as"
// AS file is in a directory at the same level as one of the above
// directories
// AS file is in a directory at the same level as the directory
// that contains the script file
// The directory is named "ALL_includes"
#include "../ALL_includes/init_script.as"

// AS file is specified by an absolute path in Windows
// Note use of forward slashes, not backslashes
#include "C:/Flash_scripts/init_script.as"

// AS file is specified by an absolute path on Macintosh
#include "Mac HD:Flash_scripts:init_script.as"
```

## #initclip 指令

`#initclip [order:Number]`

指示初始化动作块的开始。当同时初始化多个剪辑时，您可以使用 `order` 参数来指定先执行哪个初始化动作。在定义影片剪辑元件时，将执行初始化动作。如果影片剪辑是导出的元件，则初始化动作将在执行 SWF 文件的第 1 帧上的动作之前执行。否则，组件初始化动作将在包含关联影片剪辑元件的第一个实例的帧的动作之前执行，并且两者是紧挨着发生的。

初始化动作仅在播放 SWF 文件时执行一次。它们用于一次性初始化，例如类定义和注册。

不要将分号 (;) 放在包含 `#initclip` 指令的行的末尾。

可用性：Flash Player 6.0；ActionScript 1.0

## 参数

**order:**Number [ 可选 ] - 一个非负整数，指定 #initclip 代码块的执行顺序。这是一个可选参数。该值必须通过使用整数文本（只允许十进制值，不允许十六进制值）来指定，而不能使用变量来指定。如果一个影片剪辑元件中包含多个 #initclip 块，则编译器对该影片剪辑元件中的所有 #initclip 块使用在该元件中指定的最后一个 order 值。

## 示例

在下面的示例中，**ActionScript** 放置在某影片剪辑实例中的第 1 帧上。同一目录中还放置了一个文本文件 **variables.txt**。

```
#initclip

trace("initializing app");

var variables:LoadVars = new LoadVars();

variables.load("variables.txt");

variables.onLoad = function(success:Boolean) {

    trace("variables loaded:"+success);

    if (success) {
        for (i in variables) {
            trace("variables."+i+" = "+variables[i]);
        }
    }
};

#endinitclip
```

# 常数

常数是一个用于表示其值永远不会改变的属性的变量。本节说明每个脚本都可以使用的全局常数。

## 常数摘要

修饰符	常数	说明
	false	一个表示与 true 相反的唯一布尔值。
	Infinity	指定表示正无穷大的 IEEE-754 值。
	-Infinity	指定表示负无穷大的 IEEE-754 值。
	NaN	一个预定义的变量，对于 NaN（非数字）具有 IEEE-754 值。
	newline	插入一个回车符（\r），该回车符在由代码生成的文本输出中插入一个空行。
	null	一个可以分配给变量的或由未提供数据的函数返回的特殊值。
	true	一个表示与 false 相反的唯一布尔值。
	undefined	一个特殊值，通常用于指示变量尚未赋值。

## false 常数

一个表示与 true 相反的唯一布尔值。

如果自动设置数据类型将 false 转换为数字，则它变为 0；如果将 false 转换为字符串，则它变为 "false"。

可用性：Flash Player 5；ActionScript 1.0

### 示例

此示例说明自动设置数据类型如何将 false 转换为数字以及转换为字符串：

```
var bool1:Boolean = Boolean(false);

// converts it to the number 0
trace(1 + bool1); // outputs 1

// converts it to a string
trace("String: " + bool1); // outputs String: false
```

## Infinity 常数

指定表示正无穷大的 IEEE-754 值。此常数的值与 `Number.POSITIVE_INFINITY` 相同。

可用性：Flash Player 5；ActionScript 1.0

另请参见

[POSITIVE\\_INFINITY](#) ([Number.POSITIVE\\_INFINITY](#) 属性)

## -Infinity 常数

指定表示负无穷大的 IEEE-754 值。此常数的值与 `Number.NEGATIVE_INFINITY` 相同。

可用性：Flash Player 5；ActionScript 1.0

另请参见

[NEGATIVE\\_INFINITY](#) ([Number.NEGATIVE\\_INFINITY](#) 属性)

## NaN 常数

一个预定义的变量，对于 NaN（非数字）具有 IEEE-754 值。要确定一个数字是否为 NaN，请使用 `isNaN()`。

可用性：Flash Player 5；ActionScript 1.0

另请参见

[isNaN](#) 函数，[NaN](#) ([Number.NaN](#) 属性)

## newline 常数

插入一个回车符 (`\r`)，该回车符在由代码生成的文本输出中插入一个空行。使用 `newline` 为由您的代码中的函数或语句检索的信息留出空间。

可用性：Flash Player 4；ActionScript 1.0



## 示例

下面的示例说明 `newline` 如何将 `trace()` 语句的输出显示在多行上。

```
var myName:String = "Lisa", myAge:Number = 30;
trace(myName+myAge);
trace("-----");
trace(myName+newline+myAge);
// output:
Lisa30
-----
Lisa
30
```

另请参见

[trace 函数](#)

## null 常数

一个可以分配给变量的或由未提供数据的函数返回的特殊值。您可以使用 `null` 表示缺少或不具有已定义数据类型的值。

可用性: Flash Player 5 ; ActionScript 1.0

## 示例

以下示例检查索引数组的前六个值，如果没有设置值（即值 `== null`），则输出一条消息：

```
var testArray:Array = new Array();
testArray[0] = "fee";
testArray[1] = "fi";
testArray[4] = "foo";

for (i = 0; i < 6; i++) {
    if (testArray[i] == null) {
        trace("testArray[" + i + "] == null");
    }
}
```

输出如下所示：

```
testArray[2] == null
testArray[3] == null
testArray[5] == null
```

## true 常数

一个表示与 false 相反的唯一布尔值。如果自动设置数据类型将 true 转换为数字，则它变为 1；如果将 true 转换为字符串，则它变为 "true"。

可用性：Flash Player 5；ActionScript 1.0

### 示例

下面的示例说明 true 在 if 语句中的使用：

```
var shouldExecute:Boolean;
// ...
// code that sets shouldExecute to either true or false goes here
// shouldExecute is set to true for this example:

shouldExecute = true;

if (shouldExecute == true) {
    trace("your statements here");
}

// true is also implied, so the if statement could also be written:
// if (shouldExecute) {
//     trace("your statements here");
// }
```

下面的示例说明自动设置数据类型如何将 true 转换为数字 1：

```
var myNum:Number;
myNum = 1 + true;
trace(myNum); // output: 2
```

### 另请参见

[false 常数](#)，[Boolean](#)

## undefined 常数

一个特殊值，通常用于指示变量尚未赋值。对未定义值的引用会返回特殊值 `undefined`。  
**ActionScript** 代码 `typeof(undefined)` 返回字符串 `"undefined"`。`undefined` 类型的唯一值是 `undefined`。

在为 **Flash Player 6** 或更低版本发布的文件中，`String(undefined)` 的值为 `""`（一个空字符串）。在为 **Flash Player 7** 或更高版本发布的文件中，`String(undefined)` 的值为 `"undefined"`（`undefined` 转换为一个字符串）。

在为 **Flash Player 6** 或更低版本发布的文件中，`Number(undefined)` 的值是 `0`。在为 **Flash Player 7** 或更高版本发布的文件中，`Number(undefined)` 的值是 `NaN`。

值 `undefined` 与特殊值 `null` 类似。使用等于运算符 (`==`) 对 `null` 和 `undefined` 进行比较时，它们的比较结果为相等。但是，使用全等运算符 (`===`) 对 `null` 和 `undefined` 进行比较时，它们的比较结果为不相等。

可用性：Flash Player 5；ActionScript 1.0

### 示例

在以下示例中，变量 `x` 尚未声明，所以其值为 `undefined`。

在代码的第一部分，使用等于运算符 (`==`) 对 `x` 的值与 `undefined` 值进行比较，并将相应的结果发送到“输出”面板。

在代码的第二部分，使用等于运算符 (`==`) 对值 `null` 和值 `undefined` 进行比较。

```
// x has not been declared
trace("The value of x is "+x);

if (x == undefined) {
    trace("x is undefined");
} else {
    trace("x is not undefined");
}

trace("typeof (x) is "+typeof (x));

if (null == undefined) {
    trace("null and undefined are equal");
} else {
    trace("null and undefined are not equal");
}
```

下面的结果显示在“输出”面板中。

```
The value of x is undefined
x is undefined
typeof (x) is undefined
null and undefined are equal
```

# 全局函数

本节包含了一组内置函数，凡是使用 **ActionScript** 的 **SWF** 文件的任何部分都可使用这些函数。这些全局函数涵盖了各种各样的常见编程任务，如处理数据类型 (`Boolean()`、`int()` 等)、生成调试信息 (`trace()`) 以及与 **Flash Player** 或浏览器进行通讯 (`fscommand()`)。

## 全局函数摘要

修饰符	签名	说明
	<code>Array([numElements:Number], [elementN:Object])</code>	创建一个新的空数组，或者将指定的元素转换为数组。
	<code>asfunction(function:String, parameter:String)</code>	用于 HTML 文本字段中 URL 的特殊协议，该协议允许 HREF 链接调用 ActionScript 函数。
	<code>Boolean(expression:Object)</code>	将参数 <i>expression</i> 转换为布尔值并返回 <code>true</code> 或 <code>false</code> 。
	<code>call(frame:Object)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此动作，而推荐使用 <code>function</code> 语句。 在被调用帧中执行脚本，而不将播放头移动到该帧。
	<code>chr(number:Number)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String.fromCharCode()</code> 。 将 ASCII 代码数字转换为字符。
	<code>clearInterval(intervalID:Number)</code>	停止 <code>setInterval()</code> 调用。
	<code>duplicateMovieClip(target:Object, newname:String, depth:Number)</code>	当 SWF 文件正在播放时，创建一个影片剪辑的实例。
	<code>escape(expression:String)</code>	将参数转换为字符串，并以 URL 编码格式对其进行编码，在这种格式中，所有非字母数字的字符都替换为 % 十六进制序列。
	<code>eval(expression:Object)</code>	按照名称访问变量、属性、对象或影片剪辑。
	<code>fscommand(command:String, parameters:String)</code>	使 SWF 文件能够与 Flash Player 或承载 Flash Player 的程序（如 Web 浏览器）进行通讯。

修饰符	签名	说明
	<code>getProperty(my_mc:String, property)</code>	返回影片剪辑 <i>my_mc</i> 的指定属性的值。
	<code>getTimer()</code>	返回自 SWF 文件开始播放时起已经过的毫秒数。
	<code>getURL(url:String, [window:String], [method:String])</code>	将来自特定 URL 的文档加载到窗口中，或将变量传递到位于所定义的 URL 的另一个应用程序。
	<code>getVersion()</code>	返回一个包含 Flash Player 版本和平台信息的字符串。
	<code>gotoAndPlay([scene:String], frame:Object)</code>	将播放头转到场景中指定的帧并从该帧开始播放。
	<code>gotoAndStop([scene:String], frame:Object)</code>	将播放头转到场景中指定的帧并停止播放。
	<code>ifFrameLoaded([scene:String], frame:Object)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。此函数已不推荐使用。Macromedia 建议您使用 <code>MovieClip._framesloaded</code> 属性。检查特定帧的内容是否在本地图可用。
	<code>int(value:Number)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>Math.round()</code> 。通过截断小数值得将小数转换为整数值。
	<code>isFinite(expression:Object)</code>	计算 <i>expression</i> ，如果结果为有限数，则返回 <code>true</code> ；如果为无穷大或负无穷大，则返回 <code>false</code> 。
	<code>isNaN(expression:Object)</code>	计算参数，如果值为 NaN（非数字），则返回 <code>true</code> 。
	<code>length(expression:String, variable:Object)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。此函数及所有字符串函数已不推荐使用。Macromedia 建议您使用 <code>String</code> 类的方法和 <code>String.length</code> 属性来执行相同的操作。返回指定字符串或变量的长度。
	<code>loadMovie(url:String, target:Object, [method:String])</code>	在播放原始 SWF 文件的同时将 SWF 文件或 JPEG 文件加载到 Flash Player 中。

修饰符	签名	说明
	<code>loadMovieNum(url:String, level:Number, [method:String])</code>	在播放原来加载的 SWF 文件的同时将 SWF 文件或 JPEG 文件加载到 Flash Player 的某个级别中。
	<code>loadVariables(url:String, target:Object, [method:String])</code>	从外部文件（例如文本文件，或由 ColdFusion、CGI 脚本、Active Server Page (ASP)、PHP 或 Perl 脚本生成的文本）中读取数据，并设置目标影片剪辑中变量的值。
	<code>loadVariablesNum(url:String, level:Number, [method:String])</code>	从外部文件（如文本文件，或由 ColdFusion、CGI 脚本、ASP、PHP 或 Perl 脚本生成的文本）中读取数据，并设置 Flash Player 的某个级别中的变量的值。
	<code>mbchr(number:Number)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String.fromCharCode()</code> 方法。 将 ASCII 代码数字转换为多字节字符。
	<code>mblength(string:String)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String</code> 类的方法和属性。 返回多字节字符串的长度。
	<code>mbord(character:String)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String.charCodeAt()</code> 。 将指定字符转换为多字节数字。
	<code>mbsubstring(value:String, index:Number, count:Number)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String.substr()</code> 。 从多字节字符串中提取新的多字节字符串。
	<code>MMExecute(command:String)</code>	允许您从 ActionScript 中发出 Flash JavaScript API (JSAPI) 命令。
	<code>nextFrame()</code>	将播放头转到下一帧。
	<code>nextScene()</code>	将播放头转到下一场景的第 1 帧。
	<code>Number(expression:Object)</code>	将参数 <i>expression</i> 转换为数字。
	<code>Object([value:Object])</code>	创建一个新的空对象，或者将指定的数字、字符串或布尔值转换为一个对象。
	<code>on(mouseEvent:Object)</code>	指定触发动作的鼠标事件或按键。
	<code>onClipEvent(movieEvent:Object)</code>	触发为特定影片剪辑实例定义的动作。
	<code>ord(character:String)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String</code> 类的方法和属性。 将字符转换为 ASCII 代码数字。
	<code>parseFloat(string:String)</code>	将字符串转换为浮点数。

修饰符	签名	说明
	<code>parseInt(expression:String, [radix:Number])</code>	将字符串转换为整数。
	<code>play()</code>	在时间轴中向前移动播放头。
	<code>prevFrame()</code>	将播放头转到前一帧。
	<code>prevScene()</code>	将播放头转到前一场景的第 1 帧。
	<code>print(target:Object, boundingBox:String)</code>	根据在参数（bmovie、bmax 或 bframe）中指定的边界打印 target 影片剪辑。
	<code>printAsBitmap(target:Object, boundingBox:String)</code>	根据在参数（bmovie、bmax 或 bframe）中指定的边界将 target 影片剪辑打印为位图。
	<code>printAsBitmapNum(level:Number, boundingBox:String)</code>	根据在参数（bmovie、bmax 或 bframe）中指定的边界将 Flash Player 中的某个级别打印为位图。
	<code>printNum(level:Number, boundingBox:String)</code>	根据在 boundingBox 的参数（bmovie、bmax 或 bframe）中指定的边界打印 Flash Player 中的级别。
	<code>random(value:Number)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>Math.random()</code> 。返回一个随机整数，该整数介于 0 到用 value 参数中指定的整数减 1 后的整数之间。
	<code>removeMovieClip(target:Object)</code>	删除指定的影片剪辑。
	<code>setInterval(functionReference:Function, interval:Number, [param:Object], objectReference:Object, methodName:String)</code>	在播放 SWF 文件时，每隔一定时间就调用函数或对象的方法。
	<code>setProperty(target:Object, property:Object, expression:Object)</code>	当影片剪辑播放时，更改影片剪辑的属性值。
	<code>showRedrawRegions(enable:Boolean, [color:Number])</code>	使调试器播放器能够描画出正在重绘的屏幕区域的轮廓。
	<code>startDrag(target:Object, [lock:Boolean], [left,top,right,bottom:Number])</code>	使 target 影片剪辑在影片播放过程中可拖动。
	<code>stop()</code>	停止当前正在播放的 SWF 文件。
	<code>stopAllSounds()</code>	在不停止播放头的情况下停止 SWF 文件中当前正在播放的所有声音。

修饰符	签名	说明
	<code>stopDrag()</code>	停止当前的拖动操作。
	<code>String(expression:Object)</code>	返回指定参数的字符串表示形式。
	<code>substring(string:String, index:Number, count:Number)</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>String.substr()</code> 。提取部分字符串。
	<code>targetPath(targetObject:Object)</code>	返回包含 <i>movieClipObject</i> 的目标路径的字符串。
	<code>tellTarget(target:String, statement(s))</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。 Macromedia 建议使用点 (.) 记号和 with 语句。将在 <i>statements</i> 参数中指定的指令应用于在 <i>target</i> 参数中指定的时间轴。
	<code>toggleHighQuality()</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此函数，而推荐使用 <code>_quality</code> 。在 Flash Player 中启用和禁用消除锯齿功能。
	<code>trace(expression:Object)</code>	计算表达式并输出结果。
	<code>unescape(string:String)</code>	将参数 <i>x</i> 作为字符串计算，将该字符串从 URL 编码格式解码（将所有十六进制序列转换为 ASCII 字符），并返回该字符串。
	<code>unloadMovie(target:Object)</code>	从 Flash Player 中删除通过 <code>loadMovie()</code> 加载的影片剪辑。
	<code>unloadMovieNum(level:Number)</code>	从 Flash Player 中删除通过 <code>loadMovieNum()</code> 加载的 SWF 或图像。
	<code>updateAfterEvent()</code>	当在处理函数内调用它或使用 <code>setInterval()</code> 调用它时更新显示。



# Array 函数

`Array()` : Array

`Array(numElements:Number)` : Array

`Array(element0:Object, [element1, element2, ...elementN])` : Array

创建一个长度为 0 或更大的新数组，或者创建由一系列指定的元素（这些元素可能具有不同的数据类型）填充的数组。

使用 `Array()` 创建下列各项之一：

- 空数组
- 具有特定长度，但其元素未定义值的数组
- 其元素具有特定值的数组

使用此函数类似于使用 **Array** 构造函数创建数组（请参见“**Array** 类的构造函数”）。

您可以传递一个数字 (`numElements`) 或者传递包含一个或多个不同类型的一系列元素 (`element0, element1, ... elementN`)。

能够接受一种以上数据类型的参数以 `Object` 类型在签名中列出。

可用性：Flash Player 6；ActionScript 1.0

## 参数

**numElements**:Number [ 可选 ] - 一个指定数组中元素数量的正整数。您可以指定 `numElements`，也可以指定元素列表，但不能同时指定两者。

**elementN**:Object [ 可选 ] - 一个或多个参数，`element0, element1, ... , elementN`，参数值可为任意类型。能够接受一种以上数据类型的参数以 `Object` 类型列出。您可以指定 **numElements**，也可以指定元素列表，但不能同时指定两者。

## 返回

Array - 一个数组。

## 示例

```
var myArray:Array = Array();
myArray.push(12);
trace(myArray); //traces 12
myArray[4] = 7;
trace(myArray); //traces 12,undefined,undefined,undefined,7
```

用法 2：下面的示例创建一个长度为 4 但没有定义任何元素的数组：

```
var myArray:Array = Array(4);
trace(myArray.length); // traces 4
trace(myArray); // traces undefined,undefined,undefined,undefined
```

用法 3: 下面的示例创建一个具有三个已定义元素的数组:

```
var myArray:Array = Array("firstElement", "secondElement", "thirdElement");
trace (myArray); // traces firstElement,secondElement,thirdElement
Unlike the Array class constructor, the Array() function does not use the
keyword new .
```

另请参见

[Array](#)

## asfunction 协议

asfunction:function:Function, parameter:String

用于 HTML 文本字段中 URL 的特殊协议, 该协议允许 HREF 链接调用 **ActionScript** 函数。在 HTML 文本字段中, 可以使用 HTML A 标签创建链接。A 标签的 HREF 属性包含使用 HTTP、HTTPS 或 FTP 等标准协议的 URL。asfunction 协议是特定于 Flash 的一个附加协议, 它可使链接调用 **ActionScript** 函数。

可用性: Flash Player 5 ; **ActionScript** 1.0

### 参数

**function**:String - 函数的标识符。

**parameter**:String - 一个字符串, 传递给在 **function** 参数中命名的函数。

### 示例

在下面的示例中, 定义了 playMP3() 函数。创建并设置了 **TextField** 对象 **list\_txt**, 因此可以呈现 HTML 文本。文本 **Track 1** 和 **Track 2** 是文本字段中的链接。当用户单击任一链接并播放作为 **asfunction** 调用的参数传递的 MP3 时, 会调用 playMP3() 函数。

```
var myMP3:Sound = new Sound();
function playMP3(mp3:String) {
    myMP3.loadSound(mp3, true);
    myMP3.onLoad = function(success) {
        if (!success) {
            // code to handle errors here
        }
    };
}
this.createTextField("list_txt", this.getNextHighestDepth(), 0, 0, 200,
    100);
list_txt.autoSize = true;
list_txt.html = true;
list_txt.multiline = true;
list_txt.htmlText = "<a href=\"asfunction:playMP3, track1.mp3\">Track 1</a><br>";
```

```
list_txt.htmlText += "<a href=\"asfunction:playMP3, track2.mp3\">Track 2</a><br>";
```

当单击某链接时，该 MP3 声音文件会流入 Flash Player。

另请参见

[htmlText \(TextField.htmlText 属性\)](#)

## Boolean 函数

`Boolean(expression:Object) : Boolean`

将 `expression` 参数转换为布尔值，并返回一个下面列表中说明的值：

- 如果 `expression` 是布尔值，则返回值为 `expression`。
- 如果 `expression` 是数字，则当该数字不为 0 时，返回值为 `true`；否则，返回值为 `false`。

如果 `expression` 为字符串，则返回值如下所示：

- 在为 Flash Player 6 及更低版本发布的文件中，该字符串首先转换为一个数字。如果该数字不是 0，则值为 `true`；否则，返回值为 `false`。
- 在为 Flash Player 7 及更高版本发布的文件中，如果该字符串的长度大于 0，则结果为 `true`；如果该字符串是空字符串，则值为 `false`。

如果 `expression` 为字符串，则在该字符串的长度大于 0 时，结果为 `true`；如果该字符串是空字符串，则值为 `false`。

- 如果 `expression` 是 `undefined` 或 `NaN`（非数字），则返回值为 `false`。
- 如果 `expression` 是影片剪辑或对象，则返回值为 `true`。

与 `Boolean` 类构造函数不同的是，`Boolean()` 函数不使用关键字 `new`。此外，如果未指定任何参数，则 `Boolean` 类构造函数会将 `Boolean` 对象初始化为 `false`，而 `Boolean()` 函数在未指定参数时会返回 `undefined`。

可用性：Flash Player 5；ActionScript 1.0

### 参数

`expression:Object` - 一个要转换为布尔值的表达式。

### 返回

`Boolean` - 一个布尔值。

## 示例

```
trace(Boolean(-1)); // output: true
trace(Boolean(0)); // output: false
trace(Boolean(1)); // output: true
```

```
trace(Boolean(true)); // output: true
trace(Boolean(false)); // output: false
```

```
trace(Boolean("true")); // output: true
trace(Boolean("false")); // output: true
```

```
trace(Boolean("Craiggers")); // output: true
trace(Boolean("")); // output: false
```

如果文件是为 **Flash Player 6** 及更低版本发布的，则前面三个示例的结果将会不同：

```
trace(Boolean("true")); // output: false
trace(Boolean("false")); // output: false
trace(Boolean("Craiggers")); // output: false
```

此示例说明 `Boolean()` 函数的使用 and **Boolean** 类的使用之间的重要区别。`Boolean()` 函数创建布尔值，而 **Boolean** 类创建 **Boolean** 对象。布尔值是按值进行比较的，而 **Boolean** 对象是按引用进行比较的。

```
// Variables representing Boolean values are compared by value
var a:Boolean = Boolean("a"); // a is true
var b:Boolean = Boolean(1); // b is true
trace(a==b); // true
```

```
// Variables representing Boolean objects are compared by reference
var a:Boolean = new Boolean("a"); // a is true
var b:Boolean = new Boolean(1); // b is true
trace(a == b); // false
```

另请参见

[Boolean](#)

## call 函数

`call(frame)`

自 **Flash Player 5** 后不推荐使用。不推荐使用此动作，而推荐使用 `function` 语句。

在被调用帧中执行脚本，而不将播放头移动到该帧。在脚本执行后，局部变量将消失。

- 如果变量不是在块 (`{}`) 内声明的，但执行该动作列表时使用的是 `call()` 动作，则这些变量是局部变量，并且它们在当前列表结束时到期。
- 如果变量不是在块中声明的，且执行当前动作列表时使用的不是 `call()` 动作，则这些变量被解释为时间轴变量。

可用性：Flash Player 4； ActionScript 1.0

### 参数

`frame:Object` - 时间轴中帧的标签或编号。

另请参见

[function 语句](#)，[call \(Function.call 方法\)](#)

## chr 函数

`chr(number:Number) : String`

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用

`String.fromCharCode()`。

将 ASCII 代码数字转换为字符。

可用性：Flash Player 4； ActionScript 1.0

### 参数

`number:Number` - 一个 ASCII 代码数字。

### 返回

`String` - 指定的 ASCII 代码的字符值。

### 示例

下面的示例将数字 **65** 转换为字母 **A**，并将其分配给变量 `myVar`：`myVar = chr(65);`

另请参见

[fromCharCode \(String.fromCharCode 方法\)](#)

## clearInterval 函数

`clearInterval(intervalID:Number) : Void`

停止 `setInterval()` 调用。

可用性：Flash Player 6；ActionScript 1.0

### 参数

`intervalID:Number` - 通过调用 `setInterval()` 而返回的数字型（整数）标识符。

### 示例

下面的示例首先设置一个间隔调用，然后将其清除：

```
function callback() {  
    trace("interval called: "+getTimer()+" ms.");  
}
```

```
var intervalID:Number = setInterval(callback, 1000);
```

当使用完该函数后，必须清除该间隔。创建一个名为 `clearInt_btn` 的按钮并使用下面的

**ActionScript** 清除 `setInterval()`：

```
clearInt_btn.onRelease = function(){  
    clearInterval( intervalID );  
    trace("cleared interval");  
};
```

另请参见

[setInterval 函数](#)

## duplicateMovieClip 函数

`duplicateMovieClip(target:String, newname:String, depth:Number) : Void`  
`duplicateMovieClip(target:MovieClip, newname:String, depth:Number) : Void`

当 SWF 文件正在播放时，创建一个影片剪辑的实例。无论播放头在原始影片剪辑中处于什么位置，在重复的影片剪辑中，播放头始终从第 1 帧开始。原始影片剪辑中的变量不会复制到重复的影片剪辑中。使用 `removeMovieClip()` 函数或方法可以删除用 `duplicateMovieClip()` 创建的影片剪辑实例。

可用性：Flash Player 4；ActionScript 1.0

## 参数

**target:Object** - 要复制的影片剪辑的目标路径。此参数可以是一个字符串（例如 `"my_mc"`），也可以是对影片剪辑实例的直接引用（例如 `my_mc`）。能够接受一种以上数据类型以 `Object` 类型列出。

**newname:String** - 所复制的影片剪辑的唯一标识符。

**depth:Number** - 所复制的影片剪辑的唯一深度级别。深度级别是所复制的影片剪辑的堆叠顺序。这种堆叠顺序很像时间轴中图层的堆叠顺序；较低深度级别的影片剪辑隐藏在较高堆叠顺序的剪辑之下。必须为每个所复制的影片剪辑分配一个唯一的深度级别，以防止它替换已占用深度上的 `SWF` 文件。

## 示例

在下面的示例中，将创建一个名为 `img_mc` 的新影片剪辑实例。将一个图像加载到该影片剪辑中，然后复制 `img_mc` 剪辑。所复制的剪辑名为 `newImg_mc`，这个新剪辑将移至舞台上，因此它不会与原始剪辑重叠，并且将同一图像加载到第二个剪辑中。

```
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
img_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
duplicateMovieClip(img_mc, "newImg_mc", this.getNextHighestDepth());
newImg_mc._x = 200;
newImg_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

要删除重复的影片剪辑，可为名为 `myButton_btn` 的按钮添加以下代码。

```
this.myButton_btn.onRelease = function(){
    removeMovieClip(newImg_mc);
};
```

## 另请参见

[removeMovieClip 函数](#)，[duplicateMovieClip \(MovieClip.duplicateMovieClip 方法\)](#)，[removeMovieClip \(MovieClip.removeMovieClip 方法\)](#)

## escape 函数

`escape(expression:String) : String`

将参数转换为字符串，并以 `URL` 编码格式对其进行编码，在这种格式中，所有非字母数字的字符都替换为 `%` 十六进制序列。当用于 `URL` 编码的字符串中时，百分号 (`%`) 用于引入转义符，不与模运算符 (`%`) 等效。

可用性：Flash Player 5；ActionScript 1.0

## 参数

**expression:String** - 要转换为字符串并以 `URL` 编码格式进行编码的表达式。

返回

String - URL 编码的字符串。

示例

下面的代码生成结果 `someuser%40somedomain%2Ecom`:

```
var email:String = "someuser@somedomain.com";
trace(escape(email));
```

在此示例中，`@` 符号被替换为 `%40`，而点符号 `.` 被替换为 `%2E`。如果您要尝试将信息传递到远程服务器，并且数据中包含特殊字符（例如，`&` 或 `?`），则它非常有用，如下面的代码所示：

```
var redirectUrl:String = "http://
    www.somedomain.com?loggedin=true&username=Gus";
getURL("http://www.myothersite.com?returnurl="+ escape(redirectUrl));
```

另请参见

[unescape 函数](#)

## eval 函数

```
eval(expression:Object) : Object
eval(expression:String) : Object
```

按照名称访问变量、属性、对象或影片剪辑。如果表达式是变量或属性，则返回该变量或属性的值。如果表达式是对象或影片剪辑，则返回对该对象或影片剪辑的引用。如果无法找到表达式中列举的元素，则返回 `undefined`。

在 **Flash 4** 中，`eval()` 用于模拟数组；在 **Flash 5** 或更高版本中，您应该使用 `Array` 类来模拟数组。

在 **Flash 4** 中，您还可以使用 `eval()` 动态地设置和检索变量值或实例名称。然而，也可以使用数组访问运算符 `[]` 来实现这一点。

在 **Flash5** 或更高版本中，不能使用 `eval()` 动态设置和检索变量值或实例名称，因为不能在等式的左侧使用 `eval()`。例如，将代码

```
eval ("var" + i) = "first";
```

替换为：

```
this["var"+i] = "first"
```

或者替换为：

```
set ("var" + i, "first");
```

可用性：Flash Player 5；ActionScript 1.0



## 参数

**expression:Object** - 要检索的变量、属性、对象或影片剪辑的名称。此参数可以是一个字符串，也可以是对对象实例的直接引用（也就是说，引号（" "）是可选的）。

## 返回

**Object** - 一个值（对对象或影片剪辑的引用）或 **undefined**。

## 示例

下面的示例使用 **eval()** 为动态命名的影片剪辑设置属性。此 **ActionScript** 为三个影片剪辑（分别称为 **square1\_mc**、**square2\_mc** 和 **square3\_mc**）设置 **\_rotation** 属性。

```
for (var i = 1; i <= 3; i++) {  
    setProperty(eval("square"+i+"_mc"), _rotation, 5);  
}
```

也可以使用以下 **ActionScript**:

```
for (var i = 1; i <= 3; i++) {  
    this["square"+i+"_mc"]._rotation = -5;  
}
```

另请参见

[Array](#), [set variable 语句](#)

# fscommand 函数

**fscommand(command:String, parameters:String) : Void**

使 SWF 文件与 **Flash Player** 或承载 **Flash Player** 的程序（如 **Web 浏览器**）进行通讯。还可以使用 **fscommand()** 函数将消息传递给 **Macromedia Director**，或者传递给 **Visual Basic (VB)**、**Visual C++** 和其它可承载 **ActiveX** 控件的程序。

**fscommand()** 函数使 SWF 文件与 **Web** 页中的脚本能进行通讯。不过，脚本访问是由 **Web** 页的 **allowScriptAccess** 设置控制的。（您可以在嵌入 SWF 文件的 **HTML** 代码中设置此属性 - 例如，在 **Internet Explorer** 的 **PARAM** 标签或 **Netscape** 的 **EMBED** 标签中。）当 **allowScriptAccess** 设置为 "never" 时，SWF 文件无法访问 **Web** 页脚本。对于 **Flash Player 7** 及更高版本，当 **allowScriptAccess** 设置为 "always" 时，SWF 文件始终可以访问 **Web** 页脚本。当 **allowScriptAccess** 设置为 "sameDomain" 时，只允许从与该 **Web** 页位于同一域中的 SWF 文件进行脚本访问；对于以前版本的 **Flash Player**，始终允许脚本访问。如果在 **HTML** 页中未指定 **allowScriptAccess**，则默认情况下，对于第 8 版及更高版本的 SWF 文件，该属性设置为 "sameDomain"；对于第 7 版及更低版本的 SWF 文件，设置为 "always"。

用法 1: 若要使用 `fscommand()` 将消息发送给 **Flash Player**, 必须使用预定义的命令和参数。下表列出了可以为 `fscommand()` 函数的 `command` 参数和 `parameters` 参数指定的值。这些值控制在 **Flash Player** 中播放的 SWF 文件, 包括放映文件。(放映文件是以可作为独立应用程序运行 (也就是说, 不需要使用 **Flash Player** 即可运行) 的格式保存的 SWF 文件。)

命令	参数	目的
quit	无	关闭放映文件。
fullscreen	true 或者 false	指定 true 可将 <b>Flash Player</b> 设置为全屏模式。指定 false 可将播放器返回到标准菜单视图。
allowscale	true 或者 false	指定 false 可设置播放器始终按 SWF 文件的原始大小绘制 SWF 文件, 从不进行缩放。指定 true 会强制将 SWF 文件缩放到播放器的 100% 大小。
showmenu	true 或者 false	指定 true 可启用整个上下文菜单项集合。指定 false 将隐藏除“关于 <b>Flash Player</b> ”和“设置”外的所有上下文菜单项。
exec	应用程序的路径	在放映文件内执行应用程序。
trapallkeys	true 或者 false	指定 true 可将所有按键事件 (包括快捷键) 发送到 <b>Flash Player</b> 中的 <code>onClipEvent(keyDown/keyUp)</code> 处理函数。

可用性:

- 表中描述的命令在 **Web** 播放器中都不可用。
- 所有这些命令在独立的应用程序 (例如, 放映文件) 中都可用。
- 只有 `allowscale` 和 `exec` 在测试影片播放器中可用。

`exec` 命令只能包含字符 **A-Z**、**a-z**、**0-9**、句号 (.) 和下划线 ( \_ )。 `exec` 命令仅在 `fscommand` 子目录中运行。也就是说, 如果您使用 `exec` 命令调用应用程序, 该应用程序必须位于名为 `fscommand` 的子目录中。 `exec` 命令只在 **Flash** 放映文件内起作用。

用法 2: 若要使用 `fscommand()` 向 **Web** 浏览器中的脚本语言 (例如 **JavaScript**) 发送消息, 您可以在 `command` 和 `parameters` 参数中传递任意两个参数。这些参数可以是字符串或表达式, 并在处理或捕获 `fscommand()` 函数的 **JavaScript** 函数中使用。

在 **Web** 浏览器中, `fscommand()` 调用 **JavaScript** 函数 `moviename_DoFScommand`, 该函数位于包含 SWF 文件的 **Web** 页中。对于 `moviename`, 提供您用于 `EMBED` 标签的 `NAME` 属性或 `OBJECT` 标签的 `ID` 属性的 **Flash** 对象的名称。如果对 SWF 文件分配名称 `myMovie`, 则调用 **JavaScript** 函数 `myMovie_DoFScommand`。

在包含 SWF 文件的 Web 页中, 设置 `allowScriptAccess` 属性以允许或拒绝 SWF 文件访问 Web 页的能力。(您可以在嵌入 SWF 文件的 HTML 代码中设置此属性, 例如, 在 Internet Explorer 的 PARAM 标签或 Netscape 的 EMBED 标签中。) 当 `allowScriptAccess` 设置为 "never" 时, 外出脚本处理始终失败。当 `allowScriptAccess` 设置为 "always" 时, 外出脚本处理始终成功。当它设置为 "sameDomain" 时, 只允许从与该 Web 页位于同一域中的 SWF 文件进行脚本访问。如果未在 Web 页中指定 `allowScriptAccess`, 则对于 Flash Player 8, 它默认为 "sameDomain"; 对于以前的 Flash Player 版本, 它默认为 "always"。

使用此函数时, 请考虑 Flash Player 安全模型。对于 Flash Player 8, 如果执行调用的 SWF 文件在只能与本地文件系统内容交互的沙箱或只能与远程内容交互的沙箱中, 并且所包含的 HTML 页在不受信任的沙箱中, 则不允许使用 `fscommand()` 函数。有关更多信息, 请参见以下部分:

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章, “了解安全性”
- Flash Player 8 安全性白皮书
- Flash Player 8 与安全相关的 API 白皮书

用法 3: `fscommand()` 函数可以将消息发送给 Macromedia Director。这些消息由 Lingo (Director 脚本语言) 解释为字符串、事件或可执行 Lingo 代码。如果消息为字符串或事件, 则必须编写 Lingo 代码才能从 `fscommand()` 函数接收该消息并在 Director 中执行动作。有关更多信息, 请访问 Director 支持中心, 网址为 [www.macromedia.com/support/director](http://www.macromedia.com/support/director)。

用法 4: 在 VisualBasic、Visual C++ 和可承载 ActiveX 控件的其它程序中, `fscommand()` 利用可使用环境的编程语言处理的两个字符串发送 VB 事件。有关更多信息, 请使用关键字 “Flash 方法” 搜索 Flash 支持中心, 网址为 [www.macromedia.com/go/flash\\_support\\_cn](http://www.macromedia.com/go/flash_support_cn)。

注意: 如果要为 Flash Player 8 或更高版本发布, 则 `ExternalInterface` 类可为以下通讯提供更好的性能: JavaScript 与 ActionScript 之间的通讯 (用法 2); ActionScript 与 VisualBasic、Visual C++ 或可承载 ActiveX 控件的其它程序之间的通讯 (用法 4)。您应该继续使用 `fscommand()` 将消息发送到 Flash Player (用法 1) 和 Macromedia Director (用法 3)。

可用性: Flash Player 3; ActionScript 1.0

## 参数

`command:String` - 传递给主机应用程序用于任何用途的一个字符串, 或传递给 Flash Player 的一个命令。

`parameters:String` - 传递给主机应用程序用于任何用途的一个字符串, 或传递给 Flash Player 的一个值。

## 示例

在下面的示例中，`fscommand()` 将 **Flash Player** 设置为在松开 `fullscreen_btn` 或 `unfullscreen_btn` 按钮时将 **SWF** 文件放大到整个显示器屏幕大小。

```
this.fullscreen_btn.onRelease = function() {  
    fscommand("fullscreen", true);  
};  
  
this.unfullscreen_btn.onRelease = function() {  
    fscommand("fullscreen", false);  
};
```

下面的示例将 `fscommand()` 应用于 **Flash** 中的某个按钮，以用于在 **HTML** 页中打开 **JavaScript** 消息框。消息本身作为 `fscommand` 参数发送到 **JavaScript**。

您必须将一个函数添加到包含 **SWF** 文件的 **Web** 页。该函数，即 `myDocument_DoFSCommand()`，等待一个 `fscommand()` 调用。在 **Flash** 中触发 `fscommand()`（例如，用户单击按钮）时，`command` 和 `parameter` 字符串将传递给 `myDocument_DoFSCommand()` 函数。可以在 **JavaScript** 或 **VBScript** 代码中以任何需要的方式使用所传递的字符串。在此示例中，该函数包含一个条件 `if` 语句，该语句检查命令字符串是否为 `"messagebox"`。如果是，则一个 **JavaScript** 警告框显示 `fscommand()` 函数的 `parameters` 字符串的内容。

```
function myDocument_DoFSCommand(command, args) {  
    if (command == "messagebox") {  
        alert(args);  
    }  
}
```

在 **Flash** 文档中，将 `fscommand()` 添加到一个按钮：

```
fscommand("messagebox", "This is a message box called from within Flash.")
```

也可以对 `fscommand()` 函数的参数使用表达式，如下面的示例所示：

```
fscommand("messagebox", "Hello, " + name + ", welcome to our website!")
```

要测试 **SWF** 文件，请选择“文件”>“发布预览”>“**HTML**”。如果在 **Flash** 中使用 **FSCommand** 模板（在“发布设置”对话框中，选择 **HTML** 标签）发布 **SWF** 文件，**Flash** 会自动插入 `myDocument_DoFSCommand()` 函数。**SWF** 文件的 `NAME` 和 `ID` 属性将为文件名。例如，对于文件 `myDocument.fla`，这些属性将设置为 `myDocument`。

另请参见

[ExternalInterface \(flash.external.ExternalInterface\)](#)

## getProperty 函数

getProperty(my\_mc:Object, property:Object) : Object

返回影片剪辑 *my\_mc* 的指定属性的值。

可用性: Flash Player 4 ; ActionScript 1.0

### 参数

**my\_mc:String** - 要检索其属性的影片剪辑的实例名称。

**property** - 影片剪辑的一个属性。

### 返回

Object - 指定属性的值。

### 示例

下面的示例创建新的影片剪辑 *someClip\_mc*，并在“输出”面板中显示影片剪辑 *someClip\_mc* 的 **Alpha** 值 (*\_alpha*):

```
this.createEmptyMovieClip("someClip_mc", 999);
trace("The alpha of "+getProperty(someClip_mc, _name)+" is:
      "+getProperty(someClip_mc, _alpha));
```

## getTimer 函数

getTimer() : Number

返回自 SWF 文件开始播放时起已经过的毫秒数。

可用性: Flash Player 4 ; ActionScript 1.0

### 返回

Number - 自 SWF 文件开始播放时起已经过的毫秒数。

### 示例

在下面的示例中，使用 *getTimer()* 和 *setInterval()* 函数来创建一个简单的计时器：

```
this.createTextField("timer_txt", this.getNextHighestDepth(), 0, 0, 100,
22);
function updateTimer():Void {
    timer_txt.text = getTimer();
}

var intervalID:Number = setInterval(updateTimer, 100);
```

## getURL 函数

`getURL(url:String, [window:String, [method:String]]) : Void`

将来自特定 URL 的文档加载到窗口中，或将变量传递到位于所定义的 URL 的另一个应用程序。若要测试此函数，请确保要加载的文件位于指定的位置。若要使用绝对 URL（例如，<http://www.myserver.com>），则需要网络连接。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**url:String** - 可从该处获取文档的 URL。

**window:String** [ 可选 ] - 指定应将文档加载到其中的窗口或 HTML 帧。您可输入特定窗口的名称，或从下面的保留目标名称中选择：

- `_self` 指定当前窗口中的当前帧。
- `_blank` 指定一个新窗口。
- `_parent` 指定当前帧的父级。
- `_top` 指定当前窗口中的顶级帧。

**method:String** [ 可选 ] - 用于发送变量的 GET 或 POST 方法。如果没有变量，则省略此参数。GET 方法将变量附加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送长字符串的变量。

### 示例

此示例将图像加载到影片剪辑中。当单击该图像时，会在一个新的浏览器窗口中加载新的 URL。

```
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
    target_mc.onRelease = function() {
        getURL("http://www.macromedia.com/software/flash/flashpro/", "_blank");
    };
};
var logo:MovieClipLoader = new MovieClipLoader();
logo.addListener(listenerObject);
logo.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    this.createEmptyMovieClip("macromedia_mc", this.getNextHighestDepth()));
```

在下面的示例中，使用 `getURL()` 来发送电子邮件：

```
myBtn_btn.onRelease = function(){
    getURL("mailto:you@somedomain.com");
};
```

在下面的 **ActionScript** 中，将使用 **JavaScript** 在 **SWF** 文件嵌入到浏览器窗口中时打开警告窗口（请注意，当使用 `getURL()` 调用 **JavaScript** 时，`url` 参数最多只能包含 508 个字符）：

```
myBtn_btn.onRelease = function(){
    getURL("javascript:alert('you clicked me')");
};
```

您也可以使用 **GET** 或 **POST** 发送变量。下面的示例使用 **GET** 将变量追加到 **URL**：

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
myBtn_btn.onRelease = function() {
    getURL("http://www.macromedia.com", "_blank", "GET");
};
```

下面的 **ActionScript** 使用 **POST** 在 **HTTP** 标头中发送变量。一定要在浏览器窗口中测试文档，因为变量可能是使用 **GET** 发送的：

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
getURL("http://www.macromedia.com", "_blank", "POST");
```

另请参见

[loadVariables 函数](#)，[send \(XML.send 方法\)](#)，[sendAndLoad \(XML.sendAndLoad 方法\)](#)

## getVersion 函数

`getVersion() : String`

返回一个包含 **Flash Player** 版本和平台信息的字符串。`getVersion` 函数只返回 **Flash Player 5** 或 **Flash Player** 更高版本的信息。

可用性：Flash Player 5；ActionScript 1.0

### 返回

`String` - 包含 **Flash Player** 版本和平台信息的一个字符串。

### 示例

下列示例跟踪播放 **SWF** 文件的 **Flash Player** 的版本号：

```
var flashVersion:String = getVersion();
trace(flashVersion); // output: WIN 8,0,1,0
trace($version); // output: WIN 8,0,1,0
trace(System.capabilities.version); // output: WIN 8,0,1,0
```

getVersion 函数返回以下字符串：

WIN 8,0,1,0

这个返回的字符串指示平台是 Microsoft Windows, Flash Player 的主要版本号为 8, 次要版本号为 1 (8.1)。

另请参见

[os \(capabilities.os 属性\)](#), [version \(capabilities.version 属性\)](#)

## gotoAndPlay 函数

gotoAndPlay([scene:String], frame:Object) : Void

将播放头转到场景中指定的帧并从该帧开始播放。如果未指定场景，则播放头将转到当前场景中的指定帧。只能在根时间轴上使用 *scene* 参数，不能在影片剪辑或文档中的其它对象的时间轴内使用该参数。

可用性：Flash Player 2；ActionScript 1.0

### 参数

**scene:String** [ 可选 ] - 一个字符串，指定播放头要转到其中的场景的名称。

**frame:Object** - 表示播放头转到的帧编号的数字，或者表示播放头转到的帧标签的字符串。

### 示例

在以下示例中，文档具有两个场景：sceneOne 和 sceneTwo。场景一包含第 10 帧上名为 newFrame 的帧标签以及 myBtn\_btn 和 myOtherBtn\_btn 这两个按钮。此 **ActionScript** 放置在主时间轴的第 1 场景的第 1 帧上。

```
stop();
myBtn_btn.onRelease = function(){
    gotoAndPlay("newFrame");
};
```

```
myOtherBtn_btn.onRelease = function(){
    gotoAndPlay("sceneTwo", 1);
};
```

当用户单击这两个按钮时，播放头会移至指定的位置并继续播放。

另请参见

[gotoAndPlay \(MovieClip.gotoAndPlay 方法\)](#), [nextFrame 函数](#), [play 函数](#), [prevFrame 函数](#)



## gotoAndStop 函数

`gotoAndStop([scene:String], frame:Object) : Void`

将播放头转到场景中指定的帧并停止播放。如果未指定场景，播放头将转到当前场景中的帧。只能在根时间轴上使用 *scene* 参数，不能在影片剪辑或文档中的其它对象的时间轴内使用该参数。

可用性：Flash Player 2 ； ActionScript 1.0

### 参数

**scene:String** [ 可选 ] - 一个字符串，指定播放头要转到其中的场景的名称。

**frame:Object** - 表示播放头转到的帧编号的数字，或者表示播放头转到的帧标签的字符串。

### 示例

在以下示例中，文档具有两个场景：sceneOne 和 sceneTwo。场景一包含第 10 帧上名为 newFrame 的帧标签以及 myBtn\_btn 和 myOtherBtn\_btn 这两个按钮。此 **ActionScript** 放置在主时间轴的第 1 场景的第 1 帧上：

```
stop();

myBtn_btn.onRelease = function(){
    gotoAndStop("newFrame");
};

myOtherBtn_btn.onRelease = function(){
    gotoAndStop("sceneTwo", 1);
};
```

当用户单击这两个按钮时，播放头会移至指定的位置并停止。

### 另请参见

[gotoAndStop \(MovieClip.gotoAndStop 方法\)](#)，[stop 函数](#)，[play 函数](#)，[gotoAndPlay 函数](#)

## ifFrameLoaded 函数

```
ifFrameLoaded([scene:String], frame) {  
    statement(s);  
}
```

自 **Flash Player 5** 后**不推荐使用**。此函数已不推荐使用。**Macromedia** 建议您使用 `MovieClip._framesloaded` 属性。

检查特定帧的内容是否在本机可用。使用 `ifFrameLoaded` 可在 **SWF** 文件的其余部分下载到本地计算机时开始播放简单的动画。使用 `_framesloaded` 与 `ifFrameLoaded` 的区别在于 `_framesloaded` 允许您添加自定义 `if` 或 `else` 语句。

**可用性:** **Flash Player 4** ; **ActionScript 1.0**

### 参数

`scene:String` [ 可选 ] - 一个字符串，它指定必须加载的场景的名称。

`frame:Object` - 在执行下一条语句之前必须加载的帧编号或帧标签。

### 另请参见

[\\_framesloaded](#) (`MovieClip._framesloaded` 属性), [addListener](#) (`MovieClipLoader.addListener` 方法)

## int 函数

```
int(value:Number) : Number
```

自 **Flash Player 5** 后**不推荐使用**。不推荐使用此函数，而推荐使用 `Math.round()`。

通过截断小数将小数转换为整数值。当 `value` 参数为正值时，此函数等效于 `Math.floor()`；当 `value` 参数为负值时，此函数等效于 `Math.ceil()`。

**可用性:** **Flash Player 4** ; **ActionScript 1.0**

### 参数

`value:Number` - 要舍入为整数的数字。

### 返回

`Number` - 截断的整数值。

### 另请参见

[round](#) (`Math.round` 方法), [floor](#) (`Math.floor` 方法), [ceil](#) (`Math.ceil` 方法)

## isFinite 函数

`isFinite(expression:Object) : Boolean`

计算 *expression*，如果结果为有限数，则返回 `true`；如果为无穷大或负无穷大，则返回 `false`。无穷大或负无穷大的出现指示有错误的数学条件，例如除以 0。

可用性：Flash Player 5；ActionScript 1.0

### 参数

`expression:Object` - 要计算的布尔值、变量或其它表达式。

### 返回

`Boolean` - 一个布尔值。

### 示例

下面的示例显示 `isFinite` 的返回值：

```
isFinite(56)
// returns true

isFinite(Number.POSITIVE_INFINITY)
//returns false
```

## isNaN 函数

`isNaN(expression:Object) : Boolean`

计算参数，如果值为 NaN（非数字），则返回 `true`。此函数可用于检查一个数学表达式是否成功地计算为一个数字。

可用性：Flash Player 5；ActionScript 1.0

### 参数

`expression:Object` - 要计算的布尔值、变量或其它表达式。

### 返回

`Boolean` - 一个布尔值。

## 示例

下面的代码显示了 `isNaN()` 函数的返回值:

```
trace( isNaN("Tree") );  
// returns true  
  
trace( isNaN(56) );  
// returns false  
  
trace( isNaN(Number.POSITIVE_INFINITY) );  
// returns false
```

下面的示例说明如何使用 `isNaN()` 检查数学表达式是否包含错误:

```
var dividend:Number;  
var divisor:Number;  
divisor = 1;  
trace( isNaN(dividend/divisor) );  
// output: true  
// The output is true because the variable dividend is undefined.  
// Do not use isNaN() to check for division by 0 because it will return  
// false.  
// A positive number divided by 0 equals Infinity (Number.POSITIVE_INFINITY).  
// A negative number divided by 0 equals -Infinity  
// (Number.NEGATIVE_INFINITY).
```

另请参见

[NaN 常数](#), [NaN \(Number.NaN 属性\)](#)

## length 函数

`length(expression:String)length(variable)`

自 **Flash Player 5** 后不推荐使用。此函数及所有字符串函数已不推荐使用。**Macromedia** 建议您使用 **String** 类的方法和 `String.length` 属性来执行相同的操作。

返回指定字符串或变量的长度。

可用性: **Flash Player 4**; **ActionScript 1.0**

### 参数

**expression:String** - 一个字符串。

**variable:Object** - 变量的名称。

### 返回

**Number** - 指定的字符串或变量的长度。

## 示例

下面的示例返回字符串 "Hello" 的长度：length("Hello")；结果为 5。

另请参见

" 字符串分隔符运算符，String，length (String.length 属性)

## loadMovie 函数

```
loadMovie(url:String, target:Object, [method:String]) : Void
```

```
loadMovie(url:String, target:String, [method:String]) : Void
```

在播放原始 SWF 文件时，将 SWF、JPEG、GIF 或 PNG 文件加载到 Flash Player 中的影片剪辑中。在 Flash Player 8 中添加了对非动画 GIF 文件、PNG 文件和渐进式 JPEG 文件的支持。如果加载动画 GIF，则仅显示第一帧。



如果您要监视下载的进度，则使用 MovieClipLoader.loadClip() 而不是此函数。

使用 loadMovie() 函数可以一次显示多个 SWF 文件，并且无需加载另一个 HTML 文档即可在 SWF 文件之间进行切换。如果不使用 loadMovie() 函数，则 Flash Player 显示单个 SWF 文件。

如果要将 SWF 文件或 JPEG 文件加载到特定的级别中，请使用 loadMovieNum() 而不是 loadMovie()。

如果 SWF 文件加载到目标影片剪辑，则可使用该影片剪辑的目标路径来定位加载的 SWF 文件。加载到目标的 SWF 文件或图像会继承目标影片剪辑的位置、旋转和缩放属性。加载的图像或 SWF 文件的左上角与目标影片剪辑的注册点对齐。或者，如果目标为根时间轴，则该图像或 SWF 文件的左上角与舞台的左上角对齐。

使用 unloadMovie() 可删除用 loadMovie() 加载的 SWF 文件。

使用此函数时，请考虑 Flash Player 安全模型。

对于 Flash Player 8：

- 如果执行调用的影片剪辑在只能与本地文件系统的内容交互的沙箱中，并且被加载的影片剪辑来自网络沙箱，则不允许加载。
- 如果执行调用的 SWF 文件在网络沙箱中并且要加载的影片剪辑是本地的，则不允许加载。
- 从受信任的本地沙箱或从只能与远程内容交互的沙箱访问网络沙箱需要通过跨域策略文件获得网站的许可。
- 在只能与本地文件系统内容交互的沙箱中的影片剪辑不能对只能与远程内容交互的沙箱中的影片剪辑使用脚本（反之也是禁止的）。

对于 Flash Player 7 及更高版本:

- 网站可以允许通过跨域策略文件来跨域访问资源。
- 基于 SWF 文件的原始域, 在各 SWF 文件之间使用脚本受到限制。使用 `System.security.allowDomain()` 方法可调整这些限制。

有关更多信息, 请参见以下部分:

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章, “了解安全性”
- Flash Player 8 安全性白皮书
- Flash Player 8 与安全相关的 API 白皮书

可用性: Flash Player 3 ; ActionScript 1.0

## 参数

**url:String** - 要加载的 SWF 文件或 JPEG 文件的绝对或相对 URL。相对路径必须相对于级别 0 处的 SWF 文件。绝对 URL 必须包括协议引用, 例如 `http://` 或 `file:///`。

**target:Object** - 对影片剪辑对象的引用或表示目标影片剪辑路径的字符串。目标影片剪辑将被加载的 SWF 文件或图像所替换。

**method:String** [ 可选 ] - 指定用于发送变量的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量, 则省略此参数。GET 方法将变量附加到 URL 的末尾, 它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量, 它用于发送长字符串的变量。

## 示例

用法 1: 以下示例从同一目录加载 SWF 文件 `circle.swf`, 并替换舞台上已存在的名为 `mySquare` 的影片剪辑:

```
loadMovie("circle.swf", mySquare);  
// equivalent statement (Usage 1): loadMovie("circle.swf",  
    _level0.mySquare);  
// equivalent statement (Usage 2): loadMovie("circle.swf", "mySquare");
```

下面的示例从同一目录加载 SWF 文件 `circle.swf`, 但替换主影片剪辑而不是 `mySquare` 影片剪辑:

```
loadMovie("circle.swf", this);  
// Note that using "this" as a string for the target parameter will not work  
// equivalent statement (Usage 2): loadMovie("circle.swf", "_level0");
```

下面的 `loadMovie()` 语句将 SWF 文件 `sub.swf` 从同一目录加载到使用 `createEmptyMovieClip()` 创建的名为 `logo_mc` 的新影片剪辑中:

```
this.createEmptyMovieClip("logo_mc", 999);  
loadMovie("sub.swf", logo_mc);
```

您可以添加以下代码，以从加载 `sub.swf` 的 SWF 文件所在的同一目录加载名为 `image1.jpg` 的 JPEG 图像。当您单击名为 `myBtn_btn` 的按钮时，将加载该 JPEG。此代码将 JPEG 加载到 `logo_mc` 中。因此，它将 `sub.swf` 替换为 JPEG 图像。

```
myBtn_btn.onRelease = function(){
    loadMovie("image1.jpg", logo_mc);
};
```

用法 2：以下示例从同一目录加载 SWF 文件 `circle.swf`，并替换舞台上已存在的名为 `mySquare` 的影片剪辑：

```
loadMovie("circle.swf", "mySquare");
```

另请参见

[\\_level 属性](#)，[loadMovieNum 函数](#)，[loadMovie \(MovieClip.loadMovie 方法\)](#)，[loadClip \(MovieClipLoader.loadClip 方法\)](#)，[unloadMovie 函数](#)

## loadMovieNum 函数

```
loadMovieNum(url:String, level:Number, [method:String]) : Void
```

在播放原始 SWF 文件时，将 SWF、JPEG、GIF 或 PNG 文件加载到一个级别中。在 Flash Player 8 中添加了对非动画 GIF 文件、PNG 文件和渐进式 JPEG 文件的支持。如果加载动画 GIF，则仅显示第一帧。



如果您要监视下载的进度，则使用 `MovieClipLoader.loadClip()` 而不是此函数。

一般情况下，Flash Player 显示单个 SWF 文件，然后关闭。`loadMovieNum()` 动作使您可以一次显示多个 SWF 文件，并且无需加载另一个 HTML 文档即可在 SWF 文件之间进行切换。

如果要指定目标而不是级别，请使用 `loadMovie()` 而不是 `loadMovieNum()`。

Flash Player 具有从级别 0 开始的级别堆叠顺序。这些级别类似于醋酸纤维层；除了每个级别上的对象之外，它们是透明的。当使用 `loadMovieNum()` 时，必须指定 SWF 文件将加载到 Flash Player 中的哪个级别。在 SWF 文件加载到某个级别后，即可使用语法 `_level N` 定位该 SWF 文件，其中 *N* 为级别号。

当加载 SWF 文件时，可指定任何级别号，并且可将 SWF 文件加载到已加载有 SWF 文件的级别。如果执行此动作，则新的 SWF 文件将替换现有的 SWF 文件。如果将 SWF 文件加载到级别 0，则 Flash Player 中的每个级别均被卸载，并且级别 0 将替换为该新文件。处于级别 0 的 SWF 文件为其它所有加载的 SWF 文件设置帧频、背景色和帧大小。

`loadMovieNum()` 动作也允许您在播放 SWF 文件时将 JPEG 文件加载到该 SWF 文件中。对于图像和 SWF 文件，在加载文件时，图像的左上角均与舞台的左上角对齐。另外，在这两种情况下，加载的文件均继承旋转和缩放设置，并且原始内容将在指定级别中被覆盖。



不支持以渐进格式保存的 JPEG 文件。

使用 `unloadMovieNum()` 可删除用 `loadMovieNum()` 加载的 SWF 文件或图像。

使用此方法时，请考虑 **Flash Player** 安全模型。

对于 **Flash Player 8**：

- 如果执行调用的影片剪辑在只能与本地文件系统的内容交互的沙箱中，并且被加载的影片剪辑来自网络沙箱，则不允许加载。
- 如果执行调用的 SWF 文件在网络沙箱中并且要加载的影片剪辑是本地的，则不允许加载。
- 从受信任的本地沙箱或从只能与远程内容交互的沙箱访问网络沙箱需要通过跨域策略文件获得网站的许可。
- 在只能与本地文件系统的内容交互的沙箱中的影片剪辑不能对只能与远程内容交互的沙箱中的影片剪辑使用脚本（反之也是禁止的）。

对于 **Flash Player 7** 及更高版本：

- 网站可以允许通过跨域策略文件来跨域访问资源。
- 基于 SWF 文件的原始域，在各 SWF 文件之间使用脚本受到限制。使用 `System.security.allowDomain()` 方法可调整这些限制。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书
- **Flash Player 8** 与安全相关的 API 白皮书

可用性：**Flash Player 4**；**ActionScript 1.0**

## 参数

`url:String` - 要加载的 SWF 文件或 JPEG 文件的绝对或相对 URL。相对路径必须相对于级别 0 处的 SWF 文件。为了在独立的 **Flash Player** 中使用 SWF 文件或在 **Flash** 创作应用程序的测试模式下测试 SWF 文件，必须将所有的 SWF 文件存储在同一个文件夹中，并且其文件名不能包含文件夹或磁盘驱动器的规格。

`level:Number` - 一个整数，指定 SWF 文件将加载到 **Flash Player** 中的哪个级别。



`method:String` [ 可选 ] - 指定用于发送变量的 **HTTP** 方法。该参数必须是字符串 `GET` 或 `POST`。如果没有要发送的变量，则省略此参数。`GET` 方法将变量附加到 **URL** 的末尾，它用于发送少量的变量。`POST` 方法在单独的 **HTTP** 标头中发送变量，它用于发送长字符串的变量。

## 示例

下面的示例将 **JPEG** 图像 `tim.jpg` 加载到 **Flash Player** 的级别 2 中：

```
loadMovieNum("http://www.helpexamples.com/flash/images/image1.jpg", 2);
```

另请参见

[unloadMovieNum 函数](#)，[loadMovie 函数](#)，[loadClip \(MovieClipLoader.loadClip 方法\)](#)，[\\_level 属性](#)

## loadVariables 函数

```
loadVariables(url:String, target:Object, [method:String]) : Void
```

从外部文件（例如文本文件，或由 **ColdFusion**、**CGI** 脚本、**Active Server Page (ASP)**、**PHP** 或 **Perl** 脚本生成的文本）中读取数据，并设置目标影片剪辑中变量的值。此动作还可用于使用新值更新活动 **SWF** 文件中的变量。

指定 **URL** 处的文本必须是标准的 **MIME** 格式 `application/x-www-form-urlencoded`（**CGI** 脚本所使用的一种标准格式）。可以指定任意数量的变量。例如，下面的代码定义了多个变量：

```
company=Macromedia&address=600+Townsend&city=San+Francisco&zip=94103
```

如果 **SWF** 文件在比 **Flash Player 7** 更低的版本中运行，则 `url` 必须与发布此调用的 **SWF** 文件位于同一个超级域中。超级域可以通过滤除某一文件的 **URL** 最左侧的部分而得到。例如，位于 `www.someDomain.com` 的 **SWF** 文件可以从位于 `store.someDomain.com` 的源中加载数据，这是因为这两个文件都在同一个名为 `someDomain.com` 的超级域中。

如果任何版本的 **SWF** 文件在 **Flash Player 7** 或更高版本中运行，则 `url` 必须与发布此调用的 **SWF** 文件位于完全相同的域中（请参见“在 **Flash** 中使用 **ActionScript**”中的“**Flash Player** 安全功能”）。例如，位于 `www.someDomain.com` 的 **SWF** 文件只能从同样位于 `www.someDomain.com` 的源中加载数据。如果要从其它域中加载数据，则可以在承载要被访问的 **SWF** 文件的服务器上放置一个**跨域策略文件**。有关更多信息，请参见“在 **Flash** 中使用 **ActionScript**”中的“关于允许跨域数据加载”。

如果要将变量加载到特定的级别中，请使用 `loadVariablesNum()` 而不是 `loadVariables()`。

可用性：**Flash Player 4**；**ActionScript 1.0**

## 参数

**url:String** - 变量所处位置的绝对或相对 URL。如果发出此调用的 SWF 文件正在 Web 浏览器上运行, 则 *url* 必须与 SWF 文件位于同一个域中; 有关详细信息, 请参见“说明”部分。

**target:Object** - 指向接收所加载变量的影片剪辑的目标路径。

**method:String [ 可选 ]** - 指定用于发送变量的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量, 则省略此参数。GET 方法将变量附加到 URL 的末尾, 它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量, 它用于发送长字符串的变量。

## 示例

下面的示例将名为 **params.txt** 的文本文件中的信息加载到使用 `createEmptyMovieClip()` 创建的影片剪辑 `target_mc` 中。`setInterval()` 函数用于检查加载进度。该脚本检查 **params.txt** 文件中是否存在名为 `done` 的变量。

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
loadVariables("params.txt", target_mc);
function checkParamsLoaded() {
    if (target_mc.done == undefined) {
        trace("not yet.");
    } else {
        trace("finished loading. killing interval.");
        trace("-----");
        for (i in target_mc) {
            trace(i+": "+target_mc[i]);
        }
        trace("-----");
        clearInterval(param_interval);
    }
}
var param_interval:Number = setInterval(checkParamsLoaded, 100);
```

外部文件 `params.txt` 包括以下文本:

```
var1="hello"&var2="goodbye"&done="done"
```

## 另请参见

[loadVariablesNum 函数](#), [loadMovie 函数](#), [loadMovieNum 函数](#), [getURL 函数](#), [loadMovie \(MovieClip.loadMovie 方法\)](#), [loadVariables \(MovieClip.loadVariables 方法\)](#), [load \(LoadVars.load 方法\)](#)

## loadVariablesNum 函数

`loadVariablesNum(url:String, level:Number, [method:String]) : Void`

从外部文件（例如文本文件，或由 ColdFusion、CGI 脚本、ASP、PHP 或 Perl 脚本生成的文本）中读取数据，并设置 Flash Player 的某个级别中的变量的值。此函数还可用于使用新值更新活动 SWF 文件中的变量。

指定的 URL 处的文本必须为标准的 MIME 格式 *application/x-www-form-urlencoded*（CGI 脚本所使用的一种标准格式）。可以指定任意数量的变量。例如，下面的代码定义了多个变量：

```
company=Macromedia&address=601+Townsend&city=San+Francisco&zip=94103
```

如果 SWF 文件在比 Flash Player 7 更低的版本中运行，则 url 必须与发布此调用的 SWF 文件位于同一个超级域中。超级域可以通过滤除某一文件的 URL 最左侧的部分而得到。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的源中加载数据，这是因为这两个文件都在同一个名为 `someDomain.com` 的超级域中。

如果任何版本的 SWF 文件在 Flash Player 7 或更高版本中运行，则 url 必须与发布此调用的 SWF 文件位于完全相同的域中（请参见“在 Flash 中使用 ActionScript”中的“Flash Player 安全功能”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的源中加载数据。如果要从其它域中加载数据，则可以在承载该 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见“在 Flash 中使用 ActionScript”中的“关于允许跨域数据加载”。

如果要将变量加载到目标影片剪辑中，请使用 `loadVariables()` 而不是 `loadVariablesNum()`。

可用性：Flash Player 4；ActionScript 1.0

### 参数

`url:String` - 变量所处位置的绝对或相对 URL。如果发出此调用的 SWF 文件正在 Web 浏览器上运行，则 url 必须与 SWF 文件位于同一个域中；有关详细信息，请参见“说明”部分。

`level:Number` - 一个整数，指定 Flash Player 中接收这些变量的级别。

`method:String` [ 可选 ] - 指定用于发送变量的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量附加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送长字符串的变量。

## 示例

下面的示例将名为 `params.txt` 的文本文件中的信息加载到 **Flash Player** 中级别 2 处的 **SWF** 的主时间轴中。文本字段的变量名必须与 `params.txt` 文件中的变量名匹配。

`setInterval()` 函数用于检查向 **SWF** 中加载数据的进度。该脚本检查 `params.txt` 文件中是否存在名为 `done` 的变量。

```
loadVariablesNum("params.txt", 2);
function checkParamsLoaded() {
    if (_level2.done == undefined) {
        trace("not yet.");
    } else {
        trace("finished loading. killing interval.");
        trace("-----");
        for (i in _level2) {
            trace(i+": "+_level2[i]);
        }
        trace("-----");
        clearInterval(param_interval);
    }
}
var param_interval:Number = setInterval(checkParamsLoaded, 100);

// Params.txt includes the following text
var1="hello"&var2="goodbye"&done="done"
```

## 另请参见

[getUrl 函数](#), [loadMovie 函数](#), [loadMovieNum 函数](#), [loadVariables 函数](#), [loadMovie \(MovieClip.loadMovie 方法\)](#), [loadVariables \(MovieClip.loadVariables 方法\)](#), [load \(LoadVars.load 方法\)](#)

## mbchr 函数

`mbchr(number:Number)`

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用 `String.fromCharCode()` 方法。

将 **ASCII** 代码数字转换为多字节字符。

可用性: **Flash Player 4** ; **ActionScript 1.0**

## 参数

`number:Number` - 要转换为多字节字符的数字。

## 另请参见

[fromCharCode \(String.fromCharCode 方法\)](#)

## mblength 函数

`mblength(string:String) : Number`

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用 **String** 类的方法和属性。  
返回多字节字符串的长度。

可用性：Flash Player 4； ActionScript 1.0

### 参数

`string:String` - 要度量的字符串。

### 返回

Number - 多字节字符串的长度。

### 另请参见

[String.length](#) ([String.length](#) 属性)

## mbord 函数

`mbord(character:String) : Number`

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用 `String.charCodeAt()`。  
将指定字符转换为多字节数字。

可用性：Flash Player 4； ActionScript 1.0

### 参数

`character:String` - *character* 要转换为多字节数字的字符。

### 返回

Number - 已转换的字符。

### 另请参见

[charCodeAt](#) ([String.charCodeAt](#) 方法)

## mbsubstring 函数

mbsubstring(value:String, index:Number, count:Number) : String

自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String.substr()。

从多字节字符串中提取新的多字节字符串。

可用性：Flash Player 4； ActionScript 1.0

### 参数

value:String - 多字节字符串，要从其中提取一个新的多字节字符串。

index:Number - 要提取的第一个字符的编号。

count:Number - 要在已提取的字符串中包括的字符数，不包括索引字符。

### 返回

String - 从多字节字符串中提取的字符串。

### 另请参见

[substr \(String.substr 方法\)](#)

## MMExecute 函数

MMExecute("Flash JavaScript API command;":String) : String

允许您从 ActionScript 中发出 Flash JavaScript API (JSAPI) 命令。在 Flash MX2004 中，MMExecute 函数只能由用作 Flash 面板的影片（存储在 WindowSWF 目录中的文件）、“XMLtoUI”对话框或某个组件的“自定义用户界面”来调用。JSAPI 命令在播放器中、在测试影片模式中或在创作环境之外不起作用。

Flash JSAPI 提供若干对象、方法和属性，以直接复制或模拟用户可以在创作环境中输入的命令。使用 JSAPI，您可以编写通过以下若干方式扩展 Flash 的脚本：向菜单添加命令、处理舞台上的对象和重复命令序列等。

通常，用户通过选择“命令”>“运行命令”运行 JSAPI 脚本。不过，您可以在 ActionScript 中使用此函数以直接调用 JSAPI 命令。如果您在文件的第 1 帧上的脚本中使用 MMExecute()，则该命令在加载 SWF 文件时执行。

有关 JSAPI 的更多信息，请访问 [www.macromedia.com/go/jsapi\\_info\\_en](http://www.macromedia.com/go/jsapi_info_en)。

可用性：Flash Player 7； ActionScript 1.0

## 参数

`command:String` - 可在 **Flash JavaScript (JSFL)** 文件中使用的任何命令。

## 返回

`String` - 由 **JavaScript** 语句发送的结果（如果有）的字符串表示形式。

## 示例

下面的命令将当前文档库中的项数输出到跟踪窗口中。您必须将此示例作为 **Flash** 面板运行，原因是：如果在测试影片或浏览器中运行 **Flash** 文件，**Flash** 文件将无法调用 `MMExecute`。

- 将以下代码置于空 **Flash** 文档的主时间轴的第 1 帧中：

```
var numLibItems = MMExecute("fl.getDocumentDOM().library.items.length");

var message = numLibItems + " items in library";

MMExecute('fl.trace("'" + message + '"');');
```

- 将 **FLA** 文件保存在 **Configuration** 目录中的 **WindowSWF** 目录中，然后选择“文件” > “发布”（或者将它保存在他处，然后将 **SWF** 文件直接发布到该目录或将 **SWF** 文件移至该目录）。
- 退出并重新启动应用程序（第一次将文件添加到 **WindowSWF** 目录中时，需要执行这一步）。

现在，可以从“窗口” > “其它面板”菜单的底部选择您的文件。

**ActionScript trace** 函数在 **Flash** 面板中不起作用；此示例使用 **JavaScript** `fl.trace` 版本获取输出。将 `MMExecute` 的结果复制到 **Flash** 面板文件包括的文本字段中可能会更加容易。

## nextFrame 函数

`nextFrame()` : `Void`

将播放头转到下一帧。

可用性：Flash Player 2；ActionScript 1.0

## 示例

在下面的示例中，当用户按向右或向下箭头键时，播放头会转到下一帧并停止。如果用户按向左或向上箭头键，播放头会转到上一帧并停止。侦听器会进行初始化以等待箭头键被按下，`init` 变量用于防止在播放头返回到第 1 帧时重新定义侦听器。

```
stop();
```

```

if (init == undefined) {
    someListener = new Object();
    someListener.onKeyDown = function() {
        if (Key.isDown(Key.LEFT) || Key.isDown(Key.UP)) {
            _level0.prevFrame();
        } else if (Key.isDown(Key.RIGHT) || Key.isDown(Key.DOWN)) {
            _level0.nextFrame();
        }
    };
    Key.addListener(someListener);
    init = 1;
}

```

另请参见

[prevFrame 函数](#)

## nextScene 函数

nextScene() : Void

将播放头转到下一场景的第 1 帧。

可用性：Flash Player 2；ActionScript 1.0

### 示例

在下面的示例中，当用户单击在运行时创建的按钮时，播放头会转到下一场景的第 1 帧。创建两个场景，并在第 1 个场景的第 1 帧上输入以下 **ActionScript**。

```

stop();

if (init == undefined) {
    this.createEmptyMovieClip("nextscene_mc", this.getNextHighestDepth());
    nextscene_mc.createTextField("nextscene_txt", this.getNextHighestDepth(),
        200, 0, 100, 22);
    nextscene_mc.nextscene_txt.autoSize = true;
    nextscene_mc.nextscene_txt.border = true;
    nextscene_mc.nextscene_txt.text = "Next Scene";
    this.createEmptyMovieClip("prevscene_mc", this.getNextHighestDepth());
    prevscene_mc.createTextField("prevscene_txt", this.getNextHighestDepth(),
        00, 0, 100, 22);
    prevscene_mc.prevscene_txt.autoSize = true;
    prevscene_mc.prevscene_txt.border = true;
    prevscene_mc.prevscene_txt.text = "Prev Scene";
    nextscene_mc.onRelease = function() {
        nextScene();
    };
}

```



```

prevscene_mc.onRelease = function() {
    prevScene();
};

init = true;
}

```

确保将 `stop()` 动作置于第 2 个场景的第 1 帧上。

另请参见

[prevScene 函数](#)

## Number 函数

`Number(expression) : Number`

将参数 *expression* 转换为数字，并返回下面列表中说明的值：

- 如果 *expression* 是数字，则返回值为 *expression*。
- 如果 *expression* 是布尔值，则当 *expression* 是 `true` 时，返回值为 `1`；当 *expression* 是 `false` 时，返回值为 `0`。
- 如果 *expression* 为字符串，则该函数尝试将 *expression* 分析为一个带有可选尾随指数的十进制数字（例如 `1.57505e-3`）。
- 如果 *expression* 是 `NaN`，则返回值为 `NaN`。
- 如果 *expression* 是 `undefined`，则返回值如下所示：
  - 在为 **Flash Player 6** 或更低版本发布的文件中，结果为 `0`。
  - 在为 **Flash Player 7** 或更高版本发布的文件中，结果为 `NaN`。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**expression: Object** - 要转换为数字的表达式。以 **0x** 开头的数字或字符串被解释为十六进制值。以 **0** 开头的数字或字符串被解释为八进制值。

### 返回

`Number` - 一个数字或 `NaN`（非数字）。

### 示例

在下面的示例中，将在运行时在舞台上创建一个文本字段：

```

this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100,
    22);
counter_txt.autoSize = true;

```

```
counter_txt.text = 0;
function incrementInterval():Void {
    var counter:Number = counter_txt.text;
    // Without the Number() function, Flash would concatenate the value instead
    // of adding values. You could also use "counter_txt.text++;"
    counter_txt.text = Number(counter) + 1;
}
var intervalID:Number = setInterval(incrementInterval, 1000);
```

另请参见

[NaN 常数](#), [Number](#), [parseInt 函数](#), [parseFloat 函数](#)

## Object 函数

`Object([value:Object])` : `Object`

创建一个新的空对象，或者将指定的数字、字符串或布尔值转换为一个对象。此命令等效于使用 `Object` 构造函数创建一个对象（请参见“`Object` 类的构造函数”）。

可用性: Flash Player 5 ; ActionScript 1.0

### 参数

`value:Object` [ 可选 ] - 一个数字、字符串或布尔值。

### 返回

`Object` - 一个对象。

### 示例

在下面的示例中，将创建一个新的空对象，然后使用值填充该对象：

```
var company:Object = new Object();
company.name = "Macromedia, Inc.";
company.address = "600 Townsend Street";
company.city = "San Francisco";
company.state = "CA";
company.postal = "94103";
for (var i in company) {
    trace("company."+i+" = "+company[i]);
}
```

另请参见

[Object](#)

## on 处理函数

```
on(MouseEvent:Object) {  
    // your statements here  
}
```

指定触发动作的鼠标事件或按键。

可用性：Flash Player 2；ActionScript 1.0

### 参数

`MouseEvent:Object` - `MouseEvent` 是一个称为事件的触发器。当事件发生时，执行该事件后面大括号 ({ }) 中的语句。可以为 `MouseEvent` 参数指定下面的任一值：

- `press` 当鼠标指针滑到按钮上时按下鼠标按钮。
- `release` 当鼠标指针滑到按钮上时释放鼠标按钮。
- `releaseOutside` 当鼠标指针滑到按钮上时按下鼠标按钮，然后在释放鼠标按钮前滑出此按钮区域。`press` 和 `dragOut` 事件始终在 `releaseOutside` 事件之前发生。
- `rollOut` 鼠标指针滑出按钮区域。
- `rollOver` 鼠标指针滑到按钮上。
- `dragOut` 当鼠标指针滑到按钮上时按下鼠标按钮，然后滑出此按钮区域。
- `dragOver` 当鼠标指针滑到按钮上时按下鼠标按钮，然后滑出该按钮区域，接着滑回到该按钮上。
- `keyPress "< key >"` 按下指定的键盘键。对于该参数的 `key` 部分，请指定一个键常数，如“动作面板”中的代码提示所示。可以使用这个参数来截取某个按键，也就是说，覆盖所指定键的任何内置行为。该按钮可以在您的应用程序中的任何地方，可以在舞台上或不在舞台上。此技术的一个局限是不能在运行时应用 `on()` 处理函数；您必须在创作时应用它。请确保选择“控制”>“禁用快捷键”，否则在使用“控制”>“测试影片”测试应用程序时某些具有内置行为的键不会被覆盖。

若要查看键常数列表，请参见 **Key** 类。

### 示例

在下面的脚本中，当按鼠标时，将执行 `startDrag()` 函数，当释放鼠标并放下该对象时，将执行条件脚本：

```
on (press) {  
    startDrag(this);  
}  
on (release) {  
    trace("X:"+this._x);  
    trace("Y:"+this._y);  
    stopDrag();  
}
```

另请参见

[onClipEvent 处理函数](#)，[Key](#)

## onClipEvent 处理函数

```
onClipEvent(movieEvent:Object) {  
    // your statements here  
}
```

触发为特定影片剪辑实例定义的动作。

可用性：Flash Player 5；ActionScript 1.0

### 参数

**movieEvent:Object** - *movieEvent* 是一个称为事件的触发器。当事件发生时，执行该事件后面大括号 ({} ) 中的语句。可以为 *movieEvent* 参数指定下面的任一值：

- **load** 影片剪辑一旦被实例化并出现在时间轴中，即启动此动作。
- **unload** 在从时间轴中删除影片剪辑之后，此动作即在第 1 帧中启动。在将任何动作附加到受影响的帧之前处理与 **unload** 影片剪辑事件关联的动作。
- **enterFrame** 以影片剪辑的帧频连续触发该动作。在将任何帧动作附加到受影响的帧之前处理与 **enterFrame** 剪辑事件关联的动作。
- **mouseMove** 每次移动鼠标时启动此动作。使用 **\_xmouse** 和 **\_ymouse** 属性来确定鼠标的当前位置。
- **mouseDown** 当按下鼠标左键时启动此动作。
- **mouseUp** 当释放鼠标左键时启动此动作。
- **keyDown** 当按下某个键时启动此动作。使用 **Key.getCode()** 检索有关最后按下的键的信息。
- **keyUp** 当释放某个键时启动此动作。使用 **Key.getCode()** 方法检索有关最后按下的键的信息。
- **data** 在 **loadVariables()** 或 **loadMovie()** 动作中接收到数据时启动该动作。当与 **loadVariables()** 动作一起指定时，**data** 事件只在加载最后一个变量时发生一次。当与 **loadMovie()** 动作一起指定时，则在检索数据的每一部分时，**data** 事件都重复发生。

### 示例

下面的示例将 **onClipEvent()** 与 **keyDown** 影片事件一起使用，旨在附加到影片剪辑或按钮。**keyDown** 影片事件通常与 **Key** 对象的一个或多个方法和属性一起使用。下面的脚本使用 **Key.getCode()** 找出用户按下了哪个键；如果按下的键与 **Key.RIGHT** 属性相匹配，则播放头会转到下一帧；如果按下的键与 **Key.LEFT** 属性相匹配，则播放头会转到上一帧。

```
onClipEvent (keyDown) {
```

```

if (Key.getCode() == Key.RIGHT) {
    this._parent.nextFrame();
} else if (Key.getCode() == Key.LEFT) {
    this._parent.prevFrame();
}
}

```

下面的示例将 `onClipEvent()` 与 `load` 和 `mouseMove` 影片剪辑事件一起使用。`xmouse` 和 `ymouse` 属性在鼠标每次移动时跟踪鼠标的位置，鼠标位置显示在运行时创建的文本字段中。

```

onClipEvent (load) {
    this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 100,
        22);
    coords_txt.autoSize = true;
    coords_txt.selectable = false;
}
onClipEvent (mouseMove) {
    coords_txt.text = "X:"+_root._xmouse+",Y:"+_root._ymouse;
}

```

另请参见

[Key](#), [\\_xmouse](#) ([MovieClip.\\_xmouse](#) 属性), [\\_ymouse](#) ([MovieClip.\\_ymouse](#) 属性), [on](#) 处理函数, [updateAfterEvent](#) 函数

## ord 函数

`ord(character:String) : Number`

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用 **String** 类的方法和属性。将字符转换为 ASCII 代码数字。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**character:String** - 要转换为 ASCII 代码数字的字符。

### 返回

**Number** - 指定的字符的 ASCII 代码数字。

另请参见

[String](#), [charCodeAt](#) ([String.charCodeAt](#) 方法)

## parseFloat 函数

`parseFloat(string:String) : Number`

将字符串转换为浮点数。此函数读取或分析并返回字符串中的数字，直到此函数遇到不是初始数字一部分的字符。如果字符串不是以可以分析的数字开头，`parseFloat()` 将返回 NaN。有效整数前面的空白将被忽略，有效整数后面的非数字字符也将被忽略。

可用性：Flash Player 5；ActionScript 1.0

### 参数

`string:String` - 要读取并转换为浮点数的字符串。

### 返回

Number - 一个数字或 NaN（非数字）。

### 示例

下面的示例使用 `parseFloat()` 函数计算各种类型的数字：

```
trace(parseFloat("-2")); // output: -2
trace(parseFloat("2.5")); // output: 2.5
trace(parseFloat(" 2.5")); // output: 2.5
trace(parseFloat("3.5e6")); // output: 3500000
trace(parseFloat("foobar")); // output: NaN
trace(parseFloat("3.75math")); // output: 3.75
trace(parseFloat("0garbage")); // output: 0
```

### 另请参见

[NaN 常数](#)，[parseInt 函数](#)

## parseInt 函数

`parseInt(expression:String, [radix:Number]) : Number`

将字符串转换为整数。如果参数中指定的字符串不能转换为数字，则此函数返回 NaN。以 **0x** 开头的字符串被解释为十六进制数字。以 **0** 开头的整数或指定基数为 **8** 的整数被解释为八进制数字。有效整数前面的空白将被忽略，有效整数后面的非数字字符也将被忽略。

可用性：Flash Player 5；ActionScript 1.0

### 参数

`expression:String` - 要转换为整数的字符串。

`radix:Number [ 可选 ]` - 表示要分析的数字的基数（基）的整数。合法值为 2 到 36。

## 返回

Number - 一个数字或 NaN（非数字）。

## 示例

这一节中的示例使用 `parseInt()` 函数计算各种类型的数字。

以下示例返回 3:

```
parseInt("3.5")
```

以下示例返回 NaN:

```
parseInt("bar")
```

以下示例返回 4:

```
parseInt("4foo")
```

以下示例说明返回 1016 的十六进制转换:

```
parseInt("0x3F8")
```

以下示例说明使用可选的 *radix* 参数且返回 1000 的十六进制转换:

```
parseInt("3E8", 16)
```

以下示例说明二进制转换并返回 10，10 是二进制 1010 的十进制表示形式:

```
parseInt("1010", 2)
```

以下示例说明八进制数字分析并返回 511，511 是八进制 777 的十进制表示形式:

```
parseInt("0777")
```

```
parseInt("777", 8)
```

另请参见

[parseFloat 函数](#)

## play 函数

`play() : Void`

在时间轴中向前移动播放头。

可用性: Flash Player 2 ; ActionScript 1.0

## 示例

在下面的示例中，舞台上有两个影片剪辑实例，名为 `stop_mc` 和 `play_mc`。当单击 `stop_mc` 影片剪辑实例时，**ActionScript** 会停止回放 **SWF** 文件。当单击 `play_mc` 实例时，会继续进行回放。

```
this.stop_mc.onRelease = function() {  
    stop();  
};
```

```
this.play_mc.onRelease = function() {  
    play();  
};  
trace("frame 1");
```

另请参见

[gotoAndPlay 函数](#), [gotoAndPlay \(MovieClip.gotoAndPlay 方法\)](#)

## prevFrame 函数

prevFrame() : Void

将播放头转到前一帧。如果当前帧为第 1 帧，则播放头不移动。

可用性: Flash Player 2 ; ActionScript 1.0

示例

当用户单击名为 myBtn\_btn 的按钮并且将以下 **ActionScript** 置于该按钮的时间轴中的帧上时，播放头转到上一帧：

```
stop();  
this.myBtn_btn.onRelease = function(){  
    prevFrame();  
};
```

另请参见

[nextFrame 函数](#), [prevFrame \(MovieClip.prevFrame 方法\)](#)

## prevScene 函数

prevScene() : Void

将播放头转到前一场景的第 1 帧。

可用性: Flash Player 2 ; ActionScript 1.0

另请参见

[nextScene 函数](#)



## print 函数

`print(target:Object, boundingBox:String) : Void`

根据在参数（`bmovie`、`bmax` 或 `bframe`）中指定的边界打印 `target` 影片剪辑。如果要打印目标影片剪辑中的特定帧，请将 `#p` 帧标签附加到这些帧。尽管 `print()` 所实现的打印品质高于 `printAsBitmap()`，但是它不能用于打印使用 **Alpha** 透明度或特殊色彩效果的影片剪辑。

如果使用 `bmovie` 作为 `boundingBox` 的参数，但未向帧分配 `#b` 标签，则打印区域由加载的影片剪辑的舞台大小来确定。（加载的影片剪辑不继承主影片剪辑的舞台大小。）

影片剪辑中的所有可打印元素必须完全加载后才能开始打印。

**Flash Player** 打印功能支持 **PostScript** 和非 **PostScript** 打印机。非 **PostScript** 打印机将矢量转换成位图。

可用性：**Flash Player 4**；**ActionScript 1.0**

### 参数

**target:Object** - 要打印的影片剪辑的实例名称。默认情况下，打印目标实例中的所有帧。如果要打印影片剪辑中的特定帧，请将 `#p` 帧标签分配给这些帧。

**boundingBox:String** - 一个修饰符，它设置影片剪辑的打印区域。将此参数用引号（" 或 '）括起来，然后指定以下值之一：

- `bmovie` 将影片剪辑中某一特定帧的边框指定为该影片剪辑中所有可打印帧的打印区域。要将其边框用作打印区域的帧分配一个 `#b` 帧标签。
- `bmax` 将所有可打印帧的所有边框的合并区域指定为打印区域。如果影片剪辑中可打印帧的大小各不相同，请指定 `bmax`。
- `bframe` 指示应使用每个可打印帧的边框作为该帧的打印区域，这将为每个帧更改打印区域，并缩放对象以适合打印区域。如果在每个帧中有不同大小的对象，而您希望每个对象都充满打印的页面，请使用 `bframe`。

### 示例

下面的示例打印 `holder_mc` 中所有可打印的帧，而打印区域由每个帧的边框定义：

```
this.createEmptyMovieClip("holder_mc", 999);
holder_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");

this.myBtn_btn.onRelease = function() {
    print(this._parent.holder_mc, "bframe");
};
```

在前面的 **ActionScript** 中，可以将 `bframe` 替换为 `bmovie`，这样，打印区域将由附加了 `#b` 帧标签的帧的边框定义。

另请参见

[printAsBitmap 函数](#), [printAsBitmapNum 函数](#), [PrintJob](#), [printNum 函数](#)

## printAsBitmap 函数

`printAsBitmap(target:Object, boundingBox:String) : Void`

根据在参数（`bmovie`、`bmax` 或 `bframe`）中指定的边界将 `target` 影片剪辑打印为位图。使用 `printAsBitmap()` 打印特定的影片剪辑，这种影片剪辑包含的帧具有使用透明度或色彩效果的对象。`printAsBitmap()` 动作以打印机可用的最高分辨率进行打印，以达到尽可能高的清晰度和品质。

如果影片剪辑不包含 **Alpha** 透明度或色彩效果，**Macromedia** 建议您使用 `print()` 以获得更好的品质效果。

如果使用 `bmovie` 作为 `boundingBox` 的参数，但未向帧分配 `#b` 标签，则打印区域由加载的影片剪辑的舞台大小来确定。（加载的影片剪辑不继承主影片剪辑的舞台大小。）

影片剪辑中的所有可打印元素必须完全加载后才能开始打印。

**Flash Player** 打印功能支持 **PostScript** 和非 **PostScript** 打印机。非 **PostScript** 打印机将矢量转换成位图。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**target:Object** - 要打印的影片剪辑的实例名称。默认情况下，打印影片剪辑中的所有帧。如果要打印影片剪辑中的特定帧，请将 `#p` 帧标签附加到这些帧。

**boundingBox:String** - 一个修饰符，它设置影片剪辑的打印区域。将此参数用引号（" 或 '）括起来，然后指定以下值之一：

- `bmovie` 将影片剪辑中某一特定帧的边框指定为该影片剪辑中所有可打印帧的打印区域。为要将其边框用作打印区域的帧分配一个 `#b` 帧标签。
- `bmax` 将所有可打印帧的所有边框的合并区域指定为打印区域。如果影片剪辑中可打印帧的大小各不相同，请指定 `bmax` 参数。
- `bframe` 指示应使用每个可打印帧的边框作为该帧的打印区域。这将为每个帧更改打印区域，并缩放对象以适合打印区域。如果在每个帧中有不同大小的对象，而您希望每个对象都充满打印的页面，请使用 `bframe`。

## 示例

下面的示例打印 `holder_mc` 中所有可打印的帧，而打印区域由该帧的边框定义：

```
this.createEmptyMovieClip("holder_mc", 999);
holder_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");

this.myBtn_btn.onRelease = function() {
    printAsBitmap(this._parent.holder_mc, "bframe");
};
```

## 另请参见

[print 函数](#)，[printAsBitmapNum 函数](#)，[printNum 函数](#)，[PrintJob](#)

# printAsBitmapNum 函数

`printAsBitmapNum(level:Number, boundingBox:String) : Void`

根据在参数 (`bmovie`、`bmax` 或 `bframe`) 中指定的边界将 **Flash Player** 中的某个级别打印为位图。使用 `printAsBitmapNum()` 打印特定的影片剪辑，这种影片剪辑包含的帧具有使用透明度或色彩效果的对象。`printAsBitmapNum()` 动作以打印机可用的最高分辨率进行打印，以达到尽可能高的清晰度和品质。在指明将帧作为位图打印的情况下，若要计算该帧的可打印文件的大小，可将像素宽度、像素高度和打印机分辨率相乘。

如果影片剪辑不包含 **Alpha** 透明度或色彩效果，使用 `printNum()` 将得到更好的品质效果。

如果使用 `bmovie` 作为 `boundingBox` 的参数，但未向帧分配 `#b` 标签，则打印区域由加载的影片剪辑的舞台大小来确定。（加载的影片剪辑不继承主影片剪辑的舞台大小。）

影片剪辑中的所有可打印元素必须完全加载后才能开始打印。

**Flash Player** 打印功能支持 **PostScript** 和非 **PostScript** 打印机。非 **PostScript** 打印机将矢量转换成位图。

可用性：Flash Player 5；ActionScript 1.0

## 参数

**level:Number** - **Flash Player** 中要打印的级别。默认情况下，打印该级别中的所有帧。如果要打印该级别中的特定帧，请将 `#p` 帧标签分配给这些帧。

**boundingBox:String** - 一个修饰符，它设置影片剪辑的打印区域。将此参数用引号（" 或 '）括起来，然后指定以下值之一：

- `bmovie` 将影片剪辑中某一特定帧的边框指定为该影片剪辑中所有可打印帧的打印区域。要将其边框用作打印区域的帧分配一个 `#b` 帧标签。

- `bmax` 将所有可打印帧的所有边框的合并区域指定为打印区域。如果影片剪辑中可打印帧的大小各不相同，请指定 `bmax` 参数。
- `bframe` 指示应使用每个可打印帧的边框作为该帧的打印区域。这将为每个帧更改打印区域，并缩放对象以适合打印区域。如果在每个帧中有不同大小的对象，而您希望每个对象都充满打印的页面，请使用 `bframe`。

## 示例

下面的示例在用户单击按钮 `myBtn_btn` 时打印舞台上的内容。打印区域由帧的边框定义。

```
myBtn_btn.onRelease = function(){  
    printAsBitmapNum(0, "bframe")  
};
```

## 另请参见

[print 函数](#)、[printAsBitmap 函数](#)、[PrintJob](#)、[printNum 函数](#)

# printNum 函数

`printNum(level:Number, boundingBox:String) : Void`

根据在 `boundingBox` 的参数（`bmovie`、`bmax` 或 `bframe`）中指定的边界打印 **Flash Player** 中的级别。如果要打印目标影片剪辑中的特定帧，请将 `#p` 帧标签附加到这些帧。尽管使用 `printNum()` 所实现的打印品质高于 `printAsBitmapNum()`，但不能使用 `printNum()` 打印具有 **Alpha** 透明度或特殊色彩效果的影片。

如果使用 `bmovie` 作为 `boundingBox` 的参数，但未向帧分配 `#b` 标签，则打印区域由加载的影片剪辑的舞台大小来确定。（加载的影片剪辑不继承主影片剪辑的舞台大小。）

影片剪辑中的所有可打印元素必须完全加载后才能开始打印。

**Flash Player** 打印功能支持 **PostScript** 和非 **PostScript** 打印机。非 **PostScript** 打印机将矢量转换成位图。

可用性：**Flash Player 5**；**ActionScript 1.0**

## 参数

`level:Number` - **Flash Player** 中要打印的级别。默认情况下，打印该级别中的所有帧。如果要打印该级别中的特定帧，请将 `#p` 帧标签分配给这些帧。

`boundingBox:String` - 一个修饰符，它设置影片剪辑的打印区域。将此参数用引号（" 或 '）括起来，然后指定以下值之一：

- `bmovie` 将影片剪辑中某一特定帧的边框指定为该影片剪辑中所有可打印帧的打印区域。为要将其边框用作打印区域的帧分配一个 `#b` 帧标签。

- `bmax` 将所有可打印帧的所有边框的合并区域指定为打印区域。如果影片剪辑中可打印帧的大小各不相同，请指定 `bmax` 参数。
- `bframe` 指示应使用每个可打印帧的边框作为该帧的打印区域。这将为每个帧更改打印区域，并缩放对象以适合打印区域。如果在每个帧中有不同大小的对象，而您希望每个对象都充满打印的页面，请使用 `bframe`。

另请参见

[print 函数](#)，[printAsBitmap 函数](#)，[printAsBitmapNum 函数](#)，[PrintJob](#)

## random 函数

`random(value:Number) : Number`

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用 `Math.random()`。

返回一个随机整数，此整数介于 0 和小于在 *value* 参数中指定的整数之间。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**value:**Number - 一个整数。

### 返回

Number - 一个随机整数。

### 示例

下面对 `random()` 的使用将返回 0、1、2、3 或 4 中的一个值：`random(5)`；

另请参见

[random \(Math.random 方法\)](#)

## removeMovieClip 函数

`removeMovieClip(target:Object)`

删除指定的影片剪辑。

可用性: Flash Player 4 ; ActionScript 1.0

### 参数

**target:Object** - 用 `duplicateMovieClip()` 创建的影片剪辑实例的目标路径，或者是用 `MovieClip.attachMovie()`、`MovieClip.duplicateMovieClip()` 或 `MovieClip.createEmptyMovieClip()` 创建的影片剪辑的实例名称。

### 示例

下面的示例创建名为 `myClip_mc` 的新影片剪辑并直接复制该影片剪辑。第二个影片剪辑名为 `newClip_mc`。图像会加载到这两个影片剪辑中。当单击按钮 `button_mc` 时，会从舞台中删除所直接复制的影片剪辑。

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
myClip_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
duplicateMovieClip(this.myClip_mc, "newClip_mc",
    this.getNextHighestDepth());
newClip_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
newClip_mc._x = 200;
this.button_mc.onRelease = function() {
    removeMovieClip(this._parent.newClip_mc);
};
```

### 另请参见

[duplicateMovieClip 函数](#)，[duplicateMovieClip \(MovieClip.duplicateMovieClip 方法\)](#)，[attachMovie \(MovieClip.attachMovie 方法\)](#)，[removeMovieClip \(MovieClip.removeMovieClip 方法\)](#)，[createEmptyMovieClip \(MovieClip.createEmptyMovieClip 方法\)](#)

## setInterval 函数

```
setInterval(functionReference:Function, interval:Number, [param1:Object, param2,..., paramM]) :  
    Number
```

```
setInterval(objectReference:Object, methodName:String, interval:Number, [param1:Object,  
    param2, ..., paramN]) : Number
```

在播放 SWF 文件时，每隔一定时间就调用函数或对象的方法。您可以在一段时间内使用 setInterval() 重复执行任何函数。

在使用 setInterval() 时注意下列提示：

- 确定被调用的函数的范围。
- 确定设置了间隔 ID（setInterval() 的返回值）的范围。
- 在开始设置新的间隔之前清除以前设置的间隔。

下文将详细讨论这些提示。

**确定被调用的函数的范围。**要确定被调用函数的范围，请将可在其中执行 setInterval() 方法的对象（对象范围）作为第一个参数传递，将要执行的方法名称作为第二个参数传递（如第二个签名中所示）。这可以确保所需的方法从传入的对象引用的范围内执行。以这种方式执行方法时，它可以使用 this 关键字引用对象上的成员变量。

**确定设置了间隔标识符的范围。**要确定设置了间隔标识符 (intervalId) 的范围，您可以将它分配给您传递给 setInterval() 的对象范围上的一个成员变量。这样，被调用的函数就可以在 this.intervalId 找到间隔标识符。

**清除以前设置的间隔。**要在开始设置新的间隔之前清除以前设置的间隔，通常应先调用 clearInterval()，然后调用 setInterval()。这可以确保您不会覆盖或以其它方式破坏 intervalId 变量，该变量是对当前运行的间隔的唯一引用。要在调用 setInterval() 之前调用 clearInterval()，启动脚本和被执行的脚本都必须能够访问 intervalId，如示例中所示。



当需要脚本停止循环时，请始终确保调用 clearInterval()。

可用性：Flash Player 6； ActionScript 1.0

## 参数

**functionReference:**Function - 对要被调用的函数的引用。

**interval:**Number - 对传入的 **functionReference** 或 **methodName** 函数的调用所间隔的时间（以毫秒为单位）。

如果 **interval** 小于 **SWF** 文件的帧频（例如，每秒 10 帧 [fps] 相当于 100 毫秒的间隔），则尽可能按照接近 **interval** 的时间间隔值调用间隔函数。在间隔期间执行大量耗费内存的长脚本将导致延迟。如果被调用的函数启动对可视元素的更改，您应使用 **updateAfterEvent()** 函数来确保屏幕刷新率足够高。如果 **interval** 大于 **SWF** 文件的帧频，则间隔函数仅在 **interval** 已到期并且播放头已进入下一帧时才被调用；这就尽可能减轻了每次刷新屏幕时所产生的影响。

**param:**Object [ 可选 ] - 向发送给 **functionReference** 或 **methodName** 的函数传递的参数。多个参数应该用逗号隔开：*param1* , *param2* , ... , *paramN*

**objectReference:**Object - 一个对象，它包含由 **methodName** 指定的方法。

**methodName:**String - 一个方法，它存在于由 **objectReference** 指定的对象的范围中。

## 返回

Number - 一个整数，它标识间隔（间隔 ID），您可以将其传递给 **clearInterval()** 以取消间隔。

## 示例

**范例 1:** 以下示例以 20 毫秒的间隔跟踪一条消息，直到跟踪达到 10 次，然后清除该间隔。对象范围 **this** 作为第一个参数传入，方法名称 **executeCallback** 作为第二个参数传入。这可以确保 **executeCallback()** 是从调用的脚本的同一范围内执行的。

```
var intervalId:Number;
var count:Number = 0;
var maxCount:Number = 10;
var duration:Number = 20;

function executeCallback():Void {
    trace("executeCallback intervalId: " + intervalId + " count: " + count);
    if(count >= maxCount) {
        clearInterval(intervalId);
    }
    count++;
}

intervalId = setInterval(this, "executeCallback", duration);
```



范例 2：以下示例与第一个示例类似，只是它先调用 `clearInterval()`，然后调用 `setInterval()`。这有助于防止不需要的循环，并且在清除任何特定间隔之前可以多次执行启动脚本的基于事件的系统中尤其重要。

```
var intervalId:Number;
var count:Number = 0;
var maxCount:Number = 10;
var duration:Number = 20;

function executeCallback():Void {
    trace("executeCallback intervalId: " + intervalId + " count: " + count);
    if(count >= maxCount) {
        clearInterval(intervalId);
    }
    count++;
}

function beginInterval():Void {
    if(intervalId != null) {
        trace("clearInterval");
        clearInterval(intervalId);
    }
    intervalId = setInterval(this, "executeCallback", duration);
}

beginInterval();
beginInterval();
beginInterval();
```

范例 3：下面的示例说明如何将自定义参数传递给被调用的函数。

```
var intervalId:Number;
var count:Number = 0;
var maxCount:Number = 10;
var duration:Number = 20;
var colors:Array = new Array("red",
    "blue",
    "yellow",
    "purple",
    "green",
    "orange",
    "salmon",
    "pink",
    "lilac",
    "powder blue",
    "mint");

function executeCallback(param:String) {
```

```

        trace("executeCallback intervalId: " + intervalId + " count: " + count + "
            param: " + param);
        clearInterval(intervalId);
        if(count < maxCount) {
            count++;
            intervalId = setInterval(this, "executeCallback", duration, colors[count]);
        }
    }

    if(intervalId != null) {
        clearInterval(intervalId);
    }

    intervalId = setInterval(this, "executeCallback", duration, colors[count]);

```

范例 4: 下面的示例说明如何正确地从 **ActionScript 2.0** 自定义类使用 `setInterval()`。请注意，与前面的示例类似，`this` 传递给 `setInterval()` 函数以确保被调用的方法在正确的范围内执行。

```

class CustomClass {
    private var intervalId:Number;
    private var count:Number = 0;
    private var maxCount:Number = 10;
    private var duration:Number = 20;

    public function CustomClass():Void {
        beginInterval();
    }

    private function beginInterval():Void {
        if(intervalId != null) {
            trace("clearInterval");
            clearInterval(intervalId);
        }
        intervalId = setInterval(this, "executeCallback", duration);
    }

    public function executeCallback():Void {
        trace("executeCallback intervalId: " + intervalId + " count: " + count);
        if(count >= maxCount) {
            clearInterval(intervalId);
        }
        count++;
    }
}

```

在新文档中，实例化新类的一个新实例：

```
var custom:CustomClass = new CustomClass();
```

另请参见

[clearInterval 函数](#)，[updateAfterEvent 函数](#)，[class 语句](#)

## setProperty 函数

`setProperty(target:Object, property:Object, expression:Object) : Void`

当影片剪辑播放时，更改影片剪辑的属性值。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**target:Object** - 要设置其属性的影片剪辑的实例名称的路径。

**property:Object** - 要设置的属性。

**expression:Object** - 或者是属性的新的字面值，或者是计算结果为属性新值的等式。

### 示例

以下 **ActionScript** 创建一个新的影片剪辑并向其中加载一个图像。通过使用 `setProperty()` 对剪辑设置 `_x` 和 `_y` 坐标。当单击名为 `right_btn` 的按钮时，名为 `params_mc` 的影片剪辑的 `_x` 坐标会递增 20 个像素。

```
this.createEmptyMovieClip("params_mc", 999);
params_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
setProperty(this.params_mc, _y, 20);
setProperty(this.params_mc, _x, 20);
this.right_btn.onRelease = function() {
    setProperty(params_mc, _x, getProperty(params_mc, _x)+20);
};
```

另请参见

[getProperty 函数](#)

## showRedrawRegions 函数

`showRedrawRegions(enable:Boolean, [color:Number]) : Void`

使调试器播放器能够描画出正在重绘的屏幕区域（即正在更新的原有内容的区域）的轮廓。这些轮廓也可以使用“显示重绘区域”菜单选项打开。

可用性：Flash Player 8； ActionScript 1.0

### 参数

**enable:Boolean** - 指定启用 (true) 还是禁用 (false) 重绘区域。当设置为 true 时，将显示重绘矩形。当设置为 false 时，将清除重绘矩形。

**color:Number [ 可选 ]** - 绘制使用的颜色。默认值为红色：0xFF0000。

### 示例

下面的示例对 showRedrawRegions 函数进行了演示。

```
var w:Number = 100;
var h:Number = 100;

var shape1:MovieClip = createShape("shape1");
shape1.onEnterFrame = function():Void {
    this._x += 5;
    this._y += 5;
}

var shape2:MovieClip = createShape("shape2");
shape2.onEnterFrame = function():Void {
    this._y += 5;
}

_global.showRedrawRegions(true);

function createShape(name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    mc.beginFill(0xFFCC00);
    mc.moveTo(200, 200);
    mc.curveTo(300, 200, 300, 100);
    mc.curveTo(300, 0, 200, 0);
    mc.curveTo(100, 0, 100, 100);
    mc.curveTo(100, 200, 200, 200);
    mc.endFill();
    return mc;
}
```

## startDrag 函数

```
startDrag(target:Object, [lock:Boolean, left:Number, top:Number,  
right:Number, bottom:Number]) : Void
```

使 *target* 影片剪辑在影片播放过程中可拖动。一次只能拖动一个影片剪辑。执行 *startDrag()* 操作后，影片剪辑将保持可拖动状态，直到用 *stopDrag()* 显式停止拖动为止，或直到对其它影片剪辑调用了 *startDrag()* 动作为止。

可用性：Flash Player 4； ActionScript 1.0

### 参数

**target:Object** - 要拖动的影片剪辑的目标路径。

**lock:Boolean** [ 可选 ] - 一个布尔值，指定可拖动影片剪辑是锁定到鼠标位置中央 (*true*)，还是锁定到用户首次单击该影片剪辑的位置上 (*false*)。

**left,top,right,bottom:Number** [ 可选 ] - 相对于该影片剪辑的父级的坐标的值，用以指定该影片剪辑的约束矩形。

### 示例

下面的示例在运行时创建影片剪辑 *pic\_mc*，用户可以通过将 *startDrag()* 和 *stopDrag()* 动作附加到该影片剪辑将它拖至任何位置。图像是使用 **MovieClipLoader** 类加载到 *pic\_mc* 中的。

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();  
pic_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",  
  this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth()));  
var listenerObject:Object = new Object();  
listenerObject.onLoadInit = function(target_mc) {  
  target_mc.onPress = function() {  
    startDrag(this);  
  };  
  target_mc.onRelease = function() {  
    stopDrag();  
  };  
};  
pic_mcl.addListener(listenerObject);
```

### 另请参见

[stopDrag 函数](#)， [\\_droptarget](#) ([MovieClip.\\_droptarget](#) 属性)， [startDrag](#) ([MovieClip.startDrag](#) 方法)

## stop 函数

stop() : Void

停止当前正在播放的 SWF 文件。此动作最通常的用法是用按钮控制影片剪辑。

可用性: Flash Player 2 ; ActionScript 1.0

另请参见

[gotoAndStop 函数](#), [gotoAndStop \(MovieClip.gotoAndStop 方法\)](#)

## stopAllSounds 函数

stopAllSounds() : Void

在不停止播放头的情况下停止 SWF 文件中当前正在播放的所有声音。设置到流的声音在播放头移过它们所在的帧时将恢复播放。

可用性: Flash Player 3 ; ActionScript 1.0

### 示例

下面的代码会创建一个文本字段，在该字段中，会显示歌曲的 ID3 信息。将会创建一个新的 **Sound** 对象实例，并且要将您的 MP3 加载到 SWF 文件中。ID3 信息是从该声音文件中提取的。当用户单击 stop\_mc 时，声音会暂停。当用户单击 play\_mc 时，歌曲会从它暂停的位置继续播放。

```
this.createTextField("songinfo_txt", this.getNextHighestDepth, 0, 0,
    Stage.width, 22);
var bg_sound:Sound = new Sound();
bg_sound.loadSound("yourSong.mp3", true);
bg_sound.onID3 = function() {
    songinfo_txt.text = "(" + this.id3.artist + ") " + this.id3.album + " - " +
        this.id3.track + " - "
        + this.id3.songname;
    for (prop in this.id3) {
        trace(prop+" = "+this.id3[prop]);
    }
    trace("ID3 loaded.");
};
this.play_mc.onRelease = function() {
    /* get the current offset. if you stop all sounds and click the play button,
    the MP3 continues from
    where it was stopped, instead of restarting from the beginning. */
    var numSecondsOffset:Number = (bg_sound.position/1000);
    bg_sound.start(numSecondsOffset);
};
```

```
this.stop_mc.onRelease = function() {  
    stopAllSounds();  
};
```

另请参见

[Sound](#)

## stopDrag 函数

stopDrag() : Void

停止当前的拖动操作。

可用性：Flash Player 4； ActionScript 1.0

示例

下面的代码（位于主时间轴中）会在用户释放鼠标按钮时停止对影片剪辑实例 my\_mc 的拖动动作：

```
my_mc.onPress = function () {  
    startDrag(this);  
}  
  
my_mc.onRelease = function() {  
    stopDrag();  
}
```

另请参见

[startDrag 函数](#)，[\\_droptarget](#) ([MovieClip.\\_droptarget 属性](#))，[startDrag](#) ([MovieClip.startDrag 方法](#))，[stopDrag](#) ([MovieClip.stopDrag 方法](#))

## String 函数

String(expression:Object) : String

返回指定参数的字符串表示形式，如下面的列表所述：

- 如果 *expression* 是数字，则返回字符串为该数字的文本表示形式。
- 如果 *expression* 是字符串，则返回字符串为 *expression*。
- 如果 *expression* 是一个对象，则返回值是该对象的字符串表示形式，这是通过调用该对象的字符串属性生成的；如果不存在此类属性，则是通过调用 `Object.toString()` 生成的。
- 如果 *expression* 是布尔值，则返回字符串为 "true" 或 "false"。
- 如果 *expression* 是一个影片剪辑，则返回值是以斜杠 (/) 记号表示的该影片剪辑的目标路径。

如果 *expression* 是 *undefined*，则返回值如下：

- 在为 **Flash Player 6** 和更低版本发布的文件中，结果为空字符串 ("")。
- 在为 **Flash Player 7** 及更高版本发布的文件中，结果为 *undefined*。

注意：ActionScript 2.0 不支持斜杠记号。

可用性：Flash Player 4；ActionScript 1.0

### 参数

*expression:Object* - 要转换为字符串的表达式。

### 返回

*String* - 一个字符串。

### 示例

在下面的示例中，要使用 **ActionScript** 将指定的表达式转换为字符串：

```
var string1:String = String("3");  
var string2:String = String("9");  
trace(string1+string2); // output: 39
```

因为这两个参数都是字符串，所以会将值连接起来而不是加起来。

### 另请参见

[toString \(Number.toString 方法\)](#)，[toString \(Object.toString 方法\)](#)，[String](#)，[" 字符串分隔符运算符"](#)

## substring 函数

*substring(string:String, index:Number, count:Number) : String*

自 **Flash Player 5** 后不推荐使用。不推荐使用此函数，而推荐使用 *String.substr()*。

提取部分字符串。此函数是从 1 开始的，而 **String** 对象方法是从 0 开始的。

可用性：Flash Player 4；ActionScript 1.0

### 参数

*string:String* - 从中提取新字符串的字符串。

*index:Number* - 要提取的第一个字符的编号。

*count:Number* - 要在已提取的字符串中包括的字符数，不包括索引字符。



## 返回

String - 已提取的子字符串。

## 另请参见

[substr \(String.substr 方法\)](#)

# targetPath 函数

targetPath(targetObject:Object) : String

返回一个字符串，其中包含 **MovieClip**、**Button**、**TextField** 或 **Videoobject** 的目标路径。该目标路径以点记号 (.) 形式返回。若要检索以斜杠 (/) 记号表示的目标路径，请使用 `_target` 属性。

可用性：Flash Player 5； ActionScript 1.0

## 参数

**targetObject:Object** - 正在对其检索目标路径的对象的引用（例如，`_root` 或 `_parent`）。这可以是一个 **MovieClip**、**Button** 或 **TextField** 对象。

## 返回

String - 包含指定对象的目标路径的字符串。

## 示例

下面的示例在加载影片剪辑时立即跟踪该影片剪辑的目标路径：

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());  
trace(targetPath(myClip_mc)); // _level0.myClip_mc
```

## 另请参见

[eval 函数](#)

## tellTarget 函数

```
tellTarget(target:String) {  
    statement(s);  
}
```

自 Flash Player 5 后不推荐使用。Macromedia 建议使用点 (.) 记号和 with 语句。

将在 *statements* 参数中指定的指令应用于在 *target* 参数中指定的时间轴。tellTarget 动作对导航控制很有帮助。将 tellTarget 分配给用于停止或开始舞台上其它地方的影片剪辑的按钮。还可以使影片剪辑转到此剪辑的特定帧。例如，可以将 tellTarget 分配给用于停止或开始舞台上影片剪辑的按钮，或者分配给用于提示影片剪辑跳至特定帧的按钮。

在 Flash 5 或更高版本中，可以使用点 (.) 记号代替 tellTarget 动作。可以使用 with 动作向同一个时间轴发出多个动作。使用 with 动作可将任何对象作为目标，而 tellTarget 动作只能将影片剪辑作为目标。

可用性：Flash Player 3；ActionScript 1.0

### 参数

**target:String** - 一个字符串，指定要控制的时间轴的目标路径。

**statement(s)** - 条件为 true 时要执行的指令。

### 示例

此 tellTarget 语句控制主时间轴上的影片剪辑实例 **ball**。ball 实例的第 1 帧为空白而且有一个 stop() 动作，所以它在舞台上不可见。当通过以下动作单击按钮时，tellTarget 会告知 ball 中的播放头转至第 2 帧（动画在这一帧开始）：

```
on(release) {  
    tellTarget("_parent.ball") {  
        gotoAndPlay(2);  
    }  
}
```

下面的示例使用点 (.) 记号达到同样的结果：

```
on(release) {  
    _parent.ball.gotoAndPlay(2);  
}
```

如果需要向 **ball** 实例发出多个命令，可以使用 **with** 动作，如下面的语句所示：

```
on(release) {  
  with(_parent.ball) {  
    gotoAndPlay(2);  
    _alpha = 15;  
    _xscale = 50;  
    _yscale = 50;  
  }  
}
```

另请参见

[with 语句](#)

## toggleHighQuality 函数

toggleHighQuality()

自 **Flash Player 5** 后**不推荐使用**。不推荐使用此函数，而推荐使用 `_quality`。

在 **Flash Player** 中启用和禁用消除锯齿功能。消除锯齿可使对象的边缘平滑并会减缓 **SWF** 播放的速度。此动作会影响 **Flash Player** 中的所有 **SWF** 文件。

可用性：Flash Player 2；ActionScript 1.0

示例

下面的代码可以应用到一个按钮，单击该按钮时会在开启和关闭消除锯齿功能间切换：

```
on(release) {  
  toggleHighQuality();  
}
```

另请参见

[\\_quality 属性](#)

## trace 函数

`trace(expression:Object)`

您可以使用 **Flash** 调试播放器捕获来自 `trace()` 函数的输出并显示结果。

在测试 **SWF** 文件时，使用此语句可在“输出”面板中记录编程注释或显示消息。使用 *expression* 参数可以检查是否存在某种条件，或在“输出”面板中显示值。`trace()` 语句类似于 **JavaScript** 中的 `alert` 函数。

可以使用“发布设置”对话框中的“省略跟踪动作”命令将 `trace()` 动作从导出的 **SWF** 文件中删除。

可用性：Flash Player 4；ActionScript 1.0

### 参数

*expression:Object* - 要计算的表达式。在 **Flash** 创作工具中打开 **SWF** 文件时（使用“测试影片”命令），*expression* 参数的值显示在“输出”面板中。

### 示例

下面的示例使用 `trace()` 语句在“输出”面板中显示动态创建的名为 `error_txt` 的文本字段的方法和属性：

```
this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100,
    22);
for (var i in error_txt) {
    trace("error_txt."+i+" = "+error_txt[i]);
}
/* output:
error_txt.styleSheet = undefined
error_txt.mouseWheelEnabled = true
error_txt.condenseWhite = false
...
error_txt.maxscroll = 1
error_txt.scroll = 1
*/
```

## unescape 函数

unescape(string:String) : String

将参数 *x* 作为字符串计算，将该字符串从 URL 编码格式解码（将所有十六进制序列转换为 ASCII 字符），并返回该字符串。

可用性：Flash Player 5； ActionScript 1.0

### 参数

**string:String** - 要转义的十六进制序列字符串。

### 返回

**String** - 对 URL 编码的参数进行解码所得到的字符串。

### 示例

下面的示例说明转义到反向转义的转换过程：

```
var email:String = "user@somedomain.com";
trace(email);
var escapedEmail:String = escape(email);
trace(escapedEmail);
var unescapedEmail:String = unescape(escapedEmail);
trace(unescapedEmail);
```

下面的结果显示在“输出”面板中。

```
user@somedomain.com
user%40somedomain%2Ecom
user@somedomain.com
```

## unloadMovie 函数

unloadMovie(target:MovieClip) : Void

unloadMovie(target:String) : Void

从 **Flash Player** 中删除通过 loadMovie() 加载的影片剪辑。若要卸载通过 loadMovieNum() 加载的影片剪辑，应使用 unloadMovieNum() 而不是 unloadMovie()。

可用性：Flash Player 3； ActionScript 1.0

### 参数

**target:Object** - 影片剪辑的目标路径。此参数可以是一个字符串（例如 "my\_mc"），也可以是对影片剪辑实例的直接引用（例如 my\_mc）。能够接受一种以上数据类型的参数以 Object 类型列出。

## 示例

下面的示例创建一个名为 `pic_mc` 的新影片剪辑并将一个图像加载到该剪辑中。加载图像时使用的是 **MovieClipLoader** 类。当单击该图像时，影片剪辑会从 **SWF** 文件中卸载：

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
    target_mc.onRelease = function() {
        unloadMovie(pic_mc);
        /* or you could use the following, which refers to the movie clip referenced
           by 'target_mc'. */
        //unloadMovie(this);
    };
};
pic_mcl.addListener(listenerObject);
```

## 另请参见

[loadMovie \(MovieClip.loadMovie 方法\)](#)，[unloadClip \(MovieClipLoader.unloadClip 方法\)](#)

## unloadMovieNum 函数

`unloadMovieNum(level:Number) : Void`

从 **Flash Player** 中删除通过 `loadMovieNum()` 加载的 **SWF** 或图像。若要卸载通过 `MovieClip.loadMovie()` 加载的 **SWF** 或图像，应使用 `unloadMovie()` 而不是 `unloadMovieNum()`。

可用性：Flash Player 3；ActionScript 1.0

## 参数

`level:Number` - 加载的影片的级别 (`_level N`)。

## 示例

下面的示例将一个图像加载到 **SWF** 文件中。当单击 `unload_btn` 时，将删除已加载的内容。

```
loadMovieNum("yourimage.jpg", 1);
unload_btn.onRelease = function() {
    unloadMovieNum(1);
}
```

## 另请参见

[loadMovieNum 函数](#)，[unloadMovie 函数](#)，[loadMovie \(MovieClip.loadMovie 方法\)](#)

## updateAfterEvent 函数

updateAfterEvent() : Void

在 onClipEvent() 处理函数内调用此函数时，或将其作为传递给 setInterval() 的函数或方法的一部分进行调用时，将更新显示（与为影片设置的每秒帧数无关）。如果对 updateAfterEvent 的调用不在 onClipEvent() 处理函数内，也不是传递给 setInterval() 的函数或方法的一部分，则 Flash 将忽略该调用。此函数只对某些 Mouse 和 MovieClip 处理函数起作用：Mouse 类的 mouseDown、mouseUp、mouseMove、keyDown 和 keyUp 处理函数；MovieClip 类的 onMouseMove、onMouseDown、onMouseUp、onKeyDown 和 onKeyUp 处理函数。它对 Key 类不起作用。

可用性：Flash Player 5；ActionScript 1.0

### 示例

下面的示例说明如何创建名为 cursor\_mc 的自定义光标。使用 ActionScript 将鼠标光标替换为 cursor\_mc。然后，使用 updateAfterEvent() 继续刷新舞台以使光标的移动看起来顺畅。

```
Mouse.hide();
cursor_mc.onMouseMove = function() {
    this._x = this._parent._xmouse;
    this._y = this._parent._ymouse;
    updateAfterEvent();
};
```

### 另请参见

[onClipEvent 处理函数](#)，[setInterval 函数](#)

# 全局属性

全局属性在每个脚本中都有，对每个时间轴和文档范围都是可见的。例如，全局属性允许访问其它加载的影片剪辑的时间轴，包括相对的 (`_parent`) 和绝对的 (`_root`)。它们还允许您限制 (`this`) 或扩展 (`super`) 范围。而且，您可以使用全局属性调整运行时设置，如屏幕读取器辅助功能、回放质量和声音缓存大小。

## 全局属性摘要

修饰符	属性	说明
	<code>_accProps</code>	允许您在运行时控制 SWF 文件、影片剪辑、按钮、动态文本字段和输入文本字段的屏幕读取器辅助功能选项。
	<code>_focusrect</code>	属性（全局）；指定当按钮或影片剪辑具有键盘焦点时，是否在其周围显示黄色矩形。
	<code>_global</code>	对包含核心 ActionScript 类的全局对象（例如 String、Object、Math 和 Array）的引用。
	<code>_highquality</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此属性，而推荐使用 <code>_quality</code> 。 指定当前 SWF 文件所应用的消除锯齿的级别。
	<code>_level</code>	对 <code>_level N</code> 的根时间轴的引用。
	<code>maxscroll</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此属性，而推荐使用 <code>TextField.maxscroll</code> 。 指示当文本字段中的底行也可见时该字段中可见文本顶行的行号。
	<code>_parent</code>	指定或返回一个引用，该引用指向包含当前影片剪辑或对象的影片剪辑或对象。
	<code>_quality</code>	设置或检索用于影片剪辑的呈现品质。
	<code>_root</code>	指定或返回一个对根影片剪辑时间轴的引用。
	<code>scroll</code>	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此属性，而推荐使用 <code>TextField.scroll</code> 。 控制文本字段中与变量关联的信息的显示。
	<code>_soundbuftime</code>	确定要缓冲多少秒声音流。
	<code>this</code>	引用对象或影片剪辑实例。



# \_accProps 属性

`_accProps.propertyName`  
`instanceName._accProps.propertyName`

允许您在运行时控制 SWF 文件、影片剪辑、按钮、动态文本字段和输入文本字段的屏幕读取器辅助功能选项。这些属性将覆盖创作过程中在“辅助功能”面板中可用的相应设置。为使对这些属性的更改生效，您必须调用 `Accessibility.updateProperties()`。

有关“辅助功能”面板的信息，请参见使用 Flash 中的“Flash 辅助功能面板”。

若要确定播放器是否运行在支持辅助功能的环境中，请使用 `System.capabilities.hasAccessibility()` 方法。

下表列出了每个 `_accProps` 属性的名称和数据类型、它在“辅助功能”面板中的等效设置以及可以应用该属性的对象类型。术语反转逻辑是指该属性设置与“辅助功能”面板中的相应设置相反。例如，将 `silent` 属性设置为 `true` 等效于取消选择“使影片可访问”或“使对象可访问”选项。

属性	数据类型	“辅助功能”面板中的等效设置	应用于
<code>silent</code>	Boolean	使影片可访问 / 使对象可访问（反逻辑）	整个 SWF 文件影片剪辑 按钮动态文本输入文本
<code>forceSimple</code>	Boolean	使子对象可访问（反逻辑）	整个 SWF 文件影片剪辑
<code>name</code>	String	名称	整个 SWF 文件影片剪辑 按钮输入文本
<code>description</code>	String	说明	整个 SWF 文件影片剪辑 按钮动态文本输入文本
<code>shortcut</code>	String	快捷键	影片剪辑按钮输入文本

对于“快捷键”字段，应使用 `Ctrl+A` 格式的名称。向“辅助功能”面板添加一个键盘快捷键不会创建键盘快捷键；它只会向屏幕读取器告知现有快捷键。有关向可访问对象分配快捷键的信息，请参见 `Key.addListener()`。

若要指定对应于“辅助功能”面板中的“选项卡”索引设置的设置，请使用 `Button.tabIndex`、`MovieClip.tabIndex` 或 `TextField.tabIndex` 属性。

无法在运行时指定“自动标签”设置。

若要引用表示整个 **Flash** 文档的 `_accProps` 对象，请省略 `instanceName` 参数。`_accProps` 的值必须是一个对象。这表示如果不存在 `_accProps` 对象，则您必须创建一个对象，才能为 `_accProps` 对象的属性分配值，如以下示例所示：

```
if ( _accProps == undefined )
{
    _accProps = new Object();
}
_accProps.name = "My SWF file";
```

使用不带 **instanceName** 参数的 `_accProps` 时，对 `_accProps` 属性所做的更改会应用于整个 **SWF** 文件。例如，以下代码将整个 **SWF** 文件的辅助功能 `name` 属性设置为字符串 `"Pet Store"`，然后调用 `Accessibility.updateProperties()` 来促成该更改：

```
_accProps.name = "Pet Store";
Accessibility.updateProperties();
```

相反，以下代码将实例名称为 `price_mc` 的影片剪辑的 `name` 属性设置为字符串 `"Price"`：

```
price_mc._accProps.name = "Price";
Accessibility.updateProperties();
```

如果您要指定若干辅助功能属性，则应在调用 `Accessibility.updateProperties()` 前进行尽可能多的更改，而不是在每个属性语句后调用它，如以下示例所示：

```
_accProps.name = "Pet Store";

animal_mc._accProps.name = "Animal";
animal_mc._accProps.description = "Cat, dog, fish, etc.";

price_mc._accProps.name = "Price";
price_mc._accProps.description = "Cost of a single item";
```

```
Accessibility.updateProperties();
```

如果您没有指定文档或对象的辅助功能属性，则实现在“辅助功能”面板中设置的任意值。在指定一个辅助功能属性后，就不能将它的值还原为在“辅助功能”面板中设置的值。但是，可以通过从 `_accProps` 对象删除属性将该属性设置为其默认值（布尔值为 `false`；字符串值为空字符串），如以下示例所示：

```
my_mc._accProps.silent = true; // set a property
// other code here
delete my_mc._accProps.silent; // revert to default value
```

`_accProps` 的值必须是一个对象。这表示如果不存在 `_accProps` 对象，您必须创建一个对象，才能对 `_accProps` 对象的属性分配线索。

```
if (_accProps == undefined)
{
    _accProps = new Object();
}
_accProps.name = "My movie";
```

可用性: Flash Player 6,0,65,0; ActionScript 1.0

## 参数

**propertyName:** Boolean or String - 辅助功能属性名 (请参见以下有关有效名称的说明)。  
*instanceName*

**instanceName:** String - 分配给影片剪辑、按钮、动态文本字段或输入文本字段的实例的实例名称。若要引用表示整个 **Flash** 文档的 `_accProps` 对象，请省略 **instanceName**。

## 示例

如果更改某个图像并且想要更新它的辅助功能说明，可以使用以下 **ActionScript** 代码：

```
my_mc.gotoAndStop(2);
```

```
if (my_mc._accProps == undefined ) {
    my_mc._accProps = new Object();
}
```

```
my_mc._accProps.name = "Photo of Mount Rushmore";
Accessibility.updateProperties();
```

## 另请参见

[isActive](#) ([Accessibility.isActive](#) 方法), [updateProperties](#) ([Accessibility.updateProperties](#) 方法), [hasAccessibility](#) ([capabilities.hasAccessibility](#) 属性)

## `_focusrect` 属性

```
_focusrect = Boolean;
```

指定当按钮或影片剪辑具有键盘焦点时，是否在其周围显示黄色矩形。如果 `_focusrect` 设置为默认值 `true`，则当用户按 **Tab** 键在 **SWF** 文件中的对象之间导航时，在当前具有焦点的按钮或影片剪辑的周围将显示一个黄色矩形。如果不希望显示黄色矩形，请指定为 `false`。这是一个全局属性，可以被特定实例的设置所覆盖。

如果全局 `_focusrect` 属性设置为 `false`，则所有按钮和影片剪辑的默认行为是只能使用 **Tab** 键进行键盘导航。忽略所有其它键，包括 **Enter** 键和箭头键。要恢复全键盘导航，必须将 `_focusrect` 设为 `true`。要恢复特定按钮或影片剪辑的全键盘功能，可以使用 `Button._focusrect` 或 `MovieClip._focusrect` 覆盖此全局属性。



如果您使用一个组件，则 `FocusManager` 会覆盖 `Flash Player` 的焦点处理，包括此全局属性的使用。

可用性：Flash Player 4；ActionScript 1.0

### 示例

下面的示例说明如何在 **SWF** 文件中的实例在浏览器窗口中有焦点时隐藏其周围的黄色矩形。创建一些按钮或影片剪辑，并在时间轴的第 1 帧中添加以下 **ActionScript**：

```
_focusrect = false;
```

将发布设置更改为 **Flash Player 6**，然后选择“文件”>“发布预览”>“HTML”，在浏览器窗口中测试该 **SWF** 文件。在浏览器窗口中单击 **SWF** 焦点，并使用 **Tab** 键将焦点移至每个实例，即可指定 **SWF** 焦点。当 `_focusrect` 处于禁用状态时，按 **Enter** 键或空格键不会像 `_focusrect` 处于启用状态或者为 `true` 时那样调用 `onRelease` 事件处理函数。

另请参见

[\\_focusrect \(Button.\\_focusrect 属性\)](#)，[\\_focusrect \(MovieClip.\\_focusrect 属性\)](#)

## `_global` 属性

`_global.identifier`

对包含核心 **ActionScript** 类的全局对象（例如 **String**、**Object**、**Math** 和 **Array**）的引用。例如，您可以创建公开为全局 **ActionScript** 对象的库，此库类似于 **Math** 或 **Date** 对象。与时间轴声明或局部声明的变量和函数不一样，全局变量和函数只要未被内部范围中具有相同名称的标识符遮蔽，则它们在 **SWF** 文件中的每个时间轴和范围内均是可访问的。



设置全局变量的值时，必须使用变量的完全限定名称，例如 `_global.variableName`。否则，将创建一个同名的本地变量，容易与您尝试设置的全局变量混淆。

返回对包含核心 **ActionScript** 类的全局对象（例如 **String**、**Object**、**Math** 和 **Array**）的引用。

可用性：Flash Player 6；ActionScript 1.0

### 示例

下面的示例创建一个顶级函数 `factorial()`，该函数对于 **SWF** 文件中的每个时间轴和范围均可用：

```
_global.factorial = function(n:Number) {
    if(n <= 1) {
        return 1;
    }
    else {
        return n * factorial(n - 1);
    }
}
```

```
trace(factorial(1)); // 1
trace(factorial(2)); // 2
trace(factorial(3)); // 6
trace(factorial(4)); // 24
```

下面的示例说明当设置全局变量的值导致意外结果时使用完全限定变量名称的失败情况：

```
_global.myVar = "globalVariable";
trace(_global.myVar); // globalVariable
trace(myVar); // globalVariable

myVar = "localVariable";
trace(_global.myVar); // globalVariable
trace(myVar); // localVariable
```

另请参见

[var 语句](#)，[set variable 语句](#)

## `_highquality` 属性

`_highquality`

自 **Flash Player 5** 后**不推荐使用**。不推荐使用此属性，而推荐使用 `_quality`。

指定当前 **SWF** 文件所应用的消除锯齿的级别。指定 **2**（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 **1**（高品质），则应用消除锯齿功能；如果 **SWF** 文件不包含动画，这将对位图进行平滑处理。指定 **0**（低品质），则不消除锯齿。

可用性：Flash Player 4；ActionScript 1.0

### 示例

以下 **ActionScript** 位于主时间轴上，并且将全局 `quality` 属性设置为总是在非动画文件中应用位图平滑处理。`_highquality = 1`；

另请参见

[\\_quality 属性](#)

## `_level` 属性

`_levelN`

对 `_level N` 的根时间轴的引用。在使用 `_level` 属性定位 **SWF** 文件之前，必须使用 `loadMovieNum()` 将 **SWF** 文件加载到 **Flash Player** 中。还可使用 `_level N` 定位由 *N* 分配的级别的已加载 **SWF** 文件。

加载到 **Flash Player** 实例中的初始 **SWF** 文件会自动加载到 `_level0` 中。`_level0` 中的 **SWF** 文件为所有随后加载的 **SWF** 文件设置帧频、背景色和帧大小。然后，**SWF** 文件堆叠在处于 `_level0` 中的 **SWF** 文件之上的更高编号级别中。

您必须为使用 `loadMovieNum()` 加载到 **Flash Player** 中的每个 **SWF** 文件分配一个级别。您可按任意顺序分配级别。如果您分配的级别（包括 `_level0`）中已经包含 **SWF** 文件，则处于该级别的 **SWF** 文件将被卸载并由新的 **SWF** 文件替换。

可用性：Flash Player 4；ActionScript 1.0

### 示例

下面的示例在加载到 `_level9` 中的 **SWF** 文件 `sub.swf` 的主时间轴中停止播放头。`sub.swf` 文件包含动画，并且与包含以下 **ActionScript** 的文档位于同一目录中：

```
loadMovieNum("sub.swf", 9);
myBtn_btn.onRelease = function() {
    _level9.stop();
};
```

您可以将上一示例中的 `_level9.stop()` 替换为以下代码：

```
_level9.gotoAndStop(5);
```

此动作将 `_level9` 中 **SWF** 文件的主时间轴中的播放头转到第 5 帧，而不是停止播放头。

另请参见

[loadMovie 函数](#)，[swapDepths \(MovieClip.swapDepths 方法\)](#)

## maxscroll 属性

*variable\_name.maxscroll*

自 **Flash Player 5** 后不推荐使用。不推荐使用此属性，而推荐使用 `TextField.maxscroll`。

指示当文本字段中的底行也可见时该字段中可见文本顶行的行号。`maxscroll` 属性与 `scroll` 属性一起使用来控制信息在文本字段中的显示方式。可以检索该属性，但不能修改它。

可用性：Flash Player 4；ActionScript 1.0

另请参见

[maxscroll \(TextField.maxscroll 属性\)](#)，[scroll \(TextField.scroll 属性\)](#)

## \_parent 属性

*\_parent.property*

*\_parent.\_parent.property*

指定或返回一个引用，该引用指向包含当前影片剪辑或对象的影片剪辑或对象。当前对象是包含引用 `_parent` 的 **ActionScript** 代码的对象。使用 `_parent` 来指定一个相对路径，该路径指向当前影片剪辑或对象之上的影片剪辑或对象。

可用性：Flash Player 5；ActionScript 1.0

示例

在下面的示例中，舞台上有一个实例名称为 `square_mc` 的影片剪辑。在该影片剪辑中，有另一个实例名称为 `circle_mc` 的影片剪辑。使用以下 **ActionScript**，可以在单击圆时修改 `circle_mc` 父实例（即 `square_mc`）。当使用相对寻址（使用 `_parent` 而不是 `_root`）时，首先使用“动作”面板中的“插入目标路径”按钮可能会更加容易。

```
this.square_mc.circle_mc.onRelease = function() {  
    this._parent._alpha -= 5;  
};
```

另请参见

[\\_root 属性](#)，[targetPath 函数](#)

# \_quality 属性

`_quality:String`

设置或检索用于影片剪辑的呈现品质。设备字体始终带有锯齿，因此不受 `_quality` 属性的影响。

`_quality` 属性可设置为下表中描述的值。

值	说明	图形消除锯齿	位图平滑处理
"LOW"	低呈现品质。	图形未消除锯齿。	位图未进行平滑处理。
"MEDIUM"	中等呈现品质。此设置适用于不包含文本的影片。	图形使用 2 x 2 像素的网格消除锯齿。	Flash Player 8: 位图基于 <code>MovieClip.attachBitmap()</code> 和 <code>MovieClip.beginBitmapFill()</code> 调用中使用的 <code>smoothing</code> 参数进行平滑处理。 Flash Player 6 和 7: 位图未进行平滑处理。
"HIGH"	高呈现品质。此设置是 Flash 使用的默认呈现品质设置。	图形使用 4 x 4 像素的网格消除锯齿。	Flash Player 8: 位图基于 <code>MovieClip.attachBitmap()</code> 和 <code>MovieClip.beginBitmapFill()</code> 调用中使用的 <code>smoothing</code> 参数进行平滑处理。 Flash Player 6 和 7: 如果影片剪辑为静态的，则对位图进行平滑处理。
"BEST"	极高呈现品质。	图形使用 4 x 4 像素的网格消除锯齿。	Flash Player 8: 位图基于 <code>MovieClip.attachBitmap()</code> 和 <code>MovieClip.beginBitmapFill()</code> 调用中使用的 <code>smoothing</code> 参数进行平滑处理。如果 <code>smoothing</code> 设置为 "Best"，则当使用平均算法按比例缩小影片剪辑时，将以较高的品质呈现结果。例如，这可以减慢呈现速度，但可获得高品质的大图像缩略图。 Flash Player 6 和 7: 位图始终进行平滑处理。

可用性: Flash Player 5 ; ActionScript 1.0



## 示例

以下示例将呈现品质设置为 LOW:

```
_quality = "LOW";
```

## `_root` 属性

```
_root.movieClip  
_root.action  
_root.property
```

指定或返回一个对根影片剪辑时间轴的引用。如果影片剪辑有多个级别，则根影片剪辑时间轴位于包含当前正在执行脚本的级别上。例如，如果级别 1 中的脚本计算 `_root`，则返回 `_level1`。

指定 `_root` 与在当前级别内使用不推荐的斜杠记号 (`/`) 指定绝对路径的效果相同。

**注意:** 如果包含 `_root` 的影片剪辑被加载到另一个影片剪辑中，则 `_root` 指的是加载影片剪辑的时间轴，而不是包含 `_root` 的时间轴。如果要确保 `_root` 指的是被加载的影片剪辑的时间轴（即使该影片被加载到另一个影片剪辑中），请使用 `MovieClip._lockroot`。

可用性: Flash Player 5 ; ActionScript 1.0

## 参数

**movieClip:String** - 影片剪辑的实例名称。

**action:String** - 一个动作或方法。

**property:String** - `MovieClip` 对象的属性。

## 示例

下面的示例停止包含当前正在执行脚本的级别的时间轴:

```
_root.stop();
```

下面的示例在 `_root` 的范围内跟踪变量和实例:

```
for (prop in _root) {  
    trace("_root."+prop+" = "+_root[prop]);  
}
```

## 另请参见

[\\_lockroot](#) (`MovieClip._lockroot` 属性), [\\_parent](#) 属性, [targetPath](#) 函数

## scroll 属性

`textFieldVariableName.scroll = x`

自 **Flash Player 5** 后不推荐使用。不推荐使用此属性，而推荐使用 `TextField.scroll`。

控制文本字段中与变量关联的信息的显示。`scroll` 属性定义文本字段开始显示内容的位置；设置此属性后，当用户滚动该文本字段时，**Flash Player** 将更新此属性。`scroll` 属性可用于将用户定向到长篇文章的特定段落，还可用于创建滚动文本字段。可以检索和修改此属性。

可用性：Flash Player 4；ActionScript 1.0

### 示例

下面是附加到“向上”按钮的代码，该按钮用于滚动名为 `myText` 的文本字段：

```
on (release) {  
    myText.scroll = myText.scroll + 1;  
}
```

### 另请参见

[maxscroll \(TextField.maxscroll 属性\)](#)，[scroll \(TextField.scroll 属性\)](#)

## \_soundbuftime 属性

`_soundbuftime:Number = integer`

确定要缓冲多少秒声音流。默认值为 5 秒。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**integer:**Number - 在 SWF 文件开始进入流之前的秒数。

### 示例

下面的示例先对 MP3 文件进行流式处理并缓冲声音，然后再为用户进行播放。在运行时创建两个文本字段以保存计时器和调试信息。`_soundbuftime` 属性设置为将 MP3 缓冲 10 秒钟。将会为该 MP3 创建一个新的 **Sound** 对象实例。

```
// create text fields to hold debug information.  
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100,  
    22);  
this.createTextField("debug_txt", this.getNextHighestDepth(), 0, 20, 100,  
    22);  
// set the sound buffer to 10 seconds.  
_soundbuftime = 10;
```

```
// create the new sound object instance.
var bg_sound:Sound = new Sound();
// load the MP3 sound file and set streaming to true.
bg_sound.loadSound("yourSound.mp3", true);
// function is triggered when the song finishes loading.
bg_sound.onLoad = function() {
    debug_txt.text = "sound loaded";
};
debug_txt.text = "sound init";
function updateCounter() {
    counter_txt.text++;
}
counter_txt.text = 0;
setInterval(updateCounter, 1000);
```

## this 属性

this

引用对象或影片剪辑实例。执行脚本时，this 引用包含该脚本的影片剪辑实例。在调用方法时，this 包含对包含所调用方法的对象的引用。

在附加到按钮的 on() 事件处理函数中，this 引用包含该按钮的时间轴。在附加到影片剪辑的 onClipEvent() 事件处理函数中，this 引用该影片剪辑自身的时间轴。

因为 this 是在包含它的脚本的上下文中计算的，所以您不能在脚本中使用 this 来引用在类文件中定义的变量。

**可用性：**Flash Player 5；ActionScript 1.0

### 示例

创建名为 **ApplyThis.as** 的 **ActionScript** 文件，然后输入以下代码：

```
class ApplyThis {
    var str:String = "Defined in ApplyThis.as";
    function conctStr(x:String):String {
        return x+x;
    }
    function addStr():String {
        return str;
    }
}
```

接下来，在一个 **FLA** 或单独的 **ActionScript** 文件中，添加以下代码

```
var obj:ApplyThis = new ApplyThis();
var abj:ApplyThis = new ApplyThis();
abj.str = "defined in FLA or AS";
trace(obj.addStr.call(abj, null)); //output: defined in FLA or AS
trace(obj.addStr.call(this, null)); //output: undefined
trace(obj.addStr.call(obj, null)); //output: Defined in applyThis.as
```

同样，若要调用在动态类中定义的函数，您必须使用 `this` 调用适当范围内的函数：

```
// incorrect version of Simple.as
/*
dynamic class Simple {
    function callfunc() {
        trace(func());
    }
}
*/
// correct version of Simple.as
dynamic class simple {
    function callfunc() {
        trace(this.func());
    }
}
```

在 **FLA** 或单独的 **ActionScript** 文件中，添加以下代码：

```
var obj:Simple = new Simple();
obj.num = 0;
obj.func = function() {
    return true;
};
obj.callfunc();
// output: true
```

当您在 `callfunc()` 方法中使用 `this` 时，以上代码生效。不过，如果您使用了不正确的 **Simple.as** 版本，将出现语法错误（在上例中已注释掉）。

在下面的示例中，关键字 `this` 引用 **Circle** 对象：

```
function Circle(radius:Number):Void {
    this.radius = radius;
    this.area = Math.PI*Math.pow(radius, 2);
}
var myCircle = new Circle(4);
trace(myCircle.area);
```

在下面分配给影片剪辑内的帧的语句中，关键字 `this` 引用当前的影片剪辑。

```
// sets the alpha property of the current movie clip to 20
this._alpha = 20;
```

在下面的 **MovieClip.onPress** 处理函数内的语句中，关键字 `this` 引用当前的影片剪辑：

```
this.square_mc.onPress = function() {
    startDrag(this);
};
this.square_mc.onRelease = function() {
    stopDrag();
};
```

另请参见

[on 处理函数](#)，[onClipEvent 处理函数](#)

# 运算符

符号运算符是指定如何组合、比较或修改表达式值的字符。

## 运算符摘要

运算符	说明
+ (addition)	将数值表达式相加或者连接（合并）字符串。
+= (addition assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> + <i>expression2</i> 的值。
[] (array access)	用指定的元素（ <i>a0</i> 等）初始化一个新数组或多维数组，或者访问数组中的元素。
= (assignment)	将 <i>expression2</i> （位于右侧的参数）的值赋给 <i>expression1</i> 中的变量、数组元素或属性。
& (bitwise AND)	将 <i>expression1</i> 和 <i>expression2</i> 转换为 32 位无符号整数，并对整数参数的每一位执行布尔 AND 运算。
&= (bitwise AND assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> & <i>expression2</i> 的值。
<< (bitwise left shift)	将 <i>expression1</i> 和 <i>expression2</i> 转换为 32 位整数，并将 <i>expression1</i> 中的所有位向左移动由 <i>expression2</i> 转换所得到的整数指定的位数。
<<= (bitwise left shift and assignment)	此运算符执行按位向左移位 (<<=) 运算，并将内容作为结果存储在 <i>expression1</i> 中。
~ (bitwise NOT)	也称为对一求补运算符或按位求补运算符。
(bitwise OR)	将 <i>expression1</i> 和 <i>expression2</i> 转换为 32 位无符号整数，并在 <i>expression1</i> 或 <i>expression2</i> 的对应位为 1 的每个位的位置返回 1。
= (bitwise OR assignment)	对 <i>expression1</i> 赋予 <i>expression1</i>   <i>expression2</i> 的值。
>> (bitwise right shift)	将 <i>expression1</i> 和 <i>expression2</i> 转换为 32 位整数，并将 <i>expression1</i> 中的所有位向右移动由 <i>expression2</i> 的转换所得到的整数指定的位数。
>>= (bitwise right shift and assignment)	此运算符执行按位向右移位运算，并将内容作为结果存储在 <i>expression1</i> 中。
>>> (bitwise unsigned right shift)	除了不保留原始 <i>expression</i> 的符号外，此运算符与按位向右移位运算符 (>>) 相同，因为左侧的位始终用 0 填充。通过舍去小数点后面的所有位将浮点数转换为整数。

运算符	说明
>>>= (bitwise unsigned right shift and assignment)	执行无符号按位向右移位运算，并将内容作为结果存储在 <i>expression1</i> 中。
^ (bitwise XOR)	将 <i>expression1</i> 和 <i>expression2</i> 转换为 32 位无符号整数，并在 <i>expression1</i> 或 <i>expression2</i> 中为 1（但不是在两者中均为 1）的对应位的每个位的位置返回 1。
^= (bitwise XOR assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> ^ <i>expression2</i> 的值。
/*..*/ (block comment delimiter)	指示一行或多行脚本注释。
, (comma)	计算 <i>expression1</i> ，然后计算 <i>expression2</i> ，依此类推。
add (concatenation (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。Macromedia 建议在为 Flash Player 5 或更高版本创建内容时使用加运算符 (+)。此运算符在 Flash Player 8 或更高版本中不受支持。 连接两个或更多字符串。
?: (conditional)	指示 Flash 计算 <i>expression1</i> ，如果 <i>expression1</i> 的值为 true，返回 <i>expression2</i> 的值；否则返回 <i>expression3</i> 的值。
-- (decrement)	从 <i>expression</i> 中减 1 的预先递减和滞后递减一元运算符。
/ (division)	<i>expression1</i> 除以 <i>expression2</i> 。
/= (division assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> / <i>expression2</i> 的值。
. (dot)	用于定位影片剪辑层次结构，以便访问嵌套的（子级）影片剪辑、变量或属性。
== (equality)	测试两个表达式是否相等。
eq (equality (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用 == (equality) 运算符。 如果 <i>expression1</i> 的字符串表达式等于 <i>expression2</i> 的字符串表达式，则返回 true，否则返回 false。
> (greater than)	比较两个表达式，确定 <i>expression1</i> 是否大于 <i>expression2</i> ；如果是，则此运算符返回 true。
gt (greater than (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用 >（大于）运算符。 将 <i>expression1</i> 的字符串表达式与 <i>expression2</i> 的字符串表达式相比较，如果 <i>expression1</i> 大于 <i>expression2</i> ，则返回 true，否则返回 false。
>= (greater than or equal to)	比较两个表达式，确定 <i>expression1</i> 是大于或等于 <i>expression2</i> (true) 还是 <i>expression1</i> 小于 <i>expression2</i> (false)。

运算符	说明
ge (greater than or equal to (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用 >=（大于或等于）运算符。 如果 <i>expression1</i> 大于或等于 <i>expression2</i> ，则返回 true，否则返回 false。
++ (increment)	将 <i>expression</i> 加 1 的预先递增和滞后递增一元运算符。
!= (inequality)	测试结果是否与等于运算符 (==) 正好相反。
<> (inequality)	自 Flash Player 5 后 <b>不推荐使用</b> 。此运算符已不推荐使用。Macromedia 建议您使用 != (inequality) 运算符。 测试结果是否与等于运算符 (==) 正好相反。
instanceof	测试 object 是 classConstructor 的实例还是 classConstructor 的子类。
< (less than)	比较两个表达式，确定 <i>expression1</i> 是否小于 <i>expression2</i> ；如果是，则此运算符返回 true。
lt (less than (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用 <（小于）运算符。 如果 <i>expression1</i> 小于 <i>expression2</i> ，则返回 true，否则返回 false。
<= (less than or equal to)	比较两个表达式，确定 <i>expression1</i> 是否小于或等于 <i>expression2</i> ；如果是，则此运算符返回 true。
le (less than or equal to (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。在 Flash 5 中不推荐使用此运算符，而推荐使用 <=（小于或等于）运算符。 如果 <i>expression1</i> 小于或等于 <i>expression2</i> ，则返回 true，否则返回 false。
// (line comment delimiter)	指示脚本注释的开始。
&& (logical AND)	对两个表达式的值执行布尔运算。
and (logical AND)	自 Flash Player 5 后 <b>不推荐使用</b> 。Macromedia 建议使用逻辑 AND 运算符 (&&)。 在 Flash Player 4 中执行逻辑 AND (&&) 运算。
! (logical NOT)	对变量或表达式的布尔值取反。
not (logical NOT)	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用 ! (logical NOT) 运算符。 在 Flash Player 4 中执行逻辑 NOT (!) 运算。
(logical OR)	计算 <i>expression1</i> （运算符左侧的表达式），如果表达式的计算结果为 true，则返回 true。

运算符	说明
or (logical OR)	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用    (logical OR) 运算符。 计算 <i>condition1</i> 和 <i>condition2</i> ，如果任一表达式的计算结果为 true，则整个表达式的计算结果为 true。
% (modulo)	计算 <i>expression1</i> 除以 <i>expression2</i> 的余数。
%= (modulo assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> % <i>expression2</i> 的值。
* (multiplication)	将两个数值表达式相乘。
*= (multiplication assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> * <i>expression2</i> 的值。
new	创建一个新的初始匿名对象，并调用由 constructor 参数标识的函数。
ne (not equal (strings))	自 Flash Player 5 后 <b>不推荐使用</b> 。不推荐使用此运算符，而推荐使用 != (inequality) 运算符。 如果 <i>expression1</i> 不等于 <i>expression2</i> ，则返回 true；否则返回 false。
{ } (object initializer)	创建一个新对象，并用指定的 <i>name</i> 和 <i>value</i> 属性对初始化该对象。
( ) (parentheses)	对一个或多个参数执行分组运算，执行表达式的顺序计算，或者括住一个或多个参数并将它们作为参数传递给括号外的函数。
=== (strict equality)	测试两个表达式是否相等；除了不转换数据类型外，全等运算符 (===) 与等于运算符 (==) 执行运算的方式相同。
!== (strict inequality)	测试结果是否与全等运算符 (===) 正好相反。
" (string delimiter)	如果用在字符之前和之后，则这些引号 (") 表示字符具有字面值；字符将被视为一个字符串，而不是一个变量、数值或其它 ActionScript 元素。
- (subtraction)	用于执行求反或减法运算。
-= (subtraction assignment)	对 <i>expression1</i> 赋予 <i>expression1</i> - <i>expression2</i> 的值。
: (type)	用于严格数据类型指定；此运算符指定变量类型、函数返回类型或函数参数类型。
typeof	typeof 运算符计算 <i>expression</i> 并返回一个字符串，该字符串指定表达式的值为 String, MovieClip, Object, Function, Number, 还是 Boolean 值。
void	void 运算符计算表达式，然后放弃其值，返回 undefined。



## + 加法运算符

*expression1 + expression2*

将数值表达式相加或者连接（合并）字符串。如果其中一个表达式为字符串，则所有其它表达式都被转换为字符串，然后连接起来。两个表达式都为整数时，和为整数；其中一个或两个表达式为浮点数时，和为浮点数。

可用性：Flash Player 4； ActionScript 1.0

### 操作数

*expression1* - 一个数字或字符串。

*expression2* : Number - 一个数字或字符串。

### 返回

Object - 一个字符串、整数或浮点数。

### 示例

用法 1：下面的示例连接两个字符串，然后在“输出”面板中显示结果。

```
var name:String = "Cola";  
var instrument:String = "Drums";  
trace(name + " plays " + instrument); // output: Cola plays Drums
```

用法 2：此语句将整数 2 和 3 相加，然后将计算结果（整数 5）显示在“输出”面板中：

```
trace(2 + 3); // output: 5
```

此语句将浮点数 2.5 和 3.25 相加，然后将计算结果（浮点数 5.75）显示在“输出”面板中

```
trace(2.5 + 3.25); // output: 5.75
```

用法 3：与动态和输入文本字段相关联的变量是字符串数据类型。在以下示例中，变量 `deposit` 是舞台上的一个输入文本字段。在用户输入存款数目后，该脚本尝试将 `deposit` 添加到 `oldBalance`。然而，由于 `deposit` 是字符串数据类型，因此脚本连接（合并成一个字符串）变量的值，而不是对它们求和。

```
var oldBalance:Number = 1345.23;  
var currentBalance = deposit_txt.text + oldBalance;  
trace(currentBalance);
```

例如，如果用户在 `deposit` 文本字段中输入 475，则 `trace()` 语句将值 4751345.23 发送到“输出”面板。若要更正这一点，请使用 `Number()` 函数将字符串转换为数字，如下所示：

```
var oldBalance:Number = 1345.23;  
var currentBalance:Number = Number(deposit_txt.text) + oldBalance;  
trace(currentBalance);
```

下面的示例说明如何不计算字符串表达式右侧的数值和：

```
var a:String = 3 + 10 + "asdf";
trace(a); // 13asdf
var b:String = "asdf" + 3 + 10;
trace(b); // asdf310
```

## **+= 加法赋值运算符**

*expression1 += expression2*

对 *expression1* 赋予 *expression1 + expression2* 的值。例如，下面两个语句的结果是相同的：

```
x += y;
x = x + y;
```

此运算符也可以执行字符串连接运算。加法运算符 (+) 的所有规则适用于加法赋值 (+=) 运算符。

**可用性：**Flash Player 4；ActionScript 1.0

### **操作数**

*expression1* : Number - 一个数字或字符串。

*expression2* : Number - 一个数字或字符串。

### **返回**

Number - 加法运算的结果。

### **示例**

**用法 1：**此示例将 += 运算符与字符串表达式一起使用，并将 “My name is Gilbert” 发送到 “输出” 面板。

```
var x1:String = "My name is ";
x1 += "Gilbert";
trace(x1); // output: My name is Gilbert
```

**用法 2：**下面的示例说明加法赋值运算符 (+=) 的数值用法：

```
var x:Number = 5;
var y:Number = 10;
x += y;
trace(x); // output: 15
```

另请参见

[+ 加法运算符](#)

## [] 数组访问运算符

```
myArray = [ a0, a1,...aN ]  
myArray[ i ] = value  
myObject [ propertyName ]
```

用指定的元素（*a0* 等）初始化一个新数组或多维数组，或者访问数组中的元素。数组访问运算符使您能够动态地设置和检索实例、变量和对象的名称。它还使您能够访问对象属性。

**用法 1:** 数组是一个对象，其属性称为元素，这些元素由称作索引 的数字逐一标识。创建数组时，需用数组访问（`[]`）运算符（即方括号）括住元素。一个数组可以包含各种类型的元素。例如，下面这个名为 `employee` 的数组包含三个元素：第一个元素是一个数字，另外两个元素是字符串（在引号内）：

```
var employee:Array = [15, "Barbara", "Jay"];
```

可以通过嵌套方括号来模拟多维数组。您最多可以嵌套深度为 **256** 级的数组。以下代码创建一个名为 `ticTacToe` 且含有三个元素的数组；其中每个元素也是一个具有三个元素的数组：

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; // Select Debug >  
    List Variables in test mode  
// to see a list of the array elements.
```

**用法 2:** 用方括号（`[]`）括住每个元素的索引可直接对其进行访问；这样可以向数组添加新元素以及更改或检索现有元素的值。数组中第一个索引始终是 **0**，如下示例所示：

```
var my_array:Array = new Array();  
my_array[0] = 15;  
my_array[1] = "Hello";  
my_array[2] = true;
```

可以使用方括号（`[]`）来添加第四个元素，如下示例所示：

```
my_array[3] = "George";
```

可以使用方括号（`[]`）访问多维数组中的元素。第一对括号标识原始数组中的元素，第二对括号标识嵌套数组中的元素。以下几行代码将数字 **6** 发送到“输出”面板。

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];  
trace(ticTacToe[1][2]); // output: 6
```

**用法 3:** 您可以用数组访问运算符（`[]`）代替 `eval()` 函数，以动态地设置并检索影片剪辑名称的值或一个对象的任何属性。以下一行代码将数字 **6** 发送到“输出”面板。

```
name["mc" + i] = "left_corner";
```

**可用性:** Flash Player 4；ActionScript 1.0

## 操作数

`myArray` : Object - *myArray* 一个数组的名称。

`a0, a1, ... aN` : Object - *a0, a1, ... aN* 数组中的元素；任何本机类型或对象实例，包括嵌套的数组。

`i` : Number - *i* 一个大于或等于 0 的整数索引。

`myObject` : Object - *myObject* 对象的名称。

`propertyName` : String - *propertyName* 为对象的属性命名的一个字符串。

## 返回

Object -

用法 1：对一个数组的引用。

用法 2：数组中的一个值；本机类型或对象实例（包括数组实例）。

用法 3：对象的一个属性；本机类型或对象实例（包括数组实例）。

## 示例

下面的示例说明创建新的空 **Array** 对象的两种方法：第一行使用方括号 (`[]`)：

```
var my_array:Array = [];  
var my_array:Array = new Array();
```

下面的示例创建一个名为 `employee_array` 的数组，并使用 `trace()` 语句将元素发送到“输出”面板。在第四行中，更改数组中的一个元素，而第五行将刚修改过的数组发送到“输出”面板：

```
var employee_array = ["Barbara", "George", "Mary"];  
trace(employee_array); // output: Barbara,George,Mary  
employee_array[2] = "Sam";  
trace(employee_array); // output: Barbara,George,Sam
```

在下面的示例中，计算括号中的表达式 (`"piece" + i`)，并将结果用作要从影片剪辑 `my_mc` 中检索的变量的名称。在此示例中，变量 `i` 与按钮必须在同一个时间轴上。例如，如果变量 `i` 等于 5，则将影片剪辑 `my_mc` 中变量 `piece5` 的值显示在“输出”面板中：

```
myBtn_btn.onRelease = function() {  
    x = my_mc["piece"+i];  
    trace(x);  
};
```

在下面的示例中，计算括号中的表达式，并将结果用作要从影片剪辑 `name_mc` 中检索的变量的名称：

```
name_mc["A" + i];
```

如果您熟悉 **Flash 4 ActionScript** 的斜杠语法，则可以使用 `eval()` 函数来实现同样的结果：  
`eval("name_mc.A" & i);`

您可以使用以下 **ActionScript** 循环 `_root` 范围内的所有对象（这对于调试是很有用的）：

```
for (i in _root) {  
    trace(i+": "+_root[i]);  
}
```

您还可以在赋值语句的左侧使用数组访问 (`[]`) 运算符，从而动态地设置实例、变量和对象的名称：

```
employee_array[2] = "Sam";
```

另请参见

[Array](#), [Object](#), [eval 函数](#)

## = 赋值运算符

*expression1* = *expression2*

将 *expression2* (位于右侧的参数) 的值赋给 *expression1* 中的变量、数组元素或属性。可以按值也可以按引用进行赋值。按值进行赋值将复制 *expression1* 的实际值并将其存储在 *expression2* 中。在将数字或文本字符串赋予变量时，使用按值进行赋值。按引用进行赋值将把对 *expression2* 的引用存储在 *expression1* 中。按引用进行赋值通常与 `new` 运算符一起使用。使用 `new` 运算符将在内存中创建一个对象，并将对内存中该位置的引用分配给一个变量。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Object - 一个变量、数组元素或对象的属性。

*expression2* : Object - 一个任何类型的值。

### 返回

Object - 赋予的值，*expression2*。

### 示例

下面的示例使用按值进行赋值，将值 5 赋予变量 `x`。

```
var x:Number = 5;
```

下面的示例使用按值进行赋值，将值 “hello” 赋予变量 `x`：

```
var x:String;  
x = " hello ";
```

下面的示例使用按引用赋值，以创建 `moonsOfJupiter` 变量，该变量包含对新创建的 `Array` 对象的引用。然后使用按值进行赋值，将值 “**Callisto**” 复制到变量 `moonsOfJupiter` 所引用的数组的第一个元素：

```
var moonsOfJupiter:Array = new Array();
moonsOfJupiter[0] = "Callisto";
```

下面的示例使用按引用进行赋值来创建一个新对象，并将对该对象的引用赋予变量 `mercury`。然后使用按值进行赋值，将值 3030 赋予 `mercury` 对象的 `diameter` 属性：

```
var mercury:Object = new Object(); mercury.diameter = 3030; // in miles
trace (mercury.diameter); // output: 3030
```

下面的示例在上一示例的基础上创建名为 `merkur`（**mercury** 在德语中的对应词）的变量，并对它赋予 `mercury` 的值。这样，就创建了引用内存中同一对象的两个变量，也就是说您可以使用任一变量访问该对象的属性。然后可以更改 `diameter` 属性以使用千米而不是英里：

```
var merkur:Object = mercury;
merkur.diameter = 4878; // in kilometers
trace (mercury.diameter); // output: 4878
```

另请参见

[== 等于运算符](#)

## & 按位 AND 运算符

*expression1 & expression2*

将 *expression1* 和 *expression2* 转换为 32 位无符号整数，并对整数参数的每一位执行布尔 **AND** 运算。浮点数通过舍去小数点后面的所有位来转换为整数。结果是一个新的 32 位整数。

正整数转换为无符号的十六进制值，其最大值为 4294967295 或 0xFFFFFFFF；转换大于最大值的数时，会舍去最高有效位，使该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值，其最小值为 -2147483648 或 0x80000000；小于最小值的数转换为精度更高的 2 的补码，同时也会舍去最高有效位。

由于返回值解释为带符号的 2 的补码数，因此，返回值将是 -2147483648 到 2147483647 范围中的一个整数。

可用性：Flash Player 5；ActionScript 1.0

### 操作数

*expression1* : Number - 一个数字。

*expression2* : Number - 一个数字。

## 返回

Number - 按位运算的结果。

## 示例

下面的示例比较数字的位表示形式，仅当相同位置上的位都是 1 时才返回 1。在此 **ActionScript** 中，将 13（二进制的 1101）和 11（二进制的 1011）相加，仅在两个数字都具有 1 的位置中返回 1。

```
var insert:Number = 13;
var update:Number = 11;
trace(insert & update); // output : 9 (or 1001 binary)
```

将数字 13 和 11 相加的结果是 9，因为这两个数字中都只有第一个和最后一个位置具有数字 1。

下面的示例说明返回值转换的行为：

```
trace(0xFFFFFFFF); // 4294967295
trace(0xFFFFFFFF & 0xFFFFFFFF); // -1
trace(0xFFFFFFFF & -1); // -1
trace(4294967295 & -1); // -1
trace(4294967295 & 4294967295); // -1
```

## 另请参见

[&= 按位 AND 赋值运算符](#)，[^ 按位 XOR 运算符](#)，[^= 按位 XOR 赋值运算符](#)，[| 按位 OR 运算符](#)，[|= 按位 OR 赋值运算符](#)，[~ 按位 NOT 运算符](#)

# &= 按位 AND 赋值运算符

*expression1* &= *expression2*

对 *expression1* 赋予 *expression1* & *expression2* 的值。例如，以下两个表达式是等效的：

```
x &= y;
x = x & y;
```

可用性：Flash Player 5；ActionScript 1.0

## 操作数

*expression1* : Number - 一个数字。

*expression2* : Number - 一个数字。

## 返回

Number - *expression1* & *expression2* 的值。

## 示例

下面的示例将值 9 赋予 x:

```
var x:Number = 15;
var y:Number = 9;
trace(x &= y); // output: 9
```

另请参见

& 按位 AND 运算符, ^ 按位 XOR 运算符, ^= 按位 XOR 赋值运算符, | 按位 OR 运算符, |= 按位 OR 赋值运算符, ~ 按位 NOT 运算符

## << 按位向左移位运算符

*expression1 << expression2*

将 *expression1* 和 *expression2* 转换为 32 位整数值；可将其分别称为 **V1** 和 **V2**。将值 **V1** 的所有位向左移动 **V2** 个位置。将此运算中 **V1** 移到左端以外的位舍去，并在右端空出位的位置插入零。将一个值向左侧移动一位与这个值乘以 2 等效。

浮点数通过舍去小数点后面的所有位来转换为整数。正整数转换为无符号的十六进制值，其最大值为 4294967295 或 0xFFFFFFFF；转换大于最大值的数时，会舍去最高有效位，因此该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值，其最小值为 -2147483648 或 0x80000000；小于最小值的数转换为精度更高的 2 的补码，同时也会舍去最高有效位。

由于返回值解释为带符号的 2 的补码数，因此，返回值将是 -2147483648 到 2147483647 范围中的一个整数。

可用性：Flash Player 5；ActionScript 1.0

### 操作数

*expression1* : Number - 要向左移位的数字或表达式。

*expression2* : Number - 转换为从 0 到 31 的整数的数字或表达式。

### 返回

Number - 按位运算的结果。



## 示例

在下面的示例中，整数 1 向左移 10 位：x = 1 << 10 此运算的结果为 x = 1024。这是因为十进制的 1 等于二进制的 1，二进制的 1 向左移 10 位是二进制的 10000000000，而二进制的 10000000000 就是十进制的 1024。在下面的示例中，整数 7 向左移 8 位：x = 7 << 8 此运算的结果为 x = 1792。这是因为十进制的 7 等于二进制的 111，二进制的 111 向左移 8 位是二进制的 11100000000，而二进制的 11100000000 就是十进制的 1792。如果跟踪下面的示例，则会看到已经将各位向左移两位：

```
// 2 binary == 0010
// 8 binary == 1000
trace(2 << 2); // output: 8
```

## 另请参见

>>= 按位向右移位并赋值运算符， >> 按位向右移位运算符， <<= 按位向左移位并赋值运算符， >>> 按位无符号向右移位运算符， >>>= 按位无符号向右移位并赋值运算符

## <<= 按位向左移位并赋值运算符

*expression1* <<= *expression2*

此运算符执行按位向左移位 (<<=) 运算，并将内容作为结果存储在 *expression1* 中。下面的两个表达式是等效的：

```
A <<= B;
A = (A << B)
```

可用性：Flash Player 5；ActionScript 1.0

## 操作数

*expression1*：Number - 要向左移位的数字或表达式。

*expression2*：Number - 转换为从 0 到 31 的整数的数字或表达式。

## 返回

Number - 按位运算的结果。

## 示例

在下面的示例中，您使用按位向左移位并赋值运算符 (<<=) 将所有位向左移一位：

```
var x:Number = 4;
// shift all bits one slot to the left.
x <<= 1;
trace(x); // output: 8
// 4 decimal = 0100 binary
// 8 decimal = 1000 binary
```

另请参见

<< 按位向左移位运算符, >>= 按位向右移位并赋值运算符, >> 按位向右移位运算符

## ~ 按位 NOT 运算符

*~expression*

也称为对一求补运算符或按位求补运算符。将 *expression* 转换为一个带符号的 32 位整数，然后应用按位 1 的补码。即，将结果中为 0 的每一位设置为 1，并将结果中为 1 的每一位设置为 0。结果是一个带符号的 32 位整数。

例如，十六进制值 0x7777 表示为二进制数：0111011101110111

该十六进制值 ~0x7777 的按位取反的二进制数为：1000100010001000

在十六进制中，它是 0x8888。因此，~0x7777 就是 0x8888。

按位运算符的最常见用法是提供特征位（为每一位填充一个布尔值）。

浮点数通过舍去小数点后面的所有位来转换为整数。正整数转换为无符号的十六进制值，其最大值为 4294967295 或 0xFFFFFFFF；转换大于最大值的数时，会舍去最高有效位，因此该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值，其最小值为 -2147483648 或 0x80000000；小于最小值的数转换为精度更高的 2 的补码，同时也会舍去最高有效位。

由于返回值解释为带符号的 2 的补码数，因此，返回值将是 -2147483648 到 2147483647 范围中的一个整数。

可用性：Flash Player 5；ActionScript 1.0

### 操作数

*expression* : Number - 一个数字。

### 返回

Number - 按位运算的结果。

### 示例

下面的示例说明将按位 NOT (-) 运算符与特征位结合使用的方法：

```
var ReadOnlyFlag:Number = 0x0001; // defines bit 0 as the read-only flag
var flags:Number = 0;
trace(flags);
/* To set the read-only flag in the flags variable,
   the following code uses the bitwise OR:
*/
flags |= ReadOnlyFlag;
trace(flags);
```

```

/* To clear the read-only flag in the flags variable,
   first construct a mask by using bitwise NOT on ReadOnlyFlag.
   In the mask, every bit is a 1 except for the read-only flag.
   Then, use bitwise AND with the mask to clear the read-only flag.
   The following code constructs the mask and performs the bitwise AND:
*/
flags &= ~ReadOnlyFlag;
trace(flags);
// output: 0 1 0

```

另请参见

& 按位 AND 运算符, &= 按位 AND 赋值运算符, ^ 按位 XOR 运算符, ^= 按位 XOR 赋值运算符, | 按位 OR 运算符, |= 按位 OR 赋值运算符

## | 按位 OR 运算符

*expression1* | *expression2*

将 *expression1* 和 *expression2* 转换为无符号的 32 位整数, 然后对于 *expression1* 或 *expression2* 的为 1 的对应位的每一位返回 1。浮点数通过舍去小数点后面的所有位来转换为整数。结果是一个新的 32 位整数。

正整数转换为无符号的十六进制值, 其最大值为 4294967295 或 0xFFFFFFFF; 转换大于最大值的数时, 会舍去最高有效位, 因此该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值, 其最小值为 -2147483648 或 0x80000000; 小于最小值的数转换为精度更高的 2 的补码, 同时也会舍去最高有效位。

由于返回值解释为带符号的 2 的补码数, 因此, 返回值将是 -2147483648 到 2147483647 范围中的一个整数。

可用性: Flash Player 5; ActionScript 1.0

### 操作数

*expression1* : Number - 一个数字。

*expression2* : Number - 一个数字。

### 返回

Number - 按位运算的结果。

## 示例

下面是一个按位 **OR** (**|**) 运算的示例:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 | 1001 = 1111
trace(x | y); // returns 15 decimal (1111 binary)
```

不要混淆单个 **|** (按位 **OR**) 与 **||** (逻辑 **OR**)。

## 另请参见

[& 按位 AND 运算符](#), [&= 按位 AND 赋值运算符](#), [^ 按位 XOR 运算符](#), [^= 按位 XOR 赋值运算符](#),  
[|= 按位 OR 赋值运算符](#), [~ 按位 NOT 运算符](#)

## |= 按位 OR 赋值运算符

*expression1* |= *expression2*

对 *expression1* 赋予 *expression1* | *expression2* 的值。例如, 下面两个语句是等效的:

```
x |= y;
x = x | y;
```

可用性: Flash Player 5 ; ActionScript 1.0

## 操作数

*expression1* : Number - 一个数字或变量。

*expression2* : Number - 一个数字或变量。

## 返回

Number - 按位运算的结果。

## 示例

下面的示例使用按位 **OR** 赋值运算符 (**|=**):

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 |= 1001 = 1111
trace(x |= y); // returns 15 decimal (1111 binary)
```

另请参见

& 按位 AND 运算符, &= 按位 AND 赋值运算符, ^ 按位 XOR 运算符, ^= 按位 XOR 赋值运算符, | 按位 OR 运算符, |= 按位 OR 赋值运算符, ~ 按位 NOT 运算符

## » 按位向右移位运算符

*expression1* >> *expression2*

将 *expression1* 和 *expression2* 转换为 32 位整数, 并将 *expression1* 中的所有位向右移动由 *expression2* 转换所得到的整数指定的位数。移到右端以外的位将被舍去。若要保留原始 *expression* 的符号, 则如果 *expression1* 的最高有效位 (最左端的位) 为 0, 那么左侧的位都填补 0; 如果最高有效位为 1, 那么左侧的位都填补 1。将一个值右移一位等效于将它除以 2 并舍去余数。

浮点数通过舍去小数点后面的所有位来转换为整数。正整数转换为无符号的十六进制值, 其最大值为 4294967295 或 0xFFFFFFFF; 转换大于最大值的数时, 会舍去最高有效位, 因此该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值, 其最小值为 -2147483648 或 0x80000000; 小于最小值的数转换为精度更高的 2 的补码, 同时也会舍去最高有效位。

由于返回值解释为带符号的 2 的补码数, 因此, 返回值将是 -2147483648 到 2147483647 范围中的一个整数。

可用性: Flash Player 5; ActionScript 1.0

### 操作数

*expression1* : Number - 要向右移位的数字或表达式。

*expression2* : Number - 转换为从 0 到 31 的整数的数字或表达式。

### 返回

Number - 按位运算的结果。

### 示例

下面的示例将 65535 转换为 32 位整数, 然后右移 8 位:

```
var x:Number = 65535 >> 8;
trace(x); // outputs 255
```

下面的示例显示上一示例的结果:

```
var x:Number = 255;
```

这是因为十进制的 65535 等于二进制的 1111111111111111 (16 个 1), 二进制的 1111111111111111 向右移 8 位是二进制的 11111111, 而二进制的 11111111 是十进制的 255。因为这个整数是 32 位的, 最高有效位为 0, 所以用 0 来填补位。

下面的示例将 -1 转换为 32 位整数并向右移 1 位：

```
var x:Number = -1 >> 1;  
trace(x); // outputs -1
```

下面的示例显示上一示例的结果：

```
var x:Number = -1;
```

这是因为十进制的 -1 等于二进制的 11111111111111111111111111111111 (32 个 1)，向右移一位使得最低有效位（最右侧的位）被去掉而最高有效位被填补上 1。结果为二进制的 1111111111111111111111111111111 (32 个 1)，表示 32 位整数 -1。

另请参见

[>>= 按位向右移位并赋值运算符](#)

## >>= 按位向右移位并赋值运算符

```
expression1 >>= expression2
```

此运算符执行按位向右移位运算，并将内容作为结果存储在 *expression1* 中。

下面的两个语句是等效的：

```
A >>= B;  
A = (A >> B);
```

可用性：Flash Player 5；ActionScript 1.0

### 操作数

*expression1* : Number - 要向右移位的数字或表达式。

*expression2* : Number - 转换为从 0 到 31 的整数的数字或表达式。

### 返回

Number - 按位运算的结果。

### 示例

下面的带注释代码使用按位向右移位并赋值运算符 (>>=)。

```
function convertToBinary(numberToConvert:Number):String {  
    var result:String = "";  
    for (var i = 0; i<32; i++) {  
        // Extract least significant bit using bitwise AND  
        var lsb:Number = numberToConvert & 1;  
        // Add this bit to the result  
        string result = (lsb ? "1" : "0")+result;  
        // Shift numberToConvert right by one bit, to see next bit  
        numberToConvert >>= 1;  
    }  
}
```

```

    }
    return result;
}
trace(convertToBinary(479));
// Returns the string 0000000000000000000000000111011111
// This string is the binary representation of the decimal
// number 479

```

另请参见

>> [按位向右移位运算符](#)

## >>> 按位无符号向右移位运算符

*expression1 >>> expression2*

除了不保留原始 *expression* 的符号外，此运算符与按位向右移位运算符 (>>) 相同，这是因为左侧的位始终用 0 填充。

浮点数通过舍去小数点后面的所有位来转换为整数。正整数转换为无符号的十六进制值，其最大值为 4294967295 或 0xFFFFFFFF；转换大于最大值的数时，会舍去最高有效位，因此该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值，其最小值为 -2147483648 或 0x80000000；小于最小值的数转换为精度更高的 2 的补码，同时也会舍去最高有效位。

可用性：Flash Player 5；ActionScript 1.0

### 操作数

*expression1* : Number - 要向右移位的数字或表达式。

*expression2* : Number - 转换为从 0 到 31 的整数的数字或表达式。

### 返回

Number - 按位运算的结果。

### 示例

下面的示例将 -1 转换为 32 位整数并向右移 1 位：

```

var x:Number = -1 >>> 1;
trace(x); // output: 2147483647

```

这是因为十进制的 -1 是二进制的 11111111111111111111111111111111 (32 个 1)，当向右（无符号）移 1 位时，最低有效位（最右端的位）被去掉，而最高有效位（最左端的位）被填补上 0。结果为二进制的 01111111111111111111111111111111，表示 32 位整数 2147483647。

另请参见

[>>= 按位向右移位并赋值运算符](#)

## >>>= 按位无符号向右移位并赋值运算符

*expression1* >>>= *expression2*

执行无符号按位向右移位运算，并将内容作为结果存储在 *expression1* 中。下面的两个语句是等效的：

```
A >>>= B;  
A = (A >>> B);
```

可用性：Flash Player 5；ActionScript 1.0

### 操作数

*expression1*：Number - 要向右移位的数字或表达式。

*expression2*：Number - 转换为从 0 到 31 的整数的数字或表达式。

### 返回

Number - 按位运算的结果。

### 示例

另请参见

[>>> 按位无符号向右移位运算符](#)，[>>= 按位向右移位并赋值运算符](#)

## ^ 按位 XOR 运算符

*expression1* ^ *expression2*

将 *expression1* 和 *expression2* 转换为无符号 32 位整数，然后对于 *expression1* 或 *expression2* 中为 1（但不在两者中同时为 1）的相应位的每一位返回 1。浮点数通过舍去小数点后面的所有位来转换为整数。结果是一个新的 32 位整数。

正整数转换为无符号的十六进制值，其最大值为 4294967295 或 0xFFFFFFFF；转换大于最大值的数时，会舍去最高有效位，因此该值仍保持为 32 位。负数通过 2 的补码标记转换为无符号的十六进制值，其最小值为 -2147483648 或 0x80000000；小于最小值的数转换为精度更高的 2 的补码，同时也会舍去最高有效位。

由于返回值解释为带符号的 2 的补码数，因此，返回值将是 -2147483648 到 2147483647 范围中的一个整数。

可用性：Flash Player 5；ActionScript 1.0



## 操作数

expression1 : Number - 一个数字。

expression2 : Number - 一个数字。

## 返回

Number - 按位运算的结果。

## 示例

下面的示例对十进制的 **15** 和 **9** 使用按位 **XOR** 运算符，然后将结果赋予变量 **x**：

```
// 15 decimal = 1111 binary
// 9 decimal = 1001 binary
var x:Number = 15 ^ 9;
trace(x);
// 1111 ^ 1001 = 0110
// returns 6 decimal (0110 binary)
```

## 另请参见

[& 按位 AND 运算符](#), [&= 按位 AND 赋值运算符](#), [^= 按位 XOR 赋值运算符](#), [| 按位 OR 运算符](#), [|= 按位 OR 赋值运算符](#), [~ 按位 NOT 运算符](#)

## **^=** 按位 XOR 赋值运算符

*expression1* ^= *expression2*

对 *expression1* 赋予 *expression1* ^ *expression2* 的值。例如，下面两个语句是等效的：

```
x ^= y;
x = x ^ y;
```

可用性：Flash Player 5；ActionScript 1.0

## 操作数

expression1 : Number - 整数和变量。

expression2 : Number - 整数和变量。

## 返回

Number - 按位运算的结果。

## 示例

下面的示例说明按位 **XOR** 赋值 (**^=**) 运算：

```
// 15 decimal = 1111 binary
var x:Number = 15;
```

```
// 9 decimal = 1001 binary
var y:Number = 9;
trace(x ^ y); // returns 6 decimal (0110 binary)
```

另请参见

[& 按位 AND 运算符](#), [&= 按位 AND 赋值运算符](#), [^ 按位 XOR 运算符](#), [| 按位 OR 运算符](#), [|= 按位 OR 赋值运算符](#), [~ 按位 NOT 运算符](#)

## /\*..\*/ 注释块分隔符运算符

```
/* comment */
/* comment
comment */
```

指示一行或多行脚本注释。出现在注释开始标签 (/\*) 和注释结束标签 (\*/) 之间的任何字符都被 **ActionScript** 解释程序解释为注释并忽略。使用 // (注释分隔符) 可标识单行注释。使用 /\* 注释分隔符可在多个连续行上标识注释。在使用这种格式的注释分隔符时, 如果缺少结束标签 (\*/), 就会返回一条错误消息。尝试嵌套注释也会返回一条错误消息。使用了注释开始标签 (/\*) 后, 不管在开始标签和结束标签之间放置了多少个注释开始标签 (/\*), 第一个注释结束标签 (\*/) 都会结束此注释。

可用性: Flash Player 5 ; ActionScript 1.0

### 操作数

comment - 任何字符。

### 示例

下面的脚本在脚本的开头使用注释分隔符:

```
/* records the X and Y positions of
the ball and bat movie clips */
var ballX:Number = ball_mc._x;
var ballY:Number = ball_mc._y;
var batX:Number = bat_mc._x;
var batY:Number = bat_mc._y;
```

下面的嵌套注释尝试将导致一条错误消息:

```
/* this is an attempt to nest comments.
/* But the first closing tag will be paired
with the first opening tag */
and this text will not be interpreted as a comment */
```

另请参见

[// 注释行分隔符运算符](#)

## , 逗号运算符

*(expression1 , expression2 [, expressionN... ])*

计算 *expression1*, 然后计算 *expression2*, 依此类推。此运算符主要与 for 循环语句一起使用, 并且通常与括号运算符 `()` 一起使用。

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*expression1* : Number - 要计算的表达式。

*expression2* : Number - 要计算的表达式。

*expressionN* : Number - 要计算的任意数目的其它表达式。

### 返回

Object - *expression1*、*expression2* 等的值。

### 示例

下面的示例在 for 循环中使用逗号运算符 `(,)`:

```
for (i = 0, j = 0; i < 3 && j < 3; i++, j+=2) {  
    trace("i = " + i + ", j = " + j);  
}  
// Output:  
// i = 0, j = 0  
// i = 1, j = 2
```

下面的示例使用逗号 `(,)` 运算符但不使用括号 `()` 运算符, 并说明在不使用括号 `()` 运算符时逗号运算符仅返回第一个表达式的值:

```
var v:Number = 0;  
v = 4, 5, 6;  
trace(v); // output: 4
```

下面的示例使用逗号 `(,)` 运算符以及括号 `()` 运算符, 并说明逗号运算符在与括号 `()` 运算符一起使用时返回最后一个表达式的值:

```
var v:Number = 0;  
v = (4, 5, 6);  
trace(v); // output: 6
```

下面的示例使用逗号 `(,)` 运算符, 但不使用括号 `()` 运算符, 并且说明: 逗号运算符按顺序计算所有表达式, 但返回第一个表达式的值。计算第二个表达式 `z++`, 并将 `z` 加一。

```
var v:Number = 0;  
var z:Number = 0;  
v = v + 4 , z++, v + 6;  
trace(v); // output: 4  
trace(z); // output: 1
```

下面的示例除添加了括号 () 运算符外与上一示例相同，再一次说明逗号 (,) 运算符在与括号 () 运算符一起使用时返回表达式系列中最后一个表达式的值：

```
var v:Number = 0;
var z:Number = 0;
v = (v + 4, z++, v + 6);
trace(v); // output: 6
trace(z); // output: 1
```

另请参见

[\(\) 括号运算符](#)

## 添加连接（字符串）运算符

*string1* add *string2*

自 **Flash Player 5** 后不推荐使用。**Macromedia** 建议在为 **Flash Player 5** 或更高版本创建内容时使用加运算符 (+)。此运算符在 **Flash Player 8** 或更高版本中不受支持。

连接两个或更多字符串。加法运算符 (+) 替换了 **Flash 4** 的 & 运算符；当将使用 & 运算符的 **Flash Player 4** 文件引入 **Flash 5** 或更高版本的创作环境中时，就会自动对其进行转换以使用加法运算符 (+) 进行字符串连接。如果您为 **Flash Player 4** 或更早版本的播放器创建内容，请使用加法 (+) 运算符来连接字符串。

可用性：Flash Player 4；ActionScript 1.0

操作数

*string1* : String - 一个字符串。

*string2* : String - 一个字符串。

返回

String - 连接后的字符串。

另请参见

[+ 加法运算符](#)

## ? : 条件运算符

*expression1 ? expression2 : expression3*

指示 **Flash** 计算 *expression1*, 如果 *expression1* 的值为 `true`, 则返回值 *expression2*; 否则返回值 *expression3*。

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*expression1* : Object - 一个计算结果为布尔值的表达式; 通常为像 `x < 5` 这样的比较表达式。

*expression2* : Object - 任意类型的值。

*expression3* : Object - 任意类型的值。

### 返回

Object - *expression2* 或 *expression3* 的值。

### 示例

下面的语句将变量 `x` 的值赋予变量 `z`, 因为 *expression1* 的计算结果是 `true`:

```
var x:Number = 5;
var y:Number = 10;
var z = (x < 6) ? x: y;
trace (z); // returns 5
```

下面的示例说明简写的条件语句:

```
var timecode:String = (new Date().getHours() < 11) ? "AM" : "PM";
trace(timecode);
```

也可以按普通形式编写该条件语句, 如下例所示:

```
if (new Date().getHours() < 11) {
    var timecode:String = "AM";
} else {
    var timecode:String = "PM";
} trace(timecode);
```

## -- 递减运算符

--*expression*  
*expression*--

从 *expression* 中减 1 的预先递减和滞后递减一元运算符。*expression* 可以是变量、数组中的元素或对象的属性。此运算符的预先递减格式 (--*expression*) 从 *expression* 中减去 1, 然后返回结果。此运算符的滞后递减格式 (*expression*--) 从 *expression* 中减去 1, 然后返回 *expression* 的初始值 (即减去 1 之前的值)。

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*expression* : Number - 一个数字或计算结果为数字的一个变量。

### 返回

Number - 被递减的值的结果。

### 示例

此运算符的预先递减格式将 x 递减为 2 ( $x - 1 = 2$ ), 并将结果作为 y 返回:

```
var x:Number = 3;  
var y:Number = --x; //y is equal to 2
```

此运算符的滞后递减格式将 x 递减为 2 ( $x - 1 = 2$ ), 并将 x 的初始值作为结果 y 返回:

```
var x:Number = 3;  
var y:Number = x--; //y is equal to 3
```

下面的示例从 10 到 1 循环, 循环的每次迭代将计数器变量 i 减 1。

```
for (var i = 10; i>0; i--) {  
    trace(i);  
}
```

## / 除法运算符

*expression1* / *expression2*

*expression1* 除以 *expression2*。除法运算的结果为双精度浮点数。

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*expression* : Number - 一个数字或计算结果为数字的一个变量。

### 返回

Number - 运算的浮点结果。

## 示例

下面的语句除舞台的当前宽度和高度，然后将结果显示在“输出”面板中。

```
trace(Stage.width/2);  
trace(Stage.height/2);
```

对于默认的舞台宽度和高度 550 x 400，输出是 275 和 150。

另请参见

[% 模运算符](#)

## /= 除法赋值运算符

*expression1 /= expression2*

对 *expression1* 赋予 *expression1* / *expression2* 的值。例如，下面两个语句是等效的：

```
x /= y;  
x = x / y;
```

可用性：Flash Player 4；ActionScript 1.0

## 操作数

*expression1* : Number - 一个数字或计算结果为数字的一个变量。

*expression2* : Number - 一个数字或计算结果为数字的一个变量。

## 返回

Number - 一个数字。

## 示例

下面的代码说明如何将除法赋值运算符 (/=) 与变量和数字结合使用：

```
var x:Number = 10;  
var y:Number = 2;  
x /= y; trace(x); // output: 5
```

另请参见

[/ 除法运算符](#)

## . 点运算符

*object.property\_or\_methodinstance.name.variable*  
*instance.name.childinstanceinstance.name.childinstance.variable*

用于定位影片剪辑层次结构，以便访问嵌套的（子级）影片剪辑、变量或属性。点运算符也用于测试或设置对象或顶级类的属性、执行对象或顶级类的方法或创建数据结构。

可用性：Flash Player 4； ActionScript 1.0

### 操作数

**object**：Object - 类的一个实例。此对象可以是任意内置 **ActionScript** 类或自定义类的实例。此参数总是在点 (.) 运算符的左侧。

**property\_or\_method** - 与对象相关联的属性或方法的名称。内置类的所有有效方法和属性都会在该类的方法和属性摘要表中列出。此参数总是在点 (.) 运算符的右侧。

**instance.name**：MovieClip - 影片剪辑的实例名称。**variable** - 点运算符 (.) 左侧的实例名称也可以表示影片剪辑的时间轴上的变量。

**childinstance**：MovieClip - 一个从属于或嵌套在另一个影片剪辑的影片剪辑实例。

### 返回

Object - 在点运算符右侧指定的方法、属性或影片剪辑。

### 示例

下面的示例标识影片剪辑 **person\_mc** 中变量 **hairColor** 的当前值：

```
person_mc.hairColor
```

**Flash 4** 创作环境不支持点语法，但是为 **Flash Player 4** 发布的 **Flash MX 2004** 文件可以使用点运算符。前一示例等效于以下（不鼓励使用的）**Flash 4** 语法：

```
/person_mc:hairColor
```

下面的示例在 **\_root** 范围内创建新的影片剪辑。然后在名为 **container\_mc** 的影片剪辑内创建一个文本字段。该文本字段的 **autoSize** 属性设置为 **true**，然后用当前日期进行填充。

```
this.createEmptyMovieClip("container_mc", this.getNextHighestDepth());  
this.container_mc.createTextField("date_txt", this.getNextHighestDepth(), 0,  
    0, 100, 22);  
this.container_mc.date_txt.autoSize = true;  
this.container_mc.date_txt.text = new Date();
```

在针对 **SWF** 文件内的实例时以及需要为那些实例设置属性和值时，可使用点 (.) 运算符。



## == 等于运算符

*expression1 == expression2*

测试两个表达式是否相等。如果表达式相等，则结果为 `true`。

确定是否相等取决于参数的数据类型：

- 数字和布尔值按值进行比较，如果它们具有相同的值，则视为相等。
- 如果字符串表达式具有相同的字符数，而且这些字符都相同，则这些字符串表达式相等。
- 表示对象、数组和函数的变量按引用进行比较。如果两个变量引用同一个对象、数组或函数，则它们相等。而两个单独的数组即使具有相同数量的元素，也永远不会被视为相等。

当按值进行比较时，如果 *expression1* 和 *expression2* 为不同的数据类型，**ActionScript** 会尝试将 *expression2* 的数据类型转换为与 *expression1* 匹配的数据类型。

可用性：Flash Player 5； **ActionScript** 1.0

### 操作数

*expression1* : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

*expression2* : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

### 返回

Boolean - 比较的布尔结果。

### 示例

下面的示例将等于运算符 (==) 与 `if` 语句结合使用：

```
var a:String = "David", b:String = "David";
if (a == b) {
    trace("David is David");
}
```

下面的示例说明比较混合类型的运算的结果：

```
var x:Number = 5;
var y:String = "5";
trace(x == y); // output: true
var x:String = "5";
var y:String = "66";
trace(x == y); // output: false
var x:String = "chris";
var y:String = "steve";
trace(x == y); // output: false
```

下面的示例说明按引用进行比较。第一个示例比较两个具有相同长度和元素的数组。对于这两个数组，等于运算符将返回 **false**。虽然这两个数组看起来相等，但是按引用进行比较要求它们都引用同一数组。第二个示例创建 **thirdArray** 变量，该变量与变量 **firstArray** 指向同一数组。对于这两个数组，等于运算符将返回 **true**，因为这两个变量引用同一数组。

```
var firstArray:Array = new Array("one", "two", "three");
var secondArray:Array = new Array("one", "two", "three");
trace(firstArray == secondArray);
// will output false
// Arrays are only considered equal
// if the variables refer to the same array.
var thirdArray:Array = firstArray;
trace(firstArray == thirdArray); // will output true
```

另请参见

[! 逻辑 NOT 运算符](#), [!= 不等于运算符](#), [!== 不全等运算符](#), [&& 逻辑 AND 运算符](#), [|| 逻辑 OR 运算符](#), [=== 全等运算符](#)

## eq 等于（字符串）运算符

*expression1* eq *expression2*

自 **Flash Player 5** 后不推荐使用。不推荐使用此运算符，而推荐使用 **==**（equality）运算符。比较两个表达式是否相等，如果 *expression1* 的字符串表示形式等于 *expression2* 的字符串表示形式，则返回值 **true**，否则返回 **false**。

可用性：Flash Player 4；ActionScript 1.0

操作数

*expression1* : Object - 数字、字符串或变量。

*expression2* : Object - 数字、字符串或变量。

返回

Boolean - 比较的结果。

另请参见

[== 等于运算符](#)

## › 大于运算符

*expression1* > *expression2*

比较两个表达式，确定 *expression1* 是否大于 *expression2*；如果是，则此运算符返回 true。如果 *expression1* 小于或等于 *expression2*，则此运算符返回 false。使用字母顺序计算字符串表达式；所有的大写字母排在小写字母的前面。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Object - 一个数字或字符串。

*expression2* : Object - 一个数字或字符串。

### 返回

Boolean - 比较的布尔结果。

### 示例

在下面的示例中，大于运算符 (>) 用于确定文本字段 `score_txt` 的值是否大于 90：

```
if (score_txt.text>90) {  
    trace("Congratulations, you win!");  
} else {  
    trace("sorry, try again");  
}
```

## gt 大于（字符串）运算符

*expression1* gt *expression2*

自 **Flash Player 5** 后不推荐使用。不推荐使用此运算符，而推荐使用 >（大于）运算符。

将 *expression1* 的字符串表达式与 *expression2* 的字符串表达式相比较，如果 *expression1* 大于 *expression2*，则返回 true，否则返回 false。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Object - 数字、字符串或变量。

*expression2* : Object - 数字、字符串或变量。

### 返回

Boolean - 比较的布尔结果。

### 另请参见

[› 大于运算符](#)

## >= 大于或等于运算符

*expression1* >= *expression2*

比较两个表达式，确定 *expression1* 是大于或等于 *expression2* (true) 还是 *expression1* 小于 *expression2* (false)。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1*：Object - 一个字符串、整数或浮点数。

*expression2*：Object - 一个字符串、整数或浮点数。

### 返回

Boolean - 比较的布尔结果。

### 示例

在下面的示例中，大于或等于运算符 (>=) 用于确定当前小时是否大于或等于 12：

```
if (new Date().getHours() >= 12) {  
    trace("good afternoon");  
} else {  
    trace("good morning");  
}
```

## ge 大于或等于（字符串）运算符

*expression1* ge *expression2*

自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 >=（大于或等于）运算符。

将 *expression1* 的字符串表达式与 *expression2* 的字符串表达式相比较，如果 *expression1* 大于或等于 *expression2*，则返回 true，否则返回 false。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1*：Object - 数字、字符串或变量。

*expression2*：Object - 数字、字符串或变量。

### 返回

Boolean - 比较的结果。

### 另请参见

[>= 大于或等于运算符](#)

## ++ 递增运算符

```
++expression  
expression++
```

将 *expression* 加 1 的预先递增和滞后递增一元运算符。*expression* 可以是变量、数组中的元素或对象的属性。运算符的预先递增格式 (*++expression*) 将 *expression* 加 1, 然后返回结果。此运算符的滞后递增格式 (*expression++*) 将 *expression* 加 1, 并返回 *expression* 的初始值 (即增加之前的值)。

运算符的预先递增格式将 *x* 递增为 2 ( $x + 1 = 2$ ), 并将结果作为 *y* 返回:

```
var x:Number = 1;  
var y:Number = ++x;  
trace("x:"+x); //traces x:2  
trace("y:"+y); //traces y:2
```

此运算符的滞后递增格式将 *x* 递增为 2 ( $x + 1 = 2$ ), 并将 *x* 的初始值作为结果 *y* 返回:

```
var x:Number = 1;  
var y:Number = x++;  
trace("x:"+x); //traces x:2  
trace("y:"+y); //traces y:1
```

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*expression* : Number - 一个数字或计算结果为数字的一个变量。

### 返回

Number - 递增的结果。

### 示例

下面的示例将 ++ 用作滞后递增运算符, 以使 while 循环运行五次:

```
var i:Number = 0;  
while (i++ < 5) {  
    trace("this is execution " + i);  
}  
/* output:  
this is execution 1  
this is execution 2  
this is execution 3  
this is execution 4  
this is execution 5  
*/
```

下面的示例将 ++ 用作预先递增运算符：

```
var a:Array = new Array();
var i:Number = 0;
while (i < 10) {
    a.push(++i);
}
trace(a.toString()); //traces: 1,2,3,4,5,6,7,8,9,10
```

此示例也将 ++ 用作预先递增运算符。

```
var a:Array = [];
for (var i = 1; i <= 10; ++i) {
    a.push(i);
}
trace(a.toString()); //traces: 1,2,3,4,5,6,7,8,9,10
```

此脚本在“输出”面板中显示以下结果：1,2,3,4,5,6,7,8,9,10 下面的示例将 ++ 用作 while 循环中的滞后递增运算符：

```
// using a while loop
var a:Array = new Array();
var i:Number = 0;
while (i < 10) {
    a.push(i++);
}
trace(a.toString()); //traces 0,1,2,3,4,5,6,7,8,9
```

下面的示例将 ++ 用作 for 循环中的滞后递增运算符：

```
// using a for loop
var a:Array = new Array();
for (var i = 0; i < 10; i++) {
    a.push(i);
}
trace(a.toString()); //traces 0,1,2,3,4,5,6,7,8,9
```

此脚本在“输出”面板中显示以下结果：

0,1,2,3,4,5,6,7,8,9

## != 不等于运算符

*expression1* != *expression2*

测试结果是否与等于运算符 (==) 正好相反。如果 *expression1* 等于 *expression2*, 则结果为 false。与等于运算符 (==) 相同, 确定是否相等取决于所比较的数据类型, 如以下列表中的说明:

- 数字、字符串和布尔值按值进行比较。
- 对象、数组和函数按引用进行比较。
- 变量根据其类型按值或按引用进行比较。

按值进行比较是指大多数人通常所理解的相等, 即两个表达式具有相同的值。例如, 按值比较时, 表达式 (2 + 3) 等于表达式 (1 + 4)。

按引用进行比较是指只有当两个表达式都引用同一个对象、数组或函数时, 它们才相等。不对对象、数组或函数内部的值进行比较。

当按值进行比较时, 如果 *expression1* 和 *expression2* 为不同的数据类型, **ActionScript** 会尝试将 *expression2* 的数据类型转换为与 *expression1* 匹配的数据类型。

可用性: **Flash Player 5 ; ActionScript 1.0**

### 操作数

*expression1* : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

*expression2* : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

### 返回

Boolean - 比较的布尔结果。

### 示例

下面的示例说明不等于运算符 (!=) 的结果:

```
trace(5 != 8); // returns true
trace(5 != 5) //returns false
```

下面的示例说明在 if 语句中使用不等于运算符 (!=):

```
var a:String = "David";
var b:String = "Fool";
if (a != b) {
    trace("David is not a fool");
}
```

下面的示例说明按引用比较两个函数：

```
var a:Function = function() { trace("foo"); };
var b:Function = function() { trace("foo"); };
a(); // foo
b(); // foo
trace(a != b); // true
a = b;
a(); // foo
b(); // foo
trace(a != b); // false
// trace statement output: foo foo true foo foo false
```

下面的示例说明按引用比较两个数组：

```
var a:Array = [ 1, 2, 3 ];
var b:Array = [ 1, 2, 3 ];
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a!=b); // true
a = b;
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a != b); // false
// trace statement output: 1,2,3 1,2,3 true 1,2,3 1,2,3 false
```

另请参见

[! 逻辑 NOT 运算符](#)，[!= 不全等运算符](#)，[&& 逻辑 AND 运算符](#)，[|| 逻辑 OR 运算符](#)，[== 等于运算符](#)，[=== 全等运算符](#)

## <> 不等于运算符

*expression1* <> *expression2*

自 Flash Player 5 后不推荐使用。此运算符已不推荐使用。Macromedia 建议您使用 != (inequality) 运算符。

测试结果是否与等于运算符 (==) 正好相反。如果 *expression1* 等于 *expression2*，则结果为 false。与等于运算符 (==) 一样，相等的定义取决于所比较的数据类型：

- 数字、字符串和布尔值按值进行比较。
- 对象、数组和函数按引用进行比较。
- 变量是按值或按引用进行比较的，具体取决于其类型。

可用性：Flash Player 2；ActionScript 1.0



### 操作数

expression1 : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

expression2 : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

### 返回

Boolean - 比较的布尔结果。

### 另请参见

[!= 不等于运算符](#)

## instanceof 运算符

*object instanceof classConstructor*

测试 object 是 classConstructor 的实例还是 classConstructor 的子类。instanceof 运算符不会将原始类型转换为包装对象。例如，下面的代码返回 true：

```
new String("Hello") instanceof String;
```

而下面的代码则返回 **false**：

```
"Hello" instanceof String;
```

可用性：Flash Player 6；ActionScript 1.0

### 操作数

object : Object - 一个 **ActionScript** 对象。

classConstructor : Function - 对 **ActionScript** 构造函数（如 String 或 Date）的引用。

### 返回

Boolean - 如果 object 是 classConstructor 的实例或子类，则 instanceof 返回 true，否则返回 false。另外，\_global instanceof Object 返回 false。

### 另请参见

[typeof 运算符](#)

## < 小于运算符

*expression1* < *expression2*

比较两个表达式，确定 *expression1* 是否小于 *expression2*；如果是，则此运算符返回 true。如果 *expression1* 大于或等于 *expression2*，则该运算符返回 false。使用字母顺序计算字符串表达式；所有的大写字母排在小写字母的前面。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Number - 一个数字或字符串。

*expression2* : Number - 一个数字或字符串。

### 返回

Boolean - 比较的布尔结果。

### 示例

下面的示例说明为数值和字符串比较返回的 true 和 false：

```
trace(3 < 10); // true
trace(10 < 3); // false
trace("Allen" < "Jack"); // true
trace("Jack" < "Allen"); //false
trace("11" < "3"); // true
trace("11" < 3); // false (numeric comparison)
trace("C" < "abc"); // true
trace("A" < "a"); // true
```

## lt 小于（字符串）运算符

*expression1* lt *expression2*

自 **Flash Player 5** 后不推荐使用。不推荐使用此运算符，而推荐使用 <（小于）运算符。

将 *expression1* 与 *expression2* 相比较，如果 *expression1* 小于 *expression2*，则返回 true，否则返回 false。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Object - 数字、字符串或变量。

*expression2* : Object - 数字、字符串或变量。

返回

Boolean - 比较的结果。

另请参见

[< 小于运算符](#)

## <= 小于或等于运算符

*expression1* <= *expression2*

比较两个表达式，确定 *expression1* 是否小于或等于 *expression2*；如果是，则此运算符返回 true。如果 *expression1* 大于 *expression2*，则该运算符返回 false。使用字母顺序计算字符串表达式；所有的大写字母排在小写字母的前面。

可用性：Flash Player 4；ActionScript 1.0

操作数

*expression1* : Object - 一个数字或字符串。

*expression2* : Object - 一个数字或字符串。

返回

Boolean - 比较的布尔结果。

示例

下面的示例说明为数值和字符串比较返回的 true 和 false：

```
trace(5 <= 10); // true
trace(2 <= 2); // true
trace(10 <= 3); // false
trace("Allen" <= "Jack"); // true
trace("Jack" <= "Allen"); // false
trace("11" <= "3"); // true
trace("11" <= 3); // false (numeric comparison)
trace("C" <= "abc"); // true
trace("A" <= a); // true
```

## le 小于或等于（字符串）运算符

*expression1 le expression2*

自 **Flash Player 5** 后不推荐使用。在 **Flash 5** 中不推荐使用此运算符，而推荐使用 **<=**（小于或等于）运算符。

将 *expression1* 与 *expression2* 比较，如果 *expression1* 小于或等于 *expression2*，则返回值 **true**，否则返回 **false**。

可用性：**Flash Player 4**；**ActionScript 1.0**

### 操作数

*expression1*：Object - 数字、字符串或变量。

*expression2*：Object - 数字、字符串或变量。

### 返回

Boolean - 比较的结果。

### 另请参见

[<= 小于或等于运算符](#)

## // 注释行分隔符运算符

*// comment*

指示脚本注释的开始。出现在注释分隔符 (*//*) 和行结束字符之间的任何字符都被 **ActionScript** 解释程序解释为注释并忽略。

可用性：**Flash Player 1.0**；**ActionScript 1.0**

### 操作数

*comment* - 任何字符。

### 示例

下面的脚本使用注释分隔符将第一、第三、第五和第七行标识为注释：

```
// record the X position of the ball movie clip
var ballX:Number = ball_mc._x;
// record the Y position of the ball movie clip
var ballY:Number = ball_mc._y;
// record the X position of the bat movie clip
var batX:Number = bat_mc._x;
// record the Y position of the ball movie clip
var batY:Number = bat_mc._y;
```

另请参见

[/\\*..\\*/ 注释块分隔符运算符](#)

## && 逻辑 AND 运算符

*expression1* && *expression2*

对两个表达式的值执行布尔运算。如果 *expression1* 和 *expression2* 都为 true，则返回 true；否则返回 false。

表达式	计算结果
true&&true	true
true&&false	false
false&&false	false
false&&true	false

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1*：Number - 布尔值或要转换为布尔值的表达式。

*expression2*：Number - 布尔值或要转换为布尔值的表达式。

### 返回

Boolean - 逻辑运算的布尔结果。

### 示例

下面的示例使用逻辑 AND 运算符 (&&) 执行一个测试，确定游戏者是否已经赢得游戏。在游戏过程中，当游戏者赢得一轮或者得到计分点时，就会更新变量 *turns* 和 *score*。在 3 轮之内游戏者的得分达到或超过 75 时，脚本就会在“输出”面板中显示 “You Win the Game!”。

```
var turns:Number = 2;
var score:Number = 77;
if ((turns <= 3) && (score >= 75)) {
    trace("You Win the Game!");
} else {
    trace("Try Again!");
}
// output: You Win the Game!
```

另请参见

[! 逻辑 NOT 运算符](#), [!= 不等于运算符](#), [!== 不全等运算符](#), [|| 逻辑 OR 运算符](#), [== 等于运算符](#), [=== 全等运算符](#)

## and 逻辑 AND 运算符

*condition1 and condition2*

自 Flash Player 5 后不推荐使用。Macromedia 建议使用逻辑 AND 运算符 (&&)。

在 Flash Player 4 中执行逻辑 AND (&&) 运算。如果两个表达式的计算结果都是 true, 则整个表达式的计算结果为 true。

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*condition1* : Boolean - *condition1,condition2* 计算结果为 true 或 false 的条件或表达式。

*condition2* : Boolean - *condition1,condition2* 计算结果为 true 或 false 的条件或表达式。

### 返回

Boolean - 逻辑运算的布尔结果。

另请参见

[&& 逻辑 AND 运算符](#)

## ! 逻辑 NOT 运算符

*! expression*

对变量或表达式的布尔值取反。如果 *expression* 是具有绝对的或经过转换的值 true 的变量, 则 *! expression* 的值为 false。如果表达式 *x && y* 的计算结果为 false, 则表达式 *!(x && y)* 的计算结果为 true。

下面的表达式说明了使用逻辑 NOT (!) 运算符的结果:

*! true* 返回 *false* *! false* 返回 *true*

可用性: Flash Player 4 ; ActionScript 1.0

### 操作数

*expression* : Boolean - 计算结果为布尔值的一个表达式或变量。

## 返回

Boolean - 逻辑运算的布尔结果。

## 示例

在下面的示例中，变量 `happy` 设置为 `false`。if 条件计算条件 `!happy`，如果后者为 `true`，则 `trace()` 语句将一个字符串发送到“输出”面板。

```
var happy:Boolean = false;
if (!happy) {
    trace("don't worry, be happy"); //traces don't worry, be happy
}
```

该语句跟踪，因为 `!false` 等于 `true`。

## 另请参见

[!= 不等于运算符](#)，[!== 不全等运算符](#)，[&& 逻辑 AND 运算符](#)，[|| 逻辑 OR 运算符](#)，[== 等于运算符](#)，[=== 全等运算符](#)

# not 逻辑 NOT 运算符

*not expression*

自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 `!` (logical NOT) 运算符。

在 Flash Player 4 中执行逻辑 NOT (!) 运算。

可用性：Flash Player 4；ActionScript 1.0

## 操作数

*expression* : Object - 转换为布尔值的变量或其它表达式。

## 返回

Boolean - 逻辑运算的结果。

## 另请参见

[! 逻辑 NOT 运算符](#)

## || 逻辑 OR 运算符

*expression1* || *expression2*

计算 *expression1* (运算符左侧的表达式), 如果表达式的计算结果为 `true`, 则返回 `true`。如果 *expression1* 的计算结果为 `false`, 则计算 *expression2* (运算符右侧的表达式)。如果 *expression2* 的计算结果为 `false`, 则最终结果为 `false`; 否则, 最终结果为 `true`。

如果将函数调用用作 *expression2*, 则在 *expression1* 的计算结果为 `true` 时, 该调用不会执行该函数。

如果其中任一表达式 (或两者) 的计算结果为 `true`, 则结果为 `true`; 只有当两个表达式的计算结果都为 `false` 时, 结果才为 `false`。您可以将逻辑 **OR** 运算符与任意多个操作数一起使用; 如果任一操作数的计算结果为 `true`, 则结果为 `true`。

可用性: Flash Player 4; ActionScript 1.0

### 操作数

*expression1* : Number - 布尔值或要转换为布尔值的表达式。

*expression2* : Number - 布尔值或要转换为布尔值的表达式。

### 返回

Boolean - 逻辑运算的结果。

### 示例

下面的示例在 `if` 语句中使用逻辑 **OR** 运算符 (`||`)。第二个表达式的计算结果为 `true`, 因此最终结果为 `true`:

```
var x:Number = 10;
var y:Number = 250;
var start:Boolean = false;
if ((x > 25) || (y > 200) || (start)) {
    trace("the logical OR test passed"); // output: the logical OR test passed
}
```

因为 `if` 语句中的条件之一为 `true` (`y>200`), 将出现逻辑 **OR** 测试通过的消息。虽然其它两个表达式的计算结果为 `false`, 但是只要有一个条件的计算结果为 `true`, `if` 块就会执行。

下面的示例说明使用函数调用作为 *expression2* 会怎样导致意外的结果。如果运算符左侧的表达式计算结果为 `true`, 则返回该结果, 而不计算右侧的表达式 (不调用函数 `fx2()`)。

```
function fx1():Boolean {
    trace("fx1 called");
    return true;
}
function fx2():Boolean {
    trace("fx2 called");
    return true;
}
```



```
}  
if (fx1() || fx2()) {  
    trace("IF statement entered");  
}
```

将以下内容发送到“输出”面板：fx1 调用输入的 IF 语句。

另请参见

[! 逻辑 NOT 运算符](#), [!= 不等于运算符](#), [!== 不全等运算符](#), [&& 逻辑 AND 运算符](#), [== 等于运算符](#), [=== 全等运算符](#)

## or 逻辑 OR 运算符

*condition1 or condition2*

自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 `||` (logical OR) 运算符。

计算 *condition1* 和 *condition2*，如果任一表达式的计算结果为 true，则整个表达式的计算结果为 true。

可用性：Flash Player 4；ActionScript 1.0

操作数

*condition1* : Boolean - 一个计算结果为 true 或 false 的表达式。

*condition2* : Boolean - 一个计算结果为 true 或 false 的表达式。

返回

Boolean - 逻辑运算的结果。

另请参见

[|| 逻辑 OR 运算符](#), [| 按位 OR 运算符](#)

## % 模运算符

*expression1 % expression2*

计算 *expression1* 除以 *expression2* 的余数。如果有任一 *expression* 参数是非数字值，则模运算符 (%) 尝试将它们转换为数字。*expression* 可以是数字或转换为数值的字符串。

模运算结果的符号与被除数（第一个数字）的符号相匹配。例如，`-4 % 3` 和 `-4 % -3` 的计算结果都为 -1。

可用性：Flash Player 4；ActionScript 1.0

## 操作数

`expression1` : Number - 数字或计算结果为数字的表达式。

`expression2` : Number - 数字或计算结果为数字的表达式。

## 返回

Number - 算术运算的结果。

## 示例

下面的数值示例使用模运算符 (%)：

```
trace(12%5); // traces 2
trace(4.3%2.1); // traces 0.09999999999999996
trace(4%4); // traces 0
```

第一个 **trace** 返回 2，而不是 12/5 或 2.4，因为模运算符 (%) 仅返回余数。第二个 **trace** 返回 0.09999999999999996 而不是预期的 0.1，因为在二进制计算中对浮点数精度有限制。

## 另请参见

[/ 除法运算符](#)，[round \(Math.round 方法\)](#)

# %= 模赋值运算符

`expression1 %= expression2`

对 `expression1` 赋予 `expression1 % expression2` 的值。下面的两个语句是等效的：

```
x %= y;
x = x % y;
```

可用性：Flash Player 4；ActionScript 1.0

## 操作数

`expression1` : Number - 数字或计算结果为数字的表达式。

`expression2` : Number - 数字或计算结果为数字的表达式。

## 返回

Number - 算术运算的结果。

## 示例

下面的示例将值 4 赋予变量 `x`：

```
var x:Number = 14;
var y:Number = 5;
trace(x = y); // output: 4
```

另请参见

[% 模运算符](#)

## \* 乘法运算符

*expression1* \* *expression2*

将两个数值表达式相乘。如果两个表达式都是整数，则积为整数。如果其中任何一个或两个表达式是浮点数，则积为浮点数。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Number - 数字或计算结果为数字的表达式。

*expression2* : Number - 数字或计算结果为数字的表达式。

### 返回

Number - 一个整数或浮点数。

### 示例

用法 1：下面的语句将整数 2 与 3 相乘：

```
trace(2*3); // output: 6
```

结果 6 是整数。用法 2：此语句将浮点数 2.0 与 3.1416 相乘：

```
trace(2.0 * 3.1416); // output: 6.2832
```

结果 6.2832 是浮点数。

## \*= 乘法赋值运算符

*expression1* \*= *expression2*

对 *expression1* 赋予 *expression1* \* *expression2* 的值。例如，以下两个表达式是等效的：

```
x *= y;  
x = x * y
```

可用性：Flash Player 4；ActionScript 1.0

### 操作数

*expression1* : Number - 数字或计算结果为数字的表达式。

*expression2* : Number - 数字或计算结果为数字的表达式。

## 返回

`Number - expression1 * expression2` 的值。如果表达式不能转换为数值，则返回 NaN（非数字）。

## 示例

用法 1：下面的示例将值 50 赋予变量 x：

```
var x:Number = 5;
var y:Number = 10;
trace(x * y); // output: 50
```

用法 2：以下示例的第二行和第三行计算等号右侧的表达式，然后将结果赋予 x 和 y：

```
var i:Number = 5;
var x:Number = 4 - 6;
var y:Number = i + 2;
trace(x * y); // output: -14
```

另请参见

[\\* 乘法运算符](#)

## new 运算符

`new constructor()`

创建一个新的初始匿名对象，并调用由 `constructor` 参数标识的函数。`new` 运算符将括号中的任何可选参数和用关键字 `this` 引用的新建对象传递给函数。然后 `constructor` 函数可以用 `this` 来设置对象的变量。

可用性：Flash Player 5；ActionScript 1.0

## 操作数

`constructor`：Object - 一个函数，其后跟用括号括起来的任意可选参数。此函数通常是要构造的对象类型（如 `Array`、`Number` 或 `Object`）的名称。

## 示例

下面的示例创建 `Book()` 函数，然后使用 `new` 运算符来创建对象 `book1` 和 `book2`。

```
function Book(name, price){
    this.name = name;
    this.price = price;
}

book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```

下面的示例使用 `new` 运算符来创建具有 18 个元素的 `Array` 对象：

```
golfCourse_array = new Array(18);
```

另请参见

[\[\] 数组访问运算符](#)，[{} 对象初始值设定项运算符](#)

## ne 不等于（字符串）运算符

*expression1 ne expression2*

自 **Flash Player 5** 后不推荐使用。不推荐使用此运算符，而推荐使用 `!=`（inequality）运算符。

将 *expression1* 与 *expression2* 比较，如果 *expression1* 不等于 *expression2*，则返回 `true`；否则返回 `false`。

可用性：Flash Player 4；ActionScript 1.0

操作数

*expression1*：Object - 数字、字符串或变量。

*expression2*：Object - 数字、字符串或变量。

返回

Boolean - 如果 *expression1* 不等于 *expression2*，则返回 `true`；否则返回 `false`。

另请参见

[!= 不等于运算符](#)

## { } 对象初始值设定项运算符

```
object = { name1 : value1 , name2 : value2 ,... nameN : valueN }  
{expression1; [...expressionN]}
```

创建一个新对象，并用指定的 *name* 和 *value* 属性对初始化该对象。使用此运算符的效果与使用 `new Object` 语法并用赋值运算符填充属性对的效果相同。新建对象的原型通常命名为 **Object** 对象。

此运算符也用于标记与流控制语句（`for`、`while`、`if`、`else`、`switch`）和函数相关联的连续代码块。

可用性：Flash Player 5；ActionScript 1.0

## 操作数

`object` : `Object` - 要创建的对象。 `name1,2,...N` 属性名。 `value1,2,...N` 每个 `name` 属性的对应值。

## 返回

`Object` -

用法 1: 一个 `Object` 对象。

用法 2: 无, 除非函数具有一个显式的 `return` 语句, 在这种情况下, 返回类型在函数实现中指定。

## 示例

下面代码的第一行用对象初始值设定项 (`{}`) 运算符创建一个空对象; 第二行用构造函数创建一个新对象。

```
var object:Object = {};  
var object:Object = new Object();
```

下面的示例创建一个对象 `account`, 并用附带的值初始化属性 `name`、`address`、`city`、`state`、`zip` 和 `balance`:

```
var account:Object = {name:"Macromedia, Inc.", address:"600 Townsend  
    Street", city:"San Francisco", state:"California", zip:"94103",  
    balance:"1000"};  
for (i in account) {  
    trace("account." + i + " = " + account[i]);  
}
```

下面的示例说明数组和对象初始值设定项可以如何相互嵌套:

```
var person:Object = {name:"Gina Vechio", children:["Ruby", "Chickie",  
    "Puppa"]};
```

下面的示例使用上述示例中的信息, 并使用构造函数得到相同的结果:

```
var person:Object = new Object();  
person.name = "Gina Vechio";  
person.children = new Array();  
person.children[0] = "Ruby";  
person.children[1] = "Chickie";  
person.children[2] = "Puppa";
```

前面的 `ActionScript` 示例也可以按以下格式进行编写:

```
var person:Object = new Object();  
person.name = "Gina Vechio";  
person.children = new Array("Ruby", "Chickie", "Puppa");
```

## 另请参见

[Object](#)

## () 括号运算符

```
(expression1 [, expression2])  
( expression1, expression2 )  
function ( parameter1,..., parameterN )
```

对一个或多个参数执行分组运算，执行表达式的顺序计算，或者括住一个或多个参数并将它们作为参数传递给括号外的函数。

用法 1：控制表达式中运算符的执行顺序。括号覆盖正常的优先级顺序，从而导致先计算括号内的表达式。如果括号是嵌套的，则先计算最里面括号中的内容，然后计算较靠外括号中的内容。

用法 2：按顺序计算一系列用逗号分隔的表达式，并返回最终表达式的结果。

用法 3：括住一个或多个参数并将它们作为参数传递给括号外的函数。

可用性：Flash Player 4； ActionScript 1.0

### 操作数

expression1：Object - 数字、字符串、变量或文本。

expression2：Object - 数字、字符串、变量或文本。

function：Function - 要对括号中的内容执行的函数。

parameter1...parameterN：Object - 一系列参数，在将结果作为参数传递给括号外的函数之前执行这些参数。

### 示例

用法 1：下面的语句说明使用括号控制执行表达式的顺序（每个表达式的值都出现在“输出”面板中）：

```
trace((2 + 3)*(4 + 5)); // Output: 45  
trace((2 + 3) * (4 + 5)); // Output: 45  
trace(2 + (3 * (4 + 5))); // writes  
29  
trace(2 + (3 * (4 + 5))); // Output: 29  
trace(2+(3*4)+5); // writes 19  
trace(2 + (3 * 4) + 5); // Output: 19
```

用法 2：下面的示例计算函数 `foo()`，再计算函数 `bar()`，然后返回表达式 `a + b` 的结果：

```
var a:Number = 1;
var b:Number = 2;
function foo() { a += b; }
function bar() { b *= 10; }
trace((foo(), bar(), a + b)); // outputs 23
```

用法 3：下面的示例说明将括号与函数结合使用的方法：

```
var today:Date = new Date();
trace(today.getFullYear()); // traces current year
function traceParameter(param):Void { trace(param); }
traceParameter(2 * 2); //traces 4
```

另请参见

[with 语句](#)

## === 全等运算符

*expression1 === expression2*

测试两个表达式是否相等；除了不转换数据类型外，全等运算符 (`===`) 与等于运算符 (`==`) 执行运算的方式相同。如果两个表达式（包括它们的数据类型）相等，则结果为 `true`。

确定是否相等取决于参数的数据类型：

- 数字和布尔值按值进行比较，如果它们具有相同的值，则视为相等。
- 如果字符串表达式具有相同的字符数，而且这些字符都相同，则这些字符串表达式相等。
- 表示对象、数组和函数的变量按引用进行比较。如果两个变量引用同一个对象、数组或函数，则它们相等。而两个单独的数组即使具有相同数量的元素，也永远不会被视为相等。

可用性：Flash Player 6；ActionScript 1.0

### 操作数

*expression1* : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

*expression2* : Object - 数字、字符串、布尔值、变量、对象、数组或函数。

### 返回

Boolean - 比较的布尔结果。



## 示例

下面代码中的注释说明使用等于运算符和精确相等运算符的运算所返回的值:

```
// Both return true because no conversion is done
var string1:String = "5";
var string2:String = "5";
trace(string1 == string2); // true
trace(string1 === string2); // true
// Automatic data typing in this example converts 5 to "5"
var string1:String = "5";
var num:Number = 5;
trace(string1 == num); // true
trace(string1 === num); // false
// Automatic data typing in this example converts true to "1"
var string1:String = "1";
var bool1:Boolean = true;
trace(string1 == bool1); // true
trace(string1 === bool1); // false
// Automatic data typing in this example converts false to "0"
var string1:String = "0";
var bool2:Boolean = false;
trace(string1 == bool2); // true
trace(string1 === bool2); // false
```

下面的示例说明精确相等对为引用的变量和包含字面值的变量的处理方式有何不同。这是要一致地使用字符串文本以及要避免将 `new` 运算符与 **String** 类一起使用的一个原因。

```
// Create a string variable using a literal value
var str:String = "asdf";
// Create a variable that is a reference
var stringRef:String = new String("asdf");
// The equality operator does not distinguish among literals, variables,
// and references
trace(stringRef == "asdf"); // true
trace(stringRef == str); // true
trace("asdf" == str); // true
// The strict equality operator considers variables that are references
// distinct from literals and variables
trace(stringRef === "asdf"); // false
trace(stringRef === str); // false
```

## 另请参见

! 逻辑 NOT 运算符, != 不等于运算符, !== 不全等运算符, && 逻辑 AND 运算符, || 逻辑 OR 运算符, == 等于运算符

## !== 不全等运算符

*expression1* !== *expression2*

测试结果是否与全等运算符 (===) 正好相反。除了不转换数据类型外，不全等运算符执行的运算与不等于运算符相同。

如果 *expression1* 等于 *expression2*，并且它们的数据类型相等，则结果为 false。与全等运算符 (===) 相同，确定是否相等取决于所比较的数据类型，如以下列表中的说明：

- 数字、字符串和布尔值按值进行比较。
- 对象、数组和函数按引用进行比较。
- 变量根据其类型按值或按引用进行比较。

可用性：Flash Player 6；ActionScript 1.0

### 操作数

*expression1*：Object - 数字、字符串、布尔值、变量、对象、数组或函数。

*expression2*：Object - 数字、字符串、布尔值、变量、对象、数组或函数。

### 返回

Boolean - 比较的布尔结果。

### 示例

下面代码中的注释说明使用等于运算符 (==)、全等运算符 (===) 和不全等运算符 (!==) 的运算的返回值：

```
var s1:String = "5";
var s2:String = "5";
var s3:String = "Hello";
var n:Number = 5;
var b:Boolean = true;
trace(s1 == s2); // true
trace(s1 == s3); // false
trace(s1 == n); // true
trace(s1 == b); // false
trace(s1 === s2); // true
trace(s1 === s3); // false
trace(s1 === n); // false
trace(s1 === b); // false
trace(s1 !== s2); // false
trace(s1 !== s3); // true
trace(s1 !== n); // true
trace(s1 !== b); // true
```

### 另请参见

！ 逻辑 NOT 运算符， != 不等于运算符， && 逻辑 AND 运算符， || 逻辑 OR 运算符， == 等于运算符， === 全等运算符

## " 字符串分隔符运算符

`"text"`

如果用在字符之前和之后，则这些引号 (") 表示字符具有字面值；字符将被视作一个字符串，而不是一个变量、数值或其它 **ActionScript** 元素。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

`text` : `String` - 一系列（零个或多个）字符。

### 示例

下面的示例使用引号 (") 指示变量 *yourGuess* 的值是文本字符串 "Prince Edward Island"，而不是变量名。province 的值是一个变量，而不是文本；若要确定 province 的值，必须找到 *yourGuess* 的值。

```
var yourGuess:String = "Prince Edward Island";
submit_btn.onRelease = function() { trace(yourGuess); };
// displays Prince Edward Island
```

### 另请参见

[String](#), [String 函数](#)

## - 减法运算符

(Negation) `-expression`

(Subtraction) `expression1 - expression2`

用于执行求反或减法运算。

用法 1：用于执行求反时，它将数值 *expression* 的符号取反。用法 2：用于减法时，它对两个数值表达式执行算术减法运算，从 *expression1* 减去 *expression2*。两个表达式都为整数时，差为整数。其中任何一个或两个表达式为浮点数时，差为浮点数。

可用性：Flash Player 4；ActionScript 1.0

### 操作数

`expression1` : `Number` - 数字或计算结果为数字的表达式。

`expression2` : `Number` - 数字或计算结果为数字的表达式。

### 返回

`Number` - 一个整数或浮点数。

## 示例

用法 1：下面的语句将表达式  $2 + 3$  的符号反转：

```
trace(-(2+3)); // output: -5
```

用法 2：下面的语句从整数 5 中减去整数 2：

```
trace(5-2); // output: 3
```

结果 3 是整数。用法 3：下面的语句从浮点数 3.25 中减去浮点数 1.5：

```
trace(3.25-1.5); // output: 1.75
```

结果 1.75 是浮点数。

## -= 减法赋值运算符

*expression1 -= expression2*

对 *expression1* 赋予 *expression1* - *expression2* 的值。例如，下面两个语句是等效的：  
`x -= y`;  
`x = x - y`;

必须将字符串表达式转换为数字；否则返回 NaN（非数字）。

可用性：Flash Player 4；ActionScript 1.0

## 操作数

*expression1* : Number - 数字或计算结果为数字的表达式。

*expression2* : Number - 数字或计算结果为数字的表达式。

## 返回

Number - 算术运算的结果。

## 示例

下面的示例使用减法赋值运算符 (`-=`) 从 5 中减去 10，然后将结果赋予变量 `x`：

```
var x:Number = 5;  
var y:Number = 10;  
x -= y; trace(x); // output: -5
```

下面的示例说明如何将字符串转换为数字：

```
var x:String = "5";  
var y:String = "10";  
x -= y; trace(x); // output: -5
```

## 另请参见

- [减法运算符](#)

## :type 运算符

```
[ modifiers ] var variableName : type
function functionName () : type { ... }
function functionName ( parameter1:type , ... , parameterN:type ) [ :type ] { ... }
```

用于严格数据类型指定；此运算符指定变量类型、函数返回类型或函数参数类型。在变量声明或赋值中使用此运算符时，此运算符指定变量的类型；在函数声明或定义中使用此运算符时，此运算符指定函数的返回类型；在函数定义中与函数参数一起使用时，此运算符指定该参数预期的变量类型。

类型是仅限编译时的功能。所有类型都在编译时检查，并在出现不匹配的情况时生成错误。在使用点运算符 (.) 进行赋值运算、函数调用和类成员解除引用的过程中，可能出现不匹配情况。若要避免类型不匹配错误，请使用严格数据类型指定。

您可以使用的类型包括所有本机对象类型，您定义的类和接口以及 **Function** 和 **Void**。可识别的本机类型有 **Boolean**、**Number** 和 **String**。所有内置类都被视为本机类型，因此也受支持。

可用性：Flash Player 6；ActionScript 1.0

### 操作数

**variableName**：Object - 变量的标识符。**type**：本机数据类型、您已定义的类名称或者接口名称。**functionName**：函数的标识符。**parameter**：函数参数的标识符。

### 示例

用法 1：下面的示例声明一个类型为 **String** 的名为 **userName** 的公共变量，并且为该变量分配一个空字符串：

```
var userName:String = "";
```

用法 2：下面的示例说明如何通过定义名为 **randomInt()** 的函数（它采用名为 **integer** 且类型为 **Number** 的参数）来指定函数的参数类型：

```
function randomInt(integer:Number):Number {
    return Math.round(Math.random()*integer);
}
trace(randomInt(8));
```

用法 3：下面的示例定义一个名为 **squareRoot()** 的函数，该函数采用名为 **val** 且类型为 **Number** 的参数，返回 **val** 的平方根（也为 **Number** 类型）：

```
function squareRoot(val:Number):Number {
    return Math.sqrt(val);
}
trace(squareRoot(121));
```

另请参见

[var 语句](#)、[function 语句](#)

# typeof 运算符

`typeof(expression)`

计算 `expression` 并返回一个字符串，该字符串指定该表达式为字符串、影片剪辑、对象、函数、数字或布尔值。

可用性：Flash Player 5 ； ActionScript 1.0

## 操作数

`expression` : `Object` - 一个字符串、影片剪辑、按钮、对象或函数。

## 返回

`String` - `expression` 类型的一个 `String` 表示形式。下表说明对各个类型的 `expression` 使用 `typeof` 运算符的结果。

表达式类型	结果
String	string
影片剪辑	movieclip
Button	object
文本字段	object
Number	number
Boolean	boolean
Object	object
Function	function

另请参见

[instanceof 运算符](#)

# void 运算符

`void expression`

`void` 运算符计算表达式，然后放弃其值，返回 `undefined`。`void` 运算符通常在通过 `==` 运算符测试未定义值的比较中使用。

可用性：Flash Player 5 ； ActionScript 1.0

## 操作数

`expression` : `Object` - 要计算的表达式。

# 语句

语句是执行或指定动作的语言元素。例如，`return` 语句返回一个结果，作为该语句在其中执行的函数的值。`if` 语句对一个条件求值，以确定采取的下一个动作。`switch` 语句创建 `ActionScript` 语句的分支结构。

## 语句摘要

语句	说明
<code>break</code>	出现在循环（ <code>for</code> 、 <code>for..in</code> 、 <code>do..while</code> 或 <code>while</code> ）内或与 <code>switch</code> 语句中的特定情况相关的语句块内。
<code>case</code>	定义 <code>switch</code> 语句的条件。
<code>class</code>	定义一个自定义类，通过该自定义类可以实例化共享您定义的方法和属性的对象。
<code>continue</code>	跳过最里层循环中所有其余的语句并开始循环的下一个迭代，就像控制正常传递到了循环结尾一样。
<code>default</code>	定义 <code>switch</code> 语句的默认情况。
<code>delete</code>	破坏由 <i>reference</i> 参数指定的对象引用，如果成功删除了引用，则返回 <code>true</code> ；否则返回 <code>false</code> 。
<code>do..while</code>	与 <code>while</code> 循环类似，不同之处是在对条件进行初始计算前执行一次语句。
<code>dynamic</code>	指明基于所指定类的对象可以在运行时添加和访问动态属性。
<code>else</code>	指定当 <code>if</code> 语句中的条件返回 <code>false</code> 时要运行的语句。
<code>else if</code>	计算条件，并指定当初始 <code>if</code> 语句中的条件返回 <code>false</code> 时要运行的语句。
<code>extends</code>	定义一个类，它是另一个类（超类）的子类。
<code>for</code>	计算一次 <i>init</i> （初始化）表达式，然后开始一个循环序列。
<code>for..in</code>	迭代对象的属性或数组中的元素，并对每个属性或元素执行 <b>statement</b> 。

语句	说明
<code>function</code>	包含为执行特定任务而定义的一组语句。
<code>get</code>	允许隐式获取 与某些对象关联的属性， 这些对象基于外部类文件中定义的类。
<code>if</code>	对条件进行计算以确定 SWF 文件中的下一个动作。
<code>implements</code>	指定类必须定义所实现的接口中声明的所有方法。
<code>import</code>	使您不必指定类的完全限定名就可以访问类。
<code>interface</code>	定义接口。
<code>intrinsic</code>	允许对以前定义的类执行编译时类型检查。
<code>private</code>	指定变量或函数只对声明或定义该变量或函数的类或该类的子类可用。
<code>public</code>	指定变量或函数对任何调用者都可用。
<code>return</code>	指定由函数返回的值。
<code>set</code>	允许隐式设置与某些对象关联的属性， 这些对象基于外部类文件中定义的类。
<code>set variable</code>	为变量赋值。
<code>static</code>	指定某个变量或函数只为每个类创建一次， 而不是在基于该类的每个对象中都创建。
<code>super</code>	调用方法或构造函数的超类版本。
<code>switch</code>	创建 <code>ActionScript</code> 语句的分支结构。
<code>throw</code>	生成或抛出 （通过 <code>throw</code> 语句） 一个可由 <code>catch{} </code> 代码块处理或捕获的错误。
<code>try..catch..finally</code>	包含一个代码块， 如果该代码块内发生错误， 将对错误进行响应。
<code>var</code>	用于声明局部变量或时间轴变量。
<code>while</code>	计算条件， 如果条件计算结果为 <code>true</code> ， 则在循环返回以再次计算条件之前执行一条语句或一系列语句。
<code>with</code>	允许您使用 <i>object</i> 参数指定一个对象 （例如影片剪辑）， 并使用 <i>statement(s)</i> 参数计算该对象内的表达式和动作。



# break 语句

break

出现在循环（for、for..in、do..while 或 while）内或与 switch 语句中的特定情况相关的语句块内。当在循环中使用时，break 语句指示 **Flash** 跳过循环体的其余部分，停止循环动作，并执行循环语句后面的语句。当在 switch 中使用时，break 语句指示 **Flash** 跳过此 case 块中的其余语句，并跳到包含它的 switch 语句后面的第一个语句。

在嵌套循环中，break 语句只跳过当前循环的其余部分，而不是跳出整个系列的嵌套循环。关于跳出整个系列的嵌套循环，请参见 try..catch..finally。

可用性：Flash Player 4；ActionScript 1.0

## 示例

下面的示例使用 break 语句来退出一个循环（如果没有该语句，则该循环为无限循环）：

```
var i:Number = 0;
while (true) {
    trace(i);
    if (i >= 10) {
        break; // this will terminate/exit the loop
    }
    i++;
}
```

它跟踪以下输出：

```
0
1
2
3
4
5
6
7
8
9
10
```

另请参见

[for 语句](#)

## case 语句

`case expression : statement(s)`

定义 switch 语句的条件。如果 *expression* 参数全等于 (`===`) switch 语句的 *expression* 参数, 则 **Flash Player** 将执行 *statement(s)* 中的语句, 直到遇到一个 break 语句或直到 switch 语句结束。

如果在 switch 语句外部使用 case 语句, 会产生错误, 而且脚本不能编译。

**注意:** 应始终用 **break** 语句来结束 *statement(s)* 参数。如果从 *statement(s)* 参数中省略 break statement, 它将继续执行下一个 case 语句, 而不是退出 switch 语句。

**可用性:** Flash Player 4 ; ActionScript 1.0

### 参数

*expression*:String - 任何表达式。

### 示例

下面的示例为 **switch** 语句 thisMonth 定义条件。如果 thisMonth 等于 **case** 语句中的表达式, 则执行该语句。

```
var thisMonth:Number = new Date().getMonth();
switch (thisMonth) {
    case 0 :
        trace("January");
        break;
    case 1 :
        trace("February");
        break;
    case 5 :
    case 6 :
    case 7 :
        trace("Some summer month");
        break;
    case 8 :
        trace("September");
        break;
    default :
        trace("some other month");
}
```

另请参见

[break 语句](#)

## class 语句

```
[dynamic] class className [ extends superClass ] [ implements interfaceName[,  
    interfaceName... ] ] {  
    // class definition here  
}
```

定义一个自定义类，通过该自定义类可以实例化共享您定义的方法和属性的对象。例如，如果您正开发一个发票跟踪系统，则可以创建一个发票类，它定义每一发票应具有的所有方法和属性。然后您可以使用 `new invoice()` 命令来创建发票对象。

该类的名称必须与包含该类的外部文件的名称匹配。外部文件名称必须是类的名称加上文件扩展名 `.as`。例如，如果您将一个类命名为 **Student**，则定义该类的文件必须被命名为 **Student.as**。

如果类在包中，类声明必须用形式为 `base.sub1.sub2.MyClass` 的完全限定的类名称。同时，该类的 **AS** 文件必须以反映该包结构的目录结构存储在路径中，例如 `base/sub1/sub2/MyClass.as`。如果类定义的形式为“`class MyClass`”，则它位于默认包中，并且 `MyClass.as` 文件应位于路径中某个目录的顶级目录中。

因此，最好在开始创建类之前就计划好您的目录结构。否则，如果您在创建类文件之后决定移动它们，就将不得不修改类声明语句以反映其新位置。

您不能嵌套类定义；即，不能在一个类定义内定义其它类。

若要指示对象可在运行时添加和访问动态属性，请在 **class** 语句前面放置 `dynamic` 关键字。若要声明一个类实现一个接口，请使用 `implements` 关键字。若要创建一个类的子类，请使用 `extends` 关键字。（某一类只能扩展一个类，但可以实现多个接口。）您可以在一个语句中使用 `implements` 和 `extends`。下面的示例将说明 `implements` 和 `extends` 关键字的典型用法：

```
class C implements Interface_i, Interface_j // OK  
class C extends Class_d implements Interface_i, Interface_j // OK  
class C extends Class_d, Class_e // not OK
```

可用性：Flash Player 6；ActionScript 2.0

### 参数

**className:String** - 类的完全限定名。

## 示例

下面的示例创建一个名为 **Plant** 的类。**Plant** 构造函数采用两个参数。

```
// Filename Plant.as
class Plant {
    // Define property names and types
    var leafType:String;
    var bloomSeason:String;
    // Following line is constructor
    // because it has the same name as the class
    function Plant(param_leafType:String, param_bloomSeason:String) {
        // Assign passed values to properties when new Plant object is created
        this.leafType = param_leafType;
        this.bloomSeason = param_bloomSeason;
    }
    // Create methods to return property values, because best practice
    // recommends against directly referencing a property of a class
    function getLeafType():String {
        return leafType;
    }
    function getBloomSeason():String {
        return bloomSeason;
    }
}
```

在外部脚本文件或“动作”面板中，使用 **new** 运算符创建一个 **Plant** 对象。

```
var pineTree:Plant = new Plant("Evergreen", "N/A");
// Confirm parameters were passed correctly
trace(pineTree.getLeafType());
trace(pineTree.getBloomSeason());
```

下面的示例创建一个名为 **ImageLoader** 的类。**ImageLoader** 构造函数采用三个参数。

```
// Filename ImageLoader.as
class ImageLoader extends MovieClip {
    function ImageLoader(image:String, target_mc:MovieClip, init:Object) {
        var listenerObject:Object = new Object();
        listenerObject.onLoadInit = function(target) {
            for (var i in init) {
                target[i] = init[i];
            }
        };
        var JPEG_mc1:MovieClipLoader = new MovieClipLoader();
        JPEG_mc1.addListener(listenerObject);
        JPEG_mc1.loadClip(image, target_mc);
    }
}
```

在外部脚本文件或“动作”面板中，使用 `new` 运算符创建一个 **ImageLoader** 对象。

```
var jakob_mc:MovieClip = this.createEmptyMovieClip("jakob_mc",
    this.getNextHighestDepth());
var jakob:ImageLoader = new ImageLoader("http://www.helpexamples.com/flash/
    images/image1.jpg", jakob_mc, {_x:10, _y:10, _alpha:70, _rotation:-5});
```

另请参见

[dynamic 语句](#)

## continue 语句

`continue`

跳过最里层循环中所有其余的语句并开始循环的下一个迭代，就像控制正常传递到了循环结尾一样。在循环外无效。

可用性：Flash Player 4；ActionScript 1.0

### 示例

在下面的 `while` 循环中，`continue` 使 **Flash** 解释程序跳过循环体的其余部分，并转到循环的顶端（在该处进行条件测试）：

```
trace("example 1");
var i:Number = 0;
while (i < 10) {
    if (i % 3 == 0) {
        i++;
        continue;
    }
    trace(i);
    i++;
}
```

在下面的 `do..while` 循环中，`continue` 使 **Flash** 解释程序跳过循环体的其余部分，并转到循环的底端（在该处进行条件测试）：

```
trace("example 2");
var i:Number = 0;
do {
    if (i % 3 == 0) {
        i++;
        continue;
    }
    trace(i);
    i++;
}
while (i < 10);
```

在 for 循环中，continue 使 Flash 解释程序跳过循环体的其余部分。在下面的示例中，如果 i 以 3 为模等于 0，则跳过 trace(i) 语句：

```
trace("example 3");
for (var i = 0; i < 10; i++) {
    if (i % 3 == 0) {
        continue;
    }
    trace(i);
}
```

在下面的 for..in 循环中，continue 使 Flash 解释程序跳过循环体的其余部分，并跳回循环的顶端（在该处处理枚举中的下一个值）：

```
for (i in _root) {
    if (i == "$version") {
        continue;
    }
    trace(i);
}
```

另请参见

[loadMovie \(MovieClip.loadMovie 方法\)](#)

## default 语句

default: *statements*

定义 switch 语句的默认情况。如果 switch 语句的 *expression* 参数不等于（使用全等运算符 [==]）给定 switch 语句的 case 关键字后的任何 *expression* 参数，则执行这些语句。

switch 不要求一定有 default case 语句。default case 语句也不一定要出现在列表的最后。如果在 switch 语句外使用 default 语句，将会产生错误，并且脚本不能编译。

可用性：Flash Player 6； ActionScript 1.0

### 参数

statements:String - 任何语句。

## 示例

在下面的示例中，表达式 **A** 与表达式 **B** 或 **D** 不相等，所以运行 `default` 关键字后面的语句，并将 `trace()` 语句发送到“输出”面板。

```
var dayOfWeek:Number = new Date().getDay();
switch (dayOfWeek) {
    case 1 :
        trace("Monday");
        break;
    case 2 :
        trace("Tuesday");
        break;
    case 3 :
        trace("Wednesday");
        break;
    case 4 :
        trace("Thursday");
        break;
    case 5 :
        trace("Friday");
        break;
    default :
        trace("Weekend");
}
```

另请参见

[switch 语句](#)

## delete 语句

`delete reference`

破坏由 *reference* 参数指定的对象引用，如果成功删除了引用，则返回 `true`；否则返回 `false`。该运算符对于释放脚本所用的内存非常有用。可以用 `delete` 运算符删除对对象的引用。删除所有对对象的引用后，**Flash Player** 会删除该对象并释放由该对象使用的内存。

虽然 `delete` 是一个运算符，但它通常作为语句使用，如下示例所示：

```
delete x;
```

如果 *reference* 参数不存在或无法删除，`delete` 运算符将失败并返回 `false`。您不能删除预定义的对象和属性，也不能删除使用 `var` 语句在函数内声明的变量。不能使用 `delete` 运算符删除影片剪辑。

可用性：Flash Player 5；ActionScript 1.0

## 返回

Boolean - 一个布尔值。

## 参数

*reference:Object* - 要消除的变量或对象的名称。

## 示例

用法 1: 下面的示例创建一个对象并使用它, 然后在不再需要时删除它:

```
var account:Object = new Object();
account.name = "Jon";
account.balance = 10000;
trace(account.name); //output: Jon
delete account;
trace(account.name); //output: undefined
```

用法 2: 下面的示例删除对象的一个属性:

```
// create the new object "account"
var account:Object = new Object();
// assign property name to the account
account.name = "Jon";
// delete the property
delete account.name;
```

用法 3: 下面的示例删除一个对象属性:

```
var my_array:Array = new Array();
my_array[0] = "abc"; // my_array.length == 1
my_array[1] = "def"; // my_array.length == 2
my_array[2] = "ghi"; // my_array.length == 3
// my_array[2] is deleted, but Array.length is not changed
delete my_array[2];
trace(my_array.length); // output: 3
trace(my_array); // output: abc,def,undefined
```

用法 4: 下面的示例说明 delete 在对象引用上的行为:

```
var ref1:Object = new Object();
ref1.name = "Jody";
// copy the reference variable into a new variable
// and delete ref1
ref2 = ref1;
delete ref1;
trace("ref1.name "+ref1.name); //output: ref1.name undefined
trace("ref2.name "+ref2.name); //output: ref2.name Jody
```

如果 ref1 尚未复制到 ref2 中, 则当删除了 ref1 时, 该对象将被删除, 因为将没有对它的引用。如果您删除 ref2, 则没有对该对象的引用; 它将被破坏, 它所用的内存变为可用。

另请参见

[var 语句](#)



## do..while 语句

```
do { statement(s) } while (condition)
```

与 while 循环类似，不同之处是在对条件进行初始计算前执行一次语句。随后，仅当条件计算结果是 true 时执行语句。

do..while 循环确保循环内的代码至少执行一次。尽管这也可以用 while 循环通过在 while 循环开始前放一段要执行的语句副本来实现，但很多程序员认为 do..while 循环更易于阅读。

如果条件计算结果始终为 true，do..while 就会无限循环。如果进入了无限循环，则 **Flash Player** 会遇到问题，最终会发出警告信息，或播放器崩溃。如果知道要循环的次数，应该尽可能使用 for 循环。尽管 for 循环易于阅读和调试，但不是在所有情况下都能代替 do..while 循环。

可用性：Flash Player 4； ActionScript 1.0

### 参数

**condition:** Boolean - 要计算的条件。只要 *condition* 参数的计算结果为 true，就会执行 do 代码块内的 *statement(s)*。

### 示例

下面的示例使用 do..while 循环计算一个条件是否为 true，并一直跟踪 myVar，直到 myVar 大于 5。当 myVar 大于 5 时，循环将结束。

```
var myVar:Number = 0;
do {
    trace(myVar);
    myVar++;
}
while (myVar < 5);
/* output:
0
1
2
3
4
*/
```

另请参见

[break 语句](#)

## dynamic 语句

```
dynamic class className [ extends superClass ] [ implements interfaceName[,
    interfaceName... ] ] {
    // class definition here
}
```

指明基于所指定类的对象可以在运行时添加和访问动态属性。

对于动态类的类型检查比对于非动态类的类型检查更为宽松，因为在类定义内访问的成员和在类实例上访问的成员不与在类范围内定义的那些成员进行比较。但是，仍可能对类成员函数进行类型检查，以确定返回类型和参数类型。当处理 **MovieClip** 对象时，此行为特别有用，因为这种情况下可以通过多种不同的方法动态地将属性和对象添加到影片剪辑，如 `MovieClip.createEmptyMovieClip()` 和 `MovieClip.createTextField()`。

动态类的子类也是动态的。

确保在声明对象时指定类型，如下所示：

```
var x:MyClass = new MyClass();
```

如果在声明对象时没有指定类型（如下所示），则该对象被视为动态的：

```
var x = new MyClass();
```

可用性：Flash Player 6；ActionScript 2.0

### 示例

在下面的示例中，类 `Person2` 尚未标记为动态的，因此对它调用未声明的函数将在编译时生成错误：

```
class Person2 {
    var name:String;
    var age:Number;
    function Person2(param_name:String, param_age:Number) {
        trace ("anything");
        this.name = param_name;
        this.age = param_age;
    }
}
```

在位于同一目录的 **FLA** 或 **AS** 文件中，将以下 **ActionScript** 添加到时间轴上的第 1 帧：

```
// Before dynamic is added
var craig:Person2 = new Person2("Craiggers", 32);
for (i in craig) {
    trace("craig." + i + " = " + craig[i]);
}
/* output:
craig.age = 32
craig.name = Craiggers */
```

如果添加未声明的函数 `dance`，则生成一个错误，如下例所示：

```
trace("");
craig.dance = true;
for (i in craig) {
    trace("craig." + i + " = " + craig[i]);
}
/* output: **Error** Scene=Scene 1, layer=Layer 1, frame=1:Line 14: There is
no property with the name 'dance'. craig.dance = true; Total ActionScript
Errors: 1 Reported Errors: 1 */
```

将关键字 `dynamic` 添加到类 **Person2**，以便第一行如下所示：

```
dynamic class Person2 {
```

再次测试代码，您就会看到以下输出：

```
craig.dance = true craig.age = 32 craig.name = Craiggers
```

另请参见

[class 语句](#)

## else 语句

```
if (condition){
    statement(s);
} else {
    statement(s);
}
```

指定当 `if` 语句中的条件返回 `false` 时要运行的语句。如果将只执行一条语句，用来括起要由 `else` 语句执行的语句块的花括号 `{}` 是不必要的。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**condition:** Boolean - 计算结果为 `true` 或 `false` 的表达式。

### 示例

在下面的示例中，`else` 条件用于检查 `age_txt` 变量是大于还是小于 18：

```
if (age_txt.text>=18) {
    trace("welcome, user");
}
else {
    trace("sorry, junior");
    userObject.minor = true;
    userObject.accessAllowed = false;
}
```

在下面的示例中，花括号 ({} ) 是不必要的，因为 else 语句后面只有一条语句：

```
if (age_txt.text>18) { trace("welcome, user"); } else trace("sorry, junior");
```

另请参见

[if 语句](#)

## else if 语句

```
if(condition) {  
    statement(s);  
} else if(condition) {  
    statement(s);  
}
```

计算条件，并指定当初始 if 语句中的条件返回 false 时要运行的语句。如果 else if 条件返回 true，则 **Flash** 解释程序运行该条件后面花括号 ({} ) 中的语句。如果 else if 条件为 false，则 **Flash** 将跳过花括号内的语句，而运行花括号后面的语句。

使用 else if 语句可在脚本中创建分支逻辑。如果有多个分支，应该考虑使用 switch 语句。

**可用性：**Flash Player 4；ActionScript 1.0

参数

**condition:**Boolean - 计算结果为 true 或 false 的表达式。

示例

下面的示例使用 else if 语句将 score\_txt 与指定的值进行比较：

```
if (score_txt.text>90) {  
    trace("A");  
}  
else if (score_txt.text>75) {  
    trace("B");  
}  
else if (score_txt.text>60) {  
    trace("C");  
}  
else {  
    trace("F");  
}
```

另请参见

[if 语句](#)

## extends 语句

```
class className extends otherClassName {}  
interface interfaceName extends otherInterfaceName {}
```

定义一个类，它是另一个类（超类）的子类。子类继承超类中定义的所有方法、属性、函数等。

接口也可以使用 extends 关键字扩展。扩展另一个接口的接口包括原接口的所有方法声明。

可用性：Flash Player 6； ActionScript 2.0

### 参数

**className:String** - 您所定义的类的名称。

### 示例

在下面的示例中，**Car** 类扩展 **Vehicle** 类，以便继承它的所有方法、属性和函数。如果您的脚本对 **Car** 对象进行实例化，则来自 **Car** 类的方法和来自 **Vehicle** 类的方法都可以使用。

下面的示例显示名为 **Vehicle.as** 的文件（它定义 **Vehicle** 类）的内容：

```
class Vehicle {  
    var numDoors:Number;  
    var color:String;  
    function Vehicle(param_numDoors:Number, param_color:String) {  
        this.numDoors = param_numDoors;  
        this.color = param_color;  
    }  
    function start():Void {  
        trace("[Vehicle] start");  
    }  
    function stop():Void {  
        trace("[Vehicle] stop");  
    }  
    function reverse():Void {  
        trace("[Vehicle] reverse");  
    }  
}
```

下面的示例显示同一目录中名为 **Car.as** 的第二个 AS 文件。此类扩展 **Vehicle** 类，通过三种方式修改它。第一种是 **Car** 类添加变量 **fullSizeSpare** 以跟踪 **car** 对象是否具有标准尺寸的备用轮胎。第二种是它添加特定于汽车的新方法 **activateCarAlarm()**，该方法用于激活汽车的防盗警报。第三种是它覆盖 **stop()** 函数以添加 **Car** 类使用防抱死制动系统来停车的事实。

```
class Car extends Vehicle {  
    var fullSizeSpare:Boolean;  
    function Car(param_numDoors:Number, param_color:String,  
        param_fullSizeSpare:Boolean) {  
        this.numDoors = param_numDoors;  
    }  
}
```

```

this.color = param_color;
this.fullSizeSpare = param_fullSizeSpare;
}
function activateCarAlarm():Void {
trace("[Car] activateCarAlarm");
}
function stop():Void {
trace("[Car] stop with anti-lock brakes");
}
}

```

下面的示例对 **Car** 对象进行实例化，调用在 **Vehicle** 类中定义的方法 (`start()`)，然后调用由 **Car** 类覆盖的方法 (`stop()`)，最后从 **Car** 类调用一个方法 (`activateCarAlarm()`)：

```

var myNewCar:Car = new Car(2, "Red", true);
myNewCar.start(); // output: [Vehicle] start
myNewCar.stop(); // output: [Car] stop with anti-lock brakes
myNewCar.activateCarAlarm(); // output: [Car] activateCarAlarm

```

**Vehicle** 类的子类也可以使用关键字 `super` 进行编写，该子类可以使用该关键字访问超类的属性和方法。下面的示例显示第三个 **AS** 文件，该文件名为 **Truck.as**，也在同一目录中。**Truck** 类在构造函数中使用 `super` 关键字，并再次在被覆盖的 `reverse()` 函数中使用该关键字。

```

class Truck extends Vehicle {
    var numWheels:Number;
    function Truck(param_numDoors:Number, param_color:String,
        param_numWheels:Number) {
        super(param_numDoors, param_color);
        this.numWheels = param_numWheels;
    }
    function reverse():Void {
        beep();
        super.reverse();
    }
    function beep():Void {
        trace("[Truck] make beeping sound");
    }
}

```

下面的示例对 **Truck** 对象进行实例化，调用由 **Truck** 类覆盖的方法 (`reverse()`)，然后调用在 **Vehicle** 类中定义的方法 (`stop()`)：

```

var myTruck:Truck = new Truck(2, "White", 18);
myTruck.reverse(); // output: [Truck] make beeping sound [Vehicle] reverse
myTruck.stop(); // output: [Vehicle] stop

```

另请参见

[class 语句](#)

## for 语句

```
for(init; condition; next) {  
    statement(s);  
}
```

计算一次 *init* (初始化) 表达式, 然后开始一个循环序列。循环序列从计算 *condition* 表达式开始。如果 *condition* 表达式的计算结果为 **true**, 将执行 *statement* 并计算 *next* 表达式。然后循环序列再次从计算 *condition* 表达式开始。

如果将只执行一条语句, 用于括起将由 for 语句执行的语句的花括号 ({} ) 可以删除。

可用性: Flash Player 5 ; ActionScript 1.0

### 参数

**init** - 在开始循环序列前要计算的表达式, 通常为赋值表达式。还允许对此参数使用 var 语句。

### 示例

下面的示例使用 for 在数组中添加元素:

```
var my_array:Array = new Array();  
for (var i:Number = 0; i < 10; i++) {  
    my_array[i] = (i + 5) * 10;  
}  
trace(my_array); // output: 50,60,70,80,90,100,110,120,130,140
```

下面的示例使用 for 重复执行相同的动作。在代码中, for 循环将从 1 到 100 的数字相加。

```
var sum:Number = 0;  
for (var i:Number = 1; i <= 100; i++) {  
    sum += i;  
}  
trace(sum); // output: 5050
```

下面的示例说明在将仅执行一条语句时花括号 ({} ) 是不必要的:

```
var sum:Number = 0;  
for (var i:Number = 1; i <= 100; i++)  
    sum += i;  
trace(sum); // output: 5050
```

### 另请参见

[++ 递增运算符](#)

## for..in 语句

```
for (variableIterant in object) {  
    statement(s);  
}
```

迭代对象的属性或数组中的元素，并对每个属性或元素执行 **statement**。对象的方法不能由 `for..in` 动作来枚举。

有些属性不能由 `for..in` 动作来枚举。例如，不能枚举影片剪辑属性，例如 `_x` 和 `_y`。在外部类文件中，和实例成员不同的是，静态成员是不能枚举的。

`for..in` 语句迭代所迭代对象的原型链中对象的属性。首先枚举该对象的属性，接着枚举其直接原型的属性，然后枚举该原型的原型的属性，依次类推。`for..in` 语句不会将相同的属性名枚举两次。如果 `child` 对象有原型 `parent`，并且两者都包含属性 `prop`，则对 `child` 调用的 `for..in` 语句将枚举来自 `child` 的 `prop`，而忽略 `parent` 对象中的该属性。

如果只执行一条语句，用来括起要由 `for..in` 语句执行的语句块的花括号 (`{}`) 是不必要的。

如果在一个类文件（外部 **AS** 文件）中编写一个 `for..in` 循环，则实例成员对于该循环不可用，而静态成员则可用。然而，如果在一个 **FLA** 文件中为类的实例编写一个 `for..in` 循环，则实例成员在循环中可用，而静态成员不可用。

可用性：Flash Player 5；ActionScript 1.0

### 参数

**variableIterant:String** - 要作为迭代变量的变量的名称，迭代变量引用对象的每个属性或数组中的每个元素。

### 示例

下面的示例使用 `for..in` 迭代对象的属性：

```
var myObject:Object = {firstName:"Tara", age:27, city:"San Francisco"};  
for (var prop in myObject) {  
    trace("myObject."+prop+" = "+myObject[prop]);  
}  
//output  
myObject.firstName = Tara  
myObject.age = 27  
myObject.city = San Francisco
```

下面的示例使用 `for..in` 迭代数组的元素：

```
var myArray:Array = new Array("one", "two", "three");  
for (var index in myArray)  
    trace("myArray["+index+"] = " + myArray[index]);  
// output:  
myArray[2] = three  
myArray[1] = two  
myArray[0] = one
```



下面的示例将 `typeof` 运算符与 `for...in` 结合使用以迭代特定类型的子级：

```
for (var name in this) {  
    if (typeof (this[name]) == "movieclip") {  
        trace("I have a movie clip child named "+name);  
    }  
}
```



如果您有多个影片剪辑，则输出由这些剪辑的实例名构成。

下面的示例枚举影片剪辑的子级，并将每个子级发送到其各自时间轴的第 2 帧。

RadioButtonGroup 影片剪辑是具有以下三个子级的父级：\_RedRadioButton\_、\_GreenRadioButton\_ 和 \_BlueRadioButton\_。

```
for (var name in RadioButtonGroup) { RadioButtonGroup[name].gotoAndStop(2);  
}
```

## function 语句

Usage 1: (Declares a named function.)

```
function functionname( [parameter0, parameter1, ..., parameterN]){  
    statement(s)  
}
```

Usage 2: (Declares an anonymous function and returns a reference to it.)

```
function ([parameter0, parameter1, ..., parameterN]){  
    statement(s)  
}
```

包含为执行特定任务而定义的一组语句。可以在 SWF 文件的一个地方定义函数，然后从 SWF 文件的其它脚本中调用它。定义函数时，还可以为其指定参数。参数是函数要对其进行操作的值的占位符。可以在每次调用一个函数时传递不同的参数，这样就可不同的情况下重用函数。

在一个函数的 *statement(s)* 中使用 `return` 语句，以使函数生成或返回一个值。

您可以使用此语句定义一个 `function`，该函数具有指定的 *functionname*、*parameters* 和 *statement(s)*。当脚本调用函数时，就会执行函数定义中的语句。允许提前引用：在同一脚本中，函数可以先调用后声明。一个函数定义会替换同一函数以前的任何定义。只要是允许使用语句的地方就可使用此语法。

可以使用该函数语句来创建一个匿名函数并返回对它的引用。此语法用于表达式中，对于在对象中安装方法尤其有用。

若需要其它功能，可以在函数定义中使用 `arguments` 对象。`arguments` 对象的一些常见用法是创建一个接受参数数量可变的函数，以及创建一个递归的匿名函数。

可用性：Flash Player 5；ActionScript 1.0

## 返回

String - 用法 1: 不返回任何内容的声明形式。用法 2: 匿名函数的参考。

## 参数

**functionname**:String - 声明的函数的名称。

## 示例

下面的示例定义函数 `sqr`，该函数接受一个参数并返回该参数的 `Math.pow(x, 2)`：

```
function sqr(x:Number) {  
    return Math.pow(x, 2);  
}  
var y:Number = sqr(3);  
trace(y); // output: 9
```

如果在同一脚本中定义和使用该函数，则可以先使用该函数，后定义它：

```
var y:Number = sqr(3);  
trace(y); // output: 9  
function sqr(x:Number) {  
    return Math.pow(x, 2);  
}
```

下面的函数创建一个 **LoadVars** 对象，并将 **params.txt** 加载到 SWF 文件中。当文件成功加载时，**variables loaded** 将跟踪：

```
var myLV:LoadVars = new LoadVars();  
myLV.load("params.txt");  
myLV.onLoad = function(success:Boolean) {  
    trace("variables loaded");  
}
```

## get 语句

```
function get property () {  
    // your statements here  
}
```

允许隐式获取与某些对象关联的属性，这些对象基于外部类文件中定义的类。通过使用隐式获取方法，您可以不必直接访问属性就访问对象的属性。隐式 **get/set** 方法是对 **ActionScript 1.0** 中 `Object.addProperty()` 方法的句法简化。

可用性：Flash Player 6；ActionScript 2.0

## 参数

**property**:String - 用来引用 `get` 访问的属性的词；该值必须与相应的 `set` 命令中使用的值相同。

## 示例

在下面的示例中，定义一个 **Team** 类。**Team** 类包括用于在该类内检索和设置属性的 **get/set** 方法：

```
class Team {
    var teamName:String;
    var teamCode:String;
    var teamPlayers:Array = new Array();
    function Team(param_name:String, param_code:String) {
        this.teamName = param_name;
        this.teamCode = param_code;
    }
    function get name():String {
        return this.teamName;
    }
    function set name(param_name:String):Void {
        this.teamName = param_name;
    }
}
```

在时间轴上的帧中输入以下 **ActionScript**：

```
var giants:Team = new Team("San Fran", "SF0");
trace(giants.name);
giants.name = "San Francisco";
trace(giants.name);
/* output:
San Fran San Francisco */
```

在跟踪 **giants.name** 时，要使用 **get** 方法来返回属性的值。

另请参见

[addProperty](#) ([Object.addProperty](#) 方法)

## if 语句

```
if(condition) {  
    statement(s);  
}
```

对条件进行计算以确定 SWF 文件中的下一个动作。如果条件为 true，则 **Flash** 将运行条件后面花括号 ({} ) 内的语句。如果条件为 false，则 **Flash** 将跳过花括号内的语句，而运行花括号后面的语句。将 if 语句与 else 和 else if 语句一起使用，以在脚本中创建分支逻辑。

如果将只执行一条语句，用来括起要由 if 语句执行的语句块的花括号 ({} ) 是不必要的。

可用性: Flash Player 4 ; ActionScript 1.0

### 参数

**condition:** Boolean - 计算结果为 true 或 false 的表达式。

### 示例

在下面的示例中，括号内的条件对变量 name 进行计算以查看它是否具有文本值 "Erica"。如果有，则运行花括号内的 play() 函数。

```
if(name == "Erica"){  
    play();  
}
```

下面的示例使用 if 语句来计算用户单击 SWF 文件中的 submit\_btn 实例需要多长时间。如果用户在 SWF 文件播放超过 10 秒时单击该按钮，则条件的计算结果为 true，花括号 ({} ) 中的消息将出现在运行时创建（使用 createTextField()）的文本字段中。如果用户在 SWF 文件播放不到 10 秒时单击该按钮，则条件的计算结果为 false，并且会出现不同的消息。

```
this.createTextField("message_txt", this.getNextHighestDepth, 0, 0, 100,  
    22);  
message_txt.autoSize = true;  
var startTime:Number = getTimer();  
this.submit_btn.onRelease = function() {  
    var difference:Number = (getTimer() - startTime) / 1000;  
    if (difference > 10) {  
        this._parent.message_txt.text = "Not very speedy, you took "+difference+"  
            seconds.";  
    }  
    else {  
        this._parent.message_txt.text = "Very good, you hit the button in  
            "+difference+" seconds.";  
    }  
};
```

另请参见

[else 语句](#)

## implements 语句

```
myClass implements interface01 [, interface02 , ...]
```

指定类必须定义所实现的接口中声明的所有方法。

可用性：Flash Player 6；ActionScript 2.0

示例

请参阅 [interface](#)。

另请参见

[class 语句](#)

## import 语句

```
import className  
import packageName.*
```

使您不必指定类的完全限定名就可以访问类。例如，如果您要在脚本中使用自定义类 **macr.util.users.UserClass**，则必须通过其全限定名引用它或导入它；如果您导入该类，则可以通过类名称引用它：

```
// before importing  
var myUser:macr.util.users.UserClass = new macr.util.users.UserClass();  
// after importing  
import macr.util.users.UserClass;  
var myUser:UserClass = new UserClass();
```

如果要访问的包 (*working\_directory*/macr/utlils/users) 中有多个类文件，可以用一条语句将它们全部导入，如下例所示：

```
import macr.util.users.*;
```

您必须首先发出 `import` 语句，然后尝试在不完全指定导入的类的名称的情况下访问导入的类。

如果您导入一个类，但没有在脚本中使用该类，则该类不作为 **SWF** 文件的一部分导出。这意味着您导入大型包时可以不担心 **SWF** 文件的大小；只有在实际使用某一类的情况下，才会在 **SWF** 文件中包括与该类关联的字节码。

`import` 语句仅应用于调用该语句的当前脚本（帧或对象）。例如，假设您在某个 **Flash** 文档的第 1 帧上导入了 **macr.util** 包中的所有类。那么，在那一帧上则可以使用简单类名来引用该包中的类。

```
// On Frame 1 of a FLA:  
import macr.util.*;  
var myFoo:foo = new foo();
```

但在其它帧脚本中，仍需要使用完全限定名来引用该包中的类 (`var myFoo:foo = new macr.util.foo();`)，或在其它帧上添加一条 `import` 语句以导入该包中的类。

**可用性：Flash Player 6； ActionScript 2.0**

### 参数

`className:String` - 您在外部类文件中定义的类的完全限定名。

### 示例

## interface 语句

```
interface InterfaceName [extends InterfaceName ] {}
```

定义接口。接口与类相似，但也具有以下重要差异：

- 接口仅包含方法的声明，而不包含其实现。也就是说，实现接口的每个类必须为该接口中声明的每个方法提供实现。
- 在接口定义中只允许全局成员，不允许实例或类成员。
- 在接口定义中不允许使用 `get` 和 `set` 语句。

**可用性：Flash Player 6； ActionScript 2.0**

### 示例

下面的示例说明定义和实现接口的若干方法：

```
(in top-level package .as files Ia, B, C, Ib, D, Ic, E)
// filename Ia.as
interface Ia {
    function k():Number; // method declaration only
    function n(x:Number):Number; // without implementation
}
// filename B.as
class B implements Ia {
    function k():Number {
        return 25;
    }
    function n(x:Number):Number {
        return x + 5;
    }
} // external script or Actions panel // script file
var mvar:B = new B();
trace(mvar.k()); // 25
trace(mvar.n(7)); // 12
// filename c.as
class C implements Ia {
    function k():Number {
```

```

    return 25;
}
} // error: class must implement all interface methods
// filename Ib.as
interface Ib {
    function o():Void;
}
class D implements Ia, Ib {
    function k():Number {
        return 15;
    }
    function n(x:Number):Number {
        return x * x;
    }
    function o():Void {
        trace("o");
    }
} // external script or Actions panel // script file
mvar = new D();
trace(mvar.k()); // 15
trace(mvar.n(7)); // 49
trace(mvar.o()); // "o"
interface Ic extends Ia {
    function p():Void;
}
class E implements Ib, Ic {
    function k():Number {
        return 25;
    }
    function n(x:Number):Number {
        return x + 5;
    }
    function o():Void {
        trace("o");
    }
    function p():Void {
        trace("p");
    }
}

```

另请参见

[class 语句](#)

## intrinsic 语句

```
intrinsic class className [extends superClass] [implements interfaceName [,
    interfaceName...]] {
    //class definition here
}
```

允许对以前定义的类执行编译时类型检查。**Flash** 使用内部类声明来启用对内置类（如 Array、Object 和 String）的编译时类型检查。此关键字向编译器指示不需要函数实现，而且不应该为它生成字节码。

**intrinsic** 关键字也可以与变量和函数声明一起使用。**Flash** 使用此关键字来允许对全局函数和属性执行编译时类型检查。

**intrinsic** 关键字是专为对内置类和对象以及全局变量和函数启用编译时类型检查而创建的。此关键字不用于一般目的，但是对于试图允许对以前定义的类（尤其是使用 **ActionScript 1.0** 定义的类）执行编译时类型检查的开发人员可能有一些价值。

仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

可用性：Flash Player 6；ActionScript 2.0

### 示例

下面的示例说明如何对以前定义的 **ActionScript 1.0** 类启用编译时文件检查。该代码将生成编译时错误，因为调用 `myCircle.setRadius()` 将 String 值作为参数而不是 Number 值来发送。您可以避免该错误，方法是将参数更改为 Number 值（例如，将 "10" 更改为 10）。

```
// The following code must be placed in a file named Circle.as
// that resides within your classpath:
intrinsic class Circle {
    var radius:Number;
    function Circle(radius:Number);
    function getArea():Number;
    function getDiameter():Number;
    function setRadius(param_radius:Number):Number;
}

// This ActionScript 1.0 class definition may be placed in your FLA file.
// Circle class is defined using ActionScript 1.0
function Circle(radius) {
    this.radius = radius;
    this.getArea = function(){
        return Math.PI*this.radius*this.radius;
    };
    this.getDiameter = function() {
        return 2*this.radius;
    };
    this.setRadius = function(param_radius) {
        this.radius = param_radius;
    };
}
```



```
}  
}
```

```
// ActionScript 2.0 code that uses the Circle class  
var myCircle:Circle = new Circle(5);  
trace(myCircle.getArea());  
trace(myCircle.getDiameter());  
myCircle.setRadius("10");  
trace(myCircle.radius);  
trace(myCircle.getArea());  
trace(myCircle.getDiameter());
```

另请参见

[class 语句](#)

## private 语句

```
class someClassName{  
    private var name;  
    private function name() {  
        // your statements here  
    }  
}
```

指定变量或函数只对声明或定义该变量或函数的类或该类的子类可用。默认情况下，变量和函数对任何调用者都可用。使用此关键字可以限制对变量或函数的访问。此关键字旨在作为一种软件开发帮助手段，以促进良好的代码编写方法（例如封装），而不是用于隐藏或保护敏感数据的一种安全机制。不必在运行时阻止对变量的访问。

您只能在类定义中使用此关键字，不能在接口定义中使用。

**可用性：**Flash Player 6；ActionScript 2.0

### 参数

**name:**String - 要指定为私有的变量或函数的名称。

### 示例

下面的示例演示如何通过使用 `private` 关键字限制对变量或函数的访问。创建名为 **Alpha.as** 的新 AS 文件。

```
class Alpha {  
    private var privateProperty = "visible only within class and subclasses";  
    public var publicProperty = "visible everywhere";  
}
```

在 **Alpha.as** 所在的目录中，创建一个名为 **Beta.as** 的新 AS 文件，该文件包含以下代码：

```
class Beta extends Alpha {  
    function Beta() {  
        trace("privateProperty is " + privateProperty);  
    }  
}
```

如以下代码所示，**Beta** 类的构造函数能够访问从 **Alpha** 类继承的 `privateProperty` 属性：

```
var myBeta:Beta = new Beta(); // Output: privateProperty is visible only  
    within class and subclasses
```

尝试从 **Alpha** 类或从 **Alpha** 类继承的类的外部访问 `privateProperty` 变量将导致一个错误。驻留在任何类的外部的以下代码将导致一个错误：

```
trace(myBeta.privateProperty); // Error
```

另请参见

[public 语句](#)

## public 语句

```
class someClassName{  
    public var name;  
    public function name() {  
        // your statements here  
    }  
}
```

指定变量或函数对任何调用者都可用。因为默认情况下变量和函数是公共的，所以使用此关键字主要是出于格式上的原因。例如，在包含 **private** 或 **static** 变量的代码块中，您可能要使用此关键字来保持格式一致。

可用性：Flash Player 6； ActionScript 2.0

### 参数

**name:String** - 要指定为公共的变量或函数的名称。

### 示例

下面的示例说明如何在类文件中使用公共变量。创建名为 **User.as** 的新类文件，并输入以下代码：

```
class User {  
    public var age:Number;  
    public var name:String;  
}
```

然后在同一目录中创建新的 **FLA** 或 **AS** 文件，并在时间轴的第 1 帧中输入以下 **ActionScript**:

```
import User;
var jimmy:User = new User();
jimmy.age = 27;
jimmy.name = "jimmy";
```

如果将 **User** 类中的公共变量之一更改为私有变量，则在尝试访问该属性时会生成错误。

另请参见

[private 语句](#)

## return 语句

`return[expression]`

指定由函数返回的值。return 语句计算 **expression** 并将结果作为在其中执行该语句的函数的值返回。return 语句使执行立即返回到调用函数。如果单独使用 return 语句，则它返回 `undefined`。

您不能返回多个值。如果尝试返回多个值，则将只返回最后一个值。在下面的示例中，返回 `c`：

```
return a, b, c ;
```

如果需要返回多个值，可以改用数组或对象。

可用性：Flash Player 5 ； ActionScript 1.0

### 返回

String - 经过计算的 *expression* 参数（如果提供了）。

### 参数

*expression* - 要计算的字符串、数字、布尔值、数组或对象，其计算结果作为函数值返回。此参数是可选的。

### 示例

下面的示例使用 `sum()` 函数体内的 return 语句，以返回三个参数相加后的值。下一行代码调用 `sum()` 并将返回值赋予变量 `newValue`。

```
function sum(a:Number, b:Number, c:Number):Number {
    return (a + b + c);
}
var newValue:Number = sum(4, 32, 78);
trace(newValue); // output: 114
```

另请参见

[function 语句](#)

## set 语句

```
function set property(varName) {  
    // your statements here  
}
```

允许隐式设置与某些对象关联的属性，这些对象基于外部类文件中定义的类。通过使用隐式设置方法，您可以不必直接访问对象属性就改变其属性值。隐式 **get/set** 方法是对 **ActionScript 1.0** 中 `Object.addProperty()` 方法的句法简化。

可用性：Flash Player 6；ActionScript 2.0

### 参数

**property:String** - 用来引用 set 要访问的属性的词；该值必须与在相应的 get 命令中使用的值相同。

### 示例

下面的示例创建一个 **Login** 类，该类说明如何使用 set 关键字设置私有变量：

```
class Login {  
    private var loginUserName:String;  
    private var loginPassword:String;  
    public function Login(param_username:String, param_password:String) {  
        this.loginUserName = param_username;  
        this.loginPassword = param_password;  
    }  
    public function get username():String {  
        return this.loginUserName;  
    }  
    public function set username(param_username:String):Void {  
        this.loginUserName = param_username;  
    }  
    public function set password(param_password:String):Void {  
        this.loginPassword = param_password;  
    }  
}
```

在与 **Login.as** 位于同一目录中的 **FLA** 或 **AS** 文件中，在时间轴的第 1 帧中输入以下 **ActionScript**：

```
var gus:Login = new Login("Gus", "Smith");  
trace(gus.username); // output: Gus  
gus.username = "Rupert";  
trace(gus.username); // output: Rupert
```

在下面的示例中，在跟踪值时执行 `get` 函数。`set` 函数仅在您向它传递值时才触发，如下行所示：

```
gus.username = "Rupert";
```

另请参见

[get 语句](#)

## set variable 语句

```
set("variableString", expression)
```

为变量赋值。 *variable* 是保存数据的容器。容器始终不变，但内容可以更改。通过在 SWF 文件播放时更改变量的值，可以记录和保存关于用户操作的信息、记录 SWF 文件播放时更改的值，或者计算某个条件是 `true` 还是 `false`。

变量可以保存任何类型的数据（例如，字符串、数字、布尔值、对象或影片剪辑）。每个 SWF 文件和影片剪辑的时间轴都有其自己的变量集，每个变量又都有其自己的独立于其它时间轴上的变量的值。

`set` 语句中不支持严格数据类型指定。如果使用此语句将某个变量设置为一个值，而该值的数据类型与类文件中与该变量关联的数据类型不同，则不生成编译器错误。

需要牢记的一个细微而重要的区别是：参数 *variableString* 是字符串，而不是变量名。如果将一个现有的变量名作为第一个参数传递给 `set()` 而没有用引号（`"`）将名称括起来，则会首先计算该变量，然后将 *expression* 的值赋予它。例如，如果创建一个名为 `myVariable` 的字符串变量并且对它赋值 `"Tuesday"`，然后忘了使用引号，就会不小心创建一个名为 `Tuesday` 的新变量，它包含您想分配给 `myVariable` 的值：

```
var myVariable:String = "Tuesday";
set (myVariable, "Saturday");
trace(myVariable); // outputs Tuesday
trace(Tuesday); // outputs Saturday
```

可以通过使用引号（`"`）来避免这种情况：

```
set ("myVariable", "Saturday");
trace(myVariable); //outputs Saturday
```

可用性：Flash Player 4；ActionScript 1.0

### 参数

**variableString:String** - 一个字符串，它命名一个用于保存 **expression** 参数的值的变量。

## 示例

在下面的示例中，将值赋予变量。您可以将 "Jakob" 的值赋予 name 变量。

```
set("name", "Jakob");  
trace(name);
```

下面的代码循环三次，并创建三个名为 caption0、caption1 和 caption2 的新变量：

```
for (var i = 0; i < 3; i++) {  
    set("caption" + i, "this is caption " + i);  
}  
trace(caption0);  
trace(caption1);  
trace(caption2);
```

## 另请参见

[var 语句](#)

## static 语句

```
class someClassName{  
    static var name;  
    static function name() {  
        // your statements here  
    }  
}
```

指定某个变量或函数只为每个类创建一次，而不是在基于该类的每个对象中都创建。

您可以通过使用语法 `someClassName.name` 在不创建类的实例的情况下访问静态类成员。如果您确实创建类的实例，也可以使用该实例访问静态成员，但只能通过访问静态成员的非静态函数来访问。

您只能在类定义中使用此关键字，不能在接口定义中使用。

**可用性：Flash Player 6 ； ActionScript 2.0**

## 参数

**name:**String - 要指定为静态的变量或函数的名称。

## 示例

下面的示例说明如何使用 `static` 关键字创建一个计数器，该计数器跟踪已创建类的实例的数量。由于 `numInstances` 变量是静态的，因此它只对整个类创建一次，而不是对每个单独实例都创建一次。创建名为 **Users.as** 的新 **AS** 文件，并输入以下代码：

```
class Users {  
    private static var numInstances:Number = 0;  
    function Users() {  
        numInstances++;  
    }  
    static function get instances():Number {  
        return numInstances;  
    }  
}
```

在同一目录中创建一个 **FLA** 或 **AS** 文档，并在时间轴的第 1 帧中输入以下 **ActionScript**：

```
trace(Users.instances);  
var user1:Users = new Users();  
trace(Users.instances);  
var user2:Users = new Users();  
trace(Users.instances);
```

另请参见

[private 语句](#)

## super 语句

```
super.method([arg1, ..., argN])  
super([arg1, ..., argN])
```

第一种语法格式可以在对象方法体内使用，用于调用方法的超类版本，而且可以选择向超类方法传递参数 (`arg1 ... argN`)。这对于创建某些子类方法很有用，这些子类方法在向超类方法添加附加行为的同时，又调用这些超类方法执行其原始行为。

第二种语法格式可以用于构造函数体内，用以调用此构造函数的超类版本，而且可以选择向它传递参数。这对于创建子类很有用，该子类在执行附加的初始化的同时，又调用超类构造函数执行超类初始化。

可用性：Flash Player 6；ActionScript 1.0

## 返回

两种格式都调用一个函数。该函数可以返回任何值。

## 参数

**method:**Function - 要在超类中调用的方法。

**argN** - 可选参数，这些参数或者传递给方法的超类版本（语法 1），或者传递给超类的构造函数（语法 2）。

## switch 语句

```
switch (expression){  
  caseClause:  
    [defaultClause:]  
}
```

创建 **ActionScript** 语句的分支结构。与 **if** 语句一样，**switch** 语句测试一个条件，并在条件返回 **true** 值时执行语句。所有 **switch** 语句都包含一个默认 **case**。默认 **case** 中应包含一个 **break** 语句，以避免在以后添加其它 **case** 时出现落空错误。当一个 **case** 落空时，它没有 **break** 语句。

可用性：Flash Player 4；ActionScript 1.0

## 参数

**expression** - 任何表达式。

## 示例

在下面的示例中，如果 `String.fromCharCode(Key.getAscii())` 参数的计算结果为 **A**，将执行 **case "A"** 后面的 `trace()` 语句；如果该参数的计算结果为 **a**，将执行 **case "a"** 后面的 `trace()` 语句；依此类推。如果没有与 `String.fromCharCode(Key.getAscii())` 参数匹配的 **case** 表达式，将执行 **default** 关键字后面的 `trace()` 语句。

```
var listenerObj:Object = new Object();  
listenerObj.onKeyDown = function() {  
  switch (String.fromCharCode(Key.getAscii())) {  
    case "A" :  
      trace("you pressed A");  
      break;  
    case "a" :  
      trace("you pressed a");  
      break;  
    case "E" :  
    case "e" :  
      trace("you pressed E or e");  
      break;  
    case "I" :  
    case "i" :  
      trace("you pressed I or i");  
      break;
```



```
default :  
    trace("you pressed some other key");  
    break;  
}  
};  
Key.addListener(listenerObj);
```

另请参见

[=== 全等运算符](#)

## throw 语句

*throw expression*

生成或抛出（使用 **throw** 语句）一个可由 `catch()` 代码块处理或捕获的错误。如果异常未被 `catch` 代码块捕获，抛出值的字符串表示形式将被发送到“输出”面板。

通常，抛出的错误是 **Error** 类或其子类的实例（请参见“示例”部分）。

可用性：Flash Player 7； ActionScript 1.0

### 参数

**expression:**Object - 一个 **ActionScript** 表达式或对象。

### 示例

在此示例中，一个名为 `checkEmail()` 的函数检查传递给它的字符串是否为格式正确的电子邮件地址。如果该字符串不包含 **@** 符号，则该函数将抛出一个错误。

```
function checkEmail(email:String) {  
    if (email.indexOf("@") == -1) {  
        throw new Error("Invalid email address");  
    }  
}  
checkEmail("someuser_theirdomain.com");
```

然后下面的代码调用 `try` 代码块内的 `checkEmail()` 函数。如果 `email_txt` 字符串不包含有效的电子邮件地址，将在一个文本字段 (`error_txt`) 中显示错误消息。

```
try {  
    checkEmail("Joe Smith");  
}  
catch (e) {  
    error_txt.text = e.toString();  
}
```

在下面的示例中，将抛出 **Error** 类的一个子类。对 `checkEmail()` 函数进行了修改，使其抛出该子类的一个实例。

```
// Define Error subclass InvalidEmailError // In InvalidEmailError.as: class
InvalidEmailAddress extends Error { var message = "Invalid email
address."; }
```

在 **FLA** 或 **AS** 文件中，在时间轴的第 1 帧中输入以下 **ActionScript**：

```
import InvalidEmailAddress;
function checkEmail(email:String) {
    if (email.indexOf("@") == -1) {
        throw new InvalidEmailAddress();
    }
}
try {
    checkEmail("Joe Smith");
}
catch (e) {
    this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100,
        22);
    error_txt.autoSize = true;
    error_txt.text = e.toString();
}
```

另请参见

[Error](#)

## try..catch..finally 语句

```
try {
    // ... try block ...
} finally {
    // ... finally block ...
}
try {
    // ... try block ...
} catch(error [:ErrorType1]) {
    // ... catch block ...
} [catch(error[:ErrorTypeN]) {
    // ... catch block ...
}] [finally {
    // ... finally block ...
}]
```

包含一个代码块，在其中可发生错误，然后对该错误进行响应。如果 `try` 代码块中的任何代码抛出错误（使用 `throw` 语句），控制会传递到 `catch` 块（如果有），然后传递到 `finally` 代码块（如果有）。无论是否抛出了错误，始终会执行 `finally` 块。如果 `try` 代码块中的代码未抛出错误（也就是说，如果 `try` 代码块正常完成），则仍会执行 `finally` 代码块中的代码。即使 `try` 代码块使用 `return` 语句退出，仍执行 `finally` 代码块。

`try` 代码块后面必须跟有 `catch` 代码块、`finally` 代码块，或两者都有。单个 `try` 代码块可以有多个 `catch` 代码块，但只能有一个 `finally` 代码块。您可以根据需要嵌套任意层数的 `try` 代码块。

`catch` 处理函数中指定的 `error` 参数必须是一个简单的标识符，如 `e`、`theException` 或 `x`。您还可以为 `catch` 处理函数中的变量指定类型。当与多个 `catch` 代码块一起使用时，如果指定了错误类型，则可以捕获从一个 `try` 代码块抛出的多种类型的错误。

如果抛出的异常是对象，则当抛出的对象是指定类型的子类时，类型将匹配。如果抛出的错误属于特定类型，将执行处理相应错误的 `catch` 代码块。如果抛出的异常不属于指定类型，则不执行 `catch` 代码块，而自动将该异常从 `try` 代码块抛出到与其匹配的 `catch` 处理函数。

如果在某个函数内抛出了错误，而该函数不包含 `catch` 处理函数，则 **ActionScript** 解释程序退出该函数以及任何调用函数，直到找到一个 `catch` 代码块。在此过程中，在各层上都会调用 `finally` 处理函数。

可用性：Flash Player 7；ActionScript 1.0

## 参数

`error:Object` - 从 `throw` 语句抛出的表达式，通常是 **Error** 类或其子类之一的实例。

## 示例

下面的示例说明如何创建 `try..finally` 语句。由于一定会执行 `finally` 代码块中的代码，因此，它通常用于在 `try` 代码块执行完毕后执行任何必要的清理。在下面的示例中，`setInterval()` 每隔 1000 毫秒（1 秒）调用一次函数。如果发生错误，则会抛出一个错误并由 `catch` 代码块捕获。无论是否出现错误，都会执行 **finally** 代码块。由于使用了 `setInterval()`，因此必须将 `clearInterval()` 放在 `finally` 代码块中以确保从内存中清除该时间间隔：

```
myFunction = function () {  
    trace("this is myFunction");  
};  
try {  
    myInterval = setInterval(this, "myFunction", 1000);  
    throw new Error("my error");  
}  
catch (myError:Error) {  
    trace("error caught: "+myError);  
}
```

```
finally {
    clearInterval(myInterval);
    trace("error is cleared");
}
```

在下面的示例中，`finally` 代码块用于删除一个 **ActionScript** 对象，而不管是否出现错误。创建一个名为 **Account.as** 的新 **AS** 文件：

```
class Account {
    var balance:Number = 1000;
    function getAccountInfo():Number {
        return (Math.round(Math.random() * 10) % 2);
    }
}
```

在 **Account.as** 所在的目录中，创建一个新的 **AS** 或 **FLA** 文档，然后在时间轴的第 1 帧中输入以下 **ActionScript**：

```
import Account;
var account:Account = new Account();
try {
    var returnVal = account.getAccountInfo();
    if (returnVal != 0) {
        throw new Error("Error getting account information.");
    }
}
finally {
    if (account != null) {
        delete account;
    }
}
```

下面的示例说明一条 `try..catch` 语句。执行 `try` 代码块内的代码。如果由 `try` 代码块中的任何代码抛出了异常，控制将传递到 `catch` 代码块，该代码块使用 `Error.toString()` 方法在一个文本字段中显示错误消息。

在 **Account.as** 所在的目录中，创建一个新的 **FLA** 文档，然后在时间轴的第 1 帧中输入以下 **ActionScript**：

```
import Account;
var account:Account = new Account();
try {
    var returnVal = account.getAccountInfo();
    if (returnVal != 0) {
        throw new Error("Error getting account information.");
    }
    trace("success");
}
catch (e) {
    this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 100,
        22);
    status_txt.autoSize = true;
    status_txt.text = e.toString();
}
```

下面的示例说明了一个 try 代码块，该代码块带有多个指定了类型的 catch 代码块。根据所出现错误的类型，try 代码块将抛出不同类型的对象。在此情况下，myRecordSet 是一个名为 **RecordSet** 的类（假设有这样一个类）的实例。该类的 sortRows() 方法可以抛出以下两种类型的错误：**RecordSetException** 和 **MalformedRecord**。

在下面的示例中，**RecordSetException** 和 **MalformedRecord** 对象是 **Error** 类的子类。它们都是在其自己的 AS 类文件中定义的。

```
// In RecordSetException.as:
class RecordSetException extends Error {
    var message = "Record set exception occurred.";
}
// In MalformedRecord.as:
class MalformedRecord extends Error {
    var message = "Malformed record exception occurred.";
}
```

在 **RecordSet** 类的 sortRows() 方法内，将根据所发生异常的类型抛出其中一个以前定义的错误对象。下面的示例说明此代码的外观：

```
class RecordSet {
    function sortRows() {
        var returnVal:Number = randomNum();
        if (returnVal == 1) {
            throw new RecordSetException();
        }
        else if (returnVal == 2) {
            throw new MalformedRecord();
        }
    }
    function randomNum():Number {
        return Math.round(Math.random() * 10) % 3;
    }
}
```

最后，在另一个 **AS** 文件或 **FLA** 脚本中，以下代码对 **RecordSet** 类的一个实例调用 sortRows() 方法。它为 sortRows() 抛出的每种类型的错误定义 catch 代码块。

```
import RecordSet;
var myRecordSet:RecordSet = new RecordSet();
try {
    myRecordSet.sortRows();
    trace("everything is fine");
}
catch (e:RecordSetException) {
    trace(e.toString());
}
catch (e:MalformedRecord) {
    trace(e.toString());
}
```

另请参见

[Error](#)

## var 语句

```
var variableName [= value1][...,variableNameN[=valueN]]
```

用于声明局部变量。如果变量是在函数内声明的，则这些变量是局部变量。它们是为该函数声明的，在函数调用结束时到期。更具体地说，用 var 定义的变量是包含它的代码块的局部变量。代码块是用花括号 ({} ) 区分的。

如果在函数外声明变量，变量在包含该语句的整个时间轴中可用。

您不能将范围限于另一个对象的变量声明为局部变量。

```
my_array.length = 25; // ok
var my_array.length = 25; // syntax error
```

当使用 var 时，您可以严格指定变量的类型。

您可以在一条语句中声明多个变量，用逗号分隔各个声明（尽管此语法可能会降低代码的清晰程度）：

```
var first:String = "Bart", middle:String = "J.", last:String = "Bartleby";
```



在外部脚本的类定义中声明属性时，也必须使用 var。类文件还支持 public、private 和 static 变量范围。

可用性：Flash Player 5；ActionScript 1.0

### 参数

**variableName:String** - 一个标识符。

### 示例

下面的 **ActionScript** 创建一个新的产品名称数组。Array.push 将一个元素添加到该数组的末尾。如果要使用严格类型指定，则必须使用 var 关键字。如果 product\_array 前面没有 var，您在尝试使用严格类型指定时会遇到错误。

```
var product_array:Array = new Array("MX 2004", "Studio", "Dreamweaver",
    "Flash", "ColdFusion", "Contribute", "Breeze");
product_array.push("Flex");
trace(product_array);
// output: MX
    2004,Studio,Dreamweaver,Flash,ColdFusion,Contribute,Breeze,Flex
```

## while 语句

```
while(condition) {  
    statement(s);  
}
```

计算条件，如果条件计算结果为 `true`，则在循环返回以再次计算条件之前执行一条语句或一系列语句。在条件计算结果为 `false` 后，跳过该语句或语句系列并结束循环。

`while` 语句执行下面一系列步骤。第 1 步至第 4 步的每次重复，称作循环的一次迭代。每次迭代的开始将重新测试 *condition*，如下面的步骤所示：

- 计算表达式 *condition*。
- 如果 *condition* 计算结果是 `true` 或一个转换为布尔值 `true` 的值（如一个非零数），则转到第 3 步。否则，`while` 语句结束并继续执行 `while` 循环后面的下一个语句。
- 运行语句块 *statement(s)*。
- 转到步骤 1。

通常当计数器变量小于某指定值时，使用循环执行动作。在每个循环的结尾递增计数器的值，直到达到指定值为止。此时，*condition* 不再为 `true`，因此循环结束。

如果将只执行一条语句，用来括起要由 `while` 语句执行的语句块的花括号 `{ }` 是不必要的。

可用性：Flash Player 4；ActionScript 1.0

### 参数

**condition:** Boolean - 计算结果为 `true` 或 `false` 的表达式。

### 示例

在下面的示例中，`while` 语句用于测试表达式。在 `i` 的值小于 20 时，跟踪 `i` 的值。当条件不再为 `true` 时，循环将退出。

```
var i:Number = 0;  
while (i < 20) {  
    trace(i);  
    i += 3;  
}
```

下面的结果显示在“输出”面板中。

```
0  
3  
6  
9  
12  
15  
18
```

另请参见

[continue 语句](#)

## with 语句

```
with (object:Object) {  
    statement(s);  
}
```

允许您使用 *object* 参数指定一个对象（例如影片剪辑），并使用 *statement(s)* 参数计算该对象内的表达式和动作。这可以使您不必重复书写对象的名称或路径。

*object* 参数变为在其中读取 *statement(s)* 参数中的属性、变量和函数的上下文。例如，如果 *object* 是 *my\_array*，并且指定的两个属性为 *length* 和 *concat*，则这些属性自动作为 *my\_array.length* 和 *my\_array.concat* 读取。在另一个示例中，如果 *object* 是 *state.california*，则 *with* 语句内部的任何动作或语句将从 *california* 实例的内部调用。

若要查找 *statement(s)* 参数中某个标识符的值，**ActionScript** 将从 *object* 指定的范围链的开头开始查找，并按照特定的顺序在范围链的每个级别中搜索该标识符。

*with* 语句使用范围链解析标识符，该范围链从下面列表中的第一项开始，到最后一项结束：

- 在最里面的 *with* 语句中的 *object* 参数中指定的对象。
- 在最外面的 *with* 语句中的 *object* 参数中指定的对象。
- 激活的对象。（当调用函数时自动创建的临时对象，该函数包含函数所调用的局部变量。）
- 包含当前执行脚本的影片剪辑。
- 全局对象（诸如 **Math** 与 **String** 的内置对象）。

若要在 *with* 语句中设置变量，该变量必须已在 *with* 语句外部进行了声明，或者您必须输入该变量所存在的时间轴的完整路径。如果在 *with* 语句中设置了未声明的变量，*with* 语句将根据范围链查找该值。如果该变量尚不存在，则将在调用 *with* 语句的时间轴上设置此新值。

可以用直接路径来代替 *with()*。如果觉得路径输入起来又长又麻烦，可以创建一个局部变量并把路径存储到其中，这样就可以重用代码，如下面的 **ActionScript** 所示：

```
var shortcut = this._parent._parent.name_txt; shortcut.text = "Hank";  
shortcut.autoSize = true;
```

可用性：Flash Player 5；ActionScript 1.0

### 参数

*object:Object* - **ActionScript** 对象或影片剪辑的一个实例。

### 示例

下面的示例设置 *someOther\_mc* 实例的 *\_x* 和 *\_y* 属性，然后指示 *someOther\_mc* 转到第 3 帧并停止。

```
with (someOther_mc) {  
    _x = 50;  
    _y = 100;  
    gotoAndStop(3);  
}
```



下面的代码片断说明如何在不使用 with 语句的情况下编写上述代码。

```
someOther_mc._x = 50;
someOther_mc._y = 100;
someOther_mc.gotoAndStop(3);
```

with 语句对于同时访问一个范围链列表中的多个项很有用。在下面的示例中，内置 Math 对象被放置在范围链的前面。将 Math 设置为将标识符 cos、sin 和 PI 分别解析为 Math.cos、Math.sin 和 Math.PI 的默认对象。标识符 a、x、y 和 r 不是 Math 对象的方法或属性，但是，由于它们存在于函数 polar() 的对象激活范围内，它们将解析为对应的局部变量。

```
function polar(r:Number):Void {
    var a:Number, x:Number, y:Number;
    with (Math) {
        a = PI * pow(r, 2);
        x = r * cos(PI);
        y = r * sin(PI / 2);
    }
    trace("area = " + a);
    trace("x = " + x);
    trace("y = " + y);
} polar(3);
```

下面的结果显示在“输出”面板中。

```
area = 28.2743338823081
x = -3
y = 3
```



**ActionScript** 类的文档包括属于 **ActionScript** 中特定类的方法、属性和事件处理函数以及侦听器的语法、用法信息和代码示例（与全局函数或属性相对）。这些类按照字母顺序列出，并且包括 **flash.\*** 包中提供的 **Flash Player 8** 的新类。如果您不能确定某个方法或属性属于哪个类，则可以在索引中查找它。

## Accessibility

```
Object
|
+-Accessibility
```

```
public class Accessibility
extends Object
```

**Accessibility** 类管理与屏幕读取器之间的通讯。屏幕读取器是一种辅助性技术，可以为视力有缺陷的用户提供屏幕内容的音频版本。**Accessibility** 类的方法是静态方法，也就是说，您不必创建该类的实例即可使用其方法。

若要获取和设置特定对象（例如按钮、影片剪辑或文本字段）的可访问属性，请使用 `_accProps` 属性。若要确定播放器是否正在支持辅助功能的环境中运行，请使用 `System.capabilities.hasAccessibility`。

可用性：ActionScript 1.0；Flash Player 6

另请参见

[hasAccessibility](#) ([capabilities.hasAccessibility](#) 属性)，[\\_accProps](#) 属性

属性摘要  
继承自 Object 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),  
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

方法摘要

修饰符	签名	说明
static	isActive() : Boolean	指示某个辅助功能当前是否处于活动状态，并且播放器是否正在与其通讯。
static	updateProperties() : Void	使对 _accProps （辅助功能属性）对象的所有更改生效。

继承自 Object 类的方法

```
addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法),  
isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法),  
registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法),  
valueOf (Object.valueOf 方法), watch (Object.watch 方法)
```

# isActive （Accessibility.isActive 方法）

public static isActive() : Boolean

指示某个辅助功能当前是否处于活动状态，并且播放器是否正在与其通讯。当希望应用程序在有屏幕读取器或其它辅助功能的情况下行为方式不同时，可使用此方法。

提醒

如果您在播放文档的 Flash 窗口第一次出现后一秒或两秒时间内调用此方法，则可能获得返回值 false，即使有活动的 Microsoft Active Accessibility (MSAA) 客户端也是如此。这是由于 Flash 和 MSAA 客户端之间的异步通讯机制造成的。您可以通过确保在加载您的文档后延迟 1 秒到 2 秒，再调用此方法，来变通解决这一限制问题。

可用性：ActionScript 1.0 ； Flash Player 6

返回

Boolean - 一个布尔值: 如果 **Flash Player** 正在与某一辅助功能（通常为屏幕阅读器）通讯，则返回 true ； 否则返回 false。

## 示例

下面的示例检查某个辅助功能当前是否处于活动状态：

```
if (Accessibility.isActive()) {  
    trace ("An accessibility aid is currently active");  
} else {  
    trace ("There is currently no active accessibility aid");  
}
```

## 另请参见

[updateProperties](#) ([Accessibility.updateProperties](#) 方法), [\\_accProps](#) 属性,  
[hasAccessibility](#) ([capabilities.hasAccessibility](#) 属性)

## updateProperties (Accessibility.updateProperties 方法)

`public static updateProperties() : Void`

使对 `_accProps`（辅助功能属性）对象的所有更改生效。有关设置辅助功能属性的信息，请参见 `_accProps`。

如果您修改多个对象的辅助功能属性，则只需调用 `Accessibility.updateProperties()` 一次；多次调用可能导致性能降低以及屏幕阅读器的结果无法理解。

可用性：ActionScript 1.0；Flash Player 6,0,65,0

## 示例

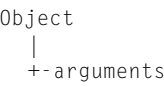
如果更改某个图像并且想要更新它的辅助功能说明，可以使用以下 **ActionScript** 代码：

```
my_mc.gotoAndStop(2);  
  
if (my_mc._accProps == undefined) {  
    my_mc._accProps = new Object();  
}  
my_mc._accProps.name = "Photo of Mount Rushmore";  
Accessibility.updateProperties();
```

## 另请参见

[isActive](#) ([Accessibility.isActive](#) 方法), [\\_accProps](#) 属性, [hasAccessibility](#) ([capabilities.hasAccessibility](#) 属性)

# arguments



```
public class arguments
extends Object
```

arguments 对象用于存储和访问函数的参数。尽管它在函数体内部，也可以用本地 arguments 变量对其进行访问。

这些参数作为数组元素存储，第一个参数作为 arguments[0] 被访问，第二个参数作为 arguments[1] 被访问，依此类推。arguments.length 属性表示传递给函数的参数数目。请注意，传递给函数的参数与该函数声明的参数的数目可能不同。

可用性：ActionScript 1.0 ； Flash Player 5

另请参见  
[Function](#)

## 属性摘要

修饰符	属性	说明
	callee:Object	对当前正在执行的函数的引用。
	caller:Object	对调用当前正在执行的函数的函数的引用；如果未从其它函数调用它，则为 null。
	length:Number	传递给函数的参数数目。

继承自 Object 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 方法摘要

继承自 Object 类的方法

```
addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法),
isPrototypeOf (Object.isPrototypeOf 方法), isPrototypeOf (Object.isPrototypeOf 方法),
registerClass (Object.registerClass 方法), toString (Object.toString 方法),
unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)
```

## callee (arguments.callee 属性)

public callee : Object

对当前正在执行的函数的引用。

可用性: **ActionScript 1.0** ; **Flash Player 5**

另请参见

[caller \(arguments.caller 属性\)](#)

## caller (arguments.caller 属性)

public caller : Object

对调用当前正在执行的函数的函数的引用；如果未从其它函数调用它，则为 null。

可用性: **ActionScript 1.0** ; **Flash Player 6**

另请参见

[callee \(arguments.callee 属性\)](#)

## length (arguments.length 属性)

public length : Number

传递给函数的参数数目。此数目可以大于或小于函数声明的参数数目。

可用性: **ActionScript 1.0** ; **Flash Player 5**

# Array

Object

|

+-Array

public dynamic class **Array**

extends Object

**Array** 类使您可以访问和处理索引数组。索引数组是一个对象，其属性由表示该属性在数组中位置的数字来标识。此数字称为索引。所有索引数组都从零开始，这意味着数组中的第一个元素为 [0]，第二个元素为 [1]，依此类推。若要创建一个 **Array** 对象，请使用构造函数 **new Array()**。若要访问数组中的元素，请使用数组访问运算符 ([ ])。

您可以在数组元素中存储各种各样的数据类型，包括数字、字符串、对象，甚至是其它数组。您可以创建一个多维数组，方法是创建一个索引数组，然后给它的每个元素分配不同的索引数组。这样的数组被视为是多维的，原因是它可用于表示表中的数据。

数组分配是按引用而不是按值进行的：当您将一个数组变量分配给另一个数组变量时，这两个变量都引用同一数组：

```
var oneArray:Array = new Array("a", "b", "c");
var twoArray:Array = oneArray; // Both array variables refer to the same
    array.
twoArray[0] = "z";
trace(oneArray); // Output: z,b,c.
```

**Array** 类不应该用于创建关联数组，关联数组是不同的数据结构，它们包含命名元素而不是编号元素。您应该使用 **Object** 类创建关联数组（也称为哈希）。虽然 **ActionScript** 允许您使用 **Array** 类创建关联数组，但您不能使用 **Array** 类的任何方法或属性。关键是，关联数组是 **Object** 类的实例，而每个键 / 值对由属性及属性的值表示。要将关联数组声明为 **Object** 数据类型还有另一个原因：您可以使用对象文本来填充关联数组（但只能在您声明它时）。下面的示例使用对象文本创建一个关联数组，使用点运算符和数组访问运算符访问项，然后通过创建一个新属性来添加新的键 / 值对：

```
var myAssocArray:Object = {fname:"John", lname:"Public"};
trace(myAssocArray.fname); // Output: John
trace(myAssocArray["lname"]); // Output: Public
myAssocArray.initial = "Q";
trace(myAssocArray.initial); // Output: Q
```

可用性：ActionScript 1.0 ； Flash Player 5

示例

在下面的示例中， **my\_array** 包含了一年中的四个月份：

```
var my_array:Array = new Array();
my_array[0] = "January";
my_array[1] = "February";
my_array[2] = "March";
my_array[3] = "April";
```

属性摘要

修饰符	属性	说明
static	CASEINSENSITIVE:Number	在排序方法中，此常数指定不区分大小写的排序。
static	DESCENDING:Number	在排序方法中，此常数指定降序排序。
	length:Number	指定数组中元素数量的非负整数。
static	NUMERIC:Number	在排序方法中，此常数指定数字（而不是字符串）排序。
static	RETURNINDEXEDARRAY:Number	指定排序返回索引数组作为调用 sort() 或 sortOn() 方法的结果。
static	UNIQUESORT:Number	在排序方法中，此常数指定唯一的排序要求。



继承自 Object 类的属性

<code>constructor</code> (Object.constructor 属性), <code>__proto__</code> (Object.__proto__ 属性), <code>prototype</code> (Object.prototype 属性), <code>__resolve</code> (Object.__resolve 属性)
---

构造函数摘要

签名	说明
<code>Array([value:Object])</code>	它使您可以创建数组。

方法摘要

修饰符	签名	说明
	<code>concat([value:Object]) : Array</code>	将参数中指定的元素与数组中的元素连接，并创建新的数组。
	<code>join([delimiter:String]) : String</code>	将数组中的元素转换为字符串、在元素间插入指定的分隔符、连接这些元素然后返回结果字符串。
	<code>pop() : Object</code>	删除数组中最后一个元素，并返回该元素的值。
	<code>push(value:Object) : Number</code>	将一个或多个元素添加到数组的结尾，并返回该数组的新长度。
	<code>reverse() : Void</code>	在当前位置倒转数组。
	<code>shift() : Object</code>	删除数组中第一个元素，并返回该元素。
	<code>slice([startIndex:Number], [endIndex:Number]) : Array</code>	返回由原始数组中某一范围的元素构成的新数组，而不修改原始数组。
	<code>sort([compareFunction:Object], [options:Number]) : Array</code>	对数组中的元素进行排序。
	<code>sortOn(fieldName:Object, [options:Object]) : Array</code>	根据数组中的一个或多个字段对数组中的元素进行排序。
	<code>splice(startIndex:Number, [deleteCount:Number], [value:Object]) : Array</code>	给数组添加元素以及从数组中删除元素。
	<code>toString() : String</code>	返回一个字符串值，该值表示所指定的 <code>Array</code> 对象中的元素。
	<code>unshift(value:Object) : Number</code>	将一个或多个元素添加到数组的开头，并返回该数组的新长度。

继承自 Object 类的方法

---

`addProperty` (Object.addProperty 方法), `hasOwnProperty` (Object.hasOwnProperty 方法), `isPropertyEnumerable` (Object.isPropertyEnumerable 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `registerClass` (Object.registerClass 方法), `toString` (Object.toString 方法), `unwatch` (Object.unwatch 方法), `valueOf` (Object.valueOf 方法), `watch` (Object.watch 方法)

---

## Array 构造函数

```
public Array([value:Object])
```

它使您可以创建数组。您可使用该构造函数来创建不同类型的数组：空数组、具有特定长度但其中元素具有未定义值的数组或其中元素具有特定值的数组。

用法 1：如果未指定任何参数，则创建长度为 0 的数组。

用法 2：如果仅指定长度，则创建元素数为 `length` 的数组。每个元素的值设置为 `undefined`。

用法 3：如果使用 `element` 参数指定值，则创建具有特定值的数组。

可用性：ActionScript 1.0；Flash Player 5

### 参数

`value:Object` [ 可选 ] - 以下二者之一：

- 一个指定数组中元素数量的整数。
- 一个包含两个或多个任意值的列表。这些值的类型可以为 `Boolean`、`Number`、`String`、`Object` 或 `Array`。数组中第一个元素的索引或位置始终为 0。

注意：如果只有一个数字参数传递给 `Array` 构造函数，则会将该数字参数假定为 `length`，并通过使用 `Integer()` 函数将其转换为整数。

### 示例

用法 1：下面的示例创建一个新的 `Array` 对象，其初始长度为 0：

```
var my_array:Array = new Array();  
trace(my_array.length); // Traces 0.
```

用法 2：下面的示例创建一个新的 `Array` 对象，其初始长度为 4：

```
var my_array:Array = new Array(4);  
trace(my_array.length); // Returns 4.  
trace(my_array[0]); // Returns undefined.  
if (my_array[0] == undefined) { // No quotation marks around undefined.  
    trace("undefined is a special value, not a string");  
} // Traces: undefined is a special value, not a string.
```

用法 3: 下面的示例创建初始长度为 5 的新 **Array** 对象 go\_gos\_array:

```
var go_gos_array:Array = new Array("Belinda", "Gina", "Kathy", "Charlotte",  
    "Jane");  
trace(go_gos_array.length); // Returns 5.  
trace(go_gos_array.join(", ")); // Displays elements.
```

标识出 go\_gos\_array 数组的初始元素, 如下面的示例所示:

```
go_gos_array[0] = "Belinda";  
go_gos_array[1] = "Gina";  
go_gos_array[2] = "Kathy";  
go_gos_array[3] = "Charlotte";  
go_gos_array[4] = "Jane";
```

下面的代码将第六个元素添加到 go\_gos\_array 数组中并更改第二个元素:

```
go_gos_array[5] = "Donna";  
go_gos_array[1] = "Nina";  
trace(go_gos_array.join(" + "));  
// Returns Belinda + Nina + Kathy + Charlotte + Jane + Donna.
```

另请参见

[\[\] 数组访问运算符](#), [length \(Array.length 属性\)](#)

## CASEINSENSITIVE (Array.CASEINSENSITIVE 属性)

```
public static CASEINSENSITIVE : Number
```

在排序方法中, 此常数指定不区分大小写的排序。您可以对 `sort()` 方法或 `sortOn()` 方法中的 `options` 参数使用此常数。

此常数的值为 1。

可用性: ActionScript 1.0 ; Flash Player 7

另请参见

[sort \(Array.sort 方法\)](#), [sortOn \(Array.sortOn 方法\)](#)

## concat (Array.concat 方法)

`public concat([value:Object]) : Array`

将参数中指定的元素与数组中的元素连接，并创建新的数组。如果 `value` 参数指定的是数组，则连接该数组的元素而不是该数组本身。数组 `my_array` 保持不变。

可用性：ActionScript 1.0；Flash Player 5

### 参数

`value:Object` [ 可选 ] - 要在新数组中连接的数字、元素或字符串。如果您没有传递任何值，则创建 `my_array` 的一个副本。

### 返回

Array - 一个数组，包含此数组中的元素，后跟参数中的元素。

### 示例

下面的代码连接两个数组：

```
var alpha_array:Array = new Array("a","b","c");
var numeric_array:Array = new Array(1,2,3);
var alphaNumeric_array:Array =alpha_array.concat(numeric_array);
trace(alphaNumeric_array);
// Creates array [a,b,c,1,2,3].
```

下面的代码连接三个数组：

```
var num1_array:Array = [1,3,5];
var num2_array:Array = [2,4,6];
var num3_array:Array = [7,8,9];
var nums_array:Array=num1_array.concat(num2_array,num3_array)
trace(nums_array);
// Creates array [1,3,5,2,4,6,7,8,9].
```

嵌套数组不能像普通数组那样展开。嵌套数组中的元素不会溶解为数组 `x_array` 中的独立元素，如下面的示例所示：

```
var a_array:Array = new Array ("a","b","c");

// 2 and 3 are elements in a nested array.
var n_array:Array = new Array(1, [2, 3], 4);

var x_array:Array = a_array.concat(n_array);
trace(x_array[0]); // a
trace(x_array[1]); // b
trace(x_array[2]); // c
trace(x_array[3]); // 1
trace(x_array[4]); // 2, 3
trace(x_array[5]); // 4
```

## DESCENDING (Array.DESENDING 属性)

`public static DESCENDING : Number`

在排序方法中，此常数指定降序排序。您可以对 `sort()` 方法或 `sortOn()` 方法中的 `options` 参数使用此常数。

此常数的值为 2。

可用性: **ActionScript 1.0 ; Flash Player 7**

另请参见

[sort \(Array.sort 方法\)](#), [sortOn \(Array.sortOn 方法\)](#)

## join (Array.join 方法)

`public join([delimiter:String]) : String`

将数组中的元素转换为字符串、在元素间插入指定的分隔符、连接这些元素然后返回结果字符串。嵌套数组总是以逗号 (,) 分隔，而不使用传递给 `join()` 方法的分隔符分隔。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**delimiter:String [ 可选 ]** - 在返回字符串中分隔数组元素的字符或字符串。如果省略此参数，则使用逗号 (,) 作为默认分隔符。

### 返回

String - 一个字符串。

### 示例

下面的示例创建一个具有三个元素的数组: **Earth**、**Moon** 和 **Sun**。然后，它将该数组接合三次: 首先通过使用默认分隔符 (逗号 [,] 和空格)，然后通过使用短划线 (-)，最后通过使用加号 (+)。

```
var a_array:Array = new Array("Earth","Moon","Sun")
trace(a_array.join());
// Displays Earth,Moon,Sun.
trace(a_array.join(" - "));
// Displays Earth - Moon - Sun.
trace(a_array.join(" + "));
// Displays Earth + Moon + Sun.
```

下面的示例创建一个包含两个数组的嵌套数组。第一个数组有三个元素：**Europa**、**Io** 和 **Callisto**。第二个数组有两个元素：**Titan** 和 **Rhea**。它通过使用加号 (+) 来接合该数组，但每个嵌套数组中的元素保持由逗号 (,) 分隔。

```
var a_nested_array:Array = new Array(["Europa", "Io", "Callisto"], ["Titan",  
    "Rhea"]);  
trace(a_nested_array.join(" + "));  
// Returns Europa, Io, Callisto + Titan, Rhea.
```

另请参见

[split \(String.split 方法\)](#)

## length (Array.length 属性)

public length : Number

指定数组中元素数量的非负整数。在向数组中添加新元素时，此属性会自动更新。当您给数组元素赋值（例如，`my_array[index] = value`）时，如果 `index` 是数字，而且 `index+1` 大于 `length` 属性，则 `length` 属性会更新为 `index+1`。



如果您向 `length` 属性分配一个小于现有长度的值，则数组会被截断。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 示例

下面的代码说明了如何更新 `length` 属性。初始长度为 **0**，然后更新到 **1**、**2** 和 **10**。如果对 `length` 属性分配一个小于现有长度的值，则数组将被截断：

```
var my_array:Array = new Array();  
trace(my_array.length); // initial length is 0  
my_array[0] = "a";  
trace(my_array.length); // my_array.length is updated to 1  
my_array[1] = "b";  
trace(my_array.length); // my_array.length is updated to 2  
my_array[9] = "c";  
trace(my_array.length); // my_array.length is updated to 10  
trace(my_array);  
// displays:  
//  
a,b,undefined,undefined,undefined,undefined,undefined,undefined,undefined,  
c  
  
// if the length property is now set to 5, the array will be truncated  
my_array.length = 5;  
trace(my_array.length); // my_array.length is updated to 5  
trace(my_array); // outputs: a,b,undefined,undefined,undefined
```

## NUMERIC (Array.NUMERIC 属性)

```
public static NUMERIC : Number
```

在排序方法中，此常数指定数字（而不是字符串）排序。在 `options` 参数中包括此常数会导致 `sort()` 方法和 `sortOn()` 方法将数字作为数值排序，而不是作为数字字符的字符串排序。如果不使用 `NUMERIC` 常数，排序会将每个数组元素视为一个字符串，并且按照 `Unicode` 顺序生成结果。

例如，假定数组的值为 `[2005, 7, 35]`，如果 `options` 参数中不包括 `NUMERIC` 常数，则经过排序的数组为 `[2005, 35, 7]`，但是如果包括 `NUMERIC` 常数，则经过排序的数组为 `[7, 35, 2005]`。

请注意，此常数仅适用于数组中的数字；它不适用于包含数字数据的字符串（如 `["23", "5"]`）。

此常数的值为 16。

可用性：ActionScript 1.0；Flash Player 7

另请参见

[sort \(Array.sort 方法\)](#)，[sortOn \(Array.sortOn 方法\)](#)

## pop (Array.pop 方法)

```
public pop() : Object
```

删除数组中最后一个元素，并返回该元素的值。

可用性：ActionScript 1.0；Flash Player 5

返回

Object - 指定的数组中最后一个元素的值。

示例

下面的代码创建包含四个元素的 `myPets_array` 数组，然后删除其最后一个元素：

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var popped:Object = myPets_array.pop();
trace(popped); // Displays fish.
trace(myPets_array); // Displays cat,dog,bird.
```

另请参见

[push \(Array.push 方法\)](#)，[shift \(Array.shift 方法\)](#)，[unshift \(Array.unshift 方法\)](#)

## push (Array.push 方法)

`public push(value:Object) : Number`

将一个或多个元素添加到数组的结尾，并返回该数组的新长度。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**value:Object** - 要追加到数组中的一个或多个值。

### 返回

**Number** - 表示新数组长度的一个整数。

### 示例

下面的示例创建包含两个元素的数组 `myPets_array`，这两个元素为 `cat` 和 `dog`。第二行将两个元素添加到数组。

因为 `push()` 方法返回数组的新长度，所以最后一行中的 `trace()` 语句将 `myPets_array` 的新长度 (4) 发送到“输出”面板。

```
var myPets_array:Array = new Array("cat", "dog");  
var pushed:Number = myPets_array.push("bird", "fish");  
trace(pushed); // Displays 4.
```

### 另请参见

[pop \(Array.pop 方法\)](#)，[shift \(Array.shift 方法\)](#)，[unshift \(Array.unshift 方法\)](#)

## RETURNINDEXEDARRAY

### (Array.RETURNINDEXEDARRAY 属性)

`public static RETURNINDEXEDARRAY : Number`

指定排序返回索引数组作为调用 `sort()` 或 `sortOn()` 方法的结果。您可以对 `sort()` 方法或 `sortOn()` 方法中的 `options` 参数使用此常数。这可以通过返回表示排序结果的数组提供预览或复制功能，并保持原始数组不被修改。

此常数的值为 8。

可用性: **ActionScript 1.0 ; Flash Player 7**

### 另请参见

[sort \(Array.sort 方法\)](#)，[sortOn \(Array.sortOn 方法\)](#)



## reverse (Array.reverse 方法)

`public reverse() : Void`

在当前位置倒转数组。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例使用此方法倒转数组 `numbers_array`:

```
var numbers_array:Array = new Array(1, 2, 3, 4, 5, 6);
trace(numbers_array); // Displays 1,2,3,4,5,6.
numbers_array.reverse();
trace(numbers_array); // Displays 6,5,4,3,2,1.
```

## shift (Array.shift 方法)

`public shift() : Object`

删除数组中第一个元素，并返回该元素。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

`Object` - 数组中的第一个元素。

### 示例

下面的代码创建数组 `myPets_array`，然后删除该数组中的第一个元素，并将其分配给变量 `shifted`:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var shifted:Object = myPets_array.shift();
trace(shifted); // Displays "cat".
trace(myPets_array); // Displays dog,bird,fish.
```

### 另请参见

[pop \(Array.pop 方法\)](#), [push \(Array.push 方法\)](#), [unshift \(Array.unshift 方法\)](#)

## slice (Array.slice 方法)

```
public slice([startIndex:Number], [endIndex:Number]) : Array
```

返回由原始数组中某一范围的元素构成的新数组，而不修改原始数组。返回的数组包括 `startIndex` 元素以及从其开始到 `endIndex` 元素（但不包括该元素）的所有元素。

如果您没有传递任何参数，则创建原始数组的一个副本。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

**startIndex:**Number [ 可选 ] - 指定片段起始点索引的数字。如果 *start* 是负数，则起始点从数组的结尾开始，其中 **-1** 指的是最后一个元素。

**endIndex:**Number [ 可选 ] - 指定片段终点索引的数字。如果省略此参数，则片段包括数组中从开头到结尾的所有元素。如果 *end* 是负数，则终点从数组的结尾指定，其中 **-1** 指的是最后一个元素。

### 返回

Array - 一个由原始数组中某一范围的元素构成的数组。

### 示例

下面的示例创建包含五个宠物的数组，并使用 `slice()` 来填充只包含四条腿宠物的新数组：

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFourLeggedPets_array:Array = new Array();
var myFourLeggedPets_array = myPets_array.slice(0, 2);
trace(myFourLeggedPets_array); // Returns cat,dog.
trace(myPets_array); // Returns cat,dog,fish,canary,parrot.
```

下面的示例创建包含五个宠物的数组，然后使用具有负 `start` 参数的 `slice()` 复制数组中的最后两个元素：

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFlyingPets_array:Array = myPets_array.slice(-2);
trace(myFlyingPets_array); // Traces canary,parrot.
```

下面的示例创建包含五个宠物的数组，然后使用具有负 `end` 参数的 `slice()` 复制数组中的中间元素：

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myAquaticPets_array:Array = myPets_array.slice(2,-2);
trace(myAquaticPets_array); // Returns fish.
```

## sort (Array.sort 方法)

`public sort([compareFunction:Object], [options:Number]) : Array`

对数组中的元素进行排序。Flash 根据 Unicode 值排序。(ASCII 是 Unicode 的一个子集。)

默认情况下，Array.sort() 按下面的列表中的说明进行排序：

- 排序区分大小写 (Z 优先于 a)。
- 按升序排序 (a 优先于 b)。
- 修改该数组以反映排序顺序；在排序后的数组中不按任何特定顺序连续放置具有相同排序字段的多个元素。
- 数值字段按字符串方式进行排序，因此 100 优先于 99，因为 "1" 的字符串值比 "9" 的低。

如果您想通过使用不同于默认设置的设置来对数组进行排序，可以使用 options 参数的条目中描述的排序选项之一，或者，也可以创建自己的自定义函数来进行排序。如果您创建自定义函数，则可以通过调用 sort() 方法来使用它，将您的自定义函数的名称用作第一个参数 (compareFunction)。

可用性：ActionScript 1.0；Flash Player 5

### 参数

`compareFunction:Object` [ 可选 ] - 一个用来确定数组中元素排序顺序的比较函数。给定元素 A 和 B，compareFunction 的结果可具有以下三个值之一：

- -1，如果 A 应在排序后的序列中出现在 B 之前
- 0，如果 A 等于 B
- 1，如果 A 应在排序后的序列中出现在 B 之后

`options:Number` [ 可选 ] - 所定义常数的一个或多个数字或名称，相互之间由 |（按位“或”）运算符隔开，它们将排序的默认行为更改为其它行为。options 参数可接受以下值：

- Array.CASEINSENSITIVE 或 1
- Array.DECENDING 或 2
- Array.UNIQUESORT 或 4
- Array.RETURNINDEXEDARRAY 或 8
- Array.NUMERIC 或 16

有关此参数的更多信息，请参见 Array.sortOn() 方法。



Array.sort() 是在 ECMA-262 中定义的，但 Flash Player 7 中引入的数组排序选项是对 ECMA-262 规范的特定于 Flash 的扩展。

## 返回

Array - 返回值取决于您是否传递任何参数，如下面的列表中所述：

- 如果您为 options 参数指定值 4 或 Array.UNIQUESORT，并且所排序的两个或多个元素具有相同的排序字段，则 **Flash** 返回值 0 并且不修改该数组。
- 如果为 options 参数指定值 8 或 Array.RETURNINDEXEDARRAY，则 **Flash** 返回反映排序结果的数组并且不修改该数组。
- 否则，**Flash** 不返回任何内容并修改该数组以反映排序顺序。

## 示例

用法 1：下面的示例显示在具有和不具有为 options 传递的值这两种情况下如何使用

Array.sort()：

```
var fruits_array:Array = new Array("oranges", "apples", "strawberries",
    "pineapples", "cherries");
trace(fruits_array); // Displays
    oranges,apples,strawberries,pineapples,cherries.
fruits_array.sort();
trace(fruits_array); // Displays
    apples,cherries,oranges,pineapples,strawberries.
trace(fruits_array); // Writes
    apples,cherries,oranges,pineapples,strawberries.
fruits_array.sort(Array.DESENDING);
trace(fruits_array); // Displays
    strawberries,pineapples,oranges,cherries,apples.
trace(fruits_array); // Writes
    strawberries,pineapples,oranges,cherries,apples.
```

用法 2：下面的示例将 Array.sort() 用于比较函数。条目是按照 **name:password** 的形式排序的。仅使用条目的名称部分作为关键字排序。

```
var passwords_array:Array = new Array("mom:glam", "ana:ring", "jay:mag",
    "anne:home", "regina:silly");
function order(a, b):Number {
    var name1:String = a.split(":")[0];
    var name2:String = b.split(":")[0];
    if (name1<name2) {
        return -1;
    } else if (name1>name2) {
        return 1;
    } else {
        return 0;
    }
}
trace("Unsorted:");
//Displays Unsorted:
trace(passwords_array);
//Displays mom:glam,ana:ring,jay:mag,anne:home,regina:silly.
```

```
//Writes mom:glam,ana:ring,jay:mag,anne:home,regina:silly
passwords_array.sort(order);
trace("Sorted:");
//Displays Sorted:
trace(passwords_array);
//Displays ana:ring,anne:home,jay:mag,mom:glam,regina:silly.
//Writes ana:ring,anne:home,jay:mag,mom:glam,regina:silly.
```

另请参见

| [按位 OR 运算符](#), [sortOn \(Array.sortOn 方法\)](#)

## sortOn (Array.sortOn 方法)

```
public sortOn(fieldName:Object, [options:Object]) : Array
```

根据数组中的一个或多个字段对数组中的元素进行排序。数组应具有下列特性：

- 该数组是索引数组，不是关联数组。
- 该数组的每个元素都包含一个具有一个或多个属性的对象。
- 所有这些对象都至少有一个公用属性，该属性的值可用于对该数组进行排序。这样的属性称为 **field**。

如果您传递多个 `fieldName` 参数，则第一个字段表示主排序字段，第二个字段表示下一个排序字段，依此类推。**Flash** 根据 **Unicode** 值排序。（**ASCII** 是 **Unicode** 的一个子集。）如果所比较的两个元素中的任何一个不包含在 `fieldName` 参数中指定的字段，则认为该字段为 `undefined`，并且在排序后的数组中不按任何特定顺序连续放置这些元素。

默认情况下，`Array.sortOn()` 按以下方式进行排序：

- 排序区分大小写（**Z** 优先于 **a**）。
- 按升序排序（**a** 优先于 **b**）。
- 修改该数组以反映排序顺序；在排序后的数组中不按任何特定顺序连续放置具有相同排序字段的多个元素。
- 数值字段按字符串方式进行排序，因此 **100** 优先于 **99**，因为 **"1"** 的字符串值比 **"9"** 的低。

**Flash Player 7** 添加了 `options` 参数，您可以使用该参数覆盖默认排序行为。若要对简单数组（例如，仅具有一个字段的数组）进行排序，或要指定一种 `options` 参数不支持的排序顺序，请使用 `Array.sort()`。

若要传递多个标志，请使用按位 “或” (`|`) 运算符分隔它们：

```
my_array.sortOn(someFieldName, Array.DESENDING | Array.NUMERIC);
```

Flash Player 8 添加了按多个字段进行排序时为每个字段指定不同的排序选项的功能。在 Flash Player 8 中，options 参数接受一组排序选项，以便每个排序选项对应于 fieldName 参数中的一个排序字段。下面的示例使用降序排序对主排序字段 a 排序，使用数字排序对第二个排序字段 b 排序，使用不区分大小写的排序对第三个排序字段 c 排序：

```
Array.sortOn(["a", "b", "c"], [Array.DESENDING, Array.NUMERIC,
    Array.CASEINSENSITIVE]);
```



fieldName 和 options 数组必须具有相同数量的元素；否则，将忽略 options 数组。此外，Array.UNIQUESORT 和 Array.RETURNINDEXEDARRAY 选项只能用作数组中的第一个元素；否则，将忽略它们。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**fieldName:**Object - 一个标识要用作排序值的字段的字符串，或一个数组，其中的第一个元素表示主排序字段，第二个元素表示第二排序字段，依此类推。

**options:**Object [ 可选 ] - 所定义常数的一个或多个数字或名称，相互之间由 bitwise OR (|) 运算符隔开，它们可以更改排序行为。options 参数可接受以下值：

- Array.CASEINSENSITIVE 或 1
- Array.DESENDING 或 2
- Array.UNIQUESORT 或 4
- Array.RETURNINDEXEDARRAY 或 8
- Array.NUMERIC 或 16

如果您使用标志的字符串形式（例如，DESCENDING），而不是数字形式 (2)，则启用代码提示。

## 返回

Array - 返回值取决于是否传递任何参数：

- 如果您为 options 参数指定值 4 或 Array.UNIQUESORT，并且要排序的两个或多个元素具有相同的排序字段，则返回值 0 并且不修改数组。
- 如果为 options 参数指定值 8 或 Array.RETURNINDEXEDARRAY，则返回反映排序结果的数组并且不修改数组。
- 否则，不返回任何结果并修改该数组以反映排序顺序。

## 示例

下面的示例创建一个新数组，并且按照 name 字段和 city 字段对该新数组进行排序。第一次排序使用 name 作为第一个排序值，使用 city 作为第二个排序值。第二次排序使用 city 作为第一个排序值，使用 name 作为第二个排序值。

```
var rec_array:Array = new Array();
rec_array.push({name: "john", city: "omaha", zip: 68144});
rec_array.push({name: "john", city: "kansas city", zip: 72345});
rec_array.push({name: "bob", city: "omaha", zip: 94010});
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// john, omaha
// john, kansas city
// bob, omaha

rec_array.sortOn(["name", "city"]);
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// bob, omaha
// john, kansas city
// john, omaha

rec_array.sortOn(["city", "name" ]);
for(i=0; i<rec_array.length; i++){
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// Results:
// john, kansas city
// bob, omaha
// john, omaha
```

下面的对象数组由说明如何使用 options 参数的后续示例使用：

```
var my_array:Array = new Array();
my_array.push({password: "Bob", age:29});
my_array.push({password: "abcd", age:3});
my_array.push({password: "barb", age:35});
my_array.push({password: "catchy", age:4});
```

对 password 字段执行默认排序将产生以下结果：

```
my_array.sortOn("password");
// Bob
// abcd
// barb
// catchy
```

对 **password** 字段执行不区分大小写的排序将产生以下结果：

```
my_array.sortOn("password", Array.CASEINSENSITIVE);  
// abcd  
// barb  
// Bob  
// catchy
```

对 **password** 字段执行不区分大小写的降序排序将产生以下结果：

```
my_array.sortOn("password", Array.CASEINSENSITIVE | Array.DESCENDING);  
// catchy  
// Bob  
// barb  
// abcd
```

对 **age** 字段执行默认排序将产生以下结果：

```
my_array.sortOn("age");  
// 29  
// 3  
// 35  
// 4
```

对 **age** 字段执行数值排序将产生以下结果：

```
my_array.sortOn("age", Array.NUMERIC);  
// my_array[0].age = 3  
// my_array[1].age = 4  
// my_array[2].age = 29  
// my_array[3].age = 35
```

对 **age** 字段执行降序数值排序将产生以下结果：

```
my_array.sortOn("age", Array.DESCENDING | Array.NUMERIC);  
// my_array[0].age = 35  
// my_array[1].age = 29  
// my_array[2].age = 4  
// my_array[3].age = 3
```

在使用 `Array.RETURNEDINDEXARRAY` 排序选项时，您必须将返回值分配给不同的数组。原始数组不会被修改。

```
var indexArray:Array = my_array.sortOn("age", Array.RETURNINDEXEDARRAY);
```

另请参见

| [按位 OR 运算符](#), [sort \(Array.sort 方法\)](#)



## splice (Array.splice 方法)

```
public splice(startIndex:Number, [deleteCount:Number], [value:Object]) :  
    Array
```

给数组添加元素以及从数组中删除元素。此方法会修改数组但不制作副本。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**startIndex**:Number - 一个整数，它指定插入或删除动作开始处的数组中元素的索引。您可以指定一个负整数来指定相对于数组结尾的位置（例如，**-1** 是数组的最后一个元素）。

**deleteCount**:Number [ 可选 ] - 一个整数，它指定要删除的元素数量。该数量包括 **startIndex** 参数中指定的元素。如果没有为 **deleteCount** 参数指定值，则该方法将删除从 **startIndex** 元素到数组中最后一个元素之间的所有值。如果该参数的值为 **0**，则不删除任何元素。

**value**:Object [ 可选 ] - 指定要在 **startIndex** 参数中指定的插入点处插入到数组中的值。

### 返回

Array - 包含从原始数组中删除的元素的一个数组。

### 示例

下面的示例创建一个数组，并使用 **startIndex** 参数的元素索引 **1** 连接该数组。这将删除数组中从第二个元素开始的所有元素，只保留原始数组中索引 **0** 处的元素：

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");  
trace( myPets_array.splice(1) ); // Displays dog,bird,fish.  
trace( myPets_array ); // cat
```

下面的示例创建一个数组，并使用 **startIndex** 参数的元素索引 **1** 和 **deleteCount** 参数的数字 **2** 连接该数组。这样就会删除数组中从第二个元素开始的两个元素，保留原始数组中的第一个和最后一个元素：

```
var myFlowers_array:Array = new Array("roses", "tulips", "lilies",  
    "orchids");  
trace( myFlowers_array.splice(1,2) ); // Displays tulips,lilies.  
trace( myFlowers_array ); // roses,orchids
```

下面的示例创建一个数组，并使用 **startIndex** 参数的元素索引 **1**、**deleteCount** 参数的数字 **0** 和 **value** 参数的字符串 **chair** 连接该数组。这样不会删除原始数组中的任何内容，但会在索引 **1** 处添加字符串 **chair**：

```
var myFurniture_array:Array = new Array("couch", "bed", "desk", "lamp");  
trace( myFurniture_array.splice(1,0, "chair" ) ); // Displays empty array.  
trace( myFurniture_array ); // displays couch,chair,bed,desk,lamp
```

## toString (Array.toString 方法)

public toString() : String

返回一个字符串值，该值表示所指定的 **Array** 对象中的元素。数组中的每一个元素（从索引 0 开始到最高索引结束）均会转换为一个连接字符串，并以逗号分隔。若要指定自定义的分隔符，请使用 `Array.join()` 方法。

可用性：ActionScript 1.0；Flash Player 5

### 返回

String - 一个字符串。

### 示例

下面的示例创建 `my_array`，并将其转换为字符串。

```
var my_array:Array = new Array();
my_array[0] = 1;
my_array[1] = 2;
my_array[2] = 3;
my_array[3] = 4;
my_array[4] = 5;
trace(my_array.toString()); // Displays 1,2,3,4,5.
```

此示例输出 1、2、3、4、5 作为 `trace` 语句的结果。

### 另请参见

[split \(String.split 方法\)](#)，[join \(Array.join 方法\)](#)

## UNIQUESORT (Array.UNIQUESORT 属性)

public static UNIQUESORT : Number

在排序方法中，此常数指定唯一的排序要求。您可以对 `sort()` 方法或 `sortOn()` 方法中的 **options** 参数使用此常数。如果任何两个要排序的元素或字段的值相同，唯一排序选项就会中止排序。

此常数的值为 4。

可用性：ActionScript 1.0；Flash Player 7

### 另请参见

[sort \(Array.sort 方法\)](#)，[sortOn \(Array.sortOn 方法\)](#)

## unshift (Array.unshift 方法)

`public unshift(value:Object) : Number`

将一个或多个元素添加到数组的开头，并返回该数组的新长度。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**value:Object** - 一个或多个要插入到数组开头的数字、元素或变量。

### 返回

**Number** - 表示该数组新长度的一个整数。

### 示例

下面的示例演示如何使用 `Array.unshift()` 方法:

```
var pets_array:Array = new Array("dog", "cat", "fish");
trace( pets_array ); // Displays dog,cat,fish.
pets_array.unshift("ferrets", "gophers", "engineers");
trace( pets_array ); // Displays ferrets,gophers,engineers,dog,cat,fish.
```

### 另请参见

[pop \(Array.pop 方法\)](#), [push \(Array.push 方法\)](#), [shift \(Array.shift 方法\)](#)

## AsBroadcaster

```
Object
|
+-AsBroadcaster
```

```
public class AsBroadcaster
extends Object
```

提供事件通知功能和侦听器管理功能，这些功能可以添加到用户定义的对象中。此类为目标用户是想要创建自定义事件处理机制的高级用户。您可以使用此类来使任何对象成为事件广播器，并创建一个或多个侦听器对象，以便在广播对象调用 `broadcastMessage()` 方法时随时接收通知。

**AsBroadcaster** 类没有构造函数。若要使用此类，请按照以下过程操作：

- 选择或创建一个对象以用作事件广播器。
- 通过调用静态 `AsBroadcaster.initialize(obj:Object)` 方法（其中 `obj` 参数是您选择用作广播器的对象的名称）使对象成为事件广播器。
- 选择或创建一个或多个侦听器对象。侦听器对象会随时在广播对象广播消息时接收通知

- 为每个侦听器对象定义侦听器方法。该侦听器方法将执行 **ActionScript** 代码以响应事件通知。方法的名称必须与广播对象广播的事件的名称匹配。
- 通过调用 `myBroadcaster.addListener(myListener)` 向事件广播器注册每个侦听器对象，其中 `myBroadcaster` 为事件广播器对象的名称，`myListener` 为侦听器对象的名称。每个事件广播器都存储着广播消息时要通知的侦听器对象的列表。使用 `addListener()` 方法将侦听器添加到列表，使用 `removeListener()` 从列表中删除侦听器。
- 最后，为了广播消息，调用 `myBroadcaster.broadcastMessage(eventName:String)` 方法，其中 `myBroadcaster` 为事件广播器的名称，`eventName` 为与侦听器方法的名称匹配的事件的名称。

提醒

一个常见错误是将 `AsBroadcaster` 的第二个字母变为大写。当调用 `AsBroadcaster.initialize()` 方法时，请确保第二个字母为小写。对 `AsBroadcaster` 的任何拼写错误都将导致失败且无任何提示。

可用性: **ActionScript 1.0 ; Flash Player 6**

属性摘要

修饰符	属性	说明
	<code>_listeners:Array [ 只读 ]</code>	对所有已注册侦听器对象的引用的列表。

继承自 `Object` 类的属性

`constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),  
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)`

方法摘要

修饰符	签名	说明
	<code>addListener(listenerObj:Object) : Boolean</code>	注册一个对象，以便接收事件通知消息。
	<code>broadcastMessage(eventName:String) : Void</code>	向侦听器列表中的每个对象发送事件消息。
<code>static</code>	<code>initialize(obj:Object) : Void</code>	将事件通知和侦听器管理功能添加到给定的对象。
	<code>removeListener(listenerObj:Object) : Boolean</code>	从接收事件通知消息的对象列表中删除对象。

继承自 Object 类的方法

---

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPropertyEnumerable](#) ([Object.isPropertyEnumerable](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

---

## addListener (AsBroadcaster.addListener 方法)

`public addListener(listenerObj:Object) : Boolean`

注册一个对象，以便接收事件通知消息。针对广播对象调用此方法，侦听器对象会作为一个参数被发送。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 参数

**listenerObj:Object** - 接收事件通知的侦听器对象的名称。

### 返回

**Boolean** - 虽然从技术角度看此方法返回的是布尔值，但实际它返回的是 **Void**，因为它总是会返回值 **true**。

### 示例

下面的示例摘自 `AsBroadcaster.initialize()` 方法的条目中提供的完整示例。

```
someObject.addListener(myListener1); // Register myListener1 as listener.  
someObject.addListener(myListener2); // Register myListener2 as listener.
```

### 另请参见

[initialize](#) ([AsBroadcaster.initialize](#) 方法), [removeListener](#) ([AsBroadcaster.removeListener](#) 方法)

## broadcastMessage (AsBroadcaster.broadcastMessage 方法)

```
public broadcastMessage(eventName:String) : Void
```

向侦听器列表中的每个对象发送事件消息。当侦听器对象接收到消息时，Flash Player 尝试调用侦听器对象上的同名函数。假定您的对象按如下方式广播事件消息：

```
obj.broadcastMessage("onAlert");
```

当收到此消息时，Flash Player 调用接收方侦听器对象上名为 onAlert() 的方法。



您可以将其它参数包含到 broadcastMessage() 方法中，从而将参数传递到侦听器函数。任何出现在 eventName 参数后的参数都会由侦听器方法作为参数接收。

您仅可以从使用 AsBroadcaster.initialize() 方法初始化的对象调用此方法。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**eventName:String** - 要广播的事件的名称。任何侦听器方法的名称都必须与此参数匹配才能接收广播事件。您可以在 eventName 后包含其它参数，从而将参数传递到侦听器方法。

### 示例

下面的示例摘自 AsBroadcaster.initialize() 方法的条目中提供的第一个完整示例：

```
someObject.broadcastMessage("someEvent"); // Broadcast the "someEvent"
message.
```

下面的示例摘自 AsBroadcaster.initialize() 方法的条目中提供的第二个完整示例。它演示如何将参数发送给侦听器方法。

```
someObject.broadcastMessage("someEvent", 3, "arbitrary string");
```

### 另请参见

[initialize \(AsBroadcaster.initialize 方法\)](#)，[removeListener \(AsBroadcaster.removeListener 方法\)](#)

## initialize (AsBroadcaster.initialize 方法)

```
public static initialize(obj:Object) : Void
```

将事件通知和侦听器管理功能添加到给定的对象。这是一个静态方法；必须通过使用 **AsBroadcaster** 类（其中 `someObject` 是要初始化为事件广播器的对象的名称）来调用它：

```
AsBroadcaster.initialize(someObject);
```

提醒

一个常见错误是将 `AsBroadcaster` 的第二个字母变为大写。当调用 `AsBroadcaster.initialize()` 方法时，请确保第二个字母为小写。对 `AsBroadcaster` 的任何拼拼写错误都将导致失败且无任何提示。

此方法将 `_listeners` 属性以及下面的三个方法添加到由 `obj` 参数指定的对象中：

- `obj.addListener()`
- `obj.removeListener()`
- `obj.broadcastMessage()`

可用性：ActionScript 1.0；Flash Player 6

### 参数

`obj:Object` - 一个要用作广播对象的对象。

### 示例

下面的示例创建一个通用对象 `someObject`，并将其变为事件广播器。输出应该是两个 `trace()` 语句中显示的字符串：

```
var someObject:Object = new Object(); // Creates broadcast object.

var myListener1:Object = new Object(); // Creates listener object.
var myListener2:Object = new Object(); // Creates listener object.

myListener1.someEvent = function() { // Creates listener method.
    trace("myListener1 received someEvent");
}
myListener2.someEvent = function() { // Createez listener method.
    trace("myListener2 received someEvent");
}

AsBroadcaster.initialize(someObject); // Makes someObject an event
    broadcaster.
someObject.addListener(myListener1); // Registers myListener1 as listener.
someObject.addListener(myListener2); // Registers myListener2 as listener.
someObject.broadcastMessage("someEvent"); // Broadcasts the "someEvent"
    message.
```

下面的示例演示如何通过使用 `broadcastMessage()` 方法将额外的参数传递到侦听器方法。输出应该是三个 `trace()` 语句中显示的三个字符串，其中还包括通过 `broadcastMessage()` 方法传入的参数。

```
var someObject:Object = new Object();

var myListener:Object = new Object();
myListener.someEvent = function(param1:Number, param2:String) {
    trace("myListener received someEvent");
    trace("param1: " + param1);
    trace("param2: " + param2);
}

AsBroadcaster.initialize(someObject);
someObject.addListener(myListener);
someObject.broadcastMessage("someEvent", 3, "arbitrary string");
```

## `_listeners` (`AsBroadcaster._listeners` 属性)

```
public _listeners : Array [read-only]
```

对所有已注册侦听器对象的引用的列表。此属性专供内部使用，不用于直接操作。通过调用 `addListener()` 方法和 `removeListener()` 方法，可以分别在数组中添加和删除对象。

您仅可以从使用 `AsBroadcaster.initialize()` 方法初始化的对象调用此属性。

**可用性:** **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例演示如何使用 `length` 属性确定当前已注册到事件广播器中的侦听器对象数量。如果将下面的代码添加到 `AsBroadcaster.initialize()` 条目的“示例”部分的第一个完整示例中，它即会生效：

```
trace(someObject._listeners.length); // Output: 2

对于高级用户，下面的示例演示如何使用 _listeners 属性列出所有注册到事件广播器中的  
侦听器以及每个侦听器对象的所有属性。下面的示例为第一个侦听器对象创建两个不同的侦  
听器方法。

var someObject:Object = new Object(); // create broadcast object

var myListener1:Object = new Object(); // create listener object
var myListener2:Object = new Object(); // create listener object

myListener1.someEvent = function() { // create listener method
    trace("myListener1 received someEvent");
}
myListener1.anotherEvent = function() { // create another listener method
    trace("myListener1 received anotherEvent");
}
```



```

myListener2.someEvent = function() { // create listener method
    trace("myListener2 received someEvent");
}

AsBroadcaster.initialize(someObject); // make someObject an event
    broadcaster
someObject.addListener(myListener1); // register myListener1 as listener
someObject.addListener(myListener2); // register myListener2 as listener

var numListeners:Number = someObject._listeners.length; // get number of
    registered listeners

// cycle through all listener objects, listing all properties of each
    listener object
for (var i:Number = 0; i < numListeners; i++) {
    trace("Listener " + i + " listens for these events:");
    for (item in someObject._listeners[i]) {
        trace (" " + item + ": " + someObject._listeners[i][item]);
    }
}

```

另请参见

[initialize \(AsBroadcaster.initialize 方法\)](#)

## removeListener (AsBroadcaster.removeListener 方法)

```
public removeListener(listenerObj:Object) : Boolean
```

从接收事件通知消息的对象列表中删除对象。

您只能从已使用 `AsBroadcaster.initialize()` 方法初始化的对象中调用此方法。

**可用性:** ActionScript 1.0 ; Flash Player 6

### 参数

**listenerObj:Object** - 侦听器对象的名称，该对象已注册，可接收来自广播对象的事件通知。

### 返回

**Boolean** - 如果已删除侦听器对象，则返回 `true`；否则返回 `false`。

## 示例

下面的示例说明如何从已注册的侦听器列表中删除侦听器。如果将下面的代码添加到 `AsBroadcaster.initialize()` 条目的“示例”部分的第一个完整示例中，它即会生效。包含 `trace()` 语句只是为了验证在调用 `removeListener()` 方法后已注册的侦听器数量减少了一个。

```
trace(someObject._listeners.length); // Output: 2
someObject.removeListener(myListener1);
trace(someObject._listeners.length); // Output: 1
```

## 另请参见

[addListener \(AsBroadcaster.addListener 方法\)](#), [initialize \(AsBroadcaster.initialize 方法\)](#)

# BevelFilter (flash.filters.BevelFilter)

```
Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.BevelFilter
```

```
public class BevelFilter
extends BitmapFilter
```

**BevelFilter** 类允许您在 **Flash** 中给各种不同对象添加斜角效果。斜角效果使对象（如按钮）具有三维外观。您可以利用不同的加亮颜色和阴影颜色、斜角上的模糊量、斜角的角度、斜角的位置和挖空效果来自定义斜角的外观。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时给影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤销该设置操作。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter` 会取得源 **BitmapData** 对象和滤镜对象，并生成一个过滤后的图像。

您也可以在创作时对图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜支持舞台缩放。但是，它不支持常规缩放、旋转和倾斜。如果对象本身进行了缩放（如果 `_xscale` 和 `_yscale` 不是 **100%**），则滤镜不进行缩放。只有放大舞台时它才会缩放。

如果结果图像的宽度或高度将超过 2880 像素，则不应用滤镜。例如，如果您在放大一个大影片剪辑时使用了滤镜，则在结果图像超过 2880 像素的限制时，该滤镜将关闭。

可用性：ActionScript 1.0 ； Flash Player 8

另请参见

[filters \(MovieClip.filters 属性\)](#) , [cacheAsBitmap \(MovieClip.cacheAsBitmap 属性\)](#) , [filters \(Button.filters 属性\)](#) , [cacheAsBitmap \(Button.cacheAsBitmap 属性\)](#) , [filters \(TextField.filters 属性\)](#) , [applyFilter \(BitmapData.applyFilter 方法\)](#) , [MovieClip](#)

属性摘要

修饰符	属性	说明
	<code>angle:Number</code>	斜角的角度。
	<code>blurX:Number</code>	水平模糊量，以像素为单位。
	<code>blurY:Number</code>	垂直模糊量，以像素为单位。
	<code>distance:Number</code>	斜角的偏移距离。
	<code>highlightAlpha:Number</code>	加亮颜色的 Alpha 透明度值。
	<code>highlightColor:Number</code>	斜角的加亮颜色。
	<code>knockout:Boolean</code>	应用挖空效果 (true)，这将有效地使对象的填色变为透明，并显示文档的背景颜色。
	<code>quality:Number</code>	应用滤镜的次数。
	<code>shadowAlpha:Number</code>	阴影颜色的 Alpha 透明度值。
	<code>shadowColor:Number</code>	斜角的阴影颜色。
	<code>strength:Number</code>	印记或散布的强度。
	<code>type:String</code>	斜角类型。

继承自 Object 类的属性

[constructor \(Object.constructor 属性\)](#) , [\\_\\_proto\\_\\_ \(Object.\\_\\_proto\\_\\_ 属性\)](#) , [prototype \(Object.prototype 属性\)](#) , [\\_\\_resolve \(Object.\\_\\_resolve 属性\)](#)

构造函数摘要

签名	说明
BevelFilter([distance:Number], [angle:Number], [highlightColor:Number], [highlightAlpha:Number], [shadowColor:Number], [shadowAlpha:Number], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [type:String], [knockout:Boolean])	用指定参数初始化新的 BevelFilter 实例。

方法摘要

修饰符	签名	说明
	clone() : BevelFilter	返回此滤镜对象的副本。

继承自 BitmapFilter 类的方法

<a href="#">clone</a> ( <a href="#">BitmapFilter.clone</a> 方法)
--

继承自 Object 类的方法

<a href="#">addProperty</a> ( <a href="#">Object.addProperty</a> 方法), <a href="#">hasOwnProperty</a> ( <a href="#">Object.hasOwnProperty</a> 方法), <a href="#">isPropertyEnumerable</a> ( <a href="#">Object.isPropertyEnumerable</a> 方法), <a href="#">isPrototypeOf</a> ( <a href="#">Object.isPrototypeOf</a> 方法), <a href="#">registerClass</a> ( <a href="#">Object.registerClass</a> 方法), <a href="#">toString</a> ( <a href="#">Object.toString</a> 方法), <a href="#">unwatch</a> ( <a href="#">Object.unwatch</a> 方法), <a href="#">valueOf</a> ( <a href="#">Object.valueOf</a> 方法), <a href="#">watch</a> ( <a href="#">Object.watch</a> 方法)
--

## angle (BevelFilter.angle 属性)

public angle : Number

斜角的角度。有效值为 0 到 360 度。默认值是 45。

角度值表示理论上的光源落在对象上的角度，它决定了效果相对于该对象的位置。如果距离设置为 0，则效果相对于对象没有偏移，因此，angle 属性没有任何效果。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例将在用户单击现有 MovieClip 实例 (rect) 时更改该实例的 angle 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelDistance");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.angle = 225;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
.8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

# BevelFilter 构造函数

```
public BevelFilter([distance:Number], [angle:Number],  
    [highlightColor:Number], [highlightAlpha:Number], [shadowColor:Number],  
    [shadowAlpha:Number], [blurX:Number], [blurY:Number], [strength:Number],  
    [quality:Number], [type:String], [knockout:Boolean])
```

用指定参数初始化新的 **BevelFilter** 实例。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**distance**:Number [ 可选 ] - 斜角的偏移距离, 以像素为单位 (浮点)。默认值是 4。

**angle**:Number [ 可选 ] - 斜角的角度, 0 至 360 度。默认值是 45。

**highlightColor**:Number [ 可选 ] - 斜角的加亮颜色, 0xRRGGBB。默认值为 0xFFFFFF。

**highlightAlpha**:Number [ 可选 ] - 加亮颜色的 Alpha 透明度值。有效值为 0 到 1。例如, .25 设置透明度值为 25%。默认值是 1。

**shadowColor**:Number [ 可选 ] - 斜角的阴影颜色, 0xRRGGBB。默认值为 0x000000。

**shadowAlpha**:Number [ 可选 ] - 阴影颜色的 Alpha 透明度值。有效值为 0 到 1。例如, .25 设置透明度值为 25%。默认值是 1。

**blurX**:Number [ 可选 ] - 水平模糊量, 以像素为单位。有效值为 0 到 255 (浮点)。默认值为 4。作为 2 的乘方的值 (如 2、4、8、16 和 32) 经过了优化, 呈现速度比其它值更快。

**blurY**:Number [ 可选 ] - 垂直模糊量, 以像素为单位。有效值为 0 到 255 (浮点)。默认值为 4。作为 2 的乘方的值 (如 2、4、8、16 和 32) 经过了优化, 呈现速度比其它值更快。

**strength**:Number [ 可选 ] - 印记或散布的强度。该值越大, 印记的颜色越深, 而且斜角与背景之间的对比度也越强。有效值为 0 到 255。默认值为 1。

**quality**:Number [ 可选 ] - 应用滤镜的次数。默认值为 1, 即表示低品质。值为 2 表示中等品质, 值为 3 表示高品质。

**type**:String [ 可选 ] - 斜角的类型。有效值为 "inner"、"outer" 和 "full"。默认值为 "inner"。

**knockout**:Boolean [ 可选 ] - 应用挖空效果 (true), 这将有效地使对象的填色变为透明, 并显示文档的背景颜色。默认值为 false (不应用挖空效果)。

## 示例

下面的示例实例化新的 **BevelFilter** 并将其应用于 **MovieClip** 实例 (rect):

```
import flash.filters.BevelFilter;

var distance:Number = 5;
var angleInDegrees:Number = 45;
var highlightColor:Number = 0xFFFF00;
var highlightAlpha:Number = .8;
var shadowColor:Number = 0x0000FF;
var shadowAlpha:Number = .8;
var blurX:Number = 5;
var blurY:Number = 5;
var strength:Number = 5;
var quality:Number = 3;
var type:String = "inner";
var knockout:Boolean = false;

var filter:BevelFilter = new BevelFilter(distance,
                                         angleInDegrees,
                                         highlightColor,
                                         highlightAlpha,
                                         shadowColor,
                                         shadowAlpha,
                                         blurX,
                                         blurY,
                                         strength,
                                         quality,
                                         type,
                                         knockout);

var rect:MovieClip = createRectangle(100, 100, 0x00CC00,
    "bevelFilterExample");
rect.filters = new Array(filter);

function createRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;
    return rect;
}
```

## blurX (BevelFilter.blurX 属性)

public blurX : Number

水平模糊量，以像素为单位。有效值为从 0 到 255（浮点）。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，它的呈现速度比其它值快。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 blurX 属性：

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelBlurX");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.blurX = 10;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```



## blurY (BevelFilter.blurY 属性)

public blurY : Number

垂直模糊量，以像素为单位。有效值为从 0 到 255（浮点）。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，它的呈现速度比其它值快。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 blurY 属性：

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelBlurY");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.blurY = 10;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## clone (BevelFilter.clone 方法)

public clone() : BevelFilter

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.filters.BevelFilter - 具有与原始 **BevelFilter** 实例相同的所有属性的新 **BevelFilter** 实例。

### 示例

下面的示例创建三个 **BevelFilter** 对象并对它们进行比较。您可以通过使用 **BevelFilter** 构造函数创建 filter\_1 对象。接着创建 filter\_2 对象，方法是将其设置为等于 filter\_1。然后通过克隆 filter\_1 创建 clonedFilter。请注意，尽管 filter\_2 等于 filter\_1，但 clonedFilter 却不等于 filter\_1，尽管它们包含相同的值。

```
import flash.filters.BevelFilter;

var filter_1:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF, .8,
    20, 20, 1, 3, "inner", false);
var filter_2:BevelFilter = filter_1;
var clonedFilter:BevelFilter = filter_1.clone();

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> type: inner
    // >> blurY: 20
    // >> blurX: 20
    // >> knockout: false
    // >> strength: 1
    // >> quality: 3
    // >> shadowAlpha: 0.8
    // >> shadowColor: 255
    // >> highlightAlpha: 0.8
    // >> highlightColor: 16776960
    // >> angle: 45
    // >> distance: 5
}
```

```

for(var i in clonedFilter) {
    trace(">> " + i + ": " + clonedFilter[i]);
    // >> clone: [type Function]
    // >> type: inner
    // >> blurY: 20
    // >> blurX: 20
    // >> knockout: false
    // >> strength: 1
    // >> quality: 3
    // >> shadowAlpha: 0.8
    // >> shadowColor: 255
    // >> highlightAlpha: 0.8
    // >> highlightColor: 16776960
    // >> angle: 45
    // >> distance: 5
}

```

为了进一步说明 `filter_1`、`filter_2` 和 `clonedFilter` 之间的关系，下面的示例将修改 `filter_1` 的 `knockout` 属性。通过修改 `knockout`，说明 `clone()` 方法是根据 `filter_1` 的值而不是引用这些值来创建实例的。

```

import flash.filters.BevelFilter;

var filter_1:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF, .8,
    20, 20, 1, 3, "inner", false);
var filter_2:BevelFilter = filter_1;
var clonedFilter:BevelFilter = filter_1.clone();

trace(filter_1.knockout); // false
trace(filter_2.knockout); // false
trace(clonedFilter.knockout); // false

filter_1.knockout = true;

trace(filter_1.knockout); // true
trace(filter_2.knockout); // true
trace(clonedFilter.knockout); // false

```

## distance (BevelFilter.distance 属性)

public distance : Number

斜角的偏移距离。有效值以像素为单位（浮点）。默认值是 4。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 distance 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelDistance");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.distance = 3;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
.8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## highlightAlpha (BevelFilter.highlightAlpha 属性)

public highlightAlpha : Number

加亮颜色的 **Alpha** 透明度值。该值被指定为从 0 到 1 的标准值。例如，.25 设置透明度值为 25%。默认值是 1。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 highlightAlpha 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelHighlightAlpha");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.highlightAlpha = .2;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## highlightColor (BevelFilter.highlightColor 属性)

public highlightColor : Number

斜角的加亮颜色。有效值采用十六进制格式，0xRRGGBB。默认值为 0xFFFFFFFF。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 highlightColor 属性：

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelHighlightColor");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.highlightColor = 0x0000FF;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## knockout (BevelFilter.knockout 属性)

public knockout : Boolean

应用挖空效果 (true)，这将有效地使对象的填色变为透明，并显示文档的背景颜色。默认值为 false（不应用挖空效果）。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 knockout 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelKnockout");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.knockout = true;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## quality (BevelFilter.quality 属性)

public quality : Number

应用滤镜的次数。默认值为 **1**，即表示低品质。值为 **2** 表示中等品质，值为 **3** 表示高品质。滤镜的值越小，呈现速度越快。

对于大多数应用，quality 的值为 **1**、**2** 或 **3** 就足够了。虽然您可以使用不超过 **15** 的其它数值来实现不同的效果，但是值越大，呈现速度越慢。除了增加 quality 的值，增加 blurX 和 blurY 的值通常也可以获得类似的效果，而且呈现速度更快。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 quality 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelQuality");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.quality = 1;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```



## shadowAlpha (BevelFilter.shadowAlpha 属性)

public shadowAlpha : Number

阴影颜色的 **Alpha** 透明度值。此值被指定为从 0 到 1 的标准值。例如，.25 设置透明度值为 25%。默认值是 1。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 shadowAlpha 属性

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelShadowAlpha");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.shadowAlpha = .2;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## shadowColor (BevelFilter.shadowColor 属性)

public shadowColor : Number

斜角的阴影颜色。有效值采用十六进制格式，0xRRGGBB。默认值为 0x000000。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 shadowColor 属性：

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelShadowColor");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.shadowColor = 0xFFFF00;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## strength (BevelFilter.strength 属性)

public strength : Number

印记或散布的强度。有效值为从 0 到 255。值越大，印记的颜色越深，斜角与背景之间的对比度也越强。默认值是 1。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 strength 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelStrength");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.strength = 10;
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

## type (BevelFilter.type 属性)

public type : String

斜角类型。有效值为 "inner"、"outer" 和 "full"。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将在用户单击现有 **MovieClip** 实例 (rect) 时更改该实例的 type 属性:

```
import flash.filters.BevelFilter;

var rect:MovieClip = createBevelRectangle("BevelType");
rect.onRelease = function() {
    var filter:BevelFilter = this.filters[0];
    filter.type = "outer";
    this.filters = new Array(filter);
}

function createBevelRectangle(name:String):MovieClip {
    var w:Number = 100;
    var h:Number = 100;
    var bgColor:Number = 0x00CC00;

    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    rect.beginFill(bgColor);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF,
        .8, 20, 20, 1, 3, "inner", false);
    rect.filters = new Array(filter);
    return rect;
}
```

# BitmapData (flash.display.BitmapData)

```
Object
|
+- flash.display.BitmapData
```

```
public class BitmapData
extends Object
```

**BitmapData** 类允许您在运行时创建任意大小的透明或不透明位图图像并采用多种方式操作这些图像。

此类允许您将位图呈现操作与 **Flash Player** 的内部显示更新例程分隔开来。通过直接操作 **BitmapData** 对象，您可以创建非常复杂的图像，不会因连续重新绘制矢量数据的内容而产生每帧开销。

**BitmapData** 类的方法支持多种无法通过通用滤镜接口获得的效果。

**BitmapData** 对象包含像素数据的数组。此数据可以表示完全不透明的位图，或表示包含 **Alpha** 通道数据的透明位图。以上任一类型的 **BitmapData** 对象都作为 32 位整数的缓冲区进行存储。每个 32 位整数确定位图中单个像素的属性。

每个 32 位整数都是四个 8 位通道值（从 0 到 255）的组合，这些值描述像素的 **Alpha** 透明度以及红色、绿色、蓝色的 (**ARGB**) 值。

当您将四个通道（红色、绿色、蓝色和 **Alpha**）与 **BitmapData.copyChannel()** 方法或 **DisplacementMapFilter.componentX** 属性和 **DisplacementMapFilter.componentY** 属性一起使用时，这些通道以数字形式表示，如下所示：

- 1（红色）
- 2（绿色）
- 4（蓝色）
- 8 (**Alpha**)

您可以通过使用 **MovieClip.attachBitmap()** 方法将 **BitmapData** 对象附加到 **MovieClip** 对象。

您可以通过使用 **MovieClip.beginBitmapFill()** 方法用 **BitmapData** 对象来填充影片剪辑中的区域。

**BitmapData** 对象的最大宽度和高度为 2880 像素。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[attachBitmap \(MovieClip.attachBitmap 方法\)](#)，[beginBitmapFill \(MovieClip.beginBitmapFill 方法\)](#)

属性摘要

修饰符	属性	说明
	height:Number [ 只读 ]	位图图像的高度，以像素为单位。
	rectangle:Rectangle [ 只读 ]	定义位图图像大小和位置的矩形。
	transparent:Boolean [ 只读 ]	定义位图图像是否支持每个像素具有不同的透明度。
	width:Number [ 只读 ]	位图图像的宽度，以像素为单位。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

构造函数摘要

签名	说明
BitmapData(width:Number, height:Number, [transparent:Boolean], [fillColor:Number])	创建一个具有指定的宽度和高度的 BitmapData 对象。

方法摘要

修饰符	签名	说明
	applyFilter(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, filter:BitmapFilter) : Number	取得一个源图像和一个滤镜对象，并生成过滤的图像。
	clone() : BitmapData	返回一个新的 BitmapData 对象，它是对原始实例的克隆，带有与所含位图完全相同的副本。
	colorTransform(rect:Rectangle, colorTransform:ColorTransform) : Void	通过使用 ColorTransform 对象调整位图图像的指定区域中的颜色值。
	copyChannel(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, sourceChannel:Number, destChannel:Number) : Void	将数据从另一 BitmapData 对象或当前 BitmapData 对象的一个通道传输到当前 BitmapData 对象的某个通道中。

修饰符	签名	说明
	<code>copyPixels(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [alphaBitmap:BitmapData], [alphaPoint:Point], [mergeAlpha:Boolean]) : Void</code>	为没有拉伸、旋转或色彩效果的图像之间的像素处理提供一个快速例程。
	<code>dispose() : Void</code>	释放用来存储 BitmapData 对象的内存。
	<code>draw(source:Object, [matrix:Matrix], [colorTransform:ColorTransform], [blendMode:Object], [clipRect:Rectangle], [smooth:Boolean]) : Void</code>	使用 Flash Player 矢量呈现器在目标图像上绘制源图像或影片剪辑。
	<code>fillRect(rect:Rectangle, color:Number) : Void</code>	使用指定的 ARGB 颜色填充一个矩形像素区域。
	<code>floodFill(x:Number, y:Number, color:Number) : Void</code>	在图像上执行倾倒填充操作，从 (x, y) 坐标开始，填充一种特定的颜色。
	<code>generateFilterRect(sourceRect:Rectangle, filter:BitmapFilter) : Rectangle</code>	确定 applyFilter() 方法调用影响的目标矩形，假定为 BitmapData 对象、源矩形和滤镜对象。
	<code>getColorBoundsRect(mask:Number, color:Number, [findColor:Boolean]) : Rectangle</code>	确定位图图像中完全包含指定颜色的所有像素的矩形区域。
	<code>getPixel(x:Number, y:Number) : Number</code>	返回一个整数，它表示特定点 (x, y) 上的 BitmapData 对象的 RGB 像素值。
	<code>getPixel32(x:Number, y:Number) : Number</code>	返回一个 ARGB 颜色值，它包含 Alpha 通道数据和 RGB 数据。
	<code>hitTest(firstPoint:Point, firstAlphaThreshold:Number, secondObject:Object, [secondBitmapPoint:Point], [secondAlphaThreshold:Number]) : Boolean</code>	在一个位图图像与一个点、矩形或其它位图图像之间执行像素级的点击检测。
static	<code>loadBitmap(id:String) : BitmapData</code>	返回一个新的 BitmapData 对象，它包含由库中指定的链接 ID 标识的元件的位图图像表示形式。

修饰符	签名	说明
	<code>merge(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, redMult:Number, greenMult:Number, blueMult:Number, alphaMult:Number) : Void</code>	对每个通道执行从源图像向目标图像的混合。
	<code>noise(randomSeed:Number, [low:Number], [high:Number], [channelOptions:Number], [grayScale:Boolean]) : Void</code>	使用表示随机杂点的像素填充图像。
	<code>paletteMap(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [redArray:Array], [greenArray:Array], [blueArray:Array], [alphaArray:Array]) : Void</code>	重新映射一个具有最多四组调色板数据（每个通道一组）的图像中的颜色通道值。
	<code>perlinNoise(baseX:Number, baseY:Number, numOctaves:Number, randomSeed:Number, stitch:Boolean, fractalNoise:Boolean, [channelOptions:Number], [grayScale:Boolean], [offsets:Object]) : Void</code>	生成 Perlin 杂点图像。
	<code>pixelDissolve(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, [randomSeed:Number], [numberOfPixels:Number], [fillColor:Number]) : Number</code>	执行源图像到目标图像的像素溶解，或使用同一图像执行像素溶解。
	<code>scroll(x:Number, y:Number) : Void</code>	将图像滚动特定数量 (x, y) 的像素。
	<code>setPixel(x:Number, y:Number, color:Number) : Void</code>	设置 BitmapData 对象单个像素的颜色。
	<code>setPixel32(x:Number, y:Number, color:Number) : Void</code>	设置 BitmapData 对象单个像素的颜色和 Alpha 透明度值。
	<code>threshold(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point, operation:String, threshold:Number, [color:Number], [mask:Number], [copySource:Boolean]) : Number</code>	根据指定的阈值测试图像中的像素值，并将通过测试的像素设置为新的颜色值。



继承自 Object 类的方法

---

```
addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)
```

---

## applyFilter (BitmapData.applyFilter 方法)

```
public applyFilter(sourceBitmap:BitmapData, sourceRect:Rectangle,  
    destPoint:Point, filter:BitmapFilter) : Number
```

取得一个源图像和一个滤镜对象，并生成过滤的图像。

此方法依赖于内置滤镜对象的行为，该对象的代码可确定受输入源矩形影响的目标矩形。

应用滤镜后，结果图像可能会大于输入图像。例如，如果使用 **BlurFilter** 类来模糊源矩形 (50,50,100,100)，并且目标点为 (10,10)，则在目标图像中更改的区域将会由于该模糊处理而大于 (10,10,60,60)。这会在 applyFilter() 调用过程中在内部发生。

如果 sourceBitmapData 参数的 sourceRect 参数是内部区域，如 200 x 200 图像中的 (50,50,100,100)，则滤镜会使用 sourceRect 参数外部的源像素来生成目标矩形。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**sourceBitmap:**flash.display.BitmapData - 要使用的输入位图图像。源图像可以是另一不同的 BitmapData 对象，也可以指当前 BitmapData 实例。

**sourceRect:**flash.geom.Rectangle - 一个矩形，定义要用作输入的源图像的区域。

**destPoint:**flash.geom.Point - 目标图像（当前 BitmapData 实例）中与源矩形的左上角对应的点。

**filter:**flash.filters.BitmapFilter - 用于执行过滤操作的滤镜对象。每种滤镜都有某些要求，如下所示：

- **BlurFilter** - 此滤镜可使用不透明或透明的源图像和目标图像。如果这两种图像的格式不匹配，则在过滤过程中生成的源图像副本将与目标图像的格式匹配。
- **BevelFilter**、**DropShadowFilter**、**GlowFilter** - 这些滤镜的目标图像必须是透明图像。调用 **DropShadowFilter** 或 **GlowFilter** 会创建包含投影或发光的 Alpha 通道数据的图像。它不会在目标图像上创建投影。如果将这些滤镜中的任何滤镜用于不透明的目标图像，将返回错误代码值 -6。
- **ConvolutionFilter** - 此滤镜可使用不透明或透明的源图像和目标图像。
- **ColorMatrixFilter** - 此滤镜可使用不透明或透明的源图像和目标图像。

- **DisplacementMapFilter** - 此滤镜可以使用不透明或透明的源图像和目标图像，但源图像和目标图像的格式必须相同。

## 返回

Number - 一个数字，指示是否成功应用了滤镜。如果返回 **0**，则说明已成功应用了滤镜。如果返回一个负数，则说明在应用该滤镜的过程中出现了错误。

## 示例

下面的示例演示如何将斜角滤镜应用于 **BitmapData** 实例：

```
import flash.display.BitmapData;
import flash.filters.BevelFilter;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xFFFFFFFF);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF, .8,
    20, 20, 1, 3, "inner", false);

mc.onPress = function() {
    myBitmapData.applyFilter(myBitmapData, myBitmapData.rectangle, new
        Point(0, 0), filter);
}
```

## 另请参见

[BevelFilter](#) ([flash.filters.BevelFilter](#)), [BlurFilter](#) ([flash.filters.BlurFilter](#)), [ColorMatrixFilter](#) ([flash.filters.ColorMatrixFilter](#)), [ConvolutionFilter](#) ([flash.filters.ConvolutionFilter](#)), [DisplacementMapFilter](#) ([flash.filters.DisplacementMapFilter](#)), [DropShadowFilter](#) ([flash.filters.DropShadowFilter](#)), [GlowFilter](#) ([flash.filters.GlowFilter](#)), [filters](#) ([MovieClip.filters](#) 属性)

# BitmapData 构造函数

```
public BitmapData(width:Number, height:Number, [transparent:Boolean],
    [fillColor:Number])
```

创建一个具有指定的宽度和高度的 **BitmapData** 对象。如果为 `fillColor` 参数指定一个值，则位图中的每个像素都将设置为该颜色。

默认情况下，位图将创建为不透明的，除非您为 `transparent` 参数传递值 `true`。一旦创建不透明位图，就无法将其更改为透明位图。不透明位图中的每个像素仅使用 **24** 位的颜色通道信息。如果将位图定义为 `transparent`，则每个像素将使用 **32** 位的颜色通道信息，其中包括 **Alpha** 透明度通道。

**BitmapData** 对象的最大宽度和高度为 **2880** 像素。如果指定的宽度值或高度值大于 **2880**，则不会创建新实例。

可用性：ActionScript 1.0 ； Flash Player 8

## 参数

**width**:Number - 位图图像的宽度，以像素为单位。

**height**:Number - 位图图像的高度，以像素为单位。

**transparent**:Boolean [ 可选 ] - 指定位图图像是否支持每个像素具有不同的透明度。默认值为 `true`（透明）。要创建完全透明的位图，请将 `transparent` 参数的值设置为 `true`，将 `fillColor` 参数的值设置为 `0x00000000`（或者为 `0`）。

**fillColor**:Number [ 可选 ] - 用于填充位图图像区域的 **32** 位 **ARGB** 颜色值。默认值为 `0xFFFFFFFF`（纯白色）。

## 示例

下面的示例创建一个新的 **BitmapData** 对象。此示例中的值是 `transparent` 参数和 `fillColor` 参数的默认值；您可以调用不含这些参数的构造函数并获得相同的结果。

```
import flash.display.BitmapData;

var width:Number = 100;
var height:Number = 80;
var transparent:Boolean = true;
var fillColor:Number = 0xFFFFFFFF;

var bitmap_1:BitmapData = new BitmapData(width, height, transparent,
    fillColor);

trace(bitmap_1.width); // 100
trace(bitmap_1.height); // 80
trace(bitmap_1.transparent); // true

var bitmap_2:BitmapData = new BitmapData(width, height);
```

```
trace(bitmap_2.width); // 100
trace(bitmap_2.height); // 80
trace(bitmap_2.transparent); // true
```

## clone (BitmapData.clone 方法)

```
public clone() : BitmapData
```

返回一个新的 **BitmapData** 对象，它是对原始实例的克隆，带有与所含位图完全相同的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.display.BitmapData - 一个新 **BitmapData** 对象，它与原始对象相同。

### 示例

下面的示例将创建三个 **BitmapData** 对象并对它们进行比较。您可以通过使用 **BitmapData** 构造函数创建 **bitmap\_1** 实例。接着创建 **bitmap\_2** 实例，方法是将其设置为等于 **bitmap\_1**。然后通过克隆 **bitmap\_1** 创建 **clonedBitmap** 实例。请注意，尽管 **bitmap\_2** 等于 **bitmap\_1**，但 **clonedBitmap** 却不等于 **bitmap\_1**，尽管它们包含相同的值。

```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(100, 80, false, 0x000000);
var bitmap_2:BitmapData = bitmap_1;
var clonedBitmap:BitmapData = bitmap_1.clone();

trace(bitmap_1 == bitmap_2); // true
trace(bitmap_1 == clonedBitmap); // false

for(var i in bitmap_1) {
    trace(">> " + i + ": " + bitmap_1[i]);
    // >> generateFilterRect: [type Function]
    // >> dispose: [type Function]
    // >> clone: [type Function]
    // >> copyChannel: [type Function]
    // >> noise: [type Function]
    // >> merge: [type Function]
    // >> paletteMap: [type Function]
    // >> hitTest: [type Function]
    // >> colorTransform: [type Function]
    // >> perlinNoise: [type Function]
    // >> getColorBoundsRect: [type Function]
    // >> floodFill: [type Function]
    // >> setPixel32: [type Function]
    // >> getPixel32: [type Function]
    // >> pixelDissolve: [type Function]
```

```

// >> draw: [type Function]
// >> threshold: [type Function]
// >> scroll: [type Function]
// >> applyFilter: [type Function]
// >> copyPixels: [type Function]
// >> fillRect: [type Function]
// >> setPixel: [type Function]
// >> getPixel: [type Function]
// >> transparent: false
// >> rectangle: (x=0, y=0, w=100, h=80)
// >> height: 80
// >> width: 100
}

for(var i in clonedBitmap) {
  trace(">> " + i + ": " + clonedBitmap[i]);
  // >> generateFilterRect: [type Function]
  // >> dispose: [type Function]
  // >> clone: [type Function]
  // >> copyChannel: [type Function]
  // >> noise: [type Function]
  // >> merge: [type Function]
  // >> paletteMap: [type Function]
  // >> hitTest: [type Function]
  // >> colorTransform: [type Function]
  // >> perlinNoise: [type Function]
  // >> getColorBoundsRect: [type Function]
  // >> floodFill: [type Function]
  // >> setPixel32: [type Function]
  // >> getPixel32: [type Function]
  // >> pixelDissolve: [type Function]
  // >> draw: [type Function]
  // >> threshold: [type Function]
  // >> scroll: [type Function]
  // >> applyFilter: [type Function]
  // >> copyPixels: [type Function]
  // >> fillRect: [type Function]
  // >> setPixel: [type Function]
  // >> getPixel: [type Function]
  // >> transparent: false
  // >> rectangle: (x=0, y=0, w=100, h=80)
  // >> height: 80
  // >> width: 100
}

```

为了进一步说明 `bitmap_1`、`bitmap_2` 和 `clonedBitmap` 之间的关系，下面的示例修改了 `bitmap_1` 在 (1, 1) 处的像素值。通过修改在 (1, 1) 处的像素值，说明 `clone()` 方法是根据 `bitmap_1` 实例的值而不是引用这些值来创建实例的。

```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(100, 80, false, 0x0000000);
var bitmap_2:BitmapData = bitmap_1;
var clonedBitmap:BitmapData = bitmap_1.clone();

trace(bitmap_1.getPixel32(1, 1)); // -16777216
trace(bitmap_2.getPixel32(1, 1)); // -16777216
trace(clonedBitmap.getPixel32(1, 1)); // -16777216

bitmap_1.setPixel32(1, 1, 0xFFFFFFFF);

trace(bitmap_1.getPixel32(1, 1)); // -1
trace(bitmap_2.getPixel32(1, 1)); // -1
trace(clonedBitmap.getPixel32(1, 1)); // -16777216
```

## colorTransform (BitmapData.colorTransform 方法)

```
public colorTransform(rect:Rectangle, colorTransform:ColorTransform) : Void
```

通过使用 **ColorTransform** 对象调整位图图像的指定区域中的颜色值。如果矩形与位图图像的边界匹配，则此方法将转换整个图像的颜色值。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**rect**:flash.geom.Rectangle - 一个 **Rectangle** 对象，它定义其中应用了 **ColorTransform** 对象的图像区域。

**colorTransform**:flash.geom.ColorTransform - 一个 **ColorTransform** 对象，它描述要应用的颜色转换值。

### 示例

下面的示例演示如何将颜色转换操作应用于 **BitmapData** 实例。

```
import flash.display.BitmapData;
import flash.geom.ColorTransform;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
```

```
mc.onPress = function() {  
    myBitmapData.colorTransform(myBitmapData.rectangle, new ColorTransform(1,  
    0, 0, 1, 255, 0, 0, 0));  
}
```

另请参见

[ColorTransform \(flash.geom.ColorTransform\)](#), [Rectangle \(flash.geom.Rectangle\)](#)

## copyChannel (BitmapData.copyChannel 方法)

```
public copyChannel(sourceBitmap:BitmapData, sourceRect:Rectangle,  
    destPoint:Point, sourceChannel:Number, destChannel:Number) : Void
```

将数据从另一 **BitmapData** 对象或当前 **BitmapData** 对象的一个通道传输到当前 **BitmapData** 对象的某个通道中。目标 **BitmapData** 对象的其它通道中的所有数据都将被保留。

源通道值和目标通道值可以是下列值之一：

- 1 (红色)
- 2 (绿色)
- 4 (蓝色)
- 8 (Alpha)

可用性：ActionScript 1.0 ； Flash Player 8

### 参数

**sourceBitmap**:flash.display.BitmapData - 要使用的输入位图图像。源图像可以是另一个 **BitmapData** 对象，也可以引用当前 **BitmapData** 对象。

**sourceRect**:flash.geom.Rectangle - 源 **Rectangle** 对象。如果您仅希望从位图中较小的区域复制通道数据，请指定一个小于 **BitmapData** 对象整体大小的源矩形。

**destPoint**:flash.geom.Point - 目标 **Point** 对象，它表示将要在其中放置新通道数据的矩形区域的左上角。如果您希望将通道数据从目标图像中的一个区域复制到其它区域，请指定一个 (0,0) 以外的点。

**sourceChannel**:Number - 源通道。使用集合 (1,2,4,8) 中的值，这些值分别表示红色、绿色、蓝色和 Alpha 通道。

**destChannel**:Number - 目标通道。使用集合 (1,2,4,8) 中的值，这些值分别表示红色、绿色、蓝色和 Alpha 通道。

## 示例

下面的示例演示如何将源 **ARGB** 通道从 `BitmapData` 对象重新复制到处于其它位置的该对象上：

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

mc.onPress = function() {
    myBitmapData.copyChannel(myBitmapData, new Rectangle(0, 0, 50, 80), new
        Point(51, 0), 3, 1);
}
```

另请参见

[Rectangle \(flash.geom.Rectangle\)](#)

## copyPixels (BitmapData.copyPixels 方法)

```
public copyPixels(sourceBitmap:BitmapData, sourceRect:Rectangle,
    destPoint:Point, [alphaBitmap:BitmapData], [alphaPoint:Point],
    [mergeAlpha:Boolean]) : Void
```

为没有拉伸、旋转或色彩效果的图像之间的像素处理提供一个快速例程。此方法在目标 **BitmapData** 对象的目标点将源图像的矩形区域复制为同样大小的矩形区域。

如果包括 `alphaBitmap` 参数和 `alphaPoint` 参数，则您可以将另一个图像用作源图像的 **Alpha** 源。如果源图像具有 **Alpha** 数据，则这两组 **Alpha** 数据都用于将源图像中的像素组合到目标图像中。`alphaPoint` 参数是 **Alpha** 图像中与源矩形左上角对应的点。源图像和 **Alpha** 图像交叉区域之外的任何像素都不会被复制到目标图像。

`mergeAlpha` 属性控制在将透明图像复制到另一透明图像时是否使用 **Alpha** 通道。若要仅复制像素（没有使用 **Alpha**），请将 `false` 属性设置为 `mergeAlpha`。然后，所有像素就会从源图像复制到目标图像。默认情况下，`mergeAlpha` 属性为 `true`。

可用性：ActionScript 1.0 ； Flash Player 8



## 参数

**sourceBitmap:**flash.display.BitmapData - 要从中复制像素的输入位图图像。源图像可以是另一个 **BitmapData** 实例，或可以指当前 **BitmapData** 实例。

**sourceRect:**flash.geom.Rectangle - 一个矩形，定义要用作输入的源图像的区域。

**destPoint:**flash.geom.Point - 目标点，表示将在其中放置新像素的矩形区域的左上角。

**alphaBitmap:**flash.display.BitmapData [ 可选 ] - 第二个 **Alpha BitmapData** 对象源。

**alphaPoint:**flash.geom.Point [ 可选 ] - **Alpha BitmapData** 源中与 **sourceRect** 参数的左上角对应的点。

**mergeAlpha:**Boolean [ 可选 ] - 布尔值：若要使用 **Alpha** 通道，请将该值设置为 **true**。若要复制不含 **Alpha** 通道的像素，请将该值设置为 **false**。

## 示例

下面的示例演示如何将像素从一个 **BitmapData** 实例复制到另一个实例。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var bitmapData_1:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);
var bitmapData_2:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_1.attachBitmap(bitmapData_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_2.attachBitmap(bitmapData_2, this.getNextHighestDepth());
mc_2._x = 101;

mc_1.onPress = function() {
    bitmapData_2.copyPixels(bitmapData_1, new Rectangle(0, 0, 50, 80), new
    Point(51, 0));
}

mc_2.onPress = function() {
    bitmapData_1.copyPixels(bitmapData_2, new Rectangle(0, 0, 50, 80), new
    Point(51, 0));
}
```

## dispose (BitmapData.dispose 方法)

```
public dispose() : Void
```

释放用来存储 **BitmapData** 对象的内存。

对图像调用此方法时，该图像的宽度和高度将设置为 0。在释放 **BitmapData** 对象的内存后，对实例的方法和属性访问调用将失败，返回值为 -1。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例演示如何释放 **BitmapData** 实例的内存，从而产生一个清除的实例。

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

mc.onPress = function() {
    myBitmapData.dispose();

    trace(myBitmapData.width); // -1
    trace(myBitmapData.height); // -1
    trace(myBitmapData.transparent); // -1
}
```

## draw (BitmapData.draw 方法)

```
public draw(source:Object, [matrix:Matrix], [colorTransform:ColorTransform],
    [blendMode:Object], [clipRect:Rectangle], [smooth:Boolean]) : Void
```

使用 **Flash Player** 矢量呈现器在目标图像上绘制源图像或影片剪辑。您可以使用 **Matrix**、**ColorTransform**、**BlendMode** 对象以及目标 **Rectangle** 对象来控制呈现的执行方式。或者，您也可以指定缩放时是否应对位图进行平滑处理。这只适用于当源对象是 **BitmapData** 对象时的情况。

此方法直接与如何在创作工具界面中使用对象的标准矢量呈现器来绘制图像相对应。

源 **MovieClip** 对象不对此调用使用其任何舞台中转换。该源 **MovieClip** 对象会被视为存在于库或文件中，没有矩阵转换、没有颜色转换，也没有混合模式。如果您希望通过使用影片剪辑自身的 **transform** 属性来绘制影片剪辑，则可以使用它的 **Transform** 对象来传递各种 **transformation** 属性。

可用性：ActionScript 1.0；Flash Player 8

## 参数

**source:**Object - 要绘制的 **BitmapData** 对象。

**matrix:**flash.geom.Matrix [ 可选 ] - 一个 **Matrix** 对象，用于缩放、旋转或转换位图的坐标。如果没有提供任何对象，位图图像将不转换。如果您必须传递此参数但又不希望转换图像，则可以将此参数设置为使用默认 new Matrix() 构造函数创建的恒等矩阵。

**colorTransform:**flash.geom.ColorTransform [ 可选 ] - 一个 **ColorTransform** 对象，用于调整位图的颜色值。如果没有提供任何对象，位图图像的颜色将不转换。如果您必须传递此参数但又不希望转换图像，则可以将此参数设置为使用默认 new ColorTransform() 构造函数创建的 **ColorTransform** 对象。

**blendMode:**Object [ 可选 ] - 一个 **BlendMode** 对象。

**clipRect:**flash.geom.Rectangle [ 可选 ] 一个 **Rectangle** 对象。如果您未提供此值，将不会发生任何剪裁。

**smooth:**Boolean [ 可选 ] - 布尔值，确定缩放时是否要对 **BitmapData** 对象进行平滑处理。默认值为 false。

## 示例

下面的示例演示如何将源 **MovieClip** 实例绘制为 **BitmapData** 对象。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc_1:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_1.attachBitmap(myBitmapData, this.getNextHighestDepth());

var mc_2:MovieClip = createRectangle(50, 40, 0xFF0000);
mc_2._x = 101;

var myMatrix:Matrix = new Matrix();
myMatrix.rotate(Math.PI/2);

var translateMatrix:Matrix = new Matrix();
translateMatrix.translate(70, 15);

myMatrix.concat(translateMatrix);

var myColorTransform:ColorTransform = new ColorTransform(0, 0, 1, 1, 0, 0,
    255, 0);
var blendMode:String = "normal";
```

```

var myRectangle:Rectangle = new Rectangle(0, 0, 100, 80);
var smooth:Boolean = true;

mc_1.onPress = function() {
    myBitmapData.draw(mc_2, myMatrix, myColorTransform, blendMode,
        myRectangle, smooth);
}

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}

```

## fillRect (BitmapData.fillRect 方法)

```
public fillRect(rect:Rectangle, color:Number) : Void
```

使用指定的 **ARGB** 颜色填充一个矩形像素区域。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 参数

**rect:**flash.geom.Rectangle - 要填充的矩形区域。

**color:**Number - 用于填充区域的 **ARGB** 颜色值。通常以十六进制格式指定 **ARGB** 颜色；例如， **0xFF336699**。

### 示例

下面的示例演示如何使用一种颜色填充由 BitmapData 中的 Rectangle 定义的区域。

```

import flash.display.BitmapData;
import flash.geom.Rectangle;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

mc.onPress = function() {
    myBitmapData.fillRect(new Rectangle(0, 0, 50, 40), 0x00FF0000);
}

```

另请参见

[Rectangle \(flash.geom.Rectangle\)](#)

## floodFill (BitmapData.floodFill 方法)

```
public floodFill(x:Number, y:Number, color:Number) : Void
```

在图像上执行倾倒填充操作，从 (x, y) 坐标开始，填充一种特定的颜色。floodFill() 方法类似于各种绘画程序中的“颜料桶”工具。该颜色是包含 Alpha 信息和颜色信息的 ARGB 颜色。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**x:**Number - 图像的 x 坐标。

**y:**Number - 图像的 y 坐标。

**color:**Number - 要用作填充的 ARGB 颜色。通常以十六进制格式指定 ARGB 颜色，例如 0xFF336699。

### 示例

下面的示例演示如何应用倾倒填充将一种颜色填充到图像中，填充的起始位置为用户在 **BitmapData** 对象中单击鼠标的位置。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

myBitmapData.fillRect(new Rectangle(0, 0, 50, 40), 0x00FF0000);

mc.onPress = function() {
    myBitmapData.floodFill(_xmouse, _ymouse, 0x000000FF);
}
```

## generateFilterRect (BitmapData.generateFilterRect 方法)

```
public generateFilterRect(sourceRect:Rectangle, filter:BitmapFilter) :  
    Rectangle
```

确定 `applyFilter()` 方法调用影响的目标矩形，假定为 **BitmapData** 对象、源矩形和滤镜对象。

例如，模糊滤镜通常影响大小比原始图像大的区域。由一个默认 **BlurFilter** 实例过滤的 **100 x 200** 像素图像，其中 `blurX = blurY = 4` 将生成一个目标矩形 `(-2,-2,104,204)`。`generateFilterRect()` 方法使您可以提前了解到此目标矩形的大小，以便您可以在执行过滤操作之前相应地调整目标图像的大小。

有些滤镜会基于源图像大小裁剪其目标矩形。例如，一个内部 `DropShadow` 不会生成比其源图像大的结果。在此 API 中，**BitmapData** 对象用作源范围而不是源 `rect` 参数。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 参数

**sourceRect:** `flash.geom.Rectangle` - 一个矩形，定义要用作输入的源图像的区域。

**filter:** `flash.filters.BitmapFilter` - 一个滤镜对象，用于计算目标矩形。

### 返回

`flash.geom.Rectangle` - 一个目标矩形，它是使用图像、`sourceRect` 参数和滤镜计算得到的。

### 示例

下面的示例演示如何确定 `applyfilter()` 方法影响的目标矩形：

```
import flash.display.BitmapData;  
import flash.filters.BevelFilter;  
import flash.geom.Rectangle;  
  
var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xCCCCCCCC);  
  
var filter:BevelFilter = new BevelFilter(5, 45, 0xFFFF00, .8, 0x0000FF, .8,  
    20, 20, 1, 3, "outter", false);  
  
var filterRect:Rectangle =  
    myBitmapData.generateFilterRect(myBitmapData.rectangle, filter);  
  
trace(filterRect); // (x=-31, y=-31, w=162, h=142)
```

## getColorBoundsRect (BitmapData.getColorBoundsRect 方法)

```
public getColorBoundsRect(mask:Number, color:Number, [findColor:Boolean]) :  
    Rectangle
```

确定位图图像中完全包含指定颜色的所有像素的矩形区域。

例如，如果您有一个源图像，并且希望确定包含非零 **Alpha** 通道的图像矩形，请传递 {mask: 0xFF000000, color: 0x00000000} 作为参数。在整个图像中搜索像素的范围，该像素满足条件 (value & mask) != color。若要确定图像周围的空白区域，请传递 {mask: 0xFFFFFFFF, color: 0xFFFFFFFF} 以查找非空白像素的范围。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 参数

**mask:**Number - 十六进制颜色值。

**color:**Number - 十六进制颜色值。

**findColor:**Boolean [可选] - 如果该值设置为 true，则返回图像中颜色值的范围。如果该值设置为 false，则返回图像中不存在此颜色的范围。默认值为 true。

### 返回

flash.geom.Rectangle - 指定颜色的图像区域。

### 示例

下面的示例演示如何确定位图图像中完全包含指定颜色的所有像素的矩形区域：

```
import flash.display.BitmapData;  
import flash.geom.Rectangle;  
  
var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);  
  
var mc:MovieClip = this.createEmptyMovieClip("mc",  
    this.getNextHighestDepth());  
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());  
myBitmapData.fillRect(new Rectangle(0, 0, 50, 40), 0x00FF0000);  
  
mc.onPress = function() {  
    var colorBoundsRect:Rectangle =  
        myBitmapData.getColorBoundsRect(0x00FFFFFF, 0x00FF0000, true);  
    trace(colorBoundsRect); // (x=0, y=0, w=50, h=40)  
}
```

## getPixel (BitmapData.getPixel 方法)

`public getPixel(x:Number, y:Number) : Number`

返回一个整数，它表示特定点 (x, y) 上的 **BitmapData** 对象的 **RGB** 像素值。`getPixel()` 方法将返回一个未经过相乘的像素值。没有返回任何 **Alpha** 信息。

**BitmapData** 对象中的所有像素都作为预乘颜色值进行存储。预乘图像像素具有已经与 **Alpha** 数据相乘的红色、绿色和蓝色通道值。例如，如果 **Alpha** 值为 0，则 **RGB** 通道的值也为 0，与它们未经过相乘的值无关。

这种丢失数据的情况可能会在执行操作时导致一些问题。所有 **Flash Player** 方法都采用并返回未经过相乘的值。内部像素表示形式在其作为值返回之前是未经过相乘的。在设置操作过程中，在设置原始图像像素之前，像素值是经过预乘的。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**x**:Number - 像素的 **x** 位置。

**y**:Number - 像素的 **y** 位置。

### 返回

Number - 一个表示 **RGB** 像素值的数字。如果 (x, y) 坐标在图像范围以外，则返回 0。

### 示例

下面的示例使用 `getPixel()` 方法检索在特定 **x** 和 **y** 位置上的像素的 **RGB** 值。

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace("0x" + myBitmapData.getPixel(0, 0).toString(16)); // 0x000000
```

### 另请参见

[getPixel32 \(BitmapData.getPixel32 方法\)](#)



## getPixel32 (BitmapData.getPixel32 方法)

public getPixel32(x:Number, y:Number) : Number

返回一个 **ARGB** 颜色值，它包含 **Alpha** 通道数据和 **RGB** 数据。此方法与 `getPixel()` 方法类似，后者返回没有 **Alpha** 通道数据的 **RGB** 颜色。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**x**:Number - 像素的 **x** 位置。

**y**:Number - 像素的 **y** 位置。

### 返回

Number - 一个表示 **ARGB** 像素值的数字。如果 (**x**, **y**) 坐标在图像范围以外，则返回 **0**。如果位图创建为一个不透明的（非透明的）位图，此方法将返回错误代码 **-1**。

### 示例

下面的示例使用 `getPixel32()` 方法检索在特定 **x** 和 **y** 位置上的像素的 **ARGB** 值：

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xFFAACCEE);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

var alpha:String = (myBitmapData.getPixel32(0, 0) >> 24 & 0xFF).toString(16);
trace(">> alpha: " + alpha); // ff

var red:String = (myBitmapData.getPixel32(0, 0) >> 16 & 0xFF).toString(16);
trace(">> red: " + red); // aa

var green:String = (myBitmapData.getPixel32(0, 0) >> 8 & 0xFF).toString(16);
trace(">> green: " + green); // cc

var blue:String = (myBitmapData.getPixel32(0, 0) & 0xFF).toString(16);
trace(">> blue: " + blue); // ee

trace("0x" + alpha + red + green + blue); // 0xffaaccee
```

### 另请参见

[getPixel \(BitmapData.getPixel 方法\)](#)

## height (BitmapData.height 属性)

public height : Number [read-only]

位图图像的高度，以像素为单位。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例通过尝试设置 BitmapData 实例的 height 属性失败说明该属性是只读的:

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace(myBitmapData.height); // 80

myBitmapData.height = 999;
trace(myBitmapData.height); // 80
```

## hitTest (BitmapData.hitTest 方法)

public hitTest(firstPoint:Point, firstAlphaThreshold:Number,  
 secondObject:Object, [secondBitmapPoint:Point],  
 [secondAlphaThreshold:Number]) : Boolean

在一个位图图像与一个点、矩形或其它位图图像之间执行像素级的点击检测。在执行点击测试时，将不会考虑两个对象中任何一个对象的拉伸、旋转或其它变形。

如果某个图像是不透明图像，则此方法会将其视为完全不透明的矩形。两个图像必须是透明图像才能执行判断透明度的像素级点击测试。当您在测试两个透明图像时，Alpha threshold 参数将控制哪些 Alpha 通道值（从 0 到 255）将被视为是不透明的。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**firstPoint:**flash.geom.Point - 定义当前 BitmapData 实例中某像素位置的点。

**firstAlphaThreshold:**Number - 最大的 Alpha 通道值，此点击测试将其视为不透明的。

**secondObject:**Object - 一个 Rectangle、Point 或 BitmapData 对象。

**secondBitmapPoint:**flash.geom.Point [可选] - 定义第二个 BitmapData 对象中某像素位置的点。仅当 secondObject 的值是 BitmapData 对象时使用此参数。

**secondAlphaThreshold**:Number [可选] - 最大的 Alpha 通道值, 它在第二个 **BitmapData** 对象中被视为不透明的。仅当 **secondObject** 的值是 **BitmapData** 对象, 并且这两个 **BitmapData** 对象都为透明时使用此参数。

## 返回

Boolean - 一个布尔值。如果进行了点击, 则返回值 true ; 否则, 返回值 false。

## 示例

下面的示例演示如何确定 **BitmapData** 对象是否正在与 **MovieClip** 发生冲突。

```
import flash.display.BitmapData;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc_1:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_1.attachBitmap(myBitmapData, this.getNextHighestDepth());

var mc_2:MovieClip = createRectangle(20, 20, 0xFF0000);

var destPoint:Point = new Point(myBitmapData.rectangle.x,
    myBitmapData.rectangle.y);
var currPoint:Point = new Point();

mc_1.onEnterFrame = function() {
    currPoint.x = mc_2._x;
    currPoint.y = mc_2._y;
    if(myBitmapData.hitTest(destPoint, 255, currPoint)) {
        trace(">> Collision at y:" + currPoint.x + " and y:" + currPoint.y);
    }
}

mc_2.startDrag(true);

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## loadBitmap (BitmapData.loadBitmap 方法)

```
public static loadBitmap(id:String) : BitmapData
```

返回一个新的 **BitmapData** 对象，它包含由库中指定的链接 **ID** 标识的元件的位图图像表示形式。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**id:String** - 库中元件的链接 **ID**。

### 返回

**flash.display.BitmapData** - 元件的位图图像表示形式。

### 示例

下面的示例将从您的库中加载一个链接 **ID** 为 **libraryBitmap** 的位图。您必须将位图附加到 **MovieClip** 对象才能为其提供一个可视表示形式。

```
import flash.display.BitmapData;

var linkageId:String = "libraryBitmap";
var myBitmapData:BitmapData = BitmapData.loadBitmap(linkageId);
trace(myBitmapData instanceof BitmapData); // true

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
```

## merge (BitmapData.merge 方法)

```
public merge(sourceBitmap:BitmapData, sourceRect:Rectangle, destPoint:Point,
    redMult:Number, greenMult:Number, blueMult:Number, alphaMult:Number) :
    Void
```

对每个通道执行从源图像向目标图像的混合。下面是用于每个通道的公式：

```
new red dest = (red source * redMult) + (red dest * (256 - redMult) / 256;
```

**redMult**、**greenMult**、**blueMult** 和 **alphaMult** 值是用于每个颜色通道的乘数。它们的有效范围为 **0** 到 **256**。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**sourceBitmap:flash.display.BitmapData** - 要使用的输入位图图像。源图像可以是另一个 **BitmapData** 对象，或可以指当前 **BitmapData** 对象。

**sourceRect:**flash.geom.Rectangle - 一个矩形，定义要用作输入的源图像的区域。

**destPoint:**flash.geom.Point - 目标图像（当前 **BitmapData** 实例）中与源矩形的左上角对应的点。

**redMult:**Number - 一个要与 **Red** 通道值相乘的数字。

**greenMult:**Number - 一个要与 **Green** 通道值相乘的数字。

**blueMult:**Number - 一个要与 **Blue** 通道值相乘的数字。

**alphaMult:**Number - 一个要与 **Alpha** 透明度值相乘的数字。

## 示例

下面的示例演示如何合并 BitmapData 的两个不同部分。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var bitmapData_1:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);
var bitmapData_2:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_1.attachBitmap(bitmapData_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_2.attachBitmap(bitmapData_2, this.getNextHighestDepth());
mc_2._x = 101;

mc_1.onPress = function() {
    bitmapData_1.merge(bitmapData_2, new Rectangle(0, 0, 50, 40), new
        Point(25, 20), 128, 0, 0, 0);
}
```

## noise（BitmapData.noise 方法）

```
public noise(randomSeed:Number, [low:Number], [high:Number],
    [channelOptions:Number], [grayScale:Boolean]) : Void
```

使用表示随机杂点的像素填充图像。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**randomSeed:**Number - 要使用的随机种子。

**low:**Number [ 可选 ] - 为每个通道生成的最低值（0 到 255）。默认值为 0。

**high:**Number [ 可选 ] - 为每个通道生成的最高值（0 到 255）。默认值为 255。

**channelOptions:**Number [ 可选 ] - 一个数字，可以是四个颜色通道值的任意组合：1（红色）、2（绿色）、4（蓝色）和 8 (Alpha)。您可以使用逻辑 **OR** 运算符 | 来组合通道值。默认值为 (1 | 2 | 4)。

**grayScale:**Boolean [ 可选 ] - 一个布尔值。如果该值为 true，则会通过将所有颜色通道设置为相同的值来创建一个灰度图像。将此参数设置为 true 不会影响 Alpha 通道的选择。默认值为 false。

## 示例

下面的示例演示如何对彩色位图和黑白位图的 **BitmapData** 对象应用像素杂点。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var bitmapData_1:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);
var bitmapData_2:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_1.attachBitmap(bitmapData_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_2.attachBitmap(bitmapData_2, this.getNextHighestDepth());
mc_2._x = 101;

mc_1.onPress = function() {
    bitmapData_1.merge(bitmapData_2, new Rectangle(0, 0, 50, 40), new
        Point(25, 20), 128, 0, 0, 0);
}

mc_1.onPress = function() {
    bitmapData_1.noise(128, 0, 255, 1, true);
}

mc_2.onPress = function() {
    bitmapData_2.noise(128);
}
```

## paletteMap (BitmapData.paletteMap 方法)

```
public paletteMap(sourceBitmap:BitmapData, sourceRect:Rectangle,  
    destPoint:Point, [redArray:Array], [greenArray:Array], [blueArray:Array],  
    [alphaArray:Array]) : Void
```

重新映射一个具有最多四组调色板数据（每个通道一组）的图像中的颜色通道值。

**Flash Player** 使用以下公式生成结果图像。

计算了红、绿、蓝和 **Alpha** 值后，使用标准 32 位整数算法将它们相加在一起。每个像素的红色、绿色、蓝色和 **Alpha** 通道值被分别提取为一个 0 到 255 的值。使用这些值在相应的数组中查找新的颜色值：redArray、greenArray、blueArray 和 alphaArray。这四个数组中的每一个都应包含 256 个值。在检索了所有四个新通道值之后，它们会被组合成一个应用于像素的标准 **ARGB** 值。

此方法可以支持跨通道效果。每个输入数组可以包含完整的 32 位值，并且在将这些值相加到一起时不会发生任何移位。此例程不支持按通道锁定。

如果没有为通道指定数组，则颜色通道会直接从源图像复制到目标图像。

您可以为多种效果（例如，常规调色板映射）使用此方法（采用一个通道并将其转换为假颜色图像）。您也可以为各种高级颜色操作算法（例如，灰度系数、曲线、级别和量化）使用此方法。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**sourceBitmap**:flash.display.BitmapData - 要使用的输入位图图像。源图像可以是另一个 BitmapData 对象，或可以指当前 BitmapData 对象。

**sourceRect**:flash.geom.Rectangle - 一个矩形，定义要用作输入的源图像的区域。

**destPoint**:flash.geom.Point - 目标图像（当前 BitmapData 对象）中与源矩形的左上角对应的点。

**redArray**:Array [可选] - 如果 redArray 不为 null，则 red = redArray[source red value] else red = source rect value。

**greenArray**:Array [可选] - 如果 greenArray 不为 null，则 green = greenArray[source green value] else green = source green value。

**blueArray**:Array [可选] - 如果 blueArray 不为 null，则 blue = blueArray[source blue value] else blue = source blue value。

**alphaArray**:Array [可选] - 如果 alphaArray 不为 null，则 alpha = alphaArray[source alpha value] else alpha = source alpha value。

## 示例

下面的示例演示如何使用“调色板”映射将一个 **BitmapData** 对象中的纯红色转换为绿色，并将纯绿色转换为红色。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth()); mc.attachBitmap(myBitmapData,
    this.getNextHighestDepth());

myBitmapData.fillRect(new Rectangle(51, 0, 50, 80), 0x0000FF00);

mc.onPress = function() {
    var redArray:Array = new Array(256);
    var greenArray:Array = new Array(256);

    for(var i = 0; i < 255; i++) {
        redArray[i] = 0x00000000;
        greenArray[i] = 0x00000000;
    }

    redArray[0xFF] = 0x0000FF00;
    greenArray[0xFF] = 0x00FF0000;

    myBitmapData.paletteMap(myBitmapData, new Rectangle(0, 0, 100, 40), new
    Point(0, 0), redArray, greenArray, null, null);
}
```

## perlinNoise (BitmapData.perlinNoise 方法)

```
public perlinNoise(baseX:Number, baseY:Number, numOctaves:Number,
    randomSeed:Number, stitch:Boolean, fractalNoise:Boolean,
    [channelOptions:Number], [grayScale:Boolean], [offsets:Object]) : Void
```

生成 **Perlin** 杂点图像。

**Perlin** 杂点生成算法会内插单个随机杂点函数（名为 **octaves**）并将它们组合成一个函数，该函数生成多个看起来很自然的随机杂点。就像音乐上的八音度，每个 **octave** 函数的频率是其前面一个 **octave** 函数频率的两倍。**Perlin** 杂点被描述为“杂点的碎片总和”，因为它将多组杂点数据与不同级别的细节组合在一起。

您可以使用 **Perlin** 杂点函数来模拟自然现象和风景，例如，木材纹理、云彩或山脉。在大多数情况下，**Perlin** 杂点函数的输出不会直接显示出来，而是用于增强其它图像并为其它图像提供伪随机变体。



简单的数字随机杂点函数通常生成具有粗糙的对比度点的图像。这种粗糙的对比度在自然界中通常是找不到的。**Perlin** 杂点算法混合了多个杂点函数，它们在不同的详细级别上进行操作。此算法在相邻的像素值间产生较小的变化。



Perlin 杂点算法是以 Ken Perlin 的名字命名的，他在为 1982 年的影片 *Tron* 生成了计算机图形后开发了该算法。1997 年，Perlin 因 Perlin 杂点函数而获得了科技成就学院奖。

可用性: ActionScript 1.0 ; Flash Player 8

## 参数

**baseX**:Number - 要在 **x** 方向上使用的频率。例如，若要生成大小适合 **64 x 128** 图像的杂点，请为 **baseX** 值传递 **64**。

**baseY**:Number - 要在 **y** 方向上使用的频率。例如，若要生成大小适合 **64 x 128** 图像的杂点，请为 **baseY** 值传递 **128**。

**numOctaves**:Number - 要组合以创建此杂点的 **octave** 函数或各个杂点函数的数目。**octave** 的数目越多，创建的图像将会具有越多的细节。**octave** 的数目越多，需要的处理时间也会越多。

**randomSeed**:Number - 要使用的随机种子数目。如果您保持使所有其它参数不变，可以通过改变随机种子值来生成不同的伪随机结果。**Perlin** 杂点函数是一个映射函数，不是真正的随机数生成函数，所以它会每次根据相同的随机种子创建相同的结果。

**stitch**:Boolean - 一个布尔值。如果该值为 **true**，则该方法将尝试平滑图像的转变边缘以创建无缝的纹理，用于作为位图填充进行平铺。

**fractalNoise**:Boolean - 一个布尔值。如果该值为 **true**，则该方法将生成碎片杂点；否则，它将生成湍流。带有湍流的图像具有可见的不连续性渐变，可以使用它处理更接近锐化的视觉效果，例如，火焰或海浪。

**channelOptions**:Number [ 可选 ] - 一个数字，它指示一个或多个颜色通道。若要创建此值，您可以使用这四个颜色通道值的任意一个或它们的组合：1（红色）、2（绿色）、4（蓝色）和 8（Alpha）。您可以通过使用逻辑 **OR** 运算符组合这些通道值；例如，您可以通过使用以下代码组合红色和绿色通道：1 | 2。

**grayScale**:Boolean [ 可选 ] - 一个布尔值。如果该值为 **true**，则通过将红色、绿色和蓝色通道的每一个值都设置为相同的值来创建一个灰度图像。如果此值设置为 **true**，则 **Alpha** 通道值将不会受到影响。默认值为 **false**。

**offsets**:Object [ 可选 ] - 与每个 **octave** 的 **x** 和 **y** 偏移量相对应的点数组。通过操作该偏移值，您可以平滑滚动 **perlinNoise** 图像的图层。偏移数组中的每个点将影响一个特定的 **octave** 杂点函数。

## 示例

下面的示例演示如何将 **Perlin** 杂点应用于 **BitmapData** 对象。

```
import flash.display.BitmapData;

var bitmapData_1:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);
var bitmapData_2:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_1.attachBitmap(bitmapData_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc_2.attachBitmap(bitmapData_2, this.getNextHighestDepth());
mc_2._x = 101;

mc_1.onPress = function() {
    var randomNum:Number = Math.floor(Math.random() * 10);
    bitmapData_1.perlinNoise(100, 80, 6, randomNum, false, true, 1, true,
        null);
}

mc_2.onPress = function() {
    var randomNum:Number = Math.floor(Math.random() * 10);
    bitmapData_2.perlinNoise(100, 80, 4, randomNum, false, false, 15, false,
        null);
}
```

## pixelDissolve (BitmapData.pixelDissolve 方法)

```
public pixelDissolve(sourceBitmap:BitmapData, sourceRect:Rectangle,
    destPoint:Point, [randomSeed:Number], [numberOfPixels:Number],
    [fillColor:Number]) : Number
```

执行源图像到目标图像的像素溶解，或使用同一图像执行像素溶解。**Flash Player** 使用 `randomSeed` 值生成随机像素溶解。函数的返回值必须在后续调用中传入才能继续进行像素溶解，直至完成。

如果源图像不等于目标图像，则会使用所有的属性将像素从源复制到目标。这就允许从空白图像溶解到完全填充的图像。

如果源图像和目标图像相等，则使用 `color` 参数填充像素。这就允许从完全填充的图像溶解掉。在此模式中，将会忽略目标 `point` 参数。

可用性：ActionScript 1.0；Flash Player 8

## 参数

**sourceBitmap:**flash.display.BitmapData - 要使用的输入位图图像。源图像可以是另一个不同的 **BitmapData** 对象，也可以指当前 **BitmapData** 实例。

**sourceRect:**flash.geom.Rectangle - 一个矩形，定义要用作输入的源图像的区域。

**destPoint:**flash.geom.Point - 目标图像（当前 **BitmapData** 实例）中与源矩形的左上角对应的点。

**randomSeed:**Number [ 可选 ] - 用于开始像素溶解的随机种子。默认值是 0。

**numberOfPixels:**Number [ 可选 ] - 默认值是源区域（宽度 x 高度）的 1/30。

**fillColor:**Number [ 可选 ] - 一个 **ARGB** 颜色值，用于填充其源值等于目标值的像素。默认值是 0。

## 返回

Number - 用于后续调用的新随机种子值。

## 示例

下面的示例使用 `pixelDissolve()` 一次溶解 40 个像素，直到已经更改了全部 8000 个像素的颜色，从而将 **BitmapData** 对象由灰色转换为红色：

```
import flash.display.BitmapData;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

mc.onPress = function() {
    var randomNum:Number = Math.floor(Math.random() * 10);
    dissolve(randomNum);
}

var intervalId:Number;
var totalDissolved:Number = 0;
var totalPixels:Number = 8000;

function dissolve(randomNum:Number) {
    var newNum:Number = myBitmapData.pixelDissolve(myBitmapData,
        myBitmapData.rectangle, new Point(0, 0), randomNum, 40, 0x00FF0000);
    clearInterval(intervalId);
    if(totalDissolved < totalPixels) {
        intervalId = setInterval(dissolve, 10, newNum);
    }
    totalDissolved += 40;
}
```

## rectangle (BitmapData.rectangle 属性)

`public rectangle : Rectangle [read-only]`

定义位图图像大小和位置的矩形。矩形的顶部和左侧为零；宽度和高度等于 **BitmapData** 对象的像素宽度和高度。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例通过尝试设置 **Bitmap** 实例的 `rectangle` 属性失败说明该属性是只读的：

```
import flash.display.BitmapData;
import flash.geom.Rectangle;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace(myBitmapData.rectangle); // (x=0, y=0, w=100, h=80)

myBitmapData.rectangle = new Rectangle(1, 2, 4, 8);
trace(myBitmapData.rectangle); // (x=0, y=0, w=100, h=80)
```

## scroll (BitmapData.scroll 方法)

`public scroll(x:Number, y:Number) : Void`

将图像滚动特定数量 (**x**, **y**) 的像素。滚动区域之外的边缘区域保持不变。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**x**:Number - 水平滚动量。

**y**:Number - 垂直滚动量。

## 示例

下面的示例说演示如何滚动 **BitmapData** 对象。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

myBitmapData.fillRect(new Rectangle(0, 0, 25, 80), 0x00FF0000);

mc.onPress = function() {
    myBitmapData.scroll(25, 0);
}
```

## setPixel (BitmapData.setPixel 方法)

```
public setPixel(x:Number, y:Number, color:Number) : Void
```

设置 **BitmapData** 对象单个像素的颜色。在此操作过程中将会保留图像像素的当前 **Alpha** 通道值。RGB 颜色参数的值被视为一个未经过相乘的颜色值。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**x**:Number - 像素值会更改的像素的 **x** 位置。

**y**:Number - 像素值会更改的像素的 **y** 位置。

**color**:Number - 要对像素设置的 **RGB** 颜色。

## 示例

下面的示例使用 **setPixel()** 方法将 **RGB** 值分配给在特定 **x** 和 **y** 位置上的像素。您可以通过拖放在已创建的位图上用 **0x000000** 颜色进行绘制。

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

mc.onPress = function() {
    this.onEnterFrame = sketch;
}
```

```
mc.onRelease = function() {
    delete this.onEnterFrame;
}

function sketch() {
    myBitmapData.setPixel(_xmouse, _ymouse, 0x000000);
}
```

另请参见

[getPixel](#) ([BitmapData.getPixel 方法](#)), [setPixel32](#) ([BitmapData.setPixel32 方法](#))

## setPixel32 (BitmapData.setPixel32 方法)

```
public setPixel32(x:Number, y:Number, color:Number) : Void
```

设置 **BitmapData** 对象单个像素的颜色和 **Alpha** 透明度值。此方法与 `setPixel()` 方法类似；主要差别在于 `setPixel32()` 方法采用包含 **Alpha** 通道信息的 **ARGB** 颜色值。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**x**:Number - 像素值会更改的像素的 **x** 位置。

**y**:Number - 像素值会更改的像素的 **y** 位置。

**color**:Number - 要对像素设置的 **ARGB** 颜色。如果您创建了一个不透明的（非透明的）位图，将忽略此颜色值的 **Alpha** 透明度部分。

### 示例

下面的示例使用 `setPixel32()` 方法将 **ARGB** 值分配给在特定 **x** 和 **y** 位置上的像素。您可以通过按住鼠标按钮并进行拖放，用不含 **Alpha** 值的 **0x000000** 颜色在已创建的位图上进行绘制。

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xFFCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

mc.onPress = function() {
    this.onEnterFrame = sketch;
}

mc.onRelease = function() {
    delete this.onEnterFrame;
}
```

```
function sketch() {  
    myBitmapData.setPixel32(_xmouse, _ymouse, 0x00000000);  
}
```

另请参见

[getPixel32 \(BitmapData.getPixel32 方法\)](#), [setPixel \(BitmapData.setPixel 方法\)](#)

## threshold (BitmapData.threshold 方法)

```
public threshold(sourceBitmap:BitmapData, sourceRect:Rectangle,  
    destPoint:Point, operation:String, threshold:Number, [color:Number],  
    [mask:Number], [copySource:Boolean]) : Number
```

根据指定的阈值测试图像中的像素值，并将通过测试的像素设置为新的颜色值。通过使用 threshold() 方法，您可以隔离和替换图像中的颜色范围，并对图像像素执行其它逻辑操作。

阈值测试的逻辑如下所示：

```
if ((pixelValue & mask) operation (threshold & mask)) then  
    set pixel to color  
  
else  
    if (copySource) then  
        set pixel to corresponding pixel value from sourceBitmap
```

operation 参数指定要用于阈值测试的比较运算符。例如，通过使用 “==”，您可以隔离图像中的特定颜色。或者通过使用 {operation: "<", mask: 0xFF000000, threshold: 0x7f000000, color: 0x00000000}，您可以将所有目标像素设置为在源图像像素的 Alpha 小于 0x7F 时是完全透明的。您可以将此技巧用于动画转变和其它效果。

可用性：ActionScript 1.0 ; Flash Player 8

## 参数

**sourceBitmap:**flash.display.BitmapData - 要使用的输入位图图像。源图像可以是另一个不同的 **BitmapData** 对象，也可以指当前 **BitmapData** 实例。

**sourceRect:**flash.geom.Rectangle - 一个矩形，定义要用作输入的源图像的区域。

**destPoint:**flash.geom.Point - 目标图像（当前 **BitmapData** 实例）中与源矩形的左上角对应的点。

**operation:**String - 下列比较运算符之一（作为字符串传递）：“<”、“<=”、“>”、“>=”、“==”、“!=”

**threshold:**Number - 测试像素时要比较的值，以查看该值是达到还是超过阈值。

**color:**Number [ 可选 ] - 阈值测试成功时对像素设置的颜色值。默认值为 0x00000000。

**mask:**Number [ 可选 ] - 用于隔离颜色成分的遮罩层。默认值为 0xFFFFFFFF。

**copySource:**Boolean [ 可选 ] - 一个布尔值。如果该值为 true，则源图像中的像素值将在阈值测试失败时复制到目标图像。如果为 false，则在阈值测试失败时不会复制源图像。默认值为 false。

## 返回

Number - 已更改的像素数目。

## 示例

下面的示例演示如何更改颜色值大于或等于特定阈值的像素的颜色值。

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());

myBitmapData.fillRect(new Rectangle(0, 0, 50, 80), 0x00FF0000);

mc.onPress = function() {
    myBitmapData.threshold(myBitmapData, new Rectangle(0, 0, 100, 40), new
        Point(0, 0), ">=", 0x00CCCCC, 0x000000FF, 0x00FF0000, false);
}
```



## transparent（BitmapData.transparent 属性）

public transparent : Boolean [read-only]

定义位图图像是否支持每个像素具有不同的透明度。仅当通过对 transparent 参数传入 true 来构造 **BitmapData** 对象时才可以设置此值。创建 **BitmapData** 对象后，可以通过查看 transparent 属性的值是否为 true 来检查该对象是否支持每个像素具有不同的透明度。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例通过尝试设置 Bitmap 实例的 transparent 属性失败说明该属性是只读的：

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace(myBitmapData.transparent); // false

myBitmapData.transparent = true;
trace(myBitmapData.transparent); // false
```

## width（BitmapData.width 属性）

public width : Number [read-only]

位图图像的宽度，以像素为单位。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例通过尝试设置 Bitmap 实例的 width 属性失败说明该属性是只读的：

```
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00CCCCC);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
trace(myBitmapData.width); // 100

myBitmapData.width = 999;
trace(myBitmapData.width); // 100
```

# BitmapFilter (flash.filters.BitmapFilter)



```
public class BitmapFilter
extends Object
```

所有图像滤镜效果的 **BitmapFilter** 基类。

**BevelFilter**、**BlurFilter**、**ColorMatrixFilter**、**ConvolutionFilter**、**DisplacementMapFilter**、**DropShadowFilter**、**GlowFilter**、**GradientBevelFilter** 和 **GradientGlowFilter** 类都扩展了 **BitmapFilter** 类。可以将这些滤镜效果应用于位图或 **MovieClip** 实例。

您只能为 **BitmapFilter** 类的上述子类创建子类。

可用性: **ActionScript 1.0** ; **Flash Player 8**

属性摘要

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 方法摘要

修饰符	签名	说明
	clone() : BitmapFilter	返回 BitmapFilter 对象，它是与原始 BitmapFilter 对象完全相同的副本。

继承自 **Object** 类的方法

```
addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)
```

## clone (BitmapFilter.clone 方法)

```
public clone() : BitmapFilter
```

返回 **BitmapFilter** 对象，它是与原始 **BitmapFilter** 对象完全相同的副本。

可用性：ActionScript 1.0；Flash Player 8

### 返回

flash.filters.BitmapFilter - 一个 **BitmapFilter** 对象。

## BlurFilter (flash.filters.BlurFilter)

```
Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.BlurFilter
```

```
public class BlurFilter
extends BitmapFilter
```

**BlurFilter** 类允许您将模糊视觉效果应用于 **Flash** 中的各种对象。模糊效果可以柔化图像的细​​节。您可以生成一些模糊效果，范围从创建一个柔化的、未聚焦的外观到高斯模糊（就像通过半透明玻璃查看图像一样的朦胧的外观）。当此滤镜的 `quality` 属性设置为 **1** 时，结果为柔化的、未聚焦的外观。当 `quality` 属性设置为 **3** 时，该属性接近高斯模糊滤镜。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时给影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤销该设置操作。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter` 会取得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作时对图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。

如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜支持舞台缩放。但是，它不支持常规缩放、旋转和倾斜。如果对象本身进行了缩放（`_xscale` 和 `_yscale` 不是 **100%**），则此滤镜效果不进行缩放。只有放大舞台时它才会缩放。

如果结果图像的宽度或高度将超过 **2880** 像素，则不应用滤镜。例如，如果您在放大一个大影片剪辑时使用了滤镜，则在结果图像超过 **2880** 像素的限制时，该滤镜将关闭。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[filters \(MovieClip.filters 属性\)](#), [cacheAsBitmap \(MovieClip.cacheAsBitmap 属性\)](#), [filters \(Button.filters 属性\)](#), [cacheAsBitmap \(Button.cacheAsBitmap 属性\)](#), [filters \(TextField.filters 属性\)](#), [applyFilter \(BitmapData.applyFilter 方法\)](#)

属性摘要

修饰符	属性	说明
	<code>blurX:Number</code>	水平模糊量。
	<code>blurY:Number</code>	垂直模糊量。
	<code>quality:Number</code>	执行模糊的次数。

继承自 `Object` 类的属性

[constructor \(Object.constructor 属性\)](#), [\\_\\_proto\\_\\_ \(Object.\\_\\_proto\\_\\_ 属性\)](#), [prototype \(Object.prototype 属性\)](#), [\\_\\_resolve \(Object.\\_\\_resolve 属性\)](#)

构造函数摘要

签名	说明
<code>BlurFilter([blurX:Number], [blurY:Number], [quality:Number])</code>	用指定参数初始化滤镜。

方法摘要

修饰符	签名	说明
	<code>clone() : BlurFilter</code>	返回此滤镜对象的副本。

继承自 `BitmapFilter` 类的方法

[clone \(BitmapFilter.clone 方法\)](#)

继承自 `Object` 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## BlurFilter 构造函数

```
public BlurFilter([blurX:Number], [blurY:Number], [quality:Number])
```

用指定参数初始化滤镜。默认值会创建一个柔化的、未聚焦的图像。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**blurX**:Number [可选] - 水平模糊量。有效值为 **0** 到 **255** (浮点值)。默认值为 **4**。作为 **2** 的乘方的值 (如 **2**、**4**、**8**、**16** 和 **32**) 经过了优化, 呈现速度比其它值更快。

**blurY**:Number [可选] - 垂直模糊量。有效值为 **0** 到 **255** (浮点值)。默认值为 **4**。作为 **2** 的乘方的值 (如 **2**、**4**、**8**、**16** 和 **32**) 经过了优化, 呈现速度比其它值更快。

**quality**:Number [可选] - 应用滤镜的次数。默认值为 **1**, 即表示低品质。值为 **2** 表示中等品质, 值为 **3** 表示高品质并且接近高斯模糊。

### 示例

下面的示例将实例化一个新的 BlurFilter 构造函数, 并将其应用于一个平面矩形形状:

```
import flash.filters.BlurFilter;
var rect:MovieClip = createRectangle(100, 100, 0x003366,
    "BlurFilterExample");

var blurX:Number = 30;
var blurY:Number = 30;
var quality:Number = 3;

var filter:BlurFilter = new BlurFilter(blurX, blurY, quality);
var filterArray:Array = new Array();
filterArray.push(filter);
rect.filters = filterArray;

function createRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    mc.beginFill(bgColor);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc._x = 20;
    mc._y = 20;
    return mc;
}
```

## blurX (BlurFilter.blurX 属性)

public blurX : Number

水平模糊量。有效值为从 0 到 255（浮点）。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将在用户单击 **MovieClip** 实例时更改该实例的 blurX 属性。

```
import flash.filters.BlurFilter;
var mc:MovieClip = createBlurFilterRectangle("BlurFilterBlurX");
mc.onRelease = function() {
    var filter:BlurFilter = this.filters[0];
    filter.blurX = 200;
    this.filters = new Array(filter);
}

function createBlurFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BlurFilter = new BlurFilter(30, 30, 2);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}
```

## blurY (BlurFilter.blurY 属性)

public blurY : Number

垂直模糊量。有效值为从 0 到 255（浮点）。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将在用户单击 **MovieClip** 实例时更改该实例的 blurY 属性。

```
import flash.filters.BlurFilter;
var mc:MovieClip = createBlurFilterRectangle("BlurFilterBlurY");
mc.onRelease = function() {
    var filter:BlurFilter = this.filters[0];
    filter.blurY = 200;
    this.filters = new Array(filter);
}

function createBlurFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:BlurFilter = new BlurFilter(30, 30, 2);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}
```

## clone (BlurFilter.clone 方法)

public clone() : BlurFilter

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.filters.BlurFilter - 具有与原始 **BlurFilter** 实例相同的所有属性的新 **BlurFilter** 实例。

### 示例

下面的示例创建三个 **BlurFilter** 对象并对它们进行比较。您可以通过使用 **BlurFilter** 构造函数创建 `filter_1` 对象。接着创建 `filter_2` 对象，方法是将其设置为等于 `filter_1`。然后通过克隆 `filter_1` 创建 `clonedFilter` 对象。请注意，尽管 `filter_2` 等效于 `filter_1`，但 `clonedFilter` 却不等效于 `filter_1`，尽管它们包含相同的值。

```
import flash.filters.BlurFilter;

var filter_1:BlurFilter = new BlurFilter(30, 30, 2);
var filter_2:BlurFilter = filter_1;
var clonedFilter:BlurFilter = filter_1.clone();

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> quality: 2
    // >> blurY: 30
    // >> blurX: 30
}

for(var i in clonedFilter) {
    trace(">> " + i + ": " + clonedFilter[i]);
    // >> clone: [type Function]
    // >> quality: 2
    // >> blurY: 30
    // >> blurX: 30
}
```



为了进一步说明 filter\_1、filter\_2 和 clonedFilter 之间的关系，下面的示例将修改 filter\_1 的 quality 属性。通过修改 quality，说明 clone() 方法是根据 filter\_1 的值而不是通过引用这些值来创建新实例的。

```
import flash.filters.BlurFilter;

var filter_1:BlurFilter = new BlurFilter(30, 30, 2);
var filter_2:BlurFilter = filter_1;
var clonedFilter:BlurFilter = filter_1.clone();

trace(filter_1.quality); // 2
trace(filter_2.quality); // 2
trace(clonedFilter.quality); // 2

filter_1.quality = 1;

trace(filter_1.quality); // 1
trace(filter_2.quality); // 1
trace(clonedFilter.quality); // 2
```

## quality（BlurFilter.quality 属性）

public quality : Number

执行模糊的次数。有效值为 0 到 15。默认值为 1，即表示低品质。值为 2 表示中等品质。值为 3 表示高品质，并且接近高斯模糊。

对于大多数应用，quality 的值为 1、2 或 3 就足够了。虽然您可以使用不超过 15 的其它数值来增加应用模糊的次数，从而获得更为模糊的效果，但请注意，该数值越高，呈现速度越慢。除了增加 quality 的值，增加 blurX 和 blurY 的值通常也可以获得类似的效果，而且呈现速度更快。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例创建一个矩形，并将 quality 值为 1 的模糊滤镜应用于该矩形。当您单击该矩形时，quality 值会增加到 3，而且矩形会变得更加模糊。

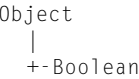
```
import flash.filters.BlurFilter;
var mc:MovieClip = createBlurFilterRectangle("BlurFilterQuality");
mc.onRelease = function() {
    var filter:BlurFilter = this.filters[0];
    filter.quality = 3;
    this.filters = new Array(filter);
}

function createBlurFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
```

```
var w:Number = 100;
var h:Number = 100;
rect.beginFill(0x003366);
rect.lineTo(w, 0);
rect.lineTo(w, h);
rect.lineTo(0, h);
rect.lineTo(0, 0);
rect._x = 20;
rect._y = 20;

var filter:BlurFilter = new BlurFilter(30, 30, 1);
var filterArray:Array = new Array();
filterArray.push(filter);
rect.filters = filterArray;
return rect;
}
```

# Boolean



```
public class Boolean
extends Object
```

**Boolean** 类是一种包装对象，它具有与标准 **JavaScript Boolean** 对象一样的功能。使用 **Boolean** 类可检索 **Boolean** 对象的基元数据类型或字符串表示形式。

必须使用构造函数 **new Boolean()** 创建 **Boolean** 对象后，才能调用其方法。

可用性: **ActionScript 1.0** ; **Flash Player 5**

## 属性摘要

继承自 **Object** 类的属性

[constructor](#) ([Object.constructor](#) 属性), [\\_\\_proto\\_\\_](#) ([Object.\\_\\_proto\\_\\_](#) 属性), [prototype](#) ([Object.prototype](#) 属性), [\\_\\_resolve](#) ([Object.\\_\\_resolve](#) 属性)

## 构造函数摘要

签名	说明
<code>Boolean([value: Object])</code>	创建 <b>Boolean</b> 对象。

方法摘要

修饰符	签名	说明
	toString() : String	返回 Boolean 对象的字符串表示形式 ("true" 或 "false")。
	valueOf() : Boolean	如果指定的 Boolean 对象的原始值类型为 true，则返回 true；否则返回 false。

继承自 Object 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## Boolean 构造函数

```
public Boolean([value:Object])
```

创建 **Boolean** 对象。如果省略 `value` 参数，则将 **Boolean** 对象的值初始化为 `false`。如果为 `value` 参数指定值，则该方法会计算它，并根据全局 `Boolean()` 函数中的规则以布尔值返回结果。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**value:Object** [ 可选 ] - 任何表达式。默认值为 `false`。

### 示例

下面的代码将创建一个名为 `myBoolean` 的新的空 **Boolean** 对象：

```
var myBoolean:Boolean = new Boolean();
```

## toString (Boolean.toString 方法)

public toString() : String

返回 **Boolean** 对象的字符串表示形式 ("true" 或 "false")。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

String - 一个字符串; "true" 或 "false"。

### 示例

此示例创建一个 **Boolean** 类型的变量, 并使用 toString() 将该值转换为一个字符串, 以便在 **trace** 语句中使用:

```
var myBool:Boolean = true;
trace("The value of the Boolean myBool is: " + myBool.toString());
myBool = false;
trace("The value of the Boolean myBool is: " + myBool.toString());
```

## valueOf (Boolean.valueOf 方法)

public valueOf() : Boolean

如果指定的 **Boolean** 对象的原始值类型为 **true**, 则返回 true; 否则返回 false。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

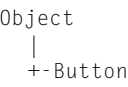
Boolean - 一个布尔值。

### 示例

下面的示例演示此方法是如何工作的, 同时还演示了新 **Boolean** 对象的原始值类型是 false:

```
var x:Boolean = new Boolean();
trace(x.valueOf()); // false
x = (6==3+3);
trace(x.valueOf()); // true
```

# Button



```
public class Button
extends Object
```

SWF 文件中的所有按钮元件都是 Button 对象的实例。您可在属性检查器中指定按钮实例名称，并通过 **ActionScript** 使用 **Button** 类的方法和属性来操纵按钮。按钮实例名称显示在影片浏览器中和“动作”面板的“插入目标路径”对话框中。

Button 类继承自 Object 类。

可用性: **ActionScript 1.0** ; **Flash Player 6**

另请参见

[Object](#)

## 属性摘要

修饰符	属性	说明
	<code>_alpha:Number</code>	由 <code>my_btn</code> 指定的按钮的 Alpha 透明度值。
	<code>blendMode:Object</code>	此按钮的混合模式。
	<code>cacheAsBitmap:Boolean</code>	如果设置为 <code>true</code> ，Flash Player 将缓存按钮的内部位图表示形式。
	<code>enabled:Boolean</code>	布尔值，指定按钮是否处于启用状态。
	<code>filters:Array</code>	索引数组，包含当前与按钮关联的每个滤镜对象。
	<code>_focusrect:Boolean</code>	布尔值，指定当按钮具有键盘焦点时，其四周是否有黄色矩形。
	<code>_height:Number</code>	按钮的高度，以像素为单位。
	<code>_highquality:Number</code>	自 Flash Player 7 后不推荐使用。不推荐使用此属性，而推荐使用 <code>Button._quality</code> 。指定当前 SWF 文件所应用的消除锯齿的级别。
	<code>menu:ContextMenu</code>	将 <code>ContextMenu</code> 对象 <code>contextMenu</code> 与按钮对象 <code>my_button</code> 相关联。
	<code>_name:String</code>	由 <code>my_btn</code> 指定的按钮的实例名称。
	<code>_parent:MovieClip</code>	对包含当前影片剪辑或对象的影片剪辑或对象的引用。
	<code>_quality:String</code>	属性（全局）；设置或检索用于 SWF 文件的呈现品质。
	<code>_rotation:Number</code>	按钮距其原始方向的旋转程度，以度为单位。

修饰符	属性	说明
	scale9Grid:Rectangle	为按钮定义九个缩放区域的矩形区域。
	_soundbuftime:Number	指定在声音开始进行流处理前预先缓冲的秒数的属性。
	tabEnabled:Boolean	指定 my_btn 是否包括在 Tab 键的自动排序中。
	tabIndex:Number	用于自定义 SWF 文件中对象的 Tab 键排序。
	_target:String [ 只读 ]	返回由 my_btn 指定的按钮实例的目标路径。
	trackAsMenu:Boolean	布尔值，指示其它按钮或影片剪辑是否可接收鼠标释放事件。
	_url:String [ 只读 ]	检索创建按钮的 SWF 文件的 URL。
	useHandCursor:Boolean	布尔值，当设置为 true（默认值）时，指示鼠标指针滑过按钮上方时是否显示手指形（手形光标）。
	_visible:Boolean	布尔值，指示由 my_btn 指定的按钮是否可见。
	_width:Number	按钮的宽度，以像素为单位。
	_x:Number	整数，用来设置按钮相对于父级影片剪辑的本地坐标的 x 坐标。
	_xmouse:Number [ 只读 ]	返回鼠标位置相对于按钮的 x 坐标。
	_xscale:Number	从按钮注册点开始应用的按钮水平缩放比例，以百分比表示。
	_y:Number	按钮相对于父级影片剪辑的本地坐标的 y 坐标。
	_ymouse:Number [ 只读 ]	指示鼠标位置相对于按钮的 y 坐标。
	_yscale:Number	从按钮注册点开始应用的按钮垂直缩放比例，以百分比表示。

继承自 Object 类的属性

constructor (Object.constructor 属性), \_\_proto\_\_ (Object.\_\_proto\_\_ 属性),  
prototype (Object.prototype 属性), \_\_resolve (Object.\_\_resolve 属性)

事件摘要

事件	说明
<code>onDragOut = function() {}</code>	当在按钮上单击鼠标按钮，然后将鼠标指针拖动到按钮之外时调用。
<code>onDragOver = function() {}</code>	当用户在按钮外部按下鼠标按钮，然后将鼠标指针拖动到按钮之上时调用。
<code>onKeyDown = function() {}</code>	当按钮具有键盘焦点而且按下某按键时调用。
<code>onKeyUp = function() {}</code>	当按钮具有输入焦点而且释放某按键时调用。
<code>onKillFocus = function(newFocus: Object) {}</code>	当按钮失去键盘焦点时调用。
<code>onPress = function() {}</code>	当按下按钮时调用。
<code>onRelease = function() {}</code>	当释放按钮时调用。
<code>onReleaseOutside = function() {}</code>	在这样的情况下调用：在鼠标指针位于按钮内部的情况下按下按钮，然后将鼠标指针移到该按钮外部并释放鼠标按钮。
<code>onRollOut = function() {}</code>	当鼠标指针移至按钮区域之外时调用。
<code>onRollOver = function() {}</code>	当鼠标指针移过按钮区域时调用。
<code>onSetFocus = function(oldFocus: Object) {}</code>	当按钮接收键盘焦点时调用。

方法摘要

修饰符	签名	说明
	<code>getDepth() : Number</code>	返回按钮实例的深度。

继承自 Object 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## \_alpha (Button.\_alpha 属性)

public \_alpha : Number

由 my\_btn 指定的按钮的 Alpha 透明度值。有效值为 0（完全透明）到 100（完全不透明）。默认值为 100。按钮的 \_alpha 设置为 0 时，其中的对象处于活动状态（即使这些对象不可见）。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

在用户单击名为 myBtn\_btn 的按钮时，以下代码将该按钮的 \_alpha 属性设置为 50%。首先，在舞台上添加按钮实例。然后，为它指定实例名称 myBtn\_btn。最后，在第 1 帧处于选中状态时，将以下代码放入“动作”面板中：

```
myBtn_btn.onRelease = function(){
    this._alpha = 50;
};
```

### 另请参见

[\\_alpha \(MovieClip.\\_alpha 属性\)](#), [\\_alpha \(TextField.\\_alpha 属性\)](#)

## blendMode (Button.blendMode 属性)

public blendMode : Object

此按钮的混合模式。当按钮所在的图层在屏幕上另一个对象之上时，混合模式会影响按钮外观。


Flash Player 对按钮的每个像素都应用 blendMode 属性。每个像素都由三种原色（红色、绿色和蓝色）组成，每种原色的值介于 0x00 和 0xFF 之间。Flash Player 将按钮中某个像素的每个原色与背景中的像素对应的颜色进行比较。例如，如果 blendMode 设置为 "lighten"，则 Flash Player 将按钮的红色值与背景的红色值进行比较，然后使用两者中较亮的一种颜色作为显示颜色的红色成分的值。





下表将对 blendMode 设置进行说明。若要设置 blendMode 属性，您可以使用从 1 到 14 的整数或使用字符串。表中的插图表示：当按钮 (2) 重叠在另一个屏幕对象 (1) 上时对按钮 (2) 应用 blendMode。



整数值	字符串值	插图	说明
1	"normal"		此按钮显示在背景的前面。按钮的像素值将覆盖背景的像素值。在按钮为透明的区域，背景是可见的。
2	"layer"		强制为按钮的预构成创建临时缓冲区。如果按钮中有多个子对象并且为子对象选择了除 "normal" 外的其它 blendMode 设置，则此操作会自动完成。
3	"multiply"		将按钮原色的值与背景色的值相乘，然后通过除以 0xFF 进行规格化，得到较暗的颜色。这通常用于获得阴影和深度效果。 例如，如果按钮中某个像素的原色（如红色）与背景中像素的相应颜色的值均为 0x88，则相乘积的结果为 0x4840。除以 0xFF 得到该原色的值 0x48，这是比按钮或背景的颜色暗的阴影。
4	"screen"		将按钮颜色的补色（反转色）与背景颜色的补色相乘，产生漂白效果。此设置通常用于产生加亮效果或用来删除按钮的黑色区域。

整数值	字符串值	插图	说明
5	"lighten"		<p>选择按钮和背景的原色中的较亮者（值较大的原色）。此设置通常用于叠加类型。</p> <p>例如，如果按钮的某个像素的 RGB 值为 0xFFCC33，背景像素的 RGB 值为 0xDDF800，则显示像素的结果 RGB 值为 0xFFFF833（因为 0xFF &gt; 0xDD，0xCC &lt; 0xF8，且 0x33 &gt; 0x00 = 33）。</p>
6	"darken"		<p>选择按钮和背景的原色中的较暗者（值较小的原色）。此设置通常用于叠加类型。</p> <p>例如，如果按钮的某个像素的 RGB 值为 0xFFCC33，背景像素的 RGB 值为 0xDDF800，则显示像素的结果 RGB 值为 0xDDCC00（因为 0xFF &gt; 0xDD，0xCC &lt; 0xF8，且 0x33 &gt; 0x00 = 33）。</p>
7	"difference"		<p>将按钮与其背景的原色比较，然后从较亮的原色中减去较暗的原色。此设置通常用于得到更明亮的颜色。</p> <p>例如，如果按钮的某个像素的 RGB 值为 0xFFCC33，背景像素的 RGB 值为 0xDDF800，则显示像素的结果 RGB 值为 0x222C33（因为 0xFF - 0xDD = 0x22，0xF8 - 0xCC = 0x2C，且 0x33 - 0x00 = 0x33）。</p>

整数值	字符串值	插图	说明
8	"add"		<p>将按钮原色的值与其背景原色的值相加，然后应用上限值 0xFF。此设置通常用于使两个对象间的加亮溶解产生动画效果。</p> <p>例如，如果按钮的某个像素的 RGB 值为 0xAAA633，背景像素的 RGB 值为 0xDD2200，则显示像素的结果 RGB 值为 0xFFC833（因为 0xAA + 0xDD &gt; 0xFF，0xA6 + 0x22 = 0xC8，且 0x33 + 0x00 = 0x33）。</p>
9	"subtract"		<p>从背景原色的值减去按钮中原色的值，然后应用下限值 0。此设置通常用于使两个对象间的变暗溶解产生动画效果。</p> <p>例如，如果按钮的某个像素的 RGB 值为 0xAA2233，背景像素的 RGB 值为 0xDDA600，则显示像素的结果 RGB 值为 0x338400（因为 0xDD - 0xAA = 0x33，0xA6 - 0x22 = 0x84，且 0x00 - 0x33 &lt; 0x00）。</p>
10	"invert"		反转背景。
11	"alpha"		<p>将按钮的每个像素的 Alpha 值应用于背景。这要求对父级按钮应用 "layer" blendMode。例如，在此插图中，父级按钮是一个白色背景，它具有 blendMode = "layer"。</p>
12	"erase"		<p>根据按钮的 Alpha 值擦除背景。这要求对父级按钮应用 "layer" blendMode 设置。例如，在此插图中，父级按钮是一个白色背景，它具有 blendMode = "layer"。</p>

整数值	字符串值	插图	说明
13	"overlay"		根据背景的暗度调整每个位图的颜色。如果背景比 50% 灰更亮，按钮和背景颜色将被遮住，产生较亮的颜色。如果背景比 50% 灰更暗，颜色将相乘，产生较暗的颜色。此设置通常用于获得阴影效果。
14	"hardlight"		根据按钮的暗度调整每个位图的颜色。如果按钮比 50% 灰更亮，按钮和背景颜色将被遮住，产生较亮的颜色。如果按钮比 50% 灰更暗，颜色将相乘，产生较暗的颜色。此设置通常用于获得阴影效果。

如果尝试将 `blendMode` 属性设置为任何其它值，则 **Flash Player** 会将该属性设置为 `"normal"`。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

在下面的示例中您可以看到：如果将属性设置为一个整数，则 **Flash Player** 会将此值转换为对应的字符串形式：

```
my_button.blendMode = 8;  
trace (my_button.blendMode) // add
```

有关相关示例，请参见 **MovieClip** 类的 `blendMode` 属性的说明。

另请参见

[blendMode \(MovieClip.blendMode 属性\)](#)

## cacheAsBitmap (Button.cacheAsBitmap 属性)

```
public cacheAsBitmap : Boolean
```

如果设置为 `true`，**Flash Player** 将缓存按钮的内部位图表示形式。这可以增强包含复杂矢量内容的按钮的性能。

对于 `cacheAsBitmap` 设置为 `true` 的按钮，**Flash Player** 将为这四种按钮状态分别存储一个位图表示形式。

具有已缓存位图的按钮的所有矢量数据都将被绘制到位图而不是主舞台。然后，将位图复制到主舞台，作为对齐到最接近像素边界的未拉伸、未旋转的像素。对于父级对象，像素按一对一进行映射。如果位图的边界发生更改，则将重新创建位图而不会拉伸它。

除非将 `cacheAsBitmap` 属性设置为 `true`，否则不会创建内部位图。

将按钮的 `cacheAsBitmap` 属性设置为 `true` 后，呈现并不更改；但是，按钮将自动执行像素对齐。动画速度可能会大大加快，具体取决于矢量内容的复杂性。

只要对按钮（当按钮的 `filter` 数组不为空时）应用滤镜，`cacheAsBitmap` 属性就自动设置为 `true`，并且如果对按钮应用了滤镜，即使将该属性设置为 `false`，也会将该按钮的 `cacheAsBitmap` 报告为 `true`。如果清除某按钮的所有滤镜，`cacheAsBitmap` 设置将更改为其上一次的设置。

在下面的情况下，即使将 `cacheAsBitmap` 属性设置为 `true`，按钮也不使用位图，而是从矢量数据呈现：

- 位图过大，即在任一方向上都大于 2880 像素。
- 位图分配内存失败（由于内存不足的错误）

最好将 `cacheAsBitmap` 属性与主要具有静态内容且不频繁缩放和旋转的按钮一起使用。对于此类按钮，在转换按钮时（更改其 `x` 和 `y` 位置时），`cacheAsBitmap` 可以提高性能。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例对名为 `myButton` 的现有按钮实例应用了投影。然后它描摹出 `cacheAsBitmap` 的值，该值在应用滤镜时设置为 `true`。

```
import flash.filters.DropShadowFilter;
trace(myButton.cacheAsBitmap); // false
var dropShadow:DropShadowFilter = new DropShadowFilter(6, 45, 0x000000, 50,
    5, 5, 1, 2, false, false, false);
myButton.filters = new Array(dropShadow);
trace(myButton.cacheAsBitmap); // true
```

## enabled（Button.enabled 属性）

`public enabled : Boolean`

布尔值，指定按钮是否处于启用状态。按钮被禁用时（`enabled` 属性设置为 `false`），该按钮虽然可见，但不能被单击。默认值为 `true`。如果想要禁用部分导航按钮，则此属性非常有用；例如，您可能希望禁用当前显示的页面上的某个按钮，以便禁止点击该按钮并禁止重新加载该页面。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例演示如何禁用和启用按钮单击。在舞台上放两个按钮（myBtn1\_btn 和 myBtn2\_btn），并添加以下 **ActionScript**，以便无法单击 myBtn2\_btn 按钮。首先，在舞台上添加两个按钮实例。然后，分别为它们指定实例名称 myBtn1\_btn 和 myBtn2\_btn。最后，将以下代码放在第 1 帧上以启用或禁用按钮。

```
myBtn1_btn.enabled = true;
myBtn2_btn.enabled = false;

//button code
// the following function will not get called
// because myBtn2_btn.enabled was set to false
myBtn1_btn.onRelease = function() {
    trace( "you clicked : " + this._name );
};
myBtn2_btn.onRelease = function() {
    trace( "you clicked : " + this._name );
};
```

## filters（Button.filters 属性）

public filters : Array

索引数组，包含当前与按钮关联的每个滤镜对象。flash.filters 包中包含的多个类定义了可使用的特定滤镜。

在设计时或者在运行时（使用 **ActionScript** 代码），可以在 **Flash** 创作工具中应用滤镜。若要使用 **ActionScript** 应用滤镜，您必须制作整个 Button.filters 数组的临时副本，修改临时数组，然后将临时数组的值重新分配给 Button.filters 数组。无法直接将新滤镜对象添加到 Button.filters 数组。以下代码对名为 myButton 的目标按钮不起作用：

```
myButton.filters[0].push(myDropShadow);
```

若要使用 **ActionScript** 添加滤镜，您必须按照下列步骤操作（假定目标按钮名为 myButton）：

- 使用所选滤镜类的构造函数创建一个新的滤镜对象。
- 将 myButton.filters 数组的值分配给临时数组，例如一个名为 myFilters 的数组。
- 将新的滤镜对象添加到临时数组 myFilters。
- 将临时数组的值分配给 myButton.filters 数组。

如果 filters 数组为空，则无需使用临时数组。相反，您可以直接分配包含一个或多个已创建的滤镜对象的数组。

若要修改现有的滤镜对象（不管它是在设计时创建的还是在运行时创建的），您必须使用修改 filters 数组副本的技巧：

- 将 myButton.filters 数组的值分配给临时数组，例如一个名为 myFilters 的数组。

- 使用临时数组 `myFilters` 修改属性。例如，如果要设置数组中第一个滤镜的 `quality` 属性，可以使用以下代码：`myList[0].quality = 1;`
- 将临时数组的值分配给 `myButton.filters` 数组。

若要清除某按钮的滤镜，请将 `filters` 设置为空数组 (`[]`)。

在加载时，如果按钮具有关联的滤镜，则将它标记为像透明位图那样缓存自身。从此时起，只要按钮具有有效的滤镜列表，播放器就会将按钮缓存为位图。此位图用作滤镜效果的源图像。每个按钮通常具有两组位图：一组用于原始的未过滤的源按钮，另一组用于过滤后的最终图像（分别位于四种按钮状态中）。呈现时使用最终图像设置。只要按钮不发生更改，最终图像就不需要更新。

如果正在使用包含多个滤镜的 `filters` 数组，并且需要跟踪分配给每个数组索引的滤镜类型，则可以维护您自己的 `filters` 数组，并使用单独的数据结构跟踪与每个数组索引关联的滤镜类型。没有简单的方法可以确定与每个 `filters` 数组索引关联的滤镜类型。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例将投影滤镜添加到名为 `myButton` 的按钮。

```
import flash.filters.DropShadowFilter;
var myDropFilter:DropShadowFilter = new DropShadowFilter(6, 45, 0x000000,
    50, 5, 5, 1, 2, false, false, false);
var myFilters:Array = myButton.filters;
myFilters.push(myDropFilter);
myButton.filters = myFilters;
```

下面的示例将数组中第一个滤镜的 `quality` 设置更改为 `15`（仅当至少一个滤镜对象已经与 `myButton` 文本字段关联时，此示例才起作用）。

```
var myList:Array = myButton.filters;
myList[0].quality = 15;
myButton.filters = myList;
```

另请参见

， `cacheAsBitmap` (`Button.cacheAsBitmap` 属性)

## `_focusrect` (Button.\_focusrect 属性)

```
public _focusrect : Boolean
```

布尔值，指定当按钮具有键盘焦点时，其四周是否有黄色矩形。此属性可覆盖全局 `_focusrect` 属性。默认情况下，按钮实例的 `_focusrect` 属性为 `null`；这意味着该按钮实例不会覆盖全局 `_focusrect` 属性。如果将按钮实例的 `_focusrect` 属性设置为 `true` 或 `false`，它将覆盖单个按钮实例的全局 `_focusrect` 属性的设置。

在 **Flash Player 4** 或 **Flash Player 5 SWF** 文件中，`_focusrect` 属性控制全局 `_focusrect` 属性。它是一个布尔值。在 **Flash Player 6** 及更高版本中更改了这一行为，以便允许在一个单独的影片剪辑上自定义 `_focusrect` 属性。

如果将 `_focusrect` 属性设置为 `false`，则该按钮的键盘导航将被限制为 **Tab** 键。忽略所有其它键，包括 **Enter** 键和箭头键。要恢复全键盘导航，必须将 `_focusrect` 设为 `true`。

可用性：ActionScript 1.0；Flash Player 6

### 示例

此示例演示当 **SWF** 文件在浏览器窗口中具有焦点时，如何隐藏该文件中指定按钮实例周围的黄色矩形。分别创建名为 `myBtn1_btn`、`myBtn2_btn` 和 `myBtn3_btn` 的三个按钮，并将以下 **ActionScript** 添加到时间轴的第 1 帧：

```
myBtn2_btn._focusrect = false;
```

将发布设置更改为 **Flash Player 6**，然后选择“文件”>“发布预览”>“HTML”，在浏览器窗口中测试该 **SWF** 文件。在浏览器窗口中单击 **SWF** 焦点，并使用 **Tab** 键将焦点移至每个实例，即可指定 **SWF** 焦点。当禁用 `_focusrect` 时，您将无法通过按 **Enter** 键或空格键来执行此按钮的代码。

## `getDepth` (Button.getDepth 方法)

```
public getDepth() : Number
```

返回按钮实例的深度。

每个影片剪辑、按钮和文本字段都有与自己关联的唯一深度，它确定对象在其它对象前或其它对象后的显示方式。深度较高的对象显示在前面。

可用性：ActionScript 1.0；Flash Player 6

### 返回

Number - 按钮实例的深度。



## 示例

如果在舞台上创建 myBtn1\_btn 和 myBtn2\_btn，可以使用以下 **ActionScript** 跟踪它们的深度：

```
trace(myBtn1_btn.getDepth());
trace(myBtn2_btn.getDepth());
```

如果将名为 **buttonMovie.swf** 的 **SWF** 文件加载到此文档中，即可使用主 **SWF** 文件中的另一个按钮跟踪该 **SWF** 文件中的按钮 myBtn4\_btn 的深度：

```
this.createEmptyMovieClip("myClip_mc", 999);
myClip_mc.loadMovie("buttonMovie.swf");
myBtn3_btn.onRelease = function(){
    trace(myClip_mc.myBtn4_btn.getDepth());
};
```

您可能注意到这些按钮中有两个按钮具有相同的深度值，一个按钮在主 **SWF** 文件中，一个按钮在加载的 **SWF** 文件中。这是容易产生误解的，原因是 **buttonMovie.swf** 加载的深度为 999，这表示它包含的按钮相对于主 **SWF** 文件中的按钮也将有 999 的深度。请记住，每个影片剪辑都有自己的内部 **z** 顺序，这意味着每个影片剪辑都有自己的一组深度值。这两个按钮可能具有相同深度值，但这些值只是在涉及相同 **Z** 顺序中的其它对象时才有意义。在此例中，虽然两个按钮具有相同的深度值，但这些值与不同的影片剪辑相关。例如，主 **SWF** 文件中的按钮的深度值与主时间轴的 **Z** 顺序相关，而加载的 **SWF** 文件中的按钮的深度值与 myClip\_mc 影片剪辑的内部 **Z** 顺序相关。

另请参见

[getDepth \(MovieClip.getDepth 方法\)](#)，[getDepth \(TextField.getDepth 方法\)](#)，

## \_height (Button.\_height 属性)

```
public _height : Number
```

按钮的高度，以像素为单位。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例将名为 my\_btn 的按钮的高度和宽度设置为指定的高度和宽度。

```
my_btn._width = 500;
my_btn._height = 200;
```

## \_highquality (Button.\_highquality 属性)

public \_highquality : Number

自 **Flash Player 7** 后不推荐使用。不推荐使用此属性，而推荐使用 `Button._quality`。

指定当前 **SWF** 文件所应用的消除锯齿的级别。指定 **2**（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 **1**（高品质）将应用消除锯齿；如果 **SWF** 文件中没有动画并且是默认值，指定 **1** 将对位图图像进行平滑处理。指定 **0**（低品质），则不消除锯齿。

可用性：ActionScript 1.0；Flash Player 6

### 示例

在舞台上添加一个按钮实例，并将它命名为 `myBtn_btn`。使用具有笔触颜色和填充颜色的“椭圆”工具在舞台上绘制一个椭圆。选择第 1 帧，并使用“动作”面板添加以下

**ActionScript**：

```
myBtn_btn.onRelease = function() {  
    myBtn_btn._highquality = 0;  
};
```

单击 `myBtn_btn` 时，圆的笔触将显示出锯齿。可以添加以下 **ActionScript** 以影响整个 **SWF**：

```
_quality = 0;
```

另请参见

[\\_quality \(Button.\\_quality 属性\)](#)，[\\_quality 属性](#)

## menu (Button.menu 属性)

public menu : ContextMenu

将 `ContextMenu` 对象 `contextMenu` 与按钮对象 `my_button` 相关联。**ContextMenu** 类用于修改当用户在 **Flash Player** 中右键单击（在 **Windows** 中）或按住 **Control** 键并单击（在 **Macintosh** 中）时显示的上下文菜单。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例将 **ContextMenu** 对象分配给名为 `myBtn_btn` 的按钮实例。**ContextMenu** 对象包含单个菜单项（标记为“**Save...**”），该菜单项具有名为 `doSave` 的关联回调处理函数。

在舞台上添加一个按钮实例，并将它命名为 `myBtn_btn`。

```
var menu_cm:ContextMenu = new ContextMenu();  
menu_cm.customItems.push(new ContextMenuItem("Save...", doSave));  
function doSave(menu:Object, obj:Object):Void {  
    trace( " You selected the 'Save...' menu item ");  
}
```

```
}  
myBtn_btn.menu = menu_cm;
```

选择“控制”>“测试影片”以测试 SWF 文件。将鼠标指针置于 myBtn\_btn 上，右键单击或按住 **Ctrl** 键单击它。会出现上下文菜单，并且菜单中有“保存”命令。当从菜单中选择“保存”时，会出现“输出”面板。

另请参见

[ContextMenu](#), [ContextMenuItem](#), [menu \(MovieClip.menu 属性\)](#), [menu \(TextField.menu 属性\)](#)

## \_name (Button.\_name 属性)

```
public _name : String
```

由 my\_btn 指定的按钮的实例名称。

可用性：ActionScript 1.0；Flash Player 6

示例

下面的示例跟踪 SWF 文件的当前时间轴内所有 **Button** 实例的所有实例名称。

```
for (i in this) {  
    if (this[i] instanceof Button) {  
        trace(this[i]._name);  
    }  
}
```

## onDragOut (Button.onDragOut 处理函数)

```
onDragOut = function() {}
```

当在按钮上单击鼠标按钮，然后将鼠标指针拖动到按钮之外时调用。必须定义一个在调用事件处理函数时执行的函数。

可用性：ActionScript 1.0；Flash Player 6

示例

下面的示例演示在将指针从按钮上拖开时，如何执行语句。在舞台上创建一个名为 my\_btn 的按钮，并在时间轴的某一帧中输入以下 **ActionScript**：

```
my_btn.onDragOut = function() {  
    trace("onDragOut: "+this._name);  
};  
my_btn.onDragOver = function() {  
    trace("onDragOver: "+this._name);  
};
```

## onDragOver (Button.onDragOver 处理函数)

```
onDragOver = function() {}
```

当用户在按钮外部按下鼠标按钮，然后将鼠标指针拖动到按钮之上时调用。必须定义一个在调用事件处理函数时执行的函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例为 **onDragOver** 处理函数定义一个函数，该函数将一个 `trace()` 语句发送到“输出”面板。在舞台上创建一个名为 `my_btn` 的按钮，并在时间轴上输入以下 **ActionScript**:

```
my_btn.onDragOut = function() {  
    trace("onDragOut: "+this._name);  
};  
my_btn.onDragOver = function() {  
    trace("onDragOver: "+this._name);  
};
```

在测试 **SWF** 文件时，请将指针从按钮实例上拖开。然后，按住鼠标按钮的同时，再次将指针拖动到按钮实例上。请注意，“输出”面板会跟踪您的动作。

另请参见

[onDragOut \(Button.onDragOut 处理函数\)](#)

## onKeyDown (Button.onKeyDown 处理函数)

```
onKeyDown = function() {}
```

当按钮具有键盘焦点而且按下某按键时调用。调用 `onKeyDown` 事件处理函数时无需使用参数。您可以使用 `Key.getAscii()` 和 `Key.getCode()` 方法来确定按下了哪个键。必须定义一个在调用事件处理函数时执行的函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

在下面的示例中，为 `onKeyDown` 处理函数定义了向“输出”面板发送文本的函数。在舞台上创建一个名为 `my_btn` 的按钮，并在时间轴的某一帧中输入以下 **ActionScript**:

```
my_btn.onKeyDown = function() {  
    trace("onKeyDown: "+this._name+" (Key: "+getKeyPressed()+")");  
};  
function getKeyPressed():String {  
    var theKey:String;  
    switch (Key.getAscii()) {  
        case Key.BACKSPACE :  
            theKey = "BACKSPACE";
```

```

        break;
    case Key.SPACE :
        theKey = "SPACE";
        break;
    default :
        theKey = chr(Key.getAscii());
    }
    return theKey;
}

```

选择“控制” > “测试影片”以测试 SWF 文件。确保在测试环境中选择“控制” > “禁用快捷键”。然后按下 **Tab** 键，直到按钮具有焦点（my\_btn 实例周围出现黄色矩形），再开始按键盘上的键。被按的键显示在“输出”面板中。

另请参见

[onKeyUp \(Button.onKeyUp 处理函数\)](#), [getAscii \(Key.getAscii 方法\)](#), [getCode \(Key.getCode 方法\)](#)

## onKeyUp (Button.onKeyUp 处理函数)

```
onKeyUp = function() {}
```

当按钮具有输入焦点而且释放某按键时调用。调用 onKeyUp 事件处理函数时无需使用参数。您可以使用 Key.getAscii() 和 Key.getCode() 方法来确定按下了哪个键。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

在下面的示例中，为 onKeyDown 处理函数定义了向“输出”面板发送文本的函数。在舞台上创建一个名为 my\_btn 的按钮，并在时间轴的某一帧中输入以下 **ActionScript**：

```

my_btn.onKeyDown = function() {
    trace("onKeyDown: "+this._name+" (Key: "+getKeyPressed()+")");
};
my_btn.onKeyUp = function() {
    trace("onKeyUp: "+this._name+" (Key: "+getKeyPressed()+")");
};
function getKeyPressed():String {
    var theKey:String;
    switch (Key.getAscii()) {
    case Key.BACKSPACE :
        theKey = "BACKSPACE";
        break;
    case Key.SPACE :
        theKey = "SPACE";
        break;
    default :
        theKey = chr(Key.getAscii());
    }
}

```

```
    }  
    return theKey;  
}
```

按 **Ctrl+Enter** 以测试 SWF 文件。确保在测试环境中选择“控制”>“禁用快捷键”。然后按下 **Tab** 键，直到按钮具有焦点（my\_btn 实例周围出现黄色矩形），再开始按键盘上的键。被按的键显示在“输出”面板中。

另请参见

`onKeyDown` (`Button.onKeyDown` 处理函数), `getAscii` (`Key.getAscii` 方法), `getCode` (`Key.getCode` 方法)

## onKillFocus (Button.onKillFocus 处理函数)

```
onKillFocus = function(newFocus:Object) {}
```

当按钮失去键盘焦点时调用。onKillFocus 处理函数接收一个参数 newFocus，该参数是表示要接收焦点的新对象。如果没有对象接收焦点，则 newFocus 将包含值 null。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**newFocus:Object** - 接收焦点的对象。

### 示例

下面的示例演示当按钮失去焦点时，如何执行语句。在舞台上创建一个名为 my\_btn 的按钮实例，并将以下 **ActionScript** 添加到时间轴的第 1 帧：

```
this.createTextField("output_txt", this.getNextHighestDepth(), 0, 0, 300,  
    200);  
output_txt.wordWrap = true;  
output_txt.multiline = true;  
output_txt.border = true;  
my_btn.onKillFocus = function() {  
    output_txt.text = "onKillFocus: "+this._name+newline+output_txt.text;  
};
```

在浏览器窗口中测试 SWF 文件，并尝试使用 **Tab** 键遍历窗口中的元素。当按钮实例失去焦点时，会向 output\_txt 文本字段发送文本。

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类代替 `MovieClip.getNextHighestDepth()` 方法。

## onPress (Button.onPress 处理函数)

```
onPress = function() {}
```

当按下按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

在下面的示例中, 为 onPress 处理函数定义了将 **trace()** 语句发送到“输出”面板的函数:

```
my_btn.onPress = function () {  
    trace ("onPress called");  
};
```

## onRelease (Button.onRelease 处理函数)

```
onRelease = function() {}
```

当释放按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

在下面的示例中, 为 onRelease 处理函数定义了将 **trace()** 语句发送到“输出”面板的函数:

```
my_btn.onRelease = function () {  
    trace ("onRelease called");  
};
```

## onReleaseOutside (Button.onReleaseOutside 处理函数)

```
onReleaseOutside = function() {}
```

在这样的情况下调用: 在鼠标指针位于按钮内部的情况下按下按钮, 然后将鼠标指针移到该按钮外部并释放鼠标按钮。必须定义一个在调用事件处理函数时执行的函数。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

在下面的示例中, 为 onReleaseOutside 处理函数定义了将 **trace()** 语句发送到“输出”面板的函数:

```
my_btn.onReleaseOutside = function () {  
    trace ("onReleaseOutside called");  
};
```

## onRollOut (Button.onRollOut 处理函数)

```
onRollOut = function() {}
```

当鼠标指针移至按钮区域之外时调用。必须定义一个在调用事件处理函数时执行的函数。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

在下面的示例中, 为 onRollOut 处理函数定义了将 **trace()** 语句发送到“输出”面板的函数:

```
my_btn.onRollOut = function () {  
    trace ("onRollOut called");  
};
```

## onRollOver (Button.onRollOver 处理函数)

```
onRollOver = function() {}
```

当鼠标指针移过按钮区域时调用。必须定义一个在调用事件处理函数时执行的函数。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

在下面的示例中, 为 onRollOver 处理函数定义了将 **trace()** 语句发送到“输出”面板的函数:

```
my_btn.onRollOver = function () {  
    trace ("onRollOver called");  
};
```

## onSetFocus (Button.onSetFocus 处理函数)

```
onSetFocus = function(oldFocus:Object) {}
```

当按钮接收键盘焦点时调用。oldFocus 参数是失去焦点的对象。例如, 如果用户按下 **Tab** 键将输入焦点从文本字段移到按钮, 则 oldFocus 包含该文本字段实例。

如果以前没有具有焦点的对象, 则 oldFocus 包含一个 **null** 值。

可用性: ActionScript 1.0 ; Flash Player 6

### 参数

oldFocus:Object - 将失去键盘焦点的对象。



## 示例

下面的示例演示：当 SWF 文件的用户将焦点从一个按钮移动到另一个按钮时，将如何执行语句。创建两个按钮（btn1\_btn 和 btn2\_btn），并在时间轴的第 1 帧中输入以下

### ActionScript:

```
Selection.setFocus(btn1_btn);
trace(Selection.getFocus());
btn2_btn.onSetFocus = function(oldFocus) {
    trace(oldFocus._name + "lost focus");
};
```

通过按 **Ctrl+Enter** 来测试 SWF 文件。确保选择“控制”>“禁用快捷键”（如果尚未选择）。将焦点设置在 btn1\_btn 上。当 btn1\_btn 失去焦点而 btn2\_btn 获得焦点时，会在“输出”面板中显示信息。

## \_parent (Button.\_parent 属性)

```
public _parent : MovieClip
```

对包含当前影片剪辑或对象的影片剪辑或对象的引用。当前对象是一个包含引用 \_parent 的 **ActionScript** 代码的对象。

使用 \_parent 来指定一个相对路径，该路径指向当前影片剪辑或对象之上的影片剪辑或对象。可以使用 \_parent 在显示列表中上移多个级别，如下所示：

```
this._parent._parent.alpha = 20;
```

可用性：ActionScript 1.0 ； Flash Player 6

## 示例

在下面的示例中，将名为 my\_btn 的按钮放置在名为 my\_mc 的影片剪辑内。下面的代码演示如何使用 \_parent 属性来获取对影片剪辑 my\_mc 的引用：

```
trace(my_mc.my_btn._parent);
```

“输出”面板将显示以下结果：

```
_level0.my_mc
```

## 另请参见

[\\_parent \(MovieClip.\\_parent 属性\)](#)，[\\_target \(MovieClip.\\_target 属性\)](#)，[\\_root 属性](#)

## \_quality (Button.\_quality 属性)

public \_quality : String

属性（全局）：设置或检索用于 SWF 文件的呈现品质。设备字体始终带有锯齿，因此不受 \_quality 属性的影响。

\_quality 属性可设置为下列值：

- “LOW” 低呈现品质。不消除图形的锯齿，位图不进行平滑处理。
- “MEDIUM” 中等呈现品质。使用 2 x 2 像素网格消除图形锯齿，但不对位图进行平滑处理。适用于不包含文本的影片。
- “HIGH” 高呈现品质。使用 4 x 4 像素网格消除图形锯齿，如果影片是静态的，则对位图进行平滑处理。这是 Flash 使用的默认呈现品质设置。
- “BEST” 极高呈现品质。使用 4 x 4 像素网格消除图形锯齿，并且始终对位图进行平滑处理。



尽管您可以为 Button 对象指定此属性，但它实际上是一个全局属性，因此您只需将它的值指定为 \_quality。

可用性：ActionScript 1.0；Flash Player 6

### 示例

此示例将名为 my\_btn 的按钮的呈现品质设置为 LOW：

```
my_btn._quality = "LOW";
```

## \_rotation (Button.\_rotation 属性)

public \_rotation : Number

按钮距其原始方向的旋转程度，以度为单位。从 0 到 180 的值表示顺时针方向旋转；从 0 到 -180 的值表示逆时针方向旋转。对于此范围之外的值，可以通过加上或减去 360 获得该范围内的值。例如，my\_btn.\_rotation = 450 语句与 my\_btn.\_rotation = 90 是相同的。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例在舞台上旋转两个按钮。在舞台上创建名为 control\_btn 和 my\_btn 的两个按钮。确保 my\_btn 不完全是圆的，以便可以看到它旋转。然后在时间轴的第 1 帧中输入以下 **ActionScript**：

```
var control_btn:Button;  
var my_btn:Button;  
control_btn.onRelease = function() {  
    my_btn._rotation += 10;  
};
```

现在，在舞台上创建名为 `myOther_btn` 的另一个按钮，确保它不完全是圆的，以便可以看到它旋转。在时间轴的第 1 帧中输入下面的 **ActionScript**。

```
var myOther_btn:Button;
this.createEmptyMovieClip("rotater_mc", this.getNextHighestDepth());
rotater_mc.onEnterFrame = function() {
    myOther_btn._rotation += 2;
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类代替 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[\\_rotation \(MovieClip.\\_rotation 属性\)](#), [\\_rotation \(TextField.\\_rotation 属性\)](#)

## scale9Grid (Button.scale9Grid 属性)

```
public scale9Grid : Rectangle
```

为按钮定义九个缩放区域的矩形区域。如果设置为 `null`，则在应用任何缩放转换时，将正常缩放整个按钮。

当定义按钮的 `scale9Grid` 属性时，该按钮被划分为以 `scale9Grid` 矩形为基础的具有九个区域的网格，该矩形定义网格的中心区域。还有其它八个网格区域，如下所示：

- 矩形外部左上角的区域
- 矩形上方的区域
- 矩形外部右上角的区域
- 矩形左侧的区域
- 矩形右侧的区域
- 矩形外部左下角的区域
- 矩形下方的区域
- 矩形外部右下角的区域

可以将中心区域（由矩形定义）之外的八个区域视为在缩放按钮时已应用特殊规则的图片帧。

在设置 `scale9Grid` 属性并缩放按钮后，会正常缩放所有文本和渐变；但是，对于对象的其它类型，将应用以下规则：

- 正常缩放中心区域中的内容。
- 不缩放转角中的内容。
- 仅水平缩放顶部和底部区域中的内容。仅垂直缩放左侧和右侧区域中的内容。

如果旋转按钮，则所有后续缩放都是正常的，并会忽略 `scale9Grid` 属性。  
`scale9Grid` 属性的常见用法是设置按钮，当缩放按钮时，按钮的边缘线会保持相同的宽度。  
有关更多信息，包括插图和相关示例，请参见 `MovieClip.scale9Grid`。

可用性: `ActionScript 1.0` ; `Flash Player 8`

另请参见

[Rectangle \(flash.geom.Rectangle\)](#), [scale9Grid \(MovieClip.scale9Grid 属性\)](#)

## `_soundbuftime` (`Button._soundbuftime` 属性)

`public _soundbuftime : Number`

指定在声音开始进行流处理前预先缓冲的秒数的属性。



尽管您可以为 `Button` 对象指定此属性，但它实际上是一个应用于所有加载的声音的全局属性，因此您只需将它的值指定为 `_soundbuftime`。设置 `Button` 对象的此属性实际上是设置全局属性。

有关更多信息和示例，请参见 `_soundbuftime` 全局属性。

可用性: `ActionScript 1.0` ; `Flash Player 6`

另请参见

[\\_soundbuftime 属性](#)

## `tabEnabled` (`Button.tabEnabled` 属性)

`public tabEnabled : Boolean`

指定 `my_btn` 是否包括在 **Tab** 键的自动排序中。默认情况下为 `undefined`。

如果 `tabEnabled` 属性为 `undefined` 或 `true`，则该对象包括在 **Tab** 键的自动排序中。如果 `tabIndex` 属性也设置为某个值，则该对象也包括在 **Tab** 键的自定义排序中。如果 `tabEnabled` 为 `false`，则即使设置了 `tabIndex` 属性，该对象也不包括在 **Tab** 键的自动或自定义排序中。

可用性: `ActionScript 1.0` ; `Flash Player 6`

## 示例

以下 **ActionScript** 用于将四个按钮中某个按钮的 `tabEnabled` 属性设置为 `false`。但是，所有四个按钮（`one_btn`、`two_btn`、`three_btn` 和 `four_btn`）均使用 `tabIndex` 排入自定义 **Tab** 键顺序中。虽然为 `three_btn` 设置了 `tabIndex`，但自定义或自动 **Tab** 键顺序中不包括 `three_btn`，原因是该实例的 `tabEnabled` 设置为 `false`。要为四个按钮设置 **Tab** 键排序，请将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
three_btn.tabEnabled = false;
two_btn.tabIndex = 1;
four_btn.tabIndex = 2;
three_btn.tabIndex = 3;
one_btn.tabIndex = 4;
```

确保在测试 SWF 文件时禁用快捷键，方法是在测试环境中选择“控制”>“禁用快捷键”。

## 另请参见

[tabIndex \(Button.tabIndex 属性\)](#)，[tabEnabled \(MovieClip.tabEnabled 属性\)](#)，[tabEnabled \(TextField.tabEnabled 属性\)](#)

## tabIndex (Button.tabIndex 属性)

`public tabIndex : Number`

用于自定义 SWF 文件中对象的 **Tab** 键排序。可以设置按钮、影片剪辑或文本字段实例的 `tabIndex` 属性；默认情况下该属性为 `undefined`。

如果 SWF 文件中当前显示的任何对象包含 `tabIndex` 属性，则禁用 **Tab** 键的自动排序，而使用该 SWF 文件中对象的 `tabIndex` 属性来计算 **Tab** 键排序。这个自定义的 **Tab** 键排序仅包括具有 `tabIndex` 属性的对象。

`tabIndex` 属性可以是非负整数。这些对象按照其 `tabIndex` 属性按升序进行排序。`tabIndex` 值为 1 的对象在 `tabIndex` 值为 2 的对象的前面。如果两个对象具有相同的 `tabIndex` 值，则在 **Tab** 键排序中在前面的那个对象为 `undefined`。

由 `tabIndex` 属性定义的 **Tab** 键的自定义排序为平构。这意味着不考虑 SWF 文件中对象的层次结构关系。SWF 文件中具有 `tabIndex` 属性的所有对象都排入 **Tab** 键顺序中，而 **Tab** 键顺序由 `tabIndex` 值的顺序确定。如果两个对象具有相同的 `tabIndex` 值，则先出现的对象为 `undefined`。不应为多个对象使用同一个 `tabIndex` 值。

可用性：ActionScript 1.0；Flash Player 6

## 示例

以下 **ActionScript** 用于将四个按钮中某个按钮的 `tabEnabled` 属性设置为 `false`。但是，所有四个按钮（`one_btn`、`two_btn`、`three_btn` 和 `four_btn`）均使用 `tabIndex` 排入自定义 **Tab** 键顺序中。虽然为 `three_btn` 设置了 `tabIndex`，但自定义或自动 **Tab** 键顺序中不包括 `three_btn`，原因是该实例的 `tabEnabled` 设置为 `false`。要为四个按钮设置 **Tab** 键排序，请将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
three_btn.tabEnabled = false;
two_btn.tabIndex = 1;
four_btn.tabIndex = 2;
three_btn.tabIndex = 3;
one_btn.tabIndex = 4;
```

确保在测试 SWF 文件时禁用快捷键，方法是在测试环境中选择“控制”>“禁用快捷键”。

## 另请参见

[tabEnabled \(Button.tabEnabled 属性\)](#)，[tabChildren \(MovieClip.tabChildren 属性\)](#)，[tabEnabled \(MovieClip.tabEnabled 属性\)](#)，[tabIndex \(MovieClip.tabIndex 属性\)](#)，[tabIndex \(TextField.tabIndex 属性\)](#)

## \_target（Button.\_target 属性）

`public _target : String [read-only]`

返回由 `my_btn` 指定的按钮实例的目标路径。

可用性：ActionScript 1.0；Flash Player 6

## 示例

在舞台上添加一个实例名称为 `my_btn` 的按钮实例，然后将以下代码添加到时间轴的第 1 帧：

```
trace(my_btn._target); //displays /my_btn
```

选择 **my\_btn** 并将其转换为影片剪辑。为新影片剪辑指定实例名称 `my_mc`。删除时间轴第 1 帧中的现有 **ActionScript**，然后将其替换为：

```
my_mc.my_btn.onRelease = function(){
    trace(this._target); //displays /my_mc/my_btn
};
```

要将记号从斜杠记号转换为点记号，请将前面的代码示例修改为：

```
my_mc.my_btn.onRelease = function(){
    trace(eval(this._target)); //displays _level0.my_mc.my_btn
};
```

这使您可以访问目标对象的方法和参数，例如：

```
my_mc.my_btn.onRelease = function(){
    var target_btn:Button = eval(this._target);
    trace(target_btn._name); //displays my_btn
};
```

另请参见

[\\_target \(MovieClip.\\_target 属性\)](#)

## trackAsMenu (Button.trackAsMenu 属性)

```
public trackAsMenu : Boolean
```

布尔值，指示其它按钮或影片剪辑是否可接收鼠标释放事件。如果您拖动一个按钮，然后在第二个按钮上释放，则会为第二个按钮注册 onRelease 事件。这将允许您创建菜单。您可以对任何按钮或影片剪辑对象设置 trackAsMenu 属性。如果 trackAsMenu 属性尚未定义，则默认行为是 false。

您可以随时更改 trackAsMenu 属性；修改后的按钮会立即具有新的行为。

**可用性：**ActionScript 1.0；Flash Player 6

### 示例

下面的示例演示如何将两个按钮作为一个菜单来跟踪。在舞台上放置名为 one\_btn 和 two\_btn 的两个按钮实例。在时间轴中输入以下 **ActionScript**：

```
var one_btn:Button;
var two_btn:Button;
one_btn.trackAsMenu = true;
two_btn.trackAsMenu = true
one_btn.onRelease = function() {
    trace("clicked one_btn");
};
two_btn.onRelease = function() {
    trace("clicked two_btn");
};
```

请通过单击 one\_btn 上方的舞台来测试 SWF 文件，这时要按住鼠标按钮，再在 two\_btn 上方松开。接下来，尝试注释掉 **ActionScript** 的包含 trackAsMenu 的两行，然后再次测试该 SWF 文件以查看按钮行为的差异。

另请参见

[trackAsMenu \(MovieClip.trackAsMenu 属性\)](#)

## \_url (Button.\_url 属性)

public \_url : String [read-only]

检索创建按钮的 SWF 文件的 URL。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

在舞台上创建名为 one\_btn 和 two\_btn 的两个按钮实例。在时间轴的第 1 帧中输入下面的 **ActionScript**:

```
var one_btn:Button;
var two_btn:Button;
this.createTextField("output_txt", 999, 0, 0, 100, 22);
output_txt.autoSize = true;
one_btn.onRelease = function() {
    trace("clicked one_btn");
    trace(this._url);
};
two_btn.onRelease = function() {
    trace("clicked "+this._name);
    var url_array:Array = this._url.split("/");
    var my_str:String = String(url_array.pop());
    output_txt.text = unescape(my_str);
};
```

单击每个按钮时，在“输出”面板中将显示包含这些按钮的 SWF 的文件名。

## useHandCursor (Button.useHandCursor 属性)

public useHandCursor : Boolean

布尔值，当设置为 true（默认值）时，指示鼠标指针滑过按钮上方时是否显示手指形（手形光标）。如果此属性设置为 false，则将改用箭头指针。

您可以随时更改 useHandCursor 属性；修改后的按钮会立即具有新的光标行为。可以从原型对象中读出 useHandCursor 属性。

可用性: **ActionScript 1.0 ; Flash Player 6**



## 示例

在舞台上创建两个按钮，它们的实例名称分别为 myBtn1\_btn 和 myBtn2\_btn。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
myBtn1_btn.useHandCursor = false;
myBtn1_btn.onRelease = buttonClick;
myBtn2_btn.onRelease = buttonClick;
function buttonClick() {
    trace(this._name);
}
```

当鼠标指针位于 myBtn1\_btn 上方并单击该按钮时，不会显示手形光标。但是，当鼠标指针位于 myBtn2\_btn 上方并单击该按钮时，您将看到手形光标。

## \_visible (Button.\_visible 属性)

```
public _visible : Boolean
```

布尔值，指示由 my\_btn 指定的按钮是否可见。禁用不可见的按钮（\_visible 属性设置为 false）。

可用性：ActionScript 1.0；Flash Player 6

## 示例

在舞台上创建两个按钮，它们的实例名称分别为 myBtn1\_btn 和 myBtn2\_btn。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
myBtn1_btn.onRelease = function() {
    this._visible = false;
    trace("clicked "+this._name);
};
myBtn2_btn.onRelease = function() {
    this._alpha = 0;
    trace("clicked "+this._name);
};
```

请注意，在将 Alpha 设置为 0 后您依然可以单击 myBtn2\_btn。

另请参见

[\\_visible \(MovieClip.\\_visible 属性\)](#)，[\\_visible \(TextField.\\_visible 属性\)](#)

## `_width` (Button.`_width` 属性)

`public _width : Number`

按钮的宽度，以像素为单位。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例增加名为 `my_btn` 的按钮的 `width` 属性，并在“输出”面板中显示宽度。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
my_btn.onRelease = function() {  
    trace(this._width);  
    this._width += 1.1;  
};
```

另请参见

[\\_width \(MovieClip.\\_width 属性\)](#)

## `_x` (Button.`_x` 属性)

`public _x : Number`

整数，用来设置按钮相对于父级影片剪辑的本地坐标的 `x` 坐标。如果按钮在主时间轴上，则其坐标系将舞台的左上角作为 `(0, 0)`。如果按钮在具有变形的影片剪辑内，则该按钮位于包含它的影片剪辑的本地坐标系中。因此，对于逆时针旋转 90 度的影片剪辑，其中的按钮将继承逆时针旋转 90 度的坐标系。按钮的坐标指的是注册点的位置。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例将 `my_btn` 在舞台上的坐标设置为 0。创建名为 `my_btn` 的按钮，并在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
my_btn._x = 0;  
my_btn._y = 0;
```

另请参见

[\\_xscale \(Button.\\_xscale 属性\)](#)，[\\_y \(Button.\\_y 属性\)](#)，[\\_yscale \(Button.\\_yscale 属性\)](#)

## `_xmouse` (`Button._xmouse` 属性)

`public _xmouse : Number [read-only]`

返回鼠标位置相对于按钮的 `x` 坐标。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例显示舞台的 `xmouse` 位置，以及一个放在舞台上的名为 `my_btn` 的按钮。在时间轴的第 1 帧中输入下面的 **ActionScript**:

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var table_str:String = "<textformat tabstops='[50,100]'">";
    table_str += "<b>Stage</b>\t"+_xmouse+"\t"+_ymouse+newline;
    table_str += "<b>Button</b>\t"+my_btn._xmouse+"\t"+my_btn._ymouse+newline;
    table_str += "</textformat>";
    mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

另请参见

[\\_ymouse](#) (`Button._ymouse` 属性)

## `_xscale` (`Button._xscale` 属性)

`public _xscale : Number`

从按钮注册点开始应用的按钮水平缩放比例，以百分比表示。默认注册点为 **(0,0)**。

缩放本地坐标系将影响 `_x` 和 `_y` 属性设置，这些设置是以像素为单位定义的。例如，如果父级影片剪辑缩放到 **50%**，则设置 `_x` 属性将移动按钮中的对象，移动距离为在 **SWF** 文件为 **100%** 时其像素数的一半。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 示例

下面的示例缩放名为 `my_btn` 的按钮。当您单击然后松开按钮时，它会在 `x` 和 `y` 轴上增大 **10%**。在时间轴的第 **1** 帧中输入下面的 **ActionScript**：

```
my_btn.onRelease = function(){
    this._xscale ~= 1.1;
    this._yscale ~= 1.1;
};
```

另请参见

[\\_x \(Button.\\_x 属性\)](#), [\\_y \(Button.\\_y 属性\)](#), [\\_xscale \(Button.\\_xscale 属性\)](#)

## \_y (Button.\_y 属性)

```
public _y : Number
```

按钮相对于父级影片剪辑的本地坐标的 `y` 坐标。如果按钮在主时间轴上，则其坐标系统将舞台的左上角作为 **(0, 0)**。如果按钮在具有变形的另一个影片剪辑内，则该按钮将位于包含它的影片剪辑的本地坐标系中。因此，对于逆时针旋转 **90** 度的影片剪辑，其中的按钮将继续承逆时针旋转 **90** 度的坐标系统。按钮的坐标指的是注册点的位置。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例将 `my_btn` 在舞台上的坐标设置为 **0**。创建名为 `my_btn` 的按钮，并在时间轴的第 **1** 帧中输入下面的 **ActionScript**：

```
my_btn._x = 0;
my_btn._y = 0;
```

另请参见

[\\_x \(Button.\\_x 属性\)](#), [\\_xscale \(Button.\\_xscale 属性\)](#), [\\_yscale \(Button.\\_yscale 属性\)](#)

## \_ymouse (Button.\_ymouse 属性)

```
public _ymouse : Number [read-only]
```

指示鼠标位置相对于按钮的 `y` 坐标。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例显示舞台的 `ymouse` 位置，以及一个放在舞台上的名为 `my_btn` 的按钮。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var table_str:String = "<textformat tabstops='[50,100]'\t>";
    table_str += "<b>Stage</b>\t"+_x:"+_xmouse+"\t"+_y:"+_ymouse+newline;
    table_str += "<b>Button</b>\t"+_x:"+_xmouse+"\t"+_y:"+_ymouse+newline;
    table_str += "</textformat>";
    mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

## 另请参见

[\\_xmouse \(Button.\\_xmouse 属性\)](#)

## \_yscale (Button.\_yscale 属性)

`public _yscale : Number`

从按钮注册点开始应用的按钮垂直缩放比例，以百分比表示。默认注册点为 (0,0)。

可用性：ActionScript 1.0 ； Flash Player 6

## 示例

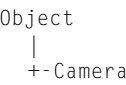
下面的示例缩放名为 `my_btn` 的按钮。当您单击然后松开按钮时，它会在 `x` 和 `y` 轴上增大 10%。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
my_btn.onRelease = function(){
    this._xscale ~ 1.1;
    this._yscale ~ 1.1;
};
```

## 另请参见

[\\_y \(Button.\\_y 属性\)](#)，[\\_x \(Button.\\_x 属性\)](#)，[\\_xscale \(Button.\\_xscale 属性\)](#)

# Camera



```
public class Camera
extends Object
```

**Camera** 类主要与 **Macromedia Flash Communication Server** 一起使用，但也可以用在服务器以外的其它地方，只是在使用上受到限制。

使用 **Camera** 类可以捕获来自连接到运行 **Macromedia Flash Player** 的计算机的视频摄像头的视频 - 例如监视来自连接到本地系统的 **Web** 摄像头的视频输入。（**Flash** 提供类似的音频功能；有关更多信息，请参见 **Microphone** 类条目。）

**警告：**当 **SWF** 文件尝试访问 **Camera.get()** 返回的摄像头时，**Flash Player** 显示“隐私”对话框，用户可从中选择是允许还是拒绝对摄像头的访问。（对于 **Camera** 类，请确保舞台大小至少为 **215 x 138** 像素；这是 **Flash** 显示该对话框所需的最小大小。）最终用户和管理用户还可以基于各个站点或全局禁用摄像头访问。

若要创建或引用 **Camera** 对象，请使用 **Camera.get()** 方法。

可用性：**ActionScript 1.0**；**Flash Player 6**

## 属性摘要

修饰符	属性	说明
	<code>activityLevel:Number</code> [ 只读 ]	数值，指定摄像头检测的运动量。
	<code>bandwidth:Number</code> [ 只读 ]	整数，指定当前输出视频输入信号可以使用的最大带宽，以字节为单位。
	<code>currentFps:Number</code> [ 只读 ]	摄像头捕获数据的速率，以每秒帧数为单位。
	<code>fps:Number</code> [ 只读 ]	希望摄像头在捕获数据时达到的最大速率，以每秒帧数为单位。
	<code>height:Number</code> [ 只读 ]	当前捕获高度，以像素为单位。
	<code>index:Number</code> [ 只读 ]	从零开始的整数，指定由 <code>Camera.names</code> 返回的数组中所反映的摄像头的索引。
	<code>motionLevel:Number</code> [ 只读 ]	数值，指定调用 <code>Camera.onActivity(true)</code> 所需的移动量。
	<code>motionTimeOut:Number</code> [ 只读 ]	摄像头停止检测运动的时刻和调用 <code>Camera.onActivity(false)</code> 的时刻之间相差的毫秒数。

修饰符	属性	说明
	muted:Boolean [ 只读 ]	布尔值，指定用户在 Flash Player 的“隐私设置”面板中是拒绝对摄像头的访问 (true) 还是允许访问 (false)。
	name:String [ 只读 ]	字符串，指定由摄像头硬件返回的当前摄像头的名称。
static	names:Array [ 只读 ]	检索一个字符串数组，该数组反映所有可用摄像头名称而不显示 Flash Player 的“隐私设置”面板。
	quality:Number [ 只读 ]	整数，指定所需的画面质量级别，该级别由应用于每一视频帧的压缩量确定。
	width:Number [ 只读 ]	当前捕获宽度，以像素为单位。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

事件摘要

事件	说明
onActivity = function(active:Boolean) {}	在摄像头开始或停止检测运动时调用。
onStatus = function(infoObject:Object) {}	当用户允许或拒绝访问摄像头时调用。

方法摘要

修饰符	签名	说明
static	get([index:Number]) : Camera	返回对用于捕获视频的 Camera 对象的引用。
	setMode([width:Number], [height:Number], [fps:Number], [favorArea:Boolean]) : Void	将摄像头的捕获模式设置为最符合指定要求的本机模式。

修饰符	签名	说明
	<code>setMotionLevel([motionLevel:Number], [timeOut:Number]) : Void</code>	指定调用 <code>Camera.onActivity(true)</code> 所需的移动量。
	<code>setQuality([bandwidth:Number], [quality:Number]) : Void</code>	设置每秒的最大带宽或当前输出视频输入信号所需的图片品质。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## activityLevel (Camera.activityLevel 属性)

`public activityLevel : Number [read-only]`

数值，指定摄像头检测的运动量。该数值的范围是从 **0**（检测到没有运动）到 **100**（检测到大运动量）。该属性的值可以帮助您确定是否需要将设置传递到

`Camera.setMotionLevel()`。

如果摄像头可用，但却因为尚未调用 `Video.attachVideo()` 而未被使用，则此属性设置为 **-1**。

如果只对未压缩的本地视频进行流式处理，则只有在为 `Camera.onActivity` 事件处理函数分配函数后才能设置此属性。否则，它为未定义。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

此示例使用 `activityLevel` 属性和一个 **ProgressBar** 实例来检测摄像头检测到的移动量。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加 **ProgressBar** 组件实例，并为它指定实例名称 `activity_pb`。然后将下面的 **ActionScript** 添加到时间轴的第 **1** 帧中：

```
// video instance on the Stage.  
var my_video:Video;  
var activity_pb:mx.controls.ProgressBar;  
var my_cam:Camera = Camera.get();  
my_video.attachVideo(my_cam);  
activity_pb.mode = "manual";
```



```

activity_pb.label = "Activity %3%";

this.onEnterFrame = function() {
    activity_pb.setProgress(my_cam.activityLevel, 100);
};
my_cam.onActivity = function(isActive:Boolean) {
    var themeColor:String = (isActive) ? "haloGreen" : "haloOrange";
    activity_pb.setStyle("themeColor", themeColor);
};

```

另请参见

[motionLevel \(Camera.motionLevel 属性\)](#), [setMotionLevel \(Camera.setMotionLevel 方法\)](#)

## bandwidth (Camera.bandwidth 属性)

public bandwidth : Number [read-only]

整数，指定当前输出视频输入信号可以使用的最大带宽，以字节为单位。值 0 意味着 Flash 视频可以使用所需数量的带宽来保持指定的帧品质。

若要设置此属性，请使用 `Camera.setQuality()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例更改摄像头输入信号使用的最大带宽量。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。将 **NumericStepper** 组件实例添加到舞台，并为它指定实例名称 `bandwidth_nstep`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```

var bandwidth_nstep:mx.controls.NumericStepper;
var my_video:Video;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
this.createTextField("bandwidth_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
bandwidth_txt.autoSize = true;
this.onEnterFrame = function() {
    bandwidth_txt.text = "Camera is currently using "+my_cam.bandwidth+" bytes (" +Math.round(my_cam.bandwidth/1024)+" KB) bandwidth.";
};
//
bandwidth_nstep.minimum = 0;
bandwidth_nstep.maximum = 128;
bandwidth_nstep.stepSize = 16;
bandwidth_nstep.value = my_cam.bandwidth/1024;
function changeBandwidth(evt:Object) {

```

```
my_cam.setQuality(evt.target.value 1024, 0);
}
bandwidth_nstep.addEventListener("change", changeBandwidth);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类代替 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setQuality](#) ([Camera.setQuality](#) 方法)

## currentFps (Camera.currentFps 属性)

```
public currentFps : Number [read-only]
```

摄像头捕获数据的速率，以每秒帧数为单位。无法设置此属性；不过，您可以使用 `Camera.setMode()` 方法来设置相关属性 `Camera.fps`，该属性指定您希望摄像头捕获数据的最大帧频。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例使用 `currentFps` 属性和 **ProgressBar** 实例检测摄像头捕获数据的速率，以每秒帧数为单位。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加 **ProgressBar** 组件实例，并为它指定实例名称 `fps_pb`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var my_video:Video;
var fps_pb:mx.controls.ProgressBar;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
this.onEnterFrame = function() {
    fps_pb.setProgress(my_cam.fps-my_cam.currentFps, my_cam.fps);
};
```

```
fps_pb.setStyle("fontSize", 10);
fps_pb.setStyle("themeColor", "haloOrange");
fps_pb.labelPlacement = "top";
fps_pb.mode = "manual";
fps_pb.label = "FPS: %2 (%3%% dropped)";
```

另请参见

[setMode](#) ([Camera.setMode](#) 方法)，[fps](#) ([Camera.fps](#) 属性)

## fps (Camera.fps 属性)

public fps : Number [read-only]

希望摄像头在捕获数据时达到的最大速率，以每秒帧数为单位。可能的最大速率取决于摄像头的性能；即，如果摄像头不支持您在此处设置的值，则不会达到此帧速率。

- 若要为该属性设置所需值，请使用 `Camera.setMode()`。
- 若要确定摄像头当前捕获数据的速率，请使用 `Camera.currentFps` 属性。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例使用 `currentFps` 属性和 **ProgressBar** 实例检测摄像头捕获数据的速率，以每秒帧数为单位。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加 **ProgressBar** 组件实例，并为它指定实例名称 `fps_pb`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var my_video:Video;
var fps_pb:mx.controls.ProgressBar;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
this.onEnterFrame = function() {
    fps_pb.setProgress(my_cam.fps-my_cam.currentFps, my_cam.fps);
};

fps_pb.setStyle("fontSize", 10);
fps_pb.setStyle("themeColor", "haloOrange");
fps_pb.labelPlacement = "top";
fps_pb.mode = "manual";
fps_pb.label = "FPS: %2 (%3%% dropped)";
```



`setMode()` 函数不一定是所要求的 fps 设置；该函数设置所要求的 fps 或可用的最快 fps。

另请参见

[currentFps \(Camera.currentFps 属性\)](#)，[setMode \(Camera.setMode 方法\)](#)

## get (Camera.get 方法)

```
public static get([index:Number]) : Camera
```

返回对用于捕获视频的 **Camera** 对象的引用。若要实际开始捕获视频，必须将 **Camera** 对象附加到 **Video** 对象（请参见 `Video.attachVideo()`）。

与使用 `new` 构造函数创建的对象不同，对 `Camera.get()` 的多个调用将引用同一摄像头。因此，如果您的脚本包含 `first_cam = Camera.get()` 和 `second_cam = Camera.get()` 行，则 `first_cam` 和 `second_cam` 均将引用同一个（默认的）摄像头。

通常，不应传递 `index` 的值；只使用 `Camera.get()` 来返回对默认摄像头的引用。通过“摄像头设置”面板（将在本节的后面进行讨论），用户可以指定 **Flash** 应使用的默认摄像头。如果您传递 `index` 的值，则可能会尝试引用非用户首选的摄像头。在极少数情况下（例如，您的应用程序同时从两个摄像头捕获视频），您可以使用 `index`。

当 **SWF** 文件尝试访问 `Camera.get()` 返回的摄像头时，**Flash Player** 显示“隐私”对话框，用户可从中选择是允许还是拒绝对摄像头的访问。（请确保舞台大小至少为 215 x 138 像素；这是 **Flash** 显示该对话框所需的最小大小。）

当用户对此对话框做出响应时，`Camera.onStatus` 事件处理函数将返回指示用户响应的信息对象。若要不处理此事件处理函数就确定用户是拒绝还是允许对摄像头的访问，请使用 `Camera.muted` 属性。

用户也可以为特定域指定永久隐私设置，方法是在 **SWF** 文件播放过程中右键单击（在 **Windows** 中）或按住 **Control** 键并单击（在 **Macintosh** 中），选择“设置”，打开“隐私”面板，然后选择“记住”。

您不能使用 **ActionScript** 来设置用户的 **Allow** 或 **Deny** 值，但可以通过使用 `System.showSettings(0)` 来为用户显示“隐私”面板。如果用户选择“记住”，则 **Flash Player** 不再为此域的 **SWF** 文件显示“隐私”对话框。

如果 `Camera.get` 返回 `null`，则表明摄像头正由其它应用程序使用，或者在系统上没有安装摄像头。若要确定是否安装了摄像头，请使用 `Camera.names.length`。若要显示 **Flash Player** 的“摄像头设置”面板（让用户选择由 `Camera.get()` 引用的摄像头），请使用 `System.showSettings(3)`。

扫描硬件以找到摄像头需要花一些时间。在 **Flash** 找到至少一个摄像头后，在该播放器实例的生存期内不再扫描硬件。不过，如果 **Flash** 没有找到任何摄像头，则在每次调用 `Camera.get` 时都进行扫描。这在用户忘记连接摄像头时非常有用；如果您的 **SWF** 文件提供了调用 `Camera.get` 的“重试”按钮，则用户不必重新启动该 **SWF** 文件，**Flash** 即可找到摄像头。



正确的语法为 `Camera.get()`。若要分配 `Camera` 对象到变量，请使用类似于 `active_cam = Camera.get()` 的语法。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**index:**Number [ 可选 ] - 从零开始的整数, 指定要获取的摄像头, 该整数根据 `Camera.names` 属性返回的数组确定。若要获取默认的摄像头 (建议用于大多数应用程序), 请省略此参数。

## 返回

Camera - 如果未指定 `index`, 则此方法返回对默认摄像头的引用; 或者, 如果默认摄像头正由其它应用程序使用, 则此方法返回对第一个可用摄像头的引用。(如果安装了多个摄像头, 则用户可以在 **Flash Player** 的“摄像头设置”面板中指定默认的摄像头。)如果没有可用摄像头或没有安装摄像头, 则该方法返回 `null`。如果指定了 `index`, 则此方法返回对请求的摄像头的引用; 如果请求的摄像头不可用, 则返回 `null`。

## 示例

下面的示例使您可以从 **ComboBox** 实例中选择一个要使用的活动摄像头。当前的活动摄像头显示在 **Label** 实例中。在“库”选项菜单中选择“新建视频”, 以创建新的视频实例。在舞台上添加实例, 并为它指定实例名称 `my_video`。在舞台上添加一个 **Label** 组件实例, 并为它指定实例名称 `camera_lbl`, 再添加一个 **ComboBox** 组件实例, 并为它指定实例名称 `cameras_cb`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
var camera_lbl:mx.controls.Label;
var cameras_cb:mx.controls.ComboBox;
camera_lbl.text = my_cam.name;
cameras_cb.dataProvider = Camera.names;
function changeCamera():Void {
    my_cam = Camera.get(cameras_cb.selectedIndex);
    my_video.attachVideo(my_cam);
    camera_lbl.text = my_cam.name;
}
cameras_cb.addEventListener("change", changeCamera);
camera_lbl.setStyle("fontSize", 9);
cameras_cb.setStyle("fontSize", 9);
```

## 另请参见

[index \(Camera.index 属性\)](#), [muted \(Camera.muted 属性\)](#), [names \(Camera.names 属性\)](#), [onStatus \(Camera.onStatus 处理函数\)](#), [setMode \(Camera.setMode 方法\)](#), [showSettings \(System.showSettings 方法\)](#), [attachVideo \(Video.attachVideo 方法\)](#)

## height (Camera.height 属性)

public height : Number [read-only]

当前捕获高度，以像素为单位。若要设置此属性的值，请使用 `Camera.setMode()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的代码显示舞台上的 **Label** 组件实例中一个视频实例的当前宽度、高度和 FPS。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加 **Label** 组件实例，并为它指定实例名称 `dimensions_lbl`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
var dimensions_lbl:mx.controls.Label;
dimensions_lbl.setStyle("fontSize", 9);
dimensions_lbl.setStyle("fontWeight", "bold");
dimensions_lbl.setStyle("textAlign", "center");
dimensions_lbl.text = "width: "+my_cam.width+", height: "+my_cam.height+",
    FPS: "+my_cam.fps;
```

另请参见 `Camera.setMode()` 的示例。

### 另请参见

[width \(Camera.width 属性\)](#), [setMode \(Camera.setMode 方法\)](#)

## index (Camera.index 属性)

public index : Number [read-only]

从零开始的整数，指定由 `Camera.names` 返回的数组中所反映的摄像头的索引。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例在一个运行时创建的文本字段中显示摄像头的数组，并告诉您当前正在使用的是哪个摄像头。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加 **Label** 组件实例，并为它指定实例名称 `camera_lbl`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var camera_lbl:mx.controls.Label;
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
```

```

camera_lbl.text = my_cam.index+" ".my_cam.name;
this.createTextField("cameras_txt", this.getNextHighestDepth(), 25, 160,
    160, 80);
cameras_txt.html = true;
cameras_txt.border = true;
cameras_txt.wordWrap = true;
cameras_txt.multiline = true;
for (var i = 0; i<Camera.names.length; i++) {
    cameras_txt.htmlText += "<li><u><a
        href=\"asfunction:changeCamera,\"+i+\">\"+Camera.names[i]+\"</a></u></li>";
}
function changeCamera(index:Number) {
    my_cam = Camera.get(index);
    my_video.attachVideo(my_cam);
    camera_lbl.text = my_cam.index+" ".my_cam.name;
}

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类代替 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[names](#) (`Camera.names` 属性), [get](#) (`Camera.get` 方法)

## motionLevel (Camera.motionLevel 属性)

`public motionLevel : Number [read-only]`

数值，指定调用 `Camera.onActivity(true)` 所需的移动量。可接受值的范围为从 **0** 到 **100**。默认值为 **50**。

不论 `motionLevel` 属性为何值都能显示视频。有关更多信息，请参阅 `Camera.setMotionLevel()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例连续检测摄像头输入信号的运动级别。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加一个 **Label** 组件实例，并为它指定实例名称 `motionLevel_lbl`；添加一个 **NumericStepper** 组件实例，并为它指定实例名称 `motionLevel_nstep`；再添加一个 **ProgressBar** 组件实例，并为它指定实例名称 `motion_pb`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```

var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

```

```

// configure the ProgressBar component instance
var motion_pb:mx.controls.ProgressBar;
motion_pb.mode = "manual";
motion_pb.label = "Motion: %3%%";

var motionLevel_lbl:mx.controls.Label;
// configure the NumericStepper component instance
var motionLevel_nstep:mx.controls.NumericStepper;
motionLevel_nstep.minimum = 0;
motionLevel_nstep.maximum = 100;
motionLevel_nstep.stepSize = 5;
motionLevel_nstep.value = my_cam.motionLevel;

// Continuously update the progress of the ProgressBar component instance to
    the activityLevel
// of the current Camera instance, which is defined in my_cam
this.onEnterFrame = function() {
    motion_pb.setProgress(my_cam.activityLevel, 100);
};

// When the level of activity goes above or below the number defined in
    Camera.motionLevel,
// trigger the onActivity event handler.
my_cam.onActivity = function(isActive:Boolean) {
    // If isActive equals true, set the themeColor variable to "haloGreen".
    // Otherwise set the themeColor to "haloOrange".
    var themeColor:String = (isActive) ? "haloGreen" : "haloOrange";
    motion_pb.setStyle("themeColor", themeColor);
};

function changeMotionLevel() {
    // Set the motionLevel property for my_cam Camera instance to the value of
    the NumericStepper
    // component instance. Maintain the current motionTimeOut value of the
    my_cam Camera instance.
    my_cam.setMotionLevel(motionLevel_nstep.value, my_cam.motionTimeOut);
}
motionLevel_nstep.addEventListener("change", changeMotionLevel);

```

## 另请参见

[onActivity](#) ([Camera.onActivity](#) 处理函数), [onStatus](#) ([Camera.onStatus](#) 处理函数),  
[setMotionLevel](#) ([Camera.setMotionLevel](#) 方法), [activityLevel](#)  
 ([Camera.activityLevel](#) 属性)



## motionTimeout (Camera.motionTimeout 属性)

```
public motionTimeout : Number [read-only]
```

摄像头停止检测运动的时刻和调用 `Camera.onActivity (false)` 的时刻之间相差的毫秒数。默认值为 **2000** (2 秒)。

若要设置此值，请使用 `Camera.setMotionLevel()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

在下面的示例中，**ProgressBar** 实例在活动级别低于运动级别时更改其光晕主题颜色。可以使用 **NumericStepper** 实例设置 `motionTimeout` 属性的秒数。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加一个 **Label** 组件实例，并为它指定实例名称 `motionLevel_lbl`；添加一个 **NumericStepper** 组件实例，并为它指定实例名称 `motionTimeout_nstep`；再添加一个 **ProgressBar** 组件实例，并为它指定实例名称 `motion_pb`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var motionLevel_lbl:mx.controls.Label;
var motion_pb:mx.controls.ProgressBar;
var motionTimeout_nstep:mx.controls.NumericStepper;
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

this.onEnterFrame = function() {
    motionLevel_lbl.text = "activityLevel: "+my_cam.activityLevel;
};

motion_pb.indeterminate = true;
my_cam.onActivity = function(isActive:Boolean) {
    if (isActive) {
        motion_pb.setStyle("themeColor", "haloGreen");
        motion_pb.label = "Motion is above "+my_cam.motionLevel;
    } else {
        motion_pb.setStyle("themeColor", "haloOrange");
        motion_pb.label = "Motion is below "+my_cam.motionLevel;
    }
};

function changeMotionTimeout() {
    my_cam.setMotionLevel(my_cam.motionLevel, motionTimeout_nstep.value 1000);
}
motionTimeout_nstep.addEventListener("change", changeMotionTimeout);
motionTimeout_nstep.value = my_cam.motionTimeout/1000;
```

另请参见

[setMotionLevel](#) ([Camera.setMotionLevel](#) 方法), [onActivity](#) ([Camera.onActivity](#) 处理函数)

## muted (Camera.muted 属性)

```
public muted : Boolean [read-only]
```

布尔值, 指定用户在 **Flash Player** 的“隐私设置”面板中是拒绝对摄像头的访问 (`true`) 还是允许访问 (`false`)。当更改此值时, 将调用 `Camera.onStatus`。有关更多信息, 请参阅 `Camera.get()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

在下面的示例中, 如果 `my_cam.muted` 的计算结果为 **true**, 就会显示一条错误消息。在“库”选项菜单中选择“新建视频”, 以创建新的视频实例。在舞台上添加实例, 并为它指定实例名称 `my_video`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
my_cam.onStatus = function(infoObj:Object) {
    if (my_cam.muted) {
        // If user is denied access to their Camera, you can display an error
        message here. You can display the user's Camera/Privacy settings again
        using System.showSettings(0);
        trace("User denied access to Camera");
        System.showSettings(0);
    }
};
```

另请参见

[get](#) ([Camera.get](#) 方法), [onStatus](#) ([Camera.onStatus](#) 处理函数)

## name (Camera.name 属性)

```
public name : String [read-only]
```

字符串，指定由摄像头硬件返回的当前摄像头的名称。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例在文本字段中显示默认摄像头的名称。在 **Windows** 中，该名称与“扫描仪和照相机”控制面板中列出的设备名称相同。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。然后将下面的

**ActionScript** 添加到时间轴的第 1 帧中：

```
var my_cam:Camera = Camera.get();  
var my_video:Video;  
my_video.attachVideo(my_cam);
```

```
this.createTextField("name_txt", this.getNextHighestDepth(), 0, 0, 100, 22);  
name_txt.autoSize = true;  
name_txt.text = my_cam.name;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类代替 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[get \(Camera.get 方法\)](#)，[names \(Camera.names 属性\)](#)

## names (Camera.names 属性)

```
public static names : Array [read-only]
```

检索一个字符串数组，该数组反映所有可用摄像头名称而不显示 **Flash Player** 的“隐私设置”面板。此数组的行为与其它任何 **ActionScript** 数组的行为相同，隐式提供每一摄像头从零开始的索引以及系统上摄像头的数目（通过 `Camera.names.length`）。有关更多信息，请参见 `Camera.names Array` 类条目。

调用 `Camera.names` 属性要求全面检查硬件，并可能需要几秒钟时间才能生成数组。在大多数情况下，您只能使用默认的摄像头。



正确的语法为 `Camera.names`。若要将该返回值分配到变量，请使用类似于 `cam_array = Camera.names` 的语法。若要确定当前摄像头的名称，请使用 `active_cam.name`。

可用性: **ActionScript 1.0 ; Flash Player 6**

## 示例

下面的示例在没有多个可用摄像机的情况下使用默认的摄像机；如果有多个可用摄像机，用户可以选择将哪个摄像机设置为默认摄像机。如果仅显示一个摄像头，则使用的是默认摄像头。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var my_video:Video;
var cam_array:Array = Camera.names;
if (cam_array.length>1) {
    System.showSettings(3);
}
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
```

## 另请参见

[get \(Camera.get 方法\)](#)，[index \(Camera.index 属性\)](#)，[name \(Camera.name 属性\)](#)

## onActivity (Camera.onActivity 处理函数)

```
onActivity = function(active:Boolean) {}
```

在摄像头开始或停止检测运动时调用。如果要对此事件处理函数做出响应，则必须创建一个函数来处理其活动值。

若要指定调用 `Camera.onActivity(true)` 所需的移动量以及必须经过多少时间没有活动才能调用 `Camera.onActivity(false)`，请使用 `Camera.setMotionLevel()`。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**active:Boolean** - 布尔值，当摄像头开始检测运动时，设置为 **true**；当运动停止时设置为 **false**。

## 示例

下面的示例在摄像头开始或停止检测运动时在“输出”面板中显示 **true** 或 **false**。

```
// Assumes a Video object named "myVideoObject" is on the Stage
active_cam = Camera.get();
myVideoObject.attachVideo(active_cam);
active_cam.setMotionLevel(10, 500);
active_cam.onActivity = function(mode)
{
    trace(mode);
}
```

## 另请参见

[setMotionLevel \(Camera.setMotionLevel 方法\)](#)

## onStatus (Camera.onStatus 处理函数)

```
onStatus = function(info:Object) {}
```

当用户允许或拒绝访问摄像头时调用。如果您要响应此事件处理函数，必须创建一个函数来处理摄像头生成的信息对象。

当一个 SWF 文件尝试访问摄像头时，Flash Player 显示“隐私”对话框，用户可以选择是允许还是拒绝访问。

- 如果用户允许访问，则 Camera.muted 属性设置为 **false**，并且用 **code** 属性为“Camera.Unmuted”且 **level** 属性为“Status”的一个信息对象调用此事件处理函数。
- 如果用户拒绝访问，则 Camera.muted 属性设置为 **true**，并且用 **code** 属性为“Camera.Muted”且 **level** 属性为“Status”的一个信息对象调用此事件处理函数。

若要不处理此事件处理函数就确定用户是拒绝还是允许对摄像头的访问，请使用 Camera.muted 属性。

注意：如果用户选择永久允许或拒绝对来自指定域的所有 SWF 文件的访问，则对于来自该域的 SWF 文件不调用此处理函数，除非用户以后更改该隐私设置。有关更多信息，请参阅 Camera.get()。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**info:Object** - 根据状态消息定义的参数。

### 示例

每当用户允许或拒绝对摄像头的访问时，下面的 ActionScript 都用于显示消息：

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
my_cam.onStatus = function(info:Object) {
    switch (infoObj.code) {
        case 'Camera.Muted' :
            trace("Camera access is denied");
            break;
        case 'Camera.Unmuted' :
            trace("Camera access granted");
            break;
    }
}
```

### 另请参见

[get](#) (Camera.get 方法), [muted](#) (Camera.muted 属性), [showSettings](#) (System.showSettings 方法), [onStatus](#) (System.onStatus 处理函数)

## quality（Camera.quality 属性）

public quality : Number [read-only]

整数，指定所需的画面质量级别，该级别由应用于每一视频帧的压缩量确定。可接受的品质值范围为从 1（最低品质，最大压缩）到 100（最高品质，无压缩）。默认值为 0，这意味着图片品质可以根据需要进行变化，以避免超出可用带宽。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例使用 **NumericStepper** 实例指定应用于摄像头输入信号的压缩量。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 my\_video。添加一个实例名称为 **quality\_nstep** 的 **NumericStepper**。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var quality_nstep:mx.controls.NumericStepper;

var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

quality_nstep.minimum = 0;
quality_nstep.maximum = 100;
quality_nstep.stepSize = 5;
quality_nstep.value = my_cam.quality;

function changeQuality() {
    my_cam.setQuality(my_cam.bandwidth, quality_nstep.value);
}
quality_nstep.addEventListener("change", changeQuality);
```

另请参见

[setQuality（Camera.setQuality 方法）](#)

## setMode (Camera.setMode 方法)

```
public setMode([width:Number], [height:Number], [fps:Number],  
               [favorArea:Boolean]) : Void
```

将摄像头的捕获模式设置为最符合指定要求的本机模式。如果摄像头不具有与您传递的所有参数相匹配的本机模式，则 **Flash** 选择最接近于请求模式的合成的捕获模式。此操作可能涉及裁切图像和删除帧。

默认情况下，**Flash** 根据需要删除帧，以保持图像大小。若要将删除的帧的数目降至最低（即使这样做意味着减小图像的大小），请将 `false` 传递给 `favorArea` 参数。

在选择本机模式时，**Flash** 尝试尽可能保持请求的纵横比。例如，如果您发出 `active_cam.setMode(400, 400, 30)` 命令，并且摄像头上可用的最大 `width` 和 `height` 值分别为 320 和 288，则 **Flash** 将 `width` 和 `height` 都设置为 288；通过将这些属性设置为相同的值，**Flash** 可以保持所请求的 1:1 的高宽比。

若要确定 **Flash** 在选择与您请求的值最匹配的模式后分配给这些属性的值，请使用 `Camera.width`、`Camera.height` 和 `Camera.fps`。

可用性：ActionScript 1.0；Flash Player 6

### 参数

`width:Number` [可选] - 请求的捕获宽度，以像素为单位。默认值为 160。

`height:Number` [可选] - 请求的捕获高度，以像素为单位。默认值为 120。

`fps:Number` [可选] - 摄像头捕获数据应使用的请求速率，以每秒帧数为单位。默认值为 15。

`favorArea:Boolean` [可选] - 布尔值，指定如果摄像头不具有满足指定要求的本机模式时如何控制宽度、高度和帧频。默认值为 `true`，这意味着支持保持捕获大小；使用此参数选择与 `width` 和 `height` 值最匹配的模式，即使这样做会由于降低帧频而对性能有不利影响。若要最大限度地提高帧频，而不考虑摄像头的高度和宽度，请将 `false` 传递给 `favorArea` 参数。

### 示例

下面的示例设置摄像头的捕获模式。可以将帧频键入到 `TextInput` 实例中，然后按 **Enter** 或 **Return** 以应用帧频。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。添加实例名称为 `fps_ti` 的 `TextInput` 组件实例。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var my_cam:Camera = Camera.get();  
var my_video:Video;  
my_video.attachVideo(my_cam);  
  
fps_ti.maxChars = 2;  
fps_ti.restrict = [0-9];  
fps_lbl.text = "Current: "+my_cam.fps+" fps";
```

```
function changeFps():Void {
    my_cam.setMode(my_cam.width, my_cam.height, fps_ti.text);
    fps_lbl.text = "Current: "+my_cam.fps+" fps";
    fps_ti.text = my_cam.fps;
    Selection.setSelection(0,2);
}
fps_ti.addEventListener("enter", changeFps);
```

另请参见

[fps](#) (Camera.fps 属性), [height](#) (Camera.height 属性), [width](#) (Camera.width 属性), [currentFps](#) (Camera.currentFps 属性)

## setMotionLevel (Camera.setMotionLevel 方法)

```
public setMotionLevel([motionLevel:Number], [timeOut:Number]) : Void
```

指定调用 Camera.onActivity(true) 所需的移动量。或者，设置必须经过多少没有活动的时间（按毫秒数计算），**Flash** 才会认为动作已停止并调用 Camera.onActivity(false)。



不论 sensitivity 参数为何值都能显示视频。此参数只确定在何时以及哪些情况下调用 Camera.onActivity，而并不确定实际上是捕获视频还是显示视频。

- 若要完全禁止摄像头检测动作，则将值 **100** 传递到 *sensitivity*：永远不调用 Camera.onActivity。（您可能只将此值用作测试目的，例如暂时禁用调用 Camera.onActivity 时发生的任何动作。）
- 若要确定摄像头当前检测到的移动量，请使用 Camera.activityLevel 属性。
- 运动敏感度值直接对应于活动值。完全不运动时活动值为 **0**。持续运动时活动值为 **100**。当您不移动时，活动值小于运动敏感度值；当您移动时，活动值通常会超过运动敏感度值。
- 此方法的用途与 Microphone.setSilenceLevel() 相同；这两种方法都用于指定应该在何时调用 onActivity 事件处理函数。但是这些方法对发布流具有非常不同的影响：
- Microphone.setSilenceLevel() 设计用于优化带宽。在认为音频流被静音时，不发送任何音频数据。所发送的是一个指示静音已启动的消息。
- Camera.setMotionLevel() 设计用于检测动作，它不影响带宽用量。即使视频流未检测到运动，仍将发送视频。

可用性：ActionScript 1.0；Flash Player 6



## 参数

**motionLevel**:Number [可选] - 数值, 指定调用 `Camera.onActivity(true)` 所需的移动量。可接受值的范围为从 0 到 100。默认值为 50。

**timeOut**:Number [可选] - 数字参数, 指定必须经过多少没有活动的时间 (按毫秒数计算), **Flash** 才会认为活动已停止并调用 `Camera.onActivity(false)` 事件处理函数。默认值为 2000 (2 秒)。

## 示例

下面的示例在视频活动开始或结束时将消息发送到“输出”面板。将值为 30 的运动敏感度值更改为更高或更低的数值, 看一下不同的值是如何影响运动检测的。

```
// Assumes a Video object named "myVideoObject" is on the Stage
active_cam = Camera.get();
x = 0;
function motion(mode) {
    trace(x + ": " + mode);
    x++;
}
active_cam.onActivity = function(mode) {
    motion(mode);
}
active_cam.setMotionLevel(30, 500);
myVideoObject.attachVideo(active_cam);
```

## 另请参见

[motionLevel](#) ([Camera.motionLevel](#) 属性), [motionTimeOut](#) ([Camera.motionTimeOut](#) 属性), [onActivity](#) ([Camera.onActivity](#) 处理函数), [activityLevel](#) ([Camera.activityLevel](#) 属性)

## setQuality (Camera.setQuality 方法)

```
public setQuality([bandwidth:Number], [quality:Number]) : Void
```

设置每秒的最大带宽或当前输出视频输入信号所需的图片品质。此方法通常只在您使用 **Flash Communication Server** 传输视频时适用。

使用此方法可以指定输出视频输入信号的哪一方面对于您的应用程序更重要: 是带宽使用率还是图片品质。

- 若要指示带宽使用率更为重要, 请将一个值传递给 *bandwidth* 并将 0 传递给 *frameQuality*。Flash 将在指定的带宽内以可能的最高品质传输视频。如有必要, Flash 将降低图片品质以避免超出指定的带宽。通常, 随着运动的增加, 品质将降低。

- 若要指示品质更为重要，请将 **0** 传递给 *bandwidth* 并将一个数值传递给 *frameQuality*。**Flash** 将根据需要使用尽量多的带宽来保持指定的品质。如有必要，**Flash** 将降低帧速率以保持图片品质。通常，随着运动的增加，带宽的使用率也将增加。
- 若要指定带宽和品质同等重要，请为这两个参数都传递数值。**Flash** 将传输达到指定品质并且不超过指定带宽的视频。如有必要，**Flash** 将降低帧速率以在不超出指定的带宽的情况下保持图片品质。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**bandwidth**:Number [ 可选 ] - 整数，指定当前输出视频可以使用的最大带宽量，以每秒字节数为单位。若要指定 **Flash** 视频可以使用所需的任何数量的带宽来保持 *frameQuality* 的值，则将 **0** 传递给 *bandwidth*。默认值为 **16384**。

**quality**:Number [ 可选 ] - 整数，指定所需的画面质量的级别，该整数由应用于每一视频帧的压缩量确定。可接受的值范围从 **1**（最低品质，最大压缩）到 **100**（最高品质，无压缩）。若要指定画面质量可以根据需要进行变化以避免超出带宽，请将 **0** 传递给 *quality*。默认值是 **0**。

### 示例

下面的示例举例说明如何使用此方法来控制带宽使用率和图片品质。

```
// Ensure that no more than 8192 (8K/second) is used to send video
active_cam.setQuality(8192,0);

// Ensure that no more than 8192 (8K/second) is used to send video
// with a minimum quality of 50
active_cam.setQuality(8192,50);

// Ensure a minimum quality of 50, no matter how much bandwidth it takes
active_cam.setQuality(0,50);
```

### 另请参见

[get\(Camera.get 方法\)](#), [quality\(Camera.quality 属性\)](#), [bandwidth\(Camera.bandwidth 属性\)](#)

## width (Camera.width 属性)

public width : Number [read-only]

当前捕获宽度，以像素为单位。若要为该属性设置所需值，请使用 `Camera.setMode()`。

可用性: **ActionScript 1.0**； **Flash Player 6**

### 示例

下面的代码显示舞台上的 **Label** 组件实例中一个视频实例的当前宽度、高度和 FPS。在“库”选项菜单中选择“新建视频”，以创建新的视频实例。在舞台上添加实例，并为它指定实例名称 `my_video`。在舞台上添加 **Label** 组件实例，并为它指定实例名称 **dimensions\_lbl**。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
var dimensions_lbl:mx.controls.Label;
dimensions_lbl.setStyle("fontSize", 9);
dimensions_lbl.setStyle("fontWeight", "bold");
dimensions_lbl.setStyle("textAlign", "center");
dimensions_lbl.text = "width: "+my_cam.width+", height: "+my_cam.height+",
    FPS: "+my_cam.fps;
```

另请参见 `Camera.setMode()` 的示例。

### 另请参见

[height \(Camera.height 属性\)](#)，[setMode \(Camera.setMode 方法\)](#)

## capabilities (System.capabilities)

Object

|

+System.capabilities

```
public class capabilities
extends Object
```

**Capabilities** 类确定承载 SWF 文件的系统和播放器的功能，以便您可以定制不同格式的内容。例如，移动电话的屏幕（黑白，100 x 100 像素）就与 1000 x 1000 像素的彩色 PC 屏幕不同。为了向尽可能多的用户提供适当的内容，可以使用 `System.capabilities` 对象来确定用户所拥有的设备的类型。然后，您可以指定服务器根据设备的功能发送不同的 SWF 文件，或者通知 SWF 文件根据设备的功能变更其播放方式。

您可以使用 GET 或 POST HTTP 方法来发送功能信息。以下示例显示了具有 MP3 支持、1600 x 1200 像素分辨率、运行 Windows XP 和 Flash Player 8 (8.0.0.0) 的计算机的服务端字符串：

```
A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&
SB=f&DEB=t&V=WIN%20%2C0%2C0%2C0&M=Macromedia%20Windows&
R=1600x1200&DP=72&COL=color&AR=1.0&OS=Windows%20XP&
L=en&PT=External&AVD=f&LFD=f&WD=f"
```

System.capabilities 对象的所有属性都是只读的。

可用性：ActionScript 1.0 ； Flash Player 6

属性摘要

修饰符	属性	说明
static	avHardwareDisable: Boolean [ 只读 ]	布尔值，指定对用户的摄像头和麦克风的访问是已经通过管理方式禁止(true)还是允许(false)。
static	hasAccessibility: Boolean [ 只读 ]	布尔值，如果播放器正在支持 Flash Player 与辅助功能之间进行通讯的环境中运行，则为 true ； 否则为 false。
static	hasAudio: Boolean [ 只读 ]	指定系统是否有音频功能。
static	hasAudioEncoder: Boolean [ 只读 ]	指定 Flash Player 是否可对音频流进行编码。
static	hasEmbeddedVideo: Boolean [ 只读 ]	布尔值，如果播放器正在支持嵌入视频的系统上运行，则为 true ； 否则为 false。
static	hasIME: Boolean [ 只读 ]	指示系统是否安装了输入法编辑器 (IME)。
static	hasMP3: Boolean [ 只读 ]	指定系统是否有 MP3 解码器。
static	hasPrinting: Boolean [ 只读 ]	布尔值，如果播放器正在支持打印的系统上运行，则为 true ； 否则为 false。
static	hasScreenBroadcast: Boolean [ 只读 ]	布尔值，如果播放器支持开发通过 Flash Communication Server 运行的屏幕广播应用程序，则为 true ； 否则为 false。
static	hasScreenPlayback: Boolean [ 只读 ]	布尔值，如果播放器支持正在播放通过 Flash Communication Server 运行的屏幕广播应用程序，则为 true ； 否则为 false。

修饰符	属性	说明
static	hasStreamingAudio: Boolean [ 只读 ]	布尔值, 如果播放器可以播放音频流, 则为 true; 否则为 false。
static	hasStreamingVideo: Boolean [ 只读 ]	布尔值, 如果播放器可以播放视频流, 则为 true; 否则为 false。
static	hasVideoEncoder: Boolean [ 只读 ]	指定 Flash Player 是否可对视频流进行编码。
static	isDebugger: Boolean [ 只读 ]	布尔值, 指示播放器是正式发布的版本 (false), 还是特殊的调试版本 (true)。
static	language: String [ 只读 ]	指示运行播放器的系统的语言。
static	localFileReadDisable: Boolean [ 只读 ]	布尔值, 指示对用户硬盘的读取权限是已经通过管理方式禁止 (true) 还是允许 (false)。
static	manufacturer: String [ 只读 ]	字符串, 指示 Flash Player 制造商, 其格式为 "Macromedia OSName" (其中 OSName 可以为 "Windows"、"Macintosh"、"Linux" 或 "Other OS Name")。
static	os: String [ 只读 ]	字符串, 指示当前的操作系统。
static	pixelAspectRatio: Number [ 只读 ]	整数, 指示屏幕的像素高宽比。
static	playerType: String [ 只读 ]	字符串, 指示播放器的类型。
static	screenColor: String [ 只读 ]	字符串, 指示屏幕的颜色。
static	screenDPI: Number [ 只读 ]	数字, 指示屏幕的每英寸点数 (dpi) 分辨率, 以像素为单位。
static	screenResolutionX: Number [ 只读 ]	整数, 指示屏幕的最大水平分辨率。
static	screenResolutionY: Number [ 只读 ]	整数, 指示屏幕的最大垂直分辨率。
static	serverString: String [ 只读 ]	URL 编码的字符串, 用于指定每个 System.capabilities 属性的值。
static	version: String [ 只读 ]	字符串, 包含 Flash Player 平台和版本信息 (例如 "WIN 8,0,0,0")。

继承自 Object 类的属性

---

`constructor` (Object.`constructor` 属性), `__proto__` (Object.`__proto__` 属性), `prototype` (Object.`prototype` 属性), `__resolve` (Object.`__resolve` 属性)

---

方法摘要

继承自 Object 类的方法

---

`addProperty` (Object.`addProperty` 方法), `hasOwnProperty` (Object.`hasOwnProperty` 方法), `isPrototypeOf` (Object.`isPrototypeOf` 方法), `isPrototypeOf` (Object.`isPrototypeOf` 方法), `registerClass` (Object.`registerClass` 方法), `toString` (Object.`toString` 方法), `unwatch` (Object.`unwatch` 方法), `valueOf` (Object.`valueOf` 方法), `watch` (Object.`watch` 方法)

---

## avHardwareDisable (capabilities.avHardwareDisable 属性)

`public static avHardwareDisable : Boolean [read-only]`

布尔值，指定对用户的摄像头和麦克风的访问是已经通过管理方式禁止 (`true`) 还是允许 (`false`)。服务器字符串为 AVD。

可用性: **ActionScript 1.0** ; **Flash Player 7**

示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.avHardwareDisable);
```

另请参见

`get` (Camera.`get` 方法), `get` (Microphone.`get` 方法), `showSettings` (System.`showSettings` 方法)

## hasAccessibility (capabilities.hasAccessibility 属性)

public static hasAccessibility : Boolean [read-only]

布尔值，如果播放器正在支持 **Flash Player** 与辅助功能之间进行通讯的环境中运行，则为 true；否则为 false。服务器字符串为 ACC。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasAccessibility);
```

另请参见

[isActive](#) ([Accessibility.isActive](#) 方法), [updateProperties](#) ([Accessibility.updateProperties](#) 方法),

## hasAudio (capabilities.hasAudio 属性)

public static hasAudio : Boolean [read-only]

指定系统是否有音频功能。布尔值，如果播放器正在具有音频功能的系统上运行，则为 true；否则为 false。服务器字符串为 A。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasAudio);
```

## hasAudioEncoder (capabilities.hasAudioEncoder 属性)

public static hasAudioEncoder : Boolean [read-only]

指定 **Flash Player** 是否可对音频流进行编码。布尔值，如果播放器可以对音频流（例如来自麦克风的音频流）进行编码，则为 true；否则为 false。服务器字符串为 AE。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasAudioEncoder);
```

## hasEmbeddedVideo (capabilities.hasEmbeddedVideo 属性)

`public static hasEmbeddedVideo : Boolean [read-only]`

布尔值，如果播放器正在支持嵌入视频的系统上运行，则为 `true`；否则为 `false`。服务器字符串为 `EV`。

可用性：ActionScript 1.0；Flash Player 6,0,65,0

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasEmbeddedVideo);
```

## hasIME (capabilities.hasIME 属性)

`public static hasIME : Boolean [read-only]`

指示系统是否安装了输入法编辑器 (IME)。值为 `true` 表示播放器正在安装了 IME 的系统上运行；值为 `false` 表示未安装 IME。服务器字符串为 `IME`。

可用性：ActionScript 1.0；Flash Player 8

### 示例

如果播放器正在安装了 IME 的系统上运行，下面的示例将 IME 设置为 `ALPHANUMERIC_FULL`。

```
if(System.capabilities.hasIME) {  
    trace(System.IME.getConversionMode());  
    System.IME.setConversionMode(System.IME.ALPHANUMERIC_FULL);  
    trace(System.IME.getConversionMode());  
}
```

## hasMP3 (capabilities.hasMP3 属性)

`public static hasMP3 : Boolean [read-only]`

指定系统是否有 MP3 解码器。布尔值，如果播放器正在具有 MP3 解码器的系统上运行，则为 `true`；否则为 `false`。服务器字符串为 `MP3`。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasMP3);
```



## hasPrinting (capabilities.hasPrinting 属性)

`public static hasPrinting : Boolean [read-only]`

布尔值，如果播放器正在支持打印的系统上运行，则为 `true`；否则为 `false`。服务器字符串为 `PR`。

可用性：ActionScript 1.0；Flash Player 6,0,65,0

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasPrinting);
```

## hasScreenBroadcast (capabilities.hasScreenBroadcast 属性)

`public static hasScreenBroadcast : Boolean [read-only]`

布尔值，如果播放器支持开发通过 **Flash Communication Server** 运行的屏幕广播应用程序，则为 `true`；否则为 `false`。服务器字符串为 `SB`。

可用性：ActionScript 1.0；Flash Player 6,0,79,0

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasScreenBroadcast);
```

## hasScreenPlayback (capabilities.hasScreenPlayback 属性)

`public static hasScreenPlayback : Boolean [read-only]`

布尔值，如果播放器支持正在播放通过 **Flash Communication Server** 运行的屏幕广播应用程序，则为 `true`；否则为 `false`。服务器字符串为 `SP`。

可用性：ActionScript 1.0；Flash Player 6,0,79,0

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasScreenPlayback);
```

## hasStreamingAudio (capabilities.hasStreamingAudio 属性)

`public static hasStreamingAudio : Boolean [read-only]`

布尔值，如果播放器可以播放音频流，则为 `true`；否则为 `false`。服务器字符串为 SA。

可用性：ActionScript 1.0；Flash Player 6,0,65,0

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasStreamingAudio);
```

## hasStreamingVideo (capabilities.hasStreamingVideo 属性)

`public static hasStreamingVideo : Boolean [read-only]`

布尔值，如果播放器可以播放视频流，则为 `true`；否则为 `false`。服务器字符串为 SV。

可用性：ActionScript 1.0；Flash Player 6,0,65,0

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasStreamingVideo);
```

## hasVideoEncoder (capabilities.hasVideoEncoder 属性)

`public static hasVideoEncoder : Boolean [read-only]`

指定 **Flash Player** 是否可对视频流进行编码。布尔值，如果播放器可对视频流（如来自 Web 摄像头的视频流）进行编码，则为 `true`；否则为 `false`。服务器字符串为 VE。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.hasVideoEncoder);
```

## isDebugger（capabilities.isDebugger 属性）

public static isDebugger : Boolean [read-only]

布尔值，指示播放器是正式发布的版本 (false)，还是特殊的调试版本 (true)。服务器字符串为 DEB。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.isDebugger);
```

## language（capabilities.language 属性）

public static language : String [read-only]

指示运行播放器的系统的语言。此属性用 ISO 639-1 中的小写的两字符语言代码指定。对于中文，另外使用 ISO 3166 中的大写的两字符国家 / 地区代码子标记进行指定，以区分简体中文和繁体中文。这些语言本身是用英语标记指定的。例如，fr 指定法语。

在 Flash Player 7 中，对这一属性进行了两种更改。首先，英语系统的语言代码不再包括国家 / 地区代码。在 Flash Player 6 中，所有英语系统都返回语言代码和两字符的国家 / 地区代码子标记 (en-US)。在 Flash Player 7 中，英语系统仅返回语言代码 (en)。其次，在 Microsoft Windows 系统上，此属性现在返回用户界面 (UI) 语言。在 Microsoft Windows 平台上的 Flash Player 6 中，System.capabilities.language 返回用户区域设置，此区域设置可以控制日期、时间、货币和大写数字格式的设置。在 Microsoft Windows 平台上的 Flash Player 7 中，此属性现在返回用户界面语言，该语言指的是所有菜单、对话框、错误信息和帮助文件所使用的语言。下表列出了可能的值：

语言	标记
捷克语	cs
丹麦语	da
荷兰语	nl
英语	en
芬兰语	fi
法语	fr
德语	de
匈牙利语	hu
意大利语	it

语言	标记
日语	ja
朝鲜语	ko
挪威语	no
其它 / 未知	xu
波兰语	pl
葡萄牙语	pt
俄语	ru
简体中文	zh-CN
西班牙语	es
瑞典语	sv
繁体中文	zh-TW
土耳其语	tr

可用性：ActionScript 1.0 ； Flash Player 6

## 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.language);
```

## localFileReadDisable (capabilities.localFileReadDisable 属性)

```
public static localFileReadDisable : Boolean [read-only]
```

布尔值，指示对用户硬盘的读取权限是已经通过管理方式禁止 (true) 还是允许 (false)。

如果设置为 true，**Flash Player** 将无法从用户的硬盘读取文件（包括用以启动 **Flash Player** 的第一个 SWF 文件）。例如，如果此属性设置为 true，则试图使用 XML.load()、LoadMovie() 或 LoadVars.load() 来读取用户硬盘上的文件将会失败。

如果此属性设置为 true，则读取运行时共享库也会被阻止，但不管该属性值为何值，都允许读取本地共享的对象。服务器字符串为 LFD。

可用性：ActionScript 1.0 ； Flash Player 7

## 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.localFileReadDisable);
```

## manufacturer (capabilities.manufacturer 属性)

`public static manufacturer : String [read-only]`

字符串, 指示 **Flash Player** 制造商, 其格式为 "Macromedia OSName" (其中 OSName 可以为 "Windows"、"Macintosh"、"Linux" 或 "Other OS Name")。服务器字符串为 M。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例跟踪此只读属性的值:

```
trace(System.capabilities.manufacturer);
```

## os (capabilities.os 属性)

`public static os : String [read-only]`

字符串, 指示当前的操作系统。os 属性可以返回下列字符串: "Windows XP"、"Windows 2000"、"Windows NT"、"Windows 98/ME"、"Windows 95"、"Windows CE" (仅在 **Flash Player SDK** 中可用, 在桌面版本中不可用)、"Linux" 和 "MacOS"。服务器字符串为 OS。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例跟踪此只读属性的值:

```
trace(System.capabilities.os);
```

## pixelAspectRatio (capabilities.pixelAspectRatio 属性)

`public static pixelAspectRatio : Number [read-only]`

整数, 指示屏幕的像素高宽比。服务器字符串为 AR。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例跟踪此只读属性的值:

```
trace(System.capabilities.pixelAspectRatio);
```

## playerType (capabilities.playerType 属性)

`public static playerType : String [read-only]`

字符串，指示播放器的类型。此属性可以是下列值之一：

- "StandAlone"，用于 **Flash** 独立播放器
- "External"，用于外部播放器使用的 **Flash Player** 版本，或用于测试影片模式
- "PlugIn"，用于 **Flash Player** 浏览器插件
- "ActiveX"，用于 **Microsoft Internet Explorer** 使用的 **Flash Player ActiveX** 控件

服务器字符串为 PT。

可用性：ActionScript 1.0 ； Flash Player 7

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.playerType);
```

## screenColor (capabilities.screenColor 属性)

`public static screenColor : String [read-only]`

字符串，指示屏幕的颜色。此属性的值可以为 “color”、“gray” 或 “bw”，分别表示彩色、灰度和黑白。服务器字符串为 COL。

可用性：ActionScript 1.0 ； Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.screenColor);
```

## screenDPI (capabilities.screenDPI 属性)

`public static screenDPI : Number [read-only]`

数字，指示屏幕的每英寸点数 (dpi) 分辨率，以像素为单位。服务器字符串为 DP。

可用性：ActionScript 1.0 ； Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.screenDPI);
```

## screenResolutionX (capabilities.screenResolutionX 属性)

`public static screenResolutionX : Number [read-only]`

整数，指示屏幕的最大水平分辨率。服务器字符串为 R（它返回屏幕的宽度和高度）。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例跟踪此只读属性的值:

```
trace(System.capabilities.screenResolutionX);
```

## screenResolutionY (capabilities.screenResolutionY 属性)

`public static screenResolutionY : Number [read-only]`

整数，指示屏幕的最大垂直分辨率。服务器字符串为 R（它返回屏幕的宽度和高度）。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例跟踪此只读属性的值:

```
trace(System.capabilities.screenResolutionY);
```

## serverString (capabilities.serverString 属性)

`public static serverString : String [read-only]`

URL 编码的字符串，用于指定每个 `System.capabilities` 属性的值。

以下示例显示了一个 URL 编码的字符串:

```
A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&
SB=f&DEB=t&V=WIN%208%2C0%2C0%2C0&M=Macromedia%20Windows&
R=1600x1200&DP=72&COL=color&AR=1.0&OS=Windows%20XP&
L=en&PT=External&AVD=f&LFD=f&WD=f
```

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例跟踪此只读属性的值:

```
trace(System.capabilities.serverString);
```

## version（capabilities.version 属性）

public static version : String [read-only]

字符串，包含 **Flash Player** 平台和版本信息（例如 "WIN 8,0,0,0"）。服务器字符串为 v。

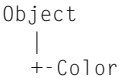
可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例跟踪此只读属性的值：

```
trace(System.capabilities.version);
```

## Color



```
public class Color
extends Object
```

自 **Flash Player 8** 后不推荐使用。不推荐使用 **Color** 类，而推荐使用 **flash.geom.ColorTransform** 类。

通过 **Color** 类，您可以设置影片剪辑的 **RGB** 颜色值和颜色转换，并可以在设置后检索这些值。

必须在使用构造函数 `new Color()` 创建 **Color** 对象后才能调用其方法。

可用性：ActionScript 1.0；Flash Player 5

### 属性摘要

继承自 **Object** 类的属性

---

`constructor` ([Object.constructor 属性](#))，`__proto__` ([Object.\\_\\_proto\\_\\_ 属性](#))，`prototype` ([Object.prototype 属性](#))，`__resolve` ([Object.\\_\\_resolve 属性](#))

---

### 构造函数摘要

签名	说明
<code>Color(target:Object)</code>	否决的类。 为 <code>target_mc</code> 参数指定的影片剪辑创建一个 <b>Color</b> 对象。



方法摘要

修饰符	签名	说明
	<code>getRGB() : Number</code>	否决的类。 返回目前正由 Color 对象使用的 R+G+B 组合。
	<code>getTransform() : Object</code>	否决的类。 返回由最后一次 <code>Color.setTransform()</code> 调用设置的转换值。
	<code>setRGB(offset:Number) : Void</code>	否决的类。 指定 Color 对象的 RGB 颜色。
	<code>setTransform(transformObject:Object) : Void</code>	否决的类。 为 Color 对象设置颜色转换信息。

继承自 Object 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

Color 构造函数

`public Color(target:Object)`

自 Flash Player 8 后不推荐使用 **Color** 类。

为 `target_mc` 参数指定的影片剪辑创建一个 **Color** 对象。然后可使用该 **Color** 对象的方法来更改整个目标影片剪辑的颜色。

可用性: **ActionScript 1.0** ; **Flash Player 5**

参数

**target:Object** - 影片剪辑的实例名称。

示例

下面的示例为影片剪辑 `my_mc` 创建一个名为 `my_color` 的 **Color** 对象, 并将其 **RGB** 值设置为橙色:

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xff9933);
```

## getRGB (Color.getRGB 方法)

public getRGB() : Number

自 **Flash Player 8** 后不推荐使用 **Color** 类。

返回目前正由 **Color** 对象使用的 **R+G+B** 组合。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 数字，表示指定的颜色的 **RGB** 数值。

### 示例

下面的代码检索 **Color** 对象 `my_color` 的 **RGB** 值，将该值转换为十六进制字符串，并将其分配给 `myValue` 变量。要查看此代码的运行情况，请在舞台上添加一个影片剪辑实例，并为它指定实例名称 `my_mc`：

```
var my_color:Color = new Color(my_mc);  
// set the color  
my_color.setRGB(0xff9933);  
var myValue:String = my_color.getRGB().toString(16);  
// trace the color value  
trace(myValue); // traces ff9933
```

### 另请参见

[setRGB \(Color.setRGB 方法\)](#)

## getTransform (Color.getTransform 方法)

public getTransform() : Object

自 **Flash Player 8** 后不推荐使用 **Color** 类。

返回由最后一次 `Color.setTransform()` 调用设置的转换值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Object - 对象，其属性包含指定的颜色的当前偏移量和百分比值。

## 示例

下面的示例获取转换对象，然后，相对于 my\_mc 的颜色和 Alpha 的百分比的当前值来设置新值。若要查看此代码的工作情况，请将实例名称为 my\_mc 的彩色影片剪辑置于舞台上。然后将下面的代码放在主时间轴的第 1 帧中，并选择“控制” > “测试影片”：

```
var my_color:Color = new Color(my_mc);
var myTransform:Object = my_color.getTransform();
myTransform = { ra: 50, ba: 50, aa: 30};
my_color.setTransform(myTransform);
```

有关颜色转换对象的参数说明，请参见 `Color.setTransform()`。

## 另请参见

[setTransform \(Color.setTransform 方法\)](#)

## setRGB (Color.setRGB 方法)

```
public setRGB(offset:Number) : Void
```

自 Flash Player 8 后不推荐使用 `Color` 类。

指定 `Color` 对象的 RGB 颜色。调用此方法将覆盖以前的任何 `Color.setTransform()` 设置。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**offset:Number** - 0x *RRGGBB*，要设置的十六进制颜色或 RGB 颜色。*RR*、*GG* 和 *BB* 均包含两个十六进制数字，这些数字指定每个颜色成分的偏移量。`ActionScript` 编译器从 0x 获知数字是十六进制值。

## 示例

此示例设置影片剪辑 my\_mc 的 RGB 颜色值。若要查看此代码的工作情况，请将实例名称为 my\_mc 的影片剪辑置于舞台上。然后将下面的代码放在主时间轴的第 1 帧中，并选择“控制” > “测试影片”：

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xFF0000); // my_mc turns red
```

## 另请参见

[setTransform \(Color.setTransform 方法\)](#)

## setTransform (Color.setTransform 方法)

```
public setTransform(transformObject:Object) : Void
```

自 Flash Player 8 后不推荐使用 **Color** 类。

为 **Color** 对象设置颜色转换信息。从 new Object 构造函数创建的 *colorTransformObject* 参数是通用对象。它具有指定颜色的红、绿、蓝和 **alpha**（透明度）成分百分比和偏移量值的参数，以 0xRRGGBBAA 的格式输入。

颜色转换对象的参数与“高级效果”对话框中的设置相对应，定义如下：

- *ra* 是红色成分的百分比（-100 到 100）。
- *rb* 是红色成分的偏移量（-255 到 255）。
- *ga* 是绿色成分的百分比（-100 到 100）。
- *gb* 是绿色成分的偏移量（-255 到 255）。
- *ba* 是蓝色成分的百分比（-100 到 100）。
- *bb* 是蓝色成分的偏移量（-255 到 255）。
- *aa* 是 Alpha 的百分比（-100 到 100）。
- *ab* 是 Alpha 的偏移量（-255 到 255）。

您可以创建 *colorTransformObject* 参数，方法如下所示：

```
var myColorTransform:Object = new Object();  
myColorTransform.ra = 50;  
myColorTransform.rb = 244;  
myColorTransform.ga = 40;  
myColorTransform.gb = 112;  
myColorTransform.ba = 12;  
myColorTransform.bb = 90;  
myColorTransform.aa = 40;  
myColorTransform.ab = 70;
```

您也可使用以下语法来创建 *colorTransformObject* 参数：

```
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb: 90,  
    aa: 40, ab: 70 }
```

可用性：ActionScript 1.0；Flash Player 5

### 参数

**transformObject:Object** - 使用 new Object 构造函数创建的对象。**Object** 类的这一实例必须具有以下指定颜色转换值的属性：*ra*、*rb*、*ga*、*gb*、*ba*、*bb*、*aa*、*ab*。下面对这些属性进行了解释。

## 示例

此示例为目标 SWF 文件创建新 **Color** 对象，使用上面定义的属性创建名为 `myColorTransform` 的通用对象，然后使用 `setTransform()` 方法将 `colorTransformObject` 传递给 **Color** 对象。若要在 **Flash (FLA)** 文档中使用此代码，请将其放在主时间轴上的第 1 帧上，然后将实例名称为 `my_mc` 的影片剪辑置于舞台上，如下代码所示：

```
// Create a color object called my_color for the target my_mc
var my_color:Color = new Color(my_mc);
// Create a color transform object called myColorTransform using
// Set the values for myColorTransform
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb:
    90, aa: 40, ab: 70};
// Associate the color transform object with the Color object
// created for my_mc
my_color.setTransform(myColorTransform);
```

另请参见

[Object](#)

# ColorMatrixFilter (flash.filters.ColorMatrixFilter)

```
Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.ColorMatrixFilter
```

```
public class ColorMatrixFilter
extends BitmapFilter
```

**ColorMatrixFilter** 类使您可以将  $4 \times 5$  矩阵转换应用于输入图像上的每个像素的 **RGBA** 颜色和 **Alpha** 值，以产生具有一组新的 **RGBA** 颜色和 **Alpha** 值的结果。该类允许饱和度更改、色相旋转、亮度为 **Alpha** 以及各种其它效果。可以将此滤镜应用于位图和 **MovieClip** 实例。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时对影片剪辑应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤消设置操作。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter()` 会获得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作时对图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

将使用下列公式，其中 `a[0]` 到 `a[19]` 对应于 20 个元素的数组属性矩阵中的条目 0 到 19。

```
redResult = a[0] * srcR + a[1] * srcG + a[2] * srcB + a[3] * srcA + a[4]
greenResult = a[5] * srcR + a[6] * srcG + a[7] * srcB + a[8] * srcA + a[9]
blueResult = a[10] * srcR + a[11] * srcG + a[12] * srcB + a[13] * srcA +
    a[14]
alphaResult = a[15] * srcR + a[16] * srcG + a[17] * srcB + a[18] * srcA +
    a[19]
```

此滤镜将每个源像素分离成它的红色、绿色、蓝色和 Alpha 成分，分别以 `srcR`、`srcG`、`srcB` 和 `srcA` 表示。最后一步，将各颜色成分重新组合为一个像素，并写出结果。

计算是对非相乘的颜色值执行的。如果输入图形由预先相乘的颜色值组成，这些值会自动转换为非相乘的颜色值以执行此操作。

可以使用下面两种经过优化的模式。

**仅 Alpha。** 当向滤镜传递仅调整 Alpha 成分的矩阵时，滤镜将优化其性能，如下所示：

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 N 0 (where N is between 0.0 and 1.0)
```

**更快的版本。** 仅可用于启用 SSE/AltiVec 加速器的处理器，如 Pentium 3 及更高版本、Apple G4 及更高版本。当乘数项在 -15.99 到 15.99 之间，并且加数项 `a[4]`、`a[9]`、`a[14]` 和 `a[19]` 在 -8000 到 8000 之间时，将使用加速器。

如果结果图像的宽度或高度将超过 2880 像素，则不应用滤镜。例如，如果您在放大某大型影片剪辑时使用了滤镜，则在结果图像超过 2880 像素的限制时，该滤镜将关闭。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例使用 `BitmapFilter` 来根据鼠标指针的位置操作图像的颜色饱和度。如果将鼠标指针置于左上角 (0,0)，图像应保持不变。随着鼠标指针的右移，绿色通道和蓝色通道一起从图像中删除。随着鼠标指针的下移，红色通道被删除。如果将鼠标指针置于舞台的右下方，图像应完全变黑。此示例假设您的库中具有链接标识符设置为“`YourImageLinkage`”的图像。

```
import flash.filters.BitmapFilter;
import flash.filters.ColorMatrixFilter;

var image:MovieClip = this.attachMovie("YourImageLinkage", "YourImage",
    this.getNextHighestDepth());
image.cacheAsBitmap = true;
```

```
var listener:Object = new Object();
listener.image = image;
listener.onMouseMove = function() {
    var xPercent:Number = 1 - (_xmouse/Stage.width);
    var yPercent:Number = 1 - (_ymouse/Stage.height);
    var matrix:Array = new Array();
    matrix = matrix.concat([yPercent, 0, 0, 0, 0]); // red
    matrix = matrix.concat([0, xPercent, 0, 0, 0]); // green
    matrix = matrix.concat([0, 0, xPercent, 0, 0]); // blue
    matrix = matrix.concat([0, 0, 0, 1, 0]); // alpha

    var filter:BitmapFilter = new ColorMatrixFilter(matrix);
    image.filters = new Array(filter);
}

Mouse.addListener(listener);
listener.onMouseMove();
```

另请参见 [getPixel \(BitmapData.getPixel 方法\)](#), [applyFilter \(BitmapData.applyFilter 方法\)](#), [filters \(MovieClip.filters 属性\)](#), [cacheAsBitmap \(MovieClip.cacheAsBitmap 属性\)](#)

属性摘要

修饰符	属性	说明
	matrix:Array	由 20 个元素组成的数组，适用于 4 x 5 颜色转换。

继承自 [Object](#) 类的属性

<a href="#">constructor</a> ( <a href="#">Object.constructor 属性</a> ), <a href="#">__proto__</a> ( <a href="#">Object.__proto__ 属性</a> ), <a href="#">prototype</a> ( <a href="#">Object.prototype 属性</a> ), <a href="#">__resolve</a> ( <a href="#">Object.__resolve 属性</a> )
--

构造函数摘要

签名	说明
<a href="#">ColorMatrixFilter</a> (matrix:Array)	用指定参数初始化新的 <a href="#">ColorMatrixFilter</a> 实例。

方法摘要

修饰符	签名	说明
	<code>clone() : ColorMatrixFilter</code>	返回此滤镜对象的副本。

继承自 `BitmapFilter` 类的方法

[clone](#) ([BitmapFilter.clone](#) 方法)

继承自 `Object` 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

# clone (ColorMatrixFilter.clone 方法)

```
public clone() : ColorMatrixFilter
```

返回此滤镜对象的副本。

可用性: **ActionScript 1.0 ; Flash Player 8**

## 返回

`flash.filters.ColorMatrixFilter` - 与原始实例具有完全相同的属性的新的 `ColorMatrixFilter` 实例。

## 示例

下面的示例创建一个新的 `ColorMatrixFilter` 实例，然后使用 `clone` 方法克隆它。不能直接更改 `matrix` 属性（例如 `clonedFilter.matrix[2] = 1;`）。相反，必须先获取对数组的引用，对该引用进行更改，然后使用 `clonedFilter.matrix = changedMatrix` 重置该值。

```
import flash.filters.ColorMatrixFilter;

var matrix:Array = new Array();
matrix = matrix.concat([1, 0, 0, 0, 0]); // red
matrix = matrix.concat([0, 1, 0, 0, 0]); // green
matrix = matrix.concat([0, 0, 1, 0, 0]); // blue
matrix = matrix.concat([0, 0, 0, 1, 0]); // alpha
```



```

var filter:ColorMatrixFilter = new ColorMatrixFilter(matrix);
trace("filter: " + filter.matrix);

var clonedFilter:ColorMatrixFilter = filter.clone();
matrix = clonedFilter.matrix;
matrix[2] = 1;
clonedFilter.matrix = matrix;
trace("clonedFilter: " + clonedFilter.matrix);

```

## ColorMatrixFilter 构造函数

```
public ColorMatrixFilter(matrix:Array)
```

用指定参数初始化新的 **ColorMatrixFilter** 实例。

可用性：ActionScript 1.0 ； Flash Player 8

### 参数

**matrix:Array** - 由 20 个元素（排列成 4 x 5 矩阵）组成的数组。

## matrix（ColorMatrixFilter.matrix 属性）

```
public matrix : Array
```

由 20 个元素组成的数组，适用于 4 x 5 颜色转换。

可用性：ActionScript 1.0 ； Flash Player 8

### 示例

下面的示例创建一个新的 **ColorMatrixFilter** 实例，然后更改其 **matrix** 属性。不能通过直接修改 **matrix** 属性的值来更改它（例如 `clonedFilter.matrix[2] = 1;`）。相反，必须先获取对数组的引用，对引用进行更改，然后使用 `clonedFilter.matrix = changedMatrix` 重置该值。

```

import flash.filters.ColorMatrixFilter;

var matrix:Array = new Array();
matrix = matrix.concat([1, 0, 0, 0, 0]); // red
matrix = matrix.concat([0, 1, 0, 0, 0]); // green
matrix = matrix.concat([0, 0, 1, 0, 0]); // blue
matrix = matrix.concat([0, 0, 0, 1, 0]); // alpha

var filter:ColorMatrixFilter = new ColorMatrixFilter(matrix);
trace("filter: " + filter.matrix);
var changedMatrix:Array = filter.matrix;
changedMatrix[2] = 1;
filter.matrix = changedMatrix;
trace("filter: " + filter.matrix);

```

# ColorTransform

## (flash.geom.ColorTransform)

```
Object
|
+- flash.geom.ColorTransform
```

```
public class ColorTransform
extends Object
```

**ColorTransform** 类使您可以精确地调整影片剪辑中的所有颜色值。颜色调整函数或颜色转换 可以应用于所有四个通道：红色、绿色、蓝色和 **Alpha** 透明度。

当 **ColorTransform** 对象应用于影片剪辑时，将按与如下类似的方法为每个颜色通道计算新值：

- 新红色值 = (旧红色值 \* redMultiplier) + redOffset
- 新绿色值 = (旧绿色值 \* greenMultiplier) + greenOffset
- 新蓝色值 = (旧蓝色值 \* blueMultiplier) + blueOffset
- 新 Alpha 值 = (旧 Alpha 值 \* alphaMultiplier) + alphaOffset

如果计算后任何颜色通道值都大于 255，该值将被设置为 255。如果该值小于 0，它将被设置为 0。

必须使用 `new ColorTransform()` 构造函数创建 **ColorTransform** 对象后，才能调用 **ColorTransform** 对象的方法。

颜色转换不会应用于影片剪辑（如加载的 SWF 对象）的背景色，它们仅应用于附加到影片剪辑的图形和元件。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[getTransform\(Color.getTransform 方法\)](#)，[setTransform\(Color.setTransform 方法\)](#)，[Transform \(flash.geom.Transform\)](#)

属性摘要

修饰符	属性	说明
	alphaMultiplier:Number	与 Alpha 透明度通道值相乘的十进制值。
	alphaOffset:Number	-255 到 255 之间的数字，它先与 alphaMultiplier 值相乘，再与 Alpha 透明度通道值相加。
	blueMultiplier:Number	与蓝色通道值相乘的十进制值。
	blueOffset:Number	-255 到 255 之间的数字，它先与 blueMultiplier 值相乘，再与蓝色通道值相加。
	greenMultiplier:Number	与绿色通道值相乘的十进制值。
	greenOffset:Number	-255 到 255 之间的数字，它先与 greenMultiplier 值相乘，再与绿色通道值相加。
	redMultiplier:Number	与红色通道值相乘的十进制值。
	redOffset:Number	-255 到 255 之间的数字，它先与 redMultiplier 值相乘，再与红色通道值相加。
	rgb:Number	ColorTransform 对象的 RGB 颜色值。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

构造函数摘要

签名	说明
ColorTransform([redMultiplier:Number], [greenMultiplier:Number], [blueMultiplier:Number], [alphaMultiplier:Number], [redOffset:Number], [greenOffset:Number], [blueOffset:Number], [alphaOffset:Number])	用指定的颜色通道值和 Alpha 值为显示对象创建 ColorTransform 对象。

方法摘要

修饰符	签名	说明
	<code>concat(second:Color Transform) : Void</code>	向影片剪辑应用第二个加色转换。
	<code>toString() : String</code>	设置字符串格式并将其返回，该字符串描述 ColorTransform 对象的所有属性。

继承自 Object 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPropertyEnumerable](#) ([Object.isPropertyEnumerable](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

## alphaMultiplier (ColorTransform.alphaMultiplier 属性)

`public alphaMultiplier : Number`

与 Alpha 透明度通道值相乘的十进制值。

如果您使用 `MovieClip._alpha` 属性直接设置影片剪辑的 Alpha 透明度值，它将影响该影片剪辑的 `ColorTransform` 对象的 `alphaMultiplier` 属性的值。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例创建 `ColorTransform` 对象 `colorTrans`，然后将其 `alphaMultiplier` 值从 1 调整到 0.5。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.alphaMultiplier); // 1

colorTrans.alphaMultiplier = .5;
trace(colorTrans.alphaMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
```

```

var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
mc.beginFill(color);
mc.lineTo(0, height);
mc.lineTo(width, height);
mc.lineTo(width, 0);
mc.lineTo(0, 0);
return mc;
}

```

另请参见

[\\_alpha \(MovieClip.\\_alpha 属性\)](#)

## alphaOffset (ColorTransform.alphaOffset 属性)

public alphaOffset : Number

-255 到 255 之间的数字, 它先与 alphaMultiplier 值相乘, 再与 Alpha 透明度通道值相加。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans, 然后将其 alphaOffset 值从 0 调整到 -128。

```

import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.alphaOffset); // 0

colorTrans.alphaOffset = -128;
trace(colorTrans.alphaOffset); // -128

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}

```

## blueMultiplier (ColorTransform.blueMultiplier 属性)

public blueMultiplier : Number

与蓝色通道值相乘的十进制值。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans，然后将其 blueMultiplier 值从 **1** 调整到 **0.5**。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.blueMultiplier); // 1

colorTrans.blueMultiplier = .5;
trace(colorTrans.blueMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0x0000FF);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## blueOffset (ColorTransform.blueOffset 属性)

public blueOffset : Number

-255 到 255 之间的数字，它先与 blueMultiplier 值相乘，再与蓝色通道值相加。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans，然后将其 blueOffset 值从 0 调整到 255。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.blueOffset); // 0

colorTrans.blueOffset = 255;
trace(colorTrans.blueOffset); // 255

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## ColorTransform 构造函数

```
public ColorTransform([redMultiplier:Number], [greenMultiplier:Number],  
    [blueMultiplier:Number], [alphaMultiplier:Number], [redOffset:Number],  
    [greenOffset:Number], [blueOffset:Number], [alphaOffset:Number])
```

用指定的颜色通道值和 Alpha 值为显示对象创建 **ColorTransform** 对象。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**redMultiplier**:Number [ 可选 ] - 红色乘数的值, 在 0 到 1 之间。默认值为 1。

**greenMultiplier**:Number [ 可选 ] - 绿色乘数的值, 在 0 到 1 之间。默认值为 1。

**blueMultiplier**:Number [ 可选 ] - 蓝色乘数的值, 在 0 到 1 之间。默认值为 1。

**alphaMultiplier**:Number [ 可选 ] - **Alpha** 透明度乘数的值, 在 0 到 1 之间。默认值为 1。

**redOffset**:Number [ 可选 ] - 红色通道值的偏移量 (-255 到 255)。默认值是 0。

**greenOffset**:Number [ 可选 ] - 绿色通道值的偏移量 (-255 到 255)。默认值是 0。

**blueOffset**:Number [ 可选 ] - 蓝色通道值的偏移量 (-255 到 255)。默认值是 0。

**alphaOffset**:Number [ 可选 ] - **Alpha** 透明度通道值的偏移量 (-255 到 255)。默认值是 0。

### 示例

下面的示例创建名为 **greenTransform** 的 **ColorTransform** 对象:

```
var greenTransform:flash.geom.ColorTransform = new  
    flash.geom.ColorTransform(0.5, 1.0, 0.5, 0.5, 10, 10, 10, 0);
```

下面的示例创建具有默认构造函数值的 **ColorTransform** 对象 **colorTrans\_1**。

**colorTrans\_1** 和 **colorTrans\_2** 跟踪同一个值的事实证明使用了默认构造函数值。

```
import flash.geom.ColorTransform;
```

```
var colorTrans_1:ColorTransform = new ColorTransform(1, 1, 1, 1, 0, 0, 0, 0);  
trace(colorTrans_1);  
//(redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,  
    redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
```

```
var colorTrans_2:ColorTransform = new ColorTransform();  
trace(colorTrans_2);  
//(redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,  
    redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
```



## concat (ColorTransform.concat 方法)

```
public concat(second:ColorTransform) : Void
```

向影片剪辑应用第二个加色转换。在完成第一个转换后，对影片剪辑的颜色应用第二组转换参数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**second:** flash.geom.ColorTransform - 要与当前 **ColorTransform** 对象合并的第二个 **ColorTransform** 对象。

### 示例

下面的示例将 **ColorTransform** 对象 colorTrans\_2 与 colorTrans\_1 连接在一起，使整个红色偏移量与 **0.5 Alpha** 乘数进行合并。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans_1:ColorTransform = new ColorTransform(1, 1, 1, 1, 255, 0, 0,
0);
trace(colorTrans_1);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,
redOffset=255, greenOffset=0, blueOffset=0, alphaOffset=0)

var colorTrans_2:ColorTransform = new ColorTransform(1, 1, 1, .5, 0, 0, 0,
0);
trace(colorTrans_2);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1,
alphaMultiplier=0.5, redOffset=0, greenOffset=0, blueOffset=0,
alphaOffset=0)

colorTrans_1.concat(colorTrans_2);
trace(colorTrans_1);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1,
alphaMultiplier=0.5, redOffset=255, greenOffset=0, blueOffset=0,
alphaOffset=0)

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans_1;

function createRectangle(width:Number, height:Number, color:Number,
scope:MovieClip):MovieClip {
scope = (scope == undefined) ? this : scope;
var depth:Number = scope.getNextHighestDepth();
var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
mc.beginFill(color);
```

```

        mc.lineTo(0, height);
        mc.lineTo(width, height);
        mc.lineTo(width, 0);
        mc.lineTo(0, 0);
        return mc;
    }

```

## greenMultiplier (ColorTransform.greenMultiplier 属性)

public greenMultiplier : Number

与绿色通道值相乘的十进制值。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans，然后将其 greenMultiplier 从 1 调整到 0.5。

```

import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.greenMultiplier); // 1

colorTrans.greenMultiplier = .5;
trace(colorTrans.greenMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0x00FF00);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}

```

## greenOffset (ColorTransform.greenOffset 属性)

public greenOffset : Number

-255 到 255 之间的数字，它先与 greenMultiplier 值相乘，再与绿色通道值相加。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans，然后将其 greenOffset 值从 0 调整到 255。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.greenOffset); // 0

colorTrans.greenOffset = 255;
trace(colorTrans.greenOffset); // 255

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## redMultiplier (ColorTransform.redMultiplier 属性)

public redMultiplier : Number

与红色通道值相乘的十进制值。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans, 然后将其 redMultiplier 值从 1 调整到 0.5。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.redMultiplier); // 1

colorTrans.redMultiplier = .5;
trace(colorTrans.redMultiplier); // .5

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## redOffset (ColorTransform.redOffset 属性)

public redOffset : Number

-255 到 255 之间的数字，它先与 redMultiplier 值相乘，再与红色通道值相加。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans，然后将其 redOffset 值从 0 调整到 255。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.redOffset); // 0

colorTrans.redOffset = 255;
trace(colorTrans.redOffset); // 255

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## rgb (ColorTransform.rgb 属性)

public rgb : Number

**ColorTransform** 对象的 RGB 颜色值。

当设置此属性时，它相应地更改三种颜色的偏移量值（redOffset、greenOffset 和 blueOffset），并将这三个颜色乘数值（redMultiplier、greenMultiplier 和 blueMultiplier）设置为 0。**Alpha** 透明度乘数和偏移量值不变。

以下面的格式传递此属性的值：**0xRRGGBB**。RR、GG 和 BB 均包含两个十六进制数字，这些数字指定每个颜色成分的偏移量。**ActionScript** 编译器从 **0x** 获知数字是十六进制值。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例创建 **ColorTransform** 对象 colorTrans，并将其 rgb 值调整到 **0xFF0000**。

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
trace(colorTrans.rgb); // 0

colorTrans.rgb = 0xFF0000;
trace(colorTrans.rgb); // 16711680
trace("0x" + colorTrans.rgb.toString(16)); // 0xff0000

var rect:MovieClip = createRectangle(20, 80, 0x000000);
var trans:Transform = new Transform(rect);
trans.colorTransform = colorTrans;

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

另请参见

[setRGB \(Color.setRGB 方法\)](#)，[getRGB \(Color.getRGB 方法\)](#)

## toString (ColorTransform.toString 方法)

```
public toString() : String
```

设置字符串格式并将其返回，该字符串描述 **ColorTransform** 对象的所有属性。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

**String** - 列出 **ColorTransform** 对象的所有属性的字符串。

### 示例

下面的示例创建 **ColorTransform** 对象 **colorTrans**，并调用其 **toString()** 方法。此方法生成以下格式的字符串: (**redMultiplier=RM**, **greenMultiplier=GM**, **blueMultiplier=BM**, **alphaMultiplier=AM**, **redOffset=RO**, **greenOffset=GO**, **blueOffset=BO**, **alphaOffset=AO**)。

```
import flash.geom.ColorTransform;

var colorTrans:ColorTransform = new ColorTransform(1, 2, 3, 4, -255, -128,
    128, 255);
trace(colorTrans.toString());
// (redMultiplier=1, greenMultiplier=2, blueMultiplier=3, alphaMultiplier=4,
    redOffset=-255, greenOffset=-128, blueOffset=128, alphaOffset=255)
```

## ContextMenu

```
Object
|
+- ContextMenu
```

```
public dynamic class ContextMenu
extends Object
```

**ContextMenu** 类提供对 **Flash Player** 上下文菜单项的运行时控制，当用户在 **Flash Player** 中右键单击（在 **Windows** 中）或按住 **Control** 键并单击（在 **Macintosh** 中）时，将出现上下文菜单。您可以使用 **ContextMenu** 类的方法和属性添加自定义菜单项，控制内置上下文菜单项的显示（例如“放大”和“打印”），或者创建菜单的副本。

您可以将 **ContextMenu** 对象附加到特定的按钮、影片剪辑或文本字段对象，也可以附加到整个影片级别。为此，您可以使用 **Button**、**MovieClip** 或 **TextField** 类的 **menu** 属性。有关 **menu** 属性的更多信息，请参见 **Button.menu**、**MovieClip.menu** 和 **TextField.menu**。

若要向 **ContextMenu** 对象添加新项目，您可以创建一个 **ContextMenuItem** 对象，然后将该对象添加到 **ContextMenu.customItems** 数组。有关创建上下文菜单项的更多信息，请参见 **ContextMenuItem** 类条目。

Flash Player 具有三种类型的上下文菜单：标准菜单（当您在 Flash Player 中右键单击时出现）、编辑菜单（当您在可选择或可编辑的文本字段上右键单击时出现）和错误菜单（当 SWF 文件未能加载到 Flash Player 中时出现）。只有标准菜单和编辑菜单才能使用 ContextMenu 类进行修改。

自定义菜单项始终出现在 Flash Player 上下文菜单的顶部，并位于所有可见内置菜单项之上；内置菜单项和自定义菜单项之间由一个分隔条加以分隔。向上下文菜单添加的自定义菜单项不得超过 15 个。您不能从上下文菜单中删除“设置”菜单项。在 Flash 中，必须有“设置”菜单项，有了这个菜单项后，用户才能访问影响其计算机上的隐私和存储的设置。您也不能从上下文菜单中删除“关于”菜单项，必须在使用这个菜单项后，用户才能了解正在使用的 Flash Player 的版本信息。

必须使用构造函数 new ContextMenu() 创建 ContextMenu 对象才能调用其方法。

可用性：ActionScript 1.0；Flash Player 7

另请参见

[ContextMenuitem](#), [menu \(Button.menu 属性\)](#), [menu \(MovieClip.menu 属性\)](#), [menu \(TextField.menu 属性\)](#)

属性摘要

修饰符	属性	说明
	<code>builtInItems:Object</code>	具有以下布尔属性的对象：zoom、quality、play、loop、rewind、forward_back 和 print。
	<code>customItems:Array</code>	ContextMenuitem 对象的数组。

继承自 Object 类的属性

[constructor \(Object.constructor 属性\)](#), [\\_\\_proto\\_\\_ \(Object.\\_\\_proto\\_\\_ 属性\)](#), [prototype \(Object.prototype 属性\)](#), [\\_\\_resolve \(Object.\\_\\_resolve 属性\)](#)

事件摘要

事件	说明
<code>onSelect = function(item:Object, item_menu:Object) {}</code>	在用户调用 Flash Player 上下文菜单并在该菜单实际显示之前调用。



构造函数摘要

签名	说明
ContextMenu([callbackFunction:Function])	创建新的 ContextMenu 对象。

方法摘要

修饰符	签名	说明
	copy() : ContextMenu	创建指定的 ContextMenu 对象的副本。
	hideBuiltInItems() : Void	隐藏指定的 ContextMenu 对象中的所有内置菜单项（“设置”除外）。

继承自 Object 类的方法

[addProperty](#) (Object.addProperty 方法), [hasOwnProperty](#) (Object.hasOwnProperty 方法), [isPropertyEnumerable](#) (Object.isPropertyEnumerable 方法), [isPrototypeOf](#) (Object.isPrototypeOf 方法), [registerClass](#) (Object.registerClass 方法), [toString](#) (Object.toString 方法), [unwatch](#) (Object.unwatch 方法), [valueOf](#) (Object.valueOf 方法), [watch](#) (Object.watch 方法)

builtInItems (ContextMenu.builtInItems 属性)

public builtInItems : Object


具有以下布尔属性的对象: zoom、quality、play、loop、rewind、forward\_back 和 print。将这些变量设置为 false 将删除指定的 ContextMenu 对象中的相应菜单项。这些属性是可枚举的属性，默认情况下设置为 true。

可用性: ActionScript 1.0 ; Flash Player 7

示例

在此示例中，对于附加到 SWF 文件的当前时间轴的 ContextMenu 对象 my\_cm，内置“品质”和“打印”菜单项被禁用。

```
var my_cm:ContextMenu = new ContextMenu ();
my_cm.builtInItems.quality=false;
my_cm.builtInItems.print=false;
this.menu = my_cm;
```



您不能从上下文菜单中禁用“设置”或“关于”菜单项。

在下一个示例中，for..in 循环枚举遍历 **ContextMenu** 对象 my\_cm 的内置菜单项的所有名称和值。

```
var my_cm:ContextMenu = new ContextMenu();
for(eachProp in my_cm.builtInItems) {
    var propName = eachProp;
    var propValue = my_cm.builtInItems[propName];
    trace(propName + ": " + propValue);
}
```

## ContextMenu 构造函数

```
public ContextMenu([callbackFunction:Function])
```

创建新的 **ContextMenu** 对象。或者，您可以在创建该对象时指定事件处理函数的标识符。指定的函数在用户调用上下文菜单并在该菜单实际显示之前被调用。在根据应用程序状态或者根据用户右键单击或按住 **Control** 键并单击的对象（影片剪辑、文本字段或按钮）的类型或时间轴自定义菜单内容时，该函数的这种行为非常有用。（有关创建事件处理函数的示例，请参见 `ContextMenu.onSelect`。）

可用性：ActionScript 1.0；Flash Player 7

### 参数

**callbackFunction:**Function [ 可选 ] - 对函数的引用，该函数在用户右键单击或按住 **Control** 键单击菜单并在菜单显示之前被调用。

### 示例

以下示例隐藏上下文菜单中的所有内置对象。（但是，由于不能禁用“设置”和“关于”菜单项，所以它们仍会出现。）

```
var newMenu:ContextMenu = new ContextMenu();
newMenu.hideBuiltInItems();
this.menu = newMenu;
```

在此示例中，指定的事件处理函数 menuHandler 根据名为 showItem 的布尔变量的值启用或禁用自定义菜单项（使用 `ContextMenu.customItems` 数组）。如果为 false，将禁用自定义菜单项；否则，将启用此项。

```
var showItem = true; // Change this to false to remove
var my_cm:ContextMenu = new ContextMenu(menuHandler);
my_cm.customItems.push(new ContextMenuItem("Hello", itemHandler));
function menuHandler(obj, menuObj) {
    if (showItem == false) {
        menuObj.customItems[0].enabled = false;
    } else {
        menuObj.customItems[0].enabled = true;
    }
}
```

```
function itemHandler(obj, item) {
    //...put code here...
    trace("selected!");
}
this.menu = my_cm;
```

当用户右键单击或按住 **Control** 单击舞台时，会显示自定义菜单。

另请参见

[menu \(Button.menu 属性\)](#), [onSelect \(ContextMenu.onSelect 处理函数\)](#), [customItems \(ContextMenu.customItems 属性\)](#), [hideBuiltInItems \(ContextMenu.hideBuiltInItems 方法\)](#), [menu \(MovieClip.menu 属性\)](#), [menu \(TextField.menu 属性\)](#)

## copy (ContextMenu.copy 方法)

```
public copy(): ContextMenu
```

创建指定的 **ContextMenu** 对象的副本。该副本继承初始菜单对象的所有属性。

可用性: **ActionScript 1.0** ; **Flash Player 7**

返回

ContextMenu - 一个 **ContextMenu** 对象。

示例

此示例创建名为 `my_cm` 的 **ContextMenu** 对象的副本（其内置菜单项被隐藏），并添加具有文本 **“Save...”** 的菜单项。然后，该示例创建 `my_cm` 的副本，并将它分配给变量 `clone_cm`，该变量继承原始菜单的所有属性。

```
var my_cm:ContextMenu = new ContextMenu();
my_cm.hideBuiltInItems();
var menuItem_cmi:ContextMenuItem = new ContextMenuItem("Save...",
    saveHandler);
my_cm.customItems.push(menuItem_cmi);
function saveHandler(obj, menuItem) {
    // saveDocument();
    // custom function (not shown)
    trace("something");
}
clone_cm = my_cm.copy();
this.menu = my_cm;
for (var i in clone_cm.customItems) {
    trace("clone_cm-> "+clone_cm.customItems[i].caption);
}
for (var i in my_cm.customItems) {
    trace("my_cm-> "+my_cm.customItems[i].caption);
}
```

## customItems (ContextMenu.customItems 属性)

public customItems : Array

**ContextMenu** 对象的数组。数组中的每个对象表示您已经定义的上下文菜单项。使用此属性可添加、删除或修改这些自定义菜单项。

若要添加新的菜单项，请首先创建一个新的 **ContextMenu** 对象，然后将其添加到 *menu\_mc*.customItems 数组（例如使用 `Array.push()`）。有关创建新菜单项的更多信息，请参见 **ContextMenu** 类条目。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例创建一个新的名为 *menuItem\_cmi* 的自定义菜单项（其标题为“发送电子邮件”）和一个名为 *emailHandler* 的回调处理函数。然后，使用 *customItems* 数组将新的菜单项添加到 **ContextMenu** 对象 *my\_cm*。最后，将新菜单附加到名为 *email\_mc* 的影片剪辑。若要使此示例正常运行，请在舞台上创建影片剪辑实例，然后使用属性检查器将该实例命名为 *email\_mc*。在“测试影片”模式中，如果在鼠标指针位于 *email\_mc* 影片剪辑上方时弹出上下文菜单，将出现新的上下文菜单项。

```
var my_cm:ContextMenu = new ContextMenu();
var menuItem_cmi:ContextMenuItem = new ContextMenuItem("Send e-mail",
    emailHandler);
my_cm.customItems.push(menuItem_cmi);
email_mc.menu = my_cm;
function emailHandler() {
    trace("sending email");
}
```

### 另请参见

[menu \(Button.menu 属性\)](#), [menu \(MovieClip.menu 属性\)](#), [menu \(TextField.menu 属性\)](#), [push \(Array.push 方法\)](#)

## hideBuiltInItems (ContextMenu.hideBuiltInItems 方法)

```
public hideBuiltInItems() : Void
```

隐藏指定的 **ContextMenu** 对象中的所有内置菜单项（“设置”除外）。如果 **Flash** 调试播放器正在运行，则显示“调试”菜单项，但如果 **SWF** 文件未启用远程调试功能，则该菜单项将变暗。

此方法仅隐藏标准上下文菜单中显示的菜单项；它不影响编辑菜单或错误菜单中显示的菜单项。

通过将 `my_cm.builtInItems` 的所有布尔成员设置为 `false` 可使此方法起作用。您可以有选择地显示内置菜单项，方法是将其在 `my_cm.builtInItems` 中的相应成员设置为 `true`（如下面的示例所示）。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例创建其内置菜单项已隐藏（“打印”除外）的新的 **ContextMenu** 对象 `my_cm`。然后将该菜单对象附加到当前时间轴。

```
var my_cm:ContextMenu = new ContextMenu();
my_cm.hideBuiltInItems();
my_cm.builtInItems.print = true;
this.menu = my_cm;
```

## onSelect (ContextMenu.onSelect 处理函数)

```
onSelect = function(item:Object, item_menu:Object) {}
```

在用户调用 **Flash Player** 上下文菜单并在该菜单实际显示之前调用。此事件处理函数允许根据当前应用程序状态自定义上下文菜单的内容。

也可以在构造新的 **ContextMenu** 对象时为 **ContextMenu** 对象指定回调处理函数。有关更多信息，请参见 **ContextMenuItem** `onSelect` 条目。

可用性：ActionScript 1.0；Flash Player 7

### 参数

**item:Object** - 对特定对象（影片剪辑、按钮或可选择的文本字段）的引用，当调用 **Flash Player** 上下文菜单时，该对象位于鼠标指针下，并且该对象的 `menu` 属性设置为有效的 **ContextMenu** 对象。

**item\_menu:Object** - 对分配给 `object` 的 `menu` 属性的 **ContextMenu** 对象的引用。

## 示例

以下示例确定对哪种类型的对象调用了上下文菜单。

```
my_cm:ContextMenu = new ContextMenu();
function menuHandler(obj:Object, menu:ContextMenu) {
    if(obj instanceof MovieClip) {
        trace("Movie clip: " + obj);
    }
    if(obj instanceof TextField) {
        trace("Text field: " + obj);
    }
    if(obj instanceof Button) {
        trace("Button: " + obj);
    }
}
my_cm.onSelect = menuHandler;
my_mc.menu = my_cm;
my_btn.menu = my_cm;
```

# ContextMenu

```
Object
|
+-ContextMenuItem
```

```
public dynamic class ContextMenuItem
extends Object
```

使用 **ContextMenuItem** 类可创建在 **Flash Player** 上下文菜单中显示的自定义菜单项。每个 **ContextMenuItem** 对象均具有在上下文菜单中显示的标题（文本）以及在选择菜单项时调用的回调处理函数（函数）。若要向上下文菜单添加新的上下文菜单项，请将其添加到 **ContextMenu** 对象的 **customItems** 数组。

您可以启用或禁用特定菜单项，显示或隐藏菜单项，或者更改与菜单项关联的标题或回调处理函数。

自定义菜单项出现在上下文菜单的顶部，并位于所有内置菜单项之上。自定义菜单项与内置菜单项始终由分隔条隔开。向上下文菜单添加的自定义菜单项不得超过 15 个。每个菜单项必须至少包含一个可见字符，控制字符、换行符和其它空白字符将被忽略。所有菜单项的长度不得超过 100 个字符。如果菜单项与任何内置菜单项或其它自定义菜单项相同，则无论匹配菜单项是否可见，均会忽略该菜单项。对菜单项进行比较时将忽略大小写、标点和空格。

自定义菜单项中不能出现以下字词：**Macromedia**、**Flash Player** 或**设置**。

可用性：ActionScript 1.0；Flash Player 7

属性摘要

修饰符	属性	说明
	caption:String	一个字符串，指定上下文菜单中显示的菜单项标题（文本）。
	enabled:Boolean	布尔值，指示是启用还是禁用指定的菜单项。
	separatorBefore:Boolean	布尔值，指示分隔条是否应显示在指定的菜单项上方。
	visible:Boolean	布尔值，指示在显示 Flash Player 上下文菜单时，指定的菜单项是否可见。

继承自 Object 类的属性

<a href="#">constructor</a> (Object.constructor 属性), <a href="#">__proto__</a> (Object.__proto__ 属性), <a href="#">prototype</a> (Object.prototype 属性), <a href="#">__resolve</a> (Object.__resolve 属性)
--

事件摘要

事件	说明
onSelect = function(obj:Object, menuItem:Object) {}	当从 Flash Player 上下文菜单中选择指定的菜单项时调用。

构造函数摘要

签名	说明
ContextMenuItem(caption:String, callbackFunction:Function, [separatorBefore:Boolean], [enabled:Boolean], [visible:Boolean])	创建一个可添加到 ContextMenu.customItems 数组的新 ContextMenuItem 对象。

方法摘要

修饰符	签名	说明
	copy() : ContextMenuItem	创建并返回指定的 ContextMenuItem 对象的副本。

继承自 Object 类的方法

<a href="#">addProperty</a> (Object.addProperty 方法), <a href="#">hasOwnProperty</a> (Object.hasOwnProperty 方法), <a href="#">isPropertyEnumerable</a> (Object.isPropertyEnumerable 方法), <a href="#">isPrototypeOf</a> (Object.isPrototypeOf 方法), <a href="#">registerClass</a> (Object.registerClass 方法), <a href="#">toString</a> (Object.toString 方法), <a href="#">unwatch</a> (Object.unwatch 方法), <a href="#">valueOf</a> (Object.valueOf 方法), <a href="#">watch</a> (Object.watch 方法)
---

## caption (ContextMenuItem.caption 属性)

public caption : String

一个字符串，指定上下文菜单中显示的菜单项标题（文本）。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例显示“输出”面板中选定菜单项（“暂停游戏”）的标题。

```
var my_cm:ContextMenu = new ContextMenu();
var menuItem_cmi:ContextMenuItem = new ContextMenuItem("Pause Game",
    onPause);
my_cm.customItems.push(menuItem_cmi);
function onPause(obj, menuItem) {
    trace("You chose: " + menuItem.caption);
}
this.menu = my_cm;
```

## ContextMenuItem 构造函数

public ContextMenuItem(caption:String, callbackFunction:Function,  
 [separatorBefore:Boolean], [enabled:Boolean], [visible:Boolean])

创建一个可添加到 ContextMenu.customItems 数组的新 ContextMenuItem 对象。

可用性：ActionScript 1.0；Flash Player 7

### 参数

**caption:**String - 一个字符串，它指定与菜单项关联的文本。

**callbackFunction:**Function - 您定义的函数，当选择菜单项时调用它。

**separatorBefore:**Boolean [ 可选 ] - 一个布尔值，指示分隔条是否应该出现在上下文菜单中的菜单项之上。默认值为 false。

**enabled:**Boolean [ 可选 ] - 一个布尔值，指示是启用还是禁用上下文菜单中的菜单项。默认值为 true。

**visible:**Boolean [ 可选 ] - 一个布尔值，指示菜单项是否可见。默认值为 true。



## 示例

此示例将“开始”和“停止”菜单项（由分隔条隔开）添加到 **ContextMenu** 对象 my\_cm。当从上下文菜单中选择“开始”时，将调用 startHandler() 函数；当选择“停止”时，将调用 stopHandler()。**ContextMenu** 对象会应用到当前时间轴。

```
var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Start", startHandler));
my_cm.customItems.push(new ContextMenuItem("Stop", stopHandler, true));
function stopHandler(obj, item) {
    trace("Stopping...");
}
function startHandler(obj, item) {
    trace("Starting...");
}
this.menu = my_cm;
```

## copy（ContextMenuItem.copy 方法）

```
public copy(): ContextMenuItem
```

创建并返回指定的 **ContextMenuItem** 对象的副本。该副本包含初始对象的所有属性。

可用性：ActionScript 1.0；Flash Player 7

## 返回

ContextMenuItem - **ContextMenuItem** 对象。

## 示例

此示例创建一个名为 original\_cmi 的新 **ContextMenuItem** 对象，其标题为“暂停”，回调处理函数设置为函数 onPause。然后，此示例创建 **ContextMenuItem** 对象的副本，并将其分配给变量 copy\_cmi。

```
var original_cmi:ContextMenuItem = new ContextMenuItem("Pause", onPause);
function onPause(obj:Object, menu:ContextMenu) {
    trace("pause me");
}

var copy_cmi:ContextMenuItem = original_cmi.copy();

var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(original_cmi);
my_cm.customItems.push(copy_cmi);

my_mc.menu = my_cm;
```

## enabled（ContextMenuItem.enabled 属性）

public enabled : Boolean

布尔值，指示是启用还是禁用指定的菜单项。默认情况下，此属性为 true。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例创建两个新的上下文菜单项：“开始”和“停止”。当用户选择“开始”时，会跟踪距 SWF 文件开始还剩下的毫秒数。然后，在菜单中禁用“开始”。当选择“停止”时，会跟踪自 SWF 文件开始以来已过去的毫秒数。“开始”菜单项会被重新启用，而“停止”菜单项会被禁用。

```
var my_cm:ContextMenu = new ContextMenu();
var startMenuItem:ContextMenu.Item = new ContextMenu.Item("Start",
    startHandler);
startMenuItem.enabled = true;
my_cm.customItems.push(startMenuItem);
var stopMenuItem:ContextMenu.Item = new ContextMenu.Item("Stop", stopHandler,
    true);
stopMenuItem.enabled = false;
my_cm.customItems.push(stopMenuItem);
function stopHandler(obj, item) {
    trace("Stopping... "+getTimer()+"ms");
    startMenuItem.enabled = true;
    stopMenuItem.enabled = false;
}
function startHandler(obj, item) {
    trace("Starting... "+getTimer()+"ms");
    startMenuItem.enabled = false;
    stopMenuItem.enabled = true;
}
this.menu = my_cm;
```

## onSelect (ContextMenuItem.onSelect 处理函数)

```
onSelect = function(obj:Object, menuItem:Object) {}
```

当从 **Flash Player** 上下文菜单中选择指定的菜单项时调用。指定的回调处理函数会接收两个参数: `obj` (对在用户调用 **Flash Player** 上下文菜单时位于鼠标下的对象的引用) 和 `item` (对表示选定菜单项的 **ContextMenuItem** 对象的引用)。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**obj:Object** - 对用户右键单击或按住 **Control** 单击的对象 (影片剪辑、时间轴、按钮或可选择的文本字段) 的引用。

**menuItem:Object** - 对选定 **ContextMenuItem** 对象的引用。

### 示例

以下示例确定对哪种类型的对象调用了上下文菜单。

```
var my_cmi:ContextMenu = new ContextMenu();
var start_cmi:ContextMenuitem = new ContextMenuItem("Start");
start_cmi.onSelect = function(obj, item) {
    trace("You chose: "+item.caption);
};
my_cmi.customItems.push(start_cmi);
my_cmi.customItems.push(new ContextMenuItem("Stop", stopHandler, true));
function stopHandler(obj, item) {
    trace("Stopping...");
}
this.menu = my_cmi;
```

### 另请参见

[onSelect \(ContextMenu.onSelect 处理函数\)](#)

## separatorBefore (ContextMenuItem.separatorBefore 属性)

public separatorBefore : Boolean

布尔值，指示分隔条是否应显示在指定的菜单项上方。默认情况下，此属性为 `false`。



任何自定义菜单项和内置菜单项之间始终会出现分隔条。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 示例

此示例创建三个菜单项，其标签为“打开”、“保存”和“打印”。“保存”和“打印”菜单项由分隔条隔开。然后，将这些菜单项添加到 **ContextMenu** 对象的 `customItems` 数组。最后，该菜单被附加到 **SWF** 文件的当前时间轴上。

```
var my_cm:ContextMenu = new ContextMenu();
var open_cmi:ContextMenuItem = new ContextMenuItem("Open", itemHandler);
var save_cmi:ContextMenuItem = new ContextMenuItem("Save", itemHandler);
var print_cmi:ContextMenuItem = new ContextMenuItem("Print", itemHandler);
print_cmi.separatorBefore = true;
my_cm.customItems.push(open_cmi, save_cmi, print_cmi);
function itemHandler(obj, menuItem) {
    trace("You chose: " + menuItem.caption);
};
this.menu = my_cm;
```

### 另请参见

[onSelect \(ContextMenu.onSelect 处理函数\)](#)

## visible (ContextMenuItem.visible 属性)

public visible : Boolean

布尔值，指示在显示 **Flash Player** 上下文菜单时，指定的菜单项是否可见。默认情况下，此属性为 true。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 示例

下面的示例创建两个新的上下文菜单项：“开始”和“停止”。当用户选择“开始”时，会显示距 **SWF** 文件开始还剩下的毫秒数。然后，“开始”在菜单中消失。当选择“停止”时，会显示自 **SWF** 文件开始以来已过去的毫秒数。“开始”菜单项会出现，而“停止”菜单项会消失。

```
var my_cm:ContextMenu = new ContextMenu();
var startMenuItem:ContextMenuitem = new ContextMenuItem("Start",
    startHandler);
startMenuItem.visible = true;
my_cm.customItems.push(startMenuItem);
var stopMenuItem:ContextMenuitem = new ContextMenuItem("Stop", stopHandler,
    true);
stopMenuItem.visible = false;
my_cm.customItems.push(stopMenuItem);
function stopHandler(obj, item) {
    trace("Stopping... "+getTimer()+"ms");
    startMenuItem.visible = true;
    stopMenuItem.visible = false;
}
function startHandler(obj, item) {
    trace("Starting... "+getTimer()+"ms");
    startMenuItem.visible = false;
    stopMenuItem.visible = true;
}
this.menu = my_cm;
```

# ConvolutionFilter

## (flash.filters.ConvolutionFilter)

```
Object
|
+- flash.filters.BitmapFilter
   |
   +- flash.filters.ConvolutionFilter
```

```
public class ConvolutionFilter
extends BitmapFilter
```

**ConvolutionFilter** 类应用矩阵卷积滤镜效果。卷积将输入图像的像素与相邻的像素合并以生成图像。通过卷积，可以实现大量不同的图像处理操作，包括模糊、边缘检测、锐化、浮雕和斜角。您可以对位图和 **MovieClip** 实例应用此效果。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时对影片剪辑应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤消设置操作。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter()` 将获得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作时对图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

矩阵盘绕基于一个 **n x m** 矩阵，该矩阵说明输入图像中的给定像素值如何与其相邻的像素值合并以生成结果像素值。每个结果像素通过将矩阵应用到相应的源像素及其相邻像素来确定。

对于 **3 x 3** 矩阵卷积，将以下公式用于每个独立的颜色通道：

$$\text{dst}(x, y) = ((\text{src}(x-1, y-1) * a_0 + \text{src}(x, y-1) * a_1 + \dots + \text{src}(x, y+1) * a_7 + \text{src}(x+1, y+1) * a_8) / \text{divisor}) + \text{bias}$$

某些规格的滤镜在由提供 SSE（SIMD 流扩展）的处理器运行时执行速度更快。

- 该滤镜必须是 **3 x 3** 滤镜。
- 所有滤镜项必须是介于 **-127** 和 **+127** 之间的整数。
- 所有滤镜项的总和不能包含大于 **127** 的绝对值。
- 如果任何滤镜项为负，则除数必须介于 **2.00001** 和 **256** 之间。
- 如果所有滤镜项都为正，则除数必须介于 **1.1** 和 **256** 之间。
- 偏差必须是整数。

如果结果图像的宽度或高度将超过 2880 像素，则不应用滤镜。例如，如果您在放大某大型影片剪辑时使用了滤镜，则在结果图像超过 2880 像素的限制时，该滤镜将关闭。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[applyFilter](#) ([BitmapData.applyFilter](#) 方法)，[filters](#) ([MovieClip.filters](#) 属性)，[cacheAsBitmap](#) ([MovieClip.cacheAsBitmap](#) 属性)

属性摘要

修饰符	属性	说明
	<code>alpha:Number</code>	替换颜色的 Alpha 透明度值。
	<code>bias:Number</code>	要添加到矩阵转换结果的偏差。
	<code>clamp:Boolean</code>	指示是否应锁定图像。
	<code>color:Number</code>	要替换源图像之外的像素的十六进制颜色。
	<code>divisor:Number</code>	矩阵转换中使用的除数。
	<code>matrix:Array</code>	用于矩阵转换的值的数组；返回一个副本。
	<code>matrixX:Number</code>	矩阵的 x 维度（矩阵中列的数目）。
	<code>matrixY:Number</code>	矩阵的 y 维度（矩阵中行的数目）。
	<code>preserveAlpha:Boolean</code>	指示卷积的应用对象。

继承自 `Object` 类的属性

[constructor](#) ([Object.constructor](#) 属性)，[\\_\\_proto\\_\\_](#) ([Object.\\_\\_proto\\_\\_](#) 属性)，[prototype](#) ([Object.prototype](#) 属性)，[\\_\\_resolve](#) ([Object.\\_\\_resolve](#) 属性)

构造函数摘要

签名	说明
<code>ConvolutionFilter(matrixX:Number, matrixY:Number, matrix:Array, [divisor:Number], [bias:Number], [preserveAlpha:Boolean], [clamp:Boolean], [color:Number], [alpha:Number])</code>	用指定参数初始化 <code>ConvolutionFilter</code> 实例。

方法摘要

修饰符	签名	说明
	<code>clone() : ConvolutionFilter</code>	返回此滤镜对象的副本。

继承自 `BitmapFilter` 类的方法

[clone](#) ([BitmapFilter.clone](#) 方法)

继承自 `Object` 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

# alpha (ConvolutionFilter.alpha 属性)

`public alpha : Number`

替换颜色的 Alpha 透明度值。有效值为 0 到 1.0。默认值为 0。例如，.25 设置透明度值为 25%。默认值为 1.0。

可用性: `ActionScript 1.0` ; `Flash Player 8`

## 示例

下面的示例将 `filter` 的 `alpha` 属性从默认值 1 更改为 .35。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var alpha:Number = .35;
var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1, 1, 1, 1], 9, 0, true, false, 0x0000FF, alpha);

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xCCFF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128, 0, 255, 1 | 2 | 4 | 8, false);

mc.onPress = function() {
```



```

        myBitmapData.applyFilter(myBitmapData, new Rectangle(0, 0, 98, 78), new
        Point(2, 2), filter);
    }

```

## bias（ConvolutionFilter.bias 属性）

public bias : Number

要添加到矩阵转换结果的偏差。默认值为 0。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例将 filter 的 bias 属性从默认值 1 更改为 50。

```

import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;

var bias:Number = 50;
var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1,
    1, 1, 1], 9, bias);

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128);

mc.onPress = function() {
    myBitmapData.applyFilter(myBitmapData, myBitmapData.rectangle, new
    Point(0, 0), filter);
}

```

## clamp（ConvolutionFilter.clamp 属性）

public clamp : Boolean

指示是否应锁定图像。对于源图像之外的像素，如果值为 true，则表明通过复制输入图像给定边缘处的颜色值，沿着输入图像的每个边框按需要扩展输入图像。如果值为 false，则表明应按照 color 和 alpha 属性中的指定使用其它颜色。默认值为 true。

可用性: ActionScript 1.0 ; Flash Player 8

## 示例

下面的示例将 `filter` 的 `clamp` 属性从其默认值 `true` 更改为 `false`。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var clamp:Boolean = false;
var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1,
    1, 1, 1], 9, 0, true, clamp, 0x00FF00, 1);

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xCCFF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128, 0, 255, 1 | 2 | 4 | 8, false);

mc.onPress = function() {
    myBitmapData.applyFilter(myBitmapData, new Rectangle(0, 0, 98, 78), new
        Point(2, -2), filter);
}
```

## clone (ConvolutionFilter.clone 方法)

```
public clone() : ConvolutionFilter
```

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 返回

`flash.filters.ConvolutionFilter` - 与原始实例具有完全相同的属性的新 **ConvolutionFilter** 实例。

## 示例

下面的示例创建三个 **ConvolutionFilter** 对象并对它们进行比较: `filter_1` 是通过使用 `ConvolutionFilter` 构造函数创建的; `filter_2` 是通过将其设置为等于 `filter_1` 创建的; 而 `clonedFilter` 是通过克隆 `filter_1` 创建的。请注意, 尽管 `filter_2` 等效于 `filter_1`, 但 `clonedFilter` 却不等效于 `filter_1`, 尽管它们包含相同的值。

```
import flash.filters.ConvolutionFilter;

var filter_1:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1,
    1, 1, 1, 1], 9);
var filter_2:ConvolutionFilter = filter_1;
var clonedFilter:ConvolutionFilter = filter_1.clone();
```

```

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> alpha: 0
    // >> color: 0
    // >> clamp: true
    // >> preserveAlpha: true
    // >> bias: 0
    // >> divisor: 9
    // >> matrix: 1,1,1,1,1,1,1,1,1
    // >> matrixY: 3
    // >> matrixX: 3
}

for(var i in clonedFilter) {
    trace(">> " + i + ": " + clonedFilter[i]);
    // >> clone: [type Function]
    // >> alpha: 0
    // >> color: 0
    // >> clamp: true
    // >> preserveAlpha: true
    // >> bias: 0
    // >> divisor: 9
    // >> matrix: 1,1,1,1,1,1,1,1,1
    // >> matrixY: 3
    // >> matrixX: 3
}

```

为了进一步说明 `filter_1`、`filter_2` 和 `clonedFilter` 之间的关系，下面的示例修改了 `filter_1` 的 `bias` 属性。通过修改 `bias`，说明 `clone()` 方法是根据 `filter_1` 的值而不是通过在引用中指向这些值来创建新实例的。

```

import flash.filters.ConvolutionFilter;

var filter_1:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1,
    1, 1, 1, 1], 9);
var filter_2:ConvolutionFilter = filter_1;
var clonedFilter:ConvolutionFilter = filter_1.clone();
trace(filter_1.bias); // 0
trace(filter_2.bias); // 0
trace(clonedFilter.bias); // 0

filter_1.bias = 20;

trace(filter_1.bias); // 20
trace(filter_2.bias); // 20
trace(clonedFilter.bias); // 0

```

## color (ConvolutionFilter.color 属性)

public color : Number

要替换源图像之外的像素的十六进制颜色。这是一个没有 Alpha 成分的 RGB 值。默认值为 0。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例将 filter 的 color 属性的默认值从 0 更改为 0xFF0000。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var color:Number = 0x0000FF;
var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1,
    1, 1, 1], 9, 0, true, false, color, 1);

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xCCFF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128, 0, 255, 1 | 2 | 4 | 8, false);

var height:Number = 100;
var width:Number = 80;
mc.onPress = function() {
    height -= 2;
    width -= 2;
    myBitmapData.applyFilter(myBitmapData, new Rectangle(0, 0, height, width),
        new Point(2, 2), filter);
}
```

## ConvolutionFilter 构造函数

```
public ConvolutionFilter(matrixX:Number, matrixY:Number, matrix:Array,
    [divisor:Number], [bias:Number], [preserveAlpha:Boolean], [clamp:Boolean],
    [color:Number], [alpha:Number])
```

用指定参数初始化 ConvolutionFilter 实例。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**matrixX:**Number - 矩阵的 x 维度（矩阵中列的数目）。默认值是 0。

**matrixY:**Number - 矩阵的 y 维度（矩阵中行的数目）。默认值是 0。

**matrix:**Array - 用于矩阵转换的值的数组；返回一个副本。数组中的项数必须等于 matrixX\*matrixY。

**divisor:**Number [ 可选 ] - 矩阵转换中使用的除数。默认值为 1。如果除数是所有矩阵值的总和，则可调平结果的总体色彩强度。忽略 0 值，此时使用默认值。

**bias:**Number [ 可选 ] - 要添加到矩阵转换结果的偏差。默认值是 0。

**preserveAlpha:**Boolean [ 可选 ] - 值为 false 表明盘绕应用于所有通道，包括 Alpha 通道。值为 true 表示只对颜色通道应用盘绕。默认值为 true。

**clamp:**Boolean [ 可选 ] - 对于源图像之外的像素，如果值为 true，则表明通过复制输入图像给定边缘处的颜色值，沿着输入图像的每个边框按需要扩展输入图像。如果值为 false，则表明应按照 color 和 alpha 属性中的指定使用其它颜色。默认值为 true。

**color:**Number [ 可选 ] - 要替换源图像之外的像素的十六进制颜色。

**alpha:**Number [ 可选 ] - 替换颜色的 Alpha。

## 示例

以下代码创建除数为 9 的 3 x 3 卷积滤镜。该滤镜将使图像模糊显示：

```
var myArray:Array = [1, 1, 1, 1, 1, 1, 1, 1, 1];
var myFilter:ConvolutionFilter = new flash.filters.ConvolutionFilter(3,
3, myArray, 9);
```

下面的示例创建一个 **ConvolutionFilter** 对象，它具有四个所需的参数，即 matrixX、matrixY、matrix 和 divisor。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;

var matrixX:Number = 3;
var matrixY:Number = 3;
var matrix:Array = [1, 1, 1, 1, 1, 1, 1, 1, 1];
var divisor:Number = 9;

var filter:ConvolutionFilter = new ConvolutionFilter(matrixX, matrixY,
matrix, divisor);

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128);

mc.onPress = function() {
myBitmapData.applyFilter(myBitmapData, myBitmapData.rectangle, new
Point(0, 0), filter);
}
```

## divisor (ConvolutionFilter.divisor 属性)

public divisor : Number

矩阵转换中使用的除数。默认值为 1。如果除数是所有矩阵值的总和，则可调平结果的总体色彩强度。忽略 0 值，此时使用默认值。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例将 filter 的 divisor 属性更改为 6。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;

var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1,
    1, 1, 1], 9);

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128);

mc.onPress = function() {
    var newDivisor:Number = 6;
    filter.divisor = newDivisor;
    myBitmapData.applyFilter(myBitmapData, myBitmapData.rectangle, new
        Point(0, 0), filter);
}
```

## matrix (ConvolutionFilter.matrix 属性)

public matrix : Array

用于矩阵转换的值的数组；返回一个副本。数组中的项数必须等于 matrixX\*matrixY。

matrix 属性不能通过直接修改这些值（例如，myFilter.matrix[2] = 1;）来更改。相反，您必须获取对数组的引用，然后对该引用进行更改并使用 filter.matrix = newMatrix; 重置该值，如下面的示例所示。

可用性: ActionScript 1.0 ; Flash Player 8

## 示例

下面的示例将 filter 的 matrix 属性从使位图模糊更改为使位图清晰。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;

var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1,
1, 1, 1], 9);

var myBitmapData:BitmapData = new BitmapData(100, 80, false, 0x00FF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128);

mc.onPress = function() {
    var newMatrix:Array = [0, -1, 0, -1, 8, -1, 0, -1, 0];
    filter.matrix = newMatrix;
    myBitmapData.applyFilter(myBitmapData, myBitmapData.rectangle, new
    Point(0, 0), filter);
}
```

## matrixX (ConvolutionFilter.matrixX 属性)

public matrixX : Number

矩阵的 x 维度（矩阵中列的数目）。默认值是 0。

**可用性:** ActionScript 1.0 ; Flash Player 8

## 示例

下面的示例显示 filter 的 matrixX 属性。

```
import flash.filters.ConvolutionFilter;

var filter:ConvolutionFilter = new ConvolutionFilter(2, 3, [1, 0, 0, 1, 0,
0], 6);
trace(filter.matrixX); // 2
```

## matrixY (ConvolutionFilter.matrixY 属性)

public matrixY : Number

矩阵的 y 维度（矩阵中行的数目）。默认值是 0。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例显示 filter 的 matrixY 属性。

```
import flash.filters.ConvolutionFilter;

var filter:ConvolutionFilter = new ConvolutionFilter(2, 3, [1, 0, 0, 1, 0,
    0], 6);
trace(filter.matrixY); // 3
```

## preserveAlpha (ConvolutionFilter.preserveAlpha 属性)

public preserveAlpha : Boolean

指示卷积的应用对象。值为 false 表明盘绕应用于所有通道，包括 Alpha 通道。值为 true 表示只对颜色通道应用盘绕。默认值为 true。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将 filter 的 preserveAlpha 属性的默认值从 true 更改为 false。

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;

var preserveAlpha:Boolean = false;
var filter:ConvolutionFilter = new ConvolutionFilter(3, 3, [1, 1, 1, 1, 1, 1,
    1, 1, 1], 9, 0, preserveAlpha);

var myBitmapData:BitmapData = new BitmapData(100, 80, true, 0xCCFF0000);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.attachBitmap(myBitmapData, this.getNextHighestDepth());
myBitmapData.noise(128, 0, 255, 1 | 2 | 4 | 8, false);

mc.onPress = function() {
    myBitmapData.applyFilter(myBitmapData, myBitmapData.rectangle, new
        Point(0, 0), filter);
}
```



# CustomActions

```
Object
|
+-CustomActions

public class CustomActions
extends Object
```

**CustomActions** 类的方法使得在 **Flash** 创作工具中播放的 **SWF** 文件可以管理任何用该创作工具注册的自定义动作。**SWF** 文件可安装和卸载自定义动作，检索自定义动作的 **XML** 定义，以及检索已注册自定义动作的列表。

您可使用这些方法来生成属于 **Flash** 创作工具扩展功能的 **SWF** 文件。例如，这样的扩展功能可使用“**Flash** 应用程序协议”定位 **UDDI** 储备库，并将 **Web** 服务下载到“动作”工具箱。

可用性：ActionScript 1.0；Flash Player 6

属性摘要  
继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 方法摘要

修饰符	签名	说明
static	get(name:String) : String	读取名为 name 的自定义动作 XML 定义文件的内容。
static	install(name:String, data:String) : Boolean	安装由 name 参数指定的新自定义动作 XML 定义文件。
static	list() : Array	返回一个 Array 对象，该对象包含 Flash 创作工具中所有已注册的自定义动作的名称。
static	uninstall(name:String) : Boolean	删除名为 name 的自定义动作 XML 定义文件。

继承自 **Object** 类的方法

```
addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)
```

## get (CustomActions.get 方法)

```
public static get(name:String) : String
```

读取名为 name 的自定义动作 XML 定义文件的内容。

定义文件的名称必须是简单文件名，没有 .xml 文件扩展名，也没有任何目录分隔符（“:”、“/” 或 “\”）。

如果找不到由 name 指定的定义文件，则返回值 undefined。如果找到了由 name 参数指定的自定义动作 XML 定义，则该定义将被完整地读取，并以字符串形式返回。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**name:String** - 要检索的自定义动作定义的名称。

### 返回

**String** - 如果找到自定义动作 XML 定义，则返回字符串；否则，返回 undefined。

### 示例

下面的示例列出 **ComboBox** 实例中的自定义动作，并在单击 **Button** 实例时获取自定义动作。将 **ComboBox**、**Button** 和 **TextArea** 的实例拖到舞台上。为 **ComboBox** 指定实例名称 customActionName\_cb；为 **TextArea** 指定实例名称 customActionXml\_ta；为 **Button** 指定实例名称 view\_button。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
import mx.controls.*;

var customActionName_cb:ComboBox;
var customActionXml_ta:TextArea;
var view_button:Button;

customActionName_cb.dataProvider = CustomActions.list();

customActionXml_ta.editable = false;

var viewListener:Object = new Object();
viewListener.click = function(evt:Object) {
    var caName:String = String(customActionName_cb.selectedItem);
    customActionXml_ta.text = CustomActions.get(caName);
};
view_button.addEventListener("click", viewListener);
```

## install (CustomActions.install 方法)

`public static install(name:String, data:String) : Boolean`

安装由 `name` 参数指定的新自定义动作 XML 定义文件。该文件的内容由字符串 `customXML` 指定。

定义文件的名称必须是简单文件名，没有 `.xml` 文件扩展名，也没有任何目录分隔符（“:”、“/”或“\”）。

如果已存在名为 `name` 的自定义动作文件，则该文件将被覆盖。

如果调用此方法时不存在目录 `Configuration/ActionsPanel/CustomActions`，则将创建该目录。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**name:String** - 要安装的自定义动作定义的名称。

**data:String** - 要安装的 XML 定义的文本。

### 返回

**Boolean** - 如果安装过程中发生错误，则返回布尔值 `false`；否则，返回值 `true`，表示该自定义动作已成功安装。

### 示例

下面的示例将 XML 文件中的信息安装到“动作”面板中。打开文本编辑器并保存一个名为 `dogclass.xml` 的新文档。输入以下代码：

```
<?xml version="1.0"?>
<customactions>
  <actionspanel>
    <folder version="7" id="DogClass" index="true" name="Dog" tiptext="Dog
Class">
      <string version="7" id="getFleas" name="getFleas" tiptext="gets number
of fleas" text=".getFleas(% fleas %)" />
    </folder>
  </actionspanel>
  <coloursyntax>
    <identifier text=".getFleas" />
  </coloursyntax>
  <codehints>
    <typeinfo pattern="_dog" object="Dog"/>
  </codehints>
</customactions>
```

然后，在同一目录中打开一个新的 **FLA** 文件并选择时间轴的第 1 帧。在“动作”面板中输入下面的代码：

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    trace(success);
    CustomActions.install("dogclass", this.firstChild);
    trace(CustomActions.list());
};
my_xml.load("dogclass.xml");
```

选择“控制” > “测试影片”，如果 XML 加载成功，您将看到 `true` 以及一个数组，该数组包含使用 **Flash** 创作工具在“输出”面板中注册的所有自定义动作的名称。关闭 **SWF** 文件，然后打开“动作”面板。您将在“动作”工具箱中看到一个名为“**Dog**”的新项，在该文件夹内，您会看到 `getFleas`。

## list (CustomActions.list 方法)

```
public static list() : Array
```

返回一个 **Array** 对象，该对象包含 **Flash** 创作工具中所有已注册的自定义动作的名称。该数组的元素均是简单名称，没有 `.xml` 文件扩展名，并且没有任何目录分隔符（例如“`:`”、“`/`”或“`\`”）。如果没有已注册的自定义动作，则 `list()` 将返回一个长度为零的数组。如果发生错误，则 `list()` 将返回值 `undefined`。

可用性：ActionScript 1.0；Flash Player 6

### 返回

Array - 一个数组。

### 示例

下面的示例列出 **ComboBox** 实例中的自定义动作，并在单击 **Button** 实例时获取自定义动作。将 **ComboBox**、**Button** 和 **TextArea** 的实例拖到舞台上。为 **ComboBox** 指定实例名称 `customActionName_cb`；为 **TextArea** 指定实例名称 `customActionXml_ta`；为 **Button** 指定实例名称 `view_button`。在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
import mx.controls.*;

var customActionName_cb:ComboBox;
var customActionXml_ta:TextArea;
var view_button:Button;

customActionName_cb.dataProvider = CustomActions.list();

customActionXml_ta.editable = false;
```

```
var viewListener:Object = new Object();
viewListener.click = function(evt:Object) {
    var caName:String = String(customActionName_cb.selectedItem);
    customActionXml_ta.text = CustomActions.get(caName);
};
view_button.addEventListener("click", viewListener);
```

## uninstall (CustomActions.uninstall 方法)

```
public static uninstall(name:String) : Boolean
```

删除名为 name 的自定义动作 XML 定义文件。

名，也没有任何目录分隔符（“:”、“/”或“\”）。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**name:**String - 要卸载的自定义动作定义的名称。

### 返回

Boolean - 如果没有找到名为 name 的自定义动作，则返回布尔值 false。如果自定义动作被成功删除，则返回值 true。

### 示例

下面的示例安装一个新的自定义动作并显示一个数组，该数组包含使用 **Flash** 创作工具在“输出”面板中注册的所有自定义动作的名称。当单击 **uninstall\_btn** 时，将会卸载该自定义动作。将会显示一个包含已安装的自定义动作的名称的数组，然后会从该数组中删除 **dogclass**。创建一个名为 **uninstall\_btn** 的按钮，然后在时间轴的第 1 帧上输入以下 **ActionScript**:

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    trace(success);
    CustomActions.install("dogclass", this.firstChild);
    trace(CustomActions.list());
};
my_xml.load("dogclass.xml");

uninstall_btn.onRelease = function() {
    CustomActions.uninstall("dogclass");
    trace(CustomActions.list());
};
```

有关创建 **dogclass.xml** 的信息，请参见 **CustomActions.install()**。

另请参见

[install \(CustomActions.install 方法\)](#)

# Date

```
Object
|
+-Date
```

```
public class Date
extends Object
```

**Date** 类用于检索相对于通用时间（格林尼治平均时，现在叫做通用时间或 UTC）或相对于运行 **Flash Player** 的操作系统的时间和值。**Date** 类的方法不是静态方法，但仅应用于调用该方法时指定的单个 **Date** 对象。`Date.UTC()` 方法属于异常；它是一个静态方法。

**Date** 类以不同的方式处理夏时制，具体方式取决于操作系统和 **Flash Player** 的版本。在下面的操作系统中，**Flash Player 6** 及更高版本以这些方式处理夏时制：

- **Windows - Date** 对象自动调整其夏时制输出。**Date** 对象检测在当前区域设置内是否应用了夏时制，如果是，它将检测标准时到夏时制转换的日期和时间。然而，当前有效的转换日期会应用到以前和将来的日期，所以如果该区域设置具有不同的转换日期，则对于以前的日期，夏时制偏差的计算结果可能会不正确。
- **Mac OS X - Date** 对象自动调整其夏时制输出。**Mac OS X** 中的时区信息数据库用于确定现在或以前的任何日期或时间是否应该应用夏时制偏差。
- **Mac OS 9 -** 该操作系统只提供了足够的信息以确定当前日期和时间是否应该应用夏时制偏差。相应地，**date** 对象假定当前的夏时制偏差应用于以前或将来的所有日期和时间。

在下面的操作系统中，**Flash Player 5** 处理夏时制方式如下所示：

- **Windows -** 始终应用美国的夏时制规则，但对于那些应用夏时制，但却具有与美国不同的转换时间的欧洲和其它地区，这将导致错误的转换。**Flash** 会正确地检测当前区域设置中是否使用了夏时制。

若要调用 **Date** 类的方法，必须首先按照本节后面部分所述使用 **Date** 类的构造函数创建一个 **Date** 对象。

可用性：ActionScript 1.0；Flash Player 5

属性摘要

继承自 Object 类的属性

<code>constructor</code> (Object.constructor 属性), <code>__proto__</code> (Object.__proto__ 属性), <code>prototype</code> (Object.prototype 属性), <code>__resolve</code> (Object.__resolve 属性)
---

构造函数摘要

签名	说明
<code>Date([yearOrTimevalue:Number], [month:Number], [date:Number], [hour:Number], [minute:Number], [second:Number], [millisecond:Number])</code>	构造一个新的 Date 对象，该对象将保存指定的日期和时间。

方法摘要

修饰符	签名	说明
	<code>getDate() : Number</code>	按照本地时间返回指定的 Date 对象中表示月中某天的值（1 到 31 之间的整数）。
	<code>getDay() : Number</code>	按照本地时间返回指定的 Date 对象中表示周几的值（0 代表星期日，1 代表星期一，依此类推）。
	<code>getFullYear() : Number</code>	按照本地时间返回指定的 Date 对象中的完整年份值（一个 4 位数，例如 2000）。
	<code>getHours() : Number</code>	按照本地时间返回指定的 Date 对象中的小时值（0 到 23 之间的整数）。
	<code>getMilliseconds() : Number</code>	按照本地时间返回指定的 Date 对象中的毫秒数（0 到 999 之间的整数）。
	<code>getMinutes() : Number</code>	按照本地时间返回指定的 Date 对象中的分钟值（0 到 59 之间的整数）。
	<code>getMonth() : Number</code>	按照本地时间返回指定的 Date 对象中的月份值（0 代表一月，1 代表二月，依此类推）。
	<code>getSeconds() : Number</code>	按照本地时间返回指定的 Date 对象中的秒钟值（0 到 59 之间的整数）。
	<code>getTime() : Number</code>	返回指定的 Date 对象自 1970 年 1 月 1 日午夜（通用时间）以来的毫秒数。
	<code>getTimezoneOffset() : Number</code>	以分钟为单位，返回计算机的本地时间和通用时间之间的差值。

修饰符	签名	说明
	<code>getUTCDate() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中表示月中某天的值（1 到 31 之间的整数）。
	<code>getUTCDay() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中表示周几的值（0 代表星期日，1 代表星期一，依此类推）。
	<code>getUTCFullYear() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中用 4 位数字表示的年份值。
	<code>getUTCHours() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中的小时值（0 到 23 之间的整数）。
	<code>getUTCMilliseconds() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中的毫秒数（0 到 999 之间的整数）。
	<code>getUTCMinutes() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中的分钟值（0 到 59 之间的整数）。
	<code>getUTCMonth() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中的月份值（0 [ 一月 ] 到 11 [ 十二月 ]）。
	<code>getUTCSeconds() : Number</code>	按照通用时间返回指定的 <code>Date</code> 对象中的秒钟值（0 到 59 之间的整数）。
	<code>getUTCYear() : Number</code>	按照通用时间 (UTC) 返回此 <code>Date</code> 的年份。
	<code>getYear() : Number</code>	按照本地时间返回指定的 <code>Date</code> 对象的年份。
	<code>setDate(date:Number) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象的月份中的日期，并以毫秒为单位返回新时间。
	<code>setFullYear(year:Number, [month:Number], [date:Number]) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象的年份，并以毫秒为单位返回新时间。
	<code>setHours(hour:Number) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象的小时值，并以毫秒为单位返回新时间。
	<code>setMilliseconds(milliseconds:Number) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象的毫秒数，并以毫秒为单位返回新时间。
	<code>setMinutes(minute:Number) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象的分钟值，并以毫秒为单位返回新时间。
	<code>setMonth(month:Number, [date:Number]) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象的月份，并以毫秒为单位返回新时间。
	<code>setSeconds(second:Number) : Number</code>	按照本地时间设置指定的 <code>Date</code> 对象中的秒钟值，并以毫秒为单位返回新时间。



修饰符	签名	说明
	<code>setTime(milliseconds:Number) : Number</code>	以毫秒为单位设置指定的 <code>Date</code> 对象自 1970 年 1 月 1 日 午夜以来的日期，并以毫秒为单位返回新时间。
	<code>setUTCDate(date:Number) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象的日期值，并以毫秒为单位返回新时间。
	<code>setUTCFullYear(year:Number, [month:Number], [date:Number]) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象 ( <code>my_date</code> ) 的年份，并以毫秒为单位返回新时间。
	<code>setUTCHours(hour:Number, [minute:Number], [second:Number], [milliseconds:Number]) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象的小时值，并以毫秒为单位返回新时间。
	<code>setUTCMilliseconds(milliseconds:Number) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象的毫秒数，并以毫秒为单位返回新时间。
	<code>setUTCMinutes(minute:Number, [second:Number], [milliseconds:Number]) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象的分钟值，并以毫秒为单位返回新时间。
	<code>setUTCMonth(month:Number, [date:Number]) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象的月份，还可以选择设置日期，并以毫秒为单位返回新时间。
	<code>setUTCSeconds(second:Number, [milliseconds:Number]) : Number</code>	按照通用时间设置指定的 <code>Date</code> 对象的秒钟值，并以毫秒为单位返回新时间。
	<code>setYear(year:Number) : Number</code>	按照本地时间设置指定 <code>Date</code> 对象的年份值，并以毫秒为单位返回新时间。
	<code>toString() : String</code>	以可读格式返回指定的 <code>Date</code> 对象的字符串值。
<code>static</code>	<code>UTC(year:Number, month:Number, [date:Number], [hour:Number], [minute:Number], [second:Number], [milliseconds:Number]) : Number</code>	返回 1970 年 1 月 1 日 午夜（通用时间）与参数中指定的时间之间相差的毫秒数。
	<code>valueOf() : Number</code>	返回此 <code>Date</code> 自 1970 年 1 月 1 日 午夜（通用时间）以来的毫秒数。

继承自 Object 类的方法

---

`addProperty` (Object.addProperty 方法), `hasOwnProperty` (Object.hasOwnProperty 方法), `isPropertyEnumerable` (Object.isPropertyEnumerable 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `registerClass` (Object.registerClass 方法), `toString` (Object.toString 方法), `unwatch` (Object.unwatch 方法), `valueOf` (Object.valueOf 方法), `watch` (Object.watch 方法)

---

## Date 构造函数

```
public Date([yearOrTimevalue:Number], [month:Number], [date:Number],  
            [hour:Number], [minute:Number], [second:Number], [millisecond:Number])
```

构造一个新的 **Date** 对象，该对象将保存指定的日期和时间。

`Date()` 构造函数使用最多七个参数 (**year**、**month**、... **millisecond**) 将日期和时间指定为毫秒。或者，可以将一个值传递至指示时间值（基于自 GMT 时间 1970 年 1 月 1 日 0:00:00 以来的毫秒数）的 `Date()` 构造函数。或者，您可以不指定任何参数，对于当前日期和时间，会为其分配 **Date** 对象 `Date()`。

例如，此代码显示了创建 **Date** 对象的几种不同方法：

```
var d1:Date = new Date();  
var d3:Date = new Date(2000, 0, 1);  
var d4:Date = new Date(65, 2, 6, 9, 30, 15, 0);  
var d5:Date = new Date(-14159025000);
```

在第一行代码中，**Date** 对象被设置为运行赋值语句的时间。

在第二行中，使用传递至 **Date** 对象的年份、月份和日期参数创建 **Date** 对象，从而生成 GMT 时间 2000 年 1 月 1 日 0:00:00。

在第三行中，使用传递至 **Date** 对象的年份、月份和日期参数创建 **Date** 对象，从而生成 GMT 时间 1965 年 3 月 6 日 09:30:15 (+ 0 毫秒)。请注意，由于将年份参数指定为两位数的整数，所以将其解释为 1965 年。

在第四行，只传递一个参数，该参数是表示 GMT 时间 1970 年 1 月 1 日 0:00:00 之前或之后的毫秒数的时间值；由于该值是负数，因此它表示 GMT 时间 1970 年 1 月 1 日 0:00:00 之前的某个时间，在此例中，该时间为 GMT 时间 1969 年 7 月 21 日 02:56:15。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**yearOrTimevalue:Number** [可选] - 如果指定了其它参数，则此数字表示年份（如 1965）；否则，它表示时间值。如果该数字表示年份，则 0 至 99 之间的值表示 1900 年至 1999 年；否则，必须指定表示年份的所有 4 位数字。如果该数字表示时间值（未指定任何其它参数），则为 GMT 时间 1970 年 1 月 1 日 0:00:00 之前或之后的毫秒数；负值表示 GMT 时间 1970 年 1 月 1 日 0:00:00 之前 的某个时间，而正值表示该时间之后的某个时间。

**month**:Number [ 可选 ] - 0 （一月）到 11 （十二月）之间的整数。

**date**:Number [ 可选 ] - 1 到 31 之间的整数。

**hour**:Number [ 可选 ] - 0 （午夜）到 23 （晚上 11 点）之间的整数。

**minute**:Number [ 可选 ] - 0 到 59 之间的整数。

**second**:Number [ 可选 ] - 0 到 59 之间的整数。

**millisecond**:Number [ 可选 ] - 0 到 999 之间的整数（毫秒）。

## 示例

下面的示例检索当前日期和时间：

```
var now_date:Date = new Date();
```

下面的示例为 **Mary** 的生日（1974 年 8 月 12 日）创建一个新的 **Date** 对象（由于月份参数从零开始，所以此示例使用 7 而不是 8 表示月份）：

```
var maryBirthDay:Date = new Date (74, 7, 12);
```

下面的示例创建一个新的 **Date** 对象并将 `Date.getMonth()`、`Date.getDate()` 和 `Date.getFullYear()` 各自的返回值连接在一起：

```
var today_date:Date = new Date();
var date_str:String = ((today_date.getMonth()+1)+"/"+today_date.getDate()+"/"
    +today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

## 另请参见

[getMonth \(Date.getMonth 方法\)](#)，[getDate \(Date.getDate 方法\)](#)，[getFullYear \(Date.getFullYear 方法\)](#)

## getDate (Date.getDate 方法)

```
public getDate() : Number
```

按照本地时间返回指定的 **Date** 对象中表示月中某天的值（1 到 31 之间的整数）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0 ； Flash Player 5

## 返回

Number - 一个整数。

## 示例

下面的示例创建一个新的 **Date** 对象并将 `Date.getMonth()`、`Date.getDate()` 和 `Date.getFullYear()` 各自的返回值连接在一起：

```
var today_date:Date = new Date();
var date_str:String = (today_date.getDate()+"/"+(today_date.getMonth()+1)+"/"
    "+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

## 另请参见

[getMonth \(Date.getMonth 方法\)](#)，[getFullYear \(Date.getFullYear 方法\)](#)

## getDay (Date.getDay 方法)

```
public getDay() : Number
```

按照本地时间返回指定的 **Date** 对象中表示周几的值（0 代表星期日，1 代表星期一，依此类推）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0；Flash Player 5

## 返回

Number - 表示某一天是周几的一个整数。

## 示例

下面的示例创建一个新的 **Date** 对象并使用 `getDay()` 确定当前为周几：

```
var dayOfWeek_array:Array = new Array("Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday", "Friday", "Saturday");
var today_date:Date = new Date();
var day_str:String = dayOfWeek_array[today_date.getDay()];
trace("Today is "+day_str);
```

## getFullYear (Date.getFullYear 方法)

```
public getFullYear() : Number
```

按照本地时间返回指定的 **Date** 对象中的完整年份值（一个 4 位数，例如 2000）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0；Flash Player 5

## 返回

Number - 表示年份的一个整数。

## 示例

下面的示例使用构造函数创建 **Date** 对象。**trace** 语句显示由 `getFullYear()` 方法返回的值。

```
var my_date:Date = new Date();
trace(my_date.getFullYear()); // displays 104
trace(my_date.getFullYear()); // displays current year
```

## getHours (Date.getHours 方法)

```
public getHours() : Number
```

按照本地时间返回指定的 **Date** 对象中的小时值（0 到 23 之间的整数）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

## 返回

Number - 一个整数。

## 示例

下面的示例使用构造函数基于当前时间创建 **Date** 对象，并使用 `getHours()` 方法显示来自该对象的小时值：

```
var my_date:Date = new Date();
trace(my_date.getHours());

var my_date:Date = new Date();
var hourObj:Object = getHoursAmPm(my_date.getHours());
trace(hourObj.hours);
trace(hourObj.ampm);

function getHoursAmPm(hour24:Number):Object {
    var returnObj:Object = new Object();
    returnObj.ampm = (hour24<12) ? "AM" : "PM";
    var hour12:Number = hour24%12;
    if (hour12 == 0) {
        hour12 = 12;
    }
    returnObj.hours = hour12;
    return returnObj;
}
```

## getMilliseconds (Date.getMilliseconds 方法)

```
public getMilliseconds() : Number
```

按照本地时间返回指定的 **Date** 对象中的毫秒数（0 到 999 之间的整数）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例使用构造函数基于当前时间创建 **Date** 对象，并使用 `getMilliseconds()` 方法返回来自该对象的毫秒数：

```
var my_date:Date = new Date();  
trace(my_date.getMilliseconds());
```

## getMinutes (Date.getMinutes 方法)

```
public getMinutes() : Number
```

按照本地时间返回指定的 **Date** 对象中的分钟值（0 到 59 之间的整数）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例使用构造函数基于当前时间创建 **Date** 对象，并使用 `getMinutes()` 方法返回来自该对象的分钟值：

```
var my_date:Date = new Date();  
trace(my_date.getMinutes());
```

## getMonth（Date.getMonth 方法）

public getMonth() : Number

按照本地时间返回指定的 **Date** 对象中的月份值（0 代表一月，1 代表二月，依此类推）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例使用构造函数基于当前时间创建 **Date** 对象，并使用 `getMonth()` 方法返回来自该对象的月份值：

```
var my_date:Date = new Date();  
trace(my_date.getMonth());
```

下面的示例使用构造函数基于当前时间创建 **Date** 对象，使用 `getMonth()` 方法将当前月份显示为数值，并显示月份的名称。

```
var my_date:Date = new Date();  
trace(my_date.getMonth());  
trace(getMonthAsString(my_date.getMonth()));  
function getMonthAsString(month:Number):String {  
    var monthNames_array:Array = new Array("January", "February", "March",  
        "April", "May", "June", "July", "August", "September", "October",  
        "November", "December");  
    return monthNames_array[month];  
}
```

## getSeconds（Date.getSeconds 方法）

public getSeconds() : Number

按照本地时间返回指定的 **Date** 对象中的秒钟值（0 到 59 之间的整数）。本地时间由运行 **Flash Player** 的操作系统确定。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

## 示例

下面的示例使用构造函数基于当前时间创建 **Date** 对象，并使用 `getSeconds()` 方法返回来自该对象的秒钟值：

```
var my_date:Date = new Date();
trace(my_date.getSeconds());
```

## getTime (Date.getTime 方法)

```
public getTime() : Number
```

返回指定的 **Date** 对象自 1970 年 1 月 1 日午夜（通用时间）以来的毫秒数。当比较两个或更多个 **Date** 对象时，使用此方法表示某一特定时刻。

可用性：ActionScript 1.0；Flash Player 5

## 返回

Number - 一个整数。

## 示例

下面的示例使用构造函数基于当前时间创建 **Date** 对象，并使用 `getTime()` 方法返回自 1970 年 1 月 1 日午夜以来的毫秒数：

```
var my_date:Date = new Date();
trace(my_date.getTime());
```

## getTimezoneOffset (Date.getTimezoneOffset 方法)

```
public getTimezoneOffset() : Number
```

以分钟为单位，返回计算机的本地时间和通用时间之间的差值。

可用性：ActionScript 1.0；Flash Player 5

## 返回

Number - 一个整数。

## 示例

下面的示例返回 **San Francisco** 本地夏时制时间和通用时间之间的差值。只有当 **Date** 对象中定义的日期在夏时制期间到达时，才考虑夏时制因素，将其计入返回结果。此示例中的输出为 420 分钟，显示在“输出”面板中（7 小时 \* 60 分钟 / 小时 = 420 分钟）。此示例为太平洋夏令时（PDT，GMT-0700）。结果因位置和年份时间而异。

```
var my_date:Date = new Date();
trace(my_date.getTimezoneOffset());
```



## getUTCDate (Date.getUTCDate 方法)

`public getUTCDate() : Number`

按照通用时间返回指定的 **Date** 对象中表示月中某天的值（1 到 31 之间的整数）。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `Date.getUTCDate()` 和 `Date.getDate()`。  
`Date.getUTCDate()` 返回的值与 `Date.getDate()` 返回的值可能会不同，具体取决于您当地的时区与通用时间之间的关系。

```
var my_date:Date = new Date(2004,8,25);  
trace(my_date.getUTCDate()); // output: 25
```

另请参见

[getDate \(Date.getDate 方法\)](#)

## getUTCDay (Date.getUTCDay 方法)

`public getUTCDay() : Number`

按照通用时间返回指定的 **Date** 对象中表示周几的值（0 代表星期日，1 代表星期一，依此类推）。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `Date.getUTCDay()` 和 `Date.getDay()`。  
`Date.getUTCDay()` 返回的值与 `Date.getDay()` 返回的值可能会不同，具体取决于您当地的时区与通用时间之间的关系。

```
var today_date:Date = new Date();  
trace(today_date.getDay()); // output will be based on local timezone  
trace(today_date.getUTCDay()); // output will equal getDay() plus or minus one
```

另请参见

[getDay \(Date.getDay 方法\)](#)

## getUTCFullYear (Date.getUTCFullYear 方法)

`public getUTCFullYear() : Number`

按照通用时间返回指定的 **Date** 对象中用 4 位数字表示的年份值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `Date.getUTCFullYear()` 和 `Date.getFullYear()`。如果当天的日期是 12 月 31 日或 1 月 1 日，则 `Date.getUTCFullYear()` 返回的值与 `Date.getFullYear()` 返回的值可能会不同，具体取决于您当地的时区与通用时间之间的关系。

```
var today_date:Date = new Date();
trace(today_date.getFullYear()); // display based on local timezone
trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

### 另请参见

[getFullYear \(Date.getFullYear 方法\)](#)

## getUTCHours (Date.getUTCHours 方法)

`public getUTCHours() : Number`

按照通用时间返回指定的 **Date** 对象中的小时值（0 到 23 之间的整数）。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `Date.getUTCHours()` 和 `Date.getHours()`。`Date.getUTCHours()` 返回的值与 `Date.getHours()` 返回的值可能会不同，具体取决于您当地的时区与通用时间之间的关系。

```
var today_date:Date = new Date();
trace(today_date.getHours()); // display based on local timezone
trace(today_date.getUTCHours()); // display equals getHours() plus or minus 12
```

另请参见

[getHours](#) ([Date.getHours](#) 方法)

## getUTCMilliseconds (Date.getUTCMilliseconds 方法)

`public getUTCMilliseconds() : Number`

按照通用时间返回指定的 **Date** 对象中的毫秒数（0 到 999 之间的整数）。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `getUTCMilliseconds()` 返回来自该 **Date** 对象的毫秒数。

```
var today_date:Date = new Date();
trace(today_date.getUTCMilliseconds());
```

## getUTCMinutes (Date.getUTCMinutes 方法)

`public getUTCMinutes() : Number`

按照通用时间返回指定的 **Date** 对象中的分钟值（0 到 59 之间的整数）。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `getUTCMinutes()` 返回来自该 **Date** 对象的分钟值：

```
var today_date:Date = new Date();
trace(today_date.getUTCMinutes());
```

## getUTCMonth (Date.getUTCMonth 方法)

`public getUTCMonth() : Number`

按照通用时间返回指定的 **Date** 对象中的月份值（0 [一月] 到 11 [十二月]）。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `Date.getUTCMonth()` 和 `Date.getMonth()`。如果当天的日期是某月的第一天或最后一天，则 `Date.getUTCMonth()` 返回的值与 `Date.getMonth()` 返回的值可能会不同，具体取决于您当地的时区与通用时间之间的关系。

```
var today_date:Date = new Date();
trace(today_date.getMonth()); // output based on local timezone
trace(today_date.getUTCMonth()); // output equals getMonth() plus or minus 1
```

### 另请参见

[getMonth \(Date.getMonth 方法\)](#)

## getUTCSeconds (Date.getUTCSeconds 方法)

`public getUTCSeconds() : Number`

按照通用时间返回指定的 **Date** 对象中的秒钟值（0 到 59 之间的整数）。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 `getUTCSeconds()` 返回来自该 **Date** 对象的秒钟值：

```
var today_date:Date = new Date();
trace(today_date.getUTCSeconds());
```

## getUTCYear (Date.getUTCYear 方法)

public getUTCYear() : Number

按照通用时间 (UTC) 返回此 Date 的年份。这里的年份是指完整的年份值减去 1900。例如，2000 年表示为 100。

可用性：ActionScript 1.0；Flash Player 8

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个新的 **Date** 对象，并使用 Date.getUTCFullYear() 和 Date.getFullYear()。如果当天的日期是 12 月 31 日或 1 月 1 日，则 Date.getUTCFullYear() 返回的值与 Date.getFullYear() 返回的值可能会不同，具体取决于您当地的时区与通用时间之间的关系。

```
var today_date:Date = new Date();

trace(today_date.getFullYear()); // display based on local timezone

trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

## getFullYear (Date.getFullYear 方法)

public getFullYear() : Number

按照本地时间返回指定的 **Date** 对象的年份。本地时间由运行 **Flash Player** 的操作系统确定。这里的年份是指完整的年份值减去 1900。例如，2000 年表示为 100。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个 **Date** 对象，将年份和月份设置为 2004 年和 5 月。Date.getFullYear() 方法返回 104，而 Date.getFullYear() 返回 2004：

```
var today_date:Date = new Date(2004,4);
trace(today_date.getFullYear()); // output: 104
trace(today_date.getFullYear()); // output: 2004
```

另请参见

[getFullYear \(Date.getFullYear 方法\)](#)

## setDate (Date.setDate 方法)

```
public setDate(date:Number) : Number
```

按照本地时间设置指定的 **Date** 对象的月份中的日期，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**date**:Number - 1 到 31 之间的整数。

### 返回

Number - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 15 日，然后使用 **Date.setDate()** 将该日期更改为 2004 年 5 月 25 日：

```
var today_date:Date = new Date(2004,4,15);  
trace(today_date.getDate()); //displays 15  
today_date.setDate(25);  
trace(today_date.getDate()); //displays 25
```

## setFullYear (Date.setFullYear 方法)

```
public setFullYear(year:Number, [month:Number], [date:Number]) : Number
```

按照本地时间设置指定的 **Date** 对象的年份，并以毫秒为单位返回新时间。如果指定了 **month** 和 **date** 参数，则将它们设置为本地时间。本地时间由运行 **Flash Player** 的操作系统确定。

调用此方法并不修改指定 **Date** 对象的其它字段，但如果因调用该方法而导致表示周几的值发生了变化，则 **Date.getUTCDay()** 和 **Date.getDay()** 可能报告一个新值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**year**:Number - 指定年份的 4 位数。两位数的数字不代表四位数的年份；例如，99 不代表 1999 年，而是表示 99 年。

**month**:Number [ 可选 ] - 0（一月）到 11（十二月）之间的整数。如果省略此参数，则不修改指定 **Date** 对象的月份字段。

**date**:Number [ 可选 ] - 1 到 31 之间的数字。如果省略此参数，则不会修改指定的 **Date** 对象的日期字段。

## 返回

Number - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 15 日，然后使用 `Date.setFullYear()` 将该日期更改为 2002 年 5 月 15 日：

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getFullYear()); //output: 2004
my_date.setFullYear(2002);
trace(my_date.getFullYear()); //output: 2002
```

## 另请参见

[getUTCDay \(Date.getUTCDay 方法\)](#), [getDay \(Date.getDay 方法\)](#)

## setHours (Date.setHours 方法)

```
public setHours(hour:Number) : Number
```

按照本地时间设置指定的 **Date** 对象的小时值，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

## 参数

**hour**:Number - 0（午夜）到 23（晚上 11 点）之间的整数。

## 返回

Number - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期和时间设置为 2004 年 5 月 15 日上午 8:00，然后使用 `Date.setHours()` 将该时间更改为下午 4:00：

```
var my_date:Date = new Date(2004,4,15,8);
trace(my_date.getHours()); // output: 8
my_date.setHours(16);
trace(my_date.getHours()); // output: 16
```

## setMilliseconds (Date.setMilliseconds 方法)

```
public setMilliseconds(milliseconds:Number) : Number
```

按照本地时间设置指定的 **Date** 对象的毫秒数，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**milliseconds**:Number - 0 到 999 之间的整数。

### 返回

Number - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 15 日上午 8:30，毫秒数设置为 250，然后使用 `Date.setMilliseconds()` 将毫秒数更改为 575：

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getMilliseconds()); // output: 250
my_date.setMilliseconds(575);
trace(my_date.getMilliseconds()); // output: 575
```

## setMinutes (Date.setMinutes 方法)

```
public setMinutes(minute:Number) : Number
```

按照本地时间设置指定的 **Date** 对象的分钟值，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**minute**:Number - 0 到 59 之间的整数。

### 返回

Number - 一个整数。



## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期和时间设置为 2004 年 5 月 15 日上午 8:00，然后使用 `Date.setMinutes()` 将该时间更改为上午 8:30:

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getMinutes()); // output: 0
my_date.setMinutes(30);
trace(my_date.getMinutes()); // output: 30
```

## setMonth（Date.setMonth 方法）

```
public setMonth(month:Number, [date:Number]) : Number
```

按照本地时间设置指定的 **Date** 对象的月份，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

## 参数

**month**:Number - 0（一月）到 11（十二月）之间的整数。

**date**:Number [ 可选 ] - 1 到 31 之间的整数。如果省略此参数，则不修改指定 **Date** 对象的日期字段。

## 返回

Number - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 15 日，然后使用 `Date.setMonth()` 将日期更改为 2004 年 6 月 15 日:

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getMonth()); //output: 4
my_date.setMonth(5);
trace(my_date.getMonth()); //output: 5
```

## setSeconds (Date.setSeconds 方法)

`public setSeconds(second:Number) : Number`

按照本地时间设置指定的 **Date** 对象中的秒钟值，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**second**:**Number** - 0 到 59 之间的整数。

### 返回

**Number** - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象，将日期和时间设置为 2004 年 5 月 15 日上午 8:00:00，然后使用 `Date.setSeconds()` 将时间更改为上午 8:00:45:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getSeconds()); // output: 0
my_date.setSeconds(45);
trace(my_date.getSeconds()); // output: 45
```

## setTime (Date.setTime 方法)

`public setTime(milliseconds:Number) : Number`

以毫秒为单位设置指定的 **Date** 对象自 1970 年 1 月 1 日 午夜以来的日期，并以毫秒为单位返回新时间。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**milliseconds**:**Number** - 一个数字；整数值，其中 0 表示通用时间 1 月 1 日午夜。

### 返回

**Number** - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期和时间设置为 2004 年 5 月 15 日上午 8:00，然后使用 `Date.setTime()` 将时间更改为上午 8:30：

```
var my_date:Date = new Date(2004,4,15,8,0,0);
var myDate_num:Number = my_date.getTime(); // convert my_date to milliseconds
myDate_num += 30 * 60 * 1000; // add 30 minutes in milliseconds
my_date.setTime(myDate_num); // set my_date Date object 30 minutes forward
trace(my_date.getFullYear()); // output: 2004
trace(my_date.getMonth()); // output: 4
trace(my_date.getDate()); // output: 15
trace(my_date.getHours()); // output: 8
trace(my_date.getMinutes()); // output: 30
```

## setUTCDate (Date.setUTCDate 方法)

```
public setUTCDate(date:Number) : Number
```

按照通用时间设置指定的 **Date** 对象的日期值，并以毫秒为单位返回新时间。调用此方法并不修改指定 **Date** 对象的其它字段，但如果因调用该方法而导致表示周几的值发生了变化，则 `Date.getUTCDay()` 和 `Date.getDay()` 可能报告一个新值。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**date:**Number - 一个数字：1 到 31 之间的整数。

### 返回

Number - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象（设置为当天的日期），使用 `Date.setUTCDate()` 将该日期值更改为 10，然后再将它更改为 25：

```
var my_date:Date = new Date();
my_date.setUTCDate(10);
trace(my_date.getUTCDate()); // output: 10
my_date.setUTCDate(25);
trace(my_date.getUTCDate()); // output: 25
```

### 另请参见

[getUTCDay \(Date.getUTCDay 方法\)](#)，[getDay \(Date.getDay 方法\)](#)

## setUTCFullYear (Date.setUTCFullYear 方法)

`public setUTCFullYear(year:Number, [month:Number], [date:Number]) : Number`

按照通用时间设置指定的 **Date** 对象 (**my\_date**) 的年份，并以毫秒为单位返回新时间。

还可选用此方法设置指定 **Date** 对象所表示的月份和日期值。调用此方法并不修改指定 **Date** 对象的其它字段，但如果因调用该方法而导致表示周几的值发生了变化，则 `Date.getUTCDay()` 和 `Date.getDay()` 可能报告一个新值。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**year**:Number - 一个整数，表示以完整的四位数年份的形式指定的年份，如 2000。

**month**:Number [ 可选 ] - 0（一月）到 11（十二月）之间的整数。如果省略此参数，则不修改指定 **Date** 对象的月份字段。

**date**:Number [ 可选 ] - 1 到 31 之间的整数。如果省略此参数，则不修改指定 **Date** 对象的日期字段。

### 返回

Number - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象（设置为当天的日期），使用

`Date.setUTCFullYear()` 将年份值更改为 2001，然后将日期更改为 1995 年 5 月 25 日：

```
var my_date:Date = new Date();
my_date.setUTCFullYear(2001);
trace(my_date.getUTCFullYear()); // output: 2001
my_date.setUTCFullYear(1995, 4, 25);
trace(my_date.getUTCFullYear()); // output: 1995
trace(my_date.getUTCMonth()); // output: 4
trace(my_date.getUTCDate()); // output: 25
```

### 另请参见

[getUTCDay \(Date.getUTCDay 方法\)](#)，[getDay \(Date.getDay 方法\)](#)

## setUTCHours (Date.setUTCHours 方法)

```
public setUTCHours(hour:Number, [minute:Number], [second:Number],  
    [millisecond:Number]) : Number
```

按照通用时间设置指定的 **Date** 对象的小时值，并以毫秒为单位返回新时间。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**hour**:Number - 一个数字； **0**（午夜）到 **23**（晚上 **11** 点）之间的整数。

**minute**:Number [ 可选 ] - 一个数字； **0** 到 **59** 之间的整数。如果省略此参数，则不会修改指定的 **Date** 对象的分钟字段。

**second**:Number [ 可选 ] - 一个数字；从 **0** 到 **59** 的整数。如果省略此参数，则不会修改指定的 **Date** 对象的秒钟字段。

**millisecond**:Number [ 可选 ] - 一个数字； **0** 到 **999** 之间的整数。如果省略此参数，则不会修改指定的 **Date** 对象的毫秒字段。

### 返回

Number - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象（设置为当天的日期），使用 `Date.setUTCHours()` 将时间更改为上午 **8:30**，然后再将时间更改为下午 **5:30:47**：

```
var my_date:Date = new Date();  
my_date.setUTCHours(8,30);  
trace(my_date.getUTCHours()); // output: 8  
trace(my_date.getUTCMinutes()); // output: 30  
my_date.setUTCHours(17,30,47);  
trace(my_date.getUTCHours()); // output: 17  
trace(my_date.getUTCMinutes()); // output: 30  
trace(my_date.getUTCSeconds()); // output: 47
```

## setUTCMilliseconds (Date.setUTCMilliseconds 方法)

```
public setUTCMilliseconds(milliseconds:Number) : Number
```

按照通用时间设置指定的 **Date** 对象的毫秒数，并以毫秒为单位返回新时间。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**milliseconds**:Number - 0 到 999 之间的整数。

### 返回

Number - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 15 日上午 8:30，毫秒数设置为 250，然后使用 `Date.setUTCMilliseconds()` 将毫秒数更改为 575：

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getUTCMilliseconds()); // output: 250
my_date.setUTCMilliseconds(575);
trace(my_date.getUTCMilliseconds()); // output: 575
```

## setUTCMinutes (Date.setUTCMinutes 方法)

```
public setUTCMinutes(minute:Number, [second:Number], [milliseconds:Number]) :
    Number
```

按照通用时间设置指定的 **Date** 对象的分钟值，并以毫秒为单位返回新时间。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**minute**:Number - 0 到 59 之间的整数。

**second**:Number [ 可选 ] - 0 到 59 之间的整数。如果省略此参数，则不会修改指定的 **Date** 对象的秒钟字段。

**milliseconds**:Number [ 可选 ] - 0 到 999 之间的整数。如果省略此参数，则不修改指定 **Date** 对象的毫秒字段。

### 返回

Number - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期和时间设置为 2004 年 5 月 15 日上午 8:00，然后使用 `Date.setUTCMinutes()` 将时间更改为上午 8:30:

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getUTCMinutes()); // output: 0
my_date.setUTCMinutes(30);
trace(my_date.getUTCMinutes()); // output: 30
```

## setUTCMonth（Date.setUTCMonth 方法）

```
public setUTCMonth(month:Number, [date:Number]): Number
```

按照通用时间设置指定的 **Date** 对象的月份，还可以选择设置日期，并以毫秒为单位返回新时间。调用此方法并不修改指定 **Date** 对象的其它字段，但如果因指定 `date` 参数的值而导致表示周几的值发生了变化，则 `Date.getUTCDay()` 和 `Date.getDay()` 可能会报告一个新值。

可用性: **ActionScript 1.0**；**Flash Player 5**

## 参数

**month:**Number - 0（一月）到 11（十二月）之间的整数。

**date:**Number [ 可选 ] - 1 到 31 之间的整数。如果省略此参数，则不修改指定 **Date** 对象的日期字段。

## 返回

Number - 一个整数。

## 示例

下面的示例首先创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 15 日，然后使用 `Date.setMonth()` 将日期更改为 2004 年 6 月 15 日:

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getUTCMonth()); // output: 4
today_date.setUTCMonth(5);
trace(today_date.getUTCMonth()); // output: 5
```

## 另请参见

[getUTCDay](#) ([Date.getUTCDay](#) 方法), [getDay](#) ([Date.getDay](#) 方法)

## setUTCSeconds (Date.setUTCSeconds 方法)

```
public setUTCSeconds(second:Number, [millisecond:Number]) : Number
```

按照通用时间设置指定的 **Date** 对象的秒钟值，并以毫秒为单位返回新时间。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**second**:Number - 0 到 59 之间的整数。

**millisecond**:Number [ 可选 ] - 0 到 999 之间的整数。如果省略此参数，则不修改指定 **Date** 对象的毫秒字段。

### 返回

Number - 一个整数。

### 示例

下面的示例首先创建一个新的 **Date** 对象，将日期和时间设置为 2004 年 5 月 15 日上午 8:00:00，然后使用 `Date.setSeconds()` 将时间更改为上午 8:30:45:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getUTCSeconds()); // output: 0
my_date.setUTCSeconds(45);
trace(my_date.getUTCSeconds()); // output: 45
```

## setYear (Date.setYear 方法)

```
public setYear(year:Number) : Number
```

按照本地时间设置指定 **Date** 对象的年份值，并以毫秒为单位返回新时间。本地时间由运行 **Flash Player** 的操作系统确定。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**year**:Number - 一个表示年份的数字。如果 **year** 是介于 0 和 99 之间的整数，则 `setYear` 将年份设置为 1900 + **year**；否则年份是 **year** 参数的值。

### 返回

Number - 一个整数。



## 示例

下面的示例创建一个新的 **Date** 对象，将日期设置为 2004 年 5 月 25 日，使用 `setYear()` 将年份更改为 1999，然后再将年份更改为 2003：

```
var my_date:Date = new Date(2004,4,25);
trace(my_date.getYear()); // output: 104
trace(my_date.getFullYear()); // output: 2004
my_date.setYear(99);
trace(my_date.getYear()); // output: 99
trace(my_date.getFullYear()); // output: 1999
my_date.setYear(2003);
trace(my_date.getYear()); // output: 103
trace(my_date.getFullYear()); // output: 2003
```

## toString（Date.toString 方法）

```
public toString() : String
```

以可读格式返回指定的 **Date** 对象的字符串值。

可用性：ActionScript 1.0；Flash Player 5

## 返回

String - 一个字符串。

## 示例

下面的示例将 **Date** 对象 `dateOfBirth_date` 中的信息作为字符串返回。**trace** 语句的输出按照本地时间进行，并根据具体时间而有所不同。太平洋夏令时的输出比通用时间的输出要早 7 个小时：**Mon Aug 12 18:15:00 GMT-0700 1974**。

```
var dateOfBirth_date:Date = new Date(74, 7, 12, 18, 15);
trace (dateOfBirth_date);
trace (dateOfBirth_date.toString());
```

## UTC (Date.UTC 方法)

```
public static UTC(year:Number, month:Number, [date:Number], [hour:Number],  
    [minute:Number], [second:Number], [millisecond:Number]) : Number
```

返回 1970 年 1 月 1 日午夜（通用时间）与参数中指定的时间之间相差的毫秒数。这是一个静态方法，它通过 **Date** 对象的构造函数进行调用，而不是通过特定的 **Date** 对象进行调用。此方法使您可以创建一个采用通用时间的 **Date** 对象，而 **Date** 构造函数采用本地时间。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**year**:Number - 表示年份的四位数整数（例如，2000）。

**month**:Number - 0（一月）到 11（十二月）之间的整数。

**date**:Number [可选] - 1 到 31 之间的整数。

**hour**:Number [可选] - 0（午夜）到 23（晚上 11 点）之间的整数。

**minute**:Number [可选] - 0 到 59 之间的整数。

**second**:Number [可选] - 0 到 59 之间的整数。

**millisecond**:Number [可选] - 0 到 999 之间的整数。

### 返回

Number - 一个整数。

### 示例

下面的示例创建一个以通用时间定义的新 **Date** 对象 `maryBirthday_date`。这是用于 `new Date` 构造函数方法的示例的通用时间变体。输出按照本地时间进行，根据具体时间而有所不同。太平洋夏令时的输出比 UTC 的输出要早 7 个小时：**Sun Aug 11 17:00:00 GMT-0700 1974**。

```
var maryBirthday_date:Date = new Date(Date.UTC(1974, 7, 12));  
trace(maryBirthday_date);
```

## valueOf (Date.valueOf 方法)

```
public valueOf() : Number
```

返回此 **Date** 自 1970 年 1 月 1 日午夜（通用时间）以来的毫秒数。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 毫秒数。

# DisplacementMapFilter

## (flash.filters.DisplacementMapFilter)

```
Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.DisplacementMapFilter
```

```
public class DisplacementMapFilter
extends BitmapFilter
```

**DisplacementMapFilter** 类使用来自指定的 **BitmapData** 对象（被称为置换映射图像）的像素值，以执行舞台对象（如 **MovieClip** 实例）的置换。您可以使用此滤镜在 **BitmapData** 或 **MovieClip** 实例上获得扭曲或斑点效果。

滤镜的具体使用取决于要应用滤镜的对象。

要在运行时对影片剪辑应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤消设置操作。

要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。在 **BitmapData** 对象上调用 `applyFilter()` 将修改该 **BitmapData** 对象，并且，此操作是无法撤消的。

您也可以在创作时对图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜使用以下公式：

$$\text{dstPixel}[x, y] = \text{srcPixel}[x + ((\text{componentX}(x, y) - 128) * \text{scaleX}) / 256, y + ((\text{componentY}(x, y) - 128) * \text{scaleY}) / 256]$$

其中，`componentX(x, y)` 从 `mapBitmap` 属性获得 `(x - mapPoint.x, y - mapPoint.y)` 处的 `componentX` 颜色值。

滤镜使用的映射图像会进行缩放，以匹配舞台缩放比例。当对象本身呈一定的比例时，它不会以任何方式进行缩放。

此滤镜支持舞台缩放，但是不支持常规缩放、旋转或倾斜。如果此对象本身进行了缩放（如果 `x` 缩放和 `y` 缩放不是 100%），则滤镜效果不缩放。只有在放大舞台时，滤镜效果才进行缩放。

下面是 `DisplacementMapFilter` 类的运行方式。对于目标位图中的每个像素 (x, y), `DisplacementMapFilter` 类执行以下操作:

- 从映射位图中的 (x,y) 获取颜色
- 根据该颜色计算偏移量
- 在源位图中查找该偏移位置 (x+dx,y+dy)
- 如果满足边界条件, 则将该像素写入目标 (x,y)

如果结果图像的宽度或高度将超过 2880 像素, 则不应用滤镜。例如, 如果您在放大某大型影片剪辑时使用了滤镜, 则在结果图像超过 2880 像素的限制时, 该滤镜将关闭。

可用性: `ActionScript 1.0` ; `Flash Player 8`

另请参见

`applyFilter` (`BitmapData.applyFilter` 方法), `filters` (`MovieClip.filters` 属性), `cacheAsBitmap` (`MovieClip.cacheAsBitmap` 属性)

属性摘要

修饰符	属性	说明
	<code>alpha:Number</code>	指定对于超出范围的替换应用的 Alpha 透明度值。
	<code>color:Number</code>	指定对于超出范围的替换应用什么颜色。
	<code>componentX:Number</code>	说明在映射图像中使用哪个颜色通道来置换 x 结果。
	<code>componentY:Number</code>	说明在映射图像中使用哪个颜色通道来置换 y 结果。
	<code>mapBitmap:BitmapData</code>	包含置换映射数据的 <code>BitmapData</code> 对象。
	<code>mapPoint:Point</code>	一个 <code>flash.geom.Point</code> 值, 它包含目标影片剪辑的左上角与映射图像左上角之间的偏移量。
	<code>mode:String</code>	滤镜模式。
	<code>scaleX:Number</code>	用于缩放映射计算的 x 置换结果的乘数。
	<code>scaleY:Number</code>	用于缩放映射计算的 y 置换结果的乘数。

继承自 `Object` 类的属性

`constructor` (`Object.constructor` 属性), `__proto__` (`Object.__proto__` 属性), `prototype` (`Object.prototype` 属性), `__resolve` (`Object.__resolve` 属性)

构造函数摘要

签名	说明
<code>DisplacementMapFilter(mapBitmap:BitmapData, mapPoint:Point, componentX:Number, componentY:Number, scaleX:Number, scaleY:Number, [mode:String], [color:Number], [alpha:Number])</code>	用指定参数初始化 <code>DisplacementMapFilter</code> 实例。

方法摘要

修饰符	签名	说明
	<code>clone() : DisplacementMapFilter</code>	返回此滤镜对象的副本。

继承自 `BitmapFilter` 类的方法

[clone \(BitmapFilter.clone 方法\)](#)

继承自 `Object` 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

alpha (DisplacementMapFilter.alpha 属性)

`public alpha : Number`

指定对于超出范围的替换应用的 **Alpha** 透明度值。它被指定为 **0.0** 到 **1.0** 之间的标准值。例如, **.25** 设置透明度值为 **25%**。默认值为 **0**。如果 `mode` 属性设置为 **3**, **COLOR**, 则使用此属性。

可用性: **ActionScript 1.0 ; Flash Player 8**

示例

下面的示例在用户单击现有 `MovieClip` `filteredMc` 时将其超出范围的 `alpha` 属性更改为 `0x00FF00`。

```
import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
```

```

import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.scaleY = 25;
    filter.mode = "color";
    filter.alpha = .25;
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
    mc.attachBitmap(bmp, this.getNextHighestDepth());

    return bmp;
}

```

```

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
        this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
        80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

## clone (DisplacementMapFilter.clone 方法)

public clone() : DisplacementMapFilter

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

返回

flash.filters.DisplacementMapFilter - 与原始实例具有完全相同的属性的新 **DisplacementMapFilter** 实例。

示例

下面的示例创建三个 **DisplacementMapFilter** 对象并对它们进行比较: filter\_1 是通过使用 DisplacementMapFilter 构造函数创建的; filter\_2 是通过将其设置为等于 filter\_1 创建的; 而 clonedFilter 是通过克隆 filter\_1 创建的。请注意, 尽管 filter\_2 等效于 filter\_1, 但 clonedFilter 却不等效于 filter\_1, 尽管它们包含相同的值。

```

import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
    "radial", true);

var filter_1:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
    new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);
var filter_2:DisplacementMapFilter = filter_1;
var clonedFilter:DisplacementMapFilter = filter_1.clone();

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> alpha: 0

```

```

        // >> color: 0
        // >> mode: wrap
        // >> scaleY: 10
        // >> scaleX: 10
        // >> componentY: 1
        // >> componentX: 1
        // >> mapPoint: (-30, -30)
        // >> mapBitmap: [object Object]
    }

    for(var i in clonedFilter) {
        trace(">> " + i + ": " + clonedFilter[i]);
        // >> clone: [type Function]
        // >> alpha: 0
        // >> color: 0
        // >> mode: wrap
        // >> scaleY: 10
        // >> scaleX: 10
        // >> componentY: 1
        // >> componentX: 1
        // >> mapPoint: (-30, -30)
        // >> mapBitmap: [object Object]
    }

    function createGradientBitmap(w:Number, h:Number, bgColor:Number,
        type:String, hide:Boolean):BitmapData {
        var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
        var matrix:Matrix = new Matrix();
        matrix.createGradientBox(w, h, 0, 0, 0);

        mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
            matrix, "pad");
        mc.lineTo(w, 0);
        mc.lineTo(w, h);
        mc.lineTo(0, h);
        mc.lineTo(0, 0);
        mc.endFill();
        (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

        var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
        bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
            bmp.rectangle, true);
        mc.attachBitmap(bmp, this.getNextHighestDepth());

        return bmp;
    }

```

为了进一步说明 `filter_1`、`filter_2` 和 `clonedFilter` 之间的关系，下面的示例将修改 `filter_1` 的 `mode` 属性。通过修改 `mode`，说明 `clone()` 方法是根据 `filter_1` 的值而不是通过在引用中指向这些值来创建新实例的。



```

import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
    "radial", true);

var filter_1:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
    new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);
var filter_2:DisplacementMapFilter = filter_1;
var clonedFilter:DisplacementMapFilter = filter_1.clone();

trace(filter_1.mode); // wrap
trace(filter_2.mode); // wrap
trace(clonedFilter.mode); // wrap

filter_1.mode = "ignore";

trace(filter_1.mode); // ignore
trace(filter_2.mode); // ignore
trace(clonedFilter.mode); // wrap

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
    mc.attachBitmap(bmp, this.getNextHighestDepth());

    return bmp;
}

```

## color (DisplacementMapFilter.color 属性)

public color : Number

指定对于超出范围的替换应用什么颜色。置换的有效范围是 **0.0** 到 **1.0**。值采用十六进制格式。color 的默认值为 **0**。如果 mode 属性设置为 3, COLOR, 则使用此属性。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有 **MovieClip** filteredMc 时将其超出范围的 color 属性更改为 0x00FF00。

```
import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.scaleY = 25;
    filter.mode = "color";
    filter.alpha = .25;
    filter.color = 0x00FF00;
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);
```

```

mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
matrix, "pad");
mc.lineTo(w, 0);
mc.lineTo(w, h);
mc.lineTo(0, h);
mc.lineTo(0, 0);
mc.endFill();
(hide == true) ? mc._alpha = 0 : mc._alpha = 100;

var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
    bmp.rectangle, true);
mc.attachBitmap(bmp, this.getNextHighestDepth());

return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
    this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
    80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

## componentX (DisplacementMapFilter.componentX 属性)

public componentX : Number

说明在映射图像中使用哪个颜色通道来置换 **x** 结果。可能的值包括 **1**（红色）、**2**（绿色）、**4**（蓝色）和 **8** (Alpha)。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的 **MovieClip** filteredMc 时更改其 componentX 属性。值从 **1** 更改为 **4**，颜色通道相应地从红色更改为蓝色。

```

import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

```

```

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.componentX = 4;
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
    mc.attachBitmap(bmp, this.getNextHighestDepth());

    return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
        this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
        80);
}

```

```

        txtBlock.txt.text = "watch the text bend with the displacement map";
        return txtBlock;
    }

```

另请参见

[BitmapData \(flash.display.BitmapData\)](#)

## componentY (DisplacementMapFilter.componentY 属性)

public componentY : Number

说明在映射图像中使用哪个颜色通道来置换 y 结果。可能的值包括 1（红色）、2（绿色）、4（蓝色）和 8 (Alpha)。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的 **MovieClip** filteredMc 时更改其 componentY 属性。值从 1 更改为 4，颜色通道相应地从红色更改为蓝色。

```

import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.componentY = 4;
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

```

```

}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
    mc.attachBitmap(bmp, this.getNextHighestDepth());

    return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
        this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
        80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

另请参见

[BitmapData \(flash.display.BitmapData\)](#)

# DisplacementMapFilter 构造函数

```
public DisplacementMapFilter(mapBitmap:BitmapData, mapPoint:Point,  
    componentX:Number, componentY:Number, scaleX:Number, scaleY:Number,  
    [mode:String], [color:Number], [alpha:Number])
```

用指定参数初始化 **DisplacementMapFilter** 实例。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**mapBitmap**:flash.display.BitmapData - 包含置换映射数据的 **BitmapData** 对象。

**mapPoint**:flash.geom.Point - 一个 flash.geom.Point 值, 它包含目标影片剪辑的左上角与映射图像左上角之间的偏移量。

**componentX**:Number - 描述在映射图像中使用哪个颜色通道来置换 **x** 结果。可能的值如下所示:

- 1 (红色)
- 2 (绿色)
- 4 (蓝色)
- 8 (Alpha)

**componentY**:Number - 描述在映射图像中使用哪个颜色通道来置换 **y** 结果。可能的值如下所示:

- 1 (红色)
- 2 (绿色)
- 4 (蓝色)
- 8 (Alpha)

**scaleX**:Number - 用于缩放映射计算的 **x** 置换结果的乘数。

**scaleY**:Number - 用于缩放映射计算的 **y** 置换结果的乘数。

**mode**:String [ 可选 ] - 滤镜的模式。可能的值如下所示:

- "wrap" - 将置换值折返到源图像的另一侧。
- "clamp" - 将置换值锁定在源图像的边缘。
- "ignore" - 如果置换值超出了范围, 则忽略置换并使用源像素。
- "color" - 如果置换值在图像外, 则替换由滤镜的 **alpha** 属性和 **color** 属性组成的像素值。

**color**:Number [ 可选 ] - 指定对于超出范围的替换应用什么颜色。置换的有效范围是 **0.0** 到 **1.0**。如果 **mode** 设置为 "color", 则使用此参数。

**alpha**:Number [ 可选 ] - 指定对于超出范围的替换应用什么 **Alpha** 值。它被指定为 **0.0** 到 **1.0** 之间的标准值。例如, **.25** 设置透明度值为 **25%**。默认值为 **1.0**。如果 **mode** 设置为 **"color"**, 则使用此参数。

## 示例

下面的构造函数创建一个新的滤镜实例:

```
myFilter = new flash.filters.DisplacementMapFilter (mapBitmap, mapPoint,  
    componentX, componentY, scale, [mode], [color], [alpha])
```

下面的示例使用放射状渐变位图实例化一个新的 **DisplacementMapFilter**, 并将其应用于包含 **MovieClip** 对象的文本 **txtBlock**。

```
import flash.filters.DisplacementMapFilter;  
import flash.display.BitmapData;  
import flash.geom.Point;  
import flash.geom.Matrix;  
import flash.geom.ColorTransform;  
  
var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,  
    "radial");  
  
var mapPoint:Point = new Point(-30, -30);  
var componentX:Number = 1;  
var componentY:Number = 1;  
var scaleX:Number = 10;  
var scaleY:Number = 10;  
var mode:String = "wrap";  
var color:Number = 0x000000;  
var alpha:Number = 0x000000;  
  
var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,  
    mapPoint, componentX, componentY, scaleX, scaleY, mode, color, alpha);  
  
var txtBlock:MovieClip = createTextBlock();  
txtBlock._x = 30;  
txtBlock._y = 30;  
  
txtBlock.filters = new Array(filter);  
  
function createGradientBitmap(w:Number, h:Number, bgColor:Number,  
    type:String, hide:Boolean):BitmapData {  
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);  
    var matrix:Matrix = new Matrix();  
    matrix.createGradientBox(w, h, 0, 0, 0);  
  
    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],  
        matrix, "pad");  
    mc.lineTo(w, 0);  
    mc.lineTo(w, h);
```



```

mc.lineTo(0, h);
mc.lineTo(0, 0);
mc.endFill();
(hide == true) ? mc._alpha = 0 : mc._alpha = 100;

var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
    bmp.rectangle, true);
mc.attachBitmap(bmp, this.getNextHighestDepth());

return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
    this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
    80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

## mapBitmap (DisplacementMapFilter.mapBitmap 属性)

public mapBitmap : BitmapData

包含置换映射数据的 **BitmapData** 对象。

mapBitmap 属性不能通过直接修改它的值来更改。相反，必须获得对 mapBitmap 的引用，对该引用进行更改，然后将 mapBitmap 设置为该引用。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的 **MovieClip** filteredMc 时更改其 mapBitmap 属性。

```

import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();
var scope:Object = this;

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.mapBitmap = scope.createGradientBitmap(300, 80, 0xFF000000,
    "linear");
    this.filters = new Array(filter);
}

```

```

}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
    mc.attachBitmap(bmp, this.getNextHighestDepth());

    return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
        this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
        80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

另请参见

[BitmapData \(flash.display.BitmapData\)](#)

## mapPoint (DisplacementMapFilter.mapPoint 属性)

public mapPoint : Point

一个 flash.geom.Point 值，它包含目标影片剪辑的左上角与映射图像左上角之间的偏移量。

mapPoint 属性不能通过直接修改它的值来更改。相反，必须获得对 mapPoint 的引用，对该引用进行更改，然后将 mapPoint 设置为该引用。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的 **MovieClip** filteredMc 时更改其 mapPoint 属性。

```
import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.mapPoint = new Point(-30, -40);
    this.filters = new Array(filter);
    this._x = 30;
    this._y = 40;
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);
```

```

mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
matrix, "pad");
mc.lineTo(w, 0);
mc.lineTo(w, h);
mc.lineTo(0, h);
mc.lineTo(0, 0);
mc.endFill();
(hide == true) ? mc._alpha = 0 : mc._alpha = 100;

var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
    bmp.rectangle, true);
mc.attachBitmap(bmp, this.getNextHighestDepth());

return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
    this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
    80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

另请参见

[Point \(flash.geom.Point\)](#)

## mode (DisplacementMapFilter.mode 属性)

public mode : String

滤镜模式。可能的值如下所示：

- "wrap" - 将置换值折返到源图像的另一侧。这是默认值。
- "clamp" - 将置换值锁定在源图像的边缘。
- "ignore" - 如果置换值超出了范围，则忽略置换并使用源像素。
- "color" - 如果置换值在图像外，则替换由滤镜的 alpha 属性和 color 属性组成的像素值。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例通过修改 `scaleY` 创建一个超出范围的置换值，然后在用户单击现有的 `MovieClip` `filteredMc` 时将其上的 `mode` 属性更改为 `ignore`。

```
import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.scaleY = 25;
    filter.mode = "ignore";
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
```

```

        bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
        mc.attachBitmap(bmp, this.getNextHighestDepth());

        return bmp;
    }

    function createTextBlock():MovieClip {
        var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
        this.getNextHighestDepth());
        txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
        80);
        txtBlock.txt.text = "watch the text bend with the displacement map";
        return txtBlock;
    }

```

## scaleX (DisplacementMapFilter.scaleX 属性)

public scaleX : Number

用于缩放映射计算的 x 置换结果的乘数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的 **MovieClip** filteredMc 时更改其 scaleX 属性。

```

import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.scaleX = 5;
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
    "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
    new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;
}

```

```

        txtBlock.filters = new Array(filter);

        return txtBlock;
    }

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    (hide == true) ? mc._alpha = 0 : mc._alpha = 100;

    var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
        bmp.rectangle, true);
    mc.attachBitmap(bmp, this.getNextHighestDepth());

    return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
        this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
        80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

## scaleY (DisplacementMapFilter.scaleY 属性)

public scaleY : Number

用于缩放映射计算的 y 置换结果的乘数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的 **MovieClip** filteredMc 时更改其 scaleY 属性。

```
import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
import flash.geom.Matrix;
import flash.geom.ColorTransform;

var filteredMc:MovieClip = createDisplacementMapRectangle();

filteredMc.onPress = function() {
    var filter:DisplacementMapFilter = this.filters[0];
    filter.scaleY = 5;
    this.filters = new Array(filter);
}

function createDisplacementMapRectangle():MovieClip {
    var mapBitmap:BitmapData = createGradientBitmap(300, 80, 0xFF000000,
        "radial");
    var filter:DisplacementMapFilter = new DisplacementMapFilter(mapBitmap,
        new Point(-30, -30), 1, 1, 10, 10, "wrap", 0x000000, 0x000000);

    var txtBlock:MovieClip = createTextBlock();
    txtBlock._x = 30;
    txtBlock._y = 30;

    txtBlock.filters = new Array(filter);

    return txtBlock;
}

function createGradientBitmap(w:Number, h:Number, bgColor:Number,
    type:String, hide:Boolean):BitmapData {
    var mc:MovieClip = this.createEmptyMovieClip("mc", 1);
    var matrix:Matrix = new Matrix();
    matrix.createGradientBox(w, h, 0, 0, 0);

    mc.beginGradientFill(type, [0xFF0000, 0x0000FF], [100, 100], [0x55, 0x99],
        matrix, "pad");
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
```



```

mc.lineTo(0, 0);
mc.endFill();
(hide == true) ? mc._alpha = 0 : mc._alpha = 100;

var bmp:BitmapData = new BitmapData(w, h, true, bgColor);
    bmp.draw(mc, new Matrix(), new ColorTransform(), "normal",
    bmp.rectangle, true);
mc.attachBitmap(bmp, this.getNextHighestDepth());

return bmp;
}

function createTextBlock():MovieClip {
    var txtBlock:MovieClip = this.createEmptyMovieClip("txtBlock",
    this.getNextHighestDepth());
    txtBlock.createTextField("txt", this.getNextHighestDepth(), 0, 0, 300,
    80);
    txtBlock.txt.text = "watch the text bend with the displacement map";
    return txtBlock;
}

```

## DropShadowFilter (flash.filters.DropShadowFilter)

```

Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.DropShadowFilter

```

```

public class DropShadowFilter
extends BitmapFilter

```

使用 **DropShadowFilter** 类，可以在 **Flash** 中为各种对象添加投影。投影样式有多个选项，包括内侧或外侧阴影和挖空模式。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时给影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤消设置操作。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter()` 会获得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作时对图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜支持舞台缩放。但不支持常规缩放、旋转和倾斜。如果对象本身进行了缩放（如果 `_xscale` 和 `_yscale` 不是 **100%**），则滤镜效果不进行缩放。只有在放大舞台时，滤镜效果才进行缩放。

如果结果图像的宽度或高度将超过 **2880** 像素，则不应用滤镜。例如，如果在放大某大型影片剪辑时应用了滤镜，则在结果图像超过 **2880** 像素的限制时滤镜将关闭。

可用性: **ActionScript 1.0**；Flash Player **8**

另请参见

[filters \(MovieClip.filters 属性\)](#)，[cacheAsBitmap \(MovieClip.cacheAsBitmap 属性\)](#)，[filters \(Button.filters 属性\)](#)，[cacheAsBitmap \(Button.cacheAsBitmap 属性\)](#)，[filters \(TextField.filters 属性\)](#)，[applyFilter \(BitmapData.applyFilter 方法\)](#)

属性摘要

修饰符	属性	说明
	<code>alpha:Number</code>	阴影颜色的 Alpha 透明度值。
	<code>angle:Number</code>	阴影的角度。
	<code>blurX:Number</code>	水平模糊量。
	<code>blurY:Number</code>	垂直模糊量。
	<code>color:Number</code>	阴影的颜色。
	<code>distance:Number</code>	阴影的偏移距离，以像素为单位。
	<code>hideObject:Boolean</code>	表示是否隐藏对象。
	<code>inner:Boolean</code>	表示阴影是否为内侧阴影。
	<code>knockout:Boolean</code>	应用挖空效果 ( <code>true</code> )，这将有效地使对象的填色变为透明，并显示文档的背景颜色。
	<code>quality:Number</code>	应用滤镜的次数。
	<code>strength:Number</code>	印记或散布的强度。

继承自 **Object** 类的属性

[constructor \(Object.constructor 属性\)](#)，[\\_\\_proto\\_\\_ \(Object.\\_\\_proto\\_\\_ 属性\)](#)，[prototype \(Object.prototype 属性\)](#)，[\\_\\_resolve \(Object.\\_\\_resolve 属性\)](#)

构造函数摘要

签名	说明
<code>DropShadowFilter([distance:Number], [angle:Number], [color:Number], [alpha:Number], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [inner:Boolean], [knockout:Boolean], [hideObject:Boolean])</code>	用指定参数创建新的 DropShadowFilter 实例。

方法摘要

修饰符	签名	说明
	<code>clone() : DropShadowFilter</code>	返回此滤镜对象的副本。

继承自 BitmapFilter 类的方法

[clone \(BitmapFilter.clone 方法\)](#)

继承自 Object 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## alpha (DropShadowFilter.alpha 属性)

public alpha : Number

阴影颜色的 **Alpha** 透明度值。有效值为 0 到 1。例如，.25 设置透明度值为 25%。默认值是 1。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例在用户单击影片剪辑时更改其 alpha 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowAlpha");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.alpha = .4;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## angle (DropShadowFilter.angle 属性)

public angle : Number

阴影的角度。有效值为 **0** 到 **360** 度。默认值是 **45**。

角度值表示理论上的光源落在对象上的角度，它决定了效果相对于该对象的位置。如果距离设置为 **0**，则该效果相对于该对象没有发生偏移，因此 **angle** 属性没有任何效果。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 **angle** 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowAngle");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.angle = 135;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## blurX (DropShadowFilter.blurX 属性)

public blurX : Number

水平模糊量。有效值为 0 到 255 (浮点)。默认值为 4。作为 2 的乘方的值 (如 2、4、8、16 和 32) 经过了优化, 呈现速度比其它值更快。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 blurX 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowBlurX");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.blurX = 40;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## blurY (DropShadowFilter.blurY 属性)

public blurY : Number

垂直模糊量。有效值为 0 到 255 (浮点)。默认值为 4。作为 2 的乘方的值 (如 2、4、8、16 和 32) 经过了优化, 呈现速度比其它值更快。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 blurY 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowBlurY");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.blurY = 40;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## clone (DropShadowFilter.clone 方法)

public clone() : DropShadowFilter

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.filters.DropShadowFilter - 具有原始实例的所有属性的新 **DropShadowFilter** 实例。

### 示例

下面的示例创建三个 **DropShadowFilter** 对象并对它们进行比较: `filter_1` 是通过使用 **DropShadowFilter** 构造函数创建的; `filter_2` 是通过将其设置为等于 `filter_1` 创建的; 而 `clonedFilter` 是通过克隆 `filter_1` 创建的。请注意, 尽管 `filter_2` 等效于 `filter_1`, 但 `clonedFilter` 却不等效于 `filter_1`, 尽管它们包含相同的值。

```
import flash.filters.DropShadowFilter;

var filter_1:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
    16, 16, 1, 3, false, false, false);
var filter_2:DropShadowFilter = filter_1;
var clonedFilter:DropShadowFilter = filter_1.clone();

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> hideObject: false
    // >> strength: 1
    // >> blurY: 16
    // >> blurX: 16
    // >> knockout: false
    // >> inner: false
    // >> quality: 3
    // >> alpha: 0.8
    // >> color: 0
    // >> angle: 45
    // >> distance: 15
}

for(var i in clonedFilter) {
    trace(">> " + i + ": " + clonedFilter[i]);
    // >> clone: [type Function]
    // >> hideObject: false
```



```

// >> strength: 1
// >> blurY: 16
// >> blurX: 16
// >> knockout: false
// >> inner: false
// >> quality: 3
// >> alpha: 0.8
// >> color: 0
// >> angle: 45
// >> distance: 15
}

```

为了进一步说明 `filter_1`、`filter_2` 和 `clonedFilter` 之间的关系，下面的示例将修改 `filter_1` 的 `knockout` 属性。通过修改 `knockout`，说明了 `clone()` 方法是根据 `filter_1` 的值而不是通过在引用中指向这些值来创建新实例的。

```

import flash.filters.DropShadowFilter;

var filter_1:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
    16, 16, 1, 3, false, false, false);
var filter_2:DropShadowFilter = filter_1;
var clonedFilter:DropShadowFilter = filter_1.clone();

trace(filter_1.knockout); // false
trace(filter_2.knockout); // false
trace(clonedFilter.knockout); // false

filter_1.knockout = true;

trace(filter_1.knockout); // true
trace(filter_2.knockout); // true
trace(clonedFilter.knockout); // false

```

## color（DropShadowFilter.color 属性）

public color : Number

阴影的颜色。有效值采用十六进制格式 `0xRRGGBB`。默认值为 `0x000000`。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 `color` 属性。

```

import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowColor");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.color = 0xFF0000;
    this.filters = new Array(filter);
}

```

```

}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}

```

## distance（DropShadowFilter.distance 属性）

public distance : Number

阴影的偏移距离，以像素为单位。默认值为 4（浮点）。

**可用性：** ActionScript 1.0 ； Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 distance 属性。

```

import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowDistance");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.distance = 40;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);

```

```

art.lineTo(0, h);
art.lineTo(0, 0);
art._x = 20;
art._y = 20;

var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
16, 16, 1, 3, false, false, false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

## DropShadowFilter 构造函数

```

public DropShadowFilter([distance:Number], [angle:Number], [color:Number],
    [alpha:Number], [blurX:Number], [blurY:Number], [strength:Number],
    [quality:Number], [inner:Boolean], [knockout:Boolean],
    [hideObject:Boolean])

```

用指定参数创建新的 **DropShadowFilter** 实例。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**distance**:Number [ 可选 ] - 阴影的偏移距离，以像素为单位。默认值为 **4**（浮点）。

**angle**:Number [ 可选 ] - 阴影的角度，**0** 到 **360** 度（浮点）。默认值是 **45**。

**color**:Number [ 可选 ] - 阴影颜色，采用十六进制格式 **0xRRGGBB**。默认值为 **0x000000**。

**alpha**:Number [ 可选 ] - 阴影颜色的 **Alpha** 透明度值。有效值为 **0** 到 **1**。例如，**.25** 设置透明度值为 **25%**。默认值是 **1**。

**blurX**:Number [ 可选 ] - 水平模糊量。有效值为 **0** 到 **255**（浮点）。默认值为 **4**。作为 **2** 的乘方的值（如 **2**、**4**、**8**、**16** 和 **32**）经过了优化，呈现速度比其它值更快。

**blurY**:Number [ 可选 ] - 垂直模糊量。有效值为 **0** 到 **255**（浮点）。默认值为 **4**。作为 **2** 的乘方的值（如 **2**、**4**、**8**、**16** 和 **32**）经过了优化，呈现速度比其它值更快。

**strength**:Number [ 可选 ] - 印记或散布的强度。该值越高，印记的颜色越深，而且阴影与背景之间的对比度也越强。有效值为 **0** 到 **255**。默认值为 **1**。

**quality**:Number [ 可选 ] - 应用滤镜的次数。有效值为 **0** 到 **15**。默认值为 **1**，它表示低品质。值为 **2** 表示中等品质，值为 **3** 表示高品质。

**inner**:Boolean [ 可选 ] - 表示阴影是否为内侧阴影。值 **true** 指定内侧阴影。默认为 **false**，即外侧阴影，它表示对象外缘周围的阴影。

**knockout: Boolean** [ 可选 ] - 应用挖空效果 (true), 这将有效地使对象的填色变为透明, 并显示文档的背景颜色。默认值为 false (不应用挖空效果)。

**hideObject: Boolean** [ 可选 ] - 表示是否隐藏对象。如果值为 true, 则表示没有绘制对象本身, 只有阴影是可见的。默认值为 false (显示对象)。

## 示例

下面的示例实例化一个新的 **DropShadowFilter** 实例, 并将其应用到一个平面矩形。

```
import flash.filters.DropShadowFilter;
var art:MovieClip = createRectangle(100, 100, 0x003366,
    "gradientGlowFilterExample");
var distance:Number = 20;
var angleInDegrees:Number = 45;
var color:Number = 0x000000;
var alpha:Number = .8;
var blurX:Number = 16;
var blurY:Number = 16;
var strength:Number = 1;
var quality:Number = 3;
var inner:Boolean = false;
var knockout:Boolean = false;
var hideObject:Boolean = false;

var filter:DropShadowFilter = new DropShadowFilter(distance,
    angleInDegrees,
    color,
    alpha,
    blurX,
    blurY,
    strength,
    quality,
    inner,
    knockout,
    hideObject);

var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;

function createRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    mc.beginFill(bgColor);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
```

```
mc._x = 20;
mc._y = 20;
return mc;
}
```

## hideObject (DropShadowFilter.hideObject 属性)

public hideObject : Boolean

表示是否隐藏对象。如果值为 true，则表示没有绘制对象本身，只有阴影是可见的。默认值为 false（显示对象）。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 hideObject 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowHideObject");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.hideObject = true;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## inner (DropShadowFilter.inner 属性)

public inner : Boolean

表示阴影是否为内侧阴影。值为 true 表明是内侧阴影。默认为 false，即外侧阴影，它表示对象外缘周围的阴影。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 inner 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowInner");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.inner = true;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## knockout (DropShadowFilter.knockout 属性)

public knockout : Boolean

应用挖空效果 (true)，这将有效地使对象的填色变为透明，并显示文档的背景颜色。默认值为 false（不应用挖空效果）。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 knockout 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowKnockout");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.knockout = true;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## quality (DropShadowFilter.quality 属性)

public quality : Number

应用滤镜的次数。有效值为 **0** 到 **15**。默认值为 **1**，它表示低品质。值为 **2** 表示中等品质，值为 **3** 表示高品质。滤镜的值越小，呈现速度越快。

对于大多数应用，quality 的值为 **1**、**2** 或 **3** 就足够了。您可以使用其它数值（最高为 **15**）来达到不同的效果，但是值越高，呈现速度越慢。除了增加 quality 的值，增加 blurX 和 blurY 的值通常也可以获得类似的效果，而且呈现速度更快。

**可用性:** **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 quality 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowQuality");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.quality = 0;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```



## strength (DropShadowFilter.strength 属性)

public strength : Number

印记或散布的强度。该值越高，印记的颜色越深，而且阴影与背景之间的对比度也越强。有效值为 0 到 255。默认值为 1。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

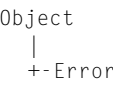
下面的示例在用户单击现有的影片剪辑时更改其 strength 属性。

```
import flash.filters.DropShadowFilter;
var mc:MovieClip = createDropShadowRectangle("DropShadowStrength");
mc.onRelease = function() {
    var filter:DropShadowFilter = this.filters[0];
    filter.strength = .6;
    this.filters = new Array(filter);
}

function createDropShadowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var filter:DropShadowFilter = new DropShadowFilter(15, 45, 0x000000, .8,
        16, 16, 1, 3, false, false, false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

# Error



```
public class Error
extends Object
```

包含关于脚本中出现的错误的信息。您可以使用 `Error` 构造函数来创建 **Error** 对象。通常，新的 **Error** 对象从 `try` 代码块中 `throw`，然后由 `catch` 或 `finally` 代码块捕获。

您也可以创建 **Error** 类的子类，然后引发该子类的实例。

可用性：ActionScript 1.0 ； Flash Player 7

## 属性摘要

修饰符	属性	说明
	<code>message:String</code>	包含与 <code>Error</code> 对象关联的消息。
	<code>name:String</code>	包含 <code>Error</code> 对象的名称。

继承自 `Object` 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 构造函数摘要

签名	说明
<code>Error([message:String])</code>	创建新的 <code>Error</code> 对象。

## 方法摘要

修饰符	签名	说明
	<code>toString() : String</code>	在默认情况下返回字符串 “Error” ； 如果已定义，则返回 <code>Error.message</code> 中包含的值。

继承自 `Object` 类的方法

```
addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法),
toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)
```

## Error 构造函数

```
public Error([message:String])
```

创建新的 **Error** 对象。如果指定了 **message**，它的值将分配给对象的 **Error.message** 属性。

可用性：ActionScript 1.0；Flash Player 7

### 参数

**message**:String [ 可选 ] - 与 **Error** 对象相关联的字符串。

### 示例

在下面的示例中，如果传递给函数的两个字符串不相同，该函数将引发错误（并显示指定的消息）：

```
function compareStrings(str1_str:String, str2_str:String):Void {
    if (str1_str != str2_str) {
        throw new Error("Strings do not match.");
    }
}
try {
    compareStrings("Dog", "dog");
    // output: Strings do not match.
} catch (e_err:Error) {
    trace(e_err.toString());
}
```

### 另请参见

[throw 语句](#)，[try..catch..finally 语句](#)

## message（Error.message 属性）

```
public message : String
```

包含与 **Error** 对象关联的消息。默认情况下，此属性的值为 “Error”。当通过将错误字符串传递给 **Error** 构造函数来创建 **Error** 对象时，可以指定 **message** 属性。

可用性：ActionScript 1.0；Flash Player 7

### 示例

在下面的示例中，函数根据在 **theNum** 中输入的参数引发指定的消息。如果可以除两个数字，则显示 **SUCCESS** 和数字。如果您尝试除以 **0** 或者只输入 **1** 个参数，则会出现特定的错误：

```
function divideNum(num1:Number, num2:Number):Number {
    if (isNaN(num1) || isNaN(num2)) {
        throw new Error("divideNum function requires two numeric parameters.");
    } else if (num2 == 0) {
```

```

        throw new Error("cannot divide by zero.");
    }
    return num1/num2;
}
try {
    var theNum:Number = divideNum(1, 0);
    trace("SUCCESS! "+theNum);
} catch (e_err:Error) {
    trace("ERROR! "+e_err.message);
    trace("\t"+e_err.name);
}

```

如果在不对所除的数字做任何修改的情况下测试此 **ActionScript**，则“输出”面板中会显示一条错误，原因是您试图除以 0。

另请参见

[throw 语句](#)，[try..catch..finally 语句](#)

## name (Error.name 属性)

public name : String

包含 **Error** 对象的名称。默认情况下，此属性的值为 “Error”。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 示例

在下面的示例中，一个函数根据您试图除的两个数字引发指定的错误。将下面的 **ActionScript** 添加到时间轴的第 1 帧：

```

function divideNumber(numerator:Number, denominator:Number):Number {
    if (isNaN(numerator) || isNaN(denominator)) {
        throw new Error("divideNum function requires two numeric parameters.");
    } else if (denominator == 0) {
        throw new DivideByZeroError();
    }
    return numerator/denominator;
}
try {
    var theNum:Number = divideNumber(1, 0);
    trace("SUCCESS! "+theNum);
    // output: DivideByZeroError -> Unable to divide by zero.
} catch (e_err:DivideByZeroError) {
    // divide by zero error occurred
    trace(e_err.name+" -> "+e_err.toString());
} catch (e_err:Error) {
    // generic error occurred
    trace(e_err.name+" -> "+e_err.toString());
}

```

要添加自定义错误，请将以下代码添加到名为 `DivideByZeroError.as` 的 `.AS` 文件，并将该类文件保存在 `FLA` 文档所在的同一目录中。

```
class DivideByZeroError extends Error {  
    var name:String = "DivideByZeroError";  
    var message:String = "Unable to divide by zero.";  
}
```

另请参见

[throw 语句](#), [try..catch..finally 语句](#)

## toString（Error.toString 方法）

```
public toString() : String
```

在默认情况下返回字符串 **“Error”**；如果已定义，则返回 `Error.message` 中包含的值。

可用性：ActionScript 1.0；Flash Player 7

### 返回

String - 一个字符串

### 示例

在下面的示例中，如果传递给函数的两个字符串不相同，该函数将引发错误（并显示指定的消息）：

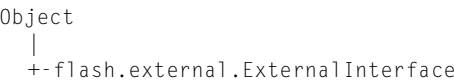
```
function compareStrings(str1_str:String, str2_str:String):Void {  
    if (str1_str != str2_str) {  
        throw new Error("Strings do not match.");  
    }  
}  
  
try {  
    compareStrings("Dog", "dog");  
    // output: Strings do not match.  
} catch (e_err:Error) {  
    trace(e_err.toString());  
}
```

另请参见

[message（Error.message 属性）](#), [throw 语句](#), [try..catch..finally 语句](#)

# ExternalInterface

## (flash.external.ExternalInterface)



```
public class ExternalInterface
extends Object
```

**ExternalInterface** 类是外部 API（在 **ActionScript** 和 **Flash Player** 的容器之间实现直接通讯的应用程序编程接口）；例如，含有 **JavaScript** 的 **HTML** 页或嵌入 **Flash Player** 的桌面应用程序。

**ExternalInterface** 在功能上与 `fscommand()`、`CallFrame()` 和 `CallLabel()` 方法相似，但它更灵活、更通用。推荐对 **JavaScript** 和 **ActionScript** 之间的通讯使用 **ExternalInterface**。从 **ActionScript** 中，您可以在 **HTML** 页上调用任何 **JavaScript** 功能，传递任何数据类型的任意数量的参数，并接收来自该调用的返回值。

从 **HTML** 页上的 **JavaScript** 中，可以调用 **Flash Player** 中的 **ActionScript** 函数。**ActionScript** 函数可以返回一个值，**JavaScript** 会立即接收它作为该调用的返回值。以下浏览器和操作系统的组合支持 **ExternalInterface**：

浏览器	操作系统	
Internet Explorer 5.0 及更高版本	Windows	
Netscape 8.0 及更高版本	Windows	Macintosh
Mozilla 1.7.5 及更高版本	Windows	Macintosh
Firefox 1.0 及更高版本	Windows	Macintosh
Safari 1.3 及更高版本		Macintosh

**ExternalInterface** 要求用户的 **Web** 浏览器支持 **ActiveX** 或由某些浏览器公开的 **NPRuntime API** 以实现插件脚本处理。请访问 <http://www.mozilla.org/projects/plugins/npruntime.html>。

可用性： **ActionScript 1.0** ； **Flash Player 8**

属性摘要

修饰符	属性	说明
static	available:Boolean[ 只读 ]	指示此播放器是否位于提供外部接口的容器中。

继承自 Object 类的属性

<a href="#">constructor</a> (Object.constructor 属性), <a href="#">__proto__</a> (Object.__proto__ 属性), <a href="#">prototype</a> (Object.prototype 属性), <a href="#">__resolve</a> (Object.__resolve 属性)
--

方法摘要

修饰符	签名	说明
static	<code>addCallback(methodName:String, instance:Object, method:Function) : Boolean</code>	将 <b>ActionScript</b> 方法注册为可从容器调用。
static	<code>call(methodName:String, [parameter1:Object]) : Object</code>	调用由 <b>Flash Player</b> 容器公开的函数，传递零个或多个参数。

继承自 Object 类的方法

<a href="#">addProperty</a> (Object.addProperty 方法), <a href="#">hasOwnProperty</a> (Object.hasOwnProperty 方法), <a href="#">isPropertyEnumerable</a> (Object.isPropertyEnumerable 方法), <a href="#">isPrototypeOf</a> (Object.isPrototypeOf 方法), <a href="#">registerClass</a> (Object.registerClass 方法), <a href="#">toString</a> (Object.toString 方法), <a href="#">unwatch</a> (Object.unwatch 方法), <a href="#">valueOf</a> (Object.valueOf 方法), <a href="#">watch</a> (Object.watch 方法)
---

addCallback（ExternalInterface.addCallback 方法）

```
public static addCallback(methodName:String, instance:Object,
    method:Function) : Boolean
```

将 **ActionScript** 方法注册为可从容器调用。成功调用 `addCallback()` 后，容器中的 **JavaScript** 或 **ActiveX** 代码可以调用在 **Flash Player** 中注册的函数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

参数

**methodName:String** - 从 **JavaScript** 调用 **ActionScript** 函数时可使用的名称。此名称不必与 **ActionScript** 方法的实际名称匹配。

**instance:Object** - `this` 在该方法中被解析成的对象。此对象不一定是在其上可找到该方法的对象，您可以指定任何对象（或 `null`）。

**method:Function** - 要从 JavaScript 调用的 ActionScript 方法。

## 返回

**true** - 如果调用成功，则返回 Boolean。如果调用由于下列原因而失败，则返回 **false**：实例不可用、遇到了安全限制、没有这种函数对象、发生了递归或类似情况。

返回值为 **false** 还可能表示包含环境属于调用代码无权访问的安全沙箱。您可以在包含环境的 **HTML** 中为 **allowScriptAccess OBJECT** 标签或 **EMBED** 标签设置一个合适的值，以解决此问题。

## 示例

下面的示例将 **goToMacromedia()** 函数注册为可使用名称 **goHome** 从容器调用。

```
import flash.external.*;

var methodName:String = "goHome";
var instance:Object = null;
var method:Function = goToMacromedia;
var wasSuccessful:Boolean = ExternalInterface.addCallback(methodName,
    instance, method);

var txtField:TextField = this.createTextField("txtField",
    this.getNextHighestDepth(), 0, 0, 200, 50);
txtField.border = true;
txtField.text = wasSuccessful.toString();

function goToMacromedia() {
    txtField.text = "http://www.macromedia.com";
    getURL("http://www.macromedia.com", "_self");
}
```

为了使上一示例能够正常运行，应复制以下代码并将其粘贴到包含 **HTML** 页中。此代码依赖 **OBJECT** 标签的 **id** 属性和 **EMBED** 标签的 **name** 属性以获得值 **externalInterfaceExample**。由于 **Internet Explorer** 和 **Netscape** 以不同方式引用 **movie** 对象，所以函数 **thisMovie** 根据浏览器返回相应的语法。除非服务器上承载 **HTML** 页，否则您的浏览器可能会出现安全警告。



请避免使用访问插件对象的其它方法（如 **document.getElementById("pluginName")** 或 **document.all.pluginName**），因为这些其它方法在所有浏览器上的运行不一致。

```
<form>
    <input type="button" onclick="callExternalInterface()" value="Call
        ExternalInterface" />
</form>
<script>
function callExternalInterface() {
    thisMovie("externalInterfaceExample").goHome();
}
```



```

}

function thisMovie(movieName) {
    if (navigator.appName.indexOf("Microsoft") != -1) {
        return window[movieName]
    }
    else {
        return document[movieName]
    }
}
</script>

```

## available（ExternalInterface.available 属性）

public static available : Boolean [read-only]

指示此播放器是否位于提供外部接口的容器中。如果外部接口可用，则此属性为 true；否则，它为 false。

可用性: ActionScript 1.0；Flash Player 8

### 示例

下面的示例使用 ExternalInterface.available 来确定播放器是否位于提供外部接口的容器中。

```

import flash.external.*;

var isAvailable:Boolean = ExternalInterface.available;
trace(isAvailable);

```

## call（ExternalInterface.call 方法）

public static call(methodName:String, [parameter1:Object]) : Object

调用由 Flash Player 容器公开的函数，传递零个或多个参数。如果所需的函数不可用，则调用返回 null；否则，它返回由该函数提供的值。不允许使用递归；递归调用会生成 null 响应。

如果容器是 HTML 页，则此方法在 <script> 元素中调用 JavaScript 函数。

如果该容器是其它某个 ActiveX 容器，则此方法会使用指定的名称广播事件，并且该容器会处理该事件。

如果该容器承载 Netscape 插件，您可以写入对新 NPRuntime 接口的自定义支持或嵌入 HTML 控件以及在 HTML 控件内嵌入 Flash Player。如果嵌入 HTML 控件，就可以通过与本机容器应用程序通讯的 JavaScript 接口与 Flash Player 进行通讯。

可用性: ActionScript 1.0；Flash Player 8

## 参数

**methodName:String** - 要在容器中调用的函数的名称。如果该函数接受参数，则这些参数必须显示在 **methodName** 参数后面。

**parameter1:Object** [ 可选 ] - 要传递到该函数的任何参数。您可以指定零个或多个参数，参数之间用逗号分隔。此参数可以是任何 **ActionScript** 数据类型。当调用 **JavaScript** 函数时，**ActionScript** 类型自动封装到 **JavaScript** 类型中。当调用某个其它 **ActiveX** 容器时，将在请求消息中对参数进行编码。

## 返回

**Object** - 从容器接收的响应。如果调用失败（例如，当容器中没有这种函数时、接口不可用时、发生递归时，或出现安全问题时），则返回 **null**。

## 示例

下面的示例在包含 **SWF** 的 **HTML** 页中调用 **JavaScript** 函数 **sayHello()**。该调用是通过使用 **ExternalInterface.call()** 方法实现的。

```
import flash.external.*;

var greeting:String;
var btn:MovieClip = createButton(100, 30, 0xCCCCCC);
btn.onPress = function() {
    greeting = String(ExternalInterface.call("sayHello", "browser"));
    this.mcTxt.text = greeting; // >> Hi Flash.
}

function createButton(width:Number, height:Number, color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    var mcFmt:TextFormat;

    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);

    mcFmt = new TextFormat();
    mcFmt.align = "center";
    mcFmt.bold = true;

    mc.createTextField("mcTxt", depth, 0, 0, width, height);
    mc.mcTxt.text = "Call JS Function";
    mc.mcTxt.setTextFormat(mcFmt);

    return mc;
}
```

为了使上一示例能够正常运行，应复制以下代码并将其粘贴到包含 **HTML** 页中。除非服务器上承载 **HTML** 页，否则您的浏览器可能会出现安全警告。

```
<script>
    function sayHello(name) {
        alert(">> Hello " + name + ".");
        return ">> Hi Flash.";
    }
</script>
```

## FileReference (flash.net.FileReference)

```
Object
|
+- flash.net.FileReference
```

```
public class FileReference
extends Object
```

**FileReference** 类提供了在用户计算机和服务器之间上载和下载文件的方法。操作系统对话框会提示用户选择要上载的文件或用于下载的位置。每个 **FileReference** 对象引用用户硬盘上的一个文件并且具有一些属性，这些属性包含有关文件大小、类型、名称、创建日期、修改日期和创建者类型（仅限 **Macintosh**）的信息。

**FileReference** 实例的创建方法有两种：

- 当将 **new** 运算符与 **FileReference** 构造函数一起使用时：`var myFileReference = new FileReference();`
- 当调用 `FileReferenceList.browse()` 时（创建 **FileReference** 对象的数组）

在上载操作期间，**FileReference** 对象的所有属性通过调用 `FileReference.browse()` 或 `FileReferenceList.browse()` 来填充。在下载操作期间，在调用 `onSelect` 之后填充 `name` 属性，在调用 `onComplete` 之后填充所有其它属性。

`browse()` 方法打开一个操作系统对话框，提示用户选择要上载的任何本地文件。

`FileReference.browse()` 方法允许用户选择一个文件；`FileReferenceList.browse()` 方法允许用户选择多个文件。成功调用 `browse()` 方法后，调用 `FileReference.upload()` 方法来每次上载一个文件。`FileReference.download()` 方法提示用户提供文件的保存位置并开始从远程 **URL** 进行下载。

**FileReference** 类和 **FileReferenceList** 类不允许为由 `browse()` 和 `download()` 调用生成的对话框设置默认文件位置。对话框中显示的默认位置是最近浏览过的文件夹（如果可以确定该位置）或桌面。这些类不允许对已传输的文件进行读取或写入。它们也不允许启动上载或下载的 **SWF** 文件访问已上载或下载的文件或用户磁盘上文件的位置。

**FileReference** 类和 **FileReferenceList** 类也不提供用于身份验证的方法。通过要求身份验证的服务器，您可以使用 **Flash Player** 浏览器插件下载文件，但上载（在所有播放器上）和下载（在独立播放器或外部播放器上）将失败。使用 **FileReference** 事件侦听器可确定操作是否成功完成并可以处理错误。

对于上载和下载操作，**SWF** 文件只能访问自己的域（包括由跨域策略文件指定的任何域）内的文件。如果启动上载或下载的 **SWF** 与文件服务器不在相同的域中，则必须将策略文件放到文件服务器上。

在执行对 **FileReference.browse()**、**FileReferenceList.browse()** 或 **FileReference.download()** 方法的调用时，**SWF** 文件播放在以下平台上将暂停：用于 **Mac OS X** 的 **Flash Player** 插件、用于 **Macintosh** 的外部 **Flash Player** 和用于 **Mac OS X 10.1** 及更低版本的独立播放器。在适用于 **Windows** 的所有播放器和适用于 **Mac OS X 10.2** 及更高版本的 **Macintosh** 的独立播放器中，**SWF** 文件继续运行。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例创建一个 **FileReference** 对象，提示用户选择要上载的图像或文本文件。它还侦听可能发生的任何事件。

```
import flash.net.FileReference;

var allTypes:Array = new Array();
var imageTypes:Object = new Object();
imageTypes.description = "Images (*.jpg, *.jpeg, *.gif, *.png)";
imageTypes.extension = "*.jpg; *.jpeg; *.gif; *.png";
allTypes.push(imageTypes);

var textTypes:Object = new Object();
textTypes.description = "Text Files (*.txt, *.rtf)";
textTypes.extension = "*.txt;*.rtf";
allTypes.push(textTypes);

var listener:Object = new Object();

listener.onSelect = function(file:FileReference):Void {
    trace("onSelect: " + file.name);
    if(!file.upload("http://www.yourdomain.com/yourUploadHandlerScript.cfm"))
    {
        trace("Upload dialog failed to open.");
    }
}

listener.onCancel = function(file:FileReference):Void {
    trace("onCancel");
}

listener.onOpen = function(file:FileReference):Void {
    trace("onOpen: " + file.name);
}
```

```
listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
    bytesTotal);
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

listener.onHTTPError = function(file:FileReference):Void {
    trace("onHTTPError: " + file.name);
}

listener.onIOError = function(file:FileReference):Void {
    trace("onIOError: " + file.name);
}

listener.onSecurityError = function(file:FileReference,
    errorString:String):Void {
    trace("onSecurityError: " + file.name + " errorString: " + errorString);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse(allTypes);
```

另请参见  
[FileReferenceList \(flash.net.FileReferenceList\)](#)

属性摘要

修饰符	属性	说明
	creationDate:Date [ 只读 ]	本地磁盘上文件的创建日期。
	creator:String [ 只读 ]	文件的 Macintosh 创建者类型。
	modificationDate:Date [ 只读 ]	本地磁盘上文件的上一次修改日期。
	name:String [ 只读 ]	本地磁盘上的文件的名称。
	size:Number [ 只读 ]	本地磁盘上文件的大小，以字节表示。
	type:String [ 只读 ]	文件类型。

继承自 Object 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

事件摘要

事件	说明
<code>onCancel = function(fileRef:FileReference) {}</code>	当用户取消文件浏览对话框时调用。
<code>onComplete = function(fileRef:FileReference) {}</code>	当上载或下载操作成功完成时调用。
<code>onHTTPError = function(fileRef:FileReference, httpError:Number) {}</code>	当上载由于 HTTP 错误而失败时调用。
<code>onIOError = function(fileRef:FileReference) {}</code>	当发生输入 / 输出错误时调用。
<code>onOpen = function(fileRef:FileReference) {}</code>	当上载或下载操作开始时调用。
<code>onProgress = function(fileRef:FileReference, bytesLoaded:Number, bytesTotal:Number) {}</code>	在文件上载或下载操作期间定期调用。
<code>onSecurityError = function(fileRef:FileReference, errorString:String) {}</code>	当上载或下载由于安全错误而失败时调用。
<code>onSelect = function(fileRef:FileReference) {}</code>	当用户从文件浏览对话框选择要上载或下载的文件时调用。

构造函数摘要

签名	说明
<code>FileReference()</code>	创建新的 <code>FileReference</code> 对象。

方法摘要

修饰符	签名	说明
	<code>addListener(listener:Object) : Void</code>	注册一个对象，以便在调用 <code>FileReference</code> 事件侦听器时接收通知。
	<code>browse([typeList:Array]) : Boolean</code>	显示文件浏览对话框，用户可以从中选择要上载的本地文件。
	<code>cancel() : Void</code>	取消正在对该 <code>FileReference</code> 对象执行的任何上载或下载操作。
	<code>download(url:String, [defaultFileName:String]) : Boolean</code>	显示对话框，用户可以在其中下载远程服务器上的文件。
	<code>removeListener(listener:Object) : Boolean</code>	从接收事件通知消息的对象列表中删除对象。
	<code>upload(url:String) : Boolean</code>	开始将用户选择的文件上载到远程服务器。

继承自 Object 类的方法

---

addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)

---

## addListener (FileReference.addListener 方法)

```
public addListener(listener:Object) : Void
```

注册一个对象，以便在调用 **FileReference** 事件侦听器时接收通知。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**listener:Object** - 一个对象，它侦听从 **FileReference** 事件侦听器发出的回调通知。

### 示例

下面的示例将侦听器添加到 **FileReference** 的实例中。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
        bytesTotal);
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
fileRef.download(url, "FlashPlatform.pdf");
```

## browse (FileReference.browse 方法)

`public browse([typelist:Array]) : Boolean`

显示文件浏览对话框，用户可以选择要上载的本地文件。该对话框对于用户的操作系统来说是本机的。当调用此方法并且用户成功选择文件时，会使用该文件的属性填充此 **FileReference** 对象的属性。后续每次调用 `FileReference.browse()` 时，**FileReference** 对象的属性都重置为用户在对话框中选择的文件。

每次只能执行一个 `browse()` 或 `download()` 会话（因为每次只能显示一个对话框）。

可以传递一个文件类型数组以确定对话框显示的文件。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 参数

**typelist:Array** [ 可选 ] - 一个文件类型数组，用于过滤对话框中显示的文件。如果省略此参数，则显示所有文件。如果包括此参数，数组必须包含用花括号 { } 括起来的一个或多个元素。数组可采用以下两种格式：

- 后面只跟 **Windows** 文件扩展名的文件类型描述列表。数组中的每个元素必须包含描述文件类型的字符串和用分号分隔的 **Windows** 文件扩展名的列表（在每个扩展名前有一个通配符 (\*)）。每个元素的语法如下所示：

```
[{description: "string describing the first set of file types", extension: "semicolon-delimited list of file extensions"}]
```

示例：

```
[{description: "Images", extension: "*.jpg;*.gif;*.png"}, {description: "Flash Movies", extension: "*.swf"}, {description: "Documents", extension: "*.doc;*.pdf"}]
```

- 后跟其 **Windows** 文件扩展名和 **Macintosh** 文件类型的文件类型描述列表。数组中的每个元素必须包含：描述文件类型的字符串；用分号分隔的 **Windows** 文件扩展名的列表（每个扩展名前都有一个通配符 (\*)）；用分号分隔的 **Macintosh** 文件类型的列表（每个类型前都有一个通配符 (\*)）。每个元素的语法如下所示：

```
[{description: "string describing the first set of file types", extension: "semicolon-delimited list of Windows file extensions", macType: "semicolon-delimited list of Macintosh file types"}]
```

示例：

```
[{description: "Image files", extension: "*.jpg;*.gif;*.png", macType: "JPEG;jp2_;GIF"}, {description: "Flash Movies", extension: "*.swf", macType: "SWFL"}]
```



这两种格式不能在一个 `browse()` 调用中互换。必须使用其中一种格式。

扩展名列表在 **Windows** 上用于过滤文件，具体取决于用户选择的文件。实际上它不显示在对话框中。要对用户显示文件类型，必须在描述字符串以及扩展名列表中列出文件类型。在 **Windows** 上，描述字符串显示在对话框中。（在 **Macintosh** 上不使用它。）在 **Macintosh** 上，如果提供 **Macintosh** 文件类型的列表，则该列表用于过滤文件。如果没有提供 **Macintosh** 文件类型的列表，则使用 **Windows** 扩展名的列表。

## 返回

Boolean - 如果参数有效并且显示文件浏览对话框，则返回 `true`。以下情况下返回 `false`：未显示对话框；正在进行另一个浏览器会话；使用 `typelist` 参数，但未能在数组的任一元素中提供描述或扩展名字符串。

## 示例

下面的示例显示一个对话框，用户可以从中选择要上传的图像文件。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(file:FileReference):Void {
    trace("Opened " + file.name);
}

listener.onCancel = function(file:FileReference):Void {
    trace("User cancelled");
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

## 另请参见

[onSelect \(FileReferenceList.onSelect 事件侦听器\)](#), [onCancel \(FileReference.onCancel 事件侦听器\)](#), [download \(FileReference.download 方法\)](#), [browse \(FileReferenceList.browse 方法\)](#)

## cancel (FileReference.cancel 方法)

```
public cancel() : Void
```

取消正在对该 **FileReference** 对象执行的任何上载或下载操作。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将所请求的文件下载大约一半, 然后取消下载。这显然不是一种典型用法。当您允许用户在下载状态对话框中单击“取消”时, 可能更常使用此方法。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
        bytesTotal);
    if(bytesLoaded >= (bytesTotal / 2)) {
        file.cancel();
    }
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
fileRef.download(url, "FlashPlatform.pdf");
```

## creationDate (FileReference.creationDate 属性)

```
public creationDate : Date [read-only]
```

本地磁盘上文件的创建日期。如果尚未填充 **FileReference** 对象, 为获得此属性的值而执行的调用将返回 null。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例检索用户选择的文件的创建日期。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(file:FileReference):Void {
    trace("creationDate: " + file.creationDate);
}
```

```
var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## creator (FileReference.creator 属性)

```
public creator : String [read-only]
```

文件的 **Macintosh** 创建者类型。在 **Windows** 上，此属性为 `null`。如果尚未填充 **FileReference** 对象，为获得此属性的值而执行的调用将返回 `null`。

可用性: **ActionScript 1.0** ; **Flash Player 8**

示例

下面的示例检索用户选择的文件的 **Macintosh** 创建者类型。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(file:FileReference):Void {
    trace("creator: " + file.creator);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## download (FileReference.download 方法)

```
public download(url:String, [defaultFileName:String]) : Boolean
```

显示对话框，用户可以在其中下载远程服务器上的文件。**Flash Player** 可以下载最多 **100 MB** 的文件。

此方法首先打开一个操作系统对话框，让用户输入文件名并选择本地计算机上用于保存文件的位置。当用户选择了位置并确认下载操作时（例如，单击“保存”），则开始从远程服务器下载。侦听器接收事件来指示下载的进度、成功或失败。为了在调用 `download()` 后确定对话框和下载操作的状态，**ActionScript** 代码必须使用事件侦听器（如 `onCancel`、`onOpen`、`onProgress` 和 `onComplete`）来侦听事件。

一旦成功下载了文件，便使用本地文件的属性填充 **FileReference** 对象的属性，并调用 `onComplete` 侦听器。

每次只能执行一个 `browse()` 或 `download()` 会话（因为每次只能显示一个对话框）。

此方法支持使用 **HTTP** 或 **HTTPS** 下载任何文件类型。您也可以通过将参数追加到 **URL**（用于进行服务器脚本分析）使用 `download()` 调用将数据发送至服务器。

提醒

如果服务器要求用户身份验证，则只有在浏览器中运行（即使用浏览器插件或 **ActiveX** 控件）的 **SWF** 文件才可以提供对话框来提示用户输入用户名和密码以进行身份验证，而且只适用于下载。对于使用插件或 **ActiveX** 控件进行的上载以及使用独立或外部播放器进行的上载和下载，文件传输会失败。

使用此方法时，请考虑 **Flash Player** 安全模型：

- 如果执行调用的 **SWF** 文件在不受信任的本地沙箱中，则不允许使用此方法。
- 默认值为拒绝在沙箱之间进行访问。网站可以通过添加跨域策略文件来实现对资源的访问。

有关更多信息，请参见以下部分：

- 《学习 **Flash** 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 **API** 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：**ActionScript 1.0**；**Flash Player 8**

## 参数

**url:String** - 要下载到本地计算机的文件的 **URL**。您可以通过将参数追加到 **URL**（用于进行服务器脚本分析）使用 `download()` 调用将数据发送至服务器。例如：<http://www.myserver.com/picture.jpg?userID=jdoe>

在某些浏览器上，**URL** 字符串长度受限。在某些浏览器或服务器上，长度超过 256 个字符的字符串可能失败。

**defaultFileName:String** [ 可选 ] - 对话框中显示的要下载的文件默认文件名。此字符串不能包含以下字符：`/ \ : * ? " < > | %`

如果省略此参数，则会分析出远程 **URL** 的文件名并用作默认文件名。

## 返回

**Boolean** - 如果显示了用户可从中选择文件的对话框，则值为 `true`。如果未显示对话框，则该方法返回 `false`。由于下列任一原因，对话框可能无法显示：

- 未传递 **url** 参数的值。
- 所传递参数的类型或格式不正确。
- **url** 参数的长度为零。

- 发生了安全侵犯；即，您的 SWF 文件试图访问位于 SWF 文件的安全沙箱以外的服务器上的文件。
- 另一个浏览会话已经在进行中。可以通过 `FileReference.browse()`、`FileReferenceList.browse()` 或 `FileReference.download()` 启动浏览器会话。
- 协议不是 HTTP 或 HTTPS。

## 示例

下面的示例尝试使用 `download` 方法下载文件。请注意，对于所有事件都有侦听器。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onSelect = function(file:FileReference):Void {
    trace("onSelect: " + file.name);
}

listener.onCancel = function(file:FileReference):Void {
    trace("onCancel");
}

listener.onOpen = function(file:FileReference):Void {
    trace("onOpen: " + file.name);
}

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
    bytesTotal);
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

listener.onIOError = function(file:FileReference):Void {
    trace("onIOError: " + file.name);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
platform_overview.pdf";
if(!fileRef.download(url, "FlashPlatform.pdf")) {
    trace("dialog box failed to open.");
}
```

另请参见

[browse \(FileReference.browse 方法\)](#) , [browse \(FileReferenceList.browse 方法\)](#) ,  
[upload \(FileReference.upload 方法\)](#)

## FileReference 构造函数

```
public FileReference()
```

创建新的 **FileReference** 对象。在填充后，**FileReference** 对象表示用户本地磁盘上的文件。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建一个新的 **FileReference** 对象，并启动 PDF 文件下载。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onComplete = function(file:FileReference) {
    trace("onComplete : " + file.name);
}

var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.download(url, "FlashPlatform.pdf");
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## modificationDate (FileReference.modificationDate 属性)

```
public modificationDate : Date [read-only]
```

本地磁盘上文件的上一次修改日期。如果尚未填充 **FileReference** 对象，为获得此属性的值而执行的调用将返回 null。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例检索用户选择的文件的 `modificationDate`。

```
import flash.net.FileReference;

var listener:Object = new Object();
```

```
listener.onSelect = function(file:FileReference):Void {  
    trace("modificationDate: " + file.modificationDate);  
}  
  
var fileRef:FileReference = new FileReference();  
fileRef.addListener(listener);  
fileRef.browse();
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## name (FileReference.name 属性)

```
public name : String [read-only]
```

本地磁盘上的文件的名称。如果尚未填充 **FileReference** 对象，为获得此属性的值而执行的调用将返回 null。

**FileReference** 对象的所有属性都通过调用 `browse()` 来填充。与其它 **FileReference** 属性不同，如果调用 `download()`，则在调用 `onSelect` 时将填充 `name` 属性。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例检索用户选择的文件的名称。

```
import flash.net.FileReference;  
  
var listener:Object = new Object();  
listener.onSelect = function(file:FileReference):Void {  
    trace("name: " + file.name);  
}  
  
var fileRef:FileReference = new FileReference();  
fileRef.addListener(listener);  
fileRef.browse();
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## onCancel (FileReference.onCancel 事件侦听器)

```
onCancel = function(fileRef:FileReference) {}
```

当用户取消文件浏览对话框时调用。当调用 `FileReference.browse()`、`FileReferenceList.browse()` 或 `FileReference.download()` 时，将显示此对话框。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**fileRef:** `flash.net.FileReference` - 启动操作的 **FileReference** 对象。

### 示例

下面的示例在用户取消文件浏览对话框时跟踪消息。此方法仅在显示对话框后用户单击“取消”或按 **Esc** 键时触发。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onCancel = function(file:FileReference):Void {
    trace("onCancel");
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
platform_overview.pdf";
if(!fileRef.download(url, "FlashPlatform.pdf")) {
    trace("dialog box failed to open.");
}
```



## onComplete (FileReference.onComplete 事件侦听器)

```
onComplete = function(fileRef:FileReference) {}
```

当上载或下载操作成功完成时调用。成功完成表示已上载或下载了整个文件。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**fileRef:**flash.net.FileReference - 启动操作的 FileReference 对象。

### 示例

下面的示例在触发 onComplete 事件时输出消息。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
fileRef.download(url, "FlashPlatform.pdf");
```

## onHTTPError (FileReference.onHTTPError 事件侦听器)

```
onHTTPError = function(fileRef:FileReference, httpError:Number) {}
```

当上载由于 HTTP 错误而失败时调用。

由于 Flash Player 在文件下载期间依赖浏览器堆栈的方式，此错误不是下载失败的原因。如果下载由于 HTTP 错误而失败，则会将错误报告为 I/O 错误。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**fileRef:**flash.net.FileReference - 启动操作的 File Reference 对象。

**httpError:**Number - 导致此上载失败的 HTTP 错误。例如，404 httpError 表示找不到某页。在 [ftp://ftp.isi.edu/in-notes/rfc2616.txt](http://ftp.isi.edu/in-notes/rfc2616.txt) 处的 HTTP 规范的 10.4 和 10.5 节中，可以找到 HTTP 错误值。

## 示例

下面的示例为每个可能的包括 `onHttpError` 的事件创建一个带有侦听器的 `FileReference` 对象。此侦听器仅在上载由于 **HTTP** 错误而失败时触发。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onSelect = function(file:FileReference):Void {
    trace("onSelect: " + file.name);
    if(!file.upload("http://www.yourdomain.com/yourUploadHandlerScript.cfm"))
    {
        trace("Upload dialog failed to open.");
    }
}

listener.onCancel = function(file:FileReference):Void {
    trace("onCancel");
}

listener.onOpen = function(file:FileReference):Void {
    trace("onOpen: " + file.name);
}

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
        bytesTotal);
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

listener.onHTTPError = function(file:FileReference):Void {
    trace("onHTTPError: " + file.name);
}

listener.onIOError = function(file:FileReference):Void {
    trace("onIOError: " + file.name);
}

listener.onSecurityError = function(file:FileReference,
    errorString:String):Void {
    trace("onSecurityError: " + file.name + " errorString: " + errorString);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

## onIOError (FileReference.onIOError 事件侦听器)

`onIOError = function(fileRef:FileReference) {}`

当发生输入 / 输出错误时调用。

当上载或下载由于下列任何原因而失败时调用此侦听器：

- 当播放器正在读取、写入或传输文件时发生输入 / 输出错误。
- SWF 文件尝试将文件上载到要求身份验证（如用户名和密码）的服务器。在上载期间，**Flash Player** 不提供用户用于输入密码的方法。如果 SWF 文件尝试将文件上载到要求身份验证的服务器，则上载会失败。
- SWF 文件尝试在独立或外部播放器中从要求身份验证的服务器下载文件。在下载期间，独立和外部播放器不提供用户用于输入密码的方法。如果这些播放器中的 SWF 文件尝试从要求身份验证的服务器下载文件，则下载会失败。只有在 **ActiveX** 控件和浏览器插件播放器中，文件下载才有可能成功。
- 传递到 `upload()` 中的 `url` 参数的值包含无效协议。有效协议包括 **HTTP** 和 **HTTPS**。

重要说明：只有在浏览器中运行的 **Flash** 应用程序（也就是说使用浏览器插件或 **ActiveX** 控件的应用程序）可以提供对话框来提示用户输入用户名和密码以进行身份验证，然后只用于下载。对于使用插件或 **ActiveX** 控件进行的上载或者使用独立或外部播放器进行的上载和下载，文件传输会失败。

可用性：ActionScript 1.0 ； Flash Player 8

### 参数

`fileRef:flash.net.FileReference` - 启动操作的 **FileReference** 对象。

### 示例

下面的示例在触发 `onIOError` 事件时跟踪消息。为简单起见，此示例中不包含其它事件侦听器。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onIOError = function(file:FileReference):Void {
    trace("onIOError");
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.download("http://www.macromedia.com/NonExistentFile.pdf",
    "NonExistentFile.pdf");
```

## onOpen（FileReference.onOpen 事件侦听器）

```
onOpen = function(fileRef:FileReference) {}
```

当上载或下载操作开始时调用。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**fileRef:**flash.net.FileReference - 启动操作的 FileReference 对象。

### 示例

下面的示例在触发 onOpen 事件时跟踪消息。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onOpen = function(file:FileReference):Void {
    trace("onOpen: " + file.name);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
fileRef.download(url, "FlashPlatform.pdf");
```

## onProgress（FileReference.onProgress 事件侦听器）

```
onProgress = function(fileRef:FileReference, bytesLoaded:Number,
    bytesTotal:Number) {}
```

在文件上载或下载操作期间定期调用。onProgress 侦听器在 **Flash Player** 将字节传输到服务器时调用，并在传输期间定期调用，即使传输最终没有成功。要确定文件传输是否成功以及何时完成，请使用 onComplete。

在某些情况下，不会调用 onProgress 侦听器；例如，当所传输的文件非常小或者上载或下载的速度非常快时。

在低于 OS X 10.3 版本的 **Macintosh** 平台上不能确定文件上载进度。onProgress 事件在上载操作期间调用，但是 bytesLoaded 参数的值为 -1，表示不能确定进度。

可用性：ActionScript 1.0；Flash Player 8

## 参数

**fileRef:**flash.net.FileReference - 启动操作的 **FileReference** 对象。

**bytesLoaded:**Number - 截止目前已传输的字节数。

**bytesTotal:**Number - 要传输的文件的总大小，以字节为单位。如果无法确定该大小，则值为 -1。

## 示例

下面的示例使用 `onProgress` 事件侦听器跟踪下载进度。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress: " + file.name + " with bytesLoaded: " + bytesLoaded + "
        bytesTotal: " + bytesTotal);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
fileRef.download(url, "FlashPlatform.pdf");
```

另请参见

## onSecurityError (FileReference.onSecurityError 事件侦听器)

`onSecurityError = function(fileRef:FileReference, errorString:String) {}`

当上载或下载由于安全错误而失败时调用。执行调用的 **SWF** 可能已经尝试访问其域外部的 **SWF** 文件，但却无权这样做。您可以通过使用跨域策略文件来尝试纠正此错误。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**fileRef:**flash.net.FileReference - 启动操作的 **FileReference** 对象。

**errorString:**String - 描述导致调用 `onSecurityError` 的错误。值为 `"securitySandboxError"`。

## 示例

下面的示例为每个可能的包括 `onSecurityError` 的事件创建一个带有侦听器的 **FileReference** 对象。`onSecurityError` 侦听器仅在上载由于安全错误而失败时触发。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onSelect = function(file:FileReference):Void {
    trace("onSelect: " + file.name);
    if(!file.upload("http://www.yourdomain.com/yourUploadHandlerScript.cfm"))
    {
        trace("Upload dialog failed to open.");
    }
}

listener.onCancel = function(file:FileReference):Void {
    trace("onCancel");
}

listener.onOpen = function(file:FileReference):Void {
    trace("onOpen: " + file.name);
}

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
        bytesTotal);
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

listener.onHTTPError = function(file:FileReference):Void {
    trace("onHTTPError: " + file.name);
}

listener.onIOError = function(file:FileReference):Void {
    trace("onIOError: " + file.name);
}

listener.onSecurityError = function(file:FileReference,
    errorString:String):Void {
    trace("onSecurityError: " + file.name + " errorString: " + errorString);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

## onSelect (FileReference.onSelect 事件侦听器)

```
onSelect = function(fileRef:FileReference) {}
```

当用户从文件浏览对话框选择要上传或下载的文件时调用。（当调用 `FileReference.browse()`、`FileReferenceList.browse()` 或 `FileReference.download()` 时将显示此对话框。）当用户选择文件并确认操作（例如，通过单击“确定”）时，会填充 **FileReference** 对象的属性。

`onSelect` 侦听器的运行方式稍微不同，具体取决于调用它的方法。当在调用 `browse()` 后调用 `onSelect` 时，**Flash Player** 能够读取 **FileReference** 对象的所有属性，因为用户所选的文件位于本地文件系统上。当在调用 `download()` 后调用 `onSelect` 时，**Flash Player** 只能读取 `name` 属性，因为在调用 `onSelect` 时该文件尚未下载到本地文件系统。在已下载该文件并调用 `onComplete` 后，**Flash Player** 就可以读取 **FileReference** 对象的所有其它属性了。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**fileRef**: `flash.net.FileReference` - 启动操作的 **FileReference** 对象。

### 示例

下面的示例在 `onSelect` 事件侦听器内跟踪消息。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(file:FileReference):Void {
    trace("onSelect: " + file.name);
    if(!file.upload("http://www.yourdomain.com/yourUploadHandlerScript.cfm"))
    {
        trace("Upload dialog failed to open.");
    }
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

## removeListener (FileReference.removeListener 方法)

```
public removeListener(listener:Object) : Boolean
```

从接收事件通知消息的对象列表中删除对象。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**listener:Object** - 一个对象，它侦听从 **FileReference** 事件侦听器发出的回调通知。

### 返回

**Boolean** - 如果成功删除了在 **listener** 参数中指定的对象，则返回 **true**。否则，此方法返回 **false**。

### 示例

下面的示例使用 **removeListener** 方法删除事件侦听器。如果用户取消下载，将删除该侦听器，以便它不再收到来自该 **FileReference** 对象的事件。

```
import flash.net.FileReference;

var listener:Object = new Object();

listener.onCancel = function(file:FileReference):Void {
    trace(file.removeListener(this)); // true
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
var url:String = "http://www.macromedia.com/platform/whitepapers/
    platform_overview.pdf";
fileRef.download(url, "FlashPlatform.pdf");
```



## size (FileReference.size 属性)

public size : Number [read-only]

本地磁盘上文件的大小，以字节表示。如果尚未填充 **FileReference** 对象，为获得此属性的值而执行的调用将返回 null。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例检索用户选择的文件的大小。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(file:FileReference):Void {
    trace("size: " + file.size + " bytes");
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## type (FileReference.type 属性)

public type : String [read-only]

文件类型。在 **Windows** 上，此属性是文件扩展名。在 **Macintosh** 上，此属性是由四个字符组成的文件类型。如果尚未填充 **FileReference** 对象，为获得此属性的值而执行的调用将返回 null。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例检索用户选择的文件的类型。

```
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(file:FileReference):Void {
    trace("type: " + file.type);
}

var fileRef:FileReference = new FileReference();
fileRef.addListener(listener);
fileRef.browse();
```

另请参见

[browse \(FileReference.browse 方法\)](#)

## upload (FileReference.upload 方法)

```
public upload(url:String) : Boolean
```

开始将用户选择的文件上载到远程服务器。**Flash Player** 可以上载最多 **100 MB** 的文件。必须在调用 `FileReference.browse()` 或 `FileReferenceList.browse()` 后，才能调用此方法。

侦听器接收事件来指示上载的进度、成功或失败。尽管可以使用 **FileReferenceList** 对象来允许用户选择多个要上载的文件，您必须逐个上载这些文件。为此，请循环访问 **FileReference** 对象的 `FileReferenceList.fileList` 数组。

文件上载到在 `url` 参数中传递的 **URL**。该 **URL** 必须是配置为接受上载的服务器脚本。**Flash Player** 使用 **HTTP POST** 方法上载文件。处理上载的服务器脚本应收到包含下列元素的 **POST** 请求：

- `multipart/form-data` 的 `Content-Type` 元素
- 一个 `Content-Disposition` 元素，其 `name` 属性设置为 `"Filedata"`，`filename` 属性设置为原始文件的名称
- 文件的二进制内容

下面是一个 **POST** 请求示例：

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="Filedata"; filename="example.jpg"
Content-Type: application/octet-stream
... contents of example.jpg ...
--AaB03x--
```

您可以通过将参数追加到 **URL** 使用 `upload()` 调用将数据发送至服务器。

**注意：**如果服务器要求用户身份验证，则只有在浏览器中运行的 **SWF** 文件（即使用浏览器插件或 **ActiveX** 控件的文件）才可以提供对话框来提示用户输入用户名和密码以进行身份验证，并且只适用于下载。对于使用插件或 **ActiveX** 控件进行的上载以及使用独立或外部播放器进行的上载和下载，文件传输会失败。

使用此方法时，请考虑 **Flash Player** 安全模型：

- 如果执行调用的 **SWF** 文件在不受信任的本地沙箱中，则不允许使用此方法。
- 默认值为拒绝在沙箱之间进行访问。网站可以通过添加跨域策略文件来实现对资源的访问。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 8

## 参数

`url:String` - 配置为通过 HTTP POST 调用处理上载的服务器脚本的 URL。该 URL 可以是 `http`，或者为了安全地进行上载，也可以是 `https`。

您可以通过将参数追加到 URL 使用 `upload()` 调用将数据发送至服务器；例如，`http://www.myserver.com/upload.cgi?userID=jdoe`

在某些浏览器上，URL 字符串长度受限。在某些浏览器或服务器上，长度超过 256 个字符的字符串可能失败。

## 返回

Boolean - 在下列任何情况下，值为 `false`：

- `FileReference.browse()` 尚未对此对象成功调用；或者  
`FileReferenceList.browse()` 尚未在其 `filelist` 数组中对此对象成功调用。
- 协议不是 HTTP 或 HTTPS。
- 发生了安全侵犯；即，SWF 文件试图从位于 SWF 文件的安全沙箱外的服务器访问文件。
- `url` 参数的类型或格式不正确。
- 此调用的参数数目不正确。

## 示例

下面的示例说明 `upload()` 方法的实现，方法是：首先提示用户选择要上载的文件，然后处理 `onSelect` 侦听器 and `onCancel` 侦听器，最后处理实际文件上载的结果。

```
import flash.net.FileReference;

var allTypes:Array = new Array();
var imageTypes:Object = new Object();
imageTypes.description = "Images (*.jpg, *.jpeg, *.gif, *.png)";
imageTypes.extension = "*.jpg; *.jpeg; *.gif; *.png";
allTypes.push(imageTypes);

var listener:Object = new Object();

listener.onSelect = function(file:FileReference):Void {
    trace("onSelect: " + file.name);
```

```

        if(!file.upload("http://www.yourdomain.com/yourUploadHandlerScript.cfm"))
        {
            trace("Upload dialog failed to open.");
        }
    }

    listener.onCancel = function(file:FileReference):Void {
        trace("onCancel");
    }

    listener.onOpen = function(file:FileReference):Void {
        trace("onOpen: " + file.name);
    }

    listener.onProgress = function(file:FileReference, bytesLoaded:Number,
        bytesTotal:Number):Void {
        trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
            bytesTotal);
    }

    listener.onComplete = function(file:FileReference):Void {
        trace("onComplete: " + file.name);
    }

    listener.onHTTPError = function(file:FileReference):Void {
        trace("onHTTPError: " + file.name);
    }

    listener.onIOError = function(file:FileReference):Void {
        trace("onIOError: " + file.name);
    }

    listener.onSecurityError = function(file:FileReference,
        errorString:String):Void {
        trace("onSecurityError: " + file.name + " errorString: " + errorString);
    }

    var fileRef:FileReference = new FileReference();
    fileRef.addListener(listener);
    fileRef.browse(allTypes);

```

另请参见

[browse \(FileReference.browse 方法\)](#) , [browse \(FileReferenceList.browse 方法\)](#) ,  
[download \(FileReference.download 方法\)](#) , [fileList \(FileReferenceList.fileList 属性\)](#)

# FileReferenceList

## (flash.net.FileReferenceList)

```
Object
|
+- flash.net.FileReferenceList
```

```
public class FileReferenceList
extends Object
```

**FileReferenceList** 类提供了让用户选择一个或多个要上载的文件的方法。**FileReferenceList** 对象将用户磁盘上的一组本地文件（一个或多个文件）表示为 **FileReference** 对象的数组。有关详细信息以及有关 **FileReference** 对象和 **FileReference** 类（可与 **FileReferenceList** 一起使用）的重要注意事项，请参见 **FileReference** 类。

使用 **FileReferenceList** 类：

- 将该类实例化：var myFileRef = new FileReferenceList();
- 调用 FileReferenceList.browse() 以显示对话框，用户可从中选择一个或多个要上载的文件：myFileRef.browse();
- 成功调用 browse() 后，用 **FileReference** 对象的数组来填充 **FileReferenceList** 对象的 fileList 属性。
- 对 fileList 数组中的每个元素调用 FileReference.upload()。

**FileReferenceList** 类包括 browse() 方法以及用于处理多个文件的 fileList 属性。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例允许用户选择多个文件，然后将每个文件上载到服务器。

```
import flash.net.FileReferenceList;
import flash.net.FileReference;

var listener:Object = new Object();

listener.onSelect = function(fileRefList:FileReferenceList) {
    trace("onSelect");
    var list:Array = fileRefList.fileList;
    var item:FileReference;
    for(var i:Number = 0; i < list.length; i++) {
        item = list[i];
        trace("name: " + item.name);
        trace(item.addListener(this));
        item.upload("http://www.yourdomain.com/");
    }
}
```

```
listener.onCancel = function():Void {
    trace("onCancel");
}

listener.onOpen = function(file:FileReference):Void {
    trace("onOpen: " + file.name);
}

listener.onProgress = function(file:FileReference, bytesLoaded:Number,
    bytesTotal:Number):Void {
    trace("onProgress with bytesLoaded: " + bytesLoaded + " bytesTotal: " +
        bytesTotal);
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

listener.onHTTPError = function(file:FileReference, httpError:Number):Void {
    trace("onHTTPError: " + file.name + " httpError: " + httpError);
}

listener.onIOError = function(file:FileReference):Void {
    trace("onIOError: " + file.name);
}

listener.onSecurityError = function(file:FileReference,
    errorString:String):Void {
    trace("onSecurityError: " + file.name + " errorString: " + errorString);
}

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef.browse();
```

另请参见  
[FileReference \(flash.net.FileReference\)](#)

属性摘要

修饰符	属性	说明
	fileList:Array	FileReference 对象的数组。

继承自 Object 类的属性

<a href="#">constructor</a> (Object.constructor 属性), <a href="#">__proto__</a> (Object.__proto__ 属性), <a href="#">prototype</a> (Object.prototype 属性), <a href="#">__resolve</a> (Object.__resolve 属性)
--

事件摘要

事件	说明
<code>onCancel = function(fileRefList:FileReferenceList) {}</code>	当用户取消文件浏览对话框时调用。
<code>onSelect = function(fileRefList:FileReferenceList) {}</code>	当用户从文件浏览对话框选择一个或多个要上载的文件时调用。

构造函数摘要

签名	说明
<code>FileReferenceList()</code>	创建新的 <code>FileReferenceList</code> 对象。

方法摘要

修饰符	签名	说明
	<code>addListener(listener:Object) : Void</code>	注册一个对象，以便在调用 <code>FileReferenceList</code> 事件侦听器时接收通知。
	<code>browse([typelist:Array]) : Boolean</code>	显示文件浏览对话框，用户可从中选择一个或多个要上载的本地文件。
	<code>removeListener(listener:Object) : Boolean</code>	从接收事件通知消息的对象列表中删除对象。

继承自 `Object` 类的方法

<code>addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)</code>
--

## addListener (FileReferenceList.addListener 方法)

```
public addListener(listener:Object) : Void
```

注册一个对象，以便在调用 **FileReferenceList** 事件侦听器时接收通知。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**listener:Object** - 一个对象，它侦听从 **FileReferenceList** 事件侦听器发出的回调通知。

### 示例

下面的示例对 `addListener()` 方法进行了演示。

```
import flash.net.FileReferenceList;

var listener:Object = new Object();
listener.onCancel = function(fileRefList:FileReferenceList) {
    trace("onCancel");
}

listener.onSelect = function(fileRefList:FileReferenceList) {
    trace("onSelect: " + fileRefList.fileList.length);
}

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef.browse();
```

## browse (FileReferenceList.browse 方法)

```
public browse([typelist:Array]) : Boolean
```

显示文件浏览对话框，用户可从中选择一个或多个要上载的本地文件。该对话框对于用户的操作系统来说是本机的。当您调用此方法并且用户成功选择文件时，此 **FileReferenceList** 对象的 `fileList` 属性用 **FileReference** 对象数组来填充，每个对象对应一个用户选择的文件。随后每次调用 `FileReferenceList.browse()` 时，**FileReferenceList.fileList** 属性都重置为用户在对话框中选择的文件。

可以传递一个文件类型数组以确定对话框显示的文件。

每次只能对 **FileReferenceList** 对象执行一个 `browse()` 或 `download()` 会话（因为每次只能显示一个对话框）。

可用性: **ActionScript 1.0** ; **Flash Player 8**



## 参数

**typelist**: Array [可选] - 一个文件类型数组, 用于过滤对话框中显示的文件。如果省略此参数, 则显示所有文件。如果包括此参数, 数组必须包含用花括号 { } 括起来的一个或多个元素。数组可采用以下两种格式:

- 后面只跟其 **Windows** 文件扩展名的文件类型描述列表。数组中的每个元素必须包含描述文件类型的字符串和用分号分隔的 **Windows** 文件扩展名的列表 (在每个扩展名前有一个通配符 (\*))。每个元素的语法如下所示:

```
[{description: "string describing the first set of file types", extension: "semicolon-delimited list of file extensions"}]
```

示例:

```
[{description: "Images", extension: "*.jpg;*.gif;*.png"}, {description: "Flash Movies", extension: "*.swf"}, {description: "Documents", extension: "*.doc;*.pdf"}]
```

- 后跟其 **Windows** 文件扩展名和 **Macintosh** 文件类型的文件类型描述列表。数组中的每个元素必须包含: 描述文件类型的字符串; 用分号分隔的 **Windows** 文件扩展名的列表 (每个扩展名前都有一个通配符 (\*)); 用分号分隔的 **Macintosh** 文件类型的列表 (每个类型名前都有一个通配符 (\*))。每个元素的语法如下所示:

```
[{description: "string describing the first set of file types", extension: "semicolon-delimited list of Windows file extensions", macType: "semicolon-delimited list of Macintosh file types"}]
```

示例:

```
[{description: "Image files", extension: "*.jpg;*.gif;*.png", macType: "JPEG;jp2_;GIF"}, {description: "Flash Movies", extension: "*.swf", macType: "SWFL"}]
```

这两种格式不能在一个 `browse()` 调用中互换。必须使用其中一种格式。

扩展名列表在 **Windows** 上用于过滤文件, 具体取决于用户选择的文件类型。实际上它不显示在对话框中。要对用户显示文件类型, 必须在描述字符串以及扩展名列表中列出文件类型。在 **Windows** 上, 描述字符串显示在对话框中。(在 **Macintosh** 上不使用它。)在 **Macintosh** 上, 如果提供 **Macintosh** 文件类型的列表, 则该列表用于过滤文件。如果没有提供 **Macintosh** 文件类型的列表, 则使用 **Windows** 扩展名的列表。

## 返回

Boolean - 如果参数有效并且显示文件浏览对话框, 则返回 true。以下情况下返回 false: 未显示对话框; 正在进行另一个浏览器会话; 使用 `typelist` 参数, 但未能在数组的任一元素中提供描述或扩展名字符串。

## 示例

下面的示例对 `browse()` 方法进行了演示。

```
import flash.net.FileReferenceList;

var allTypes:Array = new Array();
var imageTypes:Object = new Object();
imageTypes.description = "Images (*.JPG;*.JPEG;*.JPE;*.GIF;*.PNG;)";
imageTypes.extension = "*.jpg; *.jpeg; *.jpe; *.gif; *.png;";
allTypes.push(imageTypes);

var textTypes:Object = new Object();
textTypes.description = "Text Files (*.TXT;*.RTF;)";
textTypes.extension = "*.txt; *.rtf";
allTypes.push(textTypes);

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.browse(allTypes);
```

## 另请参见

[browse \(FileReference.browse 方法\)](#), [FileReference \(flash.net.FileReference\)](#)

## fileList (FileReferenceList.fileList 属性)

```
public fileList : Array
```

**FileReference** 对象的数组。

在已调用 `FileReferenceList.browse()` 方法并且用户已从 `browse()` 打开的对话框中选择一个或多个文件后，此属性用 **FileReference** 对象的数组来填充，每个对象表示用户选择的一个文件。然后，您可以通过 `FileReference.upload()` 使用此数组上载这些文件。必须一次上载一个文件。

每次对该 **FileReferenceList** 对象调用 `browse()` 时都要重新填充 `fileList` 属性。

**FileReference** 对象的属性在 **FileReference** 类文档中介绍。

可用性：ActionScript 1.0 ； Flash Player 8

## 示例

下面的示例对 `fileList` 属性进行了演示。

```
import flash.net.FileReferenceList;
import flash.net.FileReference;

var listener:Object = new Object();
listener.onSelect = function(fileRefList:FileReferenceList) {
    trace("onSelect");
    var list:Array = fileRefList.fileList;
```

```

        var item:FileReference;
        for(var i:Number = 0; i < list.length; i++) {
            item = list[i];
            trace("name: " + item.name);
        }
    }

    var fileRef:FileReferenceList = new FileReferenceList();
    fileRef.addListener(listener);
    fileRef.browse();

```

另请参见

[FileReference \(flash.net.FileReference\)](#), [upload \(FileReference.upload 方法\)](#),  
[browse \(FileReferenceList.browse 方法\)](#)

## FileReferenceList 构造函数

```
public FileReferenceList()
```

创建新的 **FileReferenceList** 对象。此对象不包含任何内容，直到对它调用 `browse()`。当对 **FileReference** 对象调用 `browse()` 时，该对象的 `fileList` 属性用 **FileReference** 对象的数组来填充。

可用性: ActionScript 1.0 ; Flash Player 8

示例

下面的示例创建一个新的 `FileReferenceList` 对象，迭代所选的每个文件并输出它们的名称。

```

import flash.net.FileReferenceList;

var listener:Object = new Object();
listener.onSelect = function(fileRefList:FileReferenceList) {
    trace("onSelect");
    var arr:Array = fileRefList.fileList;
    for(var i:Number = 0; i < arr.length; i++) {
        trace("name: " + arr[i].name);
    }
}

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef.browse();

```

另请参见

[FileReference \(flash.net.FileReference\)](#), [browse \(FileReferenceList.browse 方法\)](#)

## onCancel (FileReferenceList.onCancel 事件侦听器)

```
onCancel = function(fileRefList:FileReferenceList) {}
```

当用户取消文件浏览对话框时调用。（当调用 `FileReferenceList.browse()`、`FileReference.browse()` 或 `FileReference.download()` 方法时显示此对话框。）

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**fileRefList:**flash.net.FileReferenceList - 启动操作的 **FileReferenceList** 对象。

### 示例

下面的示例对 onCancel 侦听器进行了演示。

```
import flash.net.FileReferenceList;

var listener:Object = new Object();
listener.onCancel = function(fileRefList:FileReferenceList) {
    trace("onCancel");
}

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef.browse();
```

### 另请参见

[browse \(FileReferenceList.browse 方法\)](#)

## onSelect (FileReferenceList.onSelect 事件侦听器)

```
onSelect = function(fileRefList:FileReferenceList) {}
```

当用户从文件浏览对话框选择一个或多个要上载的文件时调用。（当调用 `FileReferenceList.browse()`、`FileReference.browse()` 或 `FileReference.download()` 方法时显示此对话框。）当用户选择文件并确认该操作（例如，通过单击“保存”）时，会使用表示用户选择的文件的 **FileReference** 对象填充 **FileReferenceList** 对象。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**fileRefList:**flash.net.FileReferenceList - 启动操作的 **FileReferenceList** 对象。

## 示例

下面的示例对 onSelect 侦听器进行了演示。

```
import flash.net.FileReferenceList;
import flash.net.FileReference;

var listener:Object = new Object();

listener.onSelect = function(fileRefList:FileReferenceList) {
    trace("onSelect");
    var list:Array = fileRefList.fileList;
    var item:FileReference;
    for(var i:Number = 0; i < list.length; i++) {
        item = list[i];
        trace("name: " + item.name);
        trace(item.addListener(this));
        item.upload("http://www.yourdomain.com/");
    }
}

listener.onComplete = function(file:FileReference):Void {
    trace("onComplete: " + file.name);
}

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef.browse();
```

## 另请参见

[browse \(FileReferenceList.browse 方法\)](#)

## removeListener (FileReferenceList.removeListener 方法)

```
public removeListener(listener:Object) : Boolean
```

从接收事件通知消息的对象列表中删除对象。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**listener:Object** - 一个对象，它侦听从 **FileReferenceList** 事件侦听器发出的回调通知。

## 返回

**Boolean** - 如果删除了该对象，则返回 **true**。否则，此方法返回 **false**。

## 示例

下面的示例对 `removeListener` 方法进行了演示。

```
import flash.net.FileReferenceList;

var listener:Object = new Object();
listener.onCancel = function(fileRefList:FileReferenceList) {
    trace("onCancel");
    trace(fileRefList.removeListener(this)); // true
}

listener.onSelect = function(fileRefList:FileReferenceList) {
    trace("onSelect: " + fileRefList.fileList.length);
}

var fileRef:FileReferenceList = new FileReferenceList();
fileRef.addListener(listener);
fileRef.browse();
```

# Function

```
Object
|
+-Function
```

```
public dynamic class Function
extends Object
```

**ActionScript** 中用户定义的函数和内置函数都由 **Function** 对象来表示，该对象是 **Function** 类的实例。

可用性：ActionScript 1.0；Flash Player 6

## 属性摘要

继承自 **Object** 类的属性

---

`constructor` (`Object.constructor` 属性), `__proto__` (`Object.__proto__` 属性),  
`prototype` (`Object.prototype` 属性), `__resolve` (`Object.__resolve` 属性)

---

方法摘要

修饰符	签名	说明
	<code>apply(thisObject:Object, [argArray:Array]) : Void</code>	指定要在 <code>ActionScript</code> 调用的任何函数内使用的 <code>thisObject</code> 的值。
	<code>call(thisObject:Object, [parameter1:Object]) : Object</code>	调用 <code>Function</code> 对象表示的函数。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

# `apply` (`Function.apply` 方法)

`public apply(thisObject:Object, [argArray:Array]) : Void`

指定要在 `ActionScript` 调用的任何函数内使用的 `thisObject` 的值。此方法还指定要传递给任何被调用函数的参数。由于 `apply()` 是 `Function` 类的方法，所以它也是 `ActionScript` 中每个 `Function` 对象的方法。

与 `Function.call()`（它将参数指定为用逗号分隔的列表）不同，该方法将参数指定为一个 `Array` 对象。如果在脚本实际执行前，无法知道要传递的参数的数量，那么这种方法通常很有用。

返回被调用函数指定为返回值的值。

可用性: `ActionScript 1.0` ; `Flash Player 6`

## 参数

`thisObject:Object` - 要应用 `myFunction` 的对象。

`argArray:Array` [ 可选 ] - 一个数组，其元素作为参数传递给 `myFunction`。

## 示例

下面的函数调用是等效的:

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

下面的简单示例演示 `apply()` 如何传递参数数组：

```
function theFunction() {
    trace(arguments);
}

// create a new array to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
theFunction.apply(null,firstArray);
// outputs: 1,2,3

// create a second array to pass as a parameter to apply()
var secondArray:Array = new Array("a", "b", "c");
theFunction.apply(null,secondArray);
// outputs a,b,c
```

下面的示例演示 `apply()` 如何传递参数数组并指定 `this` 的值：

```
// define a function
function theFunction() {
    trace("this == myObj? " + (this == myObj));
    trace("arguments: " + arguments);
}

// instantiate an object
var myObj:Object = new Object();

// create arrays to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
var secondArray:Array = new Array("a", "b", "c");

// use apply() to set the value of this to be myObj and send firstArray
theFunction.apply(myObj,firstArray);
// output:
// this == myObj? true
// arguments: 1,2,3

// use apply() to set the value of this to be myObj and send secondArray
theFunction.apply(myObj,secondArray);
// output:
// this == myObj? true
// arguments: a,b,c
```

另请参见

[call \(Function.call 方法\)](#)



## call (Function.call 方法)

```
public call(thisObject:Object, [parameter1:Object]) : Object
```

调用 **Function** 对象表示的函数。**ActionScript** 中的每个函数都由一个 **Function** 对象来表示，所以所有的函数都支持此方法。

几乎在所有的情形下，都可以使用函数调用运算符 (`()`) 来代替此方法。函数调用运算符使代码简明易读。此方法主要用于需要显式控制函数调用中的 `thisObject` 参数时。通常，如果将函数作为对象的方法来调用，则在函数体内，`thisObject` 设置为 `myObject`，如下面的示例所示：

```
myObject.myMethod(1, 2, 3);
```

在某些情况下，您可能希望 `thisObject` 指向其它地方；例如，函数必须作为对象的方法进行调用，但该函数实际上并不作为该对象的方法进行存储。

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

您可以将值 **null** 传递给 `thisObject` 参数，以便作为常规函数而不是作为对象的方法来调用函数。例如，下面的函数调用是等效的：

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

返回被调用函数指定为返回值的值。

可用性：ActionScript 1.0 ； Flash Player 6

### 参数

**thisObject:Object** - 一个对象，指定函数体内 `thisObject` 的值。

**parameter1:Object** [ 可选 ] - 要传递给 **myFunction** 的参数。可以指定零个或多个参数。

### 返回

**Object** -

### 示例

下面的示例使用 `Function.call()` 使函数表现得像另一个对象的方法，而不将函数存储在对象中：

```
function myObject() {
}
function myMethod(obj) {
    trace("this == obj? " + (this == obj));
}
var obj:Object = new myObject();
myMethod.call(obj, obj);

trace() 语句将显示：
this == obj? true
```

另请参见

[apply \(Function.apply 方法\)](#)

## GlowFilter (flash.filters.GlowFilter)

```
Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.GlowFilter
```

```
public class GlowFilter
extends BitmapFilter
```

使用 **GlowFilter** 类，您可以在 **Flash** 中给各种对象应用发光效果。有多个用于发光样式的选项，包括内侧发光或外侧发光以及挖空模式。在投影的 `distance` 属性和 `angle` 属性设置为 **0** 时，发光滤镜与投影滤镜极为相似。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时给影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤消设置操作。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter()` 将获得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作过程中给图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜支持舞台缩放。但是，它不支持常规缩放、旋转和倾斜；如果对象本身进行了缩放（如果 `_xscale` 和 `_yscale` 不是 **100%**），滤镜效果将不进行缩放。只有在放大舞台时，滤镜效果才进行缩放。

如果结果图像的宽度或高度超过 **2880** 像素，则不应用滤镜。例如，如果您在放大某个大型影片剪辑时应用了滤镜，则在结果图像超过 **2880** 像素的限制时滤镜将关闭。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[applyFilter \(BitmapData.applyFilter 方法\)](#)，[cacheAsBitmap \(Button.cacheAsBitmap 属性\)](#)，[filters \(Button.filters 属性\)](#)，[DropShadowFilter \(flash.filters.DropShadowFilter\)](#)，[cacheAsBitmap \(MovieClip.cacheAsBitmap 属性\)](#)，[filters \(MovieClip.filters 属性\)](#)，[filters \(TextField.filters 属性\)](#)

属性摘要

修饰符	属性	说明
	alpha:Number	颜色的 Alpha 透明度值。
	blurX:Number	水平模糊量。
	blurY:Number	垂直模糊量。
	color:Number	光晕颜色。
	inner:Boolean	指定发光是否为内侧发光。
	knockout:Boolean	指定对象是否具有挖空效果。
	quality:Number	应用滤镜的次数。
	strength:Number	印记或跨页的强度。

继承自 Object 类的属性

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

构造函数摘要

签名	说明
<code>GlowFilter([color:Number], [alpha:Number], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [inner:Boolean], [knockout:Boolean])</code>	用指定参数初始化新的 GlowFilter 实例。

方法摘要

修饰符	签名	说明
	clone() : GlowFilter	返回此滤镜对象的副本。

继承自 BitmapFilter 类的方法

[clone](#) ([BitmapFilter.clone](#) 方法)

继承自 Object 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

## alpha (GlowFilter.alpha 属性)

public alpha : Number

颜色的 Alpha 透明度值。有效值为 0 到 1。例如，.25 设置透明度值为 25%。默认值是 1。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 alpha 属性。

```
import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterAlpha");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.alpha = .4;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
}
```

```

rect.lineTo(0, 0);
rect._x = 20;
rect._y = 20;

var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
false);
var filterArray:Array = new Array();
filterArray.push(filter);
rect.filters = filterArray;
return rect;
}

```

## blurX (GlowFilter.blurX 属性)

public blurX : Number

水平模糊量。有效值为 0 到 255（浮点）。默认值为 6。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，它的呈现速度会比其它值更快。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 blurX 属性。

```

import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterBlurX");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.blurX = 20;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;
}

```

```

        var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
        false);
        var filterArray:Array = new Array();
        filterArray.push(filter);
        rect.filters = filterArray;
        return rect;
    }

```

## blurY (GlowFilter.blurY 属性)

public blurY : Number

垂直模糊量。有效值为 0 到 255（浮点）。默认值为 6。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，它的呈现速度会比其它值更快。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 blurY 属性。

```

import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterBlurY");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.blurY = 20;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
    this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
    false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}

```

## clone (GlowFilter.clone 方法)

public clone() : GlowFilter

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.filters.GlowFilter - 具有原始 **GlowFilter** 实例的所有属性的新 **GlowFilter** 实例。

### 示例

下面的示例创建三个 **GlowFilter** 对象并将它们进行比较: filter\_1 是通过使用 **GlowFilter** 构造函数创建的; filter\_2 是通过将其设置为等效于 filter\_1 创建的; 而 clonedFilter 是通过克隆 filter\_1 创建的。请注意, 尽管 filter\_2 等效于 filter\_1, 但 clonedFilter 却不等效于 filter\_1, 尽管它们包含相同的值。

```
import flash.filters.GlowFilter;

var filter_1:GlowFilter = new GlowFilter(0x33CCFF, .8, 35, 35, 2, 3, false,
    false);
var filter_2:GlowFilter = filter_1;
var clonedFilter:GlowFilter = filter_1.clone();

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> strength: 2
    // >> blurY: 35
    // >> blurX: 35
    // >> knockout: false
    // >> inner: false
    // >> quality: 3
    // >> alpha: 0.8
    // >> color: 3394815
}

for(var i in clonedFilter) {
    trace(">> " + i + ": " + clonedFilter[i]);
    // >> clone: [type Function]
    // >> strength: 2
    // >> blurY: 35
    // >> blurX: 35
    // >> knockout: false
```

```

    // >> inner: false
    // >> quality: 3
    // >> alpha: 0.8
    // >> color: 3394815
}

```

为了进一步说明 filter\_1、filter\_2 和 clonedFilter 之间的关系，下面的示例将修改 filter\_1 的 knockout 属性。通过修改 knockout，说明了 clone() 方法是根据 filter\_1 的值而不是通过在引用中指向这些值来创建新实例的。

```

import flash.filters.GlowFilter;

var filter_1:GlowFilter = new GlowFilter(0x33CCFF, .8, 35, 35, 2, 3, false,
    false);
var filter_2:GlowFilter = filter_1;
var clonedFilter:GlowFilter = filter_1.clone();

trace(filter_1.knockout); // false
trace(filter_2.knockout); // false
trace(clonedFilter.knockout); // false

filter_1.knockout = true;

trace(filter_1.knockout); // true
trace(filter_2.knockout); // true
trace(clonedFilter.knockout); // false

```

## color（GlowFilter.color 属性）

public color : Number

光晕颜色。有效值采用十六进制格式 0xRRGGBB。默认值为 0xFF0000。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 color 属性。

```

import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterColor");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.color = 0x00FF33;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
}

```



```

var w:Number = 100;
var h:Number = 100;
rect.beginFill(0x003366);
rect.lineTo(w, 0);
rect.lineTo(w, h);
rect.lineTo(0, h);
rect.lineTo(0, 0);
rect._x = 20;
rect._y = 20;

var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
false);
var filterArray:Array = new Array();
filterArray.push(filter);
rect.filters = filterArray;
return rect;
}

```

## GlowFilter 构造函数

```

public GlowFilter([color:Number], [alpha:Number], [blurX:Number],
    [blurY:Number], [strength:Number], [quality:Number], [inner:Boolean],
    [knockout:Boolean])

```

用指定参数初始化新的 **GlowFilter** 实例。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 参数

**color:**Number [可选] - 光晕颜色, 采用十六进制格式 **0xRRGGBB**。默认值为 **0xFF0000**。

**alpha:**Number [可选] - 颜色的 **Alpha** 透明度值。有效值为 0 到 1。例如, .25 设置透明度值为 25%。默认值是 1。

**blurX:**Number [可选] - 水平模糊量。有效值为 0 到 255 (浮点)。默认值为 6。作为 2 的乘方的值 (如 2、4、8、16 和 32) 经过了优化, 它的呈现速度会比其它值更快。

**blurY:**Number [可选] - 垂直模糊量。有效值为 0 到 255 (浮点)。默认值为 6。作为 2 的乘方的值 (如 2、4、8、16 和 32) 经过了优化, 它的呈现速度会比其它值更快。

**strength:**Number [可选] - 印记或散布的强度。该值越高, 印记的颜色越深, 而且发光与背景之间的对比度也越强。有效值为 0 到 255。默认值为 2。

**quality:**Number [可选] - 应用滤镜的次数。有效值为 0 到 15。默认值为 1, 它表示低品质。值为 2 表示中等品质, 值为 3 表示高品质。

**inner: Boolean** [ 可选 ] - 指定发光是否为内侧发光。值 `true` 表示内侧发光。默认值为 `false`，即外侧发光，它表示对象外缘周围的发光。

**knockout: Boolean** [ 可选 ] - 指定对象是否具有挖空效果，值为 `true` 将使对象的填充变为透明，并显示文档的背景颜色。默认值为 `false`（不应用挖空效果）。

## 示例

下面的示例实例化一个新 **GlowFilter** 实例，并将其应用于一个平面矩形形状。

```
import flash.filters.GlowFilter;

var rect:MovieClip = createRectangle(100, 100, 0x003366,
    "gradientGlowFilterExample");

var color:Number = 0x33CCFF;
var alpha:Number = .8;
var blurX:Number = 35;
var blurY:Number = 35;
var strength:Number = 2;
var quality:Number = 3;
var inner:Boolean = false;
var knockout:Boolean = false;

var filter:GlowFilter = new GlowFilter(color,
    alpha,
    blurX,
    blurY,
    strength,
    quality,
    inner,
    knockout);
var filterArray:Array = new Array();
filterArray.push(filter);
rect.filters = filterArray;

function createRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    mc.beginFill(bgColor);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc._x = 20;
    mc._y = 20;
    return mc;
}
```

## inner (GlowFilter.inner 属性)

public inner : Boolean

指定发光是否为内侧发光。值 true 表示内侧发光。默认值为 false，即外侧发光，它表示对象外缘周围的发光。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 inner 属性。

```
import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterInner");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.inner = true;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
        false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}
```

## knockout (GlowFilter.knockout 属性)

public knockout : Boolean

指定对象是否具有挖空效果。值为 true 将使对象的填充变为透明，并显示文档的背景颜色。默认值为 false（不应用挖空效果）。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 knockout 属性。

```
import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterKnockout");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.knockout = true;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
        false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}
```

## quality (GlowFilter.quality 属性)

public quality : Number

应用滤镜的次数。有效值为 **0** 到 **15**。默认值为 **1**，它表示低品质。值为 **2** 表示中等品质，值为 **3** 表示高品质。滤镜的值越小，呈现速度越快。

对于大多数应用，quality 的值为 **1**、**2** 或 **3** 就足够了。您可以使用其它数值（最高为 **15**）来达到不同的效果，但是值越高，呈现速度越慢。除了增加 quality 的值，增加 blurX 和 blurY 的值通常也可以获得类似的效果，而且呈现速度更快。

**可用性:** **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 quality 属性。

```
import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterQuality");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.quality = 1;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
        false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}
```

## strength (GlowFilter.strength 属性)

public strength : Number

印记或跨页的强度。该值越高，印记的颜色越深，而且发光与背景之间的对比度也越强。有效值为 0 到 255。默认值为 2。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 strength 属性。

```
import flash.filters.GlowFilter;

var mc:MovieClip = createGlowFilterRectangle("GlowFilterStrength");
mc.onRelease = function() {
    var filter:GlowFilter = this.filters[0];
    filter.strength = .8;
    this.filters = new Array(filter);
}

function createGlowFilterRectangle(name:String):MovieClip {
    var rect:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    rect.beginFill(0x003366);
    rect.lineTo(w, 0);
    rect.lineTo(w, h);
    rect.lineTo(0, h);
    rect.lineTo(0, 0);
    rect._x = 20;
    rect._y = 20;

    var filter:GlowFilter = new GlowFilter(0x000000, .8, 16, 16, 1, 3, false,
        false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    rect.filters = filterArray;
    return rect;
}
```

# GradientBevelFilter

## (flash.filters.GradientBevelFilter)

```
Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.GradientBevelFilter
```

```
public class GradientBevelFilter
extends BitmapFilter
```

使用 **GradientBevelFilter** 类，您可以在 **Flash** 中给各种对象应用渐变斜角效果。渐变斜角是位于对象外部、内部或顶部的使用渐变色增强的有斜面的边缘。有斜面的边缘使对象具有三维外观。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时给影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤销该滤镜。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter()` 会获得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作过程中给图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜支持舞台缩放。但是，它不支持常规缩放、旋转和倾斜；如果对象本身进行了缩放（如果 `_xscale` 和 `_yscale` 不是 100%），滤镜效果将不进行缩放。只有在放大舞台时，滤镜效果才进行缩放。

如果结果图像的宽度或高度超过 2880 像素，则不应用滤镜。例如，如果您在放大某个大型影片剪辑时应用了滤镜，则该滤镜将在结果图像超过 2880 像素的限制时关闭。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[ratios](#) ([GradientBevelFilter.ratios](#) 属性)，[applyFilter](#) ([BitmapData.applyFilter](#) 方法)，[BevelFilter](#) ([flash.filters.BevelFilter](#))，[filters](#) ([Button.filters](#) 属性)，[cacheAsBitmap](#) ([Button.cacheAsBitmap](#) 属性)，[cacheAsBitmap](#) ([MovieClip.cacheAsBitmap](#) 属性)，[filters](#) ([MovieClip.filters](#) 属性)，[filters](#) ([TextField.filters](#) 属性)

属性摘要

修饰符	属性	说明
	alphas:Array	colors 数组中对应颜色的 Alpha 透明度值的数组。
	angle:Number	角度，以度为单位。
	blurX:Number	水平模糊量。
	blurY:Number	垂直模糊量。
	colors:Array	渐变中使用的 RGB 十六进制颜色值数组。
	distance:Number	偏移距离。
	knockout:Boolean	指定对象是否具有挖空效果。
	quality:Number	应用滤镜的次数。
	ratios:Array	对应于 colors 数组中颜色的一组颜色分布比率。
	strength:Number	印记或跨页的强度。
	type:String	斜角效果的放置。

继承自 Object 类的属性

<a href="#">constructor</a> ( <a href="#">Object.constructor</a> 属性), <a href="#">__proto__</a> ( <a href="#">Object.__proto__</a> 属性), <a href="#">prototype</a> ( <a href="#">Object.prototype</a> 属性), <a href="#">__resolve</a> ( <a href="#">Object.__resolve</a> 属性)
--

构造函数摘要

签名	说明
<code>GradientBevelFilter([distance:Number], [angle:Number], [colors:Array], [alphas:Array], [ratios:Array], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [type:String], [knockout:Boolean])</code>	用指定参数初始化滤镜。



方法摘要

修饰符	签名	说明
	<code>clone() : GradientBevelFilter</code>	返回此滤镜对象的副本。

继承自 `BitmapFilter` 类的方法

[clone](#) ([BitmapFilter.clone](#) 方法)

继承自 `Object` 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

# alphas (GradientBevelFilter.alphas 属性)

`public alphas : Array`

`colors` 数组中对应颜色的 **Alpha** 透明度值的数组。数组中每个元素的有效值为 **0** 到 **1**。例如, `.25` 设置透明度值为 **25%**。

`alphas` 属性不能通过直接修改它的值来进行更改。相反, 您必须获取对 `alphas` 的引用, 对该引用进行更改, 然后将 `alphas` 设置为该引用。

`colors`、`alphas` 和 `ratios` 属性都是相关的。`colors` 数组中的第一个元素对应于 `alphas` 数组中的第一个元素以及 `ratios` 数组中的第一个元素, 依此类推。

可用性: **ActionScript 1.0 ; Flash Player 8**

## 示例

下面的示例演示如何在现有实体上设置 `alphas` 属性。

```
import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("alphasExample");
mc.onPress = function() {
    var arr:Array = this.filters;
    var alphas:Array = [.2, 0, .2];
    arr[0].alphas = alphas;
    this.filters = arr;
}
mc.onRelease = function() {
    var arr:Array = this.filters;
    var alphas:Array = [1, 0, 1];
    arr[0].alphas = alphas;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
        alphas, ratios, 5, 5, 5, 2, "inner", false);

    art.filters = new Array(filter);
    return art;
}
```

## 另请参见

[colors](#) ([GradientBevelFilter.colors](#) 属性), [ratios](#) ([GradientBevelFilter.ratios](#) 属性)

## angle (GradientBevelFilter.angle 属性)

public angle : Number

角度，以度为单位。有效值为 **0** 到 **360**。默认值为 **45**。

角度值表示理论上的光源落在对象上的角度。该值决定渐变色应用于对象的角度：加亮和阴影出现的位置，或第一种颜色在数组中出现的位置。然后，按这些颜色在数组中出现的顺序应用颜色。

**可用性：** ActionScript 1.0 ； Flash Player 8

### 示例

下面的示例演示如何在现有对象上设置 angle 属性。

```
import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("angleExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].angle = 45;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
        alphas, ratios, 5, 5, 5, 3, "inner", false);

    art.filters = new Array(filter);
    return art;
}
```

另请参见

[ratios \(GradientBevelFilter.ratios 属性\)](#)

## blurX (GradientBevelFilter.blurX 属性)

public blurX : Number

水平模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例演示如何在现有对象上设置 blurX 属性。

```
import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("blurXExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].blurX = 16;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
    this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
    alphas, ratios, 5, 5, 5, 3, "inner", false);

    art.filters = new Array(filter);
    return art;
}
```

## blurY (GradientBevelFilter.blurY 属性)

public blurY : Number

垂直模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例演示如何在现有对象上设置 blurY 属性。

```
import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("blurYExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].blurY = 16;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
        alphas, ratios, 5, 5, 5, 3, "inner", false);

    art.filters = new Array(filter);
    return art;
}
```

## clone (GradientBevelFilter.clone 方法)

`public clone(): GradientBevelFilter`

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

`flash.filters.GradientBevelFilter` - 与原始 **GradientBevelFilter** 实例具有完全相同的属性的新 **GradientBevelFilter** 实例。

### 示例

下面的示例创建两个矩形形状。第一个矩形形状 `sourceClip` 具有斜角效果。第二个矩形形状 `resultClip` 在被单击后才会有效果。

```
import flash.filters.GradientBevelFilter;

var sourceClip:MovieClip = setUpFlatRectangle(150, 150, 0xCCCCCC,
    "cloneSourceClip");
var resultClip:MovieClip = setUpFlatRectangle(150, 150, 0xCCCCCC,
    "cloneResultClip");

resultClip.source = sourceClip;

var sourceFilter:GradientBevelFilter = getNewFilter();
sourceClip.filters = new Array(sourceFilter);

resultClip._x = 180;
resultClip.onRelease = function() {
    this.filters = new Array(this.source.filters[0].clone());
}

function setUpFlatRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    mc.beginFill(bgColor);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    return mc;
}
```

```
function getNewFilter():GradientBevelFilter {
    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    return new GradientBevelFilter(5, 225, colors, alphas, ratios, 5, 5, 5, 2,
    "inner", false);
}
```

## colors（GradientBevelFilter.colors 属性）

public colors : Array

渐变中使用的 RGB 十六进制颜色值数组。例如，红色为 0xFF0000，蓝色为 0x0000FF，依此类推。

colors 属性不能通过直接修改它的值来进行更改。相反，您必须获取对 colors 的引用，对该引用进行更改，然后将 colors 设置为该引用。

colors、alphas 和 ratios 属性都是相关的。colors 数组中的第一个元素对应于 alphas 数组中的第一个元素以及 ratios 数组中的第一个元素，依此类推。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例演示如何在现有实体上设置 colors 属性。

```
import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("colorsExample");
mc.onPress = function() {
    var arr:Array = this.filters;
    var colors:Array = [0x000000, 0xCCCCCC, 0xFFFFFFFF];
    arr[0].colors = colors;
    this.filters = arr;
}
mc.onRelease = function() {
    var arr:Array = this.filters;
    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    arr[0].colors = colors;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
    this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
```

```

        art.lineTo(w, h);
        art.lineTo(0, h);
        art.lineTo(0, 0);

        var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
        var alphas:Array = [1, 0, 1];
        var ratios:Array = [0, 128, 255];
        var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
            alphas, ratios, 5, 5, 5, 2, "inner", false);

        art.filters = new Array(filter);
        return art;
    }

```

另请参见

[alphas \(GradientBevelFilter.alphas 属性\)](#), [ratios \(GradientBevelFilter.ratios 属性\)](#)

## distance (GradientBevelFilter.distance 属性)

public distance : Number

偏移距离。默认值是 4。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例演示如何在现有对象上设置 distance 属性。

```

import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("distanceExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].distance = 1;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
}

```



```

var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
alphas, ratios, 5, 5, 5, 3, "inner", false);

art.filters = new Array(filter);
return art;
}

```

## GradientBevelFilter 构造函数

```

public GradientBevelFilter([distance:Number], [angle:Number],
    [colors:Array], [alphas:Array], [ratios:Array], [blurX:Number],
    [blurY:Number], [strength:Number], [quality:Number], [type:String],
    [knockout:Boolean])

```

用指定参数初始化滤镜。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**distance:**Number [ 可选 ] - 偏移距离。有效值为 0 到 8。默认值为 4。

**angle:**Number [ 可选 ] - 角度，以度为单位。有效值为 0 到 360。默认值为 45。

**colors:**Array [ 可选 ] - 渐变中使用的 RGB 十六进制颜色值的数组。例如，红色为 0xFF0000，蓝色为 0x0000FF，依此类推。

**alphas:**Array [ 可选 ] - colors 数组中对应颜色的 Alpha 透明度值的数组。数组中每个元素的有效值为 0 到 1。例如，.25 设置透明度值为 25%。

**ratios:**Array [ 可选 ] - 颜色分布比例的数组；有效值为 0 到 255。

**blurX:**Number [ 可选 ] - 水平模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

**blurY:**Number [ 可选 ] - 垂直模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

**strength:**Number [ 可选 ] - 印记或散布的强度。该值越高，印记的颜色越深，而且斜角与背景之间的对比度也越强。有效值为 0 到 255。值为 0 表明没有应用滤镜。默认值是 1。

**quality:**Number [ 可选 ] - 滤镜的质量。有效值为 0 到 15。默认值为 1。几乎在所有情况下，有用值都是 1（低品质）、2（中等品质）和 3（高品质）。滤镜的值越小，呈现速度越快。

**type:String** [可选] - 斜角效果的放置。可能的值包括:

- "outer": 对象外缘上的斜角
- "inner": 对象内缘上的斜角
- "full": 对象顶部的斜角

默认值为 "inner"。

**knockout:Boolean** [可选] - 指定是否应用挖空效果。值为 `true` 将使对象的填充变为透明，并显示文档的背景颜色。默认值为 `false`，即不应用挖空效果。

## 示例

下面的示例创建一个新 **GradientBevelFilter** 实例，给它分配值并将其应用于平面矩形图像。

```
import flash.filters.GradientBevelFilter;
import flash.filters.BitmapFilter;
var art:MovieClip = setUpFlatRectangle(150, 150, 0xCCCCCC,
    "gradientBevelFilterExample");
var distance:Number = 5;
var angleInDegrees:Number = 225; // opposite 45 degrees
var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 2;
var quality:Number = 3;
var type:String = "inner";
var knockout:Boolean = true;

var filter:GradientBevelFilter = new GradientBevelFilter(distance,
    angleInDegrees,
    colors,
    alphas,
    ratios,
    blurX,
    blurY,
    strength,
    quality,
    type,
    knockout);

var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;

function setUpFlatRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
```

```

        mc.beginFill(bgColor);
        mc.lineTo(w, 0);
        mc.lineTo(w, h);
        mc.lineTo(0, h);
        mc.lineTo(0, 0);
        return mc;
    }

```

另请参见

[ratios \(GradientBevelFilter.ratios 属性\)](#)

## knockout (GradientBevelFilter.knockout 属性)

```
public knockout : Boolean
```

指定对象是否具有挖空效果。应用挖空效果将使对象的填充变为透明，并显示文档的背景颜色。值为 true 将指定应用挖空效果；默认值为 false，即不应用挖空效果。

**可用性:** **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例演示如何在现有对象上设置 knockout 属性。

```

import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("knockoutExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].knockout = true;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
}

```

```

var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
alphas, ratios, 5, 5, 5, 3, "inner", false);

art.filters = new Array(filter);
return art;
}

```

## quality (GradientBevelFilter.quality 属性)

public quality : Number

应用滤镜的次数。有效值为 **0** 到 **15**。默认值为 **1**，它表示低品质。值为 **2** 表示中等品质，值为 **3** 表示高品质。滤镜的值越小，呈现速度越快。

对于大多数应用，quality 的值为 **1**、**2** 或 **3** 就足够了。您可以使用其它数值（最高为 **15**）来达到不同的效果，但是值越高，呈现速度越慢。除了增加 quality 的值，增加 blurX 和 blurY 的值通常也可以获得类似的效果，而且呈现速度更快。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例演示如何在现有对象上设置 quality 属性。

```

import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("qualityExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].quality = 1; // low quality
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];

```

```
var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
alphas, ratios, 5, 5, 5, 3, "inner", false);

art.filters = new Array(filter);
return art;
}
```

另请参见

[ratios \(GradientBevelFilter.ratios 属性\)](#)

## ratios (GradientBevelFilter.ratios 属性)

public ratios : Array

对应于 colors 数组中颜色的一组颜色分布比率。数组中每个元素的有效值为 0 到 255。

ratios 属性不能通过直接修改它的值来进行更改。相反，您必须获取对 ratios 的引用，对该引用进行更改，然后将 ratios 设置为该引用。

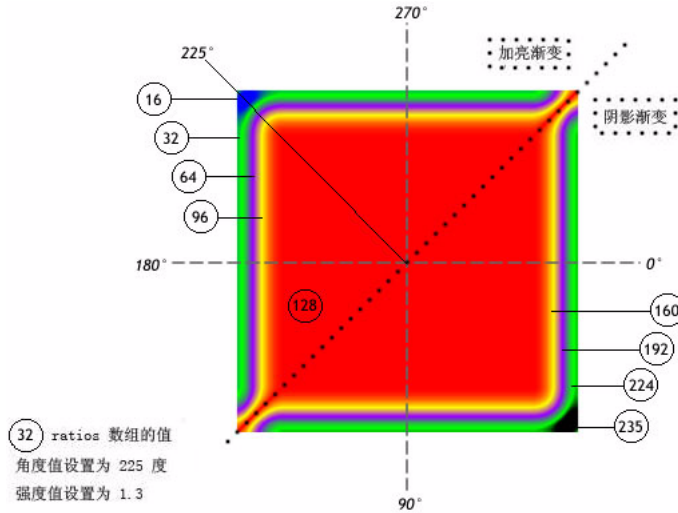
colors、alphas 和 ratios 属性都是相关的。colors 数组中的第一个元素对应于 alphas 数组中的第一个元素以及 ratios 数组中的第一个元素，依此类推。

要了解渐变斜角中的颜色是如何分布的，请先考虑您希望在渐变斜角中使用的颜色。考虑到简单斜角具有加亮颜色和阴影颜色；渐变斜角具有加亮渐变和阴影渐变。假定加亮出现在左上角，阴影出现在右下角。假定滤镜的一种可能用法：滤镜在加亮中使用四种颜色，在阴影中也使用四种颜色。除了加亮和阴影，滤镜还使用一种基本的填充颜色，这种颜色出现在加亮和阴影接合的边缘。因此所使用的颜色总数是九，比例数组中相应的元素数目也是九。

如果您将渐变看作由互相混合的各种颜色的条纹组成，则每一个比例值设置颜色在渐变半径上的位置，其中 0 表示渐变最外面的点，255 表示渐变最里面的点。对于一种典型用法，中间值为 128，这是基本的填充值。若要获得下面图像中显示的斜角效果，请使用这九种颜色的示例分配比例值，如下所示：

- 前四种颜色的范围是从 0 到 127，它们的值依次增加，以便每个值都大于或等于前一个值。这就形成了加亮斜角边缘。
- 第五种颜色（即中间的颜色）是基本填充，它设置为 128。像素值为 128 将设置基本填充，如果类型设置为 **outer**，基本填充将出现在形状外侧（以及斜角边缘周围）；或者，如果类型设置为 **inner**，基本填充将出现在形状内侧，有效地覆盖了对象自己的填充。
- 后四种颜色范围是从 129 到 255，它们的值依次增加，以便每个值都大于或等于前一个值。这就形成了阴影斜角边缘。

如果希望平均分布每个边缘的颜色，则使用奇数种颜色，其中中间的颜色为基本填充。平均分布颜色中 0 到 127 以及 129 到 255 之间的值，然后调整值以更改渐变中颜色的每个条纹的宽度。对于具有九种颜色的渐变斜角，可能的数组为 [16, 32, 64, 96, 128, 160, 192, 224, 235]。下面的图像对渐变斜角进行了描绘，如下所示：



请记住，颜色在渐变中的散布基于 blurX、blurY、strength 和 quality 属性的值以及 ratios 值。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例演示如何在现有实体上设置 ratios 属性。

```
import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("ratiosExample");
mc.onPress = function() {
    var arr:Array = this.filters;
    var ratios:Array = [127, 128, 129];
    arr[0].ratios = ratios;
    this.filters = arr;
}
mc.onRelease = function() {
    var arr:Array = this.filters;
    var ratios:Array = [0, 128, 255];
    arr[0].ratios = ratios;
    this.filters = arr;
}
```

```

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
        alphas, ratios, 5, 5, 2, "inner", false);

    art.filters = new Array(filter);
    return art;
}

```

另请参见

[alphas](#) ([GradientBevelFilter.alphas](#) 属性), [colors](#) ([GradientBevelFilter.colors](#) 属性), [beginGradientFill](#) ([MovieClip.beginGradientFill](#) 方法)

## strength (GradientBevelFilter.strength 属性)

```
public strength : Number
```

印记或跨页的强度。该值越高，印记的颜色越深，而且斜角与背景之间的对比度也越强。有效值为 0 到 255。值为 0 表明没有应用滤镜。默认值是 1。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例演示如何在现有对象上设置 strength 属性。

```

import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("strengthExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].strength = 1;
    this.filters = arr;
}

function setUpFilter(name:String):MovieClip {

```

```

var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
var w:Number = 150;
var h:Number = 150;
art.beginFill(0xCCCCCC);
art.lineTo(w, 0);
art.lineTo(w, h);
art.lineTo(0, h);
art.lineTo(0, 0);

var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
alphas, ratios, 5, 5, 5, 3, "inner", false);

art.filters = new Array(filter);
return art;
}

```

另请参见

[ratios \(GradientBevelFilter.ratios 属性\)](#)

## type (GradientBevelFilter.type 属性)

public type : String

斜角效果的放置。可能的值包括：

- "outer": 对象外缘上的斜角
- "inner": 对象内缘上的斜角
- "full": 对象顶部的斜角

默认值为 "inner"。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例演示如何在现有对象上设置 type 属性。

```

import flash.filters.GradientBevelFilter;

var mc:MovieClip = setUpFilter("typeExample");
mc.onRelease = function() {
    var arr:Array = this.filters;
    arr[0].type = "outer";
    this.filters = arr;
}

```



```

function setUpFilter(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 150;
    var h:Number = 150;
    art.beginFill(0xCCCCCC);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);

    var colors:Array = [0xFFFFFFFF, 0xCCCCCC, 0x000000];
    var alphas:Array = [1, 0, 1];
    var ratios:Array = [0, 128, 255];
    var filter:GradientBevelFilter = new GradientBevelFilter(5, 225, colors,
        alphas, ratios, 5, 5, 5, 3, "inner", false);

    art.filters = new Array(filter);
    return art;
}

```

## GradientGlowFilter (flash.filters.GradientGlowFilter)

```

Object
|
+- flash.filters.BitmapFilter
|
+- flash.filters.GradientGlowFilter

```

```

public class GradientGlowFilter
extends BitmapFilter

```

使用 **GradientGlowFilter** 类，您可以在 **Flash** 中给各种对象应用渐变发光效果。渐变发光是一种非常逼真的发光效果，您可以控制颜色渐变。可以在对象的内缘或外缘的周围或者对象的顶部应用渐变发光。

滤镜的具体使用取决于要应用滤镜的对象：

- 要在运行时给影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤销该滤镜。
- 要对 **BitmapData** 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。对 **BitmapData** 对象调用 `applyFilter()` 会获得源 **BitmapData** 对象和滤镜对象，并最终生成一个过滤图像。

您也可以在创作过程中给图像和视频应用滤镜效果。有关更多信息，请参见创作文档。

如果对影片剪辑或按钮应用滤镜，影片剪辑或按钮的 `cacheAsBitmap` 属性将设置为 `true`。如果清除所有滤镜，将恢复 `cacheAsBitmap` 的原始值。

此滤镜支持舞台缩放。但是，它不支持常规缩放、旋转和倾斜；如果对象本身进行了缩放（如果 `_xscale` 和 `_yscale` 不是 **100%**），滤镜效果将不进行缩放。只有在放大舞台时，滤镜效果才进行缩放。

如果结果图像的宽度或高度超过 **2880** 像素，则不应用滤镜。例如，如果您在放大某个大型影片剪辑时应用了滤镜，则该滤镜将在结果图像超过 **2880** 像素的限制时关闭。

可用性：ActionScript 1.0 ； Flash Player 8

另请参见

`ratios` (`GradientGlowFilter.ratios` 属性), `applyFilter` (`BitmapData.applyFilter` 方法), `cacheAsBitmap` (`Button.cacheAsBitmap` 属性), `filters` (`Button.filters` 属性), `GlowFilter` (`flash.filters.GlowFilter`), `cacheAsBitmap` (`MovieClip.cacheAsBitmap` 属性), `filters` (`MovieClip.filters` 属性), `filters` (`TextField.filters` 属性)

属性摘要

修饰符	属性	说明
	<code>alphas:Array</code>	<code>colors</code> 数组中对应颜色的 Alpha 透明度值的数组。
	<code>angle:Number</code>	角度，以度为单位。
	<code>blurX:Number</code>	水平模糊量。
	<code>blurY:Number</code>	垂直模糊量。
	<code>colors:Array</code>	定义渐变的颜色数组。
	<code>distance:Number</code>	光晕的偏移距离。
	<code>knockout:Boolean</code>	指定对象是否具有挖空效果。
	<code>quality:Number</code>	应用滤镜的次数。
	<code>ratios:Array</code>	对应于 <code>colors</code> 数组中颜色的一组颜色分布比率。
	<code>strength:Number</code>	印记或跨页的强度。
	<code>type:String</code>	滤镜效果的放置。

继承自 Object 类的属性

`constructor` (`Object.constructor` 属性), `__proto__` (`Object.__proto__` 属性), `prototype` (`Object.prototype` 属性), `__resolve` (`Object.__resolve` 属性)

构造函数摘要

签名	说明
<code>GradientGlowFilter([distance:Num ber], [angle:Number], [colors:Array], [alphas:Array], [ratios:Array], [blurX:Number], [blurY:Number], [strength:Number], [quality:Number], [type:String], [knockout:Boolean])</code>	用指定参数初始化滤镜。

方法摘要

修饰符	签名	说明
	<code>clone() : GradientGlowFilter</code>	返回此滤镜对象的副本。

继承自 `BitmapFilter` 类的方法

<a href="#">clone (BitmapFilter.clone 方法)</a>
---

继承自 `Object` 类的方法

<a href="#">addProperty (Object.addProperty 方法)</a> , <a href="#">hasOwnProperty (Object.hasOwnProperty 方法)</a> , <a href="#">isPropertyEnumerable (Object.isPropertyEnumerable 方法)</a> , <a href="#">isPrototypeOf (Object.isPrototypeOf 方法)</a> , <a href="#">registerClass (Object.registerClass 方法)</a> , <a href="#">toString (Object.toString 方法)</a> , <a href="#">unwatch (Object.unwatch 方法)</a> , <a href="#">valueOf (Object.valueOf 方法)</a> , <a href="#">watch (Object.watch 方法)</a>
---

## alphas (GradientGlowFilter.alphas 属性)

public alphas : Array

colors 数组中对应颜色的 **Alpha** 透明度值的数组。数组中每个元素的有效值为 0 到 1。例如，.25 设置 Alpha 透明度值为 25%。

alphas 属性不能通过直接修改它的值来进行更改。相反，您必须获取对 alphas 的引用，对该引用进行更改，然后将 alphas 设置为该引用。

colors、alphas 和 ratios 属性都是相关的。colors 数组中的第一个元素对应于 alphas 数组中的第一个元素以及 ratios 数组中的第一个元素，依此类推。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 alphas 属性。

```
import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowAlphas");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    var alphas:Array = filter.alphas;
    alphas.pop();
    alphas.pop();
    alphas.push(.3);
    alphas.push(1);
    filter.alphas = alphas;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;
```

```

var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

另请参见

[colors](#) ([GradientGlowFilter.colors](#) 属性), [ratios](#) ([GradientGlowFilter.ratios](#) 属性)

## angle ([GradientGlowFilter.angle](#) 属性)

`public angle : Number`

角度，以度为单位。有效值为 **0** 到 **360**。默认值为 **45**。

角度值表示理论上的光源落在对象上的角度，它决定了效果相对于该对象的位置。如果 `distance` 设置为 **0**，则效果相对于对象没有偏移，因此 `angle` 属性不起作用。

**可用性:** **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 `angle` 属性。

```

import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowAngle");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.distance = 50;
    filter.angle = 90;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
}

```

```

art._x = 20;
art._y = 20;

var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

## blurX（GradientGlowFilter.blurX 属性）

public blurX : Number

水平模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 blurX 属性。

```

import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowBlurX");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.blurX = 255;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];

```

```

var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

## blurY (GradientGlowFilter.blurY 属性)

public blurY : Number

垂直模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 blurY 属性。

```

import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowBlurY");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.blurY = 255;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;
}

```

```

var colors:Array = [0xFFFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

## clone (GradientGlowFilter.clone 方法)

public clone() : GradientGlowFilter

返回此滤镜对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.filters.GradientGlowFilter - 与原始 **GradientGlowFilter** 实例具有完全相同的属性的新 **GradientGlowFilter** 实例。

### 示例

下面的示例创建三个 **GradientGlowFilter** 对象并将他们进行比较: `filter_1` 是通过使用 **GradientGlowFilter** 构造函数创建的; `filter_2` 是通过将其设置为等效于 `filter_1` 创建的; 而 `clonedFilter` 是通过克隆 `filter_1` 创建的。请注意, 尽管 `filter_2` 等效于 `filter_1`, 但 `clonedFilter` 却不等效于 `filter_1`, 尽管它们包含相同的值。

```

import flash.filters.GradientGlowFilter;

var colors:Array = [0xFFFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter_1:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
    alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filter_2:GradientGlowFilter = filter_1;
var clonedFilter:GradientGlowFilter = filter_1.clone();

trace(filter_1 == filter_2); // true
trace(filter_1 == clonedFilter); // false

for(var i in filter_1) {
    trace(">> " + i + ": " + filter_1[i]);
    // >> clone: [type Function]
    // >> type: outer
    // >> knockout: false
}

```



```

// >> strength: 2.5
// >> quality: 2
// >> blurY: 55
// >> blurX: 55
// >> ratios: 0,63,126,255
// >> alphas: 0,1,1,1
// >> colors: 16777215,16711680,16776960,52479
// >> angle: 45
// >> distance: 0
}

for(var i in clonedFilter) {
    trace(">> " + i + ": " + clonedFilter[i]);
    // >> clone: [type Function]
    // >> type: outer
    // >> knockout: false
    // >> strength: 2.5
    // >> quality: 2
    // >> blurY: 55
    // >> blurX: 55
    // >> ratios: 0,63,126,255
    // >> alphas: 0,1,1,1
    // >> colors: 16777215,16711680,16776960,52479
    // >> angle: 45
    // >> distance: 0
}

```

为了进一步说明 filter\_1、filter\_2 和 clonedFilter 之间的关系，下面的示例将修改 filter\_1 的 knockout 属性。通过修改 knockout，说明了 clone() 方法是根据 filter\_1 的值而不是通过在引用中指向这些值来创建新实例的。

```

import flash.filters.GradientGlowFilter;

var colors:Array = [0xFFFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter_1:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
    alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filter_2:GradientGlowFilter = filter_1;
var clonedFilter:GradientGlowFilter = filter_1.clone();

trace(filter_1.knockout); // false
trace(filter_2.knockout); // false
trace(clonedFilter.knockout); // false

filter_1.knockout = true;

trace(filter_1.knockout); // true
trace(filter_2.knockout); // true
trace(clonedFilter.knockout); // false

```

## colors (GradientGlowFilter.colors 属性)

public colors : Array

定义渐变的颜色数组。例如，红色为 **0xFF0000**，蓝色为 **0x0000FF**，依此类推。

colors 属性不能通过直接修改它的值来进行更改。相反，您必须获取对 colors 的引用，对该引用进行更改，然后将 colors 设置为该引用。

colors、alphas 和 ratios 属性都是相关的。colors 数组中的第一个元素对应于 alphas 数组中的第一个元素以及 ratios 数组中的第一个元素，依此类推。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 colors 属性。

```
import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowColors");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    var colors:Array = filter.colors;
    colors.pop();
    colors.push(0xFF00FF);
    filter.colors = colors;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var colors:Array = [0xFFFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
    var alphas:Array = [0, 1, 1, 1];
    var ratios:Array = [0, 63, 126, 255];
    var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
        alphas, ratios, 55, 55, 2.5, 2, "outer", false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

另请参见

[alphas](#) ([GradientGlowFilter.alphas](#) 属性), [ratios](#) ([GradientGlowFilter.ratios](#) 属性)

## distance ([GradientGlowFilter.distance](#) 属性)

public distance : Number

光晕的偏移距离。默认值是 4。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 distance 属性。

```
import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowDistance");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.distance = 20;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var colors:Array = [0xFFFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
    var alphas:Array = [0, 1, 1, 1];
    var ratios:Array = [0, 63, 126, 255];
    var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
        alphas, ratios, 55, 55, 2.5, 2, "outer", false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

# GradientGlowFilter 构造函数

```
public GradientGlowFilter([distance:Number], [angle:Number], [colors:Array],  
    [alphas:Array], [ratios:Array], [blurX:Number], [blurY:Number],  
    [strength:Number], [quality:Number], [type:String], [knockout:Boolean])
```

用指定参数初始化滤镜。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**distance:**Number [ 可选 ] - 光晕的偏移距离。默认值为 4。

**angle:**Number [ 可选 ] - 角度，以度为单位。有效值为 0 到 360。默认值为 45。

**colors:**Array [ 可选 ] - 定义渐变的颜色的数组。例如，红色为 0xFF0000，蓝色为 0x0000FF，依此类推。

**alphas:**Array [ 可选 ] - colors 数组中对应颜色的 Alpha 透明度值的数组。数组中每个元素的有效值为 0 到 1。例如，值为 .25 设置 Alpha 透明度值为 25%。

**ratios:**Array [ 可选 ] - 颜色分布比例的数组。有效值为 0 到 255。该值定义宽度的百分比，颜色采样率为 100%。

**blurX:**Number [ 可选 ] - 水平模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

**blurY:**Number [ 可选 ] - 垂直模糊量。有效值为 0 到 255。如果模糊量小于或等于 1，则表明原始图像是按原样复制的。默认值为 4。作为 2 的乘方的值（如 2、4、8、16 和 32）经过了优化，呈现速度比其它值更快。

**strength:**Number [ 可选 ] - 印记或散布的强度。该值越高，印记的颜色越深，而且发光与背景之间的对比度也越强。有效值为 0 到 255。值越大，印记越强。值为 0 意味着未应用滤镜。默认值是 1。

**quality:**Number [ 可选 ] - 应用滤镜的次数。有效值为 0 到 15。默认值为 1，它表示低品质。值为 2 表示中等品质，值为 3 表示高品质。

**type:**String [ 可选 ] - 滤镜效果的放置。可能的值包括：

- "outer": 对象外缘上的发光
- "inner": 对象内缘上的发光；默认值
- "full": 对象顶部的发光

默认值为 "inner"。

**knockout:**Boolean [ 可选 ] - 指定对象是否具有挖空效果。应用挖空效果将使对象的填充变为透明，并显示文档的背景颜色。值为 true 将指定应用挖空效果；默认值为 false，即不应用挖空效果。

## 示例

下面的示例创建一个渐变发光滤镜，给它分配值并将其应用于平面矩形图像。

```
import flash.filters.GradientGlowFilter;
var art:MovieClip = createRectangle(100, 100, 0x003366,
    "gradientGlowFilterExample");
var distance:Number = 0;
var angleInDegrees:Number = 45;
var colors:Array = [0xFFFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var blurX:Number = 50;
var blurY:Number = 50;
var strength:Number = 2.5;
var quality:Number = 3;
var type:String = "outer";
var knockout:Boolean = false;

var filter:GradientGlowFilter = new GradientGlowFilter(distance,
    angleInDegrees,
    colors,
    alphas,
    ratios,
    blurX,
    blurY,
    strength,
    quality,
    type,
    knockout);

var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;

function createRectangle(w:Number, h:Number, bgColor:Number,
    name:String):MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    mc.beginFill(bgColor);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc._x = 20;
    mc._y = 20;
    return mc;
}
```

## knockout (GradientGlowFilter.knockout 属性)

public knockout : Boolean

指定对象是否具有挖空效果。应用挖空效果将使对象的填充变为透明，并显示文档的背景颜色。值为 true 将指定应用挖空效果；默认值为 false，即不应用挖空效果。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 knockout 属性。

```
import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowKnockout");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.knockout = true;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
    var alphas:Array = [0, 1, 1, 1];
    var ratios:Array = [0, 63, 126, 255];
    var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
        alphas, ratios, 55, 55, 2.5, 2, "outer", false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## quality (GradientGlowFilter.quality 属性)

public quality : Number

应用滤镜的次数。有效值为 **0** 到 **15**。默认值为 **1**，它表示低品质。值为 **2** 表示中等品质，值为 **3** 表示高品质。滤镜的值越小，呈现速度越快。

对于大多数应用，quality 的值为 **1**、**2** 或 **3** 就足够了。您可以使用其它数值（最高为 **15**）来达到不同的效果，但是值越高，呈现速度越慢。除了增加 quality 的值，增加 blurX 和 blurY 的值通常也可以获得类似的效果，而且呈现速度更快。

**可用性:** **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 quality 属性。

```
import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowQuality");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.quality = 3;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 100;
    art.beginFill(0x003366);
    art.lineTo(w, 0);
    art.lineTo(w, h);
    art.lineTo(0, h);
    art.lineTo(0, 0);
    art._x = 20;
    art._y = 20;

    var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
    var alphas:Array = [0, 1, 1, 1];
    var ratios:Array = [0, 63, 126, 255];
    var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
        alphas, ratios, 55, 55, 2.5, 2, "outer", false);
    var filterArray:Array = new Array();
    filterArray.push(filter);
    art.filters = filterArray;
    return art;
}
```

## ratios (GradientGlowFilter.ratios 属性)

`public ratios : Array`

对应于 `colors` 数组中颜色的一组颜色分布比率。有效值为 **0** 到 **255**。

`ratios` 属性不能通过直接修改它的值来进行更改。相反，您必须获取对 `ratios` 的引用，对该引用进行更改，然后将 `ratios` 设置为该引用。

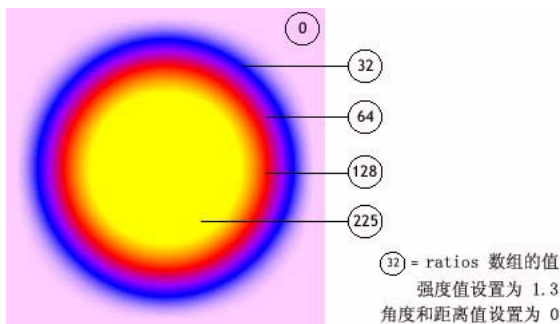
`colors`、`alphas` 和 `ratios` 属性都是相关的。`colors` 数组中的第一个元素对应于 `alphas` 数组中的第一个元素以及 `ratios` 数组中的第一个元素，依此类推。

如果 `distance` 值设置为 **0**，则将渐变发光滤镜看作从对象中心发出的具有渐变（即相互混合的颜色条纹）的发光。`colors` 数组中的第一种颜色是光晕最外面的颜色。最后一种颜色是光晕最里面的颜色。

`ratios` 数组中的每个值设置颜色在渐变的半径上的位置，其中 **0** 表示渐变最外面的点，**255** 表示渐变最里面的点。比例值范围是从 **0** 到 **255** 像素，它们的值依次增加；例如 `[0, 64, 128, 200, 255]`。从 **0** 到 **128** 的值出现在光晕的外缘上。从 **129** 到 **255** 的值出现在光晕的内侧区域中。根据颜色的比例值和滤镜的 `type` 值，滤镜颜色可能会被应用滤镜的对象遮住。

在下面的代码和图像中，将滤镜应用于黑色的圆影片剪辑，并将类型设置为 `"full"`。基于教学目的，`colors` 数组中的第一种颜色（粉色）的 `alpha` 值为 **1**，以便与白色文档背景形成鲜明对比。（实际上，您可能不希望这样显示第一种颜色。）请注意，数组中的最后一种颜色（黄色）遮住了应用滤镜的黑色圆盘：

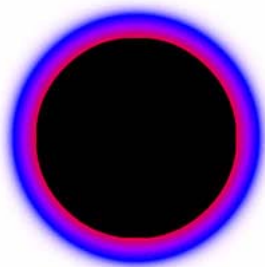
```
var colors = [0xFFCCFF, 0x0000FF, 0x9900FF, 0xFF0000, 0xFFFF00];  
var alphas = [1, 1, 1, 1, 1]; var ratios = [0, 32, 64, 128, 225];  
var myGGF = new GradientGlowFilter(0, 0, colors, alphas, ratios, 50, 50, 1,  
2, "full", false);
```



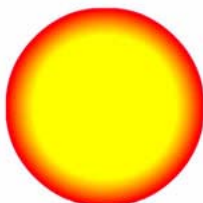
要在将 `type` 值设置为 `"outer"` 或 `"full"` 时实现与文档背景的无缝效果，请将数组中的第一种颜色设置为与文档背景相同的颜色，或将第一种颜色的 `Alpha` 值设置为 **0**；或者使用技巧将滤镜与背景混合在一起。



如果在代码中进行两处小的更改，发光效果可能会大不相同，即使采用相同的 `ratios` 和 `colors` 数组。将数组中第一种颜色的 **Alpha** 值设置为 **0**，以便滤镜和文档的白色背景混合在一起；并将 `type` 属性设置为 "outer" 或 "inner"。观察结果，如下面的图像所示。



外部发光



内部发光

请记住，颜色在渐变中的散布基于 `blurX`、`blurY`、`strength` 和 `quality` 属性的值以及 `ratios` 值。

可用性：ActionScript 1.0 ； Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 `ratios` 属性。

```
import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowRatios");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    var ratios:Array = filter.ratios;
    ratios.shift();
    ratios.unshift(40);
    filter.ratios = ratios;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {
    var art:MovieClip = this.createEmptyMovieClip(name,
        this.getNextHighestDepth());
```

```

var w:Number = 100;
var h:Number = 100;
art.beginFill(0x003366);
art.lineTo(w, 0);
art.lineTo(w, h);
art.lineTo(0, h);
art.lineTo(0, 0);
art._x = 20;
art._y = 20;

var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

另请参见

[colors](#) ([GradientGlowFilter.colors](#) 属性), [alphas](#) ([GradientGlowFilter.alphas](#) 属性), [beginGradientFill](#) ([MovieClip.beginGradientFill](#) 方法)

## strength ([GradientGlowFilter.strength](#) 属性)

public strength : Number

印记或跨页的强度。该值越高，印记的颜色越深，而且发光与背景之间的对比度也越强。有效值为 **0** 到 **255**。值为 **0** 表明没有应用滤镜。默认值是 **1**。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 `strength` 属性。

```

import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowStrength");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.strength = 1;
    this.filters = new Array(filter);
}

```

```

function createGradientGlowRectangle(name:String):MovieClip {

```

```

var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
var w:Number = 100;
var h:Number = 100;
art.beginFill(0x003366);
art.lineTo(w, 0);
art.lineTo(w, h);
art.lineTo(0, h);
art.lineTo(0, 0);
art._x = 20;
art._y = 20;

var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

## type (GradientGlowFilter.type 属性)

public type : String

滤镜效果的放置。可能的值包括：

- "outer": 对象外缘上的发光
- "inner": 对象内缘上的发光：默认值
- "full": 对象顶部的发光

默认值为 "inner"。

可用性：ActionScript 1.0 ； Flash Player 8

### 示例

下面的示例在用户单击现有的影片剪辑时更改其 type 属性。

```

import flash.filters.GradientGlowFilter;
var mc:MovieClip = createGradientGlowRectangle("GlowType");
mc.onRelease = function() {
    var filter:GradientGlowFilter = this.filters[0];
    filter.type = "inner";
    filter.strength = 1;
    this.filters = new Array(filter);
}

function createGradientGlowRectangle(name:String):MovieClip {

```

```

var art:MovieClip = this.createEmptyMovieClip(name,
this.getNextHighestDepth());
var w:Number = 100;
var h:Number = 100;
art.beginFill(0x003366);
art.lineTo(w, 0);
art.lineTo(w, h);
art.lineTo(0, h);
art.lineTo(0, 0);
art._x = 20;
art._y = 20;

var colors:Array = [0xFFFFFF, 0xFF0000, 0xFFFF00, 0x00CCFF];
var alphas:Array = [0, 1, 1, 1];
var ratios:Array = [0, 63, 126, 255];
var filter:GradientGlowFilter = new GradientGlowFilter(0, 45, colors,
alphas, ratios, 55, 55, 2.5, 2, "outer", false);
var filterArray:Array = new Array();
filterArray.push(filter);
art.filters = filterArray;
return art;
}

```

## IME (System.IME)

```

Object
|
+-System.IME

```

```

public class IME
extends Object

```

使用 **IME** 类，您可以在客户端计算机上运行的 **Flash Player** 应用程序中直接操纵操作系统的输入法编辑器 (IME)。您可以确定是否已安装 IME、是否已启用 IME 以及启用了哪个 IME。您可以在 **Flash Player** 应用程序中禁用或启用 IME，并且可以执行其它受限制的函数，具体取决于操作系统。

利用输入法编辑器，用户可以键入亚洲语言（例如，中文、日语和朝鲜语）的非 ASCII 文本字符。有关 IME 的更多信息，请参见您要为其开发应用程序的平台的操作系统的文档。下面列出了有关输入法信息的一些其它资源：

- <http://www.microsoft.com/globaldev/default.msp>
- <http://developer.apple.com/documentation/>
- <http://java.sun.com>

下表显示了此类适用的平台范围：

功能	Windows	Macintosh OSX	Macintosh 经典	Linux / Solaris XIM
确定是否已安装 IME System.capabilities.hasIME	是	是	是	是
设置是启用 IME 还是禁用 IME System.IME.setEnabled()	是	是	是	是
查明是启用 IME 还是禁用 IME System.IME.getEnabled()	是	是	是	是
设置 IME 转换模式 System.IME.setConversionMode()	是	是 **	否	是
获取 IME 转换模式 System.IME.getConversionMode()	是	是 **	否	是
向 IME 发送要转换的字符串 System.IME.setPositionString()	是 *	否	否	是
在转换之前从 IME 获取原始字符串 System.IME.addListener() listener.onIMEComposition() System.IME.removeListener()	是 *	否	否	是
将转换请求发送到 IME System.IME.doConversion()	是 *	否	否	是

\* 只有部分 Windows IME 支持上述全部操作。到目前为止，唯一支持全部操作的 IME 是日文 IME。每一种 IME 在其对 OS 调用的支持方面都有所不同。

\*\* 在 Macintosh 上，只有日文支持这些方法，第三方 IME 不支持这些方法。

可用性：ActionScript 1.0 ； Flash Player 8

属性摘要

修饰符	属性	说明
static	ALPHANUMERIC_FULL:String	一个字符串，其值为 "ALPHANUMERIC_FULL"，用于 setConversionMode() 和 getConversionMode()。
static	ALPHANUMERIC_HALF:String	一个字符串，其值为 "ALPHANUMERIC_HALF"，用于 setConversionMode() 和 getConversionMode()。
static	CHINESE:String	一个字符串，其值为 "CHINESE"，用于 setConversionMode() 和 getConversionMode()。
static	JAPANESE_HIRAGANA:String	一个字符串，其值为 "JAPANESE_HIRAGANA"，用于 setConversionMode() 和 getConversionMode()。
static	JAPANESE_KATAKANA_FULL:String	一个字符串，其值为 "JAPANESE_KATAKANA_FULL"，用于 setConversionMode() 和 getConversionMode()。
static	JAPANESE_KATAKANA_HALF:String	一个字符串，其值为 "JAPANESE_KATAKANA_HALF"，用于 setConversionMode() 和 getConversionMode()。
static	KOREAN:String	一个字符串，其值为 "KOREAN"，用于 setConversionMode() 和 getConversionMode()。
static	UNKNOWN:String	一个字符串，具有用于 getConversionMode() 的值 "UNKNOWN"。

继承自 Object 类的属性

<a href="#">constructor</a> (Object.constructor 属性), <a href="#">__proto__</a> (Object.__proto__ 属性), <a href="#">prototype</a> (Object.prototype 属性), <a href="#">__resolve</a> (Object.__resolve 属性)
--

事件摘要

事件	说明
onIMEComposition = function([readingString:String]) {}	正在设置 IME 合成字符串时获得通知。

方法摘要

修饰符	签名	说明
static	addListener(listener:Object) : Void	注册一个对象，以便在 onIMEComposition 事件调用 IME 事件处理函数时接收通知。
static	doConversion() : Boolean	指示 IME 为当前复合字符串选择第一个候选项。
static	getConversionMode() : String	指示当前 IME 的转换模式。
static	getEnabled() : Boolean	指示是否启用系统 IME。
static	removeListener(listener:Object) : Boolean	删除一个以前使用 IME.addListener() 注册到 IME 实例的侦听器对象。
static	setCompositionString(composition:String) : Boolean	设置 IME 复合字符串。
static	setConversionMode(mode:String) : Boolean	设置当前 IME 的转换模式。
static	setEnabled(enabled:Boolean) : Boolean	启用或禁用系统 IME。

继承自 Object 类的方法

<a href="#">addProperty</a> ( <a href="#">Object.addProperty</a> 方法), <a href="#">hasOwnProperty</a> ( <a href="#">Object.hasOwnProperty</a> 方法), <a href="#">isPrototypeOf</a> ( <a href="#">Object.isPrototypeOf</a> 方法), <a href="#">isPrototypeOf</a> ( <a href="#">Object.isPrototypeOf</a> 方法), <a href="#">registerClass</a> ( <a href="#">Object.registerClass</a> 方法), <a href="#">toString</a> ( <a href="#">Object.toString</a> 方法), <a href="#">unwatch</a> ( <a href="#">Object.unwatch</a> 方法), <a href="#">valueOf</a> ( <a href="#">Object.valueOf</a> 方法), <a href="#">watch</a> ( <a href="#">Object.watch</a> 方法)
--

## addListener (IME.addListener 方法)

public static addListener(listener:Object) : Void

注册一个对象，以便在 onIMEComposition 事件调用 IME 事件处理函数时接收通知。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**listener:Object** - 具有 onIMEComposition (**readingString**) 方法的对象，它侦听来自 IME 事件处理函数的回调通知。传递给此方法的读取字符串将位于 IME 的复合模式中。例如，如果用户输入平假名文本，然后选择日文汉字候选项，则读取字符串为原始平假名。

### 示例

下面的示例演示如何向在用户通过单击文本字段设置合成字符串时输出通知的 System.IME 添加侦听器对象。

```
var IMEListener:Object = new Object();
IMEListener.onIMEComposition = function(str:String) {
    trace(">> onIMEComposition: " + str);
}
System.IME.addListener(IMEListener);
trace(System.IME.length);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.createTextField("txt", this.getNextHighestDepth(), 0, 0, 0, 0);
mc.txt.border = true;
mc.txt.background = true;
mc.txt.autoSize = "left";
mc.txt.text = "Click this text to add a listener.";

mc.onPress = function() {
    if(System.capabilities.hasIME) {
        Selection.setFocus(mc.txt);
        System.IME.setCompositionString(mc.txt.text);
    }
}
```



## ALPHANUMERIC\_FULL (IME.ALPHANUMERIC\_FULL 属性)

`public static ALPHANUMERIC_FULL : String`

一个字符串，其值为 "ALPHANUMERIC\_FULL"，用于 `setConversionMode()` 和 `getConversionMode()`。此常数用于所有 IME。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 `ALPHANUMERIC_FULL`。

```
if(System.capabilities.hasIME) {
    trace(System.IME.getConversionMode());

    System.IME.setConversionMode(System.IME.ALPHANUMERIC_FULL);
    trace(System.IME.getConversionMode());
}
```

另请参见

[setConversionMode \(IME.setConversionMode 方法\)](#), [getConversionMode \(IME.getConversionMode 方法\)](#), [hasIME \(capabilities.hasIME 属性\)](#)

## ALPHANUMERIC\_HALF (IME.ALPHANUMERIC\_HALF 属性)

`public static ALPHANUMERIC_HALF : String`

一个字符串，其值为 "ALPHANUMERIC\_HALF"，用于 `setConversionMode()` 和 `getConversionMode()`。此常数用于所有 IME。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 `ALPHANUMERIC_HALF`。

```
if(System.capabilities.hasIME) {
    trace(System.IME.getConversionMode());

    System.IME.setConversionMode(System.IME.ALPHANUMERIC_HALF);
    trace(System.IME.getConversionMode());
}
```

另请参见

[setConversionMode \(IME.setConversionMode 方法\)](#), [getConversionMode \(IME.getConversionMode 方法\)](#)

## CHINESE (IME.CHINESE 属性)

`public static CHINESE : String`

一个字符串，其值为 "CHINESE"，用于 `setConversionMode()` 和 `getConversionMode()`。  
此常数用于简体中文 IME 和繁体中文 IME。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 CHINESE。

```
if(System.capabilities.hasIME) {  
    trace(System.IME.getConversionMode());  
  
    System.IME.setConversionMode(System.IME.CHINESE);  
    trace(System.IME.getConversionMode());  
}
```

另请参见

[setConversionMode \(IME.setConversionMode 方法\)](#), [getConversionMode \(IME.getConversionMode 方法\)](#)

## doConversion (IME.doConversion 方法)

`public static doConversion() : Boolean`

指示 IME 为当前复合字符串选择第一个候选项。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 返回

Boolean - 如果调用成功，则返回 `true`；否则返回 `false`。

### 示例

下面的示例演示如何选择 IME 合成字符串的第一个候选项。如果用户的系统安装了 IME，则可以通过单击文本字段选择候选项。

```
var mc:MovieClip = this.createEmptyMovieClip("mc",  
    this.getNextHighestDepth());
```

```
mc.createTextField("txt", this.getNextHighestDepth(), 0, 0, 0, 0);
mc.txt.border = true;
mc.txt.background = true;
mc.txt.autoSize = "left";
mc.txt.text = "Set this text as the composition string and convert it.";

mc.onPress = function() {
    if(System.capabilities.hasIME) {
        Selection.setFocus(mc.txt);
        System.IME.setCompositionString(mc.txt.text);
        trace(System.IME.doConversion());
    }
}
```

## getConversionMode (IME.getConversionMode 方法)

public static getConversionMode() : String

指示当前 IME 的转换模式。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 返回

String - 转换模式。可能的值为指示转换模式的 IME 模式字符串常数:

- ALPHANUMERIC\_FULL
- ALPHANUMERIC\_HALF
- CHINESE
- JAPANESE\_HIRAGANA
- JAPANESE\_KATAKANA\_FULL
- JAPANESE\_KATAKANA\_HALF
- KOREAN
- UNKNOWN

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 ( System.capabilities.hasIME) 获取 IME。

```
var mode:String = System.IME.UNKNOWN;
if(System.capabilities.hasIME) {
    mode = System.IME.getConversionMode();
}
trace(mode);
```

另请参见

[ALPHANUMERIC\\_FULL](#) ([IME.ALPHANUMERIC\\_FULL](#) 属性), [ALPHANUMERIC\\_HALF](#) ([IME.ALPHANUMERIC\\_HALF](#) 属性), [CHINESE](#) ([IME.CHINESE](#) 属性), [JAPANESE\\_HIRAGANA](#) ([IME.JAPANESE\\_HIRAGANA](#) 属性), [JAPANESE\\_KATAKANA\\_FULL](#) ([IME.JAPANESE\\_KATAKANA\\_FULL](#) 属性), [JAPANESE\\_KATAKANA\\_HALF](#) ([IME.JAPANESE\\_KATAKANA\\_HALF](#) 属性), [KOREAN](#) ([IME.KOREAN](#) 属性), [UNKNOWN](#) ([IME.UNKNOWN](#) 属性)

## getEnabled (IME.getEnabled 方法)

```
public static getEnabled(): Boolean
```

指示是否启用系统 IME。启用的 IME 执行多字节输入；禁用的 IME 执行字母数字输入。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

Boolean - 如果启用系统 IME，则返回 true；如果禁用系统 IME，则返回 false。

### 示例

下面的示例通过调用 `isEnabled()` 方法来检查是否启用了 IME。

```
if(System.capabilities.hasIME) {  
    var isImeEnabled:Boolean = System.IME.getEnabled();  
    trace(isImeEnabled);  
}
```

## JAPANESE\_HIRAGANA (IME.JAPANESE\_HIRAGANA 属性)

```
public static JAPANESE_HIRAGANA : String
```

一个字符串，其值为 "JAPANESE\_HIRAGANA"，用于 `setConversionMode()` 和 `getConversionMode()`。此常数用于日文 IME。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 JAPANESE\_HIRAGANA。

```
if(System.capabilities.hasIME) {  
    trace(System.IME.getConversionMode());  
}
```

```
System.IME.setConversionMode(System.IME.JAPANESE_HIRAGANA);
trace(System.IME.getConversionMode());
}
```

另请参见

[setConversionMode \(IME.setConversionMode 方法\)](#), [getConversionMode \(IME.getConversionMode 方法\)](#)

## JAPANESE\_KATAKANA\_FULL (IME.JAPANESE\_KATAKANA\_FULL 属性)

```
public static JAPANESE_KATAKANA_FULL : String
```

一个字符串，其值为 "JAPANESE\_KATAKANA\_FULL"，用于 `setConversionMode()` 和 `getConversionMode()`。此常数用于日文 IME。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 **IME** 设置为 `JAPANESE_KATAKANA_FULL`。

```
if(System.capabilities.hasIME) {
    trace(System.IME.getConversionMode());

    System.IME.setConversionMode(System.IME.JAPANESE_KATAKANA_FULL);
    trace(System.IME.getConversionMode());
}
```

另请参见

[setConversionMode \(IME.setConversionMode 方法\)](#), [getConversionMode \(IME.getConversionMode 方法\)](#)

## JAPANESE\_KATAKANA\_HALF (IME.JAPANESE\_KATAKANA\_HALF 属性)

`public static JAPANESE_KATAKANA_HALF : String`

一个字符串，其值为 "JAPANESE\_KATAKANA\_HALF"，用于 `setConversionMode()` 和 `getConversionMode()`。此常数用于日文 IME。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 JAPANESE\_KATAKANA\_HALF。

```
if(System.capabilities.hasIME) {  
    trace(System.IME.getConversionMode());  
  
    System.IME.setConversionMode(System.IME.JAPANESE_KATAKANA_HALF);  
    trace(System.IME.getConversionMode());  
}
```

另请参见

[setConversionMode \(IME.setConversionMode 方法\)](#)，[getConversionMode \(IME.getConversionMode 方法\)](#)

## KOREAN (IME.KOREAN 属性)

`public static KOREAN : String`

一个字符串，其值为 "KOREAN"，用于 `setConversionMode()` 和 `getConversionMode()`。此常数用于朝鲜语 IME。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 KOREAN。

```
if(System.capabilities.hasIME) {  
    trace(System.IME.getConversionMode());  
  
    System.IME.setConversionMode(System.IME.KOREAN);  
    trace(System.IME.getConversionMode());  
}
```

另请参见

`setConversionMode` (`IME.setConversionMode` 方法), `getConversionMode` (`IME.getConversionMode` 方法)

## onIMEComposition (IME.onIMEComposition 事件侦听器)

```
onIMEComposition = function([readingString:String]) {}
```

正在设置 **IME** 合成字符串时获得通知。若要使用此侦听器, 您必须创建一个侦听器对象。然后可以为此侦听器定义一个函数, 并使用 `IME.addListener()` 向 **IME** 对象注册此侦听器, 如下列代码所示:

```
var someListener:Object = new Object();
someListener.onIMEComposition = function () {
    // statements
}
System.IME.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作, 因为多个侦听器可以接收有关单个事件的通知。

**可用性:** **ActionScript 1.0** ; **Flash Player 8**

### 参数

**readingString:String** [ 可选 ] - 在用户开始选择候选项之前向 **IME** 中键入的原始文本。

### 示例

下面的示例演示如何使用回调方法 `onIMEComposition()` 将侦听器对象添加到 `System.IME`, 它在用户通过单击文本字段设置合成字符串时输出通知。

```
var IMEListener:Object = new Object();
IMEListener.onIMEComposition = function(str:String) {
    trace(">> onIMEComposition: " + str);
}
System.IME.addListener(IMEListener);
trace(System.IME.length);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.createTextField("txt", this.getNextHighestDepth(), 0, 0, 0, 0);
mc.txt.border = true;
mc.txt.background = true;
mc.txt.autoSize = "left";
mc.txt.text = "Click this text to add a listener.";

mc.onPress = function() {
    if(System.capabilities.hasIME) {
```

```

        Selection.setFocus(mc.txt);
        System.IME.setCompositionString(mc.txt.text);
    }
}

```

另请参见

[addListener \(IME.addListener 方法\)](#), [setCompositionString \(IME.setCompositionString 方法\)](#)

## removeListener (IME.removeListener 方法)

```
public static removeListener(listener:Object) : Boolean
```

删除一个以前使用 `IME.addListener()` 注册到 `IME` 实例的侦听器对象。

可用性: **ActionScript 1.0 ; Flash Player 8**

参数

**listener:Object** - 不再接收来自 `IME` 事件处理函数的回调通知的对象。

返回

**Boolean** - 如果已删除侦听器对象, 则返回 `true`; 否则返回 `false`。

示例

下面的示例演示如何在用户通过单击文本字段设置合成字符串时从 `System.IME` 删除侦听器对象。

```

var IMEListener:Object = new Object();
IMEListener.onIMEComposition = function(str:String) {
    trace(">> onIMEComposition: " + str);

    System.IME.removeListener(this);
    trace(System.IME.length); // 0
}
System.IME.addListener(IMEListener);
trace(System.IME.length); // 1

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.createTextField("txt", this.getNextHighestDepth(), 0, 0, 0, 0);
mc.txt.border = true;
mc.txt.background = true;
mc.txt.autoSize = "left";
mc.txt.text = "Click this text to add and remove a listener.";

mc.onPress = function() {

```



```

        if(System.capabilities.hasIME) {
            Selection.setFocus(mc.txt);
            System.IME.setCompositionString(mc.txt.text);
        }
    }
}

```

## setCompositionString (IME.setCompositionString 方法)

```
public static setCompositionString(composition:String) : Boolean
```

设置 **IME** 复合字符串。设置了此字符串后，用户就可以在将结果提交到当前具有焦点的文本字段之前选择 **IME** 候选项。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**composition:String** - 要发送到 **IME** 的字符串。

### 返回

**Boolean** - 如果成功设置 **IME** 合成字符串，则返回 **true**。如果没有任何文本字段具有焦点，此方法将失败并返回 **false**。

### 示例

下面的示例演示如何设置 **IME** 合成字符串。如果用户的系统安装了 **IME**，则可以通过单击文本字段显示 **IME** 选项。

```

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.createTextField("txt", this.getNextHighestDepth(), 0, 0, 0, 0);
mc.txt.border = true;
mc.txt.background = true;
mc.txt.autoSize = "left";
mc.txt.text = "Set this text as the composition string.";

mc.onPress = function() {
    if(System.capabilities.hasIME) {
        Selection.setFocus(mc.txt);
        trace(System.IME.setCompositionString(mc.txt.text));
    }
}

```

## setConversionMode (IME.setConversionMode 方法)

`public static setConversionMode(mode:String) : Boolean`

设置当前 IME 的转换模式。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**mode:String** - 转换模式。可能的值为 IME 模式字符串常数:

- `ALPHANUMERIC_FULL`
- `ALPHANUMERIC_HALF`
- `CHINESE`
- `JAPANESE_HIRAGANA`
- `JAPANESE_KATAKANA_FULL`
- `JAPANESE_KATAKANA_HALF`
- `KOREAN`

### 返回

`Boolean` - 如果成功设置转换模式, 则返回 `true`; 否则返回 `false`。

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 ( `System.capabilities.hasIME` ) 获取 IME 并将变量 `mode` 设置为该值。

```
var mode:String = System.IME.UNKNOWN;
if(System.capabilities.hasIME) {
    mode = System.IME.getConversionMode();
}
System.IME.setConversionMode(mode);
trace(System.IME.getConversionMode());
```

### 另请参见

[ALPHANUMERIC\\_FULL](#) (IME.ALPHANUMERIC\_FULL 属性), [ALPHANUMERIC\\_HALF](#) (IME.ALPHANUMERIC\_HALF 属性), [CHINESE](#) (IME.CHINESE 属性), [JAPANESE\\_HIRAGANA](#) (IME.JAPANESE\_HIRAGANA 属性), [JAPANESE\\_KATAKANA\\_FULL](#) (IME.JAPANESE\_KATAKANA\_FULL 属性), [JAPANESE\\_KATAKANA\\_HALF](#) (IME.JAPANESE\_KATAKANA\_HALF 属性), [KOREAN](#) (IME.KOREAN 属性)

## setEnabled (IME.setEnabled 方法)

`public static setEnabled(enabled:Boolean) : Boolean`

启用或禁用系统 IME。启用的 IME 执行多字节输入；禁用的 IME 执行字母数字输入。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**enabled:Boolean** - 设置为 true 可以启用系统 IME，设置为 false 可以禁用它。

### 返回

Boolean - 如果尝试启用系统 IME 成功，则为 true；否则为 false。

### 示例

下面的示例通过调用 `isEnabled()` 方法来检查是否启用了 IME 并通过调用 `setEnabled()` 方法将 IME 的启用状态更改为相反的状态。

```
if(System.capabilities.hasIME) {
    var isImeEnabled:Boolean = System.IME.isEnabled();
    trace(isImeEnabled);

    if(isImeEnabled) {
        System.IME.setEnabled(false);
    }
    else {
        System.IME.setEnabled(true);
    }

    var isImeEnabled:Boolean = System.IME.isEnabled();
    trace(isImeEnabled);
}
```

## UNKNOWN (IME.UNKNOWN 属性)

`public static UNKNOWN : String`

一个字符串，具有用于 `getConversionMode()` 的值 "UNKNOWN"。此常数用于所有 IME。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例在系统安装了输入法编辑器 (IME) 的情况下 (`System.capabilities.hasIME`) 将 IME 设置为 UNKNOWN。

```
if(System.capabilities.hasIME) {  
    trace(System.IME.getConversionMode());  
  
    System.IME.setConversionMode(System.IME.UNKNOWN);  
    trace(System.IME.getConversionMode());  
}
```

另请参见

[getConversionMode \(IME.getConversionMode 方法\)](#)

## Key

Object  
|  
+-Key

```
public class Key  
extends Object
```

**Key** 类是不通过构造函数即可使用其方法和属性的顶级类。使用 **Key** 类的方法可生成用户能够通过标准键盘控制的界面。**Key** 类的属性是一些常数，这些常数表示用于控制应用程序的常用键（如箭头键、**Page Up** 和 **Page Down**）。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性: **ActionScript 1.0** ; **Flash Player 6**

属性摘要

修饰符	属性	说明
static	BACKSPACE:Number	Backspace 键的键控代码值 (8)。
static	CAPSLock:Number	Caps Lock 键的键控代码值 (20)。
static	CONTROL:Number	Ctrl 键的键控代码值 (17)。
static	DELETEKEY:Number	Delete 键的键控代码值 (46)。
static	DOWN:Number	下箭头键的键控代码值 (40)。
static	END:Number	End 键的键控代码值 (35)。
static	ENTER:Number	Enter 键的键控代码值 (13)。
static	ESCAPE:Number	Esc 键的键控代码值 (27)。
static	HOME:Number	Home 键的键控代码值 (36)。
static	INSERT:Number	Insert 键的键控代码值 (45)。
static	LEFT:Number	左箭头键的键控代码值 (37)。
static	_listeners:Array [ 只读 ]	一个引用列表，引用对象是向 Key 对象注册的所有侦听器对象。
static	PGDN:Number	Page Down 键的键控代码值 (34)。
static	PGUP:Number	Page Up 键的键控代码值 (33)。
static	RIGHT:Number	右箭头键的键控代码值 (39)。
static	SHIFT:Number	Shift 键的键控代码值 (16)。
static	SPACE:Number	空格键的键控代码值 (32)。
static	TAB:Number	Tab 键的键控代码值 (9)。
static	UP:Number	上箭头键的键控代码值 (38)。

继承自 Object 类的属性

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

事件摘要

事件	说明
onKeyDown = function() {}	当按下某按键时获得通知。
onKeyUp = function() {}	当释放某按键时获得通知。

方法摘要

修饰符	签名	说明
static	addListener(listener:Object) : Void	注册一个对象，以便接收 onKeyDown 和 onKeyUp 通知。
static	getAscii() : Number	返回按下或释放的最后一个键的 ASCII 码。
static	getCode() : Number	返回按下的最后一个键的键控代码值。
static	isAccessible() : Boolean	根据安全限制返回一个布尔值，该值指示按下的最后一个键是否可以被其它 SWF 文件访问。
static	isDown(code:Number) : Boolean	如果按下 <b>keycode</b> 中指定的键，则返回 true；否则返回 false。
static	isToggled(code:Number) : Boolean	如果激活 Caps Lock 或 Num Lock 键（切换到活动状态），则返回 true；否则返回 false。
static	removeListener(listener:Object) : Boolean	删除以前用 Key.addListener() 注册的对象。

继承自 Object 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## addListener (Key.addListener 方法)

public static addListener(listener:Object) : Void

注册一个对象，以便接收 onKeyDown 和 onKeyUp 通知。当按下或释放某个键时，不管输入焦点情况如何，所有用 addListener() 注册的侦听对象都将调用其 onKeyDown 方法或 onKeyUp 方法。可以有多个对象侦听键盘通知。如果已经注册了侦听器 *newListener*，则不会发生任何更改。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性: **ActionScript 1.0**；Flash Player 6

### 参数

**listener:Object** - 具有 onKeyDown 和 onKeyUp 方法的对象。

## 示例

下面的示例创建新的侦听器对象，并为 `onKeyDown` 和 `onKeyUp` 定义函数。最后一行使用 `addListener()` 向 **Key** 对象注册该侦听器，以使该对象可接收 **key down** 和 **key up** 事件的通知。

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
myListener.onKeyUp = function () {
    trace ("You released a key.");
}
Key.addListener(myListener);
```

下面的示例将快捷键 **Ctrl+7** 分配给实例名为 `my_btn` 的按钮，并向屏幕阅读器（请参见 `_accProps`）提供有关该快捷键的信息。在此示例中，当您按 **Ctrl+7** 时，`myOnPress` 函数就在“输出”面板上显示 `hello` 文本。

```
function myOnPress() {
    trace("hello");
}
function myOnKeyDown() {
    // 55 is key code for 7
    if (Key.isDown(Key.CONTROL) && Key.getCode() == 55) {
        Selection.setFocus(my_btn);
        my_btn.onPress();
    }
}
var myListener:Object = new Object();
myListener.onKeyDown = myOnKeyDown;
Key.addListener(myListener);
my_btn.onPress = myOnPress;
my_btn._accProps.shortcut = "Ctrl+7";
Accessibility.updateProperties();
```

## 另请参见

[getCode](#) ([Key.getCode](#) 方法), [isDown](#) ([Key.isDown](#) 方法), [onKeyDown](#) ([Key.onKeyDown](#) 事件侦听器), [onKeyUp](#) ([Key.onKeyUp](#) 事件侦听器), [removeListener](#) ([Key.removeListener](#) 方法)

## BACKSPACE (Key.BACKSPACE 属性)

`public static BACKSPACE : Number`

Backspace 键的键控代码值 (8)。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例创建新的侦听器对象，并为 `onKeyDown` 定义函数。最后一行使用 `addListener()` 向 **Key** 对象注册侦听器，以便该对象可接收 **key down** 事件的通知。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.BACKSPACE)) {
        trace("you pressed the Backspace key.");
    } else {
        trace("you DIDN'T press the Backspace key.");
    }
};
Key.addListener(keyListener);
```

使用此示例时，请确保在测试环境中选择 “控制” > “禁用快捷键”。

## CAPSLOCK (Key.CAPSLOCK 属性)

`public static CAPSLOCK : Number`

Caps Lock 键的键控代码值 (20)。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例创建新的侦听器对象，并为 `onKeyDown` 定义函数。最后一行使用 `addListener()` 向 **Key** 对象注册侦听器，以便该对象可接收 **key down** 事件的通知。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.CAPSLOCK)) {
        trace("you pressed the Caps Lock key.");
        trace("\tCaps Lock == "+Key.isToggled(Key.CAPSLOCK));
    }
};
Key.addListener(keyListener);
```

当您按 **Caps Lock** 键时，将在“输出”面板中显示信息。“输出”面板显示 `true` 或 `false`，具体取决于是否使用 `isToggled` 方法激活了 **Caps Lock** 键。



## CONTROL (Key.CONTROL 属性)

public static CONTROL : Number

Ctrl 键的键控代码值 (17)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

下面的示例将快捷键 **Ctrl+7** 分配给实例名为 `my_btn` 的按钮，并向屏幕阅读器（请参见 `_accProps`）提供有关该快捷键的信息。在此示例中，当您按 **Ctrl+7** 时，`myOnPress` 函数就在“输出”面板上显示 `hello` 文本。

```
function myOnPress() {
    trace("hello");
}
function myOnKeyDown() {
    // 55 is key code for 7
    if (Key.isDown(Key.CONTROL) && Key.getCode() == 55) {
        Selection.setFocus(my_btn);
        my_btn.onPress();
    }
}
var myListener:Object = new Object();
myListener.onKeyDown = myOnKeyDown;
Key.addListener(myListener);
my_btn.onPress = myOnPress;
my_btn._accProps.shortcut = "Ctrl+7";
Accessibility.updateProperties();
```

## DELETEKEY (Key.DELETEKEY 属性)

public static DELETEKEY : Number

Delete 键的键控代码值 (46)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

在下面的示例中，您可以使用 **Drawing API** 和侦听器对象通过鼠标指针绘制线条。按 **Backspace** 键或 **Delete** 键可删除您绘制的线条。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.drawing = true;
    canvas_mc.moveTo(_xmouse, _ymouse);
    canvas_mc.lineStyle(3, 0x99CC00, 100);
};
```

```

mouseListener.onMouseUp = function() {
    this.drawing = false;
};
mouseListener.onMouseMove = function() {
    if (this.drawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
Mouse.addListener(mouseListener);
//
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.DELETEKEY) || Key.isDown(Key.BACKSPACE)) {
        canvas_mc.clear();
    }
};
Key.addListener(keyListener);

```

使用此示例时，请确保在测试环境中选择“控制”>“禁用快捷键”。

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## DOWN (Key.DOWN 属性)

```
public static DOWN : Number
```

下箭头键的键控代码值 (40)。

**可用性:** ActionScript 1.0 ; Flash Player 5

### 示例

在下面的示例中，当您按箭头键时，名为 `car_mc` 的影片剪辑将移动一段固定距离 (10)。按空格键时将播放声音。在此示例中，为库中的声音指定链接标识符 `horn_id`。

```

var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :

```

```

        car_mc._y -= DISTANCE;
        break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);

```

## END (Key.END 属性)

```
public static END : Number
```

**End** 键的键控代码值 (35)。

可用性: ActionScript 1.0 ; Flash Player 5

## ENTER (Key.ENTER 属性)

```
public static ENTER : Number
```

**Enter** 键的键控代码值 (13)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

在下面的示例中，当您按箭头键时，名为 `car_mc` 的影片剪辑将移动一段固定距离 (10)。当您按 **Enter** 并删除 `onEnterFrame` 事件时，`car_mc` 实例将停止移动。

```

var DISTANCE:Number = 5;
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.LEFT :
            car_mc.onEnterFrame = function() {
                this._x -= DISTANCE;
            };
            break;
        case Key.UP :
            car_mc.onEnterFrame = function() {
                this._y -= DISTANCE;
            };
            break;
        case Key.RIGHT :
            car_mc.onEnterFrame = function() {
                this._x += DISTANCE;
            };
    }
};

```

```

        break;
        case Key.DOWN :
            car_mc.onEnterFrame = function() {
                this._y += DISTANCE;
            };
            break;
        case Key.ENTER :
            delete car_mc.onEnterFrame;
            break;
    }
};
Key.addListener(keyListener);

```

使用此示例时，请确保在测试环境中选择 “控制” > “禁用快捷键”。

## ESCAPE（Key.ESCAPE 属性）

public static ESCAPE : Number

Esc 键的键控代码值 (27)。

可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例设置计时器。当您按 **Escape** 时，“输出”面板显示包含您按键所用时间的信息。

```

var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.ESCAPE)) {
        // Get the current timer, convert the value to seconds and round it to two
        decimal places.
        var timer:Number = Math.round(getTimer()/10)/100;
        trace("you pressed the Esc key: "+getTimer()+" ms ("+"timer+" s)");
    }
};
Key.addListener(keyListener);

```

使用此示例时，请确保在测试环境中选择 “控制” > “禁用快捷键”。

## getAscii (Key.getAscii 方法)

```
public static getAscii() : Number
```

返回按下或释放的最后一个键的 **ASCII** 码。返回的 **ASCII** 值为英文键盘值。例如，如果您按下 **Shift+2**，则在日文键盘上 `Key.getAscii()` 将返回 `@`，这与在英文键盘上的返回结果一样。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

**Number** - 按下的最后一个键的 **ASCII** 值。如果没有按下或释放任何键，或者由于安全原因无法访问键控代码，则此方法将返回 **0**。

### 示例

下面的示例在每次按某个键时调用 `getAscii()` 方法。该示例通过调用 `Key.getAscii()` 创建名为 `keyListener` 的侦听器对象并定义响应 `onKeyDown` 事件的函数。接着，向 `Key` 对象注册 `keyListener` 对象，这样，在播放 **SWF** 文件时，只要按任意键，都会广播 `onKeyDown` 消息。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("The ASCII code for the last key typed is: "+Key.getAscii());
};
Key.addListener(keyListener);
```

使用此示例时，请确保在测试环境中选择“控制”>“禁用快捷键”。

下面的示例添加对 `Key.getAscii()` 的调用，以显示两种方法的不同之处。主要不同之处是：`Key.getAscii()` 区分大小写字母，而 `Key.getCode()` 不区分。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("For the last key typed:");
    trace("\tThe Key code is: "+Key.getCode());
    trace("\tThe ASCII value is: "+Key.getAscii());
    trace("");
};
Key.addListener(keyListener);
```

使用此示例时，请确保在测试环境中选择“控制”>“禁用快捷键”。

### 另请参见

[isAccessible \(Key.isAccessible 方法\)](#)

## getCode (Key.getCode 方法)

```
public static getCode() : Number
```

返回按下的最后一个键的键控代码值。

注意：此方法的 **Flash Lite** 实现返回一个字符串或一个数字，具体取决于平台传入的键控代码。唯一有效的键控代码是此类接受的标准键控代码以及作为 `ExtendedKey` 类的属性列出的特殊键控代码。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 按下的最后一个键的键控代码。如果没有按下或释放任何键，或者由于安全原因无法访问键控代码，则此方法将返回 0。

### 示例

下面的示例在每次按某个键时调用 `getCode()` 方法。该示例通过调用 `Key.getCode()` 创建名为 `keyListener` 的侦听器对象并定义响应 `onKeyDown` 事件的函数。接着，向 `Key` 对象注册 `keyListener` 对象，这样，在播放 **SWF** 文件时，只要按任意键，都会广播 `onKeyDown` 消息。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    // compare return value of getCode() to constant
    if (Key.getCode() == Key.ENTER) {
        trace("Virtual key code: "+Key.getCode()+" (ENTER key)");
    }
    else {
        trace("Virtual key code: "+Key.getCode());
    }
};
Key.addListener(keyListener);
```

使用此示例时，请确保在测试环境中选择“控制”>“禁用快捷键”。

下面的示例添加对 `Key.getAscii()` 的调用，以显示两种方法的不同之处。主要不同之处是：`Key.getAscii()` 区分大小写字母，而 `Key.getCode()` 不区分。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("For the last key typed:");
    trace("\tThe Key code is: "+Key.getCode());
    trace("\tThe ASCII value is: "+Key.getAscii());
};
```

```
        trace("");
    };
    Key.addListener(keyListener);
```

使用此示例时，请确保在测试环境中选择“控制”>“禁用快捷键”。

另请参见

[getAscii \(Key.getAscii 方法\)](#), [isAccessible \(Key.isAccessible 方法\)](#)

## HOME (Key.HOME 属性)

```
public static HOME : Number
```

Home 键的键控代码值 (36)。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例在 **x** 和 **y** 坐标为 **0,0** 处附加名为 **car\_mc** 的可拖动影片剪辑。当您按 **Home** 键时，**car\_mc** 返回 **0,0**。创建一个具有链接 ID 为 **car\_id** 的影片剪辑，并将下面的 **ActionScript** 添加到时间轴中的第 1 帧：

```
this.attachMovie("car_id", "car_mc", this.getNextHighestDepth(), {_x:0,
    _y:0});
car_mc.onPress = function() {
    this.startDrag();
};
car_mc.onRelease = function() {
    this.stopDrag();
};
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.HOME)) {
        car_mc._x = 0;
        car_mc._y = 0;
    }
};
Key.addListener(keyListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## INSERT (Key.INSERT 属性)

public static INSERT : Number

Insert 键的键控代码值 (45)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

下面的示例创建新的侦听器对象, 并为 onKeyDown 定义函数。最后一行使用 addListener() 向 Key 对象注册该侦听器, 以便该对象可接收 key down 事件的通知并在 “输出” 面板中显示信息。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.INSERT)) {
        trace("You pressed the Insert key.");
    }
};
Key.addListener(keyListener);
```

## isAccessible (Key.isAccessible 方法)

public static isAccessible() : Boolean

根据安全限制返回一个布尔值, 该值指示按下的最后一个键是否可以被其它 SWF 文件访问。默认情况下, 某一个域中的 SWF 文件的代码不可以访问从另一个域中的 SWF 文件生成的键击。有关跨域安全的更多信息, 请参见《学习 Flash 中的 ActionScript 2.0》中的 “了解安全性”。

可用性: ActionScript 1.0 ; Flash Player 8

### 返回

Boolean - 值 true (当按下的最后一个键可以被访问时)。如果不允许访问, 则此方法返回 false。



## isDown (Key.isDown 方法)

`public static isDown(code:Number) : Boolean`

如果按下 `keycode` 中指定的键，则返回 `true`；否则返回 `false`。在 **Macintosh** 上，**Caps Lock** 键和 **Num Lock** 键的键控代码值相同。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性：ActionScript 1.0；Flash Player 5

### 参数

`code:Number` - 分配给特定键的键控代码值，或与特定键相关联的 **Key** 类属性。

### 返回

`Boolean` - 如果按下 `keycode` 中指定的键，则返回 `true`；否则返回 `false`。

### 示例

以下脚本使用户可以控制影片剪辑 (`car_mc`) 的位置：

```
car_mc.onEnterFrame = function() {  
    if (Key.isDown(Key.RIGHT)) {  
        this._x += 10;  
    } else if (Key.isDown(Key.LEFT)) {  
        this._x -= 10;  
    }  
};
```

## isToggled (Key.isToggled 方法)

`public static isToggled(code:Number) : Boolean`

如果激活 **Caps Lock** 或 **Num Lock** 键（切换到活动状态），则返回 `true`；否则返回 `false`。

尽管切换一词通常是指在两个选项之间切换，但只有该键切换到活动状态，**Key.isToggled()** 方法才返回 `true`。在 **Macintosh** 上，**Caps Lock** 键和 **Num Lock** 键的键控代码值相同。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性：ActionScript 1.0；Flash Player 5

### 参数

`code:Number` - **Caps Lock** 键 (20) 或 **Num Lock** 键 (144) 的键控代码。

## 返回

Boolean - 如果激活 **Caps Lock** 或 **Num Lock** 键（切换到活动状态），则返回 true；否则返回 false。

## 示例

下面的示例在每次按某个键时调用 `isToggled()` 方法，并且在每次 **Caps Lock** 键切换到活动状态时执行 `trace` 语句。该示例通过调用 `Key.isToggled()` 创建名为 `keyListener` 的侦听器对象并定义响应 `onKeyDown` 事件的函数。接着，向 `Key` 对象注册 `keyListener` 对象，这样，在播放 **SWF** 文件时，只要按任意键，都会广播 `onKeyDown` 消息。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.CAPSLOCK)) {
        trace("you pressed the Caps Lock key.");
        trace("\tCaps Lock == "+Key.isToggled(Key.CAPSLOCK));
    }
};
Key.addListener(keyListener);
```

当您按 **Caps Lock** 键时，将在“输出”面板中显示信息。“输出”面板显示 true 或 false，具体取决于是否使用 `isToggled` 方法激活了 **Caps Lock**。

下面的示例创建两个文本字段，它们会在切换 **Caps Lock** 键和 **Num Lock** 键时更新。每个文本字段在键被激活时显示 **true**，在键被取消激活时显示 **false**。

```
this.createTextField("capsLock_txt", this.getNextHighestDepth(), 0, 0, 100,
    22);
capsLock_txt.autoSize = true;
capsLock_txt.html = true;
this.createTextField("numLock_txt", this.getNextHighestDepth(), 0, 22, 100,
    22);
numLock_txt.autoSize = true;
numLock_txt.html = true;
//
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    capsLock_txt.htmlText = "<b>Caps Lock:</b> "+Key.isToggled(Key.CAPSLOCK);
    numLock_txt.htmlText = "<b>Num Lock:</b> "+Key.isToggled(144);
};
Key.addListener(keyListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## LEFT (Key.LEFT 属性)

public static LEFT : Number

左箭头键的键控代码值 (37)。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

在下面的示例中，当您按箭头键时，名为 car\_mc 的影片剪辑将移动一段固定距离 (10)。按空格键时将播放声音。在此示例中，为库中的声音指定链接标识符 horn\_id。

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);
```

## \_listeners (Key.\_listeners 属性)

```
public static _listeners : Array [read-only]
```

一个引用列表，引用对象是向 **Key** 对象注册的所有侦听器对象。此属性供内部使用，但是，如果您要确定当前向 **Key** 对象注册的侦听器的数量，此属性可能有用。通过调用 `addListener()` 方法和 `removeListener()` 方法，可以分别在数组中添加和删除对象。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例演示如何使用 `length` 属性确定当前向 **Key** 对象注册的侦听器对象的数量。

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
Key.addListener(myListener);

trace(Key._listeners.length); // Output: 1
```

## onKeyDown (Key.onKeyDown 事件侦听器)

```
onKeyDown = function() {}
```

当按下某按键时获得通知。若要使用 `onKeyDown`，必须创建一个侦听器对象。然后可以为 `onKeyDown` 定义一个函数，并使用 `addListener()` 向 **Key** 对象注册该侦听器，如下面的示例所示：

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("DOWN -> Code: "+Key.getCode()+"\tACSI: "+Key.getAscii()+"\tKey: "+chr(Key.getAscii()));
};
keyListener.onKeyUp = function() {
    trace("UP -> Code: "+Key.getCode()+"\tACSI: "+Key.getAscii()+"\tKey: "+chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

**Flash** 应用程序只能监视其焦点内发生的键盘事件。**Flash** 应用程序无法检测其它应用程序中的键盘事件。

可用性: **ActionScript 1.0** ; **Flash Player 6**

另请参见

[addListener](#) ([Key.addListener](#) 方法)

## onKeyUp (Key.onKeyUp 事件侦听器)

```
onKeyUp = function() {}
```

当释放某按键时获得通知。若要使用 onKeyUp，必须创建一个侦听器对象。然后可以为 onKeyUp 定义一个函数，并使用 addListener() 向 Key 对象注册该侦听器，如下面的示例所示：

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    trace("DOWN -> Code: "+Key.getCode()+"\tACSCII: "+Key.getAscii()+"\tKey: "+chr(Key.getAscii()));
};
keyListener.onKeyUp = function() {
    trace("UP -> Code: "+Key.getCode()+"\tACSCII: "+Key.getAscii()+"\tKey: "+chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

Flash 应用程序只能监视其焦点内发生的键盘事件。Flash 应用程序无法检测其它应用程序中的键盘事件。

可用性：ActionScript 1.0；Flash Player 6

另请参见

[addListener \(Key.addListener 方法\)](#)

## PGDN (Key.PGDN 属性)

```
public static PGDN : Number
```

Page Down 键的键控代码值 (34)。

可用性：ActionScript 1.0；Flash Player 5

示例

在下面的示例中，当您按 Page Down 和 Page Up 键时，名为 car\_mc 的影片剪辑将会旋转。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.PGDN)) {
        car_mc._rotation += 5;
    } else if (Key.isDown(Key.PGUP)) {
        car_mc._rotation -= 5;
    }
};
Key.addListener(keyListener);
```

## PGUP (Key.PGUP 属性)

public static PGUP : Number

Page Up 键的键控代码值 (33)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

在下面的示例中, 当您按 Page Down 和 Page Up 键时, 名为 car\_mc 的影片剪辑将会旋转。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.PGDN)) {
        car_mc._rotation += 5;
    } else if (Key.isDown(Key.PGUP)) {
        car_mc._rotation -= 5;
    }
};
Key.addListener(keyListener);
```

## removeListener (Key.removeListener 方法)

public static removeListener(listener:Object) : Boolean

删除以前用 Key.addListener() 注册的对象。

可用性: ActionScript 1.0 ; Flash Player 6

### 参数

listener:Object - 一个对象。

### 返回

Boolean - 如果成功删除了 listener, 则该方法返回 true。如果未能成功删除 listener (例如, 由于 listener 不在 Key 对象的侦听器列表上), 则该方法返回 false。

### 示例

下面的示例使用左箭头键和右箭头键移动名为 car\_mc 的影片剪辑。当您按 Esc 时, 将会删除侦听器, 而 car\_mc 将不再移动。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.LEFT :
            car_mc._x -= 10;
            break;
        case Key.RIGHT :
            car_mc._x += 10;
    }
};
```

```

        break;
        case Key.ESCAPE :
            Key.removeListener(keyListener);
        }
    };
    Key.addListener(keyListener);

```

## RIGHT (Key.RIGHT 属性)

public static RIGHT : Number

右箭头键的键控代码值 (39)。

**可用性:** ActionScript 1.0 ; Flash Player 5

### 示例

在下面的示例中，当您按箭头键时，名为 car\_mc 的影片剪辑将移动一段固定距离 (10)。按空格键时将播放声音。在此示例中，为库中的声音指定链接标识符 horn\_id。

```

var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);

```

## SHIFT (Key.SHIFT 属性)

public static SHIFT : Number

Shift 键的键控代码值 (16)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

下面的示例在您按 **Shift** 时缩放 car\_mc。

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.SHIFT)) {
        car_mc._xscale = 2;
        car_mc._yscale = 2;
    } else if (Key.isDown(Key.CONTROL)) {
        car_mc._xscale /= 2;
        car_mc._yscale /= 2;
    }
};
Key.addListener(keyListener);
```

## SPACE (Key.SPACE 属性)

public static SPACE : Number

空格键的键控代码值 (32)。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

在下面的示例中，当您按箭头键时，名为 car\_mc 的影片剪辑将移动一段固定距离 (10)。按空格键时将播放声音。在此示例中，为库中的声音指定链接标识符 horn\_id。

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
```



```

        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
        }
    };
    Key.addListener(keyListener_obj);

```

## TAB (Key.TAB 属性)

public static TAB : Number

Tab 键的键控代码值 (9)。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例创建一个文本字段，并且在您按 **Tab** 时在文本字段中显示日期。

```

this.createTextField("date_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
date_txt.autoSize = true;
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.TAB)) {
        var today_date:Date = new Date();
        date_txt.text = today_date.toString();
    }
};
Key.addListener(keyListener);

```

使用此示例时，请确保在测试环境中选择 “控制” > “禁用快捷键”。

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## UP (Key.UP 属性)

public static UP : Number

上箭头键的键控代码值 (38)。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 示例

在下面的示例中，当您按箭头键时，名为 car\_mc 的影片剪辑将移动一段固定距离 (10)。按空格键时将播放声音。在此示例中，为库中的声音指定链接标识符 horn\_id。

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
    switch (Key.getCode()) {
        case Key.SPACE :
            horn_sound.start();
            break;
        case Key.LEFT :
            car_mc._x -= DISTANCE;
            break;
        case Key.UP :
            car_mc._y -= DISTANCE;
            break;
        case Key.RIGHT :
            car_mc._x += DISTANCE;
            break;
        case Key.DOWN :
            car_mc._y += DISTANCE;
            break;
    }
};
Key.addListener(keyListener_obj);
```

# LoadVars



```
public dynamic class LoadVars
extends Object
```

您可以使用 **LoadVars** 类来验证数据加载是否成功，并监视下载进程。**LoadVars** 类是 `loadVariables()` 函数的替代方法，用于在 **Flash** 应用程序和服务器之间传输变量。

**LoadVars** 类使您可以将对象中的所有变量发送到指定的 **URL** 中，并且可以将指定 **URL** 中的所有变量加载到某个对象中。它也可以发送那些可使应用程序更加有效的特定变量，而不是所有变量。您可以使用 `LoadVars.onLoad` 处理函数来确保应用程序在加载数据之时（而不是之前）运行。

**LoadVars** 类的工作原理非常类似于 **XML** 类：它使用 `load()`、`send()` 和 `sendAndLoad()` 方法与服务器进行通讯。**LoadVars** 类和 **XML** 类之间的主要差别在于 **LoadVars** 传输 **ActionScript** 的名称和值对，而不是 **XML** 对象中存储的 **XML** 文档对象模型 (DOM) 树。**LoadVars** 类与 **XML** 类遵循相同的安全限制。

可用性：ActionScript 1.0 ； Flash Player 6

另请参见

[loadVariables 函数](#)，[onLoad \(LoadVars.onLoad 处理函数\)](#)，[XML](#)

## 属性摘要

修饰符	属性	说明
	<code>contentType:String</code>	在您调用 <code>LoadVars.send()</code> 或 <code>LoadVars.sendAndLoad()</code> 时发送到服务器的 MIME 类型。
	<code>loaded:Boolean</code>	一个布尔值，指示 <code>load</code> 或 <code>sendAndLoad</code> 操作是否已完成，默认情况下为 <code>undefined</code> 。

继承自 **Object** 类的属性

<a href="#">constructor (Object.constructor 属性)</a> ， <a href="#">__proto__ (Object.__proto__ 属性)</a> ， <a href="#">prototype (Object.prototype 属性)</a> ， <a href="#">__resolve (Object.__resolve 属性)</a>
---

## 事件摘要

事件	说明
<code>onData = function(src:String) {}</code>	当数据从服务器上完全下载时，或者当从服务器下载数据的过程中出现错误时调用。
<code>onHTTPStatus = function(httpStatus:Number) {}</code>	当 Flash Player 接收来自服务器的 HTTP 状态代码时调用。
<code>onLoad = function(success:Boolean) {}</code>	当 <code>LoadVars.load()</code> 或 <code>LoadVars.sendAndLoad()</code> 操作已结束时调用。

## 构造函数摘要

签名	说明
<code>LoadVars()</code>	创建 <code>LoadVars</code> 对象。

## 方法摘要

修饰符	签名	说明
	<code>addRequestHeader(header:Object, headerValue:String) : Void</code>	添加或更改与 POST 动作一起发送的 HTTP 请求标题（如 <code>Content-Type</code> 或 <code>SOAPAction</code> ）。
	<code>decode(queryString:String) : Void</code>	将变量字符串转换为指定 <code>LoadVars</code> 对象的属性。
	<code>getBytesLoaded() : Number</code>	返回 <code>LoadVars.load()</code> 或 <code>LoadVars.sendAndLoad()</code> 所下载的字节数。
	<code>getBytesTotal() : Number</code>	返回 <code>LoadVars.load()</code> 或 <code>LoadVars.sendAndLoad()</code> 所下载的总字节数。
	<code>load(url:String) : Boolean</code>	从指定的 URL 下载变量，分析变量数据，然后将结果变量放在 <code>my_lv</code> 中。
	<code>send(url:String, target:String, [method:String]) : Boolean</code>	将 <code>my_lv</code> 对象中的变量发送到指定的 URL。
	<code>sendAndLoad(url:String, target:Object, [method:String]) : Boolean</code>	将 <code>my_lv</code> 对象中的变量发送到指定的 URL。
	<code>toString() : String</code>	以 MIME 内容编码格式 <code>application/x-www-form-urlencoded</code> 返回包含 <code>my_lv</code> 中所有可枚举变量的字符串。

继承自 Object 类的方法

---

`addProperty` (Object.addProperty 方法), `hasOwnProperty` (Object.hasOwnProperty 方法), `isPropertyEnumerable` (Object.isPropertyEnumerable 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `registerClass` (Object.registerClass 方法), `toString` (Object.toString 方法), `unwatch` (Object.unwatch 方法), `valueOf` (Object.valueOf 方法), `watch` (Object.watch 方法)

---

## addRequestHeader (LoadVars.addRequestHeader 方法)

```
public addRequestHeader(header:Object, headerValue:String) : Void
```

添加或更改与 POST 动作一起发送的 **HTTP** 请求标题（如 Content-Type 或 SOAPAction）。在第一种用法中，向该方法传递了两个字符串：header 和 headerValue。在第二种用法中，传递了字符串、替代标头名称和标头值的数组。

如果通过多次调用来设置相同的标头名称，则每个后继值将替换在上一次调用中设置的值。

不能使用此方法添加或更改以下标准 **HTTP** 标头：Accept-Ranges、Age、Allow、Allowed、Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、Vary、Via、Warning 和 WWW-Authenticate。

**可用性：**ActionScript 1.0；Flash Player 6

### 参数

**header:**Object - 一个字符串或字符串数组，表示 **HTTP** 请求标题名称。

**headerValue:**String - 一个字符串，表示与 header 关联的值。

### 示例

下面的示例将具有 Foo 值的名为 SOAPAction 的自定义 **HTTP** 标题添加到 my\_lv 对象：

```
my_lv.addRequestHeader("SOAPAction", "'Foo'");
```

下面的示例创建名为 headers 的数组，该数组包含两个可交替的 **HTTP** 标题及其关联值。该数组作为参数传递给 addRequestHeader()。

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];  
my_lv.addRequestHeader(headers);
```

下面的示例创建一个新 **LoadVars** 对象，该对象添加名为 FLASH-UUID 的请求标题。该标题包含可由服务器检查的变量。

```
var my_lv:LoadVars = new LoadVars();
my_lv.setRequestHeader("FLASH-UUID", "41472");
my_lv.name = "Mort";
my_lv.age = 26;
my_lv.send("http://flash-mx.com/mm/cgivars.cfm", "_blank", "POST");
```

另请参见

[addRequestHeader](#) ([XML.addRequestHeader](#) 方法)

## contentType (LoadVars.contentType 属性)

```
public contentType : String
```

在您调用 `LoadVars.send()` 或 `LoadVars.sendAndLoad()` 时发送到服务器的 MIME 类型。默认值为 **application/x-www-form-urlencoded**。

可用性: **ActionScript 1.0** ; **Flash Player 6**

示例

下面的示例创建一个 **LoadVars** 对象，并显示发送到服务器的数据的默认内容类型。

```
var my_lv:LoadVars = new LoadVars();
trace(my_lv.contentType); // output: application/x-www-form-urlencoded
```

另请参见

[send](#) ([LoadVars.send](#) 方法), [sendAndLoad](#) ([LoadVars.sendAndLoad](#) 方法)

## decode (LoadVars.decode 方法)

```
public decode(queryString:String) : Void
```

将变量字符串转换为指定 **LoadVars** 对象的属性。

此方法由 `LoadVars.onData` 事件处理函数内部使用。绝大多数用户无需直接调用此方法。如果您覆盖 `LoadVars.onData` 事件处理函数，则可以显式调用 `LoadVars.decode()` 来分析变量字符串。

可用性: **ActionScript 1.0** ; **Flash Player 7**

参数

**queryString:String** - 包含名称 / 值对的 URL 编码的查询字符串。

## 示例

下面的示例跟踪三个变量:

```
// Create a new LoadVars object
var my_lv:LoadVars = new LoadVars();
//Convert the variable string to properties
my_lv.decode("name=Mort&score=250000");
trace(my_lv.toString());
// Iterate over properties in my_lv
for (var prop in my_lv) {
    trace(prop+" -> "+my_lv[prop]);
}
```

另请参见

[onData \(LoadVars.onData 处理函数\)](#), [parseXML \(XML.parseXML 方法\)](#)

## getBytesLoaded (LoadVars.getBytesLoaded 方法)

public getBytesLoaded() : Number

返回 LoadVars.load() 或 LoadVars.sendAndLoad() 所下载的字节数。如果当前没有执行加载操作或者尚未开始加载操作, 此方法将返回 undefined。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 返回

Number - 一个整数。

## 示例

下面的示例使用 **ProgressBar** 实例和 **LoadVars** 对象下载文本文件。测试文件时, “输出” 面板中会显示两项内容: 文件加载是否成功, 已向 **SWF** 文件中载入多少数据。必须替换 LoadVars.load() 命令的 **URL** 参数, 以便使该参数表示使用 **HTTP** 的有效文本文件。如果尝试用此示例加载驻留在硬盘上的本地文件, 此示例可能无法正常运行, 原因是 **Flash Player** 在测试影片模式下加载本地文件的所有内容。要查看此代码的运行情况, 请在舞台上添加名为 loadvars\_pb 的 **ProgressBar** 实例。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中:

```
var loadvars_pb:mx.controls.ProgressBar;
var my_lv:LoadVars = new LoadVars();
loadvars_pb.mode = "manual";
this.createEmptyMovieClip("timer_mc", 999);
timer_mc.onEnterFrame = function() {
    var lvBytesLoaded:Number = my_lv.getBytesLoaded();
    var lvBytesTotal:Number = my_lv.getBytesTotal();
    if (lvBytesTotal != undefined) {
        trace("Loaded "+lvBytesLoaded+" of "+lvBytesTotal+" bytes.");
    }
}
```

```

        loadvars_pb.setProgress(lvBytesLoaded, lvBytesTotal);
    }
};
my_lv.onLoad = function(success:Boolean) {
    loadvars_pb.setProgress(my_lv.getBytesLoaded(), my_lv.getBytesTotal());
    delete timer_mc.onEnterFrame;
    if (success) {
        trace("LoadVars loaded successfully.");
    } else {
        trace("An error occurred while loading variables.");
    }
};
my_lv.load("[place a valid URL pointing to a text file here]");

```

另请参见

[load \(LoadVars.load 方法\)](#), [sendAndLoad \(LoadVars.sendAndLoad 方法\)](#)

## getBytesTotal (LoadVars.getBytesTotal 方法)

```
public getBytesTotal() : Number
```

返回 LoadVars.load() 或 LoadVars.sendAndLoad() 所下载的总字节数。如果当前没有执行加载操作或者尚未开始加载操作，此方法将返回 undefined。如果无法确定总字节数（例如，如果下载已开始但服务器尚未传输 **HTTP Content-Length**），此方法也将返回 undefined。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 返回

Number - 一个整数。

### 示例

下面的示例使用 **ProgressBar** 实例和 **LoadVars** 对象下载文本文件。测试文件时，“输出”面板中会显示两项内容：文件加载是否成功，已向 **SWF** 文件中载入多少数据。必须替换 LoadVars.load() 命令的 **URL** 参数，以便使该参数表示使用 **HTTP** 的有效文本文件。如果尝试用此示例加载驻留在硬盘上的本地文件，它可能无法正常运行，原因是 **Flash Player** 在测试影片模式下加载本地文件的所有内容。要查看此代码的运行情况，请在舞台上添加名为 loadvars\_pb 的 **ProgressBar** 实例。然后将下面的 **ActionScript** 添加到时间轴的第 1 帧中：

```

var loadvars_pb:mx.controls.ProgressBar;
var my_lv:LoadVars = new LoadVars();
loadvars_pb.mode = "manual";
this.createEmptyMovieClip("timer_mc", 999);
timer_mc.onEnterFrame = function() {
    var lvBytesLoaded:Number = my_lv.getBytesLoaded();

```



```

var lvBytesTotal:Number = my_lv.getBytesTotal();
if (lvBytesTotal != undefined) {
    trace("Loaded "+lvBytesLoaded+" of "+lvBytesTotal+" bytes.");
    loadvars_pb.setProgress(lvBytesLoaded, lvBytesTotal);
}
};
my_lv.onLoad = function(success:Boolean) {
    loadvars_pb.setProgress(my_lv.getBytesLoaded(), my_lv.getBytesTotal());
    delete timer_mc.onEnterFrame;
    if (success) {
        trace("LoadVars loaded successfully.");
    } else {
        trace("An error occurred while loading variables.");
    }
};
my_lv.load("[place a valid URL pointing to a text file here]");

```

另请参见

[load \(LoadVars.load 方法\)](#), [sendAndLoad \(LoadVars.sendAndLoad 方法\)](#)

## load (LoadVars.load 方法)

public load(url:String) : Boolean

从指定的 **URL** 下载变量，分析变量数据，然后将结果变量放在 `my_lv` 中。`my_lv` 中任何与所下载的变量同名的属性都将被覆盖。`my_lv` 中任何与所下载的变量不同名的属性都不会被删除。这是一个异步动作。

下载的数据必须是 **MIME 内容类型** `application/x-www-form-urlencoded`。

此格式与 `loadVariables()` 所使用的格式相同。

此外，在为 **Flash Player 7** 发布的文件中，使用 `LoadVars.load()` 加载的外部变量支持区分大小写。

此方法与 `XML.load()` 相似。



如果要下载的文件包含非 ASCII 字符（在许多非英语的语言中出现），建议您使用 UTF-8 或 UTF-16 编码（而不是诸如 ASCII 的非 Unicode 格式）来保存文件。

使用此方法时，请考虑 **Flash Player 安全模型**：

对于 **Flash Player 8**：

- 如果执行调用的 **SWF** 文件在只能与本地文件系统内容交互的沙箱中，而目标资源来自网络沙箱，则不允许进行数据加载。
- 如果执行调用的 **SWF** 文件来自网络沙箱而目标资源在本地，也不允许进行数据加载。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

对于 Flash Player 7 及更高版本，网站可通过跨域策略文件允许对资源进行跨域访问。如果任何版本的 SWF 文件在 Flash Player 7 及更高版本中运行，则 url 必须位于完全相同的域中。例如，位于 **www.someDomain.com** 的 SWF 文件只能从也位于 **www.someDomain.com** 的源中加载数据。

如果 SWF 文件在低于 Flash Player 7 版本的播放器中运行，则 url 必须与发出此调用的 SWF 文件位于同一个超级域中。超级域可以通过删除某一文件的 URL 最左侧的组件而得到。例如，位于 **www.someDomain.com** 的 SWF 文件可以从位于 **store.someDomain.com** 的源中加载数据，这是因为这两个文件都在同一个名为 **someDomain.com** 的超级域中。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**url:String** - 一个字符串；从其下载变量的 URL。如果发出此调用的 SWF 文件正在 Web 浏览器中运行，则 url 必须与 SWF 文件位于同一个域中。

## 返回

**Boolean** - 如果没有任何参数（空值）被传递，则为 false；否则为 true。请使用 `onLoad()` 事件处理函数检查数据加载是否成功。

## 示例

下面的代码定义 `onLoad` 处理函数，当数据从服务器端的 PHP 脚本返回到 Flash 应用程序时，该函数会发出信号，然后加载 `passvars.php` 中的数据。

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
    if (success) {
        trace(this.toString());
    } else {
        trace("Error loading/parsing LoadVars.");
    }
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

有关其它示例，请参见 **ActionScript** 示例文件夹中的 **guestbook.fla** 文件。此文件夹的一些典型路径如下：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

另请参见

[load \(XML.load 方法\)](#)，[loaded \(LoadVars.loaded 属性\)](#)，[onLoad \(LoadVars.onLoad 处理函数\)](#)，[useCodepage \(System.useCodepage 属性\)](#)

## loaded (LoadVars.loaded 属性)

`public loaded : Boolean`

一个布尔值，指示 `load` 或 `sendAndLoad` 操作是否已完成，默认情况下为 `undefined`。开始 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 操作时，`loaded` 属性设置为 `false`；完成此操作时，`loaded` 属性设置为 `true`。如果该操作尚未完成或由于错误而失败，则 `loaded` 属性仍保持设置为 `false`。

此属性与 `XML.loaded` 属性类似。

**可用性:** ActionScript 1.0；Flash Player 6

### 示例

下面的示例加载一个文本文件，并在该操作完成时在“输出”面板中显示信息。

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
    trace("LoadVars loaded successfully: "+this.loaded);
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

另请参见

[load \(LoadVars.load 方法\)](#)，[sendAndLoad \(LoadVars.sendAndLoad 方法\)](#)，[load \(XML.load 方法\)](#)

## LoadVars 构造函数

```
public LoadVars()
```

创建 **LoadVars** 对象。然后您可使用该 **LoadVars** 对象的方法来发送和加载数据。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个名为 `my_lv` 的 **LoadVars** 对象：

```
var my_lv:LoadVars = new LoadVars();
```

## onData (LoadVars.onData 处理函数)

```
onData = function(src:String) {}
```

当数据从服务器上完全下载时，或者当从服务器下载数据的过程中出现错误时调用。此处理函数在分析数据之前调用，因此它可用于调用自定义分析例程，而不必调用 **Flash Player** 中的内置分析例程。对于分配给 `LoadVars.onData` 的函数，传递给该函数的 `src` 参数的值可以是 `undefined`，也可以是包含从服务器下载的 **URL** 编码名称 / 值对的字符串。如果 `src` 参数为 `undefined`，则从服务器下载数据时将出现错误。

`LoadVars.onLoad` 的默认实现调用 `LoadVars.onData`。您可以通过对 `LoadVars.onData` 分配自定义函数来覆盖此默认实现，但是没有调用 `LoadVars.onLoad`，除非在 `LoadVars.onData` 的实现中调用它。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**src:String** - 一个字符串或 `undefined`；来自 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 方法调用的原始（未分析）数据。

### 示例

下面的示例加载一个文本文件，并在该操作完成时在名为 `content_ta` 的 **TextArea** 实例中显示内容。如果发生错误，则在“输出”面板中显示信息。

```
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
    if (src == undefined) {
        trace("Error loading content.");
        return;
    }
    content_ta.text = src;
};
my_lv.load("content.txt", my_lv, "GET");
```

另请参见

[onLoad \(LoadVars.onLoad 处理函数\)](#), [onLoad \(LoadVars.onLoad 处理函数\)](#), [load \(LoadVars.load 方法\)](#), [sendAndLoad \(LoadVars.sendAndLoad 方法\)](#)

## onHTTPStatus (LoadVars.onHTTPStatus 处理函数)

```
onHTTPStatus = function(httpStatus:Number) {}
```

当 **Flash Player** 接收来自服务器的 **HTTP** 状态代码时调用。使用此处理函数，您可以捕获 **HTTP** 状态代码并根据该状态代码进行操作。

**onHTTPStatus** 处理函数在 **onData** 前调用，它在加载失败时触发对值为 **undefined** 的 **onLoad** 的调用。触发 **onHTTPStatus** 后，不管是否覆盖 **onHTTPStatus**，始终 触发 **onData**。要充分地使用 **onHTTPStatus** 处理函数，应该编写一个函数来捕获 **onHTTPStatus** 调用的结果；然后可以在 **onData** 和 **onLoad** 处理函数中使用该结果。如果未调用 **onHTTPStatus**，则表示播放器没有尝试发出 **URL** 请求。发生这种情况是因为此请求违反了 **SWF** 文件的安全沙箱规则。

如果 **Flash Player** 无法从服务器获取状态代码或无法与服务器进行通讯，则默认值 **0** 会传递到 **ActionScript** 代码。在任何播放器中都可生成值 **0**（例如，如果请求的 **URL** 格式不正确），并且当 **Flash Player** 插件在以下不将 **HTTP** 状态代码传递到播放器的浏览器中运行时，值 **0** 始终由 **Flash Player** 插件生成：适用于 **Macintosh** 的 **Netscape**、**Mozilla**、**Safari**、**Opera** 和 **Internet Explorer**。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**httpStatus:Number** - 由服务器返回的 **HTTP** 状态代码。例如，值为 **404** 表示服务器尚未找到请求的 **URI** 的匹配项。在 <ftp://ftp.isi.edu/in-notes/rfc2616.txt> 处的 **HTTP** 规范的 **10.4** 和 **10.5** 节中，可以找到 **HTTP** 状态代码。

### 示例

下面的示例演示如何使用 **onHTTPStatus()** 帮助调试。此示例收集 **HTTP** 状态代码并将它们的值和类型分配给 **LoadVars** 对象的实例。（请注意，此示例在运行时创建实例成员 **this.httpStatus** 和 **this.httpStatusType**。）**onData** 方法使用这些实例成员跟踪有关调试中有用的 **HTTP** 响应的信息。

```
var myLoadVars:LoadVars = new LoadVars();

myLoadVars.onHTTPStatus = function(httpStatus:Number) {
    this.httpStatus = httpStatus;
    if(httpStatus < 100) {
        this.httpStatusType = "flashError";
    }
}
```

```

        else if(httpStatus < 200) {
            this.httpStatusType = "informational";
        }
        else if(httpStatus < 300) {
            this.httpStatusType = "successful";
        }
        else if(httpStatus < 400) {
            this.httpStatusType = "redirection";
        }
        else if(httpStatus < 500) {
            this.httpStatusType = "clientError";
        }
        else if(httpStatus < 600) {
            this.httpStatusType = "serverError";
        }
    }

    myLoadVars.onData = function(src:String) {
        trace(">> " + this.httpStatusType + ": " + this.httpStatus);
        if(src != undefined) {
            this.decode(src);
            this.loaded = true;
            this.onLoad(true);
        }
        else {
            this.onLoad(false);
        }
    }

    myLoadVars.onLoad = function(success:Boolean) {

    }

    myLoadVars.load("http://weblogs.macromedia.com/mxna/flashservices/
        getMostRecentPosts.cfm");

```

另请参见

[onHTTPStatus \(XML.onHTTPStatus 处理函数\)](#), [load \(LoadVars.load 方法\)](#), [sendAndLoad \(LoadVars.sendAndLoad 方法\)](#)

## onLoad (LoadVars.onLoad 处理函数)

```
onLoad = function(success:Boolean) {}
```

当 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 操作已结束时调用。如果该操作成功，将使用该操作所下载的变量填充 `my_lv`，而这些变量将在调用此处理函数时变为可用。

默认情况下此处理函数未定义。

此事件处理函数与 `XML.onLoad` 相似。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**success:Boolean** - 一个布尔值，指示加载操作是以成功结束 (`true`) 还是以失败结束 (`false`)。

### 示例

下面的示例在舞台上添加名为 `name_ti` 的 **TextInput** 实例，名为 `result_ta` 的 **TextArea** 实例，和名为 `submit_button` 的 **Button** 实例。当用户单击“登录”按钮实例时，会创建两个 **LoadVars** 对象: `send_lv` 和 `result_lv`。`send_lv` 对象从 `name_ti` 实例复制名称并将数据发送到 `greeting.cfm`。此脚本的结果将载入 `result_lv` 对象，服务器响应则显示在 **TextArea** 实例 (`result_ta`) 中。将下面的 **ActionScript** 添加到时间轴中的第 1 帧:

```
var submitListener:Object = new Object();
submitListener.click = function(evt:Object) {
    var result_lv:LoadVars = new LoadVars();
    result_lv.onLoad = function(success:Boolean) {
        if (success) {
            result_ta.text = result_lv.welcomeMessage;
        } else {
            result_ta.text = "Error connecting to server.";
        }
    };
    var send_lv:LoadVars = new LoadVars();
    send_lv.name = name_ti.text;
    send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv,
        "POST");
};
submit_button.addEventListener("click", submitListener);
```

要查看功能更强的示例，请参见 **ActionScript** 示例文件夹中的 **login.fla** 文件。此文件夹的一些典型路径如下：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

另请参见

`onLoad (XML.onLoad 处理函数)`，`loaded (LoadVars.loaded 属性)`，`load (LoadVars.load 方法)`，`sendAndLoad (LoadVars.sendAndLoad 方法)`

## send (LoadVars.send 方法)

```
public send(url:String, target:String, [method:String]) : Boolean
```

将 `my_lv` 对象中的变量发送到指定的 URL。`my_lv` 中的所有可枚举变量将连接成一个字符串（默认情况下，格式为 `application/x-www-form-urlencoded`），然后使用 HTTP POST 方法将此字符串发送到 URL。此格式与 `loadVariables()` 所使用的格式相同。在 HTTP 请求标题中发送的 MIME 内容类型是 `my_lv.contentType` 的值或默认的 `application/x-www-form-urlencoded`。除非指定了 GET，否则将使用 POST 方法。

必须指定 `target` 参数以确保可以执行指定 URL 上的脚本或应用程序。如果省略了 `target` 参数，则该函数将返回 `true`，但不会执行脚本或应用程序。

如果希望服务器响应以下操作，则可以使用 `send()` 方法：

- 替换 SWF 内容（使用 `"_self"` 作为 `target` 参数）；
- 出现在新窗口中（使用 `"_blank"` 作为 `target` 参数）；
- 出现在父级或顶级帧中（使用 `"_parent"` 或 `"_top"` 作为 `target` 参数）；
- 出现在命名帧中（使用该帧的名称作为 `target` 参数的字符串）。

成功调用 `send()` 方法总可以始终打开一个新的浏览器窗口或替换现有窗口或帧中的内容。如果您更希望向服务器发送信息并继续播放 SWF 文件，而不打开一个新窗口或替换窗口或帧中的内容，则应使用 `LoadVars.sendAndLoad()`。

此方法与 `XML.send()` 相似。

Flash 测试环境总是使用 GET 方法。若要使用 POST 方法进行测试，请确保从浏览器内尝试使用它。



使用此方法时，请考虑 **Flash Player** 安全模型：

- 对于 **Flash Player 8**，如果执行调用的 **SWF** 文件在不受信任的本地沙箱中，则不允许使用此方法。
- 对于 **Flash Player 7** 及更高版本，如果执行调用的 **SWF** 文件是本地文件，则不允许使用此方法。

有关更多信息，请参见以下部分：

- 《学习 **Flash** 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 **API** 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 6

## 参数

**url:String** - 一个字符串；上载变量的目标 URL。

**target:String** - 一个字符串；将在其中出现任何响应的浏览器窗口或帧。您可输入特定窗口的名称，或从下面的保留目标名称中选择：

- **"\_self"** 指定当前窗口中的当前帧。
- **"\_blank"** 指定一个新窗口。
- **"\_parent"** 指定当前帧的父级。
- **"\_top"** 指定当前窗口中的顶级帧。

**method:String** [ 可选 ] - 一个字符串；**HTTP** 协议的 **GET** 或 **POST** 方法。默认值为 **POST**。

## 返回

**Boolean** - 一个布尔值；如果未指定参数，则为 **false**；否则为 **true**。

## 示例

下面的示例从文本字段复制两个值，并将数据发送到用于处理信息的 **CFM** 脚本。例如，该脚本可能检查用户是否获得了高分，然后将该数据插入数据库表。

```
var my_lv:LoadVars = new LoadVars();
my_lv.playerName = playerName_txt.text;
my_lv.playerScore = playerScore_txt.text;
my_lv.send("setscore.cfm", "_blank", "POST");
```

## 另请参见

[sendAndLoad](#) ([LoadVars.sendAndLoad](#) 方法)，[send](#) ([XML.send](#) 方法)

## sendAndLoad (LoadVars.sendAndLoad 方法)

```
public sendAndLoad(url:String, target:Object, [method:String]) : Boolean
```

将 my\_lv 对象中的变量发送到指定的 URL。下载服务器响应，并将其作为变量数据进行分析，然后将结果变量放在 target 对象中。

变量发送的方式与 LoadVars.send() 相同。变量下载到 target 中的方式与 LoadVars.load() 相同。

使用此方法时，请考虑 Flash Player 安全模型：

对于 Flash Player 8：

- 如果执行调用的 SWF 文件在只能与本地文件系统内容交互的沙箱中，而目标资源来自网络沙箱，则不允许进行数据加载。
- 如果执行调用的 SWF 文件来自网络沙箱而目标资源在本地，也不允许进行数据加载。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

对于 Flash Player 7 及更高版本：

- 网站可通过跨域策略文件允许对资源进行跨域访问。
- 如果任何版本的 SWF 文件在 Flash Player 7 及更高版本中运行，则 url 必须位于完全相同的域中。例如，位于 **www.someDomain.com** 的 SWF 文件只能从也位于 **www.someDomain.com** 的源中加载数据。

如果 SWF 文件在低于 Flash Player 7 版本的播放器中运行，则 url 必须与发出此调用的 SWF 文件位于同一个超级域中。超级域可以通过删除某一文件的 URL 最左侧的组件而得到。例如，位于 **www.someDomain.com** 的 SWF 文件可以从位于 **store.someDomain.com** 的源中加载数据，这是因为这两个文件都在同一个名为 **someDomain.com** 的超级域中。

此方法与 XML.sendAndLoad() 相似。

可用性：ActionScript 1.0：Flash Player 6

### 参数

**url:String** - 一个字符串；上载变量的目标 URL。如果发出此调用的 SWF 文件正在 Web 浏览器中运行，则 url 必须与 SWF 文件位于同一个域中。

**target:Object** - 接收已下载变量的 LoadVars 或 XML 对象。

**method:String [可选]** - 一个字符串；HTTP 协议的 GET 或 POST 方法。默认值为 POST。

## 返回

Boolean - 一个布尔值。

## 示例

对于下面的示例，请在舞台上添加名为 name\_ti 的 **TextInput** 实例，名为 result\_ta 的 **TextArea** 实例和名为 submit\_button 的 **Button** 实例。当用户单击下面示例中的 **Login** 按钮实例时，将创建两个 **LoadVars** 对象：send\_lv 和 result\_lv。send\_lv 对象从 name\_ti 实例复制名称并将数据发送到 **greeting.cfm**。此脚本的结果将载入 result\_lv 对象，服务器响应则显示在 **TextArea** 实例 (result\_ta) 中。将下面的 **ActionScript** 添加到时间轴的第 1 帧：

```
var submitListener:Object = new Object();
submitListener.click = function(evt:Object) {
    var result_lv:LoadVars = new LoadVars();
    result_lv.onLoad = function(success:Boolean) {
        if (success) {
            result_ta.text = result_lv.welcomeMessage;
        } else {
            result_ta.text = "Error connecting to server.";
        }
    };
    var send_lv:LoadVars = new LoadVars();
    send_lv.name = name_ti.text;
    send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv,
        "POST");
};
submit_button.addEventListener("click", submitListener);
```

要查看功能更强的示例，请参见 **ActionScript** 示例文件夹中的 **login fla** 文件。

**ActionScript** 示例文件夹的典型路径为：

- Windows: 引导驱动器 \Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples\ActionScript

另请参见

[send \(LoadVars.send 方法\)](#), [load \(LoadVars.load 方法\)](#), [sendAndLoad \(XML.sendAndLoad 方法\)](#)

## toString (LoadVars.toString 方法)

```
public toString() : String
```

以 MIME 内容编码格式 `application/x-www-form-urlencoded` 返回包含 `my_lv` 中所有可枚举变量的字符串。

可用性: **ActionScript 1.0** ; **Flash Player 6**

返回

String - 一个字符串。

示例

下面的示例实例化一个新 `LoadVars()` 对象并创建两个属性, 然后使用 `toString()` 以 URL 编码的格式返回包含这两个属性的字符串:

```
var my_lv:LoadVars = new LoadVars();
my_lv.name = "Gary";
my_lv.age = 26;
trace (my_lv.toString()); //output: age=26&name=Gary
```

## LocalConnection

```
Object
|
+- LocalConnection
```

```
public dynamic class LocalConnection
extends Object
```

**LocalConnection** 类用于开发 SWF 文件, 这些文件无需使用 `fscommand()` 或 **JavaScript** 即可相互发送指令。**LocalConnection** 对象只能在运行于同一台客户端计算机上的 SWF 文件之间通讯, 但这些 SWF 文件可以在不同的应用程序中运行。例如, 一个 SWF 文件在浏览器中运行, 而一个 SWF 文件在放映文件中运行。您可以使用 **LocalConnection** 对象在单个 SWF 文件中发送和接收数据, 但这不是标准的实现; 本节中的所有示例均说明不同 SWF 文件之间的通讯。

用来发送和接收数据的主要方法是 `LocalConnection.send()` 和 `LocalConnection.connect()`。在最基础的层次, 您的代码将实现以下命令: 请注意, `LocalConnection.send()` 和 `LocalConnection.connect()` 命令均指定相同的连接名称 `lc_name`:

```
// Code in the receiving SWF file
this.createTextField("result_txt", 1, 10, 100, 22);
result_txt.border = true;
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.methodToExecute = function(param1:Number, param2:Number) {
```

```
result_txt.text = param1+param2;
};
receiving_lc.connect("lc_name");

// Code in the sending SWF file
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", 5, 7);
```

使用 **LocalConnection** 对象的最简单方法就是只允许在位于同一个域中的 **LocalConnection** 对象之间进行通讯，因为这样做不会出现安全方面的问题。但如果您需要在不同域之间进行通讯，则有多种方法来实现安全性措施。有关更多信息，请参见 **LocalConnection.send()** 中对 **connectionName** 参数的讨论以及 **LocalConnection.allowDomain** 和 **LocalConnection.domain()** 条目。

可用性：ActionScript 1.0 ； Flash Player 6

属性摘要  
继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

事件摘要

事件	说明
<code>allowDomain = function([sendingDomain:String]) {}</code>	每当 <code>receiving_lc</code> 从发送方 <b>LocalConnection</b> 对象收到调用方法的请求时调用。
<code>allowInsecureDomain = function([sendingDomain:String]) {}</code>	每当 <code>receiving_lc</code> （位于使用安全协议 <b>HTTPS</b> 的域所承载的 <b>SWF</b> 文件中）从发送方 <b>LocalConnection</b> 对象 （位于使用非安全协议承载的 <b>SWF</b> 文件中）收到调用方法的请求时调用。
<code>onStatus = function(infoObject:Object) {}</code>	当发送方 <b>LocalConnection</b> 对象尝试将命令发送到接收方 <b>LocalConnection</b> 对象之后调用。

构造函数摘要

签名	说明
<code>LocalConnection()</code>	创建 <b>LocalConnection</b> 对象。

方法摘要

修饰符	签名	说明
	<code>close() : Void</code>	关闭（断开连接） <code>LocalConnection</code> 对象。
	<code>connect(connectionName:String) : Boolean</code>	准备一个 <code>LocalConnection</code> 对象，以接收来自 <code>LocalConnection.send()</code> 命令（名为发送方 <code>LocalConnection</code> 对象）的命令。
	<code>domain() : String</code>	返回一个字符串，表示当前 <code>SWF</code> 文件所在位置的域。
	<code>send(connectionName:String, methodName:String, [args:Object]) : Boolean</code>	在使用 <code>LocalConnection.connect(connectionName)</code> 命令（接收方 <code>LocalConnection</code> 对象）打开的连接上调用名为 <code>method</code> 的方法。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

allowDomain (LocalConnection.allowDomain 处理函数)

```
allowDomain = function([sendingDomain:String]) {}
```

每当 `receiving_lc` 从发送方 `LocalConnection` 对象收到调用方法的请求时调用。Flash 需要使用您在此处理函数中实现的代码来返回布尔值 `true` 或 `false`。如果此处理函数没有返回 `true`，则将忽略发送方对象的请求，并且不调用该方法。

在不存在此事件处理函数时，Flash Player 会应用默认的安全策略，其代码与以下代码等效：

```
my_lc.allowDomain = function (sendingDomain)
{
    return (sendingDomain == this.domain());
}
```

使用 `LocalConnection.allowDomain` 可从指定域或任何域中显式允许 `LocalConnection` 对象执行接收方 `LocalConnection` 对象的方法。如果您没有声明 `sendingDomain` 参数，则可能需要接受任何域中的命令，并且处理函数中的代码将只为 `return true`。如果您声明了 `sendingDomain`，则可能需要将 `sendingDomain` 的值与要从中接受命令的域进行比较。以下示例显示了这两种实现方法。

在为 **Flash Player 6** 创作的文件中，`sendingDomain` 参数包含调用方的超级域。在为 **Flash Player 7** 或更高版本创作的文件中，`sendingDomain` 参数包含该调用方的精确的域。在后一种情况中，若要允许由在 **www.domain.com** 或 **store.domain.com** 中承载的 **SWF** 文件进行访问，您必须显式允许从这两个域的访问。

```
// For Flash Player 6
receiving_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="domain.com");
}
// For Flash Player 7 or later
receiving_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="www.domain.com" ||
        sendingDomain=="store.domain.com");
}
```

此外，对于为 **Flash Player 7** 或更高版本创作的文件，您不能使用此方法让使用安全协议 (**HTTPS**) 承载的 **SWF** 文件允许从使用非安全协议承载的 **SWF** 文件的访问；必须改用 `LocalConnection.allowInsecureDomain` 事件处理函数。

您可能会偶尔遇到以下情况。假设您从另一个域中加载了一个子级 **SWF** 文件。您想要实现此方法，以便使该子级 **SWF** 文件能够对父级 **SWF** 文件进行 **LocalConnection** 调用，但您却不知道该子级 **SWF** 文件究竟来自哪个最终域。例如，当您使用加载平衡重定向或第三方服务器时就可能发生这种情况。

在这种情况下，您可以在此方法的实现中使用 `MovieClip._url` 属性。例如，如果将 **SWF** 文件载入 `my_mc`，就可以通过检查域参数是否与 `my_mc._url` 的域相匹配来实现此方法。（您必须从包含在 `my_mc._url` 中的完整 **URL** 中分析出该域。）

如果您这样做的话，请务必等待 `my_mc` 中的 **SWF** 文件加载完毕，因为只有在该文件完全加载后，`_url` 属性才会具有其最终的正确值。确定子级 **SWF** 文件何时加载完毕的最佳方法就是使用 `MovieClipLoader.onLoadComplete`。

也可能发生与此相反的另一情况：您可能创建了一个子级 **SWF** 文件，用以接受来自父级的 **LocalConnection** 调用，但却不知道其父级的域。在这种情况下，可以通过检查域参量是否与 `_parent._url` 的域相匹配来实现此方法。同样，您必须从 `_parent._url` 中的完整 **URL** 分析出该域。在这种情况下，您不必等待父级 **SWF** 文件加载；加载子级时，父级已经加载完毕。

可用性：ActionScript 1.0；Flash Player 7

## 参数

`sendingDomain:String` [ 可选 ] - 一个字符串，指定包含发送方 **LocalConnection** 对象的 **SWF** 文件的域。

## 示例

以下示例显示接收方 SWF 文件中的 **LocalConnection** 对象如何允许任何域中的 SWF 文件调用其方法。将此示例与 `LocalConnection.connect()` 中的示例进行比较, 在该示例中, 只有来自同一个域中的 SWF 文件才能调用接收方 SWF 文件中的 `trace()` 方法。有关在连接名称中使用下划线 (`_`) 的讨论, 请参见 `LocalConnection.send()`。

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100,
    20);
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String) {
    domain_txt.text = sendingDomain;
    return true;
};
my_lc.allowInsecureDomain = function(sendingDomain:String) {
    return (sendingDomain == this.domain());
}
my_lc.sayHello = function(name:String) {
    welcome_txt.text = "Hello, "+name;
};
my_lc.connect("_mylc");
```

下面的示例将字符串发送到前面的 SWF 文件, 并显示有关本地连接是否能够连接到该文件的状态消息。名为 `name_ti` 的 **TextInput** 组件、名为 `status_ta` 的 **TextArea** 实例以及名为 `send_button` 的 **Button** 实例都用于显示内容。

```
var sending_lc:LocalConnection;
var sendListener:Object = new Object();
sendListener.click = function(evt:Object) {
    sending_lc = new LocalConnection();
    sending_lc.onStatus = function(infoObject:Object) {
        switch (infoObject.level) {
            case 'status' :
                status_ta.text = "LocalConnection connected successfully.";
                break;
            case 'error' :
                status_ta.text = "LocalConnection encountered an error.";
                break;
        }
    };
    sending_lc.send("_mylc", "sayHello", name_ti.text);
};
send_button.addEventListener("click", sendListener);
```

如果您的 SWF 文件包括第 2 版组件, 请使用第 2 版组件的 **DepthManager** 类而不是在前一个示例中所使用的 `MovieClip.getNextHighestDepth()` 方法。



在下面的示例中，驻留在 `thisDomain.com` 中的接收方 SWF 文件仅接收来自位于 `thisDomain.com` 或 `thisDomain.com` 的 SWF 文件的命令：

```
var aLocalConn:LocalConnection = new LocalConnection();
aLocalConn.Trace = function(aString) {
    aTextField += aString+newline;
};
aLocalConn.allowDomain = function(sendingDomain) {
    return (sendingDomain == this.domain() || sendingDomain ==
        "www.macromedia.com");
};
aLocalConn.connect("_mylc");
```

在为 **Flash Player 7** 或更高版本发布时，将使用精确域匹配。这意味着如果 SWF 文件位于 `www.thatDomain.com`，此示例将失败，但是如果 SWF 文件位于 `thatDomain.com`，该示例将正常运行。

另请参见

`connect` (`LocalConnection.connect` 方法), `domain` (`LocalConnection.domain` 方法), `send` (`LocalConnection.send` 方法), `_url` (`MovieClip._url` 属性), `onLoadComplete` (`MovieClipLoader.onLoadComplete` 事件侦听器), `_parent` 属性

## allowInsecureDomain

### (`LocalConnection.allowInsecureDomain` 处理函数)

```
allowInsecureDomain = function([sendingDomain:String]) {}
```

每当 `receiving_lc`（位于使用安全协议 **HTTPS** 的域所承载的 SWF 文件中）从发送方 `LocalConnection` 对象（位于使用非安全协议承载的 SWF 文件中）收到调用方法的请求时调用。**Flash** 需要使用您在此处理函数中实现的代码来返回布尔值 `true` 或 `false`。如果此处理函数没有返回 `true`，则将忽略发送方对象的请求，并且不调用该方法。

默认情况下，使用 **HTTPS** 协议承载的 SWF 文件只能被其它使用 **HTTPS** 协议承载的 SWF 文件访问。这种实现保持了 **HTTPS** 协议所提供的完整性。

不建议使用此方法覆盖默认行为，因为这样做会损及 **HTTPS** 安全性。但在某些情况下您可能需要这样做；例如，您可能需要允许从为 **Flash Player 6** 发布的 **HTTP** 文件访问为 **Flash Player 7** 或更高版本发布的 **HTTPS** 文件。

为 **Flash Player 6** 发布的 SWF 文件可以使用 `LocalConnection.allowDomain` 事件处理函数来允许从 **HTTP** 到 **HTTPS** 的访问。但是，因为在 **Flash Player 7** 中实现安全性的方式不同，所以，您必须使用 `LocalConnection.allowInsecureDomain()` 方法在为 **Flash Player 7** 或更高版本发布的 SWF 文件中允许此类访问。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**sendingDomain:String** [ 可选 ] - 一个字符串, 指定包含发送方 **LocalConnection** 对象的 SWF 文件的域。

## 示例

下面的示例允许来自当前域或 **www.macromedia.com** 的连接, 或仅允许来自当前域的不安全连接。

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100, 20);
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String) {
    domain_txt.text = sendingDomain;
    return (sendingDomain == this.domain() || sendingDomain == "www.macromedia.com");
};
my_lc.allowInsecureDomain = function(sendingDomain:String) {
    return (sendingDomain == this.domain());
}
my_lc.sayHello = function(name:String) {
    welcome_txt.text = "Hello, "+name;
};
my_lc.connect("lc_name");
```

如果您的 SWF 文件包括第 2 版组件, 请使用第 2 版组件的 **DepthManager** 类而不是此示例中所使用的 **MovieClip.getNextHighestDepth()** 方法。

## 另请参见

[allowDomain \(LocalConnection.allowDomain 处理函数\)](#), [connect \(LocalConnection.connect 方法\)](#)

## close (LocalConnection.close 方法)

```
public close() : Void
```

关闭（断开连接）**LocalConnection** 对象。在您不再需要该对象来接受命令时（例如，当您要在另一个 SWF 文件中使用相同的 `connectionName` 参数发出 `LocalConnection.connect()` 命令时）发出此命令。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例在您单击名为 `close_button` 的 **Button** 组件实例时关闭名为 `receiving_lc` 的连接:

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 42, 100,44);
```

```
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.sayHello = function(name:String) {
    welcome_txt.text = "Hello, "+name;
};
receiving_lc.connect("lc_name");
var closeListener:Object = new Object();
closeListener.click = function(evt:Object) {
    receiving_lc.close();
    status_txt.text = "connection closed";
};
close_button.addEventListener("click", closeListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[connect \(LocalConnection.connect 方法\)](#)

## connect (LocalConnection.connect 方法)

```
public connect(connectionName:String) : Boolean
```

准备一个 **LocalConnection** 对象，以接收来自 `LocalConnection.send()` 命令（名为发送方 **LocalConnection** 对象）的命令。与此命令一起使用的对象称作接收方 **LocalConnection** 对象。接收方和发送方对象必须运行于同一台客户端计算机上。

请确保在调用此方法之前定义附加到 `receiving_lc` 的方法，如本节的所有示例所示。

默认情况下，Flash Player 将 `connectionName` 解析为值 `"superdomain :connectionName"`，其中 `superdomain` 是包含 `LocalConnection.connect()` 命令的 SWF 文件的超级域。例如，如果包含接收方 **LocalConnection** 对象的 SWF 文件位于 `www.someDomain.com`，则 `connectionName` 解析为 `"someDomain.com:connectionName"`。（如果 SWF 文件位于客户端计算机上，则分配给 `superdomain` 的值为 `"localhost"`。）

此外，在默认情况下，Flash Player 只允许接收方 **LocalConnection** 对象从连接名称也解析为值 `"superdomain :connectionName"` 的发送方 **LocalConnection** 对象中接受命令。这样，Flash 就使得位于同一个域中的 SWF 文件可以很容易地相互通讯。

如果您仅在同一个域中的 SWF 文件之间实现通讯，请为 `connectionName` 指定一个不以以下划线 (`_`) 开头且不指定域名的字符串（例如 `"myDomain:connectionName"`）。在 `LocalConnection.connect( connectionName )` 命令中使用同一个字符串。

如果您在位于不同域中的 SWF 文件之间实现通讯，则为 `connectionName` 指定一个以下划线 (`_`) 开头的字符串将会使具有接收方 **LocalConnection** 对象的 SWF 文件更易于在域之间活动。下面是两种可能的情形：

- 如果 `connectionName` 字符串不以以下划线 (`_`) 开头，则 Flash Player 会添加一个超级域前缀和一个冒号（例如 `"myDomain:connectionName"`）。虽然这可以确保您的连接不会与其它域中具有同一名称的连接相互冲突，但任何发送方 **LocalConnection** 对象都必须指定此超级域（例如 `"myDomain:connectionName"`）。如果具有接收方 **LocalConnection** 对象的 SWF 被移动到另一个域中，则播放器会更改该前缀，以反映新的超级域（例如 `"anotherDomain:connectionName"`）。所有发送方 **LocalConnection** 对象必须进行手动编辑，以指向这个新超级域。
- 如果 `connectionName` 字符串以下划线开头（例如 `"_connectionName"`），则 Flash Player 不会向该字符串添加前缀。这意味着接收方和发送方 **LocalConnection** 对象都将使用相同的 `connectionName` 字符串。如果接收方对象使用 `LocalConnection.allowDomain` 来指定可以接受任何域中的连接，则具有接收方 **LocalConnection** 对象的 SWF 可以移动到另一个域，而无需更改任何发送方 **LocalConnection** 对象。

有关更多信息，请参见 `LocalConnection.send()` 中对 `connectionName` 的讨论以及 `LocalConnection.allowDomain` 和 `LocalConnection.domain()` 条目。



冒号作为特殊字符，用于分隔超级域和 `connectionName` 字符串。包含冒号的 `connectionName` 字符串无效。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**connectionName:String** - 一个字符串，对应于要与 *receiving\_lc* 进行通讯的 `LocalConnection.send()` 命令中指定的连接名称。

## 返回

Boolean - 一个布尔值：如果在同一台客户端计算机上运行的其它进程都没有使用同一 *connectionName* 参数值发出过此命令，则为 true；否则为 false。

## 示例

下面的示例演示特定域中的 SWF 文件如何调用同一域接收方 SWF 文件中名为 `printOut` 的方法。

首先，使用下面的代码创建一个 SWF 文件：

```
this.createTextField("tf", this.getNextHighestDepth(), 10, 10, 300, 100);
var aLocalConnection:LocalConnection = new LocalConnection();
aLocalConnection.connect("demoConnection");
aLocalConnection.printOut = function(aString:String):Void{
    tf.text += aString;
}
```

然后，使用下面的代码创建第二个文件：

```
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("demoConnection", "printOut", "This is a message from file B.
Hello.");
```

要测试此示例，请运行第一个 SWF 文件，然后运行第二个。

下面是另一个示例。SWF 1 包含以下代码，这些代码创建一个在运行时回放 MP3 文件的新 **Sound** 对象。名为 `playback_pb` 的 **ProgressBar** 显示 MP3 文件的播放进度。名为 `song_lbl` 的 **Label** 组件实例显示 MP3 文件的名称。不同 SWF 文件中的按钮将用于控制使用 **LocalConnection** 对象的回放。

```
var playback_pb:mx.controls.ProgressBar;
var my_sound:Sound;
playback_pb.setStyle("themeColor", "haloBlue");
this.createEmptyMovieClip("timer_mc", this.getNextHighestDepth());
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.playMP3 = function(mp3Path:String, mp3Name:String) {
```

```

song_lbl.text = mp3Name;
playback_pb.indeterminate = true;
my_sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
playback_pb.indeterminate = false;
};
my_sound.onSoundComplete = function() {
delete timer_mc.onEnterFrame;
};
timer_mc.onEnterFrame = function() {
playback_pb.setProgress(my_sound.position, my_sound.duration);
};
my_sound.loadSound(mp3Path, true);
};
receiving_lc.connect("lc_name");

```

**SWF 2** 包含名为 play\_btn 的按钮。单击该按钮时，它将连接到 SWF 1 并传递两个变量。第一个变量包含要进入流的 MP3 文件，第二个变量是在 SWF 1 的 Label 组件实例中显示的文件名。

```

play_btn.onRelease = function() {
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "playMP3", "song1.mp3", "Album - 01 - Song");
};

```

**SWF 3** 包含名为 play\_btn 的按钮。单击该按钮时，它将连接到 SWF 1 并传递两个变量。第一个变量包含要进入流的 MP3 文件，第二个变量是在 SWF 1 的 Label 组件实例中显示的文件名。

```

play_btn.onRelease = function() {
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "playMP3", "song2.mp3", "Album - 02 - Another Song");
};

```

这些示例中使用的 MovieClip.getNextHighestDepth() 方法需要 Flash Player 7 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

另请参见

[send \(LocalConnection.send 方法\)](#), [allowDomain \(LocalConnection.allowDomain 处理函数\)](#), [domain \(LocalConnection.domain 方法\)](#)

## domain (LocalConnection.domain 方法)

public domain() : String

返回一个字符串，表示当前 SWF 文件所在位置的域。

在为 Flash Player 6 发布的 SWF 文件中，返回的字符串是当前 SWF 文件的超级域。例如，如果 SWF 文件位于 **www.macromedia.com**，则此命令将返回 "macromedia.com"。

在为 Flash Player 7 或更高版本发布的 SWF 文件中，返回的字符串是当前 SWF 文件的确切域。例如，如果 SWF 文件位于 **www.macromedia.com**，则此命令将返回 "www.macromedia.com"。

如果当前 SWF 文件是驻留在客户端计算机上的本地文件，则此命令将返回 "localhost"。

此命令最常见的用法是包含发送方 **LocalConnection** 对象的域名作为要在接收方 **LocalConnection** 对象中调用的方法的参数，或者与 **LocalConnection.allowDomain** 一起使用来接受来自指定域中的命令。如果您仅启用位于同一个域的 **LocalConnection** 对象之间的通讯，则可能不需要使用此命令。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 返回

String - 一个字符串，表示当前 SWF 文件的位置所在的域；有关更多信息，请参见“说明”部分。

### 示例

在下面的示例中，接收方 SWF 文件仅接受来自位于同一个域或位于 **macromedia.com** 的 SWF 文件的命令：

```
// If both the sending and receiving SWF files are Flash Player 6,  
// then use the superdomain  
var my_lc:LocalConnection = new LocalConnection();  
my_lc.allowDomain = function(sendingDomain):String{  
    return (sendingDomain==this.domain() || sendingDomain=="macromedia.com");  
}  
  
// If either the sending or receiving SWF file is Flash Player 7 or later,  
// then use the exact domain. In this case, commands from a SWF file posted  
// at www.macromedia.com will be accepted, but those from one posted at  
// a different subdomain, e.g. livedocs.macromedia.com, will not.  
var my_lc:LocalConnection = new LocalConnection();  
my_lc.allowDomain = function(sendingDomain):String{  
    return (sendingDomain==this.domain() ||  
        sendingDomain=="www.macromedia.com");  
}
```

在下面的示例中，位于 `www.yourdomain.com` 的发送方 SWF 文件调用位于 `www.mydomain.com` 的接收方 SWF 文件中的方法。发送方 SWF 文件包含其域名作为它所调用的方法的参数，使接收方 SWF 文件能够返回对正确域中的 `LocalConnection` 对象的应答值。发送方 SWF 文件还指定它将只接受来自位于 `mydomain.com` 的 SWF 文件的命令。

为了便于参考，代码中包含了行号。下面的列表说明了事件序列：

- 接收方 SWF 文件准备在名为 "sum" 的连接（第 11 行）上接收命令。Flash Player 将此连接的名称解析为 "mydomain.com:sum"（请参见 `LocalConnection.connect()`）。
- 发送方 SWF 文件准备在名为 "result" 的 `LocalConnection` 对象（第 67 行）上接收应答。它还指定它将只接受来自位于 `mydomain.com` 的 SWF 文件的命令（第 51 行到第 53 行）。
- 发送方 SWF 文件调用名为 "mydomain.com:sum" 的连接的 `aSum` 方法（第 68 行），然后传递以下参数：其超级域、要接收应答 ("result") 的连接的名称以及 `aSum` 要使用的值（123 和 456）。
- 使用以下值调用 `aSum` 方法（第 6 行）：`sender = "mydomain.com:result"`、`replyMethod = "aResult"`、`n1 = 123` 且 `n2 = 456`。然后执行以下代码行：  
`this.send("mydomain.com:result", "aResult", (123 + 456));`
- `aResult` 方法（第 54 行）显示 `aSum` 返回的值 (579)。

```
// The receiving SWF at http://www.mydomain.com/folder/movie.swf
// contains the following code
```

```
1 var aLocalConnection:LocalConnection = new LocalConnection();
2 aLocalConnection.allowDomain = function()
3 {
4     // Allow connections from any domain
5 }
6 aLocalConnection.aSum = function(sender, replyMethod, n1, n2)
7 {
8     this.send(sender, replyMethod, (n1 + n2));
9 }
10
11 aLocalConnection.connect("sum");
```

```
// The sending SWF at http://www.yourdomain.com/folder/movie.swf
// contains the following code
```

```
50 var lc:LocalConnection = new LocalConnection();
51 lc.allowDomain = function(aDomain) {
52     // Allow connections only from mydomain.com
53 }
54 lc.aResult = function(aParam) {
```



```

55 trace("The sum is " + aParam);
56 }
    // determine our domain and see if we need to truncate it
57 var channelDomain:String = lc.domain();
58 if (getVersion() >= 7 && this.getSWFVersion() >= 7)
59 {
    // split domain name into elements
60 var domainArray:Array = channelDomain.split(".");

    // if more than two elements are found,
    // chop off first element to create superdomain
61 if (domainArray.length > 2)
62 {
63 domainArray.shift();
64 channelDomain = domainArray.join(".");
65 }
66 }

67 lc.connect("result");
68 lc.send("mydomain.com:sum", "aSum", channelDomain + ':' + "result",
"aResult", 123, 456);

```

另请参见

[allowDomain \(LocalConnection.allowDomain 处理函数\)](#), [connect \(LocalConnection.connect 方法\)](#)

## LocalConnection 构造函数

```
public LocalConnection()
```

创建 **LocalConnection** 对象。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

以下示例显示接收方 SWF 文件和发送方 SWF 文件如何创建 **LocalConnnection** 对象。这两个 SWF 文件可以为其各自的 **LocalConnection** 对象使用相同的名称或不同的名称。在此示例中，它们使用不同的名称。

```

// Code in the receiving SWF file
this.createTextField("result_txt", 1, 10, 10, 100, 22);
result_txt.border = true;
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.methodToExecute = function(param1:Number, param2:Number) {
    result_txt.text = param1+param2;
};
receiving_lc.connect("lc_name");

```

下面的 SWF 文件将请求发送到第一个 SWF 文件。

```
// Code in the sending SWF file
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", 5, 7);
```

另请参见

[connect \(LocalConnection.connect 方法\)](#), [send \(LocalConnection.send 方法\)](#)

## onStatus (LocalConnection.onStatus 处理函数)

```
onStatus = function(info:Object:Object) {}
```

当发送方 **LocalConnection** 对象尝试将命令发送到接收方 **LocalConnection** 对象之后调用。如果要对此事件处理函数做出响应，则必须创建一个函数来处理 **LocalConnection** 对象所发送的信息对象。

如果此事件处理函数返回的信息对象包含 **status** 的 **level** 值，则表明 **Flash** 已将命令成功发送到接收方 **LocalConnection** 对象。这并不意味着 **Flash** 已成功调用接收方 **LocalConnection** 对象的指定方法，而只表示 **Flash** 能够发送该命令。例如，如果接收方 **LocalConnection** 对象不允许从发送方域建立连接，或者该方法不存在，则不调用该方法。确知是否已调用该方法的唯一方式是让接收方对象向发送方对象发送应答。

如果此事件处理函数返回的信息对象包含 **error** 的 **level** 值，则表明 **Flash** 不能将该命令发送到接收方 **LocalConnection** 对象，这最可能是因为没有存在与调用此处理函数的 `sending_lc.send()` 命令中指定的名称相符的接收方 **LocalConnection** 对象。

除了此 `onStatus` 处理函数外，**Flash** 还提供名为 `System.onStatus` 的“超级”函数。如果为特定对象调用了 `onStatus` 但未分配任何函数对其进行响应，则 **Flash** 将处理分配到 `System.onStatus` 的函数（如果存在）。

大多数情况下，实现此处理函数只是为了对错误条件做出响应，如以下示例所示。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**info:Object:Object** - 根据状态消息定义的参数。有关此参数的详细信息，请参见“说明”部分。

## 示例

下面的示例显示有关 SWF 文件是否连接到名为 lc\_name 的另一个本地连接对象的状态消息。名为 name\_ti 的 **TextInput** 组件、名为 status\_ta 的 **TextArea** 实例以及名为 send\_button 的 **Button** 实例都用于显示内容。

```
var sending_lc:LocalConnection;
var sendListener:Object = new Object();
sendListener.click = function(evt:Object) {
    sending_lc = new LocalConnection();
    sending_lc.onStatus = function(info:Object:Object) {
        switch (info.level) {
            case 'status' :
                status_ta.text = "LocalConnection connected successfully.";
                break;
            case 'error' :
                status_ta.text = "LocalConnection encountered an error.";
                break;
        }
    };
    sending_lc.send("lc_name", "sayHello", name_ti.text);
};
send_button.addEventListener("click", sendListener);
```

另请参见

[send \(LocalConnection.send 方法\)](#), [onStatus \(System.onStatus 处理函数\)](#)

## send (LocalConnection.send 方法)

```
public send(connectionName:String, methodName:String, [args:Object]) :
    Boolean
```

在使用 `LocalConnection.connect( connectionName )` 命令 (接收方 **LocalConnection** 对象) 打开的连接上调用名为 `method` 的方法。与此命令一起使用的对象称作“发送方 **LocalConnection** 对象”。包含发送方对象的 SWF 文件和包含接收方对象的 SWF 文件必须要在同一台客户端计算机上运行。

您能够以参数形式传递给此命令的数据量限制为 40 千字节。如果命令返回 `false`, 但是您的语法是正确的, 请尝试将 `LocalConnection.send()` 请求分为多个命令, 每一个命令的数据量不超过 40K。

如 `LocalConnection.connect()` 条目中所述, **Flash** 在默认情况下会将当前超级域添加到 `connectionName`。如果您要在不同的域之间实现通讯, 则在发送方 **LocalConnection** 对象和接收方 **LocalConnection** 对象中都需要定义 `connectionName`, 这样就使 **Flash** 不会将当前超级域添加到 `connectionName`。您可以使用以下两种方法中的一种实现这一目的:

- 在发送方 **LocalConnection** 对象和接收方 **LocalConnection** 对象中 `connectionName` 的开头使用下划线 (`_`)。在包含接收方对象的 SWF 文件中, 使用 `LocalConnection.allowDomain` 指定将接受来自任何域的连接。这一实现使您可以在任何域中存储发送方 SWF 文件和接收方 SWF 文件。
- 在发送方 **LocalConnection** 对象中包含 `connectionName` 中的超级域, 例如 `myDomain.com:myConnectionName`。在接收方对象中, 使用 `LocalConnection.allowDomain` 指定将接受来自指定超级域的连接 (本例中为 `myDomain.com`), 或者接受来自任何域的连接。



不能在接收方 **LocalConnection** 对象中指定 `connectionName` 中的超级域 (只能在发送方 **LocalConnection** 对象中指定)。

使用此方法时, 请考虑 **Flash Player** 安全模型。默认情况下, **LocalConnection** object 与创建它的 SWF 文件的沙箱相关联, 并且不允许对 **LocalConnection** 对象进行跨域调用, 除非调用了 `LocalConnection.allowDomain()` 方法。

有关更多信息, 请参见以下部分:

- 《学习 Flash 中的 **ActionScript 2.0**》的第 17 章, “了解安全性”
- **Flash Player 8** 安全性白皮书 (位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security))
- **Flash Player 8** 与安全相关的 API 白皮书 (位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis))

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 参数

**connectionName:String** - 一个字符串, 对应于要与 *sending\_lc* 进行通讯的 `LocalConnection.connect()` 命令中指定的连接名称。

**methodName:String** - 一个字符串, 指定要在接收方 **LocalConnection** 对象中调用的方法的名称。以下方法名称会导致该命令失败: `send`、`connect`、`close`、`domain`、`onStatus` 和 `allowDomain`。

**args:Object [ 可选 ]** - 要传递给指定方法的参数。

## 返回

**Boolean** - 一个布尔值: 如果 **Flash** 能够执行该请求, 则为 `true`; 否则为 `false`。



返回值为 `true` 并不一定表示 **Flash** 已成功连接到接收方 **LocalConnection** 对象; 而只表示该命令的语法正确。若要确定连接是否成功, 请参见 `LocalConnection.onStatus`。

## 示例

有关位于同一个域中的 **LocalConnection** 对象之间的通讯的示例，请参见 `LocalConnection.connect()`。有关位于任何域的 **LocalConnection** 对象之间的通讯的示例，请参见 `LocalConnection.allowDomain`。有关位于指定域的 **LocalConnection** 对象之间的通讯的示例，请参见 `LocalConnection.allowDomain` 和 `LocalConnection.domain()`。

## 另请参见

`allowDomain` (`LocalConnection.allowDomain` 处理函数), `connect` (`LocalConnection.connect` 方法), `domain` (`LocalConnection.domain` 方法), `onStatus` (`LocalConnection.onStatus` 处理函数)

# Locale (mx.lang.Locale)

```
Object
|
+-mx.lang.Locale
```

```
public class Locale
extends Object
```

使用 **mx.lang.Locale** 类，可以控制多语言文本在 SWF 文件中的显示方式。借助“Flash 字符串”面板，可以在动态文本字段中使用字符串 ID 替代字符串。这样，您就可以创建一个 SWF 文件，用它来显示从特定于语言的 XML 文件加载的文本。XML 文件必须使用“XML 本地化交换文件格式” (XLIFF)。要显示 XLIFF 文件中包含的特定于语言的字符串，有三种方法：

- "automatically at runtime" - Flash Player 使用来自 XML 文件的字符串替换字符串 ID，该 XML 文件与 `System.capabilities.language` 返回的默认系统语言代码相匹配。
- "manually using stage language" - 在编译时使用字符串替换字符串 ID，并且无法由 Flash Player 更改字符串 ID。
- "via ActionScript at runtime" - 在运行时使用 **ActionScript** 控制字符串 ID 替换。利用此选项，您可以控制字符串 ID 替换的时间和语言。

当您要在“运行时通过 **ActionScript**”来替换字符串 ID 时，可以使用此类的属性和方法。所有可用的属性和方法都是静态的，这意味着要通过 **mx.lang.Locale** 类本身而不是通过该类的实例来访问它们。

**注意:** **Locale** 类不同于 **ActionScript 2.0** 语言参考中的其它类，因为该类不是 Flash Player 的一部分。由于此类被安装在 Flash 创作类路径中，所以它会自动编译到您的 SWF 文件中。由于 **Locale** 类将编译到 SWF 中，所以使用该类会稍微增加 SWF 的大小。

可用性: **ActionScript 2.0** ; **Flash Player 7**

属性摘要

修饰符	属性	说明
static	autoReplace:Boolean	确定是否在加载 XML 文件后自动替换字符串。
static	languageCodeArray:Array [ 只读 ]	包含语言代码的数组，这些代码对应着指定的语言或加载到 FLA 文件中的语言。
static	stringIDArray:Array [ 只读 ]	包含 FLA 文件中所有字符串 ID 的数组。

继承自 Object 类的属性

<a href="#">constructor</a> (Object.constructor 属性), <a href="#">__proto__</a> (Object.__proto__ 属性), <a href="#">prototype</a> (Object.prototype 属性), <a href="#">__resolve</a> (Object.__resolve 属性)
--

方法摘要

修饰符	签名	说明
static	addDelayedInstance(instance:Object, stringID:String) : Void	将 { 实例, 字符串 ID } 对添加到内部数组中，供以后使用。
static	addXMLPath(langCode:String, path:String) : Void	将 { 语言代码和语言路径 } 对添加到内部数组中，供以后使用。
static	checkXMLStatus() : Boolean	如果加载了 XML 文件，则返回 true；否则返回 false。
static	getDefaultLang() : String	在“字符串面板”对话框中或通过调用 setDefaultLang() 方法设置的默认语言代码。
static	initialize() : Void	自动确定要使用的语言并自动加载 XML 语言文件。
static	loadLanguageXML(xmlLanguageCode:String, customXmlCompleteCallback:Function) : Void	加载指定的 XML 语言文件。
static	loadString(id:String) : String	使用当前语言返回与给定字符串 ID 关联的字符串值。
static	loadStringEx(stringID:String, languageCode:String) : String	返回与给定字符串 ID 和语言代码关联的字符串值。

修饰符	签名	说明
static	setDefaultLang(langCode:String) : Void	设置默认语言代码。
static	setLoadCallback(loadCallback:Function) : Void	设置在加载 XML 文件后调用的回调函数。
static	setString(stringID:String, languageCode:String, stringValue:String) : Void	设置给定字符串 ID 和语言代码的新字符串值。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## addDelayedInstance (Locale.addDelayedInstance 方法)

```
public static addDelayedInstance(instance:Object, stringID:String) : Void
```

将 { 实例, 字符串 ID } 对添加到内部数组中, 供以后使用。它主要由 **Flash** 在字符串替换方法为 "automatically at runtime" 时使用。

可用性: **ActionScript 2.0** ; **Flash Player 7**

### 参数

**instance:Object** - 要填充的文本字段的实例名称。

**stringID:String** - 语言字符串 ID。

### 示例

下面的示例使用 `autoReplace` 属性和 `addDelayedInstance()` 方法来用英语 XML 语言文件中的 `IDS_GREETING` 字符串填充舞台上的文本字段。

```
import mx.lang.Locale;
greeting_txt.autoSize = "left";
Locale.autoReplace = true;
Locale.addDelayedInstance(greeting_txt, "IDS_GREETING");
Locale.loadLanguageXML("en");
```

## addXMLPath (Locale.addXMLPath 方法)

```
public static addXMLPath(langCode:String, path:String) : Void
```

将 { 语言代码和语言路径 } 对添加到内部数组中，供以后使用。它主要由 **Flash** 在字符串替换方法为 "automatically at runtime" 或 "via ActionScript at runtime" 时使用。

可用性: ActionScript 2.0 ; Flash Player 7

### 参数

**langCode:String** - 语言代码。

**path:String** - 要添加的 XML 路径。

### 示例

下面的示例使用 setInterval() 方法来检查是否已成功加载语言 XML 文件。

```
import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("en");
// create interval to check if language XML file is loaded
var locale_int:Number = setInterval(checkLocaleStatus, 10);
function checkLocaleStatus():Void {
    if (Locale.checkXMLStatus()) {
        clearInterval(locale_int);
        trace("clearing interval @ " + getTimer() + " ms");
    }
}
// callback function for Locale.setLoadCallback()
function localeCallback(success:Boolean):Void {
    greeting_txt.text = Locale.loadString("IDS_GREETING");
}
```

## autoReplace (Locale.autoReplace 属性)

```
public static autoReplace : Boolean
```

确定是否在加载 XML 文件后自动替换字符串。如果该属性设置为 true，则文本替换方法等效于“字符串”面板设置 "automatically at runtime"。这意味着 **Flash Player** 将确定承载环境的默认语言，并自动使用该语言显示文本。如果该属性设置为 false，则文本替换方法等效于“字符串”面板设置 "via ActionScript at runtime"。这意味着由您负责加载相应的 XML 文件以显示文本。

此属性的默认值反映了您在“字符串”面板对话框中为“替换”字符串选择的设置：true 表示 "automatically at runtime"（默认设置），而 false 表示“运行时通过 ActionScript”。

可用性: ActionScript 2.0 ; Flash Player 8



## 示例

下面的示例使用 `Locale.autoReplace` 属性来用英语 XML 文件中的 `IDS_GREETING` 字符串的内容填充舞台上动态创建的 `greeting_txt` 文本字段。在“字符串”面板中，单击“设置”按钮以打开“设置”对话框。您可以使用“设置”对话框添加两种活动语言：英语 (**en**) 和法语 (**fr**)，将“替换字符串”单选选项设置为“via ActionScript at runtime”，然后单击“确定”。最后，在“字符串”面板中输入 `IDS_GREETING` 的字符串 ID，然后添加适合每种活动语言的文本。

```
import mx.lang.Locale;
this.createTextField("greeting_txt", 10, 40, 40, 200, 20);
greeting_txt.autoSize = "left";
Locale.autoReplace = true;
Locale.addDelayedInstance(greeting_txt, "IDS_GREETING");
Locale.loadLanguageXML("en");
```

## checkXMLStatus (Locale.checkXMLStatus 方法)

```
public static checkXMLStatus() : Boolean
```

如果加载了 XML 文件，则返回 `true`；否则返回 `false`。

可用性：ActionScript 2.0；Flash Player 7

## 返回

`Boolean` - 如果加载了 XML 文件，则返回 `true`；否则返回 `false`。

## 示例

下面的示例将使用间隔，每隔 10 毫秒查看一次是否已成功加载语言文件。加载 XML 文件后，用语言 XML 文件中的 `IDS_GREETING` 字符串填充舞台上的 `greeting_txt` 文本字段实例。

```
import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("en");
// create interval to check if language XML file is loaded
var locale_int:Number = setInterval(checkLocaleStatus, 10);
function checkLocaleStatus():Void {
    if (Locale.checkXMLStatus()) {
        clearInterval(locale_int);
        trace("clearing interval @ " + getTimer() + " ms");
    }
}
// callback function for Locale.setLoadCallback()
function localeCallback(success:Boolean):Void {
    greeting_txt.text = Locale.loadString("IDS_GREETING");
}
```

## getDefaultLang (Locale.getDefaultLang 方法)

public static getDefaultLang() : String

在“字符串面板”对话框中或通过调用 `setDefaultLang()` 方法设置的默认语言代码。

可用性: **ActionScript 2.0** ; **Flash Player 8**

### 返回

String - 返回默认的语言代码。

### 示例

下面的示例将创建一个名为 `defLang` 的变量，它用于保存 **Flash** 文档的初始默认语言。单击“字符串”面板中的“设置”按钮以启动“设置”对话框。然后添加两种活动语言：英语 (**en**) 和法语 (**fr**)，将“替换字符串”单选控制设置为 "via ActionScript at runtime"，然后单击“确定”。在“字符串”面板中，添加 **IDS\_GREETING** 的字符串 ID，然后添加适合每种活动语言的文本。

```
import mx.lang.Locale;
var defLang:String = "fr";
Locale.setDefaultLang(defLang);
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML(Locale.getDefaultLang());
function localeCallback(success:Boolean) {
    if (success) {
        trace(Locale.stringIDArray); // IDS_GREETING
        trace(Locale.loadString("IDS_GREETING"));
    } else {
        trace("unable to load XML");
    }
}
```

### 另请参见

[setDefaultLang \(Locale.setDefaultLang 方法\)](#)

## initialize (Locale.initialize 方法)

```
public static initialize(): Void
```

自动确定要使用的语言并自动加载 XML 语言文件。它主要由 **Flash** 在字符串替换方法为 "automatically at runtime" 时使用。

可用性: **ActionScript 2.0** ; **Flash Player 7**

### 示例

此示例说明如何使用 initialize() 方法自动用用户的当前操作系统语言填充舞台上的 greeting\_txt 文本字段。不直接使用 initialize() 方法，而使用字符串替换方法 "automatically at runtime"。

```
import mx.lang.Locale;
trace(System.capabilities.language);
Locale.autoReplace = true;
Locale.addDelayedInstance(greeting_txt, "IDS_GREETING");
Locale.initialize();
```

## languageCodeArray (Locale.languageCodeArray 属性)

```
public static languageCodeArray : Array [read-only]
```

包含语言代码的数组，这些代码对应着指定的语言或加载到 **FLA** 文件中的语言。语言代码不是按字母顺序排序的。

可用性: **ActionScript 2.0** ; **Flash Player 8**

### 示例

下面的示例将根据 **ComboBox** 组件的当前值加载语言 XML 文件。将 **ComboBox** 组件拖放到舞台上，并指定它的实例名称为 lang\_cb。使用“文本”工具，创建一个动态文本字段，并指定它的实例名称为 greeting\_txt。在“字符串”面板中，添加至少两种活动语言，将“替换字符串”单选选项设置为 "via ActionScript at runtime"，然后单击“确定”。之后，添加 IDS\_GREETING 的字符串 ID，然后输入适合每种活动语言的文本。最后，将以下 **ActionScript** 代码添加到主时间轴的第一帧中：

```
import mx.lang.Locale;
Locale.setLoadCallback(localeListener);
lang_cb.dataProvider = Locale.languageCodeArray.sort();
lang_cb.addEventListener("change", langListener);

function langListener(eventObj:Object):Void {
    Locale.loadLanguageXML(eventObj.target.value);
}
function localeListener(success:Boolean):Void {
    if (success) {
        greeting_txt.text = Locale.loadString("IDS_GREETING");
    }
}
```

```

    } else {
        greeting_txt.text = "unable to load language XML file.";
    }
}

```

## loadLanguageXML (Locale.loadLanguageXML 方法)

```

public static loadLanguageXML(xmlLanguageCode:String,
    customXmlCompleteCallback:Function) : Void

```

加载指定的 XML 语言文件。

可用性: **ActionScript 2.0** ; **Flash Player 8**

### 参数

**xmlLanguageCode:String** - 要加载的 XML 语言文件的语言代码。

**customXmlCompleteCallback:Function** - 在加载 XML 语言文件时调用的自定义回调函数。

### 示例

下面的示例使用 loadLanguageXML() 方法加载英语 (**en**) XML 语言文件。加载该语言文件后, 将调用 localeCallback() 方法, 并用 XML 文件中的 IDS\_GREETING 字符串的内容填充舞台上的 greeting\_txt 文本字段。

```

import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("en");
// create interval to check if language XML file is loaded
var locale_int:Number = setInterval(checkLocaleStatus, 10);
function checkLocaleStatus():Void {
    if (Locale.checkXMLStatus()) {
        clearInterval(locale_int);
        trace("clearing interval @ " + getTimer() + " ms");
    }
}
// callback function for Locale.setLoadCallback()
function localeCallback(success:Boolean):Void {
    greeting_txt.text = Locale.loadString("IDS_GREETING");
}

```

## loadString (Locale.loadString 方法)

`public static loadString(id:String) : String`

使用当前语言返回与给定字符串 ID 关联的字符串值。

可用性: **ActionScript 2.0** ; **Flash Player 7**

### 参数

`id:String` - 要加载的字符串标识 (ID) 号。

### 返回

`String` - 与给定字符串 ID 关联的字符串值, 采用当前语言。

### 示例

下面的示例将使用间隔, 每隔 10 毫秒查看一次是否已成功加载语言文件。加载 XML 文件后, 用 XML 语言文件的 IDS\_GREETING 字符串填充舞台上的 greeting\_txt 文本字段实例。

```
import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("en");
// create interval to check if language XML file is loaded
var locale_int:Number = setInterval(checkLocaleStatus, 10);
function checkLocaleStatus():Void {
    if (Locale.checkXMLStatus()) {
        clearInterval(locale_int);
        trace("clearing interval @ " + getTimer() + " ms");
    }
}
// callback function for Locale.setLoadCallback()
function localeCallback(success:Boolean):Void {
    greeting_txt.text = Locale.loadString("IDS_GREETING");
}
```

### 另请参见

[loadStringEx \(Locale.loadStringEx 方法\)](#)

## loadStringEx (Locale.loadStringEx 方法)

`public static loadStringEx(stringID:String, languageCode:String) : String`

返回与给定字符串 **ID** 和语言代码关联的字符串值。为了避免加载意外的 **XML** 文件，如果 **XML** 文件尚未加载，则 `loadStringEx()` 不会加载 **XML** 语言文件。如果您想加载某个 **XML** 语言文件，则应该确定在正确的时间来调用 `loadLanguageXML()` 方法。

可用性：ActionScript 2.0；Flash Player 8

### 参数

`stringID:String` - 要加载的字符串标识 (**ID**) 号。

`languageCode:String` - 语言代码。

### 返回

`String` - 与给定字符串 **ID** 相关联的字符串值，采用由 `languageCode` 参数指定的语言。

### 示例

下面的示例使用 `loadStringEx()` 方法跟踪当前加载的法语 **XML** 文件的 `IDS_GREETING` 字符串的值。

```
import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("fr");
function localeCallback(success:Boolean) {
    trace(success);
    trace(Locale.stringIDArray); // IDS_GREETING
    trace(Locale.loadStringEx("IDS_GREETING", "fr")); // bonjour
}
```

### 另请参见

[loadString \(Locale.loadString 方法\)](#)

## setDefaultLang (Locale.setDefaultLang 方法)

```
public static setDefaultLang(langCode:String) : Void
```

设置默认语言代码。

可用性: **ActionScript 2.0** ; **Flash Player 7**

### 参数

**langCode:String** - 一个表示语言代码的字符串。

### 示例

下面的示例将创建一个名为 **defLang** 的变量，它用于保存 **Flash** 文档的初始默认语言。单击“字符串”面板中的“设置”按钮以打开“设置”对话框。然后添加两种活动语言：英语 (**en**) 和法语 (**fr**)，将“替换字符串”单选控制设置为 "via ActionScript at runtime"，然后单击“确定”。在“字符串”面板中，添加 **IDS\_GREETING** 的字符串 ID，然后添加适合每种活动语言的文本。

```
import mx.lang.Locale;
var defLang:String = "fr";
Locale.setDefaultLang(defLang);
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML(Locale.getDefaultLang());
function localeCallback(success:Boolean) {
    if (success) {
        trace(Locale.stringIDArray); // IDS_GREETING
        trace(Locale.loadString("IDS_GREETING"));
    } else {
        trace("unable to load XML");
    }
}
```

### 另请参见

[getDefaultLang \(Locale.getDefaultLang 方法\)](#)

## setLoadCallback (Locale.setLoadCallback 方法)

```
public static setLoadCallback(loadCallback:Function) : Void
```

设置在加载 XML 文件后调用的回调函数。

可用性: **ActionScript 2.0** ; **Flash Player 7**

### 参数

**loadCallback:Function** - 在加载 XML 语言文件时调用的函数。

### 示例

下面的示例将使用间隔，每隔 10 毫秒查看一次是否已成功加载语言文件。加载 XML 文件后，用 XML 语言文件的 IDS\_GREETING 字符串填充舞台上的 greeting\_txt 文本字段实例。

```
import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("en");
// create interval to check if language XML file is loaded
var locale_int:Number = setInterval(checkLocaleStatus, 10);
function checkLocaleStatus():Void {
    if (Locale.checkXMLStatus()) {
        clearInterval(locale_int);
        trace("clearing interval @ " + getTimer() + " ms");
    }
}
// callback function for Locale.setLoadCallback()
function localeCallback(success:Boolean):Void {
    greeting_txt.text = Locale.loadString("IDS_GREETING");
}
```

## setString (Locale.setString 方法)

```
public static setString(stringID:String, languageCode:String,
    stringValue:String) : Void
```

设置给定字符串 ID 和语言代码的新字符串值。

可用性: **ActionScript 2.0** ; **Flash Player 8**

### 参数

**stringID:String** - 要设置的字符串标识 (ID) 号。

**languageCode:String** - 语言代码。

**stringValue:String** - 一个字符串值。



## 示例

下面的示例使用 `setString()` 方法设置英语 (**en**) 和法语 (**fr**) 的 `IDS_WELCOME` 字符串。

```
import mx.lang.Locale;
Locale.setString("IDS_WELCOME", "en", "hello");
Locale.setString("IDS_WELCOME", "fr", "bonjour");
trace(Locale.loadStringEx("IDS_WELCOME", "en")); // hello
```

## stringIDArray (Locale.stringIDArray 属性)

`public static stringIDArray : Array [read-only]`

包含 FLA 文件中所有字符串 ID 的数组。字符串 ID 不是按字母顺序排序的。

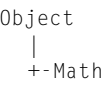
可用性: **ActionScript 2.0 ; Flash Player 8**

## 示例

下面的示例将跟踪当前加载的语言 XML 文件的 `Locale.stringIDArray` 属性。单击“字符串”面板中的“设置”按钮以打开“设置”对话框。下一步，添加两种活动语言：英语 (**en**) 和法语 (**fr**)，将“替换字符串”单选控制设置为“via ActionScript at runtime”，然后单击“确定”。在“字符串”面板中，添加 `IDS_GREETING` 的字符串 ID，然后添加适合每种活动语言的文本。

```
import mx.lang.Locale;
Locale.setLoadCallback(localeCallback);
Locale.loadLanguageXML("fr");
function localeCallback(success:Boolean) {
    trace(success);
    trace(Locale.stringIDArray); // IDS_GREETING
    trace(Locale.loadStringEx("IDS_GREETING", "fr")); // bonjour
}
```

# Math



```
public class Math
extends Object
```

**Math** 类是一个顶级类，不必使用构造函数即可使用其方法和属性。

使用此类的方法和属性可以访问和处理数学常数和函数。**Math** 类的所有属性和方法都是静态的，而且必须使用语法 `Math.method( parameter )` 或 `Math.constant` 才能调用它们。在 **ActionScript** 中，使用双精度 **IEEE-754** 浮点数的最高精度定义常数。

**Math** 类的多个方法都使用以弧度为单位的角度作为参数。在调用方法前，您可以使用以下等式来计算弧度值，然后提供计算出的值作为参数；或者您可以提供等式的整个右边（用弧度替代 `degrees`）作为弧度参数。

若要计算弧度值，请使用以下公式：

$$\text{radians} = \text{degrees} * \text{Math.PI}/180$$

下面是将等式作为参数进行传递以计算 45° 角的正弦值的示例：

```
Math.sin(45 * Math.PI/180) 等同于 Math.sin(.7854)
```

可用性：ActionScript 1.0 ； Flash Player 5

## 属性摘要

修饰符	属性	说明
static	E:Number	代表自然对数的底的数学常数，表示为 <b>e</b> 。
static	LN10:Number	代表 10 的自然对数的数学常数，表示为 <code>log<sub>e</sub>10</code> ，其近似值为 2.302585092994046。
static	LN2:Number	代表 2 的自然对数的数学常数，表示为 <code>log<sub>e</sub>2</code> ，其近似值为 0.6931471805599453。
static	LOG10E:Number	代表常数 <b>e</b> ( <code>Math.E</code> ) 以 10 为底的对数的数学常数，表示为 <code>log<sub>10</sub>e</code> ，其近似值为 0.4342944819032518。
static	LOG2E:Number	代表常数 <b>e</b> ( <code>Math.E</code> ) 以 2 为底的对数的数学常数，表示为 <code>log<sub>2</sub>e</code> ，其近似值为 1.442695040888963387。
static	PI:Number	代表一个圆的周长与其直径的比值的数学常数，表示为 <b>pi</b> ，其近似值为 3.141592653589793。

修饰符	属性	说明
static	SQRT1_2:Number	代表 1/2 的平方根的数学常数，其近似值为 0.7071067811865476。
static	SQRT2:Number	代表 2 的平方根的数学常数，其近似值为 1.4142135623730951。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

### 方法摘要

修饰符	签名	说明
static	abs(x:Number) : Number	计算并返回由参数 x 指定的数字的绝对值。
static	acos(x:Number) : Number	以弧度为单位计算并返回由参数 x 指定的数字的反余弦值。
static	asin(x:Number) : Number	以弧度为单位计算并返回由参数 x 指定的数字的正弦值。
static	atan(tangent:Number) : Number	以弧度为单位计算并返回角度值，该角度的正切值已由参数 tangent 指定。
static	atan2(y:Number, x:Number) : Number	以弧度为单位计算并返回点 y/x 的角度，该角度从圆的 x 轴（O 点在其上，O 表示圆心）沿逆时针方向测量。
static	ceil(x:Number) : Number	返回指定数字或表达式的上限值。
static	cos(x:Number) : Number	以弧度为单位计算并返回指定角度的余弦值。
static	exp(x:Number) : Number	返回自然对数的底 (e) 的 x 次幂的值，x 由参数 x 指定。
static	floor(x:Number) : Number	返回由参数 x 指定的数字或表达式的下限值。
static	log(x:Number) : Number	返回参数 x 的自然对数。
static	max(x:Number, y:Number) : Number	计算 x 和 y，并返回两者中的较大值。
static	min(x:Number, y:Number) : Number	计算 x 和 y，并返回两者中的较小值。
static	pow(x:Number, y:Number) : Number	计算并返回 x 的 y 次幂。
static	random() : Number	返回一个伪随机数 n，其中 0 <= n < 1。

修饰符	签名	说明
static	round(x:Number) : Number	将参数 x 的值向上或向下舍入为最接近的整数并返回该值。
static	sin(x:Number) : Number	以弧度为单位计算并返回指定角度的正弦值。
static	sqrt(x:Number) : Number	计算并返回指定数字的平方根。
static	tan(x:Number) : Number	计算并返回指定角度的正切值。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## abs (Math.abs 方法)

`public static abs(x:Number) : Number`

计算并返回由参数 x 指定的数字的绝对值。

可用性: `ActionScript 1.0` ; `Flash Player 5`

### 参数

`x:Number` - 一个数字。

### 返回

`Number` - 一个数字。

### 示例

下面的示例说明 `Math.abs()` 如何返回一个数字的绝对值，而不影响 `x` 参数（在此示例中名为 `num`）的值：

```
var num:Number = -12;
var numAbsolute:Number = Math.abs(num);
trace(num); // output: -12
trace(numAbsolute); // output: 12
```

## acos (Math.acos 方法)

`public static acos(x:Number) : Number`

以弧度为单位计算并返回由参数 *x* 指定的数字的反余弦值。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

*x*:Number - 从 **-1.0** 到 **1.0** 的一个数字。

### 返回

Number - 一个数字; 参数 *x* 的反余弦值。

### 示例

下面的示例显示多个值的反余弦值。

```
trace(Math.acos(-1)); // output: 3.14159265358979
trace(Math.acos(0)); // output: 1.5707963267949
trace(Math.acos(1)); // output: 0
```

### 另请参见

[asin \(Math.asin 方法\)](#), [atan \(Math.atan 方法\)](#), [atan2 \(Math.atan2 方法\)](#), [cos \(Math.cos 方法\)](#), [sin \(Math.sin 方法\)](#), [tan \(Math.tan 方法\)](#)

## asin (Math.asin 方法)

`public static asin(x:Number) : Number`

以弧度为单位计算并返回由参数 *x* 指定的数字的反正弦值。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

*x*:Number - 从 **-1.0** 到 **1.0** 的一个数字。

### 返回

Number - 介于负二分之 **pi** 和正二分之 **pi** 之间的一个数字。

## 示例

下面的示例显示多个值的反正弦值。

```
trace(Math.asin(-1)); // output: -1.5707963267949
trace(Math.asin(0)); // output: 0
trace(Math.asin(1)); // output: 1.5707963267949
```

另请参见

[acos \(Math.acos 方法\)](#), [atan \(Math.atan 方法\)](#), [atan2 \(Math.atan2 方法\)](#), [cos \(Math.cos 方法\)](#), [sin \(Math.sin 方法\)](#), [tan \(Math.tan 方法\)](#)

## atan (Math.atan 方法)

```
public static atan(tangent:Number) : Number
```

以弧度为单位计算并返回角度值，该角度的正切值已由参数 `tangent` 指定。返回值介于负二分之  $\pi$  和正二分之  $\pi$  之间。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**tangent:**Number - 表示角的正切值的一个数字。

## 返回

Number - 介于负二分之  $\pi$  和正二分之  $\pi$  之间的一个数字。

## 示例

下面的示例显示若干个正切的角度值。

```
trace(Math.atan(-1)); // output: -0.785398163397448
trace(Math.atan(0)); // output: 0
trace(Math.atan(1)); // output: 0.785398163397448
```

另请参见

[acos \(Math.acos 方法\)](#), [asin \(Math.asin 方法\)](#), [atan2 \(Math.atan2 方法\)](#), [cos \(Math.cos 方法\)](#), [sin \(Math.sin 方法\)](#), [tan \(Math.tan 方法\)](#)

## atan2 (Math.atan2 方法)

```
public static atan2(y:Number, x:Number) : Number
```

以弧度为单位计算并返回点  $y/x$  的角度，该角度从圆的  $x$  轴（0 点在其上，0 表示圆心）沿逆时针方向测量。返回值介于正  $\pi$  和负  $\pi$  之间。

可用性：ActionScript 1.0；Flash Player 5

### 参数

$y$ :Number - 指定点的  $y$  坐标的数字。

$x$ :Number - 指定点的  $x$  坐标的数字。

### 返回

Number - 一个数字。

### 示例

下面的示例以弧度为单位返回由坐标 (0, 10) 指定的点的角度，即  $x = 0$  且  $y = 10$ 。请注意 `atan2` 的第一个参数始终是  $y$  坐标。

```
trace(Math.atan2(10, 0)); // output: 1.5707963267949
```

### 另请参见

`acos` (Math.acos 方法), `asin` (Math.asin 方法), `atan` (Math.atan 方法), `cos` (Math.cos 方法), `sin` (Math.sin 方法), `tan` (Math.tan 方法)

## ceil (Math.ceil 方法)

```
public static ceil(x:Number) : Number
```

返回指定数字或表达式的上限值。数字的上限值是大于是等于该数字的最接近的整数。

可用性：ActionScript 1.0；Flash Player 5

### 参数

$x$ :Number - 一个数字或表达式。

## 返回

Number - 最接近且大于等于参数 x 的一个整数。

## 示例

下面的代码返回一个值 **13**：

```
Math.ceil(12.5);
```

## 另请参见

[floor \(Math.floor 方法\)](#), [round \(Math.round 方法\)](#)

# cos (Math.cos 方法)

```
public static cos(x:Number) : Number
```

以弧度为单位计算并返回指定角度的余弦值。若要计算弧度，请参见 **Math** 类条目的说明。

可用性：ActionScript 1.0；Flash Player 5

## 参数

x:Number - 一个数字，它表示一个以弧度为单位的角度。

## 返回

Number - 从 **-1.0** 到 **1.0** 的一个数字。

## 示例

下面的示例显示多个不同角度的余弦值。

```
trace (Math.cos(0)); // 0 degree angle. Output: 1
trace (Math.cos(Math.PI/2)); // 90 degree angle. Output: 6.12303176911189e-
    17
trace (Math.cos(Math.PI)); // 180 degree angle. Output: -1
trace (Math.cos(Math.PI*2)); // 360 degree angle. Output: 1
```

**注意：**90 度角的余弦值为零，但由于使用二进制数进行十进制计算所固有的不准确性，Flash Player 将报告一个非常接近但不完全等于零的数字。

## 另请参见

[acos \(Math.acos 方法\)](#), [asin \(Math.asin 方法\)](#), [atan \(Math.atan 方法\)](#), [atan2 \(Math.atan2 方法\)](#), [sin \(Math.sin 方法\)](#), [tan \(Math.tan 方法\)](#)



## E (Math.E 属性)

public static E : Number

代表自然对数的底的数学常数，表示为 **e**。**e** 的近似值为 2.71828182845905。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

此示例说明如何使用 Math.E 为一年期 100% 利息的简单案例连续计算复利。

```
var principal:Number = 100;
var simpleInterest:Number = 100;
var continuouslyCompoundedInterest:Number = (100 * Math.E) - principal;

trace ("Beginning principal: $" + principal);
trace ("Simple interest after one year: $" + simpleInterest);
trace ("Continuously compounded interest after one year: $" +
    continuouslyCompoundedInterest);

//
Output:
Beginning principal: $100
Simple interest after one year: $100
Continuously compounded interest after one year: $171.828182845905
```

## exp (Math.exp 方法)

public static exp(x:Number) : Number

返回自然对数的底 (**e**) 的 **x** 次幂的值，**x** 由参数 **x** 指定。常数 Math.E 可以提供 **e** 的值。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

**x**:Number - 指数；一个数字或表达式。

### 返回

Number - 一个数字。

### 示例

下面的示例显示两个数值的对数。

```
trace(Math.exp(1)); // output: 2.71828182845905
trace(Math.exp(2)); // output: 7.38905609893065
```

另请参见

[E \(Math.E 属性\)](#)

## floor (Math.floor 方法)

```
public static floor(x:Number) : Number
```

返回由参数 x 指定的数字或表达式的下限值。下限值是小于等于指定数字或表达式的最接近的整数。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

x:Number - 一个数字或表达式。

### 返回

Number - 最接近且小于等于参数 x 的一个整数。

### 示例

下面的代码返回一个值 12:

```
Math.floor(12.5);
```

下面的代码返回一个值 -7:

```
Math.floor(-6.5);
```

## LN10 (Math.LN10 属性)

```
public static LN10 : Number
```

代表 10 的自然对数的数学常数，表示为  $\log_e 10$ ，其近似值为 2.302585092994046。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

此示例跟踪 Math.LN10 的值。

```
trace(Math.LN10);  
// output: 2.30258509299405
```

## LN2 (Math.LN2 属性)

```
public static LN2 : Number
```

代表 2 的自然对数的数学常数，表示为  $\log_e 2$ ，其近似值为 0.6931471805599453。

可用性: ActionScript 1.0 ; Flash Player 5

## log (Math.log 方法)

```
public static log(x:Number) : Number
```

返回参数 x 的自然对数。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

x:Number - 其值大于 0 的数字或表达式。

### 返回

Number - 参数 x 的自然对数。

### 示例

下面的示例显示三个数值的对数。

```
trace(Math.log(0)); // output: -Infinity
trace(Math.log(1)); // output: 0
trace(Math.log(2)); // output: 0.693147180559945
trace(Math.log(Math.E)); // output: 1
```

## LOG10E (Math.LOG10E 属性)

```
public static LOG10E : Number
```

代表常数  $e$  (Math.E) 以 10 为底的对数的数学常数，表示为  $\log_{10} e$ ，其近似值为 0.4342944819032518。

Math.log() 方法计算数字的自然对数。将 Math.log() 的结果与 Math.LOG10E 相乘得到以 10 为底的对数。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

此示例演示如何得到数字的以 10 为底的对数：

```
trace(Math.log(1000) * Math.LOG10E);
// Output: 3
```

## LOG2E (Math.LOG2E 属性)

`public static LOG2E : Number`

代表常数  $e$  (`Math.E`) 以 2 为底的对数的数学常数，表示为  $\log_2 e$ ，其近似值为 1.442695040888963387。

`Math.log` 方法计算数字的自然对数。将 `Math.log()` 的结果与 `Math.LOG2E` 相乘得到以 2 为底的对数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

此示例演示如何得到数字的以 2 为底的对数：

```
trace(Math.log(16) * Math.LOG2E);  
// Output: 4
```

## max (Math.max 方法)

`public static max(x:Number, y:Number) : Number`

计算  $x$  和  $y$ ，并返回两者中的较大值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**x:**`Number` - 一个数字或表达式。

**y:**`Number` - 一个数字或表达式。

### 返回

`Number` - 一个数字。

### 示例

下面的示例将返回 Thu Dec 30 00:00:00 GMT-0700 2004，它是所计算的表达式的较大值。

```
var date1:Date = new Date(2004, 11, 25);  
var date2:Date = new Date(2004, 11, 30);  
var maxDate:Number = Math.max(date1.getTime(), date2.getTime());  
trace(new Date(maxDate).toString());
```

### 另请参见

[min \(Math.min 方法\)](#)，[Date](#)

## min (Math.min 方法)

`public static min(x:Number, y:Number) : Number`

计算 *x* 和 *y*，并返回两者中的较小值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

*x*:Number - 一个数字或表达式。

*y*:Number - 一个数字或表达式。

### 返回

Number - 一个数字。

### 示例

下面的示例将返回 Sat Dec 25 00:00:00 GMT-0700 2004, 它是所计算的表达式的较小值。

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var minDate:Number = Math.min(date1.getTime(), date2.getTime());
trace(new Date(minDate).toString());
```

### 另请参见

[max \(Math.max 方法\)](#)

## PI (Math.PI 属性)

`public static PI : Number`

代表一个圆的周长与其直径的比值的数学常数，表示为 **pi**，其近似值为 **3.141592653589793**。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例使用数学常数 **pi** 和 **Drawing API** 绘制一个圆。

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
        Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
```

```

mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
Math.sin(Math.PI/4)*r+y);
mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}

```

## pow (Math.pow 方法)

public static pow(x:Number, y:Number) : Number

计算并返回 x 的 y 次幂。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

**x:**Number - 将自乘的数字。

**y:**Number - 指定参数 x 自乘幂的数字。

### 返回

Number - 一个数字。

### 示例

下面的示例将使用 Math.pow 和 Math.sqrt 计算行的长度。

```

this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.origX = _xmouse;
    this.origY = _ymouse;
};
mouseListener.onMouseUp = function() {
    this.newX = _xmouse;
    this.newY = _ymouse;
    var minY = Math.min(this.origY, this.newY);
    var nextDepth:Number = canvas_mc.getNextHighestDepth();
    var line_mc:MovieClip =
    canvas_mc.createEmptyMovieClip("line"+nextDepth+"_mc", nextDepth);
    line_mc.moveTo(this.origX, this.origY);
    line_mc.lineStyle(2, 0x000000, 100);
    line_mc.lineTo(this.newX, this.newY);
    var hypLen:Number = Math.sqrt(Math.pow(line_mc._width,
    2)+Math.pow(line_mc._height, 2));

```

```

line_mc.createTextField("length"+nextDepth+"_txt",
canvas_mc.getNextHighestDepth(), this.origX, this.origY-22, 100, 22);
line_mc['length'+nextDepth+'_txt'].text = Math.round(hypLen) + " pixels";
};
Mouse.addListener(mouseListener);

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## random（Math.random 方法）

```
public static random() : Number
```

返回一个伪随机数 **n**，其中  $0 \leq n < 1$ 。返回的数字之所以是一个伪随机数是因为它不是由真正的随机自然现象（如放射性衰变）生成的数字。

可用性：ActionScript 1.0；Flash Player 5

### 返回

Number - 一个数字。

### 示例

下面的示例输出 4 和 11（包含）之间的 100 个随机整数：

```

function randRange(min:Number, max:Number):Number {
    var randomNum:Number = Math.floor(Math.random() * (max - min + 1)) + min;
    return randomNum;
}
for (var i = 0; i < 100; i++) {
    var n:Number = randRange(4, 11)
    trace(n);
}

```

## round（Math.round 方法）

```
public static round(x:Number) : Number
```

将参数 **x** 的值向上或向下舍入为最接近的整数并返回该值。如果参数 **x** 与两个最接近的两个整数等距离（即该数字以 .5 结尾），则该值向上舍入为下一个较高的整数。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**x**:Number - 一个数字。

## 返回

Number - 一个数字；一个整数。

## 示例

下面的示例返回两个指定整数之间的随机数。

```
function randRange(min:Number, max:Number):Number {
    var randomNum:Number = Math.round(Math.random() * (max-min+1) + (min-.5));
    return randomNum;
}
for (var i = 0; i<25; i++) {
    trace(randRange(4, 11));
}
```

## 另请参见

[ceil \(Math.ceil 方法\)](#), [floor \(Math.floor 方法\)](#)

# sin (Math.sin 方法)

```
public static sin(x:Number) : Number
```

以弧度为单位计算并返回指定角度的正弦值。若要计算弧度，请参见 **Math** 类条目的说明。

可用性：ActionScript 1.0；Flash Player 5

## 参数

x:Number - 一个数字，它表示一个以弧度为单位的角度。

## 返回

Number - 一个数字；指定角度的正弦值（介于 -1.0 和 1.0 之间）。

## 示例

下面的示例使用数学常数 **pi**、角度的正弦值和 **Drawing API** 绘制一个圆。

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
        Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
        Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
}
```



```

mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
Math.sin(Math.PI/4)*r+y);
mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}

```

另请参见

[acos \(Math.acos 方法\)](#), [asin \(Math.asin 方法\)](#), [atan \(Math.atan 方法\)](#), [atan2 \(Math.atan2 方法\)](#), [cos \(Math.cos 方法\)](#), [tan \(Math.tan 方法\)](#)

## sqrt (Math.sqrt 方法)

```
public static sqrt(x:Number) : Number
```

计算并返回指定数字的平方根。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**x:**Number - 一个大于或等于 0 的数字或表达式。

### 返回

Number - 如果参数 **x** 大于或等于零, 则返回一个数字; 否则返回 **NaN** (非数字)。

### 示例

下面的示例将使用 `Math.pow` 和 `Math.sqrt` 计算行的长度。

```

this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.origX = _xmouse;
    this.origY = _ymouse;
};
mouseListener.onMouseUp = function() {
    this.newX = _xmouse;
    this.newY = _ymouse;
    var minY = Math.min(this.origY, this.newY);
    var nextDepth:Number = canvas_mc.getNextHighestDepth();
    var line_mc:MovieClip =
    canvas_mc.createEmptyMovieClip("line"+nextDepth+"_mc", nextDepth);
    line_mc.moveTo(this.origX, this.origY);
    line_mc.lineStyle(2, 0x000000, 100);
    line_mc.lineTo(this.newX, this.newY);
    var hypLen:Number = Math.sqrt(Math.pow(line_mc._width,
    2)+Math.pow(line_mc._height, 2));

```

```
line_mc.createTextField("length"+nextDepth+"_txt",
    canvas_mc.getNextHighestDepth(), this.origX, this.origY-22, 100, 22);
line_mc['length'+nextDepth+'_txt'].text = Math.round(hypLen) + " pixels";
};
Mouse.addListener(mouseListener);
```

## SQRT1\_2 (Math.SQRT1\_2 属性)

`public static SQRT1_2 : Number`

代表  $1/2$  的平方根的数学常数，其近似值为 0.7071067811865476。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

此示例跟踪 `Math.SQRT1_2` 的值。

```
trace(Math.SQRT1_2);
// Output: 0.707106781186548
```

## SQRT2 (Math.SQRT2 属性)

`public static SQRT2 : Number`

代表 2 的平方根的数学常数，其近似值为 1.4142135623730951。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

此示例跟踪 `Math.SQRT2` 的值。

```
trace(Math.SQRT2);
// Output: 1.4142135623731
```

## tan (Math.tan 方法)

`public static tan(x:Number) : Number`

计算并返回指定角度的正切值。若要计算弧度，请参见 **Math** 类简介中所概述的信息。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**x:**`Number` - 一个数字，它表示一个以弧度为单位的角度。

### 返回

`Number` - 一个数字；参数 **x** 的正切值。

## 示例

下面的示例使用数学常数 **pi**、角度的正切值和 **Drawing API** 绘制一个圆。

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.lineStyle(2, 0xFF0000, 100);
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

## 另请参见

[acos \(Math.acos 方法\)](#), [asin \(Math.asin 方法\)](#), [atan \(Math.atan 方法\)](#), [atan2 \(Math.atan2 方法\)](#), [cos \(Math.cos 方法\)](#), [sin \(Math.sin 方法\)](#)

# Matrix (flash.geom.Matrix)

```
Object
|
+- flash.geom.Matrix
```

```
public class Matrix
extends Object
```

**flash.geom.Matrix** 类表示一个转换矩阵，它确定如何将一个坐标空间的点映射到另一个坐标空间。通过设置 **Matrix** 对象的属性并将其应用于 **MovieClip** 对象或 **BitmapData** 对象，您可以对该对象执行各种图形转换。这些转换函数包括平移（**x** 和 **y** 重新定位）、旋转、缩放和倾斜。

这些转换类型统称为仿射转换。仿射转换在转换时保持线条笔直，并且平行线保持平行。

要将转换矩阵应用于影片剪辑，您可以创建一个 **flash.geom.Transform** 对象，并将其 **Matrix** 属性设置为转换矩阵。**Matrix** 对象也用作某些方法（例如 **flash.display.BitmapData** 类的 **draw()** 方法）的参数。

转换 `matrix` 对象被视为具有如下内容的 3 x 3 的矩阵：

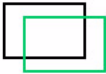

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ u & v & w \end{bmatrix}$$



在传统的转换矩阵中，`u`、`v` 和 `w` 属性具有其它功能。`Matrix` 类只能在二维空间中操作，因此始终假定属性值 `u` 和 `v` 为 `0.0`，属性值 `w` 为 `1.0`。换句话说，矩阵的有效值如下：

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

您可以获取和设置 `Matrix` 对象的全部六个其它属性的值：`a`、`b`、`c`、`d`、`tx` 和 `ty`。

`Matrix` 类支持四种主要的转换函数类型：平移、缩放、旋转和倾斜。对于这些函数中的三种，有特定的方法，如下表中所述。

转换	方法	矩阵值	显示结果	说明
平移（置换）	<code>translate(tx, ty)</code>	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$		将图像 <code>tx</code> 像素向右移动，将 <code>ty</code> 像素向下移动。
缩放	<code>scale(sx, sy)</code>	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$		调整图像的大小，方法是将每个像素的位置在 <code>x</code> 轴方向上乘以 <code>sx</code> 并在 <code>y</code> 轴方向上乘以 <code>sy</code> 。

转换	方法	矩阵值	显示结果	说明
旋转	rotate(q)	$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$		将图像旋转一个以弧度为单位的角度 q。
倾斜或剪切	无；必须设置属性 b 和 c。	$\begin{bmatrix} 0 & \text{sky} & 0 \\ \text{sk}_x & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		以平行于 X 轴或 Y 轴的方向逐渐滑动图像。skx 值充当乘数，控制沿 x 轴滑动的距离；sky 控制沿 y 轴滑动的距离。

每个转换函数都将更改当前矩阵的属性，所以您可以有效地合并多个转换。为此，请在将矩阵应用于影片剪辑或位图目标之前调用多个转换函数。

可用性：ActionScript 1.0 ； Flash Player 8

另请参见

[transform](#) (MovieClip.transform 属性), [Transform](#) (flash.geom.Transform), [draw](#) (BitmapData.draw 方法), [a](#) (Matrix.a 属性), [b](#) (Matrix.b 属性), [c](#) (Matrix.c 属性), [d](#) (Matrix.d 属性), [tx](#) (Matrix.tx 属性), [ty](#) (Matrix.ty 属性), [translate](#) (Matrix.translate 方法), [scale](#) (Matrix.scale 方法), [rotate](#) (Matrix.rotate 方法)

属性摘要

修饰符	属性	说明
	a:Number	Matrix 对象的第一行和第一列中的值，它影响在缩放或旋转图像时沿 X 轴的像素定位。
	b:Number	Matrix 对象的第一行和第二列中的值，它影响在旋转或倾斜图像时沿 Y 轴的像素定位。
	c:Number	Matrix 对象的第二行和第一列中的值，它影响在旋转或倾斜图像时沿 X 轴的像素定位。
	d:Number	Matrix 对象的第二行和第二列中的值，它影响在缩放或旋转图像时沿 Y 轴的像素定位。
	tx:Number	沿 X 轴平移每个点的距离。
	ty:Number	沿 Y 轴平移每个点的距离。

继承自 Object 类的属性

<code>constructor</code> ( <code>Object.constructor</code> 属性), <code>__proto__</code> ( <code>Object.__proto__</code> 属性), <code>prototype</code> ( <code>Object.prototype</code> 属性), <code>__resolve</code> ( <code>Object.__resolve</code> 属性)
---

构造函数摘要

签名	说明
<code>Matrix([a:Number], [b:Number], [c:Number], [d:Number], [tx:Number], [ty:Number])</code>	使用指定参数创建新的 <b>Matrix</b> 对象。

方法摘要

修饰符	签名	说明
	<code>clone() : Matrix</code>	返回一个新的 <b>Matrix</b> 对象，它是此矩阵的克隆，带有与所含对象完全相同的副本。
	<code>concat(m:Matrix) : Void</code>	将某个矩阵与当前矩阵连接，从而将这两个矩阵的几何效果有效地结合在一起。
	<code>createBox(scaleX:Number, scaleY:Number, [rotation:Number], [tx:Number], [ty:Number]) : Void</code>	包括用于缩放、旋转和转换的参数。
	<code>createGradientBox(width:Number, height:Number, [rotation:Number], [tx:Number], [ty:Number]) : Void</code>	创建 <code>MovieClip.beginGradientFill()</code> 方法所需的矩阵的特定样式。
	<code>deltaTransformPoint(pt:Point) : Point</code>	如果给定预转换坐标空间中的点，则此方法返回发生转换后该点的坐标。
	<code>identity() : Void</code>	将每个矩阵属性设置为可使变形后的影片剪辑或几何构造与原件完全相同的值。
	<code>invert() : Void</code>	执行原始矩阵的逆转换。
	<code>rotate(angle:Number) : Void</code>	在当前矩阵中设置值，以便可以使用该矩阵来应用旋转转换。
	<code>scale(sx:Number, sy:Number) : Void</code>	修改矩阵，以便其效果在应用时为调整图像的大小。
	<code>toString() : String</code>	返回列出该 <b>Matrix</b> 对象属性的文本值。

修饰符	签名	说明
	<code>transformPoint(pt:Point) : Point</code>	将该 <b>Matrix</b> 对象表示的几何转换应用于指定点。
	<code>translate(tx:Number, ty:Number) : Void</code>	修改 <b>Matrix</b> 对象，以便其转换效果是沿 <b>x</b> 轴和 <b>y</b> 轴移动对象。

继承自 **Object** 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPropertyEnumerable](#) ([Object.isPropertyEnumerable](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

## a (Matrix.a 属性)

`public a : Number`

**Matrix** 对象的第一行和第一列中的值，它影响在缩放或旋转图像时沿 **X** 轴的像素定位。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将创建 **Matrix** 对象 `myMatrix` 并设置它的 `a` 值。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.a); // 1

myMatrix.a = 2;
trace(myMatrix.a); // 2
```

## b (Matrix.b 属性)

```
public b : Number
```

**Matrix** 对象的第一行和第二列中的值，它影响在旋转或倾斜图像时沿 Y 轴的像素定位。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将创建 **Matrix** 对象 myMatrix 并设置它的 b 值。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.b); // 0

var degrees:Number = 45;
var radians:Number = (degrees/180) Math.PI;
myMatrix.b = radians;
trace(myMatrix.b); // 0.785398163397448
```

## c (Matrix.c 属性)

```
public c : Number
```

**Matrix** 对象的第二行和第一列中的值，它影响在旋转或倾斜图像时沿 X 轴的像素定位。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将创建 **Matrix** 对象 myMatrix 并设置它的 c 值。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.c); // 0

var degrees:Number = 45;
var radians:Number = (degrees/180) Math.PI;
myMatrix.c = radians;
trace(myMatrix.c); // 0.785398163397448
```



## clone (Matrix.clone 方法)

```
public clone() : Matrix
```

返回一个新的 **Matrix** 对象，它是此矩阵的克隆，带有与所含对象完全相同的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.geom.Matrix - 一个 **Matrix** 对象。

### 示例

下面的示例将从 myMatrix 变量创建 clonedMatrix 变量。**Matrix** 类没有 equals 方法，因此下面的示例将使用自定义编写的函数测试两个矩阵是否相等。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 0, 0);
var clonedMatrix:Matrix = new Matrix();

trace(myMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(clonedMatrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
trace(equals(myMatrix, clonedMatrix)); // false

clonedMatrix = myMatrix.clone();

trace(myMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(clonedMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(equals(myMatrix, clonedMatrix)); // true

function equals(m1:Matrix, m2:Matrix):Boolean {
    return m1.toString() == m2.toString();
}
```

## concat (Matrix.concat 方法)

```
public concat(m:Matrix) : Void
```

将某个矩阵与当前矩阵连接，从而将这两个矩阵的几何效果有效地结合在一起。在数学术语中，将两个矩阵连接起来与使用矩阵乘法将它们结合起来是相同的。

例如，如果矩阵 m1 使用系数 4 缩放某个对象，而矩阵 m2 使用 1.5707963267949 弧度 (Math.PI/2) 旋转该对象，则 m1.concat(m2) 会将 m1 转换为一个使用系数 4 缩放对象并使用 Math.PI/2 弧度旋转该对象的矩阵。

此方法将源矩阵替换为连接矩阵。如果要在不更改两个源矩阵中的任何一个的情况下连接这两个矩阵，则可以首先复制源矩阵的 clone() 方法，如“示例”部分中所示。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**m:** flash.geom.Matrix - 要连接到源矩阵的矩阵。

## 示例

下面的示例将创建三个矩阵，分别定义三个矩形影片剪辑的转换。前两个矩阵 rotate45Matrix 和 doubleScaleMatrix 将分别应用于两个矩形 rectangleMc\_1 和 rectangleMc\_2。然后对 rotate45Matrix 和 doubleScaleMatrix 使用 concat() 方法创建 scaleAndRotateMatrix，以创建第三个矩阵。然后将此矩阵应用于 rectangleMc\_3，以缩放并旋转该矩形。

```
import flash.geom.Matrix;
import flash.geom.Transform;

var rectangleMc_0:MovieClip = createRectangle(20, 80, 0x000000);
var rectangleMc_1:MovieClip = createRectangle(20, 80, 0xFF0000);
var rectangleMc_2:MovieClip = createRectangle(20, 80, 0x00FF00);
var rectangleMc_3:MovieClip = createRectangle(20, 80, 0x0000FF);

var rectangleTrans_1:Transform = new Transform(rectangleMc_1);
var rectangleTrans_2:Transform = new Transform(rectangleMc_2);
var rectangleTrans_3:Transform = new Transform(rectangleMc_3);

var rotate45Matrix:Matrix = new Matrix();
rotate45Matrix.rotate(Math.PI/4);
rectangleTrans_1.matrix = rotate45Matrix;
rectangleMc_1._x = 100;
trace(rotate45Matrix.toString()); // (a=0.707106781186548,
    b=0.707106781186547, c=-0.707106781186547, d=0.707106781186548, tx=0,
    ty=0)

var doubleScaleMatrix:Matrix = new Matrix();
doubleScaleMatrix.scale(2, 2);
rectangleTrans_2.matrix = doubleScaleMatrix;
rectangleMc_2._x = 200;
trace(doubleScaleMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

var scaleAndRotateMatrix:Matrix = doubleScaleMatrix.clone();
scaleAndRotateMatrix.concat(rotate45Matrix);
rectangleTrans_3.matrix = scaleAndRotateMatrix;
rectangleMc_3._x = 300;
trace(scaleAndRotateMatrix.toString()); // (a=1.4142135623731,
    b=1.41421356237309, c=-1.41421356237309, d=1.4142135623731, tx=0, ty=0)

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
```

```

mc.lineTo(0, height);
mc.lineTo(width, height);
mc.lineTo(width, 0);
mc.lineTo(0, 0);
return mc;
}

```

## createBox（Matrix.createBox 方法）

```

public createBox(scaleX:Number, scaleY:Number, [rotation:Number],
    [tx:Number], [ty:Number]) : Void

```

包括用于缩放、旋转和转换的参数。当应用于矩阵时，该方法会基于这些参数设置矩阵的值。

通过使用 createBox() 方法，您可以获得与依次应用 identity()、rotate()、scale() 和 translate() 方法时得到的矩阵相同的矩阵。例如，mat1.createBox(2,2,Math.PI/5, 100, 100) 具有与如下所示代码相同的效果：

```

import flash.geom.Matrix;

var mat1:Matrix = new Matrix();
mat1.identity();
mat1.rotate(Math.PI/4);
mat1.scale(2,2);
mat1.translate(10,20);

```

可用性：ActionScript 1.0；Flash Player 8

### 参数

**scaleX:**Number - 水平缩放所用的系数。

**scaleY:**Number - 垂直缩放所用的系数。

**rotation:**Number [ 可选 ] - 旋转量（以弧度为单位）。默认值是 0。

**tx:**Number [ 可选 ] - 沿 x 轴向右平移（移动）的像素数。默认值是 0。

**ty:**Number [ 可选 ] - 沿 y 轴向下平移（移动）的像素数。默认值是 0。

### 示例

下面的示例通过调用 myMatrix 的 createBox() 方法设置该矩阵的 scaleX 缩放、scaleY 缩放、旋转、x 位置和 y 位置。

```

import flash.geom.Matrix;
import flash.geom.Transform;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.createBox(1, 2, Math.PI/4, 100, 200);

```

```
trace(myMatrix.toString()); // (a=0.707106781186548, b=1.41421356237309, c=-
    0.707106781186547, d=1.4142135623731, tx=100, ty=200)

var rectangleMc:MovieClip = createRectangle(20, 80, 0xFF0000);
var rectangleTrans:Transform = new Transform(rectangleMc);
rectangleTrans.matrix = myMatrix;
```

另请参见

## createGradientBox (Matrix.createGradientBox 方法)

```
public createGradientBox(width:Number, height:Number, [rotation:Number],
    [tx:Number], [ty:Number]) : Void
```

创建 MovieClip.beginGradientFill() 方法所需的矩阵的特定样式。宽度和高度被缩放为 scaleX/scaleY 对，而 tx/ty 值偏移了宽度和高度的一半。

可用性: ActionScript 1.0 ; Flash Player 8

### 参数

**width:**Number - 渐变框的宽度。

**height:**Number - 渐变框的高度。

**rotation:**Number [ 可选 ] - 旋转量（以弧度为单位）。默认值是 0。

**tx:**Number [ 可选 ] - 沿 x 轴向右平移的距离（以像素为单位）。此值将偏移宽度参数的一半。默认值是 0。

**ty:**Number [ 可选 ] - 沿 y 轴向下平移的距离（以像素为单位）此值将偏移高度参数的一半。默认值是 0。

### 示例

下面的示例使用 myMatrix 作为 MovieClip 对象的 beginGradientFill() 方法的参数。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.createGradientBox(200, 200, 0, 50, 50);
trace(myMatrix.toString()); // (a=0.1220703125, b=0, c=0, d=0.1220703125,
    tx=150, ty=150)

var depth:Number = this.getNextHighestDepth();
var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
var colors:Array = [0xFF0000, 0x0000FF];
var alphas:Array = [100, 100];
```

```
var ratios:Array = [0, 0xFF];
mc.beginGradientFill("linear", colors, alphas, ratios, myMatrix);
mc.lineTo(0, 300);
mc.lineTo(300, 300);
mc.lineTo(300, 0);
mc.lineTo(0, 0);
```

另请参见

[beginGradientFill](#) ([MovieClip.beginGradientFill](#) 方法)

## d (Matrix.d 属性)

```
public d : Number
```

**Matrix** 对象的第二行和第二列中的值，它影响在缩放或旋转图像时沿 Y 轴的像素定位。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将创建 **Matrix** 对象 `myMatrix` 并设置它的 `d` 值。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.d); // 1

myMatrix.d = 2;
trace(myMatrix.d); // 2
```

## deltaTransformPoint (Matrix.deltaTransformPoint 方法)

```
public deltaTransformPoint(pt:Point) : Point
```

如果给定预转换坐标空间中的点，则此方法返回发生转换后该点的坐标。与使用 **transformPoint()** 方法应用的标准转换不同，**deltaTransformPoint()** 方法的转换不考虑转换参数 `tx` 和 `ty`。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**pt:** `flash.geom.Point` - 一个点对象。

### 返回

`flash.geom.Point` - 新的 **Point** 对象。

## 示例

下面的示例使用 `deltaTransformPoint()` 方法从 `myPoint` 创建 `deltaTransformedPoint`。在该示例中, `translate()` 方法不会更改名为 `deltaTransformedPoint` 的点的位置。但是, `scale()` 方法会影响该点的位置。它使用系数 **3** 将该点的 `x` 值从 **50** 增加到 **150**。

```
import flash.geom.Matrix;
import flash.geom.Point;

var myMatrix:Matrix = new Matrix();
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.translate(100, 0);
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=100, ty=0)

myMatrix.scale(3, 3);
trace(myMatrix); // (a=3, b=0, c=0, d=3, tx=300, ty=0)

var myPoint:Point = new Point(50,0);
trace(myPoint); // (50, 0)

var deltaTransformedPoint:Point = myMatrix.deltaTransformPoint(myPoint);
trace(deltaTransformedPoint); // (150, 0)

var pointMc_0:MovieClip = createRectangle(10, 10, 0xFF0000);
pointMc_0._x = myPoint.x;

var pointMc_1:MovieClip = createRectangle(10, 10, 0x00FF00);
pointMc_1._x = deltaTransformedPoint.x;

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## identity (Matrix.identity 方法)

`public identity() : Void`

将每个矩阵属性设置为可使变形后的影片剪辑或几何构造与原件完全相同的值。

调用 `identity()` 方法后，生成的矩阵具有下列属性: `a=1`, `b=0`, `c=0`, `d=1`, `tx=0`, `ty=0`。

在矩阵记号中，恒等矩阵与如下所示类似：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

可用性: `ActionScript 1.0` ; `Flash Player 8`

### 示例

下面的示例演示调用 `identity()` 方法会将调用的 `Matrix` 对象转换为恒等 `Matrix` 对象。以前应用于原始 `Matrix` 对象的转换数量和类型与新矩阵无关。如果调用 `identity()`，则该矩阵值将转换为 (`a=1`、`b=0`、`c=0`、`d=1`、`tx=0` 和 `ty=0`)。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 0, 0);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

myMatrix.rotate(Math.atan(3/4));
trace(myMatrix.toString()); // (a=1.6, b=1.2, c=-1.2, d=1.6, tx=0, ty=0)

myMatrix.translate(100,200);
trace(myMatrix.toString()); // (a=1.6, b=1.2, c=-1.2, d=1.6, tx=100, ty=200)

myMatrix.scale(2, 2);
trace(myMatrix.toString()); // (a=3.2, b=2.4, c=-2.4, d=3.2, tx=200, ty=400)

myMatrix.identity();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
```

## invert (Matrix.invert 方法)

```
public invert() : Void
```

执行原始矩阵的逆转换。您可以将一个逆矩阵应用于对象来撤销在应用原始矩阵时执行的转换。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例通过调用 `doubleScaleMatrix` 的 `invert()` 方法创建 `halfScaleMatrix`，然后显示这两个矩阵互为逆矩阵，即通过撤销另一个矩阵的所有转换操作可以得到的矩阵。该示例通过创建等于 `noScaleMatrix` 的 `originalAndInverseMatrix` 说明此反转。

```
import flash.geom.Matrix;
import flash.geom.Transform;

var rectangleMc_0:MovieClip = createRectangle(20, 80, 0xFF0000);
var rectangleMc_1:MovieClip = createRectangle(20, 80, 0x00FF00);
var rectangleMc_2:MovieClip = createRectangle(20, 80, 0x0000FF);
var rectangleMc_3:MovieClip = createRectangle(20, 80, 0x000000);

var rectangleTrans_0:Transform = new Transform(rectangleMc_0);
var rectangleTrans_1:Transform = new Transform(rectangleMc_1);
var rectangleTrans_2:Transform = new Transform(rectangleMc_2);
var rectangleTrans_3:Transform = new Transform(rectangleMc_3);

var doubleScaleMatrix:Matrix = new Matrix(2, 0, 0, 2, 0, 0);
rectangleTrans_0.matrix = doubleScaleMatrix;
trace(doubleScaleMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

var noScaleMatrix:Matrix = new Matrix(1, 0, 0, 1, 0, 0);
rectangleTrans_1.matrix = noScaleMatrix;
rectangleMc_1._x = 100;
trace(noScaleMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var halfScaleMatrix:Matrix = doubleScaleMatrix.clone();
halfScaleMatrix.invert();
rectangleTrans_2.matrix = halfScaleMatrix;
rectangleMc_2._x = 200;
trace(halfScaleMatrix.toString()); // (a=0.5, b=0, c=0, d=0.5, tx=0, ty=0)

var originalAndInverseMatrix:Matrix = doubleScaleMatrix.clone();
originalAndInverseMatrix.concat(halfScaleMatrix);
rectangleTrans_3.matrix = originalAndInverseMatrix;
rectangleMc_3._x = 300;
trace(originalAndInverseMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
```



```
function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## Matrix 构造函数

```
public Matrix([a:Number], [b:Number], [c:Number], [d:Number], [tx:Number],
    [ty:Number])
```

使用指定参数创建新的 **Matrix** 对象。在矩阵记号中，按如下方式组织属性：

$$\begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{t_x} \\ \mathbf{c} & \mathbf{d} & \mathbf{t_y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

如果不向新 `Matrix()` 构造函数提供任何参数，它将使用下面的值创建一个“恒等矩阵”：

a = 1	b = 0
c = 0	d = 1
tx = 0	ty = 0

在矩阵记号中，恒等矩阵与如下所示类似：

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

可用性：ActionScript 1.0；Flash Player 8

## 参数

**a**:Number [ 可选 ] - 新 **Matrix** 对象的第一行第一列的值。

**b**:Number [ 可选 ] - 新 **Matrix** 对象的第一行第二列的值。

**c**:Number [ 可选 ] - 新 **Matrix** 对象的第二行第一列的值。

**d**:Number [ 可选 ] - 新 **Matrix** 对象的第二行第二列的值。

**tx**:Number [ 可选 ] - 新 **Matrix** 对象的第三行第一列的值。

**ty**:Number [ 可选 ] - 新 **Matrix** 对象的第三行第二列的值。

## 示例

下面的示例通过不向 **Matrix** 构造函数发送参数创建 **matrix\_1**，而通过向该构造函数发送参数创建 **matrix\_2**。创建的不含任何参数的 **Matrix** 对象 **matrix\_1** 是具有值 (**a=1, b=0, c=0, d=1, tx=0, ty=0**) 的恒等矩阵。

```
import flash.geom.Matrix;

var matrix_1:Matrix = new Matrix();
trace(matrix_1); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var matrix_2:Matrix = new Matrix(1, 2, 3, 4, 5, 6);
trace(matrix_2); // (a=1, b=2, c=3, d=4, tx=5, ty=6)
```

## rotate (Matrix.rotate 方法)

```
public rotate(angle:Number) : Void
```

在当前矩阵中设置值，以便可以使用该矩阵来应用旋转变换。

**rotate()** 方法将更改 **Matrix** 对象的 **a** 和 **d** 属性。这在矩阵记号中显示为：

$$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

可用性：ActionScript 1.0 ； Flash Player 8

## 参数

**angle**:Number - 以弧度为单位的旋转角度。

## 示例

下面的示例说明 `rotate()` 方法如何顺时针旋转 `rectangleMc` 30°。将 `myMatrix` 应用于 `rectangleMc` 会重置它的 `_x` 值，所以由您手动将其重置为 100。

```
import flash.geom.Matrix;
import flash.geom.Transform;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var degrees:Number = 30;
var radians:Number = (degrees/180) Math.PI;
myMatrix.rotate(radians);
trace(myMatrix.toString()); // (a=0.866025403784439, b=0.5, c=-0.5,
    d=0.866025403784439, tx=0, ty=0)

var rectangleMc:MovieClip = createRectangle(20, 80, 0xFF0000);
trace(rectangleMc._x); // 0
rectangleMc._x = 100;
trace(rectangleMc._x); // 100

var rectangleTrans:Transform = new Transform(rectangleMc);
rectangleTrans.matrix = myMatrix;
trace(rectangleMc._x); // 0
rectangleMc._x = 100;
trace(rectangleMc._x); // 100

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

上面的示例使用 **MovieClip** 对象的 `_x` 属性定位 `rectangleMc`。通常，在处理 **Matrix** 对象定位时，混合定位技术被认为是不可取的方式。上面的示例是使用正确语法编写的，它将平移矩阵连接到 `myMatrix`，以更改 `rectangleMc` 的水平位置。下面的示例对此进行了演示。

```
import flash.geom.Matrix;
import flash.geom.Transform;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
```

```

var degrees:Number = 30;
var radians:Number = (degrees/180) Math.PI;
myMatrix.rotate(radians);
trace(myMatrix.toString()); // (a=0.866025403784439, b=0.5, c=-0.5,
    d=0.866025403784439, tx=0, ty=0)

var translateMatrix:Matrix = new Matrix();
translateMatrix.translate(100, 0);
myMatrix.concat(translateMatrix);
trace(myMatrix.toString()); // (a=0.866025403784439, b=0.5, c=-0.5,
    d=0.866025403784439, tx=100, ty=0)

var rectangleMc:MovieClip = createRectangle(20, 80, 0xFF0000);
trace(rectangleMc._x); // 0
rectangleMc._x = 100;
trace(rectangleMc._x); // 100

var rectangleTrans:Transform = new Transform(rectangleMc);
rectangleTrans.matrix = myMatrix;
trace(rectangleMc._x); // 100

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}

```

## scale（Matrix.scale 方法）

```
public scale(sx:Number, sy:Number) : Void
```

修改矩阵，以便其效果在应用时为调整图像的大小。在经过调整的图像中，每个像素的位置在 **x** 轴方向上乘以 **sx**；在 **y** 轴方向上乘以 **sy**。

**scale()** 方法将更改 **matrix** 对象的 **a** 和 **d** 属性。这在矩阵记号中显示为：

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

可用性：ActionScript 1.0；Flash Player 8

## 参数

**sx**:Number - 用于沿 **x** 轴缩放对象的乘数。

**sy**:Number - 用于沿 **y** 轴缩放对象的乘数。

## 示例

下面的示例使用 `scale()` 方法通过使用系数 **3** 和系数 **4** 分别在水平和垂直方向上缩放 `myMatrix`。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 100, 100);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=100, ty=100)

myMatrix.scale(3, 4);
trace(myMatrix.toString()); // (a=6, b=0, c=0, d=8, tx=300, ty=400)
```

## toString (Matrix.toString 方法)

`public toString() : String`

返回列出该 **Matrix** 对象属性的文本值。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 返回

`String` - 一个字符串, 它包含 **Matrix** 对象的属性值: `a`、`b`、`c`、`d`、`tx` 和 `ty`。

## 示例

下面的示例创建 `myMatrix`, 并将它的值转换为 `(a=A, b=B, c=C, d=D, tx=TX, ty=TY)` 格式的字符串。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace("myMatrix: " + myMatrix.toString()); // (a=1, b=0, c=0, d=1, tx=0,
    ty=0)
```

## transformPoint (Matrix.transformPoint 方法)

public transformPoint(pt:Point) : Point

将该 **Matrix** 对象表示的几何转换应用于指定点。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**pt**:flash.geom.Point - 要进行转换的点 (x,y)。

### 返回

flash.geom.Point - 新的 **Point** 对象。

### 示例

下面的示例使用 `transformPoint()` 方法从 `myPoint` 创建 `transformedPoint`。  
`translate()` 方法在 `transformedPoint` 的位置上起作用。在本示例中, `scale()` 使用系数 **3** 将原始 `x` 值从 **50** 增加到 **150**, `translate()` 将 `x` 值增加 **300**, 因此总值为 **450**。

```
import flash.geom.Matrix;
import flash.geom.Point;

var myMatrix:Matrix = new Matrix();
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

myMatrix.translate(100, 0);
trace(myMatrix); // (a=1, b=0, c=0, d=1, tx=100, ty=0)

myMatrix.scale(3, 3);
trace(myMatrix); // (a=3, b=0, c=0, d=3, tx=300, ty=0)

var myPoint:Point = new Point(50,0);
trace(myPoint); // (50, 0)

var transformedPoint:Point = myMatrix.transformPoint(myPoint);
trace(transformedPoint); // (450, 0)

var pointMc_0:MovieClip = createRectangle(10, 10, 0xFF0000);
pointMc_0._x = myPoint.x;

var pointMc_1:MovieClip = createRectangle(10, 10, 0x00FF00);
pointMc_1._x = transformedPoint.x;

function createRectangle(width:Number, height:Number,
    color:Number):MovieClip {
    var depth:Number = this.getNextHighestDepth();
    var mc:MovieClip = this.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
```

```

        mc.lineTo(width, height);
        mc.lineTo(width, 0);
        mc.lineTo(0, 0);
        return mc;
    }

```

## translate（Matrix.translate 方法）

```
public translate(tx:Number, ty:Number) : Void
```

修改 **Matrix** 对象，以便其转换效果是沿 **x** 轴和 **y** 轴移动对象。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**tx**:Number - 沿 **x** 轴向右移动的量，以像素为单位。

**ty**:Number - 沿 **y** 轴向下移动的量，以像素为单位。

### 示例

下面的示例使用 `translate()` 方法决定 `rectangleMc` 的位置为 **x:100** 且 **y:50**。

`translate()` 方法影响转换属性 `tx` 和 `ty`，但它不影响 `a`、`b`、`c` 或 `d` 属性。

```
import flash.geom.Matrix;
```

```

var myMatrix:Matrix = new Matrix(2, 0, 0, 2, 100, 100);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=100, ty=100)

```

```

myMatrix.translate(100, 50);
trace(myMatrix.toString()); // (a=2, b=0, c=0, d=2, tx=200, ty=150)

```

## tx（Matrix.tx 属性）

```
public tx : Number
```

沿 **X** 轴平移每个点的距离。这表示 **Matrix** 对象的第三行和第一列中的值。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将创建 **Matrix** 对象 `myMatrix` 并设置它的 `tx` 值。

```
import flash.geom.Matrix;
```

```

var myMatrix:Matrix = new Matrix();
trace(myMatrix.tx); // 0

```

```

myMatrix.tx = 50; // 50
trace(myMatrix.tx);

```

## ty (Matrix.ty 属性)

```
public ty : Number
```

沿 Y 轴平移每个点的距离。这表示 **Matrix** 对象的第三行和第二列中的值。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建 **Matrix** 对象 myMatrix 并设置它的 ty 值。

```
import flash.geom.Matrix;

var myMatrix:Matrix = new Matrix();
trace(myMatrix.ty); // 0

myMatrix.ty = 50;
trace(myMatrix.ty); // 50
```

## Microphone

```
Object
|
+-Microphone
```

```
public class Microphone
extends Object
```

**Microphone** 类用于从运行 **Flash Player** 的计算机上所连接的麦克风中捕获音频。

**Microphone** 类主要与 **Flash Communication Server** 一起使用，但也可以用在服务器以外的其它地方，只是在使用上受到限制，例如通过本地系统上的扬声器传送来自麦克风的聲音。

**警告:** **Flash Player** 显示一个“隐私”对话框，让用户选择是允许还是拒绝对麦克风的访问。请确保舞台大小至少为 215 x 138 像素；这是 **Flash** 显示该对话框所需的最小大小。

用户和管理用户还可以基于每个站点或全局禁用麦克风访问。

若要创建或引用 **Microphone** 对象，请使用 `Microphone.get()` 方法。

可用性: **ActionScript 1.0** ; **Flash Player 6**



属性摘要

修饰符	属性	说明
	activityLevel:Number [ 只读 ]	指定麦克风所检测的音量的数字值。
	gain:Number [ 只读 ]	麦克风信号的提升量。
	index:Number [ 只读 ]	一个从零开始的整数，指定麦克风的索引，它反映在 Microphone.names 返回的数组中。
	muted:Boolean [ 只读 ]	一个布尔值，指定用户是已经拒绝对麦克风的访问 (true) 还是已经允许对麦克风的访问 (false)。
	name:String [ 只读 ]	指定当前声音捕获设备的名称的字符串，它由声音捕获硬件返回。
static	names:Array [ 只读 ]	检索一个字符串数组，该数组反映所有可用声音捕获设备名称而不显示 Flash Player 的“隐私设置”面板。
	rate:Number [ 只读 ]	麦克风捕获声音的频率，单位是 kHz。
	silenceLevel:Number [ 只读 ]	一个整数，指定激活麦克风和调用 Microphone.onActivity(true) 所需的音量。
	silenceTimeOut:Number [ 只读 ]	一个数字值，表示麦克风停止检测声音的时间和调用 Microphone.onActivity(false) 的时间之间的毫秒数。
	useEchoSuppression: Boolean	属性（只读）；一个布尔值，如果启用回音抑制，则为 true；否则为 false。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

事件摘要

事件	说明
onActivity = function(active: Boolean) {}	在麦克风开始或停止检测声音时调用。
onStatus = function(infoObj: Object) {}	在用户允许或拒绝对麦克风的访问时调用。

方法摘要

修饰符	签名	说明
static	get([index:Number]) : Microphone	返回对用于捕获音频的 Microphone 对象的引用。
	setGain(gain:Number) : Void	设置麦克风增益，即麦克风在传送信号之前应该将信号放大的倍数。
	setRate(rate:Number) : Void	设置麦克风应该捕获声音的频率，单位是 kHz。
	setSilenceLevel(silenceLevel:Number, [timeOut:Number]) : Void	设置应该被视为有声的最小输入级别以及（可选）指示静音已实际启用的静音时间。
	setUseEchoSuppression(useEchoSuppression:Boolean) : Void	指定是否使用音频编解码器的回声抑制功能。

继承自 Object 类的方法

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPropertyEnumerable](#) ([Object.isPropertyEnumerable](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

activityLevel （Microphone.activityLevel 属性）

public activityLevel : Number [read-only]

指定麦克风所检测的音量的数字值。值的范围从 0（未检测到声音）到 100（检测到非常大的声音）。此属性的值有助于确定向 Microphone.setSilenceLevel() 方法传递的适当值。

如果麦克风可用，但却因为尚未调用 Microphone.get() 而未被使用，则此属性设置为 -1。

可用性：ActionScript 1.0；Flash Player 6

示例

下面的示例在名为 activityLevel\_pb 的 **ProgressBar** 实例中显示当前麦克风的活动级别。

```
var activityLevel_pb:mx.controls.ProgressBar;
activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3%";
activityLevel_pb.setStyle("themeColor", "0xFF0000");
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
```

```

sound_mc.attachAudio(active_mic);
this.onEnterFrame = function() {
    activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
    if (active) {
        var haloTheme_str:String = "haloGreen";
    } else {
        var haloTheme_str:String = "0xFF0000";
    }
    activityLevel_pb.setStyle("themeColor", haloTheme_str);
};

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[get \(Microphone.get 方法\)](#), [setSilenceLevel \(Microphone.setSilenceLevel 方法\)](#), [setGain \(Microphone.setGain 方法\)](#)

## gain (Microphone.gain 属性)

`public gain : Number [read-only]`

麦克风信号的提升量。有效值为 0 到 100。默认值为 50。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例使用名为 `gain_pb` 的 **ProgressBar** 实例显示麦克风的增益值，并使用名为 `gain_nstep` 的 **NumericStepper** 实例设置麦克风的增益值。

```

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

gain_pb.label = "Gain: %3";
gain_pb.mode = "manual";
gain_pb.setProgress(active_mic.gain, 100);
gain_nstep.value = active_mic.gain;

function changeGain() {
    active_mic.setGain(gain_nstep.value);
    gain_pb.setProgress(active_mic.gain, 100);
}
gain_nstep.addEventListener("change", changeGain);

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setGain \(Microphone.setGain 方法\)](#)

## get (Microphone.get 方法)

```
public static get([index:Number]) : Microphone
```

返回用于捕获音频的 **Microphone** 对象的引用。若要实际开始捕获音频，必须将 **Microphone** 对象附加到 **MovieClip** 对象（请参见 `MovieClip.attachAudio()`）。

与使用 `new` 构造函数创建的对象不同，对 `Microphone.get()` 的多个调用会引用同一个麦克风。因此，如果您的脚本包含 `mic1 = Microphone.get()` 和 `mic2 = Microphone.get()` 行，则 `mic1` 和 `mic2` 均将引用同一个（默认）麦克风。

通常情况下，不应该为 `index` 传递值；只要使用 `Microphone.get()` 方法返回对默认麦克风的引用即可。通过“麦克风设置”面板（将在本节的后面部分讨论），用户可以指定 **Flash** 应该使用的默认麦克风。如果要为 `index` 传递值，则可以试图引用非用户首选的其它麦克风。在极少的情况下（例如，您的应用程序同时从两个麦克风捕获音频），您可以使用 `index`。

在一个 **SWF** 文件尝试访问 `Microphone.get()` 方法返回的麦克风时（例如当您发出 `MovieClip.attachAudio()` 时），**Flash Player** 显示“隐私”对话框，用户可从中选择是允许还是拒绝对该麦克风的访问。（请确保舞台大小至少为 215 x 138 像素；这是 **Flash** 显示该对话框所需的最小大小。）

当用户对此对话框做出响应时，`Microphone.onStatus` 事件处理函数将返回指示用户响应的信息对象。若要不在处理此事件处理函数的情况下确定用户是拒绝还是允许对摄像机的访问，请使用 `Microphone.muted`。

用户也可以为特定域指定永久隐私设置，方法是在 **SWF** 文件播放过程中右键单击（在 **Windows** 中）或按住 **Control** 键并单击（在 **Macintosh** 中），选择“设置”，打开“隐私”面板，然后选择“记住”。

您不能使用 **ActionScript** 来设置用户的 **Allow** 或 **Deny** 值，但可以通过使用 `System.showSettings(0)` 来为用户显示“隐私”面板。如果用户选择“记住”，则 **Flash Player** 不再为此域的 **SWF** 文件显示“隐私”对话框。

如果 `Microphone.get()` 返回 `null`，则表明麦克风正由其它应用程序使用，或者在系统上没有安装任何麦克风。若要确定是否已安装了任何麦克风，请使用 `Microphones.names.length`。若要显示 **Flash Player** “麦克风设置”面板（它允许用户选择 `Microphone.get()` 所引用的麦克风），请使用 `System.showSettings(2)`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 参数

**index:**Number [ 可选 ] - 一个从零开始的整数，指定要获取的麦克风，该整数根据 `Microphone.names` 包含的数组确定。若要获取默认的麦克风（建议大多数应用程序采用此设置），请省略此参数。

## 返回

`Microphone` -

- 如果未指定 `index`，则此方法返回对默认麦克风的引用；如果默认麦克风不可用，则返回对第一个可用麦克风的引用。如果没有可用的麦克风或者未安装麦克风，则该方法返回 `null`。
- 如果指定了 `index`，则此方法返回对请求的麦克风的引用；如果请求的麦克风不可用，则返回 `null`。

## 示例

下面的示例让用户指定默认麦克风，然后捕获音频并在本地回放。若要避免回馈，可能需要带着耳机测试此代码。

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
System.showSettings(2);
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[get \(Microphone.get 方法\)](#), [index \(Microphone.index property\)](#), [muted \(Microphone.muted 属性\)](#), [names \(Microphone.names 属性\)](#), [onStatus \(Microphone.onStatus 处理函数\)](#), [attachAudio \(MovieClip.attachAudio 方法\)](#), [showSettings \(System.showSettings 方法\)](#)

## index (Microphone.index property)

public index : Number [read-only]

一个从零开始的整数，指定麦克风的索引，它反映在 `Microphone.names` 返回的数组中。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例在名为 `mic_cb` 的 **ComboBox** 实例中显示计算机系统上可用的声音捕获设备的名称。名为 `mic_lbl` 的 **Label** 组件的实例，显示索引麦克风。可以使用 **ComboBox** 在设备之间切换。

```
var mic_lbl:mx.controls.Label;
var mic_cb:mx.controls.ComboBox;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
mic_lbl.text = "["+active_mic.index+"] "+active_mic.name;
mic_cb.dataProvider = Microphone.names;
mic_cb.selectedIndex = active_mic.index;

var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    active_mic = Microphone.get(evt.target.selectedIndex);
    sound_mc.attachAudio(active_mic);
    mic_lbl.text = "["+active_mic.index+"] "+active_mic.name;
};
mic_cb.addEventListener("change", cbListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[get \(Microphone.get 方法\)](#), [names \(Microphone.names 属性\)](#)

## muted (Microphone.muted 属性)

public muted : Boolean [read-only]

一个布尔值，指定用户是已经拒绝对麦克风的访问 (true) 还是已经允许对麦克风的访问 (false)。当此值出现更改时，将调用 `Microphone.onStatus`。有关更多信息，请参阅 `Microphone.get()`。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

此示例获取默认麦克风，并检查它是否已静音。

```
var active_mic:Microphone = Microphone.get();
trace(active_mic.muted);
```

### 另请参见

[get \(Microphone.get 方法\)](#)，[onStatus \(Microphone.onStatus 处理函数\)](#)

## name (Microphone.name 属性)

public name : String [read-only]

一个字符串，指定由声音捕获硬件返回的当前声音捕获设备的名称。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

以下示例显示有关计算机系统上声音捕获设备的信息，其中包括名称数组和默认设备。

```
var status_ta:mx.controls.TextArea;
status_ta.html = false;
status_ta.setStyle("fontSize", 9);
var microphone_array:Array = Microphone.names;
var active_mic:Microphone = Microphone.get();
status_ta.text = "The default device is: "+active_mic.name+newline+newline;
status_ta.text += "You have "+microphone_array.length+" device(s)
    installed."+newline+newline;
for (var i = 0; i<microphone_array.length; i++) {
    status_ta.text += "["+i+"] "+microphone_array[i]+newline;
}
```

### 另请参见

[get \(Microphone.get 方法\)](#)，[names \(Microphone.names 属性\)](#)

## names (Microphone.names 属性)

`public static names : Array [read-only]`

检索一个字符串数组，该数组反映所有可用声音捕获设备名称而不显示 **Flash Player** 的“隐私设置”面板。此数组的行为与任何其它 **ActionScript** 数组的行为相同，即隐式提供每个声音捕获设备从零开始的索引以及系统上声音捕获设备的数量（通过 `Microphone.names.length`）。有关更多信息，请参见 **Microphone.names** 数组类条目。

调用 `Microphone.names` 要求全面检查硬件，并可能需要几秒钟时间才能生成数组。大多数情况下，使用默认麦克风即可。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

以下示例显示有关计算机系统上声音捕获设备的信息，其中包括名称数组和默认设备。

```
var status_ta:mx.controls.TextArea;
status_ta.html = false;
status_ta.setStyle("fontSize", 9);
var microphone_array:Array = Microphone.names;
var active_mic:Microphone = Microphone.get();
status_ta.text = "The default device is: "+active_mic.name+newline+newline;
status_ta.text += "You have "+microphone_array.length+" device(s)
    installed."+newline+newline;
for (var i = 0; i<microphone_array.length; i++) {
    status_ta.text += "["+i+"] "+microphone_array[i]+newline;
}
```

例如，可能会显示以下信息：

```
The default device is: Logitech USB Headset
You have 2 device(s) installed.
[0] Logitech USB Headset
[1] YAMAHA AC-XG WDM Audio
```

### 另请参见

[name \(Microphone.name 属性\)](#), [get \(Microphone.get 方法\)](#)



## onActivity (Microphone.onActivity 处理函数)

`onActivity = function(active:Boolean) {}`

在麦克风开始或停止检测声音时调用。如果要对此事件处理函数做出响应，则必须创建一个函数来处理其活动值。

若要指定调用 `Microphone.onActivity(true)` 所需的音量以及必须经过多少时间没有声音才调用 `Microphone.onActivity(false)`，请使用 `Microphone.setSilenceLevel()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**active:Boolean** - 布尔值，在麦克风开始检测声音时设置为 **true**，在停止时设置为 **false**。

### 示例

下面的示例在名为 `activityLevel_pb` 的 **ProgressBar** 实例中显示活动级别的量。当麦克风检测到声音时，它将调用 `onActivity` 函数，该函数会修改 **ProgressBar** 实例。

```
var activityLevel_pb:mx.controls.ProgressBar;
activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3%";
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
active_mic.onActivity = function(active:Boolean) {
    if (active) {
        activityLevel_pb.indeterminate = false;
        activityLevel_pb.label = "Activity Level: %3%";
    } else {
        activityLevel_pb.indeterminate = true;
        activityLevel_pb.label = "Activity Level: (inactive)";
    }
};
this.onEnterFrame = function() {
    activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setSilenceLevel \(Microphone.setSilenceLevel 方法\)](#)

## onStatus (Microphone.onStatus 处理函数)

```
onStatus = function(infoObject:Object) {}
```

在用户允许或拒绝对麦克风的访问时调用。如果要响应此事件处理函数，必须创建一个函数来处理麦克风生成的信息对象。

当一个 SWF 文件尝试访问麦克风时，Flash Player 显示“隐私”对话框，用户可以从中选择是允许还是拒绝访问。

- 如果用户允许访问，则 `Microphone.muted` 属性设置为 `false`，并且用代码属性为“`Microphone.Unmuted`”且级别属性为“`Status`”的一个信息对象调用此事件处理函数。
- 如果用户拒绝访问，则 `Microphone.muted` 属性设置为 `true`，并且用代码属性为“`Microphone.Muted`”且级别属性为“`Status`”的一个信息对象调用此事件处理函数。

若要在不处理此事件处理函数的情况下确定用户是拒绝还是允许对麦克风的访问，请使用 `Microphone.muted`。

注意：如果用户选择永久允许或拒绝对来自指定域的所有 SWF 文件的访问，则对于来自该域的 SWF 文件不调用此处理函数，除非用户以后更改该隐私设置。

有关更多信息，请参阅 `Microphone.get()`。

可用性：ActionScript 1.0；Flash Player 6

### 参数

`infoObject:Object` - 根据状态消息定义的参数。

### 示例

下面的示例启动“隐私”对话框，用户可以从中选择当单击超链接时是允许还是拒绝对麦克风的访问。如果用户选择拒绝访问，则以较大的红色文本显示已静音。如果允许访问麦克风，用户不会看到此文本。

```
this.createTextField("muted_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
muted_txt.autoSize = true;
muted_txt.html = true;
muted_txt.selectable = false;
muted_txt.htmlText = "<a href=\"asfunction:System.showSettings\"><u>Click Here</u></a> to Allow/Deny access.";
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
active_mic.onStatus = function(infoObj:Object) {
    status_txt._visible = active_mic.muted;
    muted_txt.htmlText = "Status: <a href=\"asfunction:System.showSettings\"><u>"+infoObj.code+"</u></a>";
```

```

};
this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 100,
    22);
status_txt.html = true;
status_txt.autoSize = true;
status_txt.htmlText = "<font size='72' color='#FF0000'>muted</font>";
status_txt._x = (Stage.width-status_txt._width)/2;
status_txt._y = (Stage.height-status_txt._height)/2;
status_txt._visible = active_mic.muted;

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[get \(Microphone.get 方法\)](#), [muted \(Microphone.muted 属性\)](#), [showSettings \(System.showSettings 方法\)](#), [onStatus \(System.onStatus 处理函数\)](#)

## rate (Microphone.rate 属性)

```
public rate : Number [read-only]
```

麦克风捕获声音的频率，单位是 **kHz**。如果您的声音捕获设备支持 **8 kHz**，则默认值为 **8 kHz**。否则，默认值为您的声音捕获设备支持且高于 **8 kHz** 的下一个可用捕获级别，通常为 **11 kHz**。

若要设置此值，请使用 `Microphone.setRate()`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的代码使您可以使用名为 `rate_cb` 的 **ComboBox** 实例更改麦克风捕获声音的频率。当前频率显示在名为 `rate_lbl1` 的 **Label** 实例中。

```

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
var rate_array:Array = new Array(5, 8, 11, 22, 44);
rate_cb.dataProvider = rate_array;
rate_cb.labelFunction = function(item:Object) {
    return (item+" kHz");
};
for (var i = 0; i<rate_array.length; i++) {
    if (rate_cb.getItemAt(i) == active_mic.rate) {
        rate_cb.selectedIndex = i;
        break;
    }
}

```

```
function changeRate() {
    active_mic.setRate(rate_cb.selectedItem);
    rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
}
rate_cb.addEventListener("change", changeRate);
rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setRate \(Microphone.setRate 方法\)](#)

## setGain (Microphone.setGain 方法)

```
public setGain(gain:Number) : Void
```

设置麦克风增益，即麦克风在传送信号之前应该将信号放大的倍数。值 0 指示 **Flash** 乘以 0；即麦克风不传送声音。

您可以将此设置想象为立体声音响上的音量旋钮：0 表示没有音量，50 表示正常音量；小于 50 的数字指定低于正常音量，而大于 50 的数字指定高于正常音量。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**gain**:Number - 指定麦克风信号提升量的整数。有效值的范围是 0 到 100。默认值为 50；但是，用户可以在 **Flash Player** “麦克风设置” 面板中更改该值。

### 示例

下面的示例使用名为 `gain_pb` 的 **ProgressBar** 实例显示麦克风的增益值，并使用名为 `gain_nstep` 的 **NumericStepper** 实例设置麦克风的增益值。

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

gain_pb.label = "Gain: %3";
gain_pb.mode = "manual";
gain_pb.setProgress(active_mic.gain, 100);
gain_nstep.value = active_mic.gain;

function changeGain() {
    active_mic.setGain(gain_nstep.value);
    gain_pb.setProgress(active_mic.gain, 100);
}
```

```
gain_nstep.addEventListener("change", changeGain);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[gain \(Microphone.gain 属性\)](#), [setUseEchoSuppression \(Microphone.setUseEchoSuppression 方法\)](#)

## setRate (Microphone.setRate 方法)

```
public setRate(rate:Number) : Void
```

设置麦克风应该捕获声音的频率，单位是 **kHz**。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**rate:**`Number` - 麦克风捕获声音应使用的频率，单位是 **kHz**。可接受的值为 **5**、**8**、**11**、**22** 和 **44**。如果您的声音捕获设备支持，则默认值为 **8 kHz**。否则，默认值为您的声音捕获设备支持且高于 **8 kHz** 的下一个可用捕获级别，通常为 **11 kHz**。

### 示例

下面的示例将麦克风频率设置为用户的首选项（已分配给 `userRate` 变量的值），前提是它是以下值之一: **5**、**8**、**11**、**22** 或 **44**。否则，该值将舍入到最近似的且声音捕获设备支持的可接受值。

```
active_mic.setRate(userRate);
```

下面的示例使您可以使用名为 `rate_cb` 的 **ComboBox** 实例更改麦克风捕获声音的频率。当前频率显示在名为 `rate_lbl` 的 **Label** 实例中。

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
var rate_array:Array = new Array(5, 8, 11, 22, 44);
rate_cb.dataProvider = rate_array;
rate_cb.labelFunction = function(item:Object) {
    return (item+" kHz");
};
for (var i = 0; i<rate_array.length; i++) {
    if (rate_cb.getItemAt(i) == active_mic.rate) {
        rate_cb.selectedIndex = i;
        break;
    }
}
```

```
function changeRate() {
    active_mic.setRate(rate_cb.selectedItem);
    rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
}
rate_cb.addEventListener("change", changeRate);
rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[rate \(Microphone.rate 属性\)](#)

## setSilenceLevel (Microphone.setSilenceLevel 方法)

```
public setSilenceLevel(silenceLevel:Number, [timeOut:Number]) : Void
```

设置应该被视为有声的最小输入级别以及（可选）指示静音已实际启用的静音时间。

- 若要防止麦克风检测到任何声音，请为 *level* 传递值 **100**；这样就决不会调用 `Microphone.onActivity`。
- 若要确定麦克风当前所检测的音量，请使用 `Microphone.activityLevel`。

活动检测是检测声音级别在何时表示某人正在谈话的功能。当某人没有谈话时，由于不需要发送关联的音频流，因此可以节约带宽。此信息也可用于视频反馈，以便让用户知道他们（或其他人）没有谈话。

静音值与活动值直接对应。完全静音时活动值为 **0**。持续噪音（可以根据当前增益设置检测到的噪音）时活动值为 **100**。当增益得到适当调整之后，在您未谈话时，活动值将小于静音值；而在您谈话时活动值将大于静音值。

此方法的用途与 `Camera.setMotionLevel()` 相同；这两种方法都用于指定应该在何时调用 `onActivity` 事件处理函数。但是这些方法对发布流具有非常不同的影响：

- `Camera.setMotionLevel()` 设计用于检测运动，它不影响带宽用量。即使视频流未检测到运动，仍将发送视频。
- `Microphone.setSilenceLevel()` 设计用于优化带宽。在认为音频流被静音时，不发送任何音频数据。所发送的是一个指示静音已启动的消息。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**silenceLevel**:Number - 一个整数, 指定激活麦克风和调用 Microphone.onActivity(true) 所需的音量。可接受值的范围为从 0 到 100。默认值为 10。

**timeOut**:Number [ 可选 ] - 一个整数, 指定必须经过多少毫秒的不活动时间, **Flash** 才能认为声音已停止并调用 Microphone.onActivity(false)。默认值为 2000 (2 秒)。

## 示例

以下示例根据用户在名为 silenceLevel\_nstep 的 **NumericStepper** 实例中的输入来更改静音级别。名为 silenceLevel\_pb 的 **ProgressBar** 实例根据是否将音频流视为静音来修改其外观。否则, 它显示音频流的活动级别。

```
var silenceLevel_pb:mx.controls.ProgressBar;
var silenceLevel_nstep:mx.controls.NumericStepper;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

silenceLevel_pb.label = "Activity level: %3";
silenceLevel_pb.mode = "manual";
silenceLevel_nstep.minimum = 0;
silenceLevel_nstep.maximum = 100;
silenceLevel_nstep.value = active_mic.silenceLevel;

var nstepListener:Object = new Object();
nstepListener.change = function(evt:Object) {
    active_mic.setSilenceLevel(evt.target.value, active_mic.silenceTimeOut);
};
silenceLevel_nstep.addEventListener("change", nstepListener);

this.onEnterFrame = function() {
    silenceLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
    if (active) {
        silenceLevel_pb.indeterminate = false;
        silenceLevel_pb.setStyle("themeColor", "haloGreen");
        silenceLevel_pb.label = "Activity level: %3";
    } else {
        silenceLevel_pb.indeterminate = true;
        silenceLevel_pb.setStyle("themeColor", "0xFF0000");
        silenceLevel_pb.label = "Activity level: (inactive)";
    }
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

`setMotionLevel` (`Camera.setMotionLevel` 方法), `activityLevel` (`Microphone.activityLevel` 属性), `onActivity` (`Microphone.onActivity` 处理函数), `setGain` (`Microphone.setGain` 方法), `silenceLevel` (`Microphone.silenceLevel` 属性), `silenceTimeOut` (`Microphone.silenceTimeOut` 属性)

## setUseEchoSuppression (`Microphone.setUseEchoSuppression` 方法)

```
public setUseEchoSuppression(useEchoSuppression:Boolean) : Void
```

指定是否使用音频编解码器的回声抑制功能。除非用户已经在 **Flash Player** 的“麦克风设置”面板中选择了“降低回声”，否则默认值为 `false`。

回声抑制是指降低音频回馈效果，当扬声器发出的声音由同一台计算机上的麦克风拾取时，将导致音频回馈。（这不同于回声消除，后者完全消除回馈。）

通常情况下，当所捕获的声音通过同一台计算机上的扬声器（而不是耳机）播放时，建议使用回声抑制。如果您的 **SWF** 文件允许用户指定声音输出设备，则当他们指定使用扬声器并且还使用麦克风时，您可能需要调用 `Microphone.setUseEchoSuppression(true)`。

用户也可以在 **Flash Player** 的“麦克风设置”面板中调整这些设置。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**useEchoSuppression:Boolean** - 一个布尔值，指示是否应该使用 (`true`) 或不使用 (`false`) 回音抑制。

### 示例

下面的示例将在用户选择名为 `useEchoSuppression_ch` 的实例时打开回音抑制。名为 `activityLevel_pb` 的 **ProgressBar** 实例将显示音频流的当前活动级别。

```
var useEchoSuppression_ch:mx.controls.CheckBox;
var activityLevel_pb:mx.controls.ProgressBar;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

activityLevel_pb.mode = "manual";
```



```

activityLevel_pb.label = "Activity Level: %3";
useEchoSuppression_ch.selected = active_mic.useEchoSuppression;
this.onEnterFrame = function() {
    activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
var chListener:Object = new Object();
chListener.click = function(evt:Object) {
    active_mic.setUseEchoSuppression(evt.target.selected);
};
useEchoSuppression_ch.addEventListener("click", chListener);

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setUseEchoSuppression \(Microphone.setUseEchoSuppression 方法\)](#),  
[useEchoSuppression \(Microphone.useEchoSuppression 属性\)](#)

## silenceLevel (Microphone.silenceLevel 属性)

```
public silenceLevel : Number [read-only]
```

一个整数，指定激活麦克风和调用 `Microphone.onActivity(true)` 所需的音量。默认值为 10。

可用性: **ActionScript 1.0** ; **Flash Player 6**

示例

以下示例根据用户在名为 `silenceLevel_nstep` 的 **NumericStepper** 实例中的输入来更改静音级别。名为 `silenceLevel_pb` 的 **ProgressBar** 实例根据是否将音频流视为静音来修改其外观。否则，它显示音频流的活动级别。

```

var silenceLevel_pb:mx.controls.ProgressBar;
var silenceLevel_nstep:mx.controls.NumericStepper;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

silenceLevel_pb.label = "Activity level: %3";
silenceLevel_pb.mode = "manual";
silenceLevel_nstep.minimum = 0;
silenceLevel_nstep.maximum = 100;
silenceLevel_nstep.value = active_mic.silenceLevel;

```

```

var nstepListener:Object = new Object();
nstepListener.change = function(evt:Object) {
    active_mic.setSilenceLevel(evt.target.value, active_mic.silenceTimeout);
};
silenceLevel_nstep.addEventListener("change", nstepListener);

this.onEnterFrame = function() {
    silenceLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
    if (active) {
        silenceLevel_pb.indeterminate = false;
        silenceLevel_pb.setStyle("themeColor", "haloGreen");
        silenceLevel_pb.label = "Activity level: %3";
    } else {
        silenceLevel_pb.indeterminate = true;
        silenceLevel_pb.setStyle("themeColor", "0xFF0000");
        silenceLevel_pb.label = "Activity level: (inactive)";
    }
};

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[gain \(Microphone.gain 属性\)](#), [setSilenceLevel \(Microphone.setSilenceLevel 方法\)](#)

## silenceTimeout (Microphone.silenceTimeout 属性)

`public silenceTimeout : Number [read-only]`

一个数字值，表示麦克风停止检测声音的时间和调用 `Microphone.onActivity(false)` 的时间之间的毫秒数。默认值为 **2000**（2 秒）。

若要设置此值，请使用 `Microphone.setSilenceLevel()`。

**可用性:** **ActionScript 1.0** ; **Flash Player 6**

示例

下面的示例使用户能够控制麦克风停止检测声音的时间与调用

`Microphone.onActivity(false)` 的时间之间的时间量。用户使用名为

**silenceTimeout\_nstep** 的 **NumericStepper** 实例控制该值。名为 `silenceLevel_pb` 的 **ProgressBar** 实例根据是否将音频流视为静音来修改其外观。否则，它显示音频流的活动级别。

```

var silenceLevel_pb:mx.controls.ProgressBar;
var silenceTimeOut_nstep:mx.controls.NumericStepper;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

silenceLevel_pb.label = "Activity level: %3";
silenceLevel_pb.mode = "manual";
silenceTimeOut_nstep.minimum = 0;
silenceTimeOut_nstep.maximum = 10;
silenceTimeOut_nstep.value = active_mic.silenceTimeOut/1000;

var nstepListener:Object = new Object();
nstepListener.change = function(evt:Object) {
    active_mic.setSilenceLevel(active_mic.silenceLevel, evt.target.value
    1000);
};
silenceTimeOut_nstep.addEventListener("change", nstepListener);

this.onEnterFrame = function() {
    silenceLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
    if (active) {
        silenceLevel_pb.indeterminate = false;
        silenceLevel_pb.setStyle("themeColor", "haloGreen");
        silenceLevel_pb.label = "Activity level: %3";
    } else {
        silenceLevel_pb.indeterminate = true;
        silenceLevel_pb.setStyle("themeColor", "0xFF0000");
        silenceLevel_pb.label = "Activity level: (inactive)";
    }
};

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setSilenceLevel \(Microphone.setSilenceLevel 方法\)](#)

## useEchoSuppression (Microphone.useEchoSuppression 属性)

public useEchoSuppression : Boolean

属性（只读）：一个布尔值，如果启用回音抑制，则为 true；否则为 false。除非用户已经在 **Flash Player** 的“麦克风设置”面板中选择了“降低回声”，否则默认值为 false。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例将在用户选择名为 useEchoSuppression\_ch 的实例时打开回音抑制。名为 activityLevel\_pb 的 **ProgressBar** 实例将显示音频流的当前活动级别。

```
var useEchoSuppression_ch:mx.controls.CheckBox;
var activityLevel_pb:mx.controls.ProgressBar;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

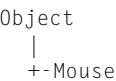
activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3";
useEchoSuppression_ch.selected = active_mic.useEchoSuppression;
this.onEnterFrame = function() {
    activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
var chListener:Object = new Object();
chListener.click = function(evt:Object) {
    active_mic.setUseEchoSuppression(evt.target.selected);
};
useEchoSuppression_ch.addEventListener("click", chListener);
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

### 另请参见

[setUseEchoSuppression \(Microphone.setUseEchoSuppression 方法\)](#)

# Mouse



```
public class Mouse
extends Object
```

**Mouse** 类是不通过构造函数即可访问其属性和方法的顶级类。您可以使用 **Mouse** 类的方法来隐藏和显示 **SWF** 文件中的鼠标指针（光标）。默认情况下鼠标指针是可见的，但是您可以将其隐藏并实现用影片剪辑创建的自定义指针。

**Flash** 应用程序仅可以监视在其焦点以内发生的鼠标事件。**Flash** 应用程序无法检测另一个应用程序中的鼠标事件。

可用性：ActionScript 1.0 ； Flash Player 5

### 属性摘要

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

### 事件摘要

事件	说明
onMouseDown = function() {}	当按下鼠标时获得通知。
onMouseMove = function() {}	当鼠标移动时获得通知。
onMouseUp = function() {}	当释放鼠标时获得通知。
onMouseWheel = function([delta: Number], [scrollTarget:String]) {}	当用户滚动鼠标滚轮时获得通知。

方法摘要

修饰符	签名	说明
static	<code>addListener(listener:Object) : Void</code>	注册一个对象以接收 <code>onMouseDown</code> 、 <code>onMouseMove</code> 、 <code>onMouseUp</code> 和 <code>onMouseWheel</code> 侦听器的通知。
static	<code>hide() : Number</code>	在 SWF 文件中隐藏指针。
static	<code>removeListener(listener:Object) : Boolean</code>	删除以前向 <code>addListener()</code> 注册的对象。
static	<code>show() : Number</code>	在 SWF 文件中显示鼠标指针。

继承自 `Object` 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## addListener (Mouse.addListener 方法)

`public static addListener(listener:Object) : Void`

注册一个对象以接收 `onMouseDown`、`onMouseMove`、`onMouseUp` 和 `onMouseWheel` 侦听器的通知。（仅 **Windows** 中支持 `onMouseWheel` 侦听器。）

`listener` 参数应包含具有至少一个侦听器的已定义方法的对象。

当按下、移动、释放鼠标或使用鼠标滚动时，不论输入焦点位于何处，用此方法注册的所有侦听对象都将调用其 `onMouseDown`、`onMouseMove`、`onMouseUp` 或 `onMouseWheel` 方法。可以有多个对象侦听鼠标通知。如果已经注册了 `listener`，则不会发生任何更改。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

`listener:Object` - 一个对象。

## 示例

此示例摘自 **ActionScript** 示例文件夹中的 **animation.fla** 文件。

```
// Create a mouse listener object
var mouseListener:Object = new Object();

// Every time the mouse cursor moves within the SWF file,
// update the position of the crosshair movie clip
// instance on the Stage.
mouseListener.onMouseMove = function() {
    crosshair_mc._x = _xmouse;
    crosshair_mc._y = _ymouse;
};

// When you click the mouse, check to see if the cursor is within the
// boundaries of the Stage. If so, increment the number of shots.
mouseListener.onMouseDown = function() {
    if (bg_mc.hitTest(_xmouse, _ymouse, false)) {
        _global.shots++;
    }
};
Mouse.addListener(mouseListener);
```

若要查看整个脚本，请参见 **ActionScript** 示例文件夹中的 **animation.fla** 文件。下面的列表显示到 **ActionScript** 示例文件夹的典型路径：

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

另请参见

[onMouseDown](#) ([Mouse.onMouseDown 事件侦听器](#))，[onMouseMove](#) ([Mouse.onMouseMove 事件侦听器](#))，[onMouseUp](#) ([Mouse.onMouseUp 事件侦听器](#))，[onMouseWheel](#) ([Mouse.onMouseWheel 事件侦听器](#))

## hide (Mouse.hide 方法)

public static hide() : Number

在 SWF 文件中隐藏指针。默认情况下，指针是可见的。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数；值为 0 或 1。如果鼠标指针在调用 `Mouse.hide()` 前被隐藏，则返回值 0。如果鼠标指针在调用 `Mouse.hide()` 前被显示，则返回值 1。

### 示例

下面的代码隐藏标准鼠标指针，并将 `pointer_mc` 影片剪辑实例的 `x` 和 `y` 位置设置为 `x` 和 `y` 指针位置。创建一个影片剪辑并将它的链接标识符设置为 `pointer_id`。将下面的

**ActionScript** 添加到时间轴的第 1 帧：

```
// to use this script you need a symbol
// in your library with a Linkage Identifier of "pointer_id".
this.attachMovie("pointer_id", "pointer_mc", this.getNextHighestDepth());
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    pointer_mc._x = _xmouse;
    pointer_mc._y = _ymouse;
    updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[show \(Mouse.show 方法\)](#)，[\\_xmouse \(MovieClip.\\_xmouse 属性\)](#)，[\\_ymouse \(MovieClip.\\_ymouse 属性\)](#)



## onMouseDown (Mouse.onMouseDown 事件侦听器)

```
onMouseDown = function() {}
```

当按下鼠标时获得通知。若要使用 onMouseDown 侦听器，您必须创建一个侦听器对象。然后可以为 onMouseDown 定义一个函数，并使用 addListener() 注册含有 **Mouse** 对象的侦听器，如以下代码所示：

```
var someListener:Object = new Object();
someListener.onMouseDown = function () { ... };
Mouse.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

**Flash** 应用程序仅可以监视在其焦点以内发生的鼠标事件。**Flash** 应用程序无法检测另一个应用程序中的鼠标事件。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例每当用户在运行时单击、拖动和松开鼠标时使用 **Drawing API** 绘制一个矩形。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    this.orig_x = _xmouse;
    this.orig_y = _ymouse;
    this.target_mc = canvas_mc.createEmptyMovieClip("",
        canvas_mc.getNextHighestDepth());
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        this.target_mc.clear();
        this.target_mc.lineStyle(1, 0xFF0000, 100);
        this.target_mc.moveTo(this.orig_x, this.orig_y);
        this.target_mc.lineTo(_xmouse, this.orig_y);
        this.target_mc.lineTo(_xmouse, _ymouse);
        this.target_mc.lineTo(this.orig_x, _ymouse);
        this.target_mc.lineTo(this.orig_x, this.orig_y);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

另请参见

[addListener](#) ([Mouse.addListener](#) 方法)

## onMouseMove (Mouse.onMouseMove 事件侦听器)

```
onMouseMove = function() {}
```

当鼠标移动时获得通知。若要使用 `onMouseMove` 侦听器，您必须创建一个侦听器对象。然后可以为 `onMouseMove` 定义一个函数，并使用 `addListener()` 注册含有 **Mouse** 对象的侦听器，如以下代码所示：

```
var someListener:Object = new Object();
someListener.onMouseMove = function () { ... };
Mouse.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

**Flash** 应用程序仅可以监视在其焦点以内发生的鼠标事件。**Flash** 应用程序无法检测另一个应用程序中的鼠标事件。

**可用性：** **ActionScript 1.0**； **Flash Player 6**

### 示例

以下示例将鼠标指针用作工具，以便使用 `onMouseMove` 和 **Drawing API** 来绘制线条。用户拖动鼠标指针就可以绘制线条。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

下面的示例隐藏标准鼠标指针，并将 pointer\_mc 影片剪辑实例的 x 和 y 位置设置为 x 和 y 指针位置。创建一个影片剪辑并将它的链接标识符设置为 pointer\_id。将下面的

**ActionScript** 添加到时间轴的第 1 帧：

```
// to use this script you need a symbol
// in your library with a Linkage Identifier of "pointer_id".
this.attachMovie("pointer_id", "pointer_mc", this.getNextHighestDepth());
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    pointer_mc._x = _xmouse;
    pointer_mc._y = _ymouse;
    updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

另请参见

[addListener](#) ([Mouse.addListener](#) 方法)

## onMouseUp (Mouse.onMouseUp 事件侦听器)

```
onMouseUp = function() {}
```

当释放鼠标时获得通知。若要使用 onMouseUp 侦听器，您必须创建一个侦听器对象。然后可以为 onMouseUp 定义一个函数，并使用 addListener() 注册含有 **Mouse** 对象的侦听器，如下代码所示：

```
var someListener:Object = new Object();
someListener.onMouseUp = function () { ... };
Mouse.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

**Flash** 应用程序仅可以监视在其焦点以内发生的鼠标事件。**Flash** 应用程序无法检测另一个应用程序中的鼠标事件。

可用性：ActionScript 1.0；Flash Player 6

## 示例

以下示例将鼠标指针用作工具，以便使用 `onMouseMove` 和 **Drawing API** 来绘制线条。用户拖动鼠标指针就可以绘制线条。用户释放鼠标按钮就可以停止绘制线条。

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[addListener \(Mouse.addListener 方法\)](#)

## onMouseWheel (Mouse.onMouseWheel 事件侦听器)

```
onMouseWheel = function([delta:Number], [scrollTarget:String]) {}
```

当用户滚动鼠标滚轮时获得通知。若要使用 `onMouseWheel` 侦听器，您必须创建一个侦听器对象。然后可以为 `onMouseWheel` 定义一个函数，并使用 `addListener()` 注册含有 **Mouse** 对象的侦听器。

**注意：**鼠标滚轮事件侦听器只有在 **Windows** 版的 **Flash Player** 中才可用。

**Flash** 应用程序仅可以监视在其焦点以内发生的鼠标事件。**Flash** 应用程序无法检测另一个应用程序中的鼠标事件。

**可用性：** **ActionScript 1.0**； **Flash Player 6**

## 参数

**delta:**Number [ 可选 ] - 一个数字，指示用户每滚动鼠标滚轮一个刻度将滚动多少行。正 delta 值指示向上滚动；负值指示向下滚动。典型值为从 1 到 3；值越大，滚动得越快。

**scrollTarget:**String [ 可选 ] - 一个参数，表示滚动鼠标滚轮时，位于鼠标指针下面最顶端的影片剪辑实例。如果您希望为 scrollTarget 指定值，但不希望为 delta 指定值，则请为 delta 传递 null。

## 示例

以下示例显示如何创建对鼠标滚轮事件做出响应的侦听器对象。在此示例中，每当用户滚动鼠标滚轮时，名为 clip\_mc 的影片剪辑对象的 x 坐标都会改变：

```
var mouseListener:Object = new Object();
mouseListener.onMouseWheel = function(delta) {
    clip_mc._x += delta;
}
Mouse.addListener(mouseListener);
```

下面的示例绘制一个线条，该线条在用户滚动鼠标滚轮时会旋转。在运行时单击 SWF 文件，然后滚动鼠标滚轮以查看影片剪辑的活动情况。

```
this.createEmptyMovieClip("line_mc", this.getNextHighestDepth());
line_mc.lineStyle(2, 0xFF0000, 100);
line_mc.moveTo(0, 100);
line_mc.lineTo(0, 0);
line_mc._x = 200;
line_mc._y = 200;
```

```
var mouseListener:Object = new Object();
mouseListener.onMouseWheel = function(delta:Number) {
    line_mc._rotation += delta;
};
mouseListener.onMouseDown = function() {
    trace("Down");
};
Mouse.addListener(mouseListener);
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 Flash Player 7 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

## 另请参见

[addListener \(Mouse.addListener 方法\)](#)，[mouseWheelEnabled \(TextField.mouseWheelEnabled 属性\)](#)

## removeListener (Mouse.removeListener 方法)

public static removeListener(listener:Object) : Boolean

删除以前向 addListener() 注册的对象。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**listener:Object** - 一个对象。

### 返回

Boolean - 如果成功删除了 listener 对象, 则该方法返回 true ; 如果没有成功删除 listener (例如, 如果 listener 不在 **Mouse** 对象的侦听器列表中), 则该方法返回 false。

### 示例

下面的实例在舞台上附加三个按钮, 并让用户在运行时用鼠标指针在 **SWF** 文件中绘制线条。一个按钮从 **SWF** 文件中清除所有线条。第二个按钮删除鼠标侦听器, 所以用户无法绘制线条。第三个按钮在鼠标侦听器被删除后添加鼠标侦听器, 所以用户可以再次绘制线条。

将下面的 **ActionScript** 添加到时间轴的第 1 帧:

```
this.createClassObject(mx.controls.Button, "clear_button",
    this.getNextHighestDepth(), {_x:10, _y:10, label:'clear'});
this.createClassObject(mx.controls.Button, "stopDrawing_button",
    this.getNextHighestDepth(), {_x:120, _y:10, label:'stop drawing'});
this.createClassObject(mx.controls.Button, "startDrawing_button",
    this.getNextHighestDepth(), {_x:230, _y:10, label:'start drawing'});
startDrawing_button.enabled = false;
//
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    this.isDrawing = true;
    canvas_mc.lineStyle(2, 0xFF0000, 100);
    canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
    if (this.isDrawing) {
        canvas_mc.lineTo(_xmouse, _ymouse);
    }
    updateAfterEvent();
};
mouseListener.onMouseUp = function() {
    this.isDrawing = false;
};
Mouse.addListener(mouseListener);
var clearListener:Object = new Object();
```

```

clearListener.click = function() {
    canvas_mc.clear();
};
clear_button.addEventListener("click", clearListener);
//
var stopDrawingListener:Object = new Object();
stopDrawingListener.click = function(evt:Object) {
    Mouse.removeListener(mouseListener);
    evt.target.enabled = false;
    startDrawing_button.enabled = true;
};
stopDrawing_button.addEventListener("click", stopDrawingListener);
var startDrawingListener:Object = new Object();
startDrawingListener.click = function(evt:Object) {
    Mouse.addListener(mouseListener);
    evt.target.enabled = false;
    stopDrawing_button.enabled = true;
};
startDrawing_button.addEventListener("click", startDrawingListener);

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## show (Mouse.show 方法)

`public static show() : Number`

在 **SWF** 文件中显示鼠标指针。默认情况下，指针是可见的。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

**Number** - 一个整数；值为 0 或 1。如果鼠标指针在调用 `Mouse.show()` 前被隐藏，则返回值 0。如果鼠标指针在调用 `Mouse.show()` 前被显示，则返回值 1。

### 示例

下面的示例在鼠标指针移过名为 `my_mc` 的影片剪辑时从库中附加一个自定义指针。指定库中的影片剪辑的链接标识符为 `cursor_help_id`，并将以下 **ActionScript** 添加到时间轴的第一帧：

```

my_mc.onRollOver = function() {
    Mouse.hide();
    this.attachMovie("cursor_help_id", "cursor_mc",
        this.getNextHighestDepth(), {_x:this._xmouse, _y:this._ymouse});
};
my_mc.onMouseMove = function() {
    this.cursor_mc._x = this._xmouse;

```

```

        this.cursor_mc._y = this._ymouse;
    };
    my_mc.onRollOut = function() {
        Mouse.show();
        this.cursor_mc.removeMovieClip();
    };

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[hide \(Mouse.hide 方法\)](#), [\\_xmouse \(MovieClip.\\_xmouse 属性\)](#), [\\_ymouse \(MovieClip.\\_ymouse 属性\)](#)

## MovieClip

```

Object
|
+-MovieClip

```

```

public dynamic class MovieClip
extends Object

```

**MovieClip** 类的方法提供的功能与定位影片剪辑的动作所提供的功能相同。还有一些其它方法在“动作”面板中的“动作”工具箱中没有等效动作。

请不要使用构造函数方法来创建影片剪辑。要创建新的影片剪辑实例，您可以从以下三种方法中选择：

- 通过 `attachMovie()` 方法，可以基于库中存在的影片剪辑元件创建影片剪辑实例。
- 通过 `createEmptyMovieClip()` 方法，可以基于其它影片剪辑创建新的空影片剪辑实例以作为子级。
- 通过 `duplicateMovieClip()` 方法，可以基于其它影片剪辑创建影片剪辑实例。

若要调用 **MovieClip** 类的方法，请使用以下语法按名称引用影片剪辑实例，其中 `my_mc` 是影片剪辑实例：

```

my_mc.play();
my_mc.gotoAndPlay(3);

```

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 3



属性摘要

修饰符	属性	说明
	<code>_alpha:Number</code>	影片剪辑的 Alpha 透明度值。
	<code>blendMode:Object</code>	此影片剪辑的混合模式。
	<code>cacheAsBitmap:Boolean</code>	如果设置为 <code>true</code> ，则 Flash Player 将缓存影片剪辑的内部位图表示。
	<code>_currentframe:Number</code> [ 只读 ]	返回指定帧的编号，该帧中的播放头位于影片剪辑的时间轴中。
	<code>_droptarget:String</code> [ 只读 ]	返回在其上放置此影片剪辑的影片剪辑实例的绝对路径，以斜杠语法记号表示。
	<code>enabled:Boolean</code>	一个布尔值，指示影片剪辑是否处于活动状态。
	<code>filters:Array</code>	一个索引数组，包含当前与影片剪辑相关联的每个过滤器对象。
	<code>focusEnabled:Boolean</code>	如果值为 <code>undefined</code> 或 <code>false</code> ，则除非影片剪辑是一个按钮，否则它无法获得输入焦点。
	<code>_focusrect:Boolean</code>	一个布尔值，指定当影片剪辑具有键盘焦点时其周围是否有黄色矩形。
	<code>_framesloaded:Number</code> [ 只读 ]	从流式 SWF 文件加载的帧数。
	<code>_height:Number</code>	影片剪辑的高度，以像素为单位。
	<code>_highquality:Number</code>	自 Flash Player 7 后不推荐使用。不推荐使用此属性，而推荐使用 <code>MovieClip._quality</code> 。指定当前 SWF 文件所应用的消除锯齿的级别。
	<code>hitArea:Object</code>	将另一个影片剪辑指定为影片剪辑的点击区域。
	<code>_lockroot:Boolean</code>	一个布尔值，指定将 SWF 文件加载到影片剪辑中时 <code>_root</code> 引用的内容。
	<code>menu:ContextMenu</code>	将指定的 <code>ContextMenu</code> 对象与影片剪辑相关联。
	<code>_name:String</code>	影片剪辑的实例名称。
	<code>opaqueBackground:Number</code>	由数字（RGB 十六进制值）指定的颜色的影片剪辑的不透明背景颜色。
	<code>_parent:MovieClip</code>	对包含当前影片剪辑或对象的影片剪辑或对象的引用。
	<code>_quality:String</code>	设置或检索用于 SWF 文件的呈现品质。
	<code>_rotation:Number</code>	指定影片剪辑相对于其原始方向的旋转程度，以度为单位。

修饰符	属性	说明
	scale9Grid:Rectangle	矩形区域，它定义影片剪辑的九个缩放区域。
	scrollRect:Object	通过 scrollRect 属性，可以快速滚动影片剪辑内容，并具有一个用来查看较大内容的窗口。
	_soundbuftime:Number	指定在声音开始进入流之前，预先缓冲的秒数。
	tabChildren:Boolean	确定影片剪辑的子级是否包括在 Tab 键的自动排序中。
	tabEnabled:Boolean	指定影片剪辑是否包括在 Tab 键的自动排序中。
	tabIndex:Number	可用于自定义影片中对象的 Tab 键排序。
	_target:String [ 只读 ]	返回影片剪辑实例的目标路径，以斜杠记号表示。
	_totalframes:Number [ 只读 ]	返回由 MovieClip 参数指定的影片剪辑实例中的总帧数。
	trackAsMenu:Boolean	布尔值，指示其它按钮或影片剪辑是否可接收鼠标释放事件。
	transform:Transform	一个对象，具有与影片剪辑的矩阵、颜色转换和像素范围有关的属性。
	_url:String [ 只读 ]	检索从其下载影片剪辑的 SWF、JPEG、GIF 或 PNG 文件的 URL。
	useHandCursor:Boolean	一个布尔值，指示当鼠标滑过影片剪辑时是否显示手指形（手形光标）。
	_visible:Boolean	一个布尔值，指示影片剪辑是否处于可见状态。
	_width:Number	影片剪辑的宽度，以像素为单位。
	_x:Number	一个整数，它设置影片剪辑相对于父级影片剪辑的本地坐标的 x 坐标。
	_xmouse:Number [ 只读 ]	返回鼠标位置的 x 坐标。
	_xscale:Number	确定从影片剪辑注册点开始应用的影片剪辑水平缩放比例 (percentage)。
	_y:Number	设置影片剪辑相对于父级影片剪辑的本地坐标的 y 坐标。
	_ymouse:Number [ 只读 ]	指示鼠标位置的 y 坐标。
	_yscale:Number	设置从影片剪辑注册点开始应用的影片剪辑垂直缩放比例 (percentage)。

继承自 Object 类的属性

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

事件摘要

事件	说明
<code>onData = function() {}</code>	在影片剪辑从 <code>MovieClip.loadVariables()</code> 调用或 <code>MovieClip.loadMovie()</code> 调用获得数据时调用。
<code>onDragOut = function() {}</code>	当按下鼠标按钮并且指针滑出对象时调用。
<code>onDragOver = function() {}</code>	当鼠标指针在影片剪辑外拖动并且随后拖过该影片剪辑时调用。
<code>onEnterFrame = function() {}</code>	以 SWF 文件的帧频重复调用。
<code>onKeyDown = function() {}</code>	当影片剪辑具有输入焦点并且用户按下某个键时调用。
<code>onKeyUp = function() {}</code>	当释放按键时调用。
<code>onKillFocus = function(newFocus:Object) {}</code>	当影片剪辑失去键盘焦点时调用。
<code>onLoad = function() {}</code>	当影片剪辑被实例化并显示在时间轴上时调用。
<code>onMouseDown = function() {}</code>	当按下鼠标按钮时调用。
<code>onMouseMove = function() {}</code>	当鼠标移动时调用。
<code>onMouseUp = function() {}</code>	释放鼠标按钮时调用。
<code>onPress = function() {}</code>	当鼠标指针处于影片剪辑之上而用户单击鼠标时调用。

事件	说明
<code>onRelease = function() {}</code>	当用户在影片剪辑上释放鼠标按钮时调用。
<code>onReleaseOutside = function() {}</code>	用户在影片剪辑区域中按下鼠标按钮并且在影片剪辑区域之外释放它后调用。
<code>onRollOut = function() {}</code>	当鼠标指针移到影片剪辑区域的外面时调用。
<code>onRollOver = function() {}</code>	当鼠标指针滑过影片剪辑区域时调用。
<code>onSetFocus = function(oldFocus:Object) {}</code>	当影片剪辑获得键盘焦点时调用。
<code>onUnload = function() {}</code>	从时间轴删除影片剪辑后，在第 1 帧中调用。

## 方法摘要

修饰符	签名	说明
	<code>attachAudio(id:Object) : Void</code>	指定要播放的音频源。
	<code>attachBitmap(bmp:BitmapData, depth:Number, [pixelSnapping:String], [smoothing:Boolean]) : Void</code>	将位图图像附加到影片剪辑。
	<code>attachMovie(id:String, name:String, depth:Number, [initObject:Object]) : MovieClip</code>	从库中取得一个元件并将其附加到影片剪辑中。
	<code>beginBitmapFill(bmp:BitmapData, [matrix:Matrix], [repeat:Boolean], [smoothing:Boolean]) : Void</code>	用位图图像填充绘画区域。
	<code>beginFill(rgb:Number, [alpha:Number]) : Void</code>	指示新的绘画路径的开始。
	<code>beginGradientFill(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object, [spreadMethod:String], [interpolationMethod:String], [focalPointRatio:Number]) : Void</code>	指示新的绘画路径的开始。

修饰符	签名	说明
	<code>clear() : Void</code>	删除使用影片剪辑绘画方法（包括用 <code>MovieClip.lineStyle()</code> 指定的线条样式）在运行时创建的所有图形。
	<code>createEmptyMovieClip(name:String, depth:Number) : MovieClip</code>	创建一个空影片剪辑作为现有影片剪辑的子级。
	<code>createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:Number, height:Number) : TextField</code>	创建一个新的空文本字段作为在其上调用此方法的影片剪辑的子级。
	<code>curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number) : Void</code>	通过由 <code>((controlX,controlY)</code> 指定的控制点，使用当前线条样式绘制一条曲线，该曲线从当前绘画位置到 <code>(anchorX,anchorY)</code> 。
	<code>duplicateMovieClip(name:String, depth:Number, [initObject:Object]) : MovieClip</code>	在 SWF 文件正在播放时，创建指定影片剪辑的实例。
	<code>endFill() : Void</code>	对从上一次调用 <code>beginFill()</code> 或 <code>beginGradientFill()</code> 之后存在的直线或曲线应用填充。
	<code>getBounds(bounds:Object) : Object</code>	基于 <code>bounds</code> 参数，返回作为影片剪辑的最小和最大 <code>x</code> 和 <code>y</code> 坐标值的属性。
	<code>getBytesLoaded() : Number</code>	返回已加载（流处理）的影片剪辑的字节数。
	<code>getBytesTotal() : Number</code>	以字节为单位返回影片剪辑的大小。
	<code>getDepth() : Number</code>	返回影片剪辑实例的深度。
	<code>getInstanceAtDepth(depth:Number) : MovieClip</code>	确定特定深度是否已被影片剪辑占用。
	<code>getNextHighestDepth() : Number</code>	确定可传递给 <code>MovieClip.attachMovie()</code> 、 <code>MovieClip.duplicateMovieClip()</code> 或 <code>MovieClip.createEmptyMovieClip()</code> 的深度值，以确保 Flash 将该影片剪辑呈现在当前影片剪辑中同一级和同一层上所有其它对象的前面。
	<code>getRect(bounds:Object) : Object</code>	基于 <code>bounds</code> 参数，返回作为影片剪辑的最小和最大 <code>x</code> 和 <code>y</code> 坐标值的属性，不包括形状上的任何笔触。

修饰符	签名	说明
	<code>getSWFVersion() : Number</code>	返回一个整数，该整数指示所发布的影片剪辑的 Flash Player 版本。
	<code>getTextSnapshot() : TextSnapshot</code>	返回一个 <code>TextSnapshot</code> 对象，该对象包含指定影片剪辑的所有静态文本字段中的文本；不包括子级影片剪辑中的文本。
	<code>getURL(url:String, [window:String], [method:String]) : Void</code>	从指定 URL 将文档加载到指定窗口。
	<code>globalToLocal(pt:Object) : Void</code>	将 <code>pt</code> 对象从舞台（全局）坐标转换为影片剪辑（本地）坐标。
	<code>gotoAndPlay(frame:Object) : Void</code>	从指定帧开始播放 SWF 文件。
	<code>gotoAndStop(frame:Object) : Void</code>	将播放头移到影片剪辑的指定帧并停在那里。
	<code>hitTest() : Boolean</code>	计算影片剪辑，以确认其是否与由 <code>target</code> 或 <code>x</code> 和 <code>y</code> 坐标参数标识的点击区域发生重叠或相交。
	<code>lineGradientStyle(fillType:String, colors:Array, alphas:Array, ratios:Array, matrix:Object, [spreadMethod:String], [interpolationMethod:String], [focalPointRatio:Number]) : Void</code>	指定 Flash 用于后续 <code>lineTo()</code> 和 <code>curveTo()</code> 方法调用的线条样式，在以不同参数调用 <code>lineStyle()</code> 方法或 <code>lineGradientStyle()</code> 方法之前，线条样式不会改变。
	<code>lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number) : Void</code>	指定 Flash 用于后续 <code>lineTo()</code> 和 <code>curveTo()</code> 方法调用的线条样式，在以不同参数调用 <code>lineStyle()</code> 方法之前，线条样式不会改变。
	<code>lineTo(x:Number, y:Number) : Void</code>	使用当前线条样式绘制一条从当前绘画位置到 <code>(x,y)</code> 的线条；当前绘画位置随后会设置为 <code>(x,y)</code> 。
	<code>loadMovie(url:String, [method:String]) : Void</code>	在播放原始 SWF 文件时，将 SWF、JPEG、GIF 或 PNG 文件加载到 Flash Player 中的影片剪辑中。
	<code>loadVariables(url:String, [method:String]) : Void</code>	从外部文件读取数据并设置影片剪辑中变量的值。

修饰符	签名	说明
	<code>localToGlobal(pt:Object) : Void</code>	将 pt 对象从影片剪辑（本地）坐标转换为舞台（全局）坐标。
	<code>moveTo(x:Number, y:Number) : Void</code>	将当前绘画位置移动到 (x, y)。
	<code>nextFrame() : Void</code>	将播放头转到下一帧并停止。
	<code>play() : Void</code>	在影片剪辑的时间轴中移动播放头。
	<code>prevFrame() : Void</code>	将播放头转到前一帧并停止。
	<code>removeMovieClip() : Void</code>	删除用 <code>duplicateMovieClip()</code> 、 <code>MovieClip.duplicateMovieClip()</code> 、 <code>MovieClip.createEmptyMovieClip()</code> 或 <code>MovieClip.attachMovie()</code> 创建的影片剪辑实例。
	<code>setMask(mc:Object) : Void</code>	使参数 mc 中的影片剪辑成为展示调用影片剪辑的遮罩层。
	<code>startDrag([lockCenter:Boolean], [left:Number], [top:Number], [right:Number], [bottom:Number]) : Void</code>	允许用户拖动指定的影片剪辑。
	<code>stop() : Void</code>	停止当前正在播放的影片剪辑。
	<code>stopDrag() : Void</code>	结束 <code>MovieClip.startDrag()</code> 方法。
	<code>swapDepths(target:Object) : Void</code>	交换此影片剪辑与另一影片剪辑的堆栈或深度级别（z- 顺序），另一影片剪辑由 target 参数指定，或指定为当前占用由 target 参数指定的深度级别的影片剪辑。
	<code>unloadMovie() : Void</code>	删除影片剪辑实例的内容。

继承自 Object 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## `_alpha` (MovieClip.\_alpha 属性)

`public _alpha : Number`

影片剪辑的 **Alpha** 透明度值。有效值为 **0**（完全透明）到 **100**（完全不透明）。默认值为 **100**。`_alpha` 设置为 **0** 的影片剪辑中的对象即使不可见，也将处于活动状态。例如，您仍可以单击 `_alpha` 属性设置为 **0** 的影片剪辑中的按钮。若要完全禁用该按钮，可以将影片剪辑的 `_visible` 属性设置为 **false**。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

下面的代码在鼠标滑过影片剪辑时将动态创建的名为 `triangle` 的影片剪辑的 `_alpha` 属性设置为 **50%**。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件:

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());
```

```
triangle.beginFill(0x0000FF, 100);  
triangle.moveTo(10, 10);  
triangle.lineTo(10, 100);  
triangle.lineTo(100, 10);  
triangle.lineTo(10, 10);
```

```
triangle.onRollOver = function() {  
    this._alpha = 50;  
};  
triangle.onRollOut = function() {  
    this._alpha = 100;  
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[\\_alpha \(Button.\\_alpha 属性\)](#), [\\_alpha \(TextField.\\_alpha 属性\)](#), [\\_visible \(MovieClip.\\_visible 属性\)](#)



## attachAudio (MovieClip.attachAudio 方法)

```
public attachAudio(id:Object) : Void
```

指定要播放的音频源。若要停止播放音频源，请传递 `id` 的值 `false`。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

`id:Object` - 包含要播放的音频的对象。有效值是 **Microphone** 对象、播放 **FLV** 文件的 **NetStream** 对象以及 `false` (停止播放音频)。

### 示例

下列示例创建一个新的 **NetStream** 连接。通过打开“库”面板并从“库”选项菜单中选择“新建视频”，添加一个新的视频元件。为元件指定实例名称 `my_video`。在运行时动态加载 **FLV** 视频。使用 `attachAudio()` 方法将音频从 **FLV** 文件附加到舞台上的影片剪辑。然后，可以使用 **Sound** 类和名为 `volUp_btn` 和 `volDown_btn` 的两个按钮控制影片剪辑中的音频。

```
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("yourVideo.flv");
this.createEmptyMovieClip("flv_mc", this.getNextHighestDepth());
flv_mc.attachAudio(my_ns);
var audio_sound:Sound = new Sound(flv_mc);

// Add volume buttons.
volUp_btn.onRelease = function() {
    if (audio_sound.getVolume()<100) {
        audio_sound.setVolume(audio_sound.getVolume()+10);
        updateVolume();
    }
};
volDown_btn.onRelease = function() {
    if (audio_sound.getVolume()>0) {
        audio_sound.setVolume(audio_sound.getVolume()-10);
        updateVolume();
    }
};

// Updates the volume.
this.createTextField("volume_txt", this.getNextHighestDepth(), 0, 0, 100,
    22);
updateVolume();
function updateVolume() {
```

```
    volume_txt.text = "Volume: "+audio_sound.getVolume();  
}
```

下面的示例将麦克风指定为动态创建的名为 `audio_mc` 的影片剪辑实例的音频源：

```
var active_mic:Microphone = Microphone.get();  
this.createEmptyMovieClip("audio_mc", this.getNextHighestDepth());  
audio_mc.attachAudio(active_mic);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[Microphone](#), [play \(NetStream.play 方法\)](#), [Sound](#), [attachVideo \(Video.attachVideo 方法\)](#)

## attachBitmap (MovieClip.attachBitmap 方法)

```
public attachBitmap(bmp:BitmapData, depth:Number, [pixelSnapping:String],  
    [smoothing:Boolean]) : Void
```

将位图图像附加到影片剪辑。

将位图附加到影片剪辑后，将建立从影片剪辑到位图对象的引用。附加位图时，可以指定 `pixelSnapping` 和 `smoothing` 参数以影响位图的外观。

将对象添加到影片剪辑后，它就不是可访问的对象了。对 `depth`、`pixelSnapping` 和 `smoothing` 参数的设置只能在 `attachBitmap()` 方法调用期间进行，且以后无法更改。

首先使用 `createEmptyMovieClip()` 创建空的影片剪辑，然后使用 `attachBitmap()` 方法。这样，您就可以将转换应用于影片剪辑以转换位图（例如，通过影片剪辑的 `matrix` 属性）。

采用像素对齐模式会将位图位置强制为最接近的完整像素值，而不是定位在部分像素上。有以下三种像素对齐模式：

- **Auto** 模式，只要位图没有被拉伸或旋转，它就执行像素对齐。
- **Always** 模式，始终执行像素对齐，而不管是否拉伸和旋转。
- **Never** 模式，关闭影片剪辑的像素对齐。

在对图像缩放时，平滑处理模式会影响图像的外观。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**bmp:**`flash.display.BitmapData` - 透明的或不透明的位图图像。

**depth:**`Number` - 一个整数，它指定应该放置位图图像的影片剪辑内的深度级别。

**pixelSnapping:String** [ 可选 ] - 像素对齐模式包括 `auto`、`always` 和 `never`。默认模式为 `auto`。

**smoothing:Boolean** [ 可选 ] - 平滑处理模式为 `true`（表示启用）或 `false`（表示禁用）。默认模式为禁用。

## 示例

下面的示例将一个非常基本的位图附加到影片剪辑：

```
import flash.display.*;
```

```
this.createEmptyMovieClip("bmp1", this.getNextHighestDepth());  
var bmpData1:BitmapData = new BitmapData(200, 200, false, 0xaa3344);  
bmp1.attachBitmap(bmpData1, 2, "auto", true);
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## attachMovie（MovieClip.attachMovie 方法）

```
public attachMovie(id:String, name:String, depth:Number,  
    [initObject:Object]) : MovieClip
```

从库中取得一个元件并将其附加到影片剪辑中。使用 `MovieClip.removeMovieClip()` 或 `MovieClip.unloadMovie()` 删除通过 `attachMovie()` 方法附加的 **SWF** 文件。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

**可用性：** **ActionScript 1.0**； **Flash Player 5**

## 参数

**id:String** - 库中要附加到舞台上某影片剪辑的影片剪辑元件的链接名称。这是在“链接属性”对话框中的“标识符”字段中输入的名称。

**name:String** - 附加到该影片剪辑的影片剪辑实例的唯一名称。

**depth:Number** - 一个整数，指定 **SWF** 文件所放位置的深度级别。

**initObject:Object** [ 可选 ] - （**Flash Player 6** 和更高版本支持）包含要用来填充新附加的影片剪辑的属性的对象。此参数使动态创建的影片剪辑能够接收剪辑参数。如果 `initObject` 不是对象，则忽略它。`initObject` 的所有属性都已复制到新实例中。使用 `initObject` 指定的属性对于构造函数是可用的。

## 返回

**MovieClip** - 对新创建的实例的引用。

## 示例

下面的示例将链接标识符为 `circle` 的元件附加到位于 **SWF** 文件舞台上的影片剪辑实例中。

```
this.attachMovie("circle", "circle1_mc", this.getNextHighestDepth());
this.attachMovie("circle", "circle2_mc", this.getNextHighestDepth(),
    {_x:100, _y:100});
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[removeMovieClip](#) ([MovieClip.removeMovieClip](#) 方法), [unloadMovie](#) ([MovieClip.unloadMovie](#) 方法), [removeMovieClip](#) 函数

## beginBitmapFill (MovieClip.beginBitmapFill 方法)

```
public beginBitmapFill(bmp:BitmapData, [matrix:Matrix], [repeat:Boolean],
    [smoothing:Boolean]) : Void
```

用位图图像填充绘画区域。可以重复或平铺位图以填充区域。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**bmp:**flash.display.BitmapData - 透明的或不透明的位图图像。

**matrix:**flash.geom.Matrix [可选] - 一个 **matrix** 对象 (属于 **flash.geom.Matrix** 类), 您可以使用它在位图上定义转换。例如, 您可以使用以下矩阵将位图旋转 45 度 ( $\pi/4$  弧度):

```
var matrix = new flash.geom.Matrix();
matrix.rotate(Math.PI/4);
```

**repeat:**Boolean [optional] - 如果为 `true`, 则位图图像按平铺模式重复。如果为 `false`, 位图图像不会重复, 并且位图边缘将用于所有扩展出位图的填充区域。

例如, 请考虑下列位图 (20 x 20 像素的棋盘图案):



当 `repeat` 设置为 `true` 时 (如下例所示), 位图填充将重复位图:



repeat 设置为 false 时，位图填充将在位图外的填充区域中使用边缘像素：



**smoothing:** Boolean [可选] - 如果为 false，则使用最近邻点算法来呈现放大的位图图像，而且该图像看起来是像素化的。如果为 true，则使用双线性算法来呈现放大的位图图像。使用最近邻点算法呈现通常会快得多。此参数的默认值为 false。

## 示例

下面的代码定义一个简单位图，然后以平铺该位图的方式使用 beginBitmapFill() 填充影片剪辑：

```
import flash.display.*;
import flash.geom.*;

var bmpd:BitmapData = new BitmapData(20,20);
var rect1:Rectangle = new Rectangle(0,0,10,10);
var rect2:Rectangle = new Rectangle(0, 10, 10, 20);
var rect3:Rectangle = new Rectangle(10, 0, 20, 10);
var rect4:Rectangle = new Rectangle(10, 10, 20, 20);
bmpd.fillRect(rect1, 0xAA0000FF);
bmpd.fillRect(rect2, 0xAA00FF00);
bmpd.fillRect(rect3, 0xAAFF0000);
bmpd.fillRect(rect4, 0xAA999999);

this.createEmptyMovieClip("bmp_fill_mc", this.getNextHighestDepth());
with (bmp_fill_mc) {
    matrix = new Matrix();
    matrix.rotate(Math.PI/8);
    repeat = true;
    smoothing = true;
    beginBitmapFill(bmpd, matrix, repeat, smoothing);
    moveTo(0, 0);
    lineTo(0, 60);
    lineTo(60, 60);
    lineTo(60, 0);
    lineTo(0, 0);
    endFill();
}

bmp_fill_mc._xscale = 200;
bmp_fill_mc._yscale = 200;
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 DepthManager 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

## beginFill (MovieClip.beginFill 方法)

`public beginFill(rgb:Number, [alpha:Number]) : Void`

指示新的绘画路径的开始。如果存在一个开放路径（即如果当前的绘画位置不等于 `MovieClip.moveTo()` 方法中指定的上一个位置），并且该路径具有与其关联的填充，则用线条闭合该路径，然后进行填充。这类似于调用 `MovieClip.endFill()` 方法时的情形。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**rgb**:Number - 十六进制颜色值：例如，红色为 `0xFF0000`，蓝色为 `0x0000FF`。如果未提供或未定义该值，则不创建填充。

**alpha**:Number [可选] - 一个 0 到 100 之间的整数，指定填充的 Alpha 值。如果未提供该值，则使用 100（纯色）。如果该值小于 0，则 Flash 使用 0。如果该值大于 100，则 Flash 使用 100。

### 示例

以下示例在舞台上创建带有红色填充的正方形：

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
square_mc.beginFill(0xFF0000);
square_mc.moveTo(10, 10);
square_mc.lineTo(100, 10);
square_mc.lineTo(100, 100);
square_mc.lineTo(10, 100);
square_mc.lineTo(10, 10);
square_mc.endFill();
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 Flash Player 7 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

一个示例也包含在 `Samples\ActionScript\DrawingAPI` 中的 `drawingapi.fla` 文件中。下面的列表指定到此文件夹的典型路径：

- Windows: \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\
- Macintosh: HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/

另请参见

[moveTo \(MovieClip.moveTo 方法\)](#), [endFill \(MovieClip.endFill 方法\)](#),  
[beginGradientFill \(MovieClip.beginGradientFill 方法\)](#)

# beginGradientFill (MovieClip.beginGradientFill 方法)

```
public beginGradientFill(fillType:String, colors:Array, alphas:Array,
    ratios:Array, matrix:Object, [spreadMethod:String],
    [interpolationMethod:String], [focalPointRatio:Number]) : Void
```

指示新的绘画路径的开始。如果第一个参数为 undefined，或者未传递任何参数，则该路径无填充。如果存在一个开放路径（即如果当前的绘画位置不等于 MovieClip.moveTo() 方法中指定的上一个位置），并且该路径具有与其关联的填充，则用线条闭合该路径，然后进行填充。这类似于调用 MovieClip.endFill() 方法时的情形。

如果存在下列任意一种情况，则此方法将失败：

- colors、alphas 和 ratios 参数中的项数不相等。
- fillType 参数不是 "linear" 或 "radial"。
- matrix 参数的对象中的任意一个字段缺少或无效。

您可以通过创建子类来扩展 MovieClip 类的方法和事件处理函数。

可用性：ActionScript 1.0 ； Flash Player 6

## 参数




**fillType:String** - 有效值为字符串 "linear" 和字符串 "radial"。

**colors:Array** - 用于渐变色的 RGB 十六进制颜色值的数组；例如，红色为 0xFF0000，蓝色为 0x0000FF。可以至多指定 15 种颜色。对于每种颜色，请确保在 alphas 和 ratios 参数中指定对应值。

**alphas:Array** - colors 数组中对应颜色的 Alpha 值数组；有效值为 0 到 100。如果值小于 0，则 Flash 使用 0。如果值大于 100，则 Flash 使用 100。

**ratios:Array** - 颜色分布比率数组；有效值为 0 到 255。该值定义颜色采样率为 100% 之处的宽度百分比。为 colors 参数中的每个值指定一个值。

例如，对于包含蓝和绿两种颜色的线性渐变，下图显示了基于不同 ratios 数组值的颜色配比：

ratios	渐变
[0, 127]	
[0, 255]	
[127, 255]	

数组中的值必须持续增加；例如，[0, 63, 127, 190, 255]。

**matrix:Object** - 可以是任意以下三种形式的转换矩阵：

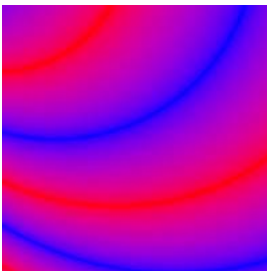
- **matrix** 对象（仅 Flash Player 8 和更高版本支持它），如 **flash.geom.Matrix** 类定义的那样。**flash.geom.Matrix** 类包括 **createGradientBox()** 方法，通过该方法可以方便地设置矩阵，以便用于 **MovieClip** 类的 **beginGradientFill()** 方法。对于 Flash Player 8 和更高版本，**Macromedia** 建议使用此形式的矩阵。

下列示例将 **beginGradientFill()** 方法与此类型的 **matrix** 参数一起使用：

```
import flash.geom.*

this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());
with (gradient_mc)
{
    colors = [0xFF0000, 0x0000FF];
    fillType = "radial";
    alphas = [100, 100];
    ratios = [0, 0xFF];
    spreadMethod = "reflect";
    interpolationMethod = "linearRGB";
    focalPointRatio = 0.9;
    matrix = new Matrix();
    matrix.createGradientBox(100, 100, Math.PI, 0, 0);
    beginGradientFill(fillType, colors, alphas, ratios, matrix,
        spreadMethod, interpolationMethod, focalPointRatio);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

此代码在屏幕上绘制下列图像：





- 可以使用属性 a、b、c、d、e、f、g、h 和 i，这些属性可用于描述下列格式的 3 x 3 矩阵：

```
a b c
d e f
g h i
```

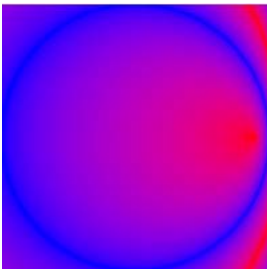
**提醒**

对于 Flash Player 8 及更高版本，Macromedia 建议以 flash.geom.Matrix 对象的形式（如此列表的第一个项目所述）定义矩阵参数。

- 下列示例将 beginGradientFill() 方法与此类型的 matrix 参数一起使用：

```
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());
with (gradient_mc)
{
    colors = [0xFF0000, 0x0000FF];
    fillType = "radial";
    alphas = [100, 100];
    ratios = [0, 0xFF];
    spreadMethod = "reflect";
    interpolationMethod = "linearRGB";
    focalPointRatio = 0.9;
    matrix = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};
    beginGradientFill(fillType, colors, alphas, ratios, matrix,
        spreadMethod,
        interpolationMethod, focalPointRatio);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

此代码在屏幕上绘制下列图像：



- 具有下列属性的对象：matrixType、x、y、w、h、r。

- 这些属性表示下列含义：matrixType 是字符串 "box"，x 是渐变的左上角相对于父级剪辑注册点的水平位置，y 是渐变的左上角相对于父级剪辑注册点的垂直位置，w 是渐变的宽度，h 是渐变的高度，r 是渐变的旋转程度（以弧度为单位）。

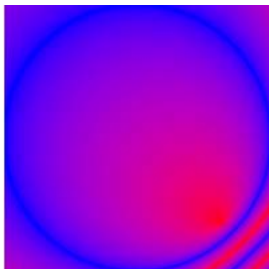


对于 Flash Player 8 及更高版本，Macromedia 建议以 flash.geom.Matrix 对象的形式（如此列表的第一个项目所述）定义 matrix 参数。

- 下列示例将 beginGradientFill() 方法与此类型的 matrix 参数一起使用：

```
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());
with (gradient_mc)
{
    colors = [0xFF0000, 0x0000FF];
    fillType = "radial";
    alphas = [100, 100];
    ratios = [0, 0xFF];
    spreadMethod = "reflect";
    interpolationMethod = "linearRGB";
    focalPointRatio = 0.9;
    matrix = {matrixType:"box", x:100, y:100, w:200, h:200, r:(45/
180)*Math.PI};
    beginGradientFill(fillType, colors, alphas, ratios, matrix,
        spreadMethod, interpolationMethod, focalPointRatio);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

此代码在屏幕上绘制下列图像：

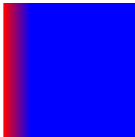


`spreadMethod:String` [ 可选 ] - 在 Flash Player 8 中添加。可以是 “pad”、 “reflect” 或 “repeat”，它控制渐变填充的模式。默认值为 “pad”。

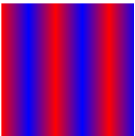
例如，请考虑两种颜色之间的简单线性渐变：

```
import flash.geom.*;
var fillType:String = "linear"
var colors:Array = [0xFF0000, 0x0000FF];
var alphas:Array = [100, 100];
var ratios:Array = [0x00, 0xFF];
var matrix:Matrix = new Matrix();
matrix.createGradientBox(20, 20, 0, 0, 0);
var spreadMethod:String = "pad";
this.beginGradientFill(fillType, colors, alphas, ratios, matrix,
    spreadMethod);
this.moveTo(0, 0);
this.lineTo(0, 100);
this.lineTo(100, 100);
this.lineTo(100, 0);
this.lineTo(0, 0);
this.endFill();
```

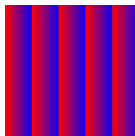
此示例将 “pad” 用于 `spread` 方法，因此渐变填充与如下所示类似：



如果将 “reflect” 用于 `spread` 方法，则渐变填充将与如下所示类似：

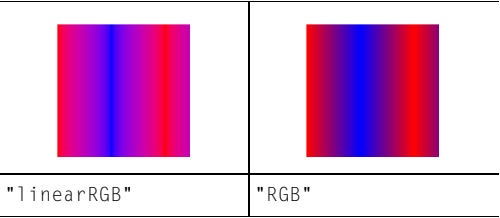


如果将 “repeat” 用于 `spread` 方法，则渐变填充将与如下所示类似：

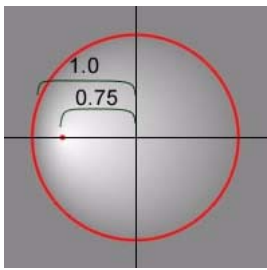


`interpolationMethod:String` [ 可选 ] - 在 **Flash Player 8** 中添加。可以是 “**RGB**” 或 “**linearRGB**”。如果使用 “**linearRGB**”，则在渐变中以线性方式分布颜色。默认值为 “**RGB**”。

例如，请考虑两种颜色之间的简单线性渐变（`spreadMethod` 参数设置为 “**reflect**”）。不同的 **interpolation** 方法对外观的影响如下所示：



`focalPointRatio:Number` [optional] - 在 **Flash Player 8** 中添加。一个数字，控制渐变焦点的位置。值 **0** 表示焦点位于中心。值 **1** 表示焦点位于渐变圆的一条边界上。值 **-1** 表示焦点位于渐变圆的另一条边界上。小于 **-1** 或大于 **1** 的值将被舍入为 **-1** 或 **1**。例如，下列图像显示设置为 **0.75** 的 `focalPointRatio`：



### 示例

下面的代码创建球形阴影效果：

```
import flash.geom.*
this.createEmptyMovieClip("gradient_mc", this.getNextHighestDepth());
with (gradient_mc)
{
    fillType = "radial"
    colors = [0x000000, 0xFFFFFF];
    alphas = [50, 90];
    ratios = [0, 0xFF];
    spreadMethod = "pad";
    interpolationMethod = "RGB";
    focalPointRatio = 0.3;
    matrix = new Matrix();
    matrix.createGradientBox(100, 100, 0, 0, 0);
    beginGradientFill(fillType, colors, alphas, ratios, matrix,
```

```

        spreadMethod, interpolationMethod, focalPointRatio);
moveTo(0, 0);
lineTo(0, 100);
lineTo(100, 100);
lineTo(100, 0);
lineTo(0, 0);
endFill();
}

```

这将绘制下列图像（图像缩小了 50%）：



如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[createGradientBox \(Matrix.createGradientBox 方法\)](#), [beginFill \(MovieClip.beginFill 方法\)](#), [endFill \(MovieClip.endFill 方法\)](#), [lineStyle \(MovieClip.lineStyle 方法\)](#), [lineTo \(MovieClip.lineTo 方法\)](#), [moveTo \(MovieClip.moveTo 方法\)](#)

## blendMode（MovieClip.blendMode 属性）

```
public blendMode : Object
```

此影片剪辑的混合模式。当影片剪辑位于屏幕上另一对象的上一层时，混合模式将影响影片剪辑的外观。

**Flash Player** 在影片剪辑的每一个像素上应用 `blendMode` 属性。每个像素都由三种原色（红色、绿色和蓝色）组成，每种原色的值介于 `0x00` 和 `0xFF` 之间。**Flash Player** 将影片剪辑中一个像素的每种构成颜色与背景中像素的对应颜色进行比较。例如，如果 `blendMode` 设置为 "lighten"，则 **Flash Player** 将按钮的红色值与背景的红色值进行比较，然后使用两者中较亮的一种颜色作为显示颜色的红色成份的值。

下表将对 blendMode 设置进行说明。若要设置 blendMode 属性，您可以使用从 1 到 14 的整数或使用字符串。表中的插图表示：blendMode 值应用于交叠于屏幕对象 (1) 之上的圆形影片剪辑 (2)。



整数值	字符串值	插图	说明
1	"normal"		影片剪辑显示在背景之前。影片剪辑的像素值将覆盖背景的像素值。在影片剪辑为透明的区域，背景是可见的。
2	"layer"		强制为影片剪辑的预构成创建临时缓冲区。如果影片剪辑中有多个子对象，并且为子级选择了 "normal" 以外的其它 blendMode 设置，则该操作将自动完成。
3	"multiply"		将影片剪辑构成颜色的值与背景颜色的值相乘，然后通过除以 0xFF 进行标准化，从而得到较暗的颜色。此设置通常用于阴影和深度效果。 例如，如果影片剪辑中一个像素的某个原色（例如红色）与背景中的对应的像素的颜色的值均为 0x88，则相乘结果为 0x4840。除以 0xFF 将得到该原色的值 0x48，该值是比影片剪辑或背景颜色暗的阴影。
4	"screen"		将影片剪辑颜色的补色（反转色）与背景颜色的补色相乘，从而得到漂白效果。此设置通常用于产生加亮效果或用来删除影片剪辑的黑色区域。

整数值	字符串值	插图	说明
5	"lighten"		<p>选择影片剪辑和背景中相对较亮的原色值（相对较大的值）。此设置通常用于叠加类型。</p> <p>例如，如果影片剪辑的某个像素的 RGB 值为 0xFFCC33，背景像素的 RGB 值为 0xDDF800，则显示像素的结果 RGB 值为 0xFFFF833（因为 0xFF &gt; 0xDD，0xCC &lt; 0xF8 且 0x33 &gt; 0x00 = 33）。</p>
6	"darken"		<p>选择影片剪辑与背景中相对较暗的原色值（相对较小的值）。此设置通常用于叠加类型。</p> <p>例如，如果影片剪辑的某个像素的 RGB 值为 0xFFCC33，背景像素的 RGB 值为 0xDDF800，则显示像素的结果 RGB 值为 0xDDCC00（因为 0xFF &gt; 0xDD，0xCC &lt; 0xF8 且 0x33 &gt; 0x00 = 33）。</p>
7	"difference"		<p>将影片剪辑与背景的原色进行比较，然后从较亮的原色减去较暗的原色。此设置通常用于得到更明亮的颜色。</p> <p>例如，如果影片剪辑的某个像素的 RGB 值为 0xFFCC33，背景像素的 RGB 值为 0xDDF800，则显示像素的结果 RGB 值为 0x222C33（因为 0xFF - 0xDD = 0x22，0xF8 - 0xCC = 0x2C 且 0x33 - 0x00 = 0x33）。</p>
8	"add"		<p>将影片剪辑与背景的原色值相加，并应用 0xFF 的上限。此设置通常用于使两个对象间的加亮溶解产生动画效果。</p> <p>例如，如果影片剪辑的某个像素的 RGB 值为 0xAAA633，背景像素的 RGB 值为 0xDD2200，则显示像素的结果 RGB 值为 0xFFC833（因为 0xAA + 0xDD &gt; 0xFF，0xA6 + 0x22 = 0xC8 且 0x33 + 0x00 = 0x33）。</p>

整数值	字符串值	插图	说明
9	"subtract"		从背景原色的值中减去影片剪辑中原色的值，然后应用下限值 0。此设置通常用于使两个对象间的变暗溶解产生动画效果。例如，如果影片剪辑的某个像素的 RGB 值为 0xAA2233，背景像素的 RGB 值为 0xDDA600，则显示像素的结果 RGB 值为 0x338400（因为 0xDD - 0xAA = 0x33，0xA6 - 0x22 = 0x84 且 0x00 - 0x33 < 0x00）。
10	"invert"		反转背景。
11	"alpha"		将影片剪辑每个像素的 Alpha 值应用于背景。这要求对父级影片剪辑应用 "layer" blendMode 设置。例如，在此插图中，父级影片剪辑是一个白色背景，它具有 blendMode = "layer"。
12	"erase"		根据影片剪辑的 Alpha 值擦除背景。这要求对父级影片剪辑应用 "layer" blendMode。例如，在此插图中，父级影片剪辑是一个白色背景，它具有 blendMode = "layer"。
13	"overlay"		根据背景的暗度调整每个位图的颜色。如果背景的亮度高于 50% 灰度，影片剪辑和背景颜色叠加后，将显示较亮的颜色。如果背景的亮度低于 50% 灰度，颜色将相乘，产生较暗的颜色。此设置通常用于获得阴影效果。
14	"hardlight"		根据影片剪辑的暗度调整每个位图的颜色。如果影片剪辑的亮度高于 50% 灰度，影片剪辑和背景颜色叠加后，将显示较亮的颜色。如果影片剪辑的亮度低于 50% 灰度，颜色将相乘，产生较暗的颜色。此设置通常用于获得阴影效果。



如果尝试将 `blendMode` 属性设置为任何其它值，则 **Flash Player** 会将该属性设置为 `"normal"`。

但是，如果将该属性设置为一个整数，则 **Flash Player** 会将此值转换为对应的字符串形式：

```
this.createEmptyMovieClip("mclip", this.getNextHighestDepth());
mclip.blendMode = 8;
trace (mclip.blendMode) // add
```

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 示例

下面的示例设置两个具有渐变填充的影片剪辑，并且每秒钟会更改前景中某个影片剪辑的混合模式。为使 `"alpha"` 混合模式显示出效果，`mc2` 影片剪辑的渐变包括一定范围的 **Alpha** 比例，并且 `"layer"` 混合模式要应用于父级影片剪辑 (`this.blendMode="layer"`)。

```
this.createEmptyMovieClip("mc1", this.getNextHighestDepth());
this.createEmptyMovieClip("mc2", this.getNextHighestDepth());
this.blendMode="layer";
this.createTextField("blendLabel", this.getNextHighestDepth(), 50, 150,
100, 100)

fillClip(mc1, 0x00AA00, 0x22FFFF, 100, 100)
fillClip(mc2, 0xFF0000, 0x2211FF, 100, 50)
mc2._x = 33;
mc2._y = 33;

var blendModeIndex = 0;

setInterval(changeBlendMode, 1000);
function changeBlendMode()
{
    mc2.blendMode = blendModeIndex % 14 + 1 ;
    // values 1 - 14
    blendLabel.text = (blendModeIndex% 14 + 1) + ": " + mc2.blendMode;
    blendModeIndex++;
}

function fillClip(mc:MovieClip, color1:Number, color2:Number,
    alpha1:Number, alpha2: Number)
{
    matrix = {a:100, b:0, c:0, d:0, e:100, f:0, g:50, h:20, i:1};
    mc.beginGradientFill("linear", [color1, color2], [alpha1, alpha2], [0,
0xFF], matrix);
    mc.lineStyle(8,0x888888,100)
    mc.moveTo(0, 0);
    mc.lineTo(0, 100);
    mc.lineTo(100, 100);
}
```

```
mc.lineTo(100, 0);  
mc.lineTo(0, 0);  
mc.endFill();  
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代前一示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## cacheAsBitmap（MovieClip.cacheAsBitmap 属性）

```
public cacheAsBitmap : Boolean
```

如果设置为 `true`，则 **Flash Player** 将缓存影片剪辑的内部位图表示。这可以提高包含复杂矢量内容的影片剪辑的性能。

具有已缓存位图的影片剪辑的所有矢量数据都会被绘制到位图而不是主舞台。然后，将位图复制到主舞台，作为对齐到最接近像素边界的未拉伸、未旋转的像素。对于父级对象，像素按一对一进行映射。如果位图的边界发生更改，则将重新创建位图而不会拉伸它。

除非将 `cacheAsBitmap` 属性设置为 `true`，否则不会创建内部位图。

将影片剪辑的 `cacheAsBitmap` 属性设置为 `true` 后，呈现不会更改，但是，影片剪辑将自动执行像素对齐。动画速度可能会大大加快，具体取决于矢量内容的复杂性。

无论何时将滤镜应用于影片剪辑（当它的 `filter` 数组不为空时），`cacheAsBitmap` 属性将自动设置为 `true`。如果影片剪辑具有应用到它的滤镜，该影片剪辑的 `cacheAsBitmap` 将报告为 `true`，即使您将该属性设置为 `false`。如果清除影片剪辑的所有滤镜，则 `cacheAsBitmap` 设置将更改为它上次的“设置”。

在下列情况下，即使将 `cacheAsBitmap` 属性设置为 `true`，影片剪辑也不使用位图，而是从矢量数据呈现：

- 位图太大：在任一方向上大于 2880 像素。
- 位图内存分配失败（由于内存不足的错误）。

最好将 `cacheAsBitmap` 属性与主要具有静态内容且不频繁缩放和旋转的影片剪辑一起使用。对于这样的影片剪辑，在转换影片剪辑时（更改其 `x` 和 `y` 位置时），`cacheAsBitmap` 可以提高性能。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下列示例向影片剪辑实例应用投影。然后它跟踪 `cacheAsBitmap` 属性的值，应用滤镜时该值为 `true`。

```
import flash.filters.DropShadowFilter;

var container:MovieClip = setUpShape();
trace(container.cacheAsBitmap); // false
var dropShadow:DropShadowFilter = new DropShadowFilter(6, 45, 0x000000, 50,
    5, 5, 1, 2, false, false, false);
container.filters = new Array(dropShadow);
trace(container.cacheAsBitmap); // true

function setUpShape():MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip("container",
        this.getNextHighestDepth());
    mc._x = 10;
    mc._y = 10;
    var w:Number = 50;
    var h:Number = 50;
    mc.beginFill(0xFFCC00);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    return mc;
}
```

## 另请参见

[opaqueBackground](#) ([MovieClip.opaqueBackground](#) 属性), [cacheAsBitmap](#) ([MovieClip.cacheAsBitmap](#) 属性)

## clear (MovieClip.clear 方法)

```
public clear() : Void
```

删除使用影片剪辑绘画方法（包括用 `MovieClip.lineStyle()` 指定的线条样式）在运行时创建的所有图形。创作过程中（用 **Flash** 绘画工具）手动绘制的形状和线条不受影响。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例在舞台上绘制一个框。当用户单击框图形时，它会从舞台上删除该图形。

```
this.createEmptyMovieClip("box_mc", this.getNextHighestDepth());
box_mc.onRelease = function() {
    this.clear();
}
```

```

};
drawBox(box_mc, 10, 10, 320, 240);
function drawBox(mc:MovieClip, x:Number, y:Number, w:Number, h:Number):Void
{
    mc.lineStyle(0);
    mc.beginFill(0xEEEEEE);
    mc.moveTo(x, y);
    mc.lineTo(x+w, y);
    mc.lineTo(x+w, y+h);
    mc.lineTo(x, y+h);
    mc.lineTo(x, y);
    mc.endFill();
}

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

一个示例也包含在 **Samples\ActionScript\DrawingAPI** 文件夹的 **drawingapi.fla** 文件夹中。下面的列表指定到此文件夹的典型路径：

- Windows: \Program Files\Macromedia\Fash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript

另请参见

[lineStyle \(MovieClip.lineStyle 方法\)](#)

## createEmptyMovieClip (MovieClip.createEmptyMovieClip 方法)

```
public createEmptyMovieClip(name:String, depth:Number) : MovieClip
```

创建一个空影片剪辑作为现有影片剪辑的子级。此方法的行为类似于 `attachMovie()` 方法，但是不必为新的影片剪辑提供外部链接标识符。新创建的空影片剪辑的注册点为左上角。如果缺少任意一个参数，则此方法将失败。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**name:String** - 标识新影片剪辑的实例名称的字符串。

**depth:Number** - 指定新影片剪辑的深度的整数。

返回

MovieClip - 对新创建的影片剪辑的引用。

示例

下列示例创建名为 container 的空 MovieClip，在其中新建 TextField，然后设置新的 TextField.text 属性。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getNextHighestDepth());
var label:TextField = container.createTextField("label", 1, 0, 0, 150, 20);
label.text = "Hello World";
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 Flash Player 7 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

另请参见

[attachMovie \(MovieClip.attachMovie 方法\)](#)

## createTextField (MovieClip.createTextField 方法)

```
public createTextField(instanceName:String, depth:Number, x:Number,
    y:Number, width:Number, height:Number) : TextField
```

创建一个新的空文本字段作为在其上调用此方法的影片剪辑的子级。可以使用

createTextField() 方法在 SWF 文件播放时创建文本字段。depth 参数确定新文本字段在影片剪辑中的深度级别 (z-顺序位置)。每个深度级别只能包含一个对象。如果您在已具有文本字段的深度上创建一个新文本字段，则新文本字段将替换现有文本字段。若要避免覆盖现有文本字段，可使用 MovieClip.getInstanceAtDepth() 方法确定特定深度是否已被占用，或者使用 MovieClip.getNextHighestDepth() 方法确定未占用的最高深度。文本字段位于 (x, y)，尺寸为 width 乘 height。参数 x 和 y 相对于容器影片剪辑；这些参数对应于文本字段的 \_x 和 \_y 属性。width 和 height 参数对应于文本字段的 \_width 和 \_height 属性。

文本字段的默认属性如下所示：

```
type = "dynamic"
border = false
background = false
password = false
multiline = false
html = false
embedFonts = false
selectable = true
wordWrap = false
mouseWheelEnabled = true
condenseWhite = false
```

```
restrict = null  
variable = null  
maxChars = null  
styleSheet = undefined  
tabInded = undefined
```

用 `createTextField()` 创建的文本字段获得以下默认 **TextFormat** 对象设置：

```
font = "Times New Roman" // "Times" on Mac OS  
size = 12  
color = 0x000000  
bold = false  
italic = false  
underline = false  
url = ""  
target = ""  
align = "left"  
leftMargin = 0  
rightMargin = 0  
indent = 0  
leading = 0  
blockIndent = 0  
bullet = false  
display = block  
tabStops = [] // (empty array)
```

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**instanceName:String** - 一个字符串，标识新文本字段的实例名称。

**depth:Number** - 一个正整数，指定新文本字段的深度。

**x:Number** - 一个整数，指定新文本字段的 x 坐标。

**y:Number** - 一个整数，指定新文本字段的 y 坐标。

**width:Number** - 一个正整数，指定新文本字段的宽度。

**height:Number** - 一个正整数，指定新文本字段的高度。

## 返回

**TextField** - **Flash Player 8** 返回对所创建的 **TextField** 对象的引用。**Flash Player 8** 之前的版本返回 `void`。

## 示例

下面的示例创建一个宽 300，高 100 的文本字段，其 x 坐标为 100，y 坐标为 100，该文本字段没有边框，文本为红色并带下划线：

```
this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
var my_fmt:TextFormat = new TextFormat();
my_fmt.color = 0xFF0000;
my_fmt.underline = true;
my_txt.text = "This is my first test field object text.";
my_txt.setTextFormat(my_fmt);
```

一个示例也包含在 `Samples\ActionScriptAnimation` 文件夹的 `animation.fla` 文件中。下面的列表指定到此文件夹的典型路径：

- Windows: \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\
- Macintosh: HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/

## 另请参见

[getInstanceAtDepth](#) ([MovieClip.getInstanceAtDepth 方法](#))，[getNextHighestDepth](#) ([MovieClip.getNextHighestDepth 方法](#))，[TextFormat](#)

## \_currentframe (MovieClip.\_currentframe 属性)

```
public _currentframe : Number [read-only]
```

返回指定帧的编号，该帧中的播放头位于影片剪辑的时间轴中。

可用性：ActionScript 1.0；Flash Player 4

## 示例

下面的示例使用 `_currentframe` 属性指示 `actionClip_mc` 影片剪辑的播放头从其当前位置前进 5 帧：

```
actionClip_mc.gotoAndStop(actionClip_mc._currentframe + 5);
```

## curveTo (MovieClip.curveTo 方法)

```
public curveTo(controlX:Number, controlY:Number, anchorX:Number,  
    anchorY:Number) : Void
```

通过由 (controlX, controlY) 指定的控制点，使用当前线条样式绘制一条曲线，该曲线从当前绘画位置到 (anchorX, anchorY)。当前绘画位置随后设置为 (anchorX, anchorY)。如果正在其中绘制的影片剪辑包含用 **Flash** 绘画工具创建的内容，则调用 curveTo() 方法将在该内容下面进行绘制。如果在调用 moveTo() 方法之前调用 curveTo() 方法，当前绘画位置将设置为默认值 (0,0)。如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**controlX:Number** - 一个整数，指定控制点相对于父级影片剪辑注册点的水平位置。

**controlY:Number** - 一个整数，指定控制点相对于父级影片剪辑注册点的垂直位置。

**anchorX:Number** - 一个整数，指定下一个锚点相对于父级影片剪辑注册点的水平位置。

**anchorY:Number** - 一个整数，指定下一个锚点相对于父级影片剪辑注册点的垂直位置。

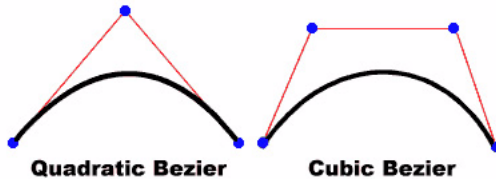
### 示例

下列示例绘制一条近似圆形的曲线，该曲线具有纯蓝色的极细笔触和纯红色的填充：

```
this.createEmptyMovieClip("circle_mc", 1);  
with (circle_mc) {  
    lineStyle(0, 0x0000FF, 100);  
    beginFill(0xFF0000);  
    moveTo(0, 100);  
    curveTo(0,200,100,200);  
    curveTo(200,200,200,100);  
    curveTo(200,0,100,0);  
    curveTo(0,0,0,100);  
    endFill();  
}
```



在此示例中绘制的曲线是二次贝塞尔曲线。二次贝塞尔曲线包含两个锚点和一个控制点。该曲线内插这两个锚点，并向控制点弯曲。



下列脚本使用 `curveTo()` 方法和 `Math` 类创建圆形：

```
this.createEmptyMovieClip("circle2_mc", 2);
circle2_mc.lineStyle(0, 0x000000);
drawCircle(circle2_mc, 100, 100, 100);
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
    mc.moveTo(x+r, y);
    mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
    Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
    mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x,
    -Math.sin(Math.PI/4)*r+y);
    mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
    mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x,
    -Math.sin(Math.PI/4)*r+y);
    mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

一个示例也包含在 `Samples\ActionScript\DrawingAPI` 中的 `drawingapi.fla` 文件中。下面的列表指定到此文件夹的典型路径：

- Windows: \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\
- Macintosh: HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/

另请参见

[beginFill](#) ([MovieClip.beginFill](#) 方法), [createEmptyMovieClip](#) ([MovieClip.createEmptyMovieClip](#) 方法), [endFill](#) ([MovieClip.endFill](#) 方法), [lineStyle](#) ([MovieClip.lineStyle](#) 方法), [lineTo](#) ([MovieClip.lineTo](#) 方法), [moveTo](#) ([MovieClip.moveTo](#) 方法), [Math](#)

## \_droptarget (MovieClip.\_droptarget 属性)

public \_droptarget : String [read-only]

返回在其上放置此影片剪辑的影片剪辑实例的绝对路径，以斜杠语法记号表示。\_droptarget 属性始终返回以斜杠 (/) 开始的路径。若要将实例的 \_droptarget 属性与引用进行比较，请使用 eval() 函数将返回值从以斜杠语法表示转换为点语法表示的引用。



如果您正使用 ActionScript 2.0，则必须执行此转换，因为 ActionScript 2.0 不支持斜杠语法。

可用性: ActionScript 1.0 ; Flash Player 4

### 示例

下面的示例计算 garbage\_mc 影片剪辑实例的 \_droptarget 属性并使用 eval() 将其从斜杠语法转换为点语法表示的引用。然后，将 garbage\_mc 引用与对 trashcan\_mc 影片剪辑实例的引用进行比较。如果两个引用相等，则将 garbage\_mc 的可见性设置为 false。如果它们不相等，则 garbage 实例将重置为其原始位置。

```
origX = garbage_mc._x;
origY = garbage_mc._y;
garbage_mc.onPress = function() {
    this.startDrag();
};
garbage_mc.onRelease = function() {
    this.stopDrag();
    if (eval(this._droptarget) == trashcan_mc) {
        this._visible = false;
    } else {
        this._x = origX;
        this._y = origY;
    }
};
```

### 另请参见

[startDrag \(MovieClip.startDrag 方法\)](#), [stopDrag \(MovieClip.stopDrag 方法\)](#), [eval 函数](#)

## duplicateMovieClip (MovieClip.duplicateMovieClip 方法)

```
public duplicateMovieClip(name:String, depth:Number, [initObject:Object]) :  
    MovieClip
```

在 SWF 文件正在播放时，创建指定影片剪辑的实例。无论调用 duplicateMovieClip() 方法时原始影片剪辑位于哪一帧，所复制的影片剪辑始终从第 1 帧开始播放。父级影片剪辑中的变量不复制到重复的影片剪辑中。在调用父影片剪辑的 duplicateMovieClip() 方法时，由 duplicateMovieClip() 方法创建的子影片剪辑不会被复制。如果删除父级影片剪辑，则重复的影片剪辑也被删除。如果您已经使用 MovieClip.loadMovie() 或 MovieClipLoader 类加载了影片剪辑，则 SWF 文件的内容不被复制。这意味着您无法通过加载 JPEG、GIF、PNG 或 SWF 文件并接着复制影片剪辑来节省带宽。

将此方法与 duplicateMovieClip() 的全局函数版本进行对比。此方法的全局版本需要指定要复制的目标影片剪辑的参数。对于 MovieClip 类版本，这样的参数是不必要的，因为该方法的目标是对其调用该方法的影片剪辑实例。此外，duplicateMovieClip() 的全局版本既不支持 initObject 参数，也不支持对新创建的 MovieClip 实例的引用的返回值。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**name:String** - 已重制的影片剪辑的唯一标识符。

**depth:Number** - 一个唯一整数，指定要放置新影片剪辑的深度。使用深度 -16384 可将新影片剪辑实例放置在创作环境中创建的所有内容之下。介于 -16383 和 -1（含）之间的值是保留供创作环境使用的，不应与此方法一起使用。其余的有效深度值介于 0 和 1048575（含）之间。

**initObject:Object [ 可选 ]** - （Flash Player 6 和更高版本支持。）包含用于填充复制影片剪辑的属性的对象。此参数使动态创建的影片剪辑能够接收剪辑参数。如果 initObject 不是对象，则忽略它。initObject 的所有属性都已复制到新实例中。使用 initObject 指定的属性对于构造函数是可用的。

### 返回

MovieClip - 对复制的影片剪辑的引用（Flash Player 6 和更高版本支持）。

## 示例

下列示例多次复制新创建的 **MovieClip**，并且跟踪每次复制的目标。

```
var container:MovieClip = setUpContainer();
var ln:Number = 10;
var spacer:Number = 1;
var duplicate:MovieClip;
for(var i:Number = 1; i < ln; i++) {
    var newY:Number = i * (container._height + spacer);
    duplicate = container.duplicateMovieClip("clip-" + i, i, {_y:newY});
    trace(duplicate); // _level0.clip-[number]
}

function setUpContainer():MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip("container",
        this.getNextHighestDepth());
    var w:Number = 100;
    var h:Number = 20;
    mc.beginFill(0x333333);
    mc.lineTo(w, 0);
    mc.lineTo(w, h);
    mc.lineTo(0, h);
    mc.lineTo(0, 0);
    mc.endFill();
    return mc;
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[loadMovie](#) ([MovieClip.loadMovie](#) 方法), [removeMovieClip](#) ([MovieClip.removeMovieClip](#) 方法), [duplicateMovieClip](#) 函数

## enabled (MovieClip.enabled 属性)

public enabled : Boolean

一个布尔值，指示影片剪辑是否处于活动状态。enabled 的默认值是 true。如果将 enabled 属性设置为 false，则不再调用影片剪辑的 **callback** 方法和 *on action* 事件处理函数，并禁用 **Over**、**Down** 和 **Up** 帧。enabled 属性不影响影片剪辑的时间轴；如果影片剪辑正在播放，则会继续播放。影片剪辑会继续接收影片剪辑事件（例如 `mouseDown`、`mouseUp`、`keyDown` 和 `keyUp`）。

enabled 属性仅控制影片剪辑的按钮式属性。可以随时更改 enabled 属性；修改后的影片剪辑将被立即启用或禁用。可以从原型对象中读出 enabled 属性。如果将 enabled 属性设置为 false，则该对象将不包含在 **Tab** 键的自动排序中。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下列示例在用户单击 `circle_mc` 影片剪辑时禁用该剪辑：

```
circle_mc.onRelease = function() {  
    trace("disabling the "+this._name+" movie clip.");  
    this.enabled = false;  
};
```

## endFill (MovieClip.endFill 方法)

public endFill() : Void

对从上一次调用 `beginFill()` 或 `beginGradientFill()` 之后存在的直线或曲线应用填充。

**Flash** 使用的是对 `beginFill()` 或 `beginGradientFill()` 的先前调用中指定的填充。如果当前绘画位置不等于 `moveTo()` 方法中指定的上一个位置，而且定义了填充，则用线条闭合该路径，然后进行填充。

可用性：ActionScript 1.0；Flash Player 6

### 示例

以下示例在舞台上创建带有红色填充的正方形：

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());  
square_mc.beginFill(0xFF0000);  
square_mc.moveTo(10, 10);  
square_mc.lineTo(100, 10);  
square_mc.lineTo(100, 100);  
square_mc.lineTo(10, 100);  
square_mc.lineTo(10, 10);  
square_mc.endFill();
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

一个示例也包含在 **Samples\ActionScript\DrawingAPI** 中的 **drawingapi fla** 文件中。下面的列表指定到此文件夹的典型路径：

- **Windows:** \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\
- **Macintosh:** HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/

另请参见

[beginFill \(MovieClip.beginFill 方法\)](#), [beginGradientFill \(MovieClip.beginGradientFill 方法\)](#), [moveTo \(MovieClip.moveTo 方法\)](#)

## filters (MovieClip.filters 属性)

```
public filters : Array
```

一个索引数组，包含当前与影片剪辑相关联的每个过滤器对象。**flash.filters** 包中的多个类定义了可使用的特定滤镜。

在设计时或者在运行时（使用 **ActionScript** 代码），可以在 **Flash** 创作工具中应用滤镜。若要使用 **ActionScript** 应用滤镜，您必须制作整个 `MovieClip.filters` 数组的临时副本，修改临时数组，然后将临时数组的值重新分配给 `MovieClip.filters` 数组。无法直接将新滤镜对象添加到 `MovieClip.filters` 数组。下面的代码对名为 `myMC` 的目标影片剪辑不起作用：

```
myMC.filters[0].push(myDropShadow);
```

若要使用 **ActionScript** 添加滤镜，您必须按照以下步骤操作（假定目标影片剪辑名为 `myMC`）：

- 使用所选滤镜类的构造函数创建一个新的滤镜对象。
- 将 `myMC.filters` 数组的值分配给临时数组，例如一个名为 `myFilters` 的数组。
- 将新的滤镜对象添加到临时数组 `myFilters`。
- 将临时数组的值分配给 `myMC.filters` 数组。

如果 `filters` 数组为空，则无需使用临时数组。相反，您可以直接分配包含一个或多个已创建的滤镜对象的数组。

若要修改现有的滤镜对象（不管它是在设计时创建的还是在运行时创建的），您必须使用修改 `filters` 数组副本的技巧：

- 将 `myMC.filters` 数组的值分配给临时数组，例如一个名为 `myFilters` 的数组。
- 使用临时数组 `myFilters` 修改属性。例如，如果要设置数组中第一个滤镜的品质属性，可以使用以下代码：`myList[0].quality = 1;`

- 将临时数组的值分配给 `myMC.filters` 数组。

若要清除影片剪辑的滤镜，请将 `filters` 设置为空数组 (`[]`)。

在加载时，如果影片剪辑具有关联滤镜，则将它标记为像透明位图那样缓存本身。从此时起，只要影片剪辑具有有效的滤镜列表，播放器就会将影片剪辑缓存为位图。此源位图用作滤镜效果的源图像。每个影片剪辑通常具有两个位图：一个用于原始未过滤的源影片剪辑，另一个用于过滤后的最终图像。呈现时使用最终图像。只要影片剪辑不发生更改，最终图像就不需要更新。

如果正在使用包含多个滤镜的 `filters` 数组，并且需要跟踪分配给每个数组索引的滤镜类型，则可以维护您自己的 `filters` 数组，并使用单独的数据结构跟踪与每个数组索引关联的滤镜类型。没有简单的方法可以确定与每个 `filters` 数组索引关联的滤镜类型。

**可用性：ActionScript 1.0；Flash Player 8**

### 示例

下列示例将投影滤镜添加到名为 `myMC` 的影片剪辑。

```
var myDropFilter = new flash.filters.DropShadowFilter();
var myFilters:Array = myMC.filters;
myFilters.push(myDropFilter);
myMC.filters = myFilters;
```

下列示例将数组中第一个滤镜的 `quality` 设置更改为 `15`（仅当至少一个滤镜对象已经与 `myMC` 影片剪辑关联时，此示例才起作用）：

```
var myList:Array = myMC.filters;
myList[0].quality = 15;
myMC.filters = myList;
```

另请参见

## focusEnabled（MovieClip.focusEnabled 属性）

```
public focusEnabled : Boolean
```

如果值为 `undefined` 或 `false`，则除非影片剪辑是一个按钮，否则它无法获得输入焦点。如果 `focusEnabled` 属性值为 `true`，则即使影片剪辑不是按钮，它也可以获得输入焦点。

**可用性：ActionScript 1.0；Flash Player 6**

### 示例

下列示例将影片剪辑 `my_mc` 的 `focusEnabled` 属性设置为 `false`：

```
my_mc.focusEnabled = false;
```

## `_focusrect` (MovieClip.\_focusrect 属性)

`public _focusrect : Boolean`

一个布尔值，指定当影片剪辑具有键盘焦点时其周围是否有黄色矩形。此属性可覆盖全局 `_focusrect` 属性。影片剪辑实例的 `_focusrect` 属性的默认值为 `null`；这意味着该影片剪辑实例不会覆盖全局 `_focusrect` 属性。如果影片剪辑实例的 `_focusrect` 属性设置为 `true` 或 `false`，则它会覆盖单个影片剪辑实例的全局 `_focusrect` 属性设置。

在 **Flash Player 4** 或 **Flash Player 5 SWF** 文件中，`_focusrect` 属性控制全局 `_focusrect` 属性。它是一个布尔值。在 **Flash Player 6** 和更高版本中，此行为更改为允许基于单个影片剪辑自定义 `_focusrect`。

如果 `_focusrect` 属性设置为 `false`，则将该影片剪辑的键盘导航限制为 **Tab** 键。忽略所有其它键，包括 **Enter** 键和箭头键。要恢复全键盘导航，必须将 `_focusrect` 设为 `true`。

可用性：ActionScript 1.0；Flash Player 6

### 示例

此示例演示：当 SWF 文件中指定的影片剪辑实例周围的黄色矩形在浏览器窗口中具有焦点时，如何隐藏该矩形。创建名为 `mc1_mc`、`mc2_mc` 和 `mc3_mc` 的三个影片剪辑，然后将以下 **ActionScript** 添加到时间轴的第 1 帧：

```
mc1_mc._focusrect = true;
mc2_mc._focusrect = false;
mc3_mc._focusrect = true;

mc1_mc.onRelease = traceOnRelease;
mc3_mc.onRelease = traceOnRelease;

function traceOnRelease() {
    trace(this._name);
}
```

在浏览器窗口中测试 SWF 文件，方法是：选择“文件”>“发布预览”>“HTML”。指定 SWF 焦点，方法是在浏览器窗口中单击它，然后按 **Tab** 键将焦点移至每个实例。`_focusrect` 禁用时，不能通过按 **Enter** 键或空格键来执行此影片剪辑的代码。

此外，您可以在测试环境中测试 SWF 文件。在测试环境中，从主菜单中选择“控制”>“禁用快捷键”。这样，您就可以查看 SWF 文件中实例周围的焦点矩形。

另请参见

[\\_focusrect 属性](#)，[\\_focusrect \(Button.\\_focusrect 属性\)](#)



## `_framesloaded` (MovieClip.\_framesloaded 属性)

`public _framesloaded : Number [read-only]`

从流式 SWF 文件加载的帧数。此属性可用于确定特定帧及其前面所有帧的内容是否已经加载，并且是否可在浏览器本地使用。也可用于监视大 SWF 文件的下载。例如，可能需要向用户显示一条消息以表明在完成 SWF 文件中指定帧的加载前，SWF 文件将会一直进行加载。

可用性：ActionScript 1.0；Flash Player 4

### 示例

下列示例在加载了所有的帧时使用 `_framesloaded` 属性来启动 SWF 文件。如果尚未加载所有帧，则会按比例增大 `bar_mc` 影片剪辑实例的 `_xscale` 属性，以创建进度栏。

在时间轴的第 1 帧中输入下面的 **ActionScript**：

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
    this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

将以下代码添加到第 2 帧：

```
if (this._framesloaded < this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

将您的内容放在第 3 帧上或其后。然后将以下代码添加到第 3 帧：

```
stop();
```

另请参见

[MovieClipLoader](#)

## getBounds (MovieClip.getBounds 方法)

public getBounds(bounds:Object) : Object

基于 bounds 参数，返回作为影片剪辑的最小和最大 x 和 y 坐标值的属性。



使用 MovieClip.localToGlobal() 和 MovieClip.globalToLocal() 方法分别将影片剪辑的本地坐标转换为舞台坐标，或将舞台坐标转换为本地坐标。

您可以通过创建子类来扩展 MovieClip 类的方法和事件处理函数。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

bounds:Object - 要将其坐标系统用作参考点的时间轴的目标路径。

### 返回

Object - 具有下列属性的对象: xMin、xMax、yMin 和 yMax。

### 示例

下面的示例创建一个名为 square\_mc 的影片剪辑。代码为该影片剪辑绘制一个正方形，并使用 MovieClip.getBounds() 在“输出”面板中显示实例的坐标值。

```
this.createEmptyMovieClip("square_mc", 1);
square_mc._x = 10;
square_mc._y = 10;
square_mc.beginFill(0xFF0000);
square_mc.moveTo(0, 0);
square_mc.lineTo(100, 0);
square_mc.lineTo(100, 100);
square_mc.lineTo(0, 100);
square_mc.lineTo(0, 0);
square_mc.endFill();

var bounds_obj:Object = square_mc.getBounds(this);
for (var i in bounds_obj) {
    trace(i+" --> "+bounds_obj[i]);
}
```

在“输出”面板中显示下面的信息:

```
yMax --> 110
yMin --> 10
xMax --> 110
xMin --> 10
```

另请参见

[getRect \(MovieClip.getRect 方法\)](#), [globalToLocal \(MovieClip.globalToLocal 方法\)](#),  
[localToGlobal \(MovieClip.localToGlobal 方法\)](#)

## getBytesLoaded (MovieClip.getBytesLoaded 方法)

`public getBytesLoaded() : Number`

返回已加载（流处理）的影片剪辑的字节数。您可以将此值与 `MovieClip.getBytesTotal()` 返回的值进行比较以确定已加载影片剪辑的百分比。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

**可用性:** ActionScript 1.0 ; Flash Player 5

### 返回

Number - 一个指示已加载的字节数的整数。

### 示例

下列示例在加载了所有的帧时使用 `_framesloaded` 属性来启动 SWF 文件。如果尚未加载所有帧，则会按比例增大 `loader` 影片剪辑实例的 `_xscale` 属性，以创建进度栏。

在时间轴的第 1 帧中输入下面的 **ActionScript**:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/this.getBytesTotal()  
    * 100);  
bar_mc._xscale = pctLoaded;
```

将以下代码添加到第 2 帧:

```
if (this._framesloaded<this._totalframes) {  
    this.gotoAndPlay(1);  
} else {  
    this.gotoAndStop(3);  
}
```

将您的内容放在第 3 帧上或其后。然后将以下代码添加到第 3 帧:

```
stop();
```

另请参见

[getBytesTotal \(MovieClip.getBytesTotal 方法\)](#)

## getBytesTotal (MovieClip.getBytesTotal 方法)

`public getBytesTotal() : Number`

以字节为单位返回影片剪辑的大小。对于那些外部的影片剪辑（加载到某个目标或某个级别的根 SWF 文件或影片剪辑），返回值为未压缩的 SWF 文件的大小。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数，以字节为单位指示影片剪辑的总共大小。

### 示例

下列示例在加载了所有的帧时使用 `_framesloaded` 属性来启动 SWF 文件。如果尚未加载所有帧，则会按比例增大 `loader` 影片剪辑实例的 `_xscale` 属性，以创建进度栏。

在时间轴的第 1 帧中输入下面的 **ActionScript**:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
    this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

将以下代码添加到第 2 帧:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

将您的内容放在第 3 帧上或其后。然后将以下代码添加到第 3 帧:

```
stop();
```

### 另请参见

[getBytesLoaded \(MovieClip.getBytesLoaded 方法\)](#)

## getDepth (MovieClip.getDepth 方法)

`public getDepth() : Number`

返回影片剪辑实例的深度。

每个影片剪辑、按钮和文本字段都有与自己关联的唯一深度，它确定对象在其它对象前或其它对象后的显示方式。具有较大深度值的对象显示在前面。在设计时（使用创作工具）创建的内容开始于深度 **-16383**。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 返回

Number - 影片剪辑的深度。

### 示例

下面的代码跟踪舞台上所有影片剪辑实例的深度：

```
for (var i in this) {  
    if (typeof (this[i]) == "movieclip") {  
        trace("movie clip '"+this[i]._name+"' is at depth "+this[i].getDepth());  
    }  
}
```

### 另请参见

[getInstanceAtDepth \(MovieClip.getInstanceAtDepth 方法\)](#), [getNextHighestDepth \(MovieClip.getNextHighestDepth 方法\)](#), [swapDepths \(MovieClip.swapDepths 方法\)](#), [getDepth \(TextField.getDepth 方法\)](#), [getDepth \(Button.getDepth 方法\)](#)

## getInstanceAtDepth (MovieClip.getInstanceAtDepth 方法)

```
public getInstanceAtDepth(depth:Number) : MovieClip
```

确定特定深度是否已被影片剪辑占用。您可以在使用 `MovieClip.attachMovie()`、`MovieClip.duplicateMovieClip()` 或 `MovieClip.createEmptyMovieClip()` 前使用此方法确定要传递给这些方法的深度参数所指定的深度是否已包含影片剪辑。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**depth:**Number - 一个整数, 指定要查询的深度级别。

### 返回

**MovieClip** - 对位于指定深度的 **MovieClip** 实例的引用; 如果该深度处没有影片剪辑, 则为 `undefined`。

### 示例

下列示例将影片剪辑实例 `triangle` 占用的深度显示在“输出”面板中:

```
this.createEmptyMovieClip("triangle", 1);
```

```
triangle.beginFill(0x0000FF, 100);  
triangle.moveTo(100, 100);  
triangle.lineTo(100, 150);  
triangle.lineTo(150, 100);  
triangle.lineTo(100, 100);
```

```
trace(this.getInstanceAtDepth(1)); // output: _level0.triangle
```

### 另请参见

[attachMovie](#) ([MovieClip.attachMovie](#) 方法), [duplicateMovieClip](#) ([MovieClip.duplicateMovieClip](#) 方法), [createEmptyMovieClip](#) ([MovieClip.createEmptyMovieClip](#) 方法), [getDepth](#) ([MovieClip.getDepth](#) 方法), [getNextHighestDepth](#) ([MovieClip.getNextHighestDepth](#) 方法), [swapDepths](#) ([MovieClip.swapDepths](#) 方法)

## getNextHighestDepth (MovieClip.getNextHighestDepth 方法)

public getNextHighestDepth() : Number

确定可传递给 MovieClip.attachMovie()、MovieClip.duplicateMovieClip() 或 MovieClip.createEmptyMovieClip() 的深度值，以确保 **Flash** 将该影片剪辑呈现在当前影片剪辑中同一级和同一层上所有其它对象的前面。返回的值为 **0** 或更大的数字（即，不返回负数）。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。



如果使用第 2 版的组件，则不要使用此方法。如果您在舞台上或库中放置了第 2 版的版组件，getNextHighestDepth() 方法有时会返回深度 **1048676**，它超出了有效范围。如果使用第 2 版的组件，则应始终使用第 2 版的组件 DepthManager 类。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 返回

Number - 一个整数，反映下一个可用的深度索引，采用该深度值的对象将呈现在影片剪辑中同一级和同一层上所有其它对象之上。

### 示例

下列示例使用 getNextHighestDepth() 方法作为 createEmptyMovieClip() 方法的 depth 参数绘制三个影片剪辑实例，并使用每个影片剪辑的深度作为各自的标签：

```
for (i = 0; i < 3; i++) {  
    drawClip(i);  
}  
  
function drawClip(n:Number):Void {  
    this.createEmptyMovieClip("triangle" + n, this.getNextHighestDepth());  
    var mc:MovieClip = eval("triangle" + n);  
    mc.beginFill(0x00aaFF, 100);  
    mc.lineStyle(4, 0xFF0000, 100);  
    mc.moveTo(0, 0);  
    mc.lineTo(100, 100);  
    mc.lineTo(0, 100);  
    mc.lineTo(0, 0);  
    mc._x = n * 30;  
    mc._y = n * 50  
    mc.createTextField("label", this.getNextHighestDepth(), 20, 50, 200, 200)  
    mc.label.text = mc.getDepth();  
}
```

另请参见

`getDepth` (`MovieClip.getDepth` 方法), `getInstanceAtDepth` (`MovieClip.getInstanceAtDepth` 方法), `swapDepths` (`MovieClip.swapDepths` 方法), `attachMovie` (`MovieClip.attachMovie` 方法), `duplicateMovieClip` (`MovieClip.duplicateMovieClip` 方法), `createEmptyMovieClip` (`MovieClip.createEmptyMovieClip` 方法)

## getRect (MovieClip.getRect 方法)

```
public getRect(bounds:Object) : Object
```

基于 `bounds` 参数, 返回作为影片剪辑的最小和最大 `x` 和 `y` 坐标值的属性, 不包括形状上的任何笔触。 `getRect()` 返回的值小于或等于由 `MovieClip.getBounds()` 返回的值。



使用 `MovieClip.localToGlobal()` 和 `MovieClip.globalToLocal()` 方法分别将影片剪辑的本地坐标转换为舞台坐标, 或将舞台坐标转换为本地坐标。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**bounds:Object** - 要将其坐标系统用作参考点的时间轴的目标路径。

### 返回

**Object** - 具有下列属性的对象: `xMin`、`xMax`、`yMin` 和 `yMax`。

### 示例

下列示例创建影片剪辑并在其中绘制一个笔触宽度为 4 像素的方形。然后, 该示例调用 `MovieClip.getBounds()` 和 `MovieClip.getRect()` 方法以显示它们之间的差异。

`getBounds()` 方法返回整个影片剪辑的最小和最大坐标值, 其中包括方形的笔触宽度。

`getRect()` 方法返回排除 4 像素笔触宽度之后的最小和最大坐标值。

```
this.createEmptyMovieClip("square_mc", 1);
square_mc._x = 10;
square_mc._y = 10;
square_mc.beginFill(0xFF0000);
square_mc.lineStyle(4, 0xFF00FF, 100, true, "none", "round", "miter", 1);
square_mc.moveTo(0, 0);
square_mc.lineTo(100, 0);
square_mc.lineTo(100, 100);
square_mc.lineTo(0, 100);
square_mc.lineTo(0, 0);
square_mc.endFill();
```



```

var bounds_obj:Object = square_mc.getBounds(this);
trace("getBounds() output:");
for (var i in bounds_obj) {
    trace(i+" --> "+bounds_obj[i]);
}

var rect_obj:Object = square_mc.getRect(this);
trace("getRect() output:");
for (var i in rect_obj) {
    trace(i+" --> "+rect_obj[i]);
}

```

trace() 语句导致下列输出。

```

getBounds() output:
yMax --> 112
yMin --> 8
xMax --> 112
xMin --> 8
getRect() output:
yMax --> 110
yMin --> 10
xMax --> 110
xMin --> 10

```

另请参见

[getBounds \(MovieClip.getBounds 方法\)](#), [globalToLocal \(MovieClip.globalToLocal 方法\)](#), [localToGlobal \(MovieClip.localToGlobal 方法\)](#)

## getSWFVersion (MovieClip.getSWFVersion 方法)

```
public getSWFVersion() : Number
```

返回一个整数，该整数指示所发布的影片剪辑的 **Flash Player** 版本。如果影片剪辑是 JPEG、GIF 或 PNG 文件，或者如果出现错误而且 **Flash Player** 无法确定影片剪辑的 SWF 版本，则返回 -1。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 返回

Number - 一个整数，该整数指定在加载到影片剪辑中的 **SWF** 文件发布时以其为目标播放器的 **Flash Player** 版本。

## 示例

下列示例创建新的容器并输出 `getSWFVersion()` 的值。然后，它使用 **MovieClipLoader** 加载发布到 **Flash Player 7** 的外部 **SWF** 文件，并在 `onLoadInit` 处理函数被触发后输出 `getSWFVersion()` 的值。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getUpperEmptyDepth());
var listener:Object = new Object();
listener.onLoadInit = function(target:MovieClip):Void {
    trace("target: " + target.getSWFVersion()); // target: 7
}
var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(listener);
trace("container: " + container.getSWFVersion()); // container: 8
mcLoader.loadClip("FlashPlayer7.swf", container);
```

## getTextSnapshot (MovieClip.getTextSnapshot 方法)

`public getTextSnapshot(): TextSnapshot`

返回一个 **TextSnapshot** 对象，该对象包含指定影片剪辑的所有静态文本字段中的文本；不包括子级影片剪辑中的文本。此方法始终返回 **TextSnapshot** 对象。

**Flash** 连接文本并将文本放置于 **TextSnapshot** 对象中，放置的顺序反映影片剪辑中静态文本字段的 **Tab** 键索引顺序。没有 **Tab** 键索引值的文本字段将按随机顺序放置于该对象中，并且放在所有具有 **Tab** 键索引值的字段的文本之前。没有换行符或格式指示一个字段从哪里结束、另一个字段从哪里开始。



您不能在 **Flash** 中为静态文本指定 **Tab** 键索引值。但是，其它产品（例如 **Macromedia FlashPaper**）可能可以。

**TextSnapshot** 对象的内容不是动态内容；即，如果影片剪辑移到不同的帧，或者以某种方式被更改（例如，影片剪辑中的对象被添加或删除），则 **TextSnapshot** 对象可能不显示影片剪辑中的当前文本。若要确保该对象的内容是最新的，应根据需要重新发布此命令。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 7

## 返回

**TextSnapshot** - 包含来自影片剪辑的静态文本的一个 **TextSnapshot** 对象。

## 示例

下列示例说明如何使用此方法。要使用此代码，请将包含文本 “TextSnapshot Example” 的静态文本字段放在舞台上。

```
var textSnap:TextSnapshot = this.getTextSnapshot();  
trace(textSnap.getText(0, textSnap.getCount(), false));
```

另请参见

[TextSnapshot](#)

## getURL (MovieClip.getURL 方法)

```
public getURL(url:String, [window:String], [method:String]) : Void
```

从指定 URL 将文档加载到指定窗口。也可以使用 getURL() 方法将变量传递给通过使用 GET 或 POST 方法在 URL 中定义的另一个应用程序。

承载 Flash 内容的 Web 页必须显式设置 allowScriptAccess 属性，以便允许或拒绝从 HTML 代码（对于 Internet Explorer，在 PARAM 标记中；对于 Netscape Navigator，在 EMBED 标记中）为 Flash Player 编写脚本。

- allowScriptAccess 为 "never" 时，外出脚本将始终失败。
- allowScriptAccess 为 "always" 时，外出脚本将始终成功。
- allowScriptAccess 为 "sameDomain"（可由第 8 版以后的 SWF 文件支持）时，如果 SWF 文件与承载 Web 页来自同一域，则允许外出脚本。
- 如果 HTML 页未指定 allowScriptAccess，则对于第 8 版 SWF 文件，默认值为 "sameDomain"，而对于以前版本的 SWF 文件，默认值为 "always"。

使用此方法时，请考虑 Flash Player 安全模型。对于 Flash Player 8，如果发出调用的 SWF 文件位于只能与本地文件系统的内容交互的沙箱中且资源是非本地的，则不允许使用此方法。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

您可以通过创建子类来扩展 MovieClip 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**url:String** - 可从该处获取文档的 URL。

**window:String [ 可选 ]** - 一个参数，指定名称、框架或表达式，这些内容指定文档将加载到其中的窗口或 HTML 框架。也可以使用下列保留的目标名称之一：\_self 指定当前窗口中的当前帧，\_blank 指定一个新窗口，\_parent 指定当前帧的父级，而 \_top 指定当前窗口中的顶级帧。

**method:String [ 可选 ]** - 字符串 ("GET" 或 "POST")，指定用于发送与要加载的 SWF 文件相关联的变量的方法。如果不存在变量，则可忽略此参数；否则，请指定是否使用 GET 或 POST 方法加载变量。GET 将变量追加到 URL 的末尾并用于少量变量的情况。POST 在单独的 HTTP 标头中发送变量并用于长字符串变量的情况。

## 示例

下列 **ActionScript** 创建一个新的影片剪辑实例，并在新的浏览器窗口中打开 **Macromedia** 网站：

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.getURL("http://www.macromedia.com", "_blank");
```

getURL() 方法还允许您将变量发送到远程服务器端脚本，如下列代码所示：

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.username = "some user input";
loader_mc.password = "random string";
loader_mc.getURL("http://www.flash-mx.com/mm/viewscope.cfm", "_blank",
    "GET");
```

这些示例中使用的 MovieClip.getNextHighestDepth() 方法需要 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

使用此方法时，请考虑 **Flash Player** 安全模型。

- 对于 **Flash Player 8**，如果发出调用的 SWF 文件位于带有文件系统的本地沙箱中且资源是非本地的，则不允许使用 MovieClip.getURL()。

有关更多信息，请参见以下部分：

- 《学习 **Flash** 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

另请参见

[getURL 函数](#)，[sendAndLoad \(LoadVars.sendAndLoad 方法\)](#)，[send \(LoadVars.send 方法\)](#)

## globalToLocal (MovieClip.globalToLocal 方法)

```
public globalToLocal(pt:Object) : Void
```

将 pt 对象从舞台（全局）坐标转换为影片剪辑（本地）坐标。

通过 MovieClip.globalToLocal() 方法，可以将任何给定的 **x** 和 **y** 坐标从相对于舞台左上角的值转换为相对于特定影片剪辑左上角的值。

您必须首先创建一个具有两个属性（即 **x** 和 **y**）的通用对象。这些 **x** 和 **y** 值（而且它们必须称为 **x** 和 **y**）称为全局坐标，因为它们相对于舞台的左上角。**x** 属性表示从左上角的水平偏移量。换句话说，它表示点所在位置向右的距离。例如，如果 **x = 50**，则该点在左上角向右 **50** 像素处。**y** 属性表示从左上角的垂直偏移量。换句话说，它表示点所在位置向下的距离。例如，如果 **y = 20**，则点在左上角向下 **20** 像素处。下列代码使用这些坐标创建一个通用对象：

```
var myPoint:Object = new Object();  
myPoint.x = 50;  
myPoint.y = 20;
```

或者，您也可以使用 **Object** 文本值同时创建对象并分配值。

```
var myPoint:Object = {x:50, y:20};
```

使用全局坐标创建点对象后，您可以将全局坐标转换为本地坐标。globalToLocal() 方法不返回值，因为它更改通用对象中作为参数发送的 **x** 和 **y** 值。该方法将它们从相对于舞台的值（全局坐标）更改为相对于特定影片剪辑的值（本地坐标）。

例如，如果要创建放置在点 (**\_x:100**, **\_y:100**) 上的影片剪辑，并且将表示舞台左上角的全局点 (**x:0**, **y:0**) 传递给 globalToLocal() 方法，该方法应将 **x** 和 **y** 值转换为本地坐标，在本例中则是 (**x:-100**, **y:-100**)。之所以发生此转换，是因为 **x** 和 **y** 坐标现在是相对于影片剪辑的左上角表示的，而不是相对于舞台的左上角。值为负，因为若要从影片剪辑的左上角到达舞台的左上角，必须向左移动 **100** 像素（负 **x**）并向上移动 **100** 像素（负 **y**）。

影片剪辑坐标是使用 **\_x** 和 **\_y** 来表示的，因为它们是用来设置 **MovieClips** 的 **x** 和 **y** 值的 **MovieClip** 属性。但是，通用对象使用不带下划线的 **x** 和 **y**。下面的代码将 **x** 和 **y** 值转换为本地坐标：

```
var myPoint:Object = {x:0, y:0}; // Create your generic point object.  
this.createEmptyMovieClip("myMovieClip", this.getNextHighestDepth());  
myMovieClip._x = 100; // _x for movieclip x position  
myMovieClip._y = 100; // _y for movieclip y position  
  
myMovieClip.globalToLocal(myPoint);  
trace ("x: " + myPoint.x); // output: -100  
trace ("y: " + myPoint.y); // output: -100
```

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**pt:Object** - 用通用 **Object** 类创建的对象名称或标识符。该对象指定 **x** 和 **y** 坐标作为属性。

## 示例

将以下 **ActionScript** 添加到与名为 **photo1.jpg** 的图像在同一目录中的 **FLA** 或 **AS** 文件:

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
coords_txt.html = true;
coords_txt.multiline = true;
coords_txt.autoSize = true;
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc._x = 100;
target_mc._y = 100;
target_mc.loadMovie("photo1.jpg");

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var point:Object = {x:_xmouse, y:_ymouse};
    target_mc.globalToLocal(point);
    var rowHeaders = "<b> &nbsp; \t</b><b>_x\t</b><b>_y</b>";
    var row_1 = "_root\t"+_xmouse+"\t"+_ymouse;
    var row_2 = "target_mc\t"+point.x+"\t"+point.y;
    coords_txt.htmlText = "<textformat tabstops='[100, 150]'\t";
    coords_txt.htmlText += rowHeaders;
    coords_txt.htmlText += row_1;
    coords_txt.htmlText += row_2;
    coords_txt.htmlText += "</textformat>";
};
Mouse.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[getBounds \(MovieClip.getBounds 方法\)](#), [localToGlobal \(MovieClip.localToGlobal 方法\)](#), [Object](#)

## gotoAndPlay (MovieClip.gotoAndPlay 方法)

```
public gotoAndPlay(frame:Object) : Void
```

从指定帧开始播放 SWF 文件。若要指定场景以及帧，请使用 gotoAndPlay()。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**frame:Object** - 表示播放头转到的帧编号的数字，或者表示播放头转到的帧标签的字符串。

### 示例

下列示例在加载了所有的帧时使用 `_framesloaded` 属性来启动 SWF 文件。如果尚未加载所有帧，则会按比例增大 `loader` 影片剪辑实例的 `_xscale` 属性，以创建进度栏。

在时间轴的第 1 帧中输入下面的 **ActionScript**:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
    this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

将以下代码添加到第 2 帧:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

将您的内容放在第 3 帧上或其后。然后将以下代码添加到第 3 帧:

```
stop();
```

另请参见

[gotoAndPlay 函数](#), [play 函数](#)

## gotoAndStop (MovieClip.gotoAndStop 方法)

```
public gotoAndStop(frame:Object) : Void
```

将播放头移到影片剪辑的指定帧并停在那里。除了指定帧以外，您还要指定场景，请使用 `gotoAndStop()` 方法。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**frame:Object** - 要将播放头发送给的帧的编号。

### 示例

下列示例在加载了所有的帧时使用 `_framesloaded` 属性来启动 **SWF** 文件。如果尚未加载所有帧，则会按比例增大 `loader` 影片剪辑实例的 `_xscale` 属性，以创建进度栏。

在时间轴的第 1 帧中输入下面的 **ActionScript**:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
    this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

将以下代码添加到第 2 帧:

```
if (this._framesloaded<this._totalframes) {
    this.gotoAndPlay(1);
} else {
    this.gotoAndStop(3);
}
```

将您的内容放在第 3 帧上或其后。然后将以下代码添加到第 3 帧:

```
stop();
```

### 另请参见

[gotoAndStop 函数](#)，[stop 函数](#)



## `_height` (`MovieClip._height` 属性)

`public _height : Number`

影片剪辑的高度，以像素为单位。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

下列代码示例在“输出”面板中显示影片剪辑的高度和宽度:

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var image_mc1:MovieClipLoader = new MovieClipLoader();
var mc1Listener:Object = new Object();
mc1Listener.onLoadInit = function(target_mc:MovieClip) {
    trace(target_mc._name+" = "+target_mc._width+" X "+target_mc._height+"
    pixels");
};
image_mc1.addListener(mc1Listener);

image_mc1.loadClip("example.jpg", image_mc);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

此示例中使用的 `MovieClipLoader` 类需要 **Flash Player 7** 或更高版本。

另请参见

[\\_width \(MovieClip.\\_width 属性\)](#)

## `_highquality` (`MovieClip._highquality` 属性)

`public _highquality : Number`

自 **Flash Player 7** 后**不推荐使用**。不推荐使用此属性，而推荐使用 `MovieClip._quality`。

指定当前 **SWF** 文件所应用的消除锯齿的级别。指定 **2**（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 **1**（高品质），则应用消除锯齿功能；如果 **SWF** 文件不包含动画，将对位图进行平滑处理。指定 **0**（低品质），则不消除锯齿。此属性可以覆盖全局 `_highquality` 属性。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 示例

下面的 **ActionScript** 指定应该将最高品质的消除锯齿应用于 SWF 文件。

```
my_mc._highquality = 2;
```

## 另请参见

[\\_quality \(MovieClip.\\_quality 属性\)](#), [\\_quality 属性](#)

## hitArea (MovieClip.hitArea 属性)

```
public hitArea : Object
```

将另一个影片剪辑指定为影片剪辑的点击区域。如果 hitArea 属性不存在或者其值为 null 或 undefined，则影片剪辑本身将用作点击区域。hitArea 属性的值可以是对影片剪辑对象的一个引用。

可以随时更改 hitArea 属性；修改后的影片剪辑会立即使用新的点击区域行为。指定为点击区域的影片剪辑不必是可见的；虽然不可见，其图形形状仍作为点击区域被检测。

**可用性：** ActionScript 1.0 ； Flash Player 6

## 示例

下面的示例将 circle\_mc 影片剪辑设置为 square\_mc 影片剪辑的点击区域。将这两个影片剪辑放在舞台上并测试文档。当您单击 circle\_mc 时，square\_mc 影片剪辑会对它已被单击进行跟踪。

```
square_mc.hitArea = circle_mc;
square_mc.onRelease = function() {
    trace("hit! "+this._name);
};
```

您也可以将 circle\_mc 影片剪辑的 visible 属性设置为 false，以隐藏 square\_mc 的点击区域。

```
circle_mc._visible = false;
```

## 另请参见

[hitTest \(MovieClip.hitTest 方法\)](#)

## hitTest (MovieClip.hitTest 方法)

`public hitTest() : Boolean`

计算影片剪辑，以确认其是否与由 `target` 或 `x` 和 `y` 坐标参数标识的点击区域发生重叠或相交。

**用法 1:** 根据 `shapeFlag` 设置，将 `x` 和 `y` 坐标与指定实例的形状或边框进行比较。如果 `shapeFlag` 设置为 `true`，则只计算在舞台上的实例实际占据的区域，并且如果 `x` 和 `y` 在任意一点重叠，则返回 `true` 值。此评估对于确定影片剪辑是否处于指定的点击区域或热点区域中很有用。

**用法 2:** 计算 `target` 和指定实例的边框，如果它们在任意一点上重叠或交叉，则返回 `true`。

**参数** `x`: Number 舞台上点击区域的 `x` 坐标。`y`: Number 舞台上点击区域的 `y` 坐标。`x` 和 `y` 坐标都在全局坐标空间中定义。`shapeFlag`: Boolean 一个布尔值，指定是计算指定实例的整个形状 (`true`) 还是仅计算边框 (`false`)。只有当用 `x` 和 `y` 坐标参数标识点击区域时，才可以指定该参数。`target`: Object 可能与影片剪辑相交或重叠的点击区域的目标路径。`target` 参数通常表示一个按钮或一个文本输入字段。

**可用性:** `ActionScript 1.0` ; `Flash Player 5`

### 返回

Boolean - 一个布尔值，如果影片剪辑与指定点击区域重叠，则为 `true`，否则为 `false`。

### 示例

下面的示例使用 `hitTest()` 确定在用户释放鼠标按钮时影片剪辑 `circle_mc` 是否与影片剪辑 `square_mc` 重叠或交叉：

```
square_mc.onPress = function() {
    this.startDrag();
};
square_mc.onRelease = function() {
    this.stopDrag();
    if (this.hitTest(circle_mc)) {
        trace("you hit the circle");
    }
};
```

### 另请参见

[getBounds \(MovieClip.getBounds 方法\)](#), [globalToLocal \(MovieClip.globalToLocal 方法\)](#), [localToGlobal \(MovieClip.localToGlobal 方法\)](#)


# lineGradientStyle (MovieClip.lineGradientStyle 方法)

```
public lineGradientStyle(fillType:String, colors:Array, alphas:Array,
    ratios:Array, matrix:Object, [spreadMethod:String],
    [interpolationMethod:String], [focalPointRatio:Number]) : Void
```

指定 **Flash** 用于后续 `lineTo()` 和 `curveTo()` 方法调用的线条样式，在以不同参数调用 `lineStyle()` 方法或 `lineGradientStyle()` 方法之前，线条样式不会改变。可以在绘制路径的中间调用 `lineGradientStyle()` 方法以为路径中的不同线段指定不同的样式。



在调用 `lineGradientStyle()` 之前调用 `lineStyle()` 以启用笔触，否则线条样式的值仍然是 `undefined`。



调用 `clear()` 会将线条样式重新设置为 `undefined`。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 8

## 参数




**fillType:String** - 有效值为 "linear" 或 "radial"。

**colors:Array** - 在渐变中使用的 **RGB** 十六进制颜色值数组（例如，红色为 `0xFF0000`，蓝色为 `0x0000FF`，等等）。可以至多指定 15 种颜色。对于每种颜色，请确保在 `alphas` 和 `ratios` 参数中指定对应值。

**alphas:Array** - `colors` 数组中对应颜色的 **Alpha** 值数组；有效值为 0 到 100。如果值小于 0，则 **Flash** 使用 0。如果值大于 100，则 **Flash** 使用 100。

**ratios:Array** - 颜色分布比率数组；有效值为 0 到 255。该值定义颜色采样率为 100% 之处的宽度百分比。为 `colors` 参数中的每个值指定一个值。

例如，对于包含蓝和绿两种颜色的线性渐变，下图显示了基于不同 `ratios` 数组值的颜色配比：

ratios	渐变
[0, 127]	
[0, 255]	
[127, 255]	

数组中的值必须持续增加；例如，`[0, 63, 127, 190, 255]`。

**matrix:Object** - 一个转换矩阵，它是具有下列若干组属性之一的对象。

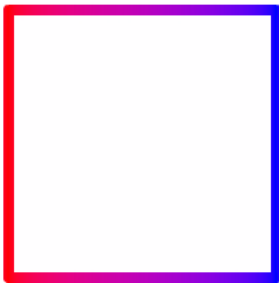
- 可以使用属性 a、b、c、d、e、f、g、h 和 i 描述下列格式的 **3 x 3** 矩阵：

```
a b c  
d e f  
g h i
```

下列示例使用带有 **matrix** 参数的 **lineGradientFill()** 方法，该参数是具有下列属性的对象：

```
this.createEmptyMovieClip("gradient_mc", 1);  
with (gradient_mc) {  
    colors = [0xFF0000, 0x0000FF];  
    alphas = [100, 100];  
    ratios = [0, 0xFF];  
    matrix = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};  
    spreadMethod = "reflect";  
    interpolationMethod = "linearRGB";  
    focalPointRatio = 0.9;  
    lineStyle(8);  
    lineGradientStyle("linear", colors, alphas, ratios, matrix,  
        spreadMethod, interpolationMethod, focalPointRatio);  
    moveTo(100, 100);  
    lineTo(100, 300);  
    lineTo(300, 300);  
    lineTo(300, 100);  
    lineTo(100, 100);  
    endFill();  
}
```

此代码在屏幕上绘制下列图像：



■ `matrixType, x, y, w, h, r.`

这些属性表示下列含义: `matrixType` 是字符串 "box", `x` 是渐变的左上角相对于父级剪辑注册点的水平位置, `y` 是渐变的左上角相对于父级剪辑注册点的垂直位置, `w` 是渐变的宽度, `h` 是渐变的高度, `r` 是渐变的旋转程度 (以弧度为单位)。

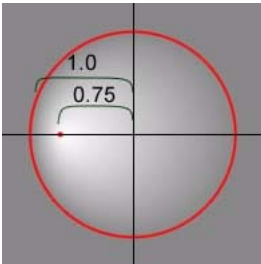
下列示例使用带有 `matrix` 参数的 `lineGradientFill()` 方法, 该参数是具有这些属性的对象:

```
this.createEmptyMovieClip("gradient_mc", 1);
with (gradient_mc) {
    colors = [0xFF0000, 0x0000FF];
    alphas = [100, 100];
    ratios = [0, 0xFF];
    matrix = {matrixType:"box", x:100, y:100, w:200, h:200, r:(45/
    180)*Math.PI};
    spreadMethod = "reflect";
    interpolationMethod = "linearRGB";
    lineStyle(8);
    lineGradientStyle("linear", colors, alphas, ratios, matrix,
        spreadMethod, interpolationMethod);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
}
```

`spreadMethod:String` [ 可选 ] - 有效值为 "pad"、"reflect" 或 "repeat", 它控制渐变填充的模式。

`interpolationMethod:String` [ 可选 ] - 有效值为 "RGB" 或 "linearRGB"。

`focalPointRatio:Number` [ 可选 ] - 一个数字, 控制渐变焦点的位置。值 0 表示焦点位于中心。值 1 表示焦点位于渐变圆的一条边界上。值 -1 表示焦点位于渐变圆的另一条边界上。小于 -1 或大于 1 的值被舍入为 -1 或 1。下列图像显示 `focalPointRatio` 为 -0.75 的渐变:

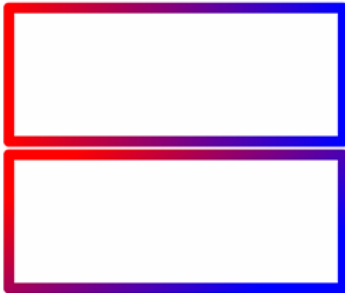


## 示例

下面的代码使用这两种方法绘制两个具有红蓝线渐变填充的堆叠矩形：

```
this.createEmptyMovieClip("gradient_mc", 1);
with (gradient_mc) {
    colors = [0xFF0000, 0x0000FF];
    alphas = [100, 100];
    ratios = [0, 0xFF];
    matrix = {a:500, b:0, c:0, d:0, e:200, f:0, g:350, h:200, i:1};
    lineStyle(16);
    lineGradientStyle("linear", colors, alphas, ratios, matrix);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(600, 300);
    lineTo(600, 100);
    lineTo(100, 100);
    endFill();
    matrix2 = {matrixType:"box", x:100, y:310, w:500, h:200, r:(30/
    180)*Math.PI};
    lineGradientStyle("linear", colors, alphas, ratios, matrix2);
    moveTo(100, 320);
    lineTo(100, 520);
    lineTo(600, 520);
    lineTo(600, 320);
    lineTo(100, 320);
    endFill();
}
```

此代码将绘制下列图像（图像缩小了 50%）：



另请参见

[beginGradientFill](#) ([MovieClip.beginGradientFill](#) 方法), [lineStyle](#) ([MovieClip.lineStyle](#) 方法), [lineTo](#) ([MovieClip.lineTo](#) 方法), [moveTo](#) ([MovieClip.moveTo](#) 方法)

## lineStyle (MovieClip.lineStyle 方法)

```
public lineStyle(thickness:Number, rgb:Number, alpha:Number,  
    pixelHinting:Boolean, noScale:String, capsStyle:String, jointStyle:String,  
    miterLimit:Number) : Void
```

指定 **Flash** 用于后续 `lineTo()` 和 `curveTo()` 方法调用的线条样式，在以不同参数调用 `lineStyle()` 方法之前，线条样式不会改变。可以在绘制路径的中间调用 `lineStyle()` 以为路径中的不同线段指定不同的样式。



对 `clear()` 方法的调用会将线条样式的值设置回 `undefined`。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**thickness:**`Number` - 一个整数，以磅为单位指示线条的粗细；有效值为 **0** 到 **255**。如果未指定数值，或者该参数为 `undefined`，则不绘制线条。如果传递的值小于 **0**，则 **Flash Player** 使用 **0**。数值 **0** 表示极细的粗细；最大粗细为 **255**。如果传递的值大于 **255**，则 **Flash** 解释程序使用 **255**。

**rgb:**`Number` - 线条的十六进制颜色值（例如，红色为 **0xFF0000**，蓝色为 **0x0000FF**，等等）。如果未指示值，则 **Flash** 使用 **0x000000**（黑色）。

**alpha:**`Number` - 一个整数，指示线条颜色的 **Alpha** 值；有效值为 **0** 到 **100**。如果未指示值，则 **Flash** 使用 **100**（纯色）。如果该值小于 **0**，则 **Flash** 使用 **0**。如果该值大于 **100**，则 **Flash** 使用 **100**。

**pixelHinting:**`Boolean` - 在 **Flash Player 8** 中添加。一个布尔值，它指定是否提示笔触采用完整像素。此值同时影响曲线锚点的位置以及线条笔触大小本身。如果未指示值，则 **Flash Player** 不使用像素提示。

**noScale:**`String` - 在 **Flash Player 8** 中添加。一个字符串，指定如何缩放笔触。有效值如下：

- "normal" 始终缩放粗细（默认值）。
- "none" 从不缩放粗细。
- "vertical" 如果仅垂直缩放对象，则不缩放粗细。
- "horizontal" 如果仅水平缩放对象，则不缩放粗细。

**capsStyle:**`String` - 在 **Flash Player 8** 中添加。一个字符串，指定线条终点的端点类型。有效值为："round"、"square" 和 "none"。如果未指示值，则 **Flash** 使用圆角端点。



例如，下列插图显示了不同的 capsStyle 设置。对于每种设置，插图显示了一条粗细为 30 的蓝色线条（应用 capsStyle 的线条），以及覆于其上的粗细为 1 的黑色线条（未应用 capsStyle 的线条）：



**jointStyle:String** - 在 **Flash Player 8** 中添加。一个字符串，指定用于拐角的连接外观的类型。有效值为："round"、"miter" 和 "bevel"。如果未指示值，则 **Flash** 使用圆形连接。

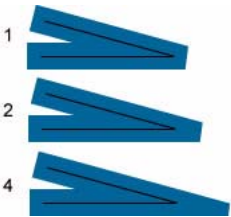
例如，下列插图显示了不同的 capsStyle 设置。对于每种设置，插图显示了一条粗细为 30 的带拐角的蓝色线条（应用 jointStyle 的线条），以及覆于其上的粗细为 1 的带拐角的黑色线条（未应用 jointStyle 的线条）：



请注意，对于设置为 "miter" 的 jointStyle，可以使用 miterLimit 参数限制尖角处的长度。

**miterLimit:Number** - 在 **Flash Player 8** 中添加。一个数字，指示切断尖角的限制。有效值的范围是 1 到 255（超出该范围的值将舍入为 1 或 255）。此值只可用于 jointStyle 设置为 "miter" 的情况下。如果未指定值，Flash 将使用 3。miterLimit 值表示向外延伸的尖角超出角边相交所形成的结合点的长度。此值表示为线条 thickness 的因子。例如，miterLimit 因子为 2.5 且 thickness 为 10 像素时，尖角将在 25 像素处切断。

例如，请考虑下列带拐角的线条，每个线条都以 thickness 20 进行绘制，但它们的 miterLimit 分别设置为 1、2 和 4。覆在其上的黑色参考线条显示了结合处的联结点：



请注意，对于给定的 miterLimit 值，会存在一个可被切断的特定最大角度。下表列出了部分示例：

Value of miterLimit 值：	小于此角度将被切断：
1.414	90 度
2	60 度
4	30 度
8	15 度

示例

下列代码绘制一个三角形，它具有 5 像素的纯洋红色线条，没有填充，具有像素提示，没有笔触缩放，没有端点，具有 miterLimit 设置为 1 的尖角结合点：

```
this.createEmptyMovieClip("triangle_mc", this.getNextHighestDepth());
triangle_mc.lineStyle(5, 0xff00ff, 100, true, "none", "round", "miter", 1);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 DepthManager 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

另请参见

```
beginFill (MovieClip.beginFill 方法), beginGradientFill
(MovieClip.beginGradientFill 方法), clear (MovieClip.clear 方法), curveTo
(MovieClip.curveTo 方法),.lineTo (MovieClip.lineTo 方法), moveTo
(MovieClip.moveTo 方法)
```

lineTo (MovieClip.lineTo 方法)

```
public lineTo(x:Number, y:Number) : Void
```

使用当前线条样式绘制一条从当前绘画位置到 (x, y) 的线条；当前绘画位置随后会设置为 (x, y)。如果正在其中绘制的影片剪辑包含用 Flash 绘画工具创建的内容，则调用 lineTo() 方法将在该内容下面进行绘制。如果在对 moveTo() 进行任何调用之前调用了 lineTo()，则当前绘画位置默认为 (0, 0)。如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

您可以通过创建子类来扩展 MovieClip 类的方法和事件处理函数。

可用性：ActionScript 1.0 ； Flash Player 6

## 参数

**x:**Number - 一个整数，指示相对于父级影片剪辑注册点的水平位置。

**y:**Number - 一个整数，指示相对于父级影片剪辑注册点的垂直位置。

## 示例

下面的示例绘制一个三角形，它具有 5 像素的纯洋红色线条和部分透明的蓝色填充：

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

## 另请参见

[beginFill \(MovieClip.beginFill 方法\)](#), [createEmptyMovieClip \(MovieClip.createEmptyMovieClip 方法\)](#), [endFill \(MovieClip.endFill 方法\)](#), [lineStyle \(MovieClip.lineStyle 方法\)](#), [moveTo \(MovieClip.moveTo 方法\)](#)

## loadMovie (MovieClip.loadMovie 方法)

```
public loadMovie(url:String, [method:String]) : Void
```

在播放原始 SWF 文件时，将 SWF、JPEG、GIF 或 PNG 文件加载到 Flash Player 中的影片剪辑中。在 Flash Player 8 中添加了对非动画 GIF 文件、PNG 文件和渐进式 JPEG 文件的支持。如果加载动画 GIF，则仅显示第一帧。



若要监视下载的进度，请使用 `MovieClipLoader.loadClip()` 方法取代 `loadMovie()` 方法。

如果不使用 `loadMovie()` 方法，则 Flash Player 会显示单个 SWF 文件，然后关闭。使用 `loadMovie()` 方法可以一次显示几个 SWF 文件并且无需加载另一个 HTML 文档即可在 SWF 文件间进行切换。

加载到影片剪辑的 SWF 文件或图像会继承该影片剪辑的位置、旋转和缩放属性。可以用影片剪辑的目标路径来定位加载的 SWF 文件。

在调用 `loadMovie()` 方法时，可按下列代码示例所示在加载器影片中将

`MovieClip._lockroot` 属性设置为 `true`。如果您不在加载器影片中将 `_lockroot` 设置为 `true`，则任何对已加载影片中 `_root` 的引用都会指向加载器的 `_root`，而不是已加载影片的 `_root`：

```
myMovieClip._lockroot = true;
```

使用 `MovieClip.unloadMovie()` 方法可以删除用 `loadMovie()` 方法加载的 SWF 文件或图像。

使用 `MovieClip.loadVariables()` 方法、XML 对象、Flash Remoting 或 Runtime Shared Objects 保留活动的 SWF 文件，并向其加载新数据。

将事件处理函数和 `MovieClip.loadMovie()` 一起使用，其处理结果是无法预知的。如果使用 `on()` 将事件处理函数附加到按钮，或是使用诸如 `MovieClip.onPress()` 的事件处理函数方法创建动态处理函数，然后调用 `loadMovie()`，则在加载新内容之后，事件处理函数将不再可用。然而，如果使用 `onClipEvent()` 或 `on()` 将事件处理函数附加到影片剪辑，然后对该影片剪辑调用 `loadMovie()`，则在加载新内容之后，事件处理函数将仍然可用。

使用此方法时，请考虑 Flash Player 安全模型。

对于 Flash Player 8:

- 如果执行调用的影片剪辑在只能与本地文件系统的内容交互的沙箱中，并且被加载的影片剪辑来自网络沙箱，则不允许加载。
- 如果执行调用的 SWF 文件在网络沙箱中并且要加载的影片剪辑是本地的，则不允许加载。
- 从受信任的本地沙箱或只能与远程内容交互的沙箱访问网络沙箱需要通过跨域策略文件获得网站的许可。
- 在只能与本地文件系统的内容交互的沙箱中的影片剪辑不能对只能与远程内容交互的沙箱中的影片剪辑使用脚本（反之也是禁止的）。

对于 Flash Player 7 及更高版本:

- 网站可以允许通过跨域策略文件来跨域访问资源。
- 各 SWF 文件之间的脚本使用将依据这些 SWF 文件的原始域而予以限制。使用 `System.security.allowDomain()` 方法可调整这些限制。

有关更多信息，请参见以下部分:

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

您可以通过创建子类来扩展 `MovieClip` 类的方法和事件处理函数。

可用性: ActionScript 1.0 ; Flash Player 5

## 参数

**url:String** - 要加载的 SWF、JPEG、GIF 和 PNG 文件的绝对或相对 URL。相对路径必须相对于级别 0 处的 SWF 文件。绝对 URL 必须包括协议引用，例如 http:// 或 file:///。

**method:String** [ 可选 ] - 指定用于发送或加载变量的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量附加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送长字符串的变量。

## 示例

下列示例新建影片剪辑，然后在其中创建子级并将 PNG 图像加载到子级中。此示例允许父级在调用 loadMovie 之前返回任何已分配的实例值。

```
var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mc.onRelease = function():Void {
    trace(this.image._url); // http://www.w3.org/Icons/w3c_main.png
}
var image:MovieClip = mc.createEmptyMovieClip("image",
    mc.getNextHighestDepth());
image.loadMovie("http://www.w3.org/Icons/w3c_main.png");
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 Flash Player 7 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

## 另请参见

[\\_lockroot \(MovieClip.\\_lockroot 属性\)](#), [unloadMovie \(MovieClip.unloadMovie 方法\)](#), [loadVariables \(MovieClip.loadVariables 方法\)](#), [loadMovie \(MovieClip.loadMovie 方法\)](#), [onPress \(MovieClip.onPress 处理函数\)](#), [MovieClipLoader](#), [onClipEvent 处理函数](#), [on 处理函数](#), [loadMovieNum 函数](#), [unloadMovie 函数](#), [unloadMovieNum 函数](#)

## loadVariables (MovieClip.loadVariables 方法)

```
public loadVariables(url:String, [method:String]) : Void
```

从外部文件读取数据并设置影片剪辑中变量的值。外部文件可以是 **Macromedia ColdFusion** 生成的文本文件、**CGI** 脚本、**Active Server Page (ASP)**、**PHP** 脚本或任何其它格式正确的文本文件。此文件可以包含任意数量的变量。

使用 `loadVariables()` 方法还可通过新值更新活动影片剪辑中的变量。

`loadVariables()` 方法要求 **URL** 的文本使用标准的 **MIME** 格式：**application/x-www-form-urlencoded** (**CGI** 脚本格式)。

如果 **SWF** 文件在比 **Flash Player 7** 更低的版本中运行，则 `url` 必须与发布此调用的 **SWF** 文件位于同一个超级域中。超级域可以通过删除某一文件的 **URL** 最左侧的组件而得到。例如，位于 **www.someDomain.com** 的 **SWF** 文件可以从位于 **store.someDomain.com** 的源中加载数据，这是因为这两个文件都在同一个名为 **someDomain.com** 的超级域中。

如果任何版本的 **SWF** 文件运行于 **Flash Player 7** 或更高版本中，则 `url` 必须与发布此调用的 **SWF** 文件位于完全相同的域中。例如，位于 **www.someDomain.com** 的 **SWF** 文件只能从同样位于 **www.someDomain.com** 的源中加载数据。若要从其它域中加载数据，则可以在承载被访问数据源的服务器上放置一个交叉域策略文件。

若要将变量加载到特定级别，请使用 `loadVariablesNum()` 而不是 `loadVariables()`。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**url:String** - 包含要加载变量的外部文件的绝对或相对 **URL**。如果发布此调用的 **SWF** 文件正在 **Web** 浏览器上运行，则 `url` 必须与 **SWF** 文件位于同一个域中；有关详细信息，请参见下面的说明。

**method:String** [ 可选 ] - 指定用于发送变量的 **HTTP** 方法。该参数必须是字符串 **GET** 或 **POST**。如果没有发送变量，则省略此参数。**GET** 方法将变量附加到 **URL** 的末尾，它用于发送少量的变量。**POST** 方法在单独的 **HTTP** 标头中发送变量，它用于发送长字符串的变量。

### 示例

下列示例将信息从名为 `params.txt` 的文本文件加载到使用 `createEmptyMovieClip()` 创建的影片剪辑 `target_mc` 中。使用 `setInterval()` 函数可检查加载过程。该脚本在 `params.txt` 文件中检查是否存在名为 `done` 的变量。

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc.loadVariables("params.txt");
function checkParamsLoaded() {
    if (target_mc.done == undefined) {
        trace("not yet.");
    }
}
```

```

    } else {
        trace("finished loading. killing interval.");
        trace("-----");
        for (i in target_mc) {
            trace(i+": "+target_mc[i]);
        }
        trace("-----");
        clearInterval(param_interval);
    }
}
var param_interval = setInterval(checkParamsLoaded, 100);

```

**params.txt** 文件包括以下文本：

```
var1="hello"&var2="goodbye"&done="done"
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[loadMovie](#) ([MovieClip.loadMovie](#) 方法), [loadVariablesNum](#) 函数, [unloadMovie](#) ([MovieClip.unloadMovie](#) 方法)

## localToGlobal (MovieClip.localToGlobal 方法)

```
public localToGlobal(pt:Object) : Void
```

将 `pt` 对象从影片剪辑（本地）坐标转换为舞台（全局）坐标。

通过 `MovieClip.localToGlobal()` 方法，可以将任何给定的 **x** 和 **y** 坐标从相对于特定影片剪辑左上角的值转换为相对于舞台左上角的值。

必须首先创建拥有 **x** 和 **y** 这两个属性的通用对象。这些 **x** 和 **y** 值（并且它们必须被命名为 **x** 和 **y**）被称为本地坐标，因为它们是相对于影片剪辑左上角的坐标。**x** 属性表示从影片剪辑左上角的水平偏移量。换句话说，它表示点所在位置向右的距离。例如，如果 **x = 50**，则该点在左上角向右 50 像素处。**y** 属性表示从影片剪辑左上角的垂直偏移量。换句话说，它表示点所在位置向下的距离。例如，如果 **y = 20**，则点在左上角向下 20 像素处。下面的代码使用这些坐标创建一个通用对象。

```

var myPoint:Object = new Object();
myPoint.x = 50;
myPoint.y = 20;

```

或者，您也可以使用 **Object** 文本值同时创建对象并分配值。

```
var myPoint:Object = {x:50, y:20};
```

使用本地坐标创建点对象后，可以将本地坐标转换为全局坐标。`localToGlobal()` 方法不返回值，因为它更改通用对象中作为参数发送的 `x` 和 `y` 值。该方法将它们从相对于特定影片剪辑的值（本地坐标）更改为相对于舞台的值（全局坐标）。

例如，如果创建一个位于点 `(_x:100, _y:100)` 的影片剪辑，并且将表示影片剪辑左上角 `(x:10, y:10)` 附近的点的本地点传递到 `localToGlobal()` 方法，则该方法应该将 `x` 和 `y` 值转换为全局坐标（在本例中为 `(x:110, y:110)`）。之所以发生此转换，是因为 `x` 和 `y` 坐标现在是相对于舞台的左上角表示的，而不是相对于影片剪辑的左上角。

影片剪辑坐标是使用 `_x` 和 `_y` 来表示的，因为它们是用来设置 **MovieClips** 的 `x` 和 `y` 值的 **MovieClip** 属性。但是，通用对象使用不带下划线的 `x` 和 `y`。下面的代码将 `x` 和 `y` 坐标转换为全局坐标：

```
var myPoint:Object = {x:10, y:10}; // create your generic point object
this.createEmptyMovieClip("myMovieClip", this.getNextHighestDepth());
myMovieClip._x = 100; // _x for movieclip x position
myMovieClip._y = 100; // _y for movieclip y position

myMovieClip.localToGlobal(myPoint);
trace ("x: " + myPoint.x); // output: 110
trace ("y: " + myPoint.y); // output: 110
```

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 5

## 参数

`pt:Object` - 使用 **Object** 类创建的对象名称或标识符，将 `x` 和 `y` 坐标指定为属性。

## 示例

下面的示例将 `my_mc` 对象的 `x` 和 `y` 坐标从影片剪辑（本地）坐标转换为舞台（全局）坐标。在单击并拖动实例后，将反映影片剪辑的中心点。

```
this.createTextField("point_txt", this.getNextHighestDepth(), 0, 0, 100,
    22);
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    var point:Object = {x:my_mc._width/2, y:my_mc._height/2};
    my_mc.localToGlobal(point);
    point_txt.text = "x:"+point.x+", y:"+point.y;
};
Mouse.addListener(mouseListener);
my_mc.onPress = function() {
    this.startDrag();
};
my_mc.onRelease = function() {
    this.stopDrag();
};
```



此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[globalToLocal \(MovieClip.globalToLocal 方法\)](#)

## `_lockroot` (MovieClip.\_lockroot 属性)

```
public _lockroot : Boolean
```

一个布尔值，指定将 **SWF** 文件加载到影片剪辑中时 `_root` 引用的内容。默认情况下，`_lockroot` 属性为 `undefined`。您可以在正在被加载的 **SWF** 文件中或正在加载该影片剪辑的处理函数中设置此属性。

例如，假设您有一个名为 **Games.fla** 的文档，用于让用户选择要玩的游戏并将该游戏（例如 **Chess.swf**）加载到 `game_mc` 影片剪辑中。请确保，在加载到 **Games.swf** 中之后，在 **Chess.swf** 中对 `_root` 的任何使用都指向 **Chess.swf** 中的 `_root`（而非 **Games.swf** 中的 `_root`）。如果对 **Chess.fla** 具有访问权限并将其发布到 **Flash Player 7** 或更高版本，则可以将此语句添加到 **Chess.fla** 的主时间轴上：

```
this._lockroot = true;
```

如果您无法访问 **Chess.fla**（例如，如果您从其他人的站点中将 **Chess.swf** 加载到 `chess_mc`），则可以在加载时设置 **Chess.swf** `_lockroot` 属性。将下列 **ActionScript** 放在 **Games.fla** 的主时间轴上：

```
chess_mc._lockroot = true;
```

在这种情况下，只要发布了 **Flash Player 7** 或更高版本的 **Games.swf**，就可以发布任何 **Flash Player** 版本的 **Chess.swf**。

在调用 `loadMovie()` 时，可按下列代码所示在加载器影片中将 `MovieClip._lockroot` 属性设置为 `true`。如果您不在加载器影片中将 `_lockroot` 设置为 `true`，则任何对已加载影片中 `_root` 的引用都会指向加载器的 `_root`，而不是已加载影片的 `_root`：

```
myMovieClip._lockroot = true;
```

可用性：ActionScript 1.0；Flash Player 7

## 示例

在下面的示例中，**lockroot.fl**a 将 `_lockroot` 应用于主 SWF 文件。如果 SWF 文件被加载到另一 FLA 文档中，则 `_root` 始终指的是 **lockroot.swf** 的范围，这有助于防止冲突。将以下 **ActionScript** 放在 **lockroot.fl**a 的主时间轴上：

```
this._lockroot = true;
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from lockroot.swf");
for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
}
trace("");
```

它跟踪以下信息：

```
from lockroot.swf
myOtherVar -> 2
myVar -> 1
_lockroot -> true
$version -> WIN 7,0,19,0
```

下面的示例加载两个 SWF 文件，即 **lockroot.swf** 和 **nolockroot.swf**。**lockroot.fl**a 文档包含前面示例中的 **ActionScript**。**nolockroot.fl**a 文件中，下列代码被添加到时间轴的第 1 帧：

```
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from nolockroot.swf");
for (i in _root) {
    trace(" "+i+" -> "+_root[i]);
}
trace("");
```

**lockroot.swf** 文件已应用了 `_lockroot`，而 **nolockroot.swf** 则未应用。文件被加载后，每个文件将从它们的 `_root` 范围中输出值变量。将以下 **ActionScript** 放在 FLA 文档的主时间轴上：

```
this.createEmptyMovieClip("lockroot_mc", this.getNextHighestDepth());
lockroot_mc.loadMovie("lockroot.swf");
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc.loadMovie("nolockroot.swf");
function dumpRoot() {
    trace("from current SWF file");
    for (i in _root) {
        trace(" "+i+" -> "+_root[i]);
    }
    trace("");
}
dumpRoot();
```

它跟踪以下信息：

```
from current SWF file
```

```

dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myVar -> 1
i -> lockroot_mc
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from lockroot.swf
myOtherVar -> 2
myVar -> 1

```

未应用任何 `_lockroot` 的文件也包含根 **SWF** 文件所包含的所有其它变量。如果您无权访问 **nolockroot.flc**，则可以使用以下添加到主时间轴的 **ActionScript** 更改前面的主 **FLA** 文档中的 `_lockroot`：

```

this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc._lockroot = true;
nolockroot_mc.loadMovie("nolockroot.swf");

```

然后，它将跟踪以下内容：

```

from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myOtherVar -> 2
myVar -> 1

from lockroot.swf
myOtherVar -> 2
myVar -> 1

```

另请参见

[\\_root 属性](#)，[\\_lockroot \(MovieClip.\\_lockroot 属性\)](#)，[attachMovie \(MovieClip.attachMovie 方法\)](#)，[loadMovie \(MovieClip.loadMovie 方法\)](#)，[onLoadInit \(MovieClipLoader.onLoadInit 事件侦听器\)](#)

## menu (MovieClip.menu 属性)

public menu : ContextMenu

将指定的 **ContextMenu** 对象与影片剪辑相关联。**ContextMenu** 类用于修改当用户在 **Flash Player** 中右键单击（在 **Windows** 中）或按住 **Control** 键并单击（在 **Macintosh** 中）时显示的上下文菜单。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 示例

下面的示例将 **ContextMenu** 对象 menu\_cm 分配给影片剪辑 image\_mc。**ContextMenu** 对象包含标记为“在浏览器中查看图像”且具有名为 viewImage() 的关联函数的自定义菜单项。

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("View Image in Browser...",
    viewImage));
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc.menu = menu_cm;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("photo1.jpg", image_mc);

function viewImage(target_mc:MovieClip, obj:Object) {
    getURL(target_mc._url, "_blank");
}
```

当您在运行时右键单击 (**Windows**) 或按住 **Control** 键单击 (**Macintosh**) 图像时，从上下文菜单中选择“在浏览器中查看图像”，以在浏览器窗口中打开该图像。

另请参见

[menu \(Button.menu 属性\)](#), [ContextMenu](#), [ContextMenuItem](#), [menu \(TextField.menu 属性\)](#)

## moveTo (MovieClip.moveTo 方法)

```
public moveTo(x:Number, y:Number) : Void
```

将当前绘画位置移动到 (x, y)。如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**x:**Number - 一个整数，指示相对于父级影片剪辑注册点的水平位置。

**y:**Number - 一个整数，指示相对于父级影片剪辑注册点的垂直位置。

### 示例

下面的示例绘制一个三角形，它具有 5 像素的纯洋红色线条和部分透明的蓝色填充：

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

### 另请参见

[createEmptyMovieClip \(MovieClip.createEmptyMovieClip 方法\)](#), [lineStyle \(MovieClip.lineStyle 方法\)](#), [lineTo \(MovieClip.lineTo 方法\)](#)

## \_name (MovieClip.\_name 属性)

```
public _name : String
```

影片剪辑的实例名称。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

下面的示例允许您右键单击 (**Windows**) 或按住 **Control** 键并单击 (**Macintosh**) 舞台上的影片剪辑，然后从上下文菜单中选择 **Info** 以查看有关该实例的信息。添加几个具有实例名称的影片剪辑，然后将以下 **ActionScript** 添加到 **AS** 或 **FLA** 文件：

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Info...", getMCInfo));
function getMCInfo(target_mc:MovieClip, obj:Object) {
```

```

        trace("You clicked on the movie clip '"+target_mc._name+"'.");
        trace("\t width:"+target_mc._width+", height:"+target_mc._height);
        trace("");
    }
    for (var i in this) {
        if (typeof (this[i]) == 'movieclip') {
            this[i].menu = menu_cm;
        }
    }
}

```

另请参见

[\\_name \(Button.\\_name 属性\)](#)

## nextFrame (MovieClip.nextFrame 方法)

```
public nextFrame() : Void
```

将播放头转到下一帧并停止。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下列示例使用 `_framesloaded` 和 `nextFrame()` 将内容加载到 **SWF** 文件中。请不要给第 1 帧添加任何代码，但要将以下 **ActionScript** 添加到时间轴的第 2 帧：

```

if (this._framesloaded >= 3) {
    this.nextFrame();
} else {
    this.gotoAndPlay(1);
}

```

然后，在第 3 帧上添加以下代码（以及要加载的内容）：

```
stop();
```

另请参见

[nextFrame 函数](#), [prevFrame 函数](#), [prevFrame \(MovieClip.prevFrame 方法\)](#)

## onData (MovieClip.onData 处理函数)

```
onData = function() {}
```

在影片剪辑从 `MovieClip.loadVariables()` 调用或 `MovieClip.loadMovie()` 调用获得数据时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类并链接到库中的元件的类文件中定义该函数。

此处理函数只能用于 `MovieClip.loadVariables()` 方法或 `loadVariables()` 全局函数。如果希望通过 `MovieClip.loadMovie()` 方法或 `loadMovie()` 函数调用事件处理函数，则必须使用 `onClipEvent(data)` 而不是此处理函数。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下列示例显示 `MovieClip.onData()` 的正确用法。它从与 **FLA** 相同的目录中加载名为 **OnData.txt** 的文件。当文件中的数据加载到 `MovieClip` 对象中时，将执行 `onData()`，这时我们可以输出该数据。

```
var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());

mc.onData = function() {
    for(var i in this) {
        trace(">> " + i + ": " + this[i]);
    }
}

mc.loadVariables("OnData.txt");
```

另请参见

[onClipEvent 处理函数](#), [loadVariables \(MovieClip.loadVariables 方法\)](#)

## onDragOut (MovieClip.onDragOut 处理函数)

```
onDragOut = function() {}
```

当按下鼠标按钮并且指针滑出对象时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 onDragOut 方法定义一个函数，该函数将 trace() 动作发送到“输出”面板：

```
my_mc.onDragOut = function () {  
    trace ("onDragOut called");  
}
```

另请参见

[onDragOver \(MovieClip.onDragOver 处理函数\)](#)

## onDragOver (MovieClip.onDragOver 处理函数)

```
onDragOver = function() {}
```

当鼠标指针在影片剪辑外拖动并且随后拖过该影片剪辑时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 onDragOver 方法定义一个函数，该函数将 trace() 动作发送到“输出”面板：

```
my_mc.onDragOver = function () {  
    trace ("onDragOver called");  
}
```

另请参见

[onDragOut \(MovieClip.onDragOut 处理函数\)](#)



## onEnterFrame (MovieClip.onEnterFrame 处理函数)

```
onEnterFrame = function() {}
```

以 SWF 文件的帧频重复调用。分配给 onEnterFrame 事件处理函数的函数将在附加到受影响的帧上的所有 **ActionScript** 代码之前处理。

必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例为 onEnterFrame 事件处理函数定义一个函数，该函数将 trace() 动作发送到“输出”面板：

```
my_mc.onEnterFrame = function () {  
    trace ("onEnterFrame called");  
}
```

## onKeyDown (MovieClip.onKeyDown 处理函数)

```
onKeyDown = function() {}
```

当影片剪辑具有输入焦点并且用户按下某个键时调用。调用 onKeyDown 事件处理函数时无需使用参数。您可以使用 Key.getAscii() 和 Key.getCode() 方法来确定按下了哪个键。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

仅当影片剪辑的输入焦点被启用和设置后，onKeyDown 事件处理函数才能起作用。首先，必须将影片剪辑的 MovieClip.focusEnabled 属性设置为 true。然后，必须使该剪辑获得焦点。可以通过使用 Selection.setFocus() 或通过设置 **Tab** 键以导航到影片剪辑，以实现此操作。

如果使用 Selection.setFocus()，则必须将影片剪辑的路径传递给 Selection.setFocus()。对于其它元素，在用户移动鼠标后取回焦点是很容易的。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 示例

下列示例为 `onKeyDown()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：创建一个名为 **my\_mc** 的影片剪辑，并将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
my_mc.onKeyDown = function () {  
    trace ("key was pressed");  
}
```

影片剪辑必须具有焦点，`onKeyDown` 事件处理函数才能起作用。添加以下 **ActionScript** 以设置输入焦点：

```
my_mc.tabEnabled = true;  
my_mc.focusEnabled = true;  
Selection.setFocus(my_mc);
```

用户按下一个键后，`key was pressed` 将显示在“输出”面板上。但是，移动鼠标后不会出现此情况，因为影片剪辑失去了焦点。因此，应该在大多数情况下使用 `Key.onKeyDown`。

## 另请参见

[getAscii \(Key.getAscii 方法\)](#)，[getCode \(Key.getCode 方法\)](#)，[onKeyDown \(Key.onKeyDown 事件侦听器\)](#)，[focusEnabled \(MovieClip.focusEnabled 属性\)](#)，[onKeyUp \(MovieClip.onKeyUp 处理函数\)](#)，[setFocus \(Selection.setFocus 方法\)](#)

## onKeyUp (MovieClip.onKeyUp 处理函数)

```
onKeyUp = function() {}
```

当释放按键时调用。调用 `onKeyUp` 事件处理函数时无需使用参数。您可以使用 `Key.getAscii()` 和 `Key.getCode()` 方法来确定用户按下了哪个键。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

仅当影片剪辑的输入焦点被启用和设置后，`onKeyUp` 事件处理函数才能起作用。首先，必须将影片剪辑的 `MovieClip.focusEnabled` 属性设置为 `true`。然后，必须使该影片剪辑获得焦点。可以通过使用 `Selection.setFocus()` 或通过设置 **Tab** 键以导航到影片剪辑，以实现此操作。

如果使用 `Selection.setFocus()`，则必须将影片剪辑的路径传递给 `Selection.setFocus()`。对于其它元素，在用户移动鼠标后取回焦点是很容易的。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下列示例为 `onKeyUp` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onKeyUp = function () {  
    trace ("onKey called");  
}
```

以下示例设置输入焦点：

```
my_mc.focusEnabled = true;  
Selection.setFocus(my_mc);
```

## 另请参见

[getAscii \(Key.getAscii 方法\)](#)，[getCode \(Key.getCode 方法\)](#)，[onKeyDown \(Key.onKeyDown 事件侦听器\)](#)，[focusEnabled \(MovieClip.focusEnabled 属性\)](#)，[onKeyDown \(MovieClip.onKeyDown 处理函数\)](#)，[setFocus \(Selection.setFocus 方法\)](#)

## onKillFocus (MovieClip.onKillFocus 处理函数)

```
onKillFocus = function(newFocus:Object) {}
```

当影片剪辑失去键盘焦点时调用。`onKillFocus` 方法接收一个参数 `newFocus`，该参数是一个对象，表示要获得焦点的新对象。如果没有对象接收焦点，则 `newFocus` 将包含值 `null`。

必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

**可用性：** **ActionScript 1.0**； **Flash Player 6**

## 参数

**newFocus:Object** - 要接收键盘焦点的对象。

## 示例

下列示例报告有关失去焦点的影片剪辑以及当前具有焦点的实例的信息。在舞台上有两个影片剪辑，名为 `my_mc` 和 `other_mc`。请将以下 **ActionScript** 添加到 **AS** 或 **FLA** 文档：

```
my_mc.onRelease = Void;  
other_mc.onRelease = Void;  
my_mc.onKillFocus = function(newFocus) {  
    trace("onKillFocus called, new focus is: "+newFocus);  
};
```

在两个实例之间切换，而且信息会显示在“输出”面板中。

## 另请参见

[onSetFocus \(MovieClip.onSetFocus 处理函数\)](#)

## onLoad (MovieClip.onLoad 处理函数)

```
onLoad = function() {}
```

当影片剪辑被实例化并显示在时间轴上时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

此处理函数只能与某些影片剪辑一起使用，对于这些影片剪辑，它们在库中有与类关联的元件。如果要在加载特定影片剪辑时（例如当使用 `MovieClip.loadMovie()` 动态加载 SWF 文件时）调用某个事件处理函数，则必须使用 `onClipEvent(load)` 或 **MovieClipLoader** 类而不是此处理函数。与 `MovieClip.onLoad` 不同，其它处理函数可以在加载任何影片剪辑时调用。

可用性：ActionScript 1.0；Flash Player 6

### 示例

此示例显示如何在扩展 **MovieClip** 类的 **ActionScript 2.0** 类定义中使用 `onLoad` 事件处理函数。首先，创建名为 **Oval.as** 的类文件并定义名为 `onLoad()` 的类方法。然后，确保按下例示例所示将类文件放置到正确的类路径上：

```
// contents of Oval.as
class Oval extends MovieClip{
    public function onLoad () {
        trace ("onLoad called");
    }
}
```

第二步，在库中创建一个影片剪辑元件，并将其命名为 **Oval**。右键单击“库”面板中的元件，然后从弹出菜单中选择“链接...”。单击“为 ActionScript 导出”选项并在“标识符”和“ActionScript 2.0 类”字段中输入 `Oval`。保持选中“在第一帧导出”，然后单击“确定”。

第三步，转到文件的第一帧，然后在“动作”面板中输入以下代码：

```
var myOval:Oval = Oval(attachMovie("Oval","Oval_1",1));
```

最后，制作一个测试影片，您应该看到输出文本“onLoad called”。

另请参见

[loadMovie \(MovieClip.loadMovie 方法\)](#)，[onClipEvent 处理函数](#)，[MovieClipLoader](#)

## onMouseDown (MovieClip.onMouseDown 处理函数)

```
onMouseDown = function() {}
```

当按下鼠标按钮时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下列示例为 `onMouseDown()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onMouseDown = function () {  
    trace ("onMouseDown called");  
}
```

## onMouseMove (MovieClip.onMouseMove 处理函数)

```
onMouseMove = function() {}
```

当鼠标移动时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下列示例为 `onMouseMove()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onMouseMove = function () {  
    trace ("onMouseMove called");  
}
```

## onMouseUp (MovieClip.onMouseUp 处理函数)

```
onMouseUp = function() {}
```

释放鼠标按钮时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下列示例为 `onMouseUp()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onMouseUp = function () {  
    trace ("onMouseUp called");  
}
```

## onPress (MovieClip.onPress 处理函数)

```
onPress = function() {}
```

当鼠标指针处于影片剪辑之上而用户单击鼠标时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在库中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 `onPress()` 方法定义一个函数, 该函数将 `trace()` 动作发送到“输出”面板:

```
my_mc.onPress = function () {  
    trace ("onPress called");  
}
```

## onRelease (MovieClip.onRelease 处理函数)

```
onRelease = function() {}
```

当用户在影片剪辑上释放鼠标按钮时调用。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数, 也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 `onRelease()` 方法定义一个函数, 该函数将 `trace()` 动作发送到“输出”面板:

```
my_mc.onRelease = function () {  
    trace ("onRelease called");  
}
```

## onReleaseOutside (MovieClip.onReleaseOutside 处理函数)

```
onReleaseOutside = function() {}
```

用户在影片剪辑区域中按下鼠标按钮并且在影片剪辑区域之外释放它后调用。

必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 `onReleaseOutside()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onReleaseOutside = function () {  
    trace ("onReleaseOutside called");  
}
```

## onRollOut (MovieClip.onRollOut 处理函数)

```
onRollOut = function() {}
```

当鼠标指针移到影片剪辑区域的外面时调用。

必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 `onRollOut()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onRollOut = function () {  
    trace ("onRollOut called");  
}
```

## onRollOver (MovieClip.onRollOver 处理函数)

```
onRollOver = function() {}
```

当鼠标指针滑过影片剪辑区域时调用。

必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下列示例为 `onRollOver()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onRollOver = function () {  
    trace ("onRollOver called");  
}
```

## onSetFocus (MovieClip.onSetFocus 处理函数)

```
onSetFocus = function(oldFocus:Object) {}
```

当影片剪辑获得键盘焦点时调用。`oldFocus` 参数是失去焦点的对象。例如，如果用户按 **Tab** 键将输入焦点从影片剪辑移到文本字段，则 `oldFocus` 将包含此影片剪辑实例。

如果以前没有具有焦点的对象，则 `oldFocus` 包含一个 **null** 值。

必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 **MovieClip** 类或链接到库中的元件的类文件中定义该函数。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

`oldFocus:Object` - 将失去焦点的对象。

### 示例

下面的示例显示有关接收键盘焦点的影片剪辑和以前具有焦点的实例的信息。在舞台上有两个影片剪辑，名为 `my_mc` 和 `other_mc`。请将以下 **ActionScript** 添加到 **AS** 或 **FLA** 文档：

```
my_mc.onRelease = Void;  
other_mc.onRelease = Void;  
my_mc.onSetFocus = function(oldFocus) {  
    trace("onSetFocus called, previous focus was: "+oldFocus);  
}
```

在两个实例之间切换，而且信息会显示在“输出”面板中。

### 另请参见

[onKillFocus \(MovieClip.onKillFocus 处理函数\)](#)



## onUnload (MovieClip.onUnload 处理函数)

`onUnload = function() {}`

从时间轴删除影片剪辑后，在第 1 帧中调用。Flash 先处理与 `onUnload` 事件处理函数关联的动作，然后将所有动作附加到受影响的帧。必须定义一个在调用事件处理函数时执行的函数。您可以在时间轴上定义该函数，也可以在扩展 `MovieClip` 类或链接到库中的元件的类文件中定义该函数。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下列示例为 `MovieClip.onUnload()` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_mc.onUnload = function () {  
    trace ("onUnload called");  
}
```

## opaqueBackground (MovieClip.opaqueBackground 属性)

`public opaqueBackground : Number`

由数字（RGB 十六进制值）指定的颜色的影片剪辑的不透明背景颜色。如果该值为 `null` 或 `undefined`，则没有不透明的背景。对于 `cacheAsBitmap` 属性设置为 `true` 的影片剪辑，设置 `opaqueBackground` 可以提高呈现性能。

如果未设置 `opaqueBackground`，则对具有许多透明区域的影片剪辑的性能有更大的提高。

**注意：**在 `shapeFlag` 参数设置为 `true` 的 `hitTest()` 方法中，不匹配不透明背景区域。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下列示例创建三角形轮廓，并将 `opaqueBackground` 属性设置为特定颜色：

```
var triangle:MovieClip = this.createEmptyMovieClip("triangle",  
    this.getNextHighestDepth());  
triangle._x = triangle._y = 50;  
triangle.lineStyle(3, 0xFFCC00);  
triangle.lineTo(0, 30);  
triangle.lineTo(50, 0);  
triangle.lineTo(0, 0);  
triangle.endFill();  
triangle.opaqueBackground = 0xCCCCCC;
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 `DepthManager` 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[cacheAsBitmap \(MovieClip.cacheAsBitmap 属性\)](#)，[hitTest \(MovieClip.hitTest 方法\)](#)

## `_parent` (MovieClip.\_parent 属性)

```
public _parent : MovieClip
```

对包含当前影片剪辑或对象的影片剪辑或对象的引用。当前对象是引用 `_parent` 属性的对象。使用 `_parent` 属性可以指定一个相对路径，该路径指向当前影片剪辑或对象之上的影片剪辑或对象。

可以使用 `_parent` 在显示列表中上移多个级别，如下列代码所示：

```
this._parent._parent._alpha = 20;
```

可用性：ActionScript 1.0；Flash Player 5

### 示例

下列示例跟踪对影片剪辑的引用以及其父时间轴。创建具有实例名称 `my_mc` 的影片剪辑，然后将它添加到主时间轴。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
my_mc.onRelease = function() {  
    trace("You clicked the movie clip: "+this);  
    trace("The parent of "+this._name+" is: "+this._parent);  
}
```

单击影片剪辑时，下列信息将显示在“输出”面板中：

```
You clicked the movie clip: _level0.my_mc  
The parent of my_mc is: _level0
```

另请参见

[\\_parent \(Button.\\_parent 属性\)](#)，[\\_root 属性](#)，[targetPath 函数](#)，[\\_parent \(TextField.\\_parent 属性\)](#)

## play (MovieClip.play 方法)

```
public play() : Void
```

在影片剪辑的时间轴中移动播放头。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

使用以下 **ActionScript** 播放 SWF 文件的主时间轴。此 **ActionScript** 用于主时间轴上名为 my\_mc 的影片剪辑按钮:

```
stop();
my_mc.onRelease = function() {
    this._parent.play();
};
```

使用以下 **ActionScript** 播放 SWF 文件中影片剪辑的时间轴。此 **ActionScript** 用于主时间轴上名为 my\_btn 的按钮 (用于播放名为 animation\_mc 的影片剪辑):

```
animation_mc.stop();
my_btn.onRelease = function(){
    animation_mc.play();
};
```

### 另请参见

[play 函数](#), [gotoAndPlay \(MovieClip.gotoAndPlay 方法\)](#), [gotoAndPlay 函数](#)

## prevFrame (MovieClip.prevFrame 方法)

```
public prevFrame() : Void
```

将播放头转到前一帧并停止。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

在下列示例中, 两个影片剪辑按钮控制着时间轴。prev\_mc 按钮将播放头移动到前一帧, next\_mc 按钮将播放头移动到下一帧。将内容添加到时间轴上的一系列帧, 然后将下列 **ActionScript** 添加到时间轴的第 1 帧:

```
stop();
prev_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe>1) {
        parent_mc.prevFrame();
    }
};
```

```
    } else {
        parent_mc.gotoAndStop(parent_mc._totalframes);
    }
};
next_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe<parent_mc._totalframes) {
        parent_mc.nextFrame();
    } else {
        parent_mc.gotoAndStop(1);
    }
};
```

另请参见  
[prevFrame 函数](#)

## \_quality (MovieClip.\_quality 属性)

public \_quality : String

设置或检索用于 SWF 文件的呈现品质。设备字体始终带有锯齿，因此不受 \_quality 属性的影响。

可以将 \_quality 设置为下列值：

值	说明	图形消除锯齿	位图平滑处理
"LOW"	低呈现品质。	图形未消除锯齿。	位图未进行平滑处理。
"MEDIUM"	中等呈现品质。此设置适用于不包含文本的影片。	图形使用 2x2 像素的网格消除锯齿。	Flash Player 8：位图基于 MovieClip.attachBitmap() 和 MovieClip.beginBitmapFill() 调用中使用的 smoothing 参数进行平滑处理。 Flash Player 6 和 7：位图未进行平滑处理。

值	说明	图形消除锯齿	位图平滑处理
"HIGH"	高呈现品质。此设置是 Flash 使用的默认呈现品质设置。	图形使用 4 x 4 像素的网格消除锯齿。	Flash Player 8: 位图基于 MovieClip.attachBitmap() 和 MovieClip.beginBitmapFill() 调用中使用的 smoothing 参数进行平滑处理。 Flash Player 6 和 7: 如果影片剪辑为静态的, 则对位图进行平滑处理。
"BEST"	极高呈现品质。	图形使用 4 x 4 像素的网格消除锯齿。	Flash Player 8: 位图基于 MovieClip.attachBitmap() 和 MovieClip.beginBitmapFill() 调用中使用的 smoothing 参数进行平滑处理。设置 smoothing 后, 通过使用平均算法, 随着影片剪辑的缩小, 呈现结果的品质越高。这会减慢呈现速度, 但它适用于生成大图像的高质量缩略图的应用程序。 Flash Player 6 和 7: 位图始终进行平滑处理。

注意: 尽管您可以为 **MovieClip** 对象指定此属性, 但它实际上是一个全局属性, 因此可以简单地将它的值指定为 `_quality`。

可用性: **ActionScript 1.0** : **Flash Player 6**

### 示例

此示例将名为 `my_mc` 的影片剪辑的呈现品质设置为 `LOW`:

```
my_mc._quality = "LOW";
```

另请参见

[\\_quality 属性](#)

# removeMovieClip (MovieClip.removeMovieClip 方法)

public removeMovieClip() : Void

删除用 duplicateMovieClip()、MovieClip.duplicateMovieClip()、MovieClip.createEmptyMovieClip() 或 MovieClip.attachMovie() 创建的影片剪辑实例。此方法不删除分配给负深度值的影片剪辑。默认情况下会给出在创作工具中创建的影片剪辑分配负深度值。若要删除分配给负深度值的影片剪辑，请首先使用 MovieClip.swapDepths() 方法将影片剪辑移动到正深度值。

**提醒** 如果使用第 2 版的组件，则不要使用此方法。如果您在舞台上或库中放置了第 2 版的版组件，getNextHighestDepth() 方法有时会返回深度 1048676，它超出了有效范围。如果使用第 2 版的组件，则应始终使用第 2 版的组件 DepthManager 类。

**提醒** 如果正使用第 2 版的组件，并且使用 MovieClip.getNextHighestDepth() 而不是第 2 版的组件 DepthManager 类分配深度值，则您可能会发现 removeMovieClip() 将失败且无任何提示。在使用任何第 2 版的版组件时，DepthManager 类都将自动保留光标和工具提示的最高 (1048575) 和最低 (-16383) 可用深度。对 getNextHighestDepth() 的后续调用返回 1048576，这超出了有效范围。如果遇到有效范围之外的深度值，则 removeMovieClip() 方法将失败且无任何提示。如果必须对第 2 版的组件使用 getNextHighestDepth()，则可以使用 swapDepths() 分配有效的深度值或使用 MovieClip.unloadMovie() 删除影片剪辑的内容。或者，您也可以使用 DepthManager 类来分配有效范围内的深度值。

您可以通过创建子类来扩展 MovieClip 类的方法和事件处理函数。

可用性: ActionScript 1.0 ; Flash Player 5

## 示例

在下面的示例中每次单击按钮时，都会将影片剪辑实例附加到舞台上的随机位置。单击影片剪辑实例时，将从 SWF 文件中删除该实例。

```
function randRange(min:Number, max:Number):Number {
    var randNum:Number = Math.round(Math.random()*(max-min))+min;
    return randNum;
}
var bugNum:Number = 0;
addBug_btn.onRelease = addBug;
function addBug() {
    var thisBug:MovieClip = this._parent.attachMovie("bug_id",
        "bug"+bugNum+"_mc", bugNum,
        {_x:randRange(50, 500), _y:randRange(50, 350)});
    thisBug.onRelease = function() {
        this.removeMovieClip();
    };
    bugNum++;
}
```

另请参见

[duplicateMovieClip 函数](#), [createEmptyMovieClip \(MovieClip.createEmptyMovieClip 方法\)](#), [duplicateMovieClip \(MovieClip.duplicateMovieClip 方法\)](#), [attachMovie \(MovieClip.attachMovie 方法\)](#), [swapDepths \(MovieClip.swapDepths 方法\)](#)

## `_rotation` (MovieClip.\_rotation 属性)

`public _rotation : Number`

指定影片剪辑相对于其原始方向的旋转程度，以度为单位。从 **0** 到 **180** 的值表示顺时针方向旋转；从 **0** 到 **-180** 的值表示逆时针方向旋转。对于超出此范围的值，可通过加上或减去 **360** 获得该范围内的值；例如，语句 `my_mc._rotation = 450` 和语句 `my_mc._rotation = 90` 相同。

可用性：ActionScript 1.0；Flash Player 4

### 示例

下列示例将动态创建 `triangle` 影片剪辑实例。运行 SWF 文件时，单击影片剪辑可旋转它。

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());
```

```
triangle.beginFill(0x0000FF, 100);  
triangle.moveTo(100, 100);  
triangle.lineTo(100, 150);  
triangle.lineTo(150, 100);  
triangle.lineTo(100, 100);
```

```
triangle.onMouseUp= function() {  
    this._rotation += 15;  
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[\\_rotation \(Button.\\_rotation 属性\)](#), [\\_rotation \(TextField.\\_rotation 属性\)](#)

## scale9Grid (MovieClip.scale9Grid 属性)

```
public scale9Grid : Rectangle
```

矩形区域，它定义影片剪辑的九个缩放区域。如果设置为 null，则在应用任何缩放转换时，将正常缩放整个影片剪辑。

如果为影片剪辑定义了 scale9Grid 属性，则根据 scale9Grid 矩形（用于定义网格的中心区域）将影片剪辑划分为包含九个区域的网格。网格包含八个其它区域：

- 矩形外部左上角的区域
- 矩形上方的区域
- 矩形外部右上角的区域
- 矩形左侧的区域
- 矩形右侧的区域
- 矩形外部左下角的区域
- 矩形下方的区域
- 矩形外部右下角的区域

可以认为中心区域（由矩形定义）之外的八个区域类似于在缩放时已应用特殊规则的图片帧。

在设置 scale9Grid 属性并缩放影片剪辑后，对于所有文本和子级影片剪辑，无论它们位于 scale9 网格的哪些区域中，都可以正常缩放；但是，对于其它对象类型，则应用以下规则：

- 中心区域中的所有内容正常缩放。
- 中心区域缩放到 0 时，仅缩放转角中的任何内容。
- 仅水平缩放顶部和底部区域中的任何内容。仅垂直缩放左侧区域和右侧区域中的内容。
- 拉伸所有填充（包括位图、视频和渐变）以适合其形状。

如果旋转影片剪辑，则所有后续缩放都是正常缩放（而且忽略 scale9Grid 属性）。

例如，请考虑以下影片剪辑和作为该影片剪辑的 scale9Grid 属性应用的矩形：





缩放或拉伸影片剪辑时，矩形内的对象正常缩放，但是矩形外的对象则按照 scale9Grid 规则进行缩放：

缩放到 75%:	
缩放到 50%:	
缩放到 25%:	
水平拉伸 150%:	

设置 scale9Grid 的常见用法是设置组件，当缩放该组件时，其中的边缘线条保持相同的宽度。

在 **Macromedia Flash** 创作环境中，可以对库中的影片剪辑元件启用“9 切片缩放”的辅助线。这样就可用图形化的方式确定对象的 scale9grid。为元件设置 9 切片缩放时，将自动设置该元件的任何实例的 scale9grid 属性。对于启用 9 切片缩放的元件，创建 SWF 文件时，任何跨越 9 切片缩放网格中多个区域的曲线将被划分为每个网格区域内单独的曲线。例如，请考虑启用 9 切片缩放的影片剪辑元件中的曲线，以及未启用 9 切片缩放的影片剪辑元件中的相同的曲线：

启用 9 切片缩放的元件：	
未启用 9 切片缩放的元件:	

Flash 创建 SWF 文件时，第一个影片剪辑中的曲线显示了被划分为三条曲线。而第二个影片剪辑（未启用 9 切片缩放）中的曲线则不是这样。即使将第二个影片剪辑的 `scale9Grid` 设置为与第一个影片剪辑的 `scale9Grid` 相匹配的矩形，在缩放这些影片剪辑时，结果仍将不同，其原因是 Flash 划分第一个影片剪辑中的曲线的方法：

启用 9 切片缩放的元件	
未启用 9 切片缩放的元件	

可用性：ActionScript 1.0：Flash Player 8

示例

下面创建一个影片剪辑，它包含一个 20 像素行（它构成边框）和渐变填充。影片剪辑根据鼠标位置进行缩放，由于为影片剪辑设置了 `scale9Grid`，因此在剪辑缩放时 20 像素行的粗细保持不变（虽然影片剪辑中的渐变确实在进行缩放）：

```
import flash.geom.Rectangle;
import flash.geom.Matrix;

this.createEmptyMovieClip("my_mc", this.getNextHighestDepth());

var grid:Rectangle = new Rectangle(20, 20, 260, 260);
my_mc.scale9Grid = grid ;

my_mc._x = 50;
my_mc._y = 50;

function onMouseMove()
{
    my_mc._width = _xmouse;
    my_mc._height = _ymouse;
}

my_mc.lineStyle(20, 0xff3333, 100);
var gradient_matrix:Matrix = new Matrix();
gradient_matrix.createGradientBox(15, 15, Math.PI, 10, 10);
my_mc.beginGradientFill("radial", [0xffff00, 0x0000ff],
    [100, 100], [0, 0xFF], gradient_matrix,
    "reflect", "RGB", 0.9);
my_mc.moveTo(0, 0);
my_mc.lineTo(0, 300);
my_mc.lineTo(300, 300);
my_mc.lineTo(300, 0);
my_mc.lineTo(0, 0);
my_mc.endFill();
```

另请参见

[Rectangle \(flash.geom.Rectangle\)](#)

## scrollRect (MovieClip.scrollRect 属性)

`public scrollRect : Object`

通过 `scrollRect` 属性，可以快速滚动影片剪辑内容，并具有一个用来查看较大内容的窗口。文本字段和复杂内容的滚动要快得多，这是因为使用像素级复制来滚动数据，而不是必须从矢量数据重新生成整个影片剪辑。若要再次查看性能，可将 `scrollRect` 与 `cacheAsBitmap` 设置为 `true` 的影片剪辑配合使用。

按特定宽度、高度和滚动偏移量裁切和滚动影片剪辑。`scrollRect` 属性存储在影片剪辑的坐标空间中，并且就像整个影片剪辑那样缩放。滚动影片剪辑上已裁切窗口的转角范围是影片剪辑的原点 `(0, 0)` 和 `(scrollWidth, scrollHeight)` 点。它们不按原点居中，而是使用位于左上角的原点。滚动的影片剪辑始终按整像素增量滚动。如果将影片剪辑旋转 **90 度** 并使它左右滚动（通过设置 `scrollRect.x` 属性），则它将上下滚动。

如果设置为 `flash.geom.Rectangle` 对象，则将此影片剪辑裁切到一定的大小并滚动它。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例设置 `MovieClip` 层次结构（通过调用 `setUpContainer()` 函数），然后将新矩形设置为 `scrollRect` 属性。

```
import flash.geom.Rectangle;
var container:MovieClip = setUpContainer();
var window:Rectangle = new Rectangle(0, 0, 100, 40);
container.scrollRect = window;

function setUpContainer():MovieClip {
    var mc:MovieClip = this.createEmptyMovieClip("container",
        this.getNextHighestDepth());
    mc._x = 50;
    mc._y = 50;
    mc.opaqueBackground = 0xCCCCCC;

    var content:MovieClip = mc.createEmptyMovieClip("content",
        mc.getNextHighestDepth());
    var colors:Array = [0xFF0000, 0x0000FF];
    var alphas:Array = [100, 100];
    var ratios:Array = [0, 0xFF];
    var matrix:Object = {a:150, b:0, c:0, d:0, e:150, f:0, g:150, h:150, i:1};
    content.beginGradientFill("linear", colors, alphas, ratios, matrix);
    content.lineTo(300, 0);
    content.lineTo(300, 300);
}
```

```

        content.lineTo(0, 300);
        content.lineTo(0, 0);
        content.endFill();
        content._rotation = -90;

        mc.onEnterFrame = function() {
            this.content._y += 1;
        }

        return mc;
    }
}

```

setUpContainer() 函数执行下列步骤：

- 创建名为 container 的 MovieClip
- 在 container 内创建名为 content 的 MovieClip
- 在 MovieClip content 内绘制一个渐变形状
- 返回对 MovieClip container 的引用

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 DepthManager 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

## setMask (MovieClip.setMask 方法)

```
public setMask(mc:Object) : Void
```

使参数 mc 中的影片剪辑成为展示调用影片剪辑的遮罩层。

setMask() 方法允许具有复杂、多层内容的多帧影片剪辑充当遮罩（通过使用遮罩层，此操作是可行的）。如果在使用遮罩的影片剪辑中具有设备字体，则它们可以进行绘制但不能被遮罩。您不能将影片剪辑设置为它自己的遮罩，例如 my\_mc.setMask(my\_mc)。

如果创建包含影片剪辑的遮罩层，然后对其应用 setMask() 方法，则优先调用 setMask()，并且这种调用是不可逆转的。例如，在名为 UIMask 的遮罩层中有一个影片剪辑，该遮罩层对另一个包含名为 UIMaskee 的影片剪辑遮罩层进行遮罩。如果在 SWF 文件播放时调用 UIMask.setMask(UIMaskee)，从这时起，UIMask 将由 UIMaskee 遮罩。

若要取消用 **ActionScript** 创建的遮罩，请向 setMask() 方法传递值 null。下列代码可以取消遮罩而不影响时间轴中的遮罩层。

```
UIMask.setMask(null);
```

您可以通过创建子类来扩展 MovieClip 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**mc:Object** - 将成为遮罩的影片剪辑的实例名称。此项可以是 **String** 或 **MovieClip**。

## 示例

下列代码使用 `circleMask_mc` 影片剪辑遮罩 `theMaskee_mc` 影片剪辑：

```
theMaskee_mc.setMask(circleMask_mc);
```

## `_soundbuftime` (`MovieClip._soundbuftime` 属性)

```
public _soundbuftime : Number
```

指定在声音开始进入流之前，预先缓冲的秒数。

**注意：**尽管您可以为 **MovieClip** 对象指定此属性，但它实际上是一个应用于所有已加载声音的全局属性，因此可以简单地将其值指定为 `_soundbuftime`。为 **MovieClip** 对象设置此属性实际上是设置全局属性。

**可用性：** **ActionScript 1.0**； **Flash Player 6**

另请参见

[\\_soundbuftime 属性](#)

## `startDrag` (`MovieClip.startDrag` 方法)

```
public startDrag([lockCenter:Boolean], [left:Number], [top:Number],  
                [right:Number], [bottom:Number]) : Void
```

允许用户拖动指定的影片剪辑。该影片剪辑将一直保持可拖动，直到通过对 `MovieClip.stopDrag()` 的调用明确停止为止，或者直到另一个影片剪辑变为可拖动为止。在同一时间只有一个影片剪辑是可拖动的。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

**可用性：** **ActionScript 1.0**； **Flash Player 5**

## 参数

**lockCenter:Boolean** [ 可选 ] - 一个布尔值，指定可拖动影片剪辑是锁定到鼠标位置中央 (`true`)，还是锁定到用户首次单击该影片剪辑的位置上 (`false`)。

**left:Number** [ 可选 ] - 相对于该影片剪辑父级的坐标的值，该值指定该影片剪辑的约束矩形。

**top:Number** [ 可选 ] - 相对于该影片剪辑父级的坐标的值，该值指定该影片剪辑的约束矩形。

**right:Number** [ 可选 ] - 相对于该影片剪辑父级的坐标的值，该值指定该影片剪辑的约束矩形。

**bottom:Number** [ 可选 ] - 相对于该影片剪辑父级的坐标的值，该值指定该影片剪辑的约束矩形。

## 示例

下列示例创建一个名为 mc\_1 的可拖动影片剪辑实例。

```
this.createEmptyMovieClip("mc_1", 1);
```

```
with (mc_1) {  
    lineStyle(1, 0xCCCCC);  
    beginFill(0x4827CF);  
    moveTo(0, 0);  
    lineTo(80, 0);  
    lineTo(80, 60);  
    lineTo(0, 60);  
    lineTo(0, 0);  
    endFill();  
}  
  
mc_1.onPress = function() {  
    this.startDrag();  
};  
mc_1.onRelease = function() {  
    this.stopDrag();  
};
```

另请参见

[\\_droptarget](#) ([MovieClip.\\_droptarget](#) 属性), [startDrag](#) 函数, [stopDrag](#) ([MovieClip.stopDrag](#) 方法)

## stop (MovieClip.stop 方法)

```
public stop() : Void
```

停止当前正在播放的影片剪辑。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0 ; Flash Player 5**

## 示例

下列示例显示如何停止名为 aMovieClip 的影片剪辑:

```
aMovieClip.stop();
```

另请参见

[stop](#) 函数

## stopDrag (MovieClip.stopDrag 方法)

```
public stopDrag() : Void
```

结束 `MovieClip.startDrag()` 方法。在添加 `stopDrag()` 方法之前, 或在另一个影片剪辑变为可拖动之前, 通过该方法变为可拖动的影片剪辑将一直保持可拖动状态。在同一时间只有一个影片剪辑是可拖动的。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下列示例创建一个名为 `mc_1` 的可拖动影片剪辑实例。

```
this.createEmptyMovieClip("mc_1", 1);
```

```
with (mc_1) {  
    lineStyle(1, 0xCCCCCC);  
    beginFill(0x4827CF);  
    moveTo(0, 0);  
    lineTo(80, 0);  
    lineTo(80, 60);  
    lineTo(0, 60);  
    lineTo(0, 0);  
    endFill();  
}  
  
mc_1.onPress = function() {  
    this.startDrag();  
};  
mc_1.onRelease = function() {  
    this.stopDrag();  
};
```

### 另请参见

[\\_droptarget \(MovieClip.\\_droptarget 属性\)](#), [startDrag \(MovieClip.startDrag 方法\)](#), [stopDrag 函数](#)

## swapDepths (MovieClip.swapDepths 方法)

`public swapDepths(target:Object) : Void`

交换此影片剪辑与另一影片剪辑的堆栈或深度级别（z- 顺序），另一影片剪辑由 `target` 参数指定，或指定为当前占用由 `target` 参数指定的深度级别的影片剪辑。两个影片剪辑必须具有相同的父级影片剪辑。交换影片剪辑的深度级别的作用是将一个影片剪辑移到另一个影片剪辑的前面或后面。如果调用该方法时影片剪辑正在补间，则补间会停止。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**target:Object** - 此参数可以采用两种格式之一：

- 一个数字，指定要将影片剪辑放置到的深度级别。
- 一个字符串，指定一个影片剪辑实例，应用此方法的影片剪辑将与其交换深度。两个影片剪辑必须具有相同的父级影片剪辑。

### 示例

下面的示例交换两个影片剪辑实例的堆叠顺序。在舞台上重叠名为 `myMC1_mc` 和 `myMC2_mc` 的两个影片剪辑实例，然后将下列脚本添加到父时间轴上：

```
myMC1_mc.onRelease = function() {  
    this.swapDepths(myMC2_mc);  
};  
myMC2_mc.onRelease = function() {  
    this.swapDepths(myMC1_mc);  
};
```

### 另请参见

[\\_level 属性](#)，[getDepth \(MovieClip.getDepth 方法\)](#)，[getInstanceAtDepth \(MovieClip.getInstanceAtDepth 方法\)](#)，[getNextHighestDepth \(MovieClip.getNextHighestDepth 方法\)](#)



## tabChildren (MovieClip.tabChildren 属性)

public tabChildren : Boolean

确定影片剪辑的子级是否包括在 **Tab** 键的自动排序中。如果 tabChildren 属性为 undefined 或 true，则影片剪辑的子级包括在 **Tab** 键的自动排序中。如果 tabChildren 的值为 false，则影片剪辑的子级不包括在 **Tab** 键的自动排序中。默认值为 undefined。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

列表框用户界面窗口小部件，构建为包含几个项目的影片剪辑。用户可以单击每个项目以选择它，所以每个项目被实现为一个按钮。但是，只有列表框自身应为 **Tab** 键停靠位。列表框内部的项目应该排除在 **Tab** 键排序之外。为此，可将列表框的 tabChildren 属性设置为 false。

如果使用 tabIndex 属性，则 tabChildren 属性不起作用； tabChildren 属性只影响 **Tab** 键的自动排序。

下列示例禁用所有子级影片剪辑在名为 menu\_mc 的父级影片剪辑内的 **Tab** 键排序：

```
menu_mc.onRelease = function(){};
menu_mc.menu1_mc.onRelease = function(){};
menu_mc.menu2_mc.onRelease = function(){};
menu_mc.menu3_mc.onRelease = function(){};
menu_mc.menu4_mc.onRelease = function(){};
```

```
menu_mc.tabChildren = false;
```

将代码的最后一行更改为以下内容，以便将 menu\_mc 的子级影片剪辑实例包括在 **Tab** 键的自动排序中：

```
menu_mc.tabChildren = true;
```

### 另请参见

[tabIndex \(Button.tabIndex 属性\)](#) , [tabEnabled \(MovieClip.tabEnabled 属性\)](#) , [tabIndex \(MovieClip.tabIndex 属性\)](#) , [tabIndex \(TextField.tabIndex 属性\)](#)

## tabEnabled (MovieClip.tabEnabled 属性)

```
public tabEnabled : Boolean
```

指定影片剪辑是否包括在 **Tab** 键的自动排序中。它默认情况下为 `undefined`。

如果 `tabEnabled` 属性为 `undefined`，则只有当对象定义了至少一个影片剪辑处理函数（例如 `MovieClip.onRelease`），它才被包含在 **Tab** 键自动排序中。如果 `tabEnabled` 为 `true`，则该对象包括在 **Tab** 键的自动排序中。如果 `tabIndex` 属性也设置为某个值，则该对象也包括在 **Tab** 键的自定义排序中。

如果 `tabEnabled` 为 `false`，则即使设置了 `tabIndex` 属性，该对象也不包括在 **Tab** 键的自动或自定义排序中。但是，如果 `MovieClip.tabChildren` 为 `true`，则即使 `tabEnabled` 设置为 `false`，仍可以将影片剪辑的子级包含到 **Tab** 键自动排序中。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例不将 `myMC2_mc` 包括在 **Tab** 键的自动排序中：

```
myMC1_mc.onRelease = function() {};  
myMC2_mc.onRelease = function() {};  
myMC3_mc.onRelease = function() {};  
myMC2_mc.tabEnabled = false;
```

另请参见

[onRelease \(MovieClip.onRelease 处理函数\)](#)，[tabEnabled \(Button.tabEnabled 属性\)](#)，[tabChildren \(MovieClip.tabChildren 属性\)](#)，[tabIndex \(MovieClip.tabIndex 属性\)](#)，[tabEnabled \(TextField.tabEnabled 属性\)](#)

## tabIndex (MovieClip.tabIndex 属性)

```
public tabIndex : Number
```

可用于自定义影片中对象的 **Tab** 键排序。默认情况下，`tabIndex` 属性为 `undefined`。可以对按钮、影片剪辑或文本字段实例设置 `tabIndex` 属性。

如果 **SWF** 文件中的一个对象包含 `tabIndex` 属性，则禁用 **Tab** 键自动排序，而使用该 **SWF** 文件中对象的 `tabIndex` 属性来计算 **Tab** 键排序。这个自定义的 **Tab** 键排序仅包括指定了 `tabIndex` 属性的对象。

`tabIndex` 属性必须是一个正整数。这些对象按照其 `tabIndex` 属性按升序进行排序。`tabIndex` 值为 1 的对象在 `tabIndex` 值为 2 的对象的前面。**Tab** 键的自定义排序会忽略 **SWF** 文件中对象的层次关系。**SWF** 文件中具有 `tabIndex` 属性的所有对象都排入 **Tab** 键顺序中。不要对多个对象使用相同的 `tabIndex` 值。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的 **ActionScript** 为三个影片剪辑实例设置自定义 **Tab** 键顺序。

```
myMC1_mc.onRelease = function() {};  
myMC2_mc.onRelease = function() {};  
myMC3_mc.onRelease = function() {};  
myMC1_mc.tabIndex = 2;  
myMC2_mc.tabIndex = 1;  
myMC3_mc.tabIndex = 3;
```

## 另请参见

[tabIndex \(Button.tabIndex 属性\)](#), [tabIndex \(TextField.tabIndex 属性\)](#)

## \_target (MovieClip.\_target 属性)

```
public _target : String [read-only]
```

返回影片剪辑实例的目标路径，以斜杠记号表示。使用 `eval()` 函数可以将目标路径转换为以点符号表示。

可用性: **ActionScript 1.0** ; **Flash Player 4**

## 示例

下面的示例以斜杠记号和点记号显示 **SWF** 文件中影片剪辑实例的目标路径。

```
for (var i in this) {  
    if (typeof (this[i]) == "movieclip") {  
        trace("name: " + this[i]._name + ",\t target: " + this[i]._target + ",\t  
            target(2):"  
            + eval(this[i]._target));  
    }  
}
```

## `_totalframes` (`MovieClip._totalframes` 属性)

`public _totalframes : Number [read-only]`

返回由 `MovieClip` 参数指定的影片剪辑实例中的总帧数。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

在以下示例中，两个影片剪辑按钮控制着时间轴。`prev_mc` 按钮将播放头移动到前一帧，`next_mc` 按钮将播放头移动到下一帧。将内容添加到时间轴上的一系列帧，然后将下列

**ActionScript** 添加到时间轴的第 1 帧：

```
stop();
prev_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe>1) {
        parent_mc.prevFrame();
    } else {
        parent_mc.gotoAndStop(parent_mc._totalframes);
    }
};
next_mc.onRelease = function() {
    var parent_mc:MovieClip = this._parent;
    if (parent_mc._currentframe<parent_mc._totalframes) {
        parent_mc.nextFrame();
    } else {
        parent_mc.gotoAndStop(1);
    }
};
```

## `trackAsMenu` (`MovieClip.trackAsMenu` 属性)

`public trackAsMenu : Boolean`

布尔值，指示其它按钮或影片剪辑是否可接收鼠标释放事件。`trackAsMenu` 属性允许您创建菜单。您可以对任何按钮或影片剪辑对象设置 `trackAsMenu` 属性。如果不存在 `trackAsMenu` 属性，则默认行为是 `false`。

可以随时更改 `trackAsMenu` 属性；修改后的影片剪辑会立即使用新的行为。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 示例

下面的示例为舞台上的三个影片剪辑设置 `trackAsMenu` 属性。单击一个影片剪辑并在第二个影片剪辑上释放鼠标按钮，以查看哪个实例接收事件。

```
myMC1_mc.trackAsMenu = true;
myMC2_mc.trackAsMenu = true;
myMC3_mc.trackAsMenu = false;

myMC1_mc.onRelease = clickMC;
myMC2_mc.onRelease = clickMC;
myMC3_mc.onRelease = clickMC;

function clickMC() {
    trace("you clicked the "+this._name+" movie clip.");
};
```

## 另请参见

[trackAsMenu \(Button.trackAsMenu 属性\)](#)

## transform (MovieClip.transform 属性)

`public transform : Transform`

一个对象，具有与影片剪辑的矩阵、颜色转换和像素范围有关的属性。在 **Transform** 类的条目中对特定属性 **matrix**、**colorTransform** 和三个只读属性（**concatenatedMatrix**、**concatenatedColorTransform** 和 **pixelBounds**）进行说明。

**transform** 对象的每个属性本身都是一个对象。这是很重要的，因为设置 **matrix** 或 **colorTransform** 对象的新值的唯一方法是，创建新对象并将该对象复制到 **transform.matrix** 或 **transform.colorTransform** 属性。

例如，若要增大影片剪辑的矩阵的 **tx** 值，必须制作整个 **matrix** 对象的副本，修改新对象的 **tx** 属性，然后将新对象复制到 **transform** 对象的 **matrix** 属性中：

```
var myMatrix:Object = myDisplayObject.transform.matrix;
myMatrix.tx += 10;
myDisplayObject.transform.matrix = myMatrix;
```

不能直接设置 **tx** 属性。下面的代码对 `myDisplayObject` 不起作用：

```
myDisplayObject.transform.matrix.tx += 10;
```

您也可以复制整个 **transform** 对象并为其分配其它影片剪辑的 **transform** 属性。例如，下面的代码将整个 **transform** 对象从 `myOldDisplayObj` 复制到 `myNewDisplayObj`：

```
myNewDisplayObj.transform = myOldDisplayObj.transform;
```

现在，新影片剪辑 `myNewDisplayObj` 与旧影片剪辑 `myOldDisplayObj` 具有相同的矩阵、颜色转换和像素范围值。

可用性：ActionScript 1.0；Flash Player 8

## 示例

下列示例显示如何使用影片剪辑的 `transform` 属性通过矩阵定位访问并修改影片剪辑的位置。

```
import flash.geom.Matrix;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var translateMatrix:Matrix = new Matrix();
translateMatrix.translate(10, 0);

rect.onPress = function() {
    var tmpMatrix:Matrix = this.transform.matrix;
    tmpMatrix.concat(translateMatrix);
    this.transform.matrix = tmpMatrix;
}

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[Transform \(flash.geom.Transform\)](#)

## unloadMovie (MovieClip.unloadMovie 方法)

```
public unloadMovie() : Void
```

删除影片剪辑实例的内容。保留实例属性和剪辑处理函数。

若要删除实例（包括其属性和剪辑处理函数），可使用 `MovieClip.removeMovieClip()`。

您可以通过创建子类来扩展 **MovieClip** 类的方法和事件处理函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例在用户单击 box 影片剪辑时卸载名为 box 的影片剪辑实例：

```
this.createEmptyMovieClip("box", 1);
```

```
with (box) {  
    lineStyle(1, 0xCCCCCC);  
    beginFill(0x4827CF);  
    moveTo(0, 0);  
    lineTo(80, 0);  
    lineTo(80, 60);  
    lineTo(0, 60);  
    lineTo(0, 0);  
    endFill();  
}  
  
box.onRelease = function() {  
    box.unloadMovie();  
};
```

### 另请参见

[removeMovieClip](#) ([MovieClip.removeMovieClip](#) 方法), [attachMovie](#) ([MovieClip.attachMovie](#) 方法), [loadMovie](#) ([MovieClip.loadMovie](#) 方法), [unloadMovie](#) 函数, [unloadMovieNum](#) 函数

## \_url (MovieClip.\_url 属性)

public \_url : String [read-only]

检索从其下载影片剪辑的 SWF、JPEG、GIF 或 PNG 文件的 URL。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

下列示例将已加载到 image\_mc 实例中的图像的 URL 显示在 “输出” 面板中。

```
this.createEmptyMovieClip("image_mc", 1);
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace("_url: "+target_mc._url);
};
var image_mc1:MovieClipLoader = new MovieClipLoader();
image_mc1.addListener(mcListener);
image_mc1.loadClip("http://www.macromedia.com/images/shared/product_boxes/
112x112/box_studio_112x112.jpg", image_mc);
```

下面的示例将 **ContextMenu** 对象 menu\_cm 分配给影片剪辑 image\_mc。menu\_cm 对象包含标签为 View Image in Browser 的自定义菜单项目, 它具有名为 viewImage() 的相关函数。

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("View Image in Browser...",
    viewImage));
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc.menu = menu_cm;
};
var image_mc1:MovieClipLoader = new MovieClipLoader();
image_mc1.addListener(mcListener);
image_mc1.loadClip("photo1.jpg", image_mc);

function viewImage(target_mc:MovieClip, obj:Object) {
    getURL(target_mc._url, "_blank");
}
```

当您在运行时右键单击 (**Windows**) 或按住 **Control** 键单击 (**Macintosh**) 图像时, 从上下文菜单中选择 “在浏览器中查看图像”, 以在浏览器窗口中打开该图像。

这些示例中使用的 **MovieClipLoader** 类需要 **Flash Player 7** 或更高版本。这些示例中使用的 **MovieClip.getNextHighestDepth()** 方法需要 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 **MovieClip.getNextHighestDepth()** 方法。



## useHandCursor (MovieClip.useHandCursor 属性)

`public useHandCursor : Boolean`

一个布尔值，指示当鼠标滑过影片剪辑时是否显示手指形（手形光标）。`useHandCursor` 属性的默认值是 `true`。如果 `useHandCursor` 设置为 `true`，则当鼠标滑过按钮影片剪辑时会显示用于按钮的手形光标。如果 `useHandCursor` 为 `false`，则将改用箭头指针。

可以随时更改 `useHandCursor` 属性；修改后的影片剪辑会立即使用新的光标行为。可以从原型对象中读出 `useHandCursor` 属性。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例为名为 `myMC1_mc` 和 `myMC2_mc` 的两个影片剪辑设置 `useHandCursor` 属性。对于其中一个实例，该属性设置为 `true`；对于另一个实例，它设置为 `false`。请注意这两个实例如何仍能接收事件。

```
myMC1_mc.onRelease = traceMC;
myMC2_mc.onRelease = traceMC;
myMC2_mc.useHandCursor = false;

function traceMC() {
    trace("you clicked: "+this._name);
};
```

## \_visible (MovieClip.\_visible 属性)

`public _visible : Boolean`

一个布尔值，指示影片剪辑是否处于可见状态。不可见的影片剪辑（将 `_visible` 属性设置为 `false`）被禁用。例如，无法单击影片剪辑中 `_visible` 设置为 `false` 的按钮。

可用性：ActionScript 1.0；Flash Player 4

### 示例

下面的示例为名为 `myMC1_mc` 和 `myMC2_mc` 的两个影片剪辑设置 `_visible` 属性。对于其中一个实例，该属性设置为 `true`；对于另一个实例，它设置为 `false`。请注意，将 `_visible` 属性设置为 `false` 后，无法单击 `myMC1_mc` 实例。

```
myMC1_mc.onRelease = function() {
    trace(this._name+"._visible = false");
    this._visible = false;
};
myMC2_mc.onRelease = function() {
    trace(this._name+"._alpha = 0");
    this._alpha = 0;
};
```

另请参见

[\\_visible \(Button.\\_visible 属性\)](#), [\\_visible \(TextField.\\_visible 属性\)](#)

## `_width` (`MovieClip._width` 属性)

`public _width : Number`

影片剪辑的宽度，以像素为单位。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

下列代码示例在“输出”面板中显示影片剪辑的高度和宽度：

```
this.createEmptyMovieClip("triangle", this.getNextHighestDepth());
```

```
triangle.beginFill(0x0000FF, 100);  
triangle.moveTo(100, 100);  
triangle.lineTo(100, 150);  
triangle.lineTo(150, 100);  
triangle.lineTo(100, 100);
```

```
trace(triangle._name + " = " + triangle._width + " X " + triangle._height + "  
pixels");
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[\\_height \(MovieClip.\\_height 属性\)](#)

## `_x` (MovieClip.\_x 属性)

`public _x : Number`

一个整数，它设置影片剪辑相对于父级影片剪辑的本地坐标的 **x** 坐标。如果影片剪辑在主时间轴中，则其坐标系统将舞台的左上角作为 **(0, 0)**。如果影片剪辑位于另一个具有变形的影片剪辑中，则该影片剪辑位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 **90°** 的影片剪辑，该影片剪辑的子级将继续逆时针旋转 **90°** 的坐标系统。影片剪辑的坐标指的是注册点的位置。

可用性: **ActionScript 1.0 ; Flash Player 3**

### 示例

下面的示例将具有链接标识符 `cursor_id` 的影片剪辑附加到 **SWF** 文件。影片剪辑名为 `cursor_mc`，它用于替换默认的鼠标指针。下面的 **ActionScript** 将影片剪辑实例的当前坐标设置为鼠标指针的位置：

```
this.attachMovie("cursor_id", "cursor_mc", this.getNextHighestDepth(),
    {_x:_xmouse, _y:_ymouse});
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    cursor_mc._x = _xmouse;
    cursor_mc._y = _ymouse;
    updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[\\_xscale](#) (MovieClip.\_xscale 属性), [\\_y](#) (MovieClip.\_y 属性), [\\_yscale](#) (MovieClip.\_yscale 属性)



## `_xscale` (`MovieClip._xscale` 属性)

`public _xscale : Number`

确定从影片剪辑注册点开始应用的影片剪辑水平缩放比例 (*percentage*)。默认注册点为 (0,0)。

缩放本地坐标系将影响 `_x` 和 `_y` 属性设置，这些设置是以整像素定义的。例如，如果父级影片剪辑缩放到 50%，则设置 `_x` 属性将移动影片剪辑中的对象，移动距离为在影片设置为 100% 时其像素数的一半。

可用性: **ActionScript 1.0** ; **Flash Player 4**

### 示例

下面的示例在运行时创建一个名为 `box_mc` 的影片剪辑。**Drawing API** 在此实例中用于绘制一个框，当鼠标滑过该框时，水平和垂直缩放将应用于影片剪辑。当鼠标滑离实例时，它将返回到以前的缩放比例。

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
    lineStyle(1, 0xCCCCC);
    beginFill(0xEEEEEE);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
};
box_mc.onRollOver = function() {
    this._x -= this._width/2;
    this._y -= this._height/2;
    this._xscale = 200;
    this._yscale = 200;
};
box_mc.onRollOut = function() {
    this._xscale = 100;
    this._yscale = 100;
    this._x += this._width/2;
    this._y += this._height/2;
};
```

### 另请参见

[\\_width](#) (`MovieClip._width` 属性), [\\_x](#) (`MovieClip._x` 属性), [\\_y](#) (`MovieClip._y` 属性), [\\_yscale](#) (`MovieClip._yscale` 属性)

## `_y` (MovieClip.\_y 属性)

`public _y : Number`

设置影片剪辑相对于父级影片剪辑的本地坐标的 `y` 坐标。如果影片剪辑在主时间轴中，则其坐标系统将舞台的左上角作为 (0, 0)。如果影片剪辑位于另一个具有变形的影片剪辑中，则该影片剪辑位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 90° 的影片剪辑，该影片剪辑的子级将继承逆时针旋转 90° 的坐标系统。影片剪辑的坐标指的是注册点的位置。

可用性: **ActionScript 1.0 ; Flash Player 3**

### 示例

下面的示例将具有链接标识符 `cursor_id` 的影片剪辑附加到 **SWF** 文件。影片剪辑名为 `cursor_mc`，它用于替换默认的鼠标指针。下面的 **ActionScript** 将影片剪辑实例的当前坐标设置为鼠标指针的位置：

```
this.attachMovie("cursor_id", "cursor_mc", this.getNextHighestDepth(),
    {_x:_xmouse, _y:_ymouse});
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    cursor_mc._x = _xmouse;
    cursor_mc._y = _ymouse;
    updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[\\_x](#) (MovieClip.\_x 属性), [\\_xscale](#) (MovieClip.\_xscale 属性), [\\_yscale](#) (MovieClip.\_yscale 属性)



## `_yscale` (`MovieClip._yscale` 属性)

`public _yscale : Number`

设置从影片剪辑注册点开始应用的影片剪辑垂直缩放比例 (*percentage*)。默认注册点为 (0,0)。

缩放本地坐标系将影响 `_x` 和 `_y` 属性设置，这些设置是以整像素定义的。例如，如果父级影片剪辑缩放到 50%，则设置 `_x` 属性将移动影片剪辑中的对象，移动距离为在影片设置为 100% 时其像素数的一半。

可用性: **ActionScript 1.0 ; Flash Player 4**

### 示例

下面的示例在运行时创建一个名为 `box_mc` 的影片剪辑。**Drawing API** 在此实例中用于绘制一个框，当鼠标滑过该框时，水平和垂直缩放将应用于影片剪辑。当鼠标滑离实例时，它将返回到以前的缩放比例。

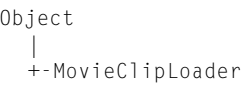
```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
    lineStyle(1, 0xCCCCCC);
    beginFill(0xEEEEEE);
    moveTo(0, 0);
    lineTo(80, 0);
    lineTo(80, 60);
    lineTo(0, 60);
    lineTo(0, 0);
    endFill();
};
box_mc.onRollOver = function() {
    this._x -= this._width/2;
    this._y -= this._height/2;
    this._xscale = 200;
    this._yscale = 200;
};
box_mc.onRollOut = function() {
    this._xscale = 100;
    this._yscale = 100;
    this._x += this._width/2;
    this._y += this._height/2;
};
```

### 另请参见

[\\_height](#) (`MovieClip._height` 属性), [\\_x](#) (`MovieClip._x` 属性), [\\_xscale](#) (`MovieClip._xscale` 属性), [\\_y](#) (`MovieClip._y` 属性)



# MovieClipLoader



```
public class MovieClipLoader
extends Object
```

此类用于实现在 SWF、JPEG、GIF 和 PNG 文件正被加载到影片剪辑中时提供状态信息的侦听器回调。若要使用 **MovieClipLoader** 功能，请使用 `MovieClipLoader.loadClip()` 代替 `loadMovie()` 或 `MovieClip.loadMovie()` 来加载 **SWF** 文件。

在您发出 `MovieClipLoader.loadClip()` 命令后，下列事件将按列出顺序发生：

- 在下载的文件的第一字节写入硬盘后，调用 `MovieClipLoader.onLoadStart` 侦听器。
- 如果您已实现了 `MovieClipLoader.onLoadProgress` 侦听器，则在加载过程中调用它。  
注意：您可以在加载过程中随时调用 `MovieClipLoader.getProgress()`。
- 在下载了整个文件都写入硬盘后，调用 `MovieClipLoader.onLoadComplete` 侦听器。
- 在执行完下载的文件的第一帧动作后，调用 `MovieClipLoader.onLoadInit` 侦听器。

在调用 `MovieClipLoader.onLoadInit` 后，您可以设置属性、使用方法，还可与加载的影片进行交互。

如果文件未能完全加载，则调用 `MovieClipLoader.onLoadError` 侦听器。

可用性：ActionScript 1.0 ； Flash Player 7

## 属性摘要

继承自 `Object` 类的属性

---

`constructor` (`Object.constructor` 属性), `__proto__` (`Object.__proto__` 属性), `prototype` (`Object.prototype` 属性), `__resolve` (`Object.__resolve` 属性)

---

## 事件摘要

事件	说明
<code>onLoadComplete = function([target_mc:MovieClip], [httpStatus:Number]) {}</code>	当使用 <code>MovieClipLoader.loadClip()</code> 加载的文件完全下载时调用。
<code>onLoadError = function(target_mc:MovieClip, errorCode:String, [httpStatus:Number]) {}</code>	当使用 <code>MovieClipLoader.loadClip()</code> 加载的文件未能加载时调用。

事件	说明
onLoadInit = function([target_mc:MovieClip]) { }	当执行加载的剪辑的第一帧上的动作时调用。
onLoadProgress = function([target_mc:MovieClip], loadedBytes:Number, totalBytes:Number) { }	在加载过程中（即在 MovieClipLoader.onLoadStart 和 MovieClipLoader.onLoadComplete 之间时），每当正加载的内容写入硬盘时调用。
onLoadStart = function([target_mc:MovieClip]) { }	当对 MovieClipLoader.loadClip() 的调用已开始下载文件时调用。

## 构造函数摘要

签名	说明
MovieClipLoader()	创建一个 MovieClipLoader 对象，您可以使用该对象来实现多个侦听器，以便在下载 SWF、JPEG、GIF 或 PNG 文件时响应事件。

## 方法摘要

修饰符	签名	说明
	addListener(listener:Object) : Boolean	注册一个对象，以便在调用 MovieClipLoader 事件处理函数时接收通知。
	getProgress(target:Object) : Object	返回正在通过使用 MovieClipLoader.loadClip() 加载的文件的已加载字节数和总字节数；对于压缩的影片，返回压缩后的字节数。
	loadClip(url:String, target:Object) : Boolean	在播放原始影片时，将 SWF、JPEG、渐进式 JPEG、非动画 GIF 或 PNG 文件加载到 Flash Player 中的影片剪辑中。
	removeListener(listener:Object) : Boolean	删除在调用 MovieClipLoader 事件处理函数时用来接收通知的侦听器。
	unloadClip(target:Object) : Boolean	删除通过使用 MovieClipLoader.loadClip() 加载的影片剪辑。

继承自 Object 类的方法

---

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

---

## addListener (MovieClipLoader.addListener 方法)

```
public addListener(listener:Object) : Boolean
```

注册一个对象，以便在调用 `MovieClipLoader` 事件处理函数时接收通知。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**listener:Object** - 一个对象，它侦听从 `MovieClipLoader` 事件处理函数发出的回调通知。

### 返回

**Boolean** - 一个布尔值。如果建立侦听器成功，则返回 `true`；否则返回 `false`。

### 示例

下面的示例将图像加载到名为 `image_mc` 的影片剪辑中。该影片剪辑实例在舞台上旋转并居中，并且会围绕舞台和影片剪辑的周边绘制笔触。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._x = Stage.width/2-target_mc._width/2;
    target_mc._y = Stage.height/2-target_mc._width/2;
    var w:Number = target_mc._width;
    var h:Number = target_mc._height;
    target_mc.lineStyle(4, 0x000000);
    target_mc.moveTo(0, 0);
    target_mc.lineTo(w, 0);
    target_mc.lineTo(w, h);
    target_mc.lineTo(0, h);
    target_mc.lineTo(0, 0);
    target_mc._rotation = 3;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mcListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

`onLoadComplete` (`MovieClipLoader.onLoadComplete` 事件侦听器), `onLoadError` (`MovieClipLoader.onLoadError` 事件侦听器), `onLoadInit` (`MovieClipLoader.onLoadInit` 事件侦听器), `onLoadProgress` (`MovieClipLoader.onLoadProgress` 事件侦听器), `onLoadStart` (`MovieClipLoader.onLoadStart` 事件侦听器), `removeListener` (`MovieClipLoader.removeListener` 方法)

## getProgress (MovieClipLoader.getProgress 方法)

`public getProgress(target:Object) : Object`

返回正在通过使用 `MovieClipLoader.loadClip()` 加载的文件的已加载字节数和总字节数；对于压缩的影片，返回压缩后的字节数。`getProgress` 方法使您可以显式请求此信息，而不是（或除此之外还）编写 `MovieClipLoader.onLoadProgress` 侦听器函数。

可用性：ActionScript 1.0；Flash Player 7

### 参数

**target:Object** - 通过使用 `MovieClipLoader.loadClip()` 加载的 SWF、JPEG、GIF 或 PNG 文件。

### 返回

`Object` - 一个具有以下两个整数属性的对象：`bytesLoaded` 和 `bytesTotal`。

### 示例

下面的示例演示 `getProgress()` 方法的用法。通常，您会创建一个侦听器对象（而不是使用此方法）来侦听 `onLoadProgress` 事件。另请注意，对 `getProgress()` 的第一个同步调用会返回容器的已加载字节数和总字节数，而不是外部请求的对象的值。

```
var container:MovieClip = this.createEmptyMovieClip("container",
    this.getNextHighestDepth());
var image:MovieClip = container.createEmptyMovieClip("image",
    container.getNextHighestDepth());

var mcLoader:MovieClipLoader = new MovieClipLoader();
var listener:Object = new Object();
listener.onLoadProgress = function(target:MovieClip, bytesLoaded:Number,
    bytesTotal:Number):Void {
```

```

        trace(target + ".onLoadProgress with " + bytesLoaded + " bytes of " +
bytesTotal);
    }
    mcLoader.addListener(listener);
    mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", image);

    var interval:Object = new Object();
    interval.id = setInterval(checkProgress, 100, mcLoader, image, interval);

    function checkProgress(mcLoader:MovieClipLoader, image:MovieClip,
        interval:Object):Void {
        trace(">> checking progress now with : " + interval.id);
        var progress:Object = mcLoader.getProgress(image);
        trace("bytesLoaded: " + progress.bytesLoaded + " bytesTotal: " +
progress.bytesTotal);
        if(progress.bytesLoaded == progress.bytesTotal) {
            clearInterval(interval.id);
        }
    }
}

```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[loadClip \(MovieClipLoader.loadClip 方法\)](#), [onLoadProgress \(MovieClipLoader.onLoadProgress 事件侦听器\)](#)

## loadClip (MovieClipLoader.loadClip 方法)

```
public loadClip(url:String, target:Object) : Boolean
```

在播放原始影片时，将 SWF、JPEG、渐进式 JPEG、非动画 GIF 或 PNG 文件加载到 **Flash Player** 中的影片剪辑中。如果您加载 GIF 动画，仅显示第一帧。使用此方法可以一次显示多个 SWF 文件，并且无需加载另一个 HTML 文档即可在 SWF 文件间进行切换。

使用 `loadClip()` 方法代替 `loadMovie()` 或 `MovieClip.loadMovie()` 具有许多优点。通过使用

`MovieClipLoader.addListener(listenerObject)` 向 **MovieClipLoader** 类注册侦听器，可以激活该侦听器。

- 在加载开始时调用 `MovieClipLoader.onLoadStart` 处理函数。
- 在无法加载剪辑时调用 `MovieClipLoader.onLoadError` 处理函数。
- 在加载进程正进行时调用 `MovieClipLoader.onLoadProgress` 处理函数。
- 在文件完成下载但已加载的影片剪辑的方法和属性尚不可用时调用 `MovieClipLoader.onLoadComplete` 处理函数。在 `onLoadInit` 处理函数之前调用此处理函数。

- 在执行该剪辑的第一帧中的动作后调用 `MovieClipLoader.onLoadInit` 处理函数，以便您可以开始处理加载的剪辑。在 `onLoadComplete` 处理函数之后调用此处理函数。在大多数情况下，请使用 `onLoadInit` 处理函数。

加载到影片剪辑的 SWF 文件或图像会继承该影片剪辑的位置、旋转和缩放属性。可以用该影片剪辑的目标路径来定位加载的影片。

您可以使用 `loadClip()` 方法将一个或多个文件加载到单个影片剪辑或级别中；将 **MovieClipLoader** 侦听器对象作为参数传递给正加载的目标影片剪辑实例。或者，您可以为加载的每个文件创建不同的 **MovieClipLoader** 对象。

使用 `MovieClipLoader.unloadClip()` 可删除用此方法加载的影片或图像，或者取消正在进行的加载操作。

如果这些文件是本地文件，`MovieClipLoader.getProgress()` 和 `MovieClipLoaderListener.onLoadProgress` 将不报告创作播放器中实际的 `bytesLoaded` 和 `bytesTotal` 值。您在创作环境中使用“带宽设置”功能时，`MovieClipLoader.getProgress()` 和 `MovieClipLoaderListener.onLoadProgress` 按实际下载速率报告下载状态，而不按“带宽设置”提供的降低的带宽速率报告。

使用此方法时，请考虑 **Flash Player** 安全模型。

对于 **Flash Player 8**：

- 如果执行调用的影片剪辑在只能与本地文件系统的内容交互的沙箱中，并且被加载的影片剪辑来自网络沙箱，则不允许加载。
- 如果执行调用的 SWF 文件在网络沙箱中并且要加载的影片剪辑是本地的，则不允许加载。
- 从受信任的本地沙箱或只能与远程内容交互的沙箱访问网络沙箱需要通过跨域策略文件获得网站的许可。
- 在只能与本地文件系统的内容交互的沙箱中的影片剪辑不能对只能与远程内容交互的沙箱中的影片剪辑使用脚本（反之也是禁止的）。

对于 **Flash Player 7** 及更高版本：

- 网站可以允许通过跨域策略文件来跨域访问资源。
- 各 SWF 文件之间的脚本使用将依据这些 SWF 文件的原始域而予以限制。使用 `System.security.allowDomain()` 方法可调整这些限制。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：**ActionScript 1.0**；**Flash Player 7**

## 参数

**url:String** - 要加载的 SWF、JPEG、GIF 或 PNG 文件的绝对或相对 URL。相对路径必须相对于级别 0 处的 SWF 文件。绝对 URL 必须包括协议引用，例如 `http://` 或 `file:///`。文件名不能包括磁盘驱动器规格。

**target:Object** - 影片剪辑的目标路径，或者指定 Flash Player 中影片将加载到的级别的整数。目标影片剪辑将被加载的 SWF 文件或图像所替换。

## 返回

Boolean - 一个布尔值。如果发送 URL 请求成功，则为 true；否则为 false。

## 示例

下面的示例演示如何通过为 onLoadInit 事件创建处理函数，然后发出请求来使用 MovieClipLoader.loadClip() 方法。

下面的代码应该直接置于时间轴上的帧动作中，或者粘贴到扩展影片剪辑的类中。此代码还需要名为 **YourImage.jpg** 的图像位于经过编译的 SWF 文件所在的目录中。

```
var container:MovieClip = createEmptyMovieClip("container",
    getNextHighestDepth());
var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(this);
mcLoader.loadClip("YourImage.jpg", container);
```

```
function onLoadInit(mc:MovieClip) {
    trace("onLoadInit: " + mc);
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 DepthManager 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

## 另请参见

[onLoadInit \(MovieClipLoader.onLoadInit 事件侦听器\)](#)

# MovieClipLoader 构造函数

```
public MovieClipLoader()
```

创建一个 MovieClipLoader 对象，您可以使用该对象来实现多个侦听器，以便在下载 SWF、JPEG、GIF 或 PNG 文件时响应事件。

可用性：ActionScript 1.0；Flash Player 7

## 示例

请参见 MovieClipLoader.loadClip()。

另请参见

`addListener (MovieClipLoader.addListener 方法), loadClip (MovieClipLoader.loadClip 方法)`

## onLoadComplete

### (MovieClipLoader.onLoadComplete 事件侦听器)

```
onLoadComplete = function([target_mc:MovieClip], [httpStatus:Number]) {}
```

当使用 `MovieClipLoader.loadClip()` 加载的文件完全下载时调用。对使用 `MovieClipLoader.addListener()` 添加的侦听器对象调用此侦听器。**Flash Player** 将 `onLoadComplete` 事件侦听器传递到您的代码中，但您不必在侦听器函数中实现所有参数。`target_mc` 的值标识作为这一调用的目标的影片剪辑。此标识在使用同一组侦听器加载多个文件时非常有用。

在 **Flash Player 8** 中，此侦听器会返回一个 **HTTP** 状态代码。如果 **Flash Player** 无法从服务器获取状态代码或者 **Flash Player** 无法与服务器进行通讯，则将默认值 **0** 传递到您的 **ActionScript** 代码。在任何播放器中都可生成值 **0**（例如，当请求了格式不正确的 **URL** 时），而且当 **Flash Player** 插件在下列浏览器（这些浏览器不能将 **HTTP** 状态代码从服务器传递到 **Flash Player**）中运行时，值 **0** 始终由 **Flash Player** 插件生成：适用于 **Macintosh** 的 **Netscape**、**Mozilla**、**Safari**、**Opera** 和 **Internet Explorer**。

了解 `MovieClipLoader.onLoadComplete` 和 `MovieClipLoader.onLoadInit` 之间的差异非常重要。`onLoadComplete` 事件在加载 **SWF**、**JPEG**、**GIF** 或 **PNG** 文件之后但在应用程序初始化之前被调用。此时，无法访问已加载的影片剪辑的方法和属性，因此您无法调用函数、移动到特定帧，等等。在多数情况下，最好改为使用 `onLoadInit` 事件，它在内容已加载并完全初始化后被调用。

可用性：ActionScript 1.0；Flash Player 7

#### 参数

**target\_mc:MovieClip** [可选] - 通过 `MovieClipLoader.loadClip()` 方法加载的影片剪辑。

**httpStatus:Number** [可选] - （仅适用于 **Flash Player 8**）由服务器返回的 **HTTP** 状态代码。例如，状态代码 **404** 表明服务器尚未找到请求的 **URI** 的任何匹配项。有关 **HTTP** 状态代码的更多信息，请参见 **HTTP** 规范（地址是 <ftp://ftp.isi.edu/in-notes/rfc2616.txt>）的 10.4 和 10.5 节。



## 示例

下面的示例创建一个影片剪辑，一个新的 `MovieClipLoader` 实例以及一个匿名事件侦听器，该侦听器会侦听 `onLoadComplete` 事件，但却等待发生 `onLoadInit` 事件以便与已加载的元素属性进行交互。

```
var loadListener:Object = new Object();

loadListener.onLoadComplete = function(target_mc:MovieClip,
    httpStatus:Number):Void {
    trace(">> loadListener.onLoadComplete()");
    trace(">> =====");
    trace(">> target_mc._width: " + target_mc._width); // 0
    trace(">> httpStatus: " + httpStatus);
}

loadListener.onLoadInit = function(target_mc:MovieClip):Void {
    trace(">> loadListener.onLoadInit()");
    trace(">> =====");
    trace(">> target_mc._width: " + target_mc._width); // 315
}

var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(loadListener);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", mc);
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[addListener](#) (`MovieClipLoader.addListener` 方法), [loadClip](#) (`MovieClipLoader.loadClip` 方法), [onLoadStart](#) (`MovieClipLoader.onLoadStart` 事件侦听器), [onLoadError](#) (`MovieClipLoader.onLoadError` 事件侦听器), [onLoadInit](#) (`MovieClipLoader.onLoadInit` 事件侦听器)

## onLoadError (MovieClipLoader.onLoadError 事件侦听器)

```
onLoadError = function(target_mc:MovieClip, errorCode:String,  
    [httpStatus:Number]) {}
```

当使用 `MovieClipLoader.loadClip()` 加载的文件未能加载时调用。出于各种原因，会调用此侦听器；例如服务器关闭、找不到文件或发生安全侵犯。

对通过使用 `MovieClipLoader.addListener()` 添加的侦听器对象调用此侦听器。

`target_mc` 的值标识作为这一调用的目标的影片剪辑。此参数在使用同一组侦听器加载多个文件时非常有用。

对于 `errorCode` 参数，如果 `MovieClipLoader.onLoadStart` 和 `MovieClipLoader.onLoadComplete` 都尚未被调用，则返回字符串 `"URLNotFound"`；例如，当服务器关闭或找不到文件时。如果调用了 `MovieClipLoader.onLoadStart`，但未调用 `MovieClipLoader.onLoadComplete`，则返回字符串 `"LoadNeverCompleted"`；例如，当下载由于服务器超载、服务器崩溃等原因中断时。

在 **Flash Player 8** 中，此侦听器会在 `httpStatus` 参数中返回 **HTTP** 状态代码。如果 **Flash Player** 无法从服务器获取状态代码或者 **Flash Player** 无法与服务器进行通讯，则将默认值 **0** 传递到您的 **ActionScript** 代码。在任何播放器中都可生成值 **0**（例如，当请求了格式不正确的 **URL** 时），而且当 **Flash Player** 插件在下列浏览器（这些浏览器不能将 **HTTP** 状态代码从服务器传递到 **Flash Player**）中运行时，值 **0** 始终由 **Flash Player** 插件生成：适用于 **Macintosh** 的 **Netscape**、**Mozilla**、**Safari**、**Opera** 和 **Internet Explorer**。如果播放器未尝试发出执行加载操作的 **URL** 请求，也会生成值 **0**。发生这种情况是因为此请求违反了 **SWF** 文件的安全沙箱规则。

可用性：ActionScript 1.0；Flash Player 7

### 参数

**target\_mc:MovieClip** - 通过 `MovieClipLoader.loadClip()` 方法加载的影片剪辑。

**errorCode:String** - 解释失败原因的字符串，为 `"URLNotFound"` 或 `"LoadNeverCompleted"`。

**httpStatus:Number [可选]** - （仅适用于 **Flash Player 8**）由服务器返回的 **HTTP** 状态代码。例如，状态代码 **404** 表明服务器尚未找到请求的 **URI** 的任何匹配项。有关 **HTTP** 状态代码的更多信息，请参见 **HTTP** 规范（地址是 <ftp://ftp.isi.edu/in-notes/rfc2616.txt>）的 10.4 和 10.5 节。

## 示例

下面的示例在加载图像失败时在“输出”面板中显示信息。此示例中使用的 URL 仅用于演示目的；请将其替换为您自己的有效 URL。

```
var loadListener:Object = new Object();

loadListener.onLoadError = function(target_mc:MovieClip, errorCode:String,
    httpStatus:Number) {
    trace(">> loadListener.onLoadError()");
    trace(">> =====");
    trace(">> errorCode: " + errorCode);
    trace(">> httpStatus: " + httpStatus);
}

var mcLoader:MovieClipLoader = new MovieClipLoader();
mcLoader.addListener(loadListener);

var mc:MovieClip = this.createEmptyMovieClip("mc",
    this.getNextHighestDepth());
mcLoader.loadClip("http://www.fakedomain.com/images/bad_hair_day.jpg", mc);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[addListener \(MovieClipLoader.addListener 方法\)](#), [loadClip \(MovieClipLoader.loadClip 方法\)](#), [onLoadStart \(MovieClipLoader.onLoadStart 事件侦听器\)](#), [onLoadComplete \(MovieClipLoader.onLoadComplete 事件侦听器\)](#)

## onLoadInit (MovieClipLoader.onLoadInit 事件侦听器)

```
onLoadInit = function([target_mc:MovieClip]) {}
```

当执行加载的剪辑的第一帧上的动作时调用。在调用此侦听器后，您可以设置属性、使用方法，还可以与加载的影片交互。对通过使用 `MovieClipLoader.addListener()` 添加的侦听器对象调用此侦听器。

`target_mc` 的值标识作为这一调用的目标的影片剪辑。此参数在使用同一组侦听器加载多个文件时非常有用。

可用性: **ActionScript 1.0** ; **Flash Player 7**

## 参数

**target\_mc:MovieClip** [可选] - 通过 `MovieClipLoader.loadClip()` 方法加载的影片剪辑。

## 示例

下面的示例将图像加载到名为 image\_mc 的影片剪辑实例中。onLoadInit 和 onLoadComplete 事件用于确定加载图像所用的时间。此信息显示在名为 timer\_txt 的文本字段中。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mcListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mcListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
        target_mc._height,
        target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mc1:MovieClipLoader = new MovieClipLoader();
image_mc1.addListener(mcListener);
image_mc1.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

下面的示例检查是否已将影片加载到在运行时创建的影片剪辑中。此示例中使用的 URL 仅用于演示目的；请将其替换为您自己的有效 URL。

```
this.createEmptyMovieClip("tester_mc", 1);
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace("movie loaded");
}
var image_mc1:MovieClipLoader = new MovieClipLoader();
image_mc1.addListener(mcListener);
image_mc1.loadClip("http://www.yourserver.com/your_movie.swf", tester_mc);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

## 另请参见

[addListener \(MovieClipLoader.addListener 方法\)](#), [loadClip \(MovieClipLoader.loadClip 方法\)](#), [onLoadStart \(MovieClipLoader.onLoadStart 事件侦听器\)](#)

## onLoadProgress (MovieClipLoader.onLoadProgress 事件侦听器)

```
onLoadProgress = function([target_mc:MovieClip], loadedBytes:Number,  
    totalBytes:Number) {}
```

在加载过程中（即在 MovieClipLoader.onLoadStart 和 MovieClipLoader.onLoadComplete 之间时），每当正加载的内容写入硬盘时调用。对通过使用 MovieClipLoader.addListener() 添加的侦听器对象调用此侦听器。您可以使用此方法并使用 loadedBytes 和 totalBytes 参数，显示有关下载进度的信息。

target\_mc 的值标识作为这一调用的目标的影片剪辑。这在使用同一组侦听器加载多个文件时非常有用。



如果您试图在测试模式中与本站留在硬盘上的本地文件一起使用 onLoadProgress，则它将不能正常工作，这是因为在测试模式中，Flash Player 会加载本地文件的所有内容。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**target\_mc:MovieClip** [可选] - 通过 MovieClipLoader.loadClip() 方法加载的影片剪辑。

**loadedBytes:Number** - 在调用该侦听器时已加载字节数。

**totalBytes:Number** - 正在加载的文件的总字节数。

### 示例

下面的示例创建一个影片剪辑、一个新的 MovieClipLoader 实例以及一个匿名事件侦听器。它定期输出加载进度，最后在加载完成且资源对 **ActionScript** 可用时提供通知。

```
var container:MovieClip = this.createEmptyMovieClip("container",  
    this.getNextHighestDepth());  
var mcLoader:MovieClipLoader = new MovieClipLoader();  
var listener:Object = new Object();  
listener.onLoadProgress = function(target:MovieClip, bytesLoaded:Number,  
    bytesTotal:Number):Void {  
    trace(target + ".onLoadProgress with " + bytesLoaded + " bytes of " +  
        bytesTotal);  
}  
listener.onLoadInit = function(target:MovieClip):Void {  
    trace(target + ".onLoadInit");  
}  
mcLoader.addListener(listener);  
mcLoader.loadClip("http://www.w3.org/Icons/w3c_main.png", container);
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

另请参见

`addListener` (`MovieClipLoader.addListener` 方法), `loadClip` (`MovieClipLoader.loadClip` 方法), `getProgress` (`MovieClipLoader.getProgress` 方法)

## onLoadStart (MovieClipLoader.onLoadStart 事件侦听器)

```
onLoadStart = function([target_mc:MovieClip]) {}
```

当对 `MovieClipLoader.loadClip()` 的调用已开始下载文件时调用。对通过使用 `MovieClipLoader.addListener()` 添加的侦听器对象调用此侦听器。

`target_mc` 的值标识作为这一调用的目标的影片剪辑。此参数在使用同一组侦听器加载多个文件时非常有用。

可用性: **ActionScript 1.0 ; Flash Player 7**

### 参数

**target\_mc:MovieClip** [可选] - 通过 `MovieClipLoader.loadClip()` 方法加载的影片剪辑。

### 示例

下面的示例将图像加载到名为 `image_mc` 的影片剪辑实例中。`onLoadInit` 和 `onLoadComplete` 事件用于确定加载图像所用的时间。此信息显示在名为 `timer_txt` 的文本字段中。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadStart = function(target_mc:MovieClip) {
    target_mc.startTimer = getTimer();
};
mcListener.onLoadComplete = function(target_mc:MovieClip) {
    target_mc.completeTimer = getTimer();
};
mcListener.onLoadInit = function(target_mc:MovieClip) {
    var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
        target_mc._height,
        target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mcListener);
image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

`addListener` (`MovieClipLoader.addListener` 方法), `loadClip` (`MovieClipLoader.loadClip` 方法), `onLoadError` (`MovieClipLoader.onLoadError` 事件侦听器), `onLoadInit` (`MovieClipLoader.onLoadInit` 事件侦听器), `onLoadComplete` (`MovieClipLoader.onLoadComplete` 事件侦听器)

## removeListener (MovieClipLoader.removeListener 方法)

```
public removeListener(listener:Object) : Boolean
```

删除在调用 `MovieClipLoader` 事件处理函数时用来接收通知的侦听器。不会收到进一步的加载消息。

可用性: **ActionScript 1.0 ; Flash Player 7**

参数

**listener:Object** - 已使用 `MovieClipLoader.addListener()` 添加的侦听器对象。

返回

**Boolean** - 一个布尔值。如果删除侦听器成功，则返回 `true`；否则返回 `false`。

示例

下面的示例将图像加载到影片剪辑中，并且使用户可以使用名为 `start_button` 和 `stop_button` 的两个按钮来开始和停止加载过程。当用户开始或停止该过程时，“输出”面板中会显示信息。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadStart = function(target_mc:MovieClip) {
    trace("\t onLoadStart");
};
mcListener.onLoadComplete = function(target_mc:MovieClip) {
    trace("\t onLoadComplete");
};
mcListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
    trace("\t onLoadError: "+errorCode);
};
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace("\t onLoadInit");
    start_button.enabled = true;
```

```

        stop_button.enabled = false;
    };
    var image_mcl:MovieClipLoader = new MovieClipLoader();
    //
    start_button.clickHandler = function() {
        trace("Starting...");
        start_button.enabled = false;
        stop_button.enabled = true;
        //
        image_mcl.addListener(mclListener);
        image_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
            image_mc);
    };
    stop_button.clickHandler = function() {
        trace("Stopping...");
        start_button.enabled = true;
        stop_button.enabled = false;
        //
        image_mcl.removeListener(mclListener);
    };
    stop_button.enabled = false;

```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[addListener \(MovieClipLoader.addListener 方法\)](#)

## unloadClip (MovieClipLoader.unloadClip 方法)

```
public unloadClip(target:Object) : Boolean
```

删除通过使用 `MovieClipLoader.loadClip()` 加载的影片剪辑。如果您在正加载影片时发出此命令，则调用 `MovieClipLoader.onLoadError`。

**可用性:** **ActionScript 1.0** ; **Flash Player 7**

### 参数

**target:Object** - 传递至对 `my_mcl.loadClip()` 的相应调用的字符串或整数。

### 返回

**Boolean** - 一个布尔值。如果删除影片剪辑成功，则返回 `true`；否则返回 `false`。



## 示例

下面的示例将图像加载到名为 `image_mc` 的影片剪辑中。如果单击影片剪辑，则会删除该影片剪辑，并且信息会显示在“输出”面板中。

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._x = 100;
    target_mc._y = 100;
    target_mc.onRelease = function() {
        trace("Unloading clip...");
        trace("\t name: "+target_mc._name);
        trace("\t url: "+target_mc._url);
        image_mc1.unloadClip(target_mc);
    };
};
var image_mc1:MovieClipLoader = new MovieClipLoader();
image_mc1.addListener(mcListener);
image_mc1.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    image_mc);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 `DepthManager` 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[loadClip \(MovieClipLoader.loadClip 方法\)](#), [onLoadError \(MovieClipLoader.onLoadError 事件侦听器\)](#)

# NetConnection

```
Object
|
+- NetConnection
```

```
public dynamic class NetConnection
extends Object
```

**NetConnection** 类提供从本地驱动器或 HTTP 地址播放 FLV 文件流的方法。



当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此类。有关更多信息，请参见 Flash Communication Server 文档。

可用性: **ActionScript 1.0** ; **Flash Player 7**

属性摘要  
继承自 Object 类的属性

---

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

---

构造函数摘要

签名	说明
<code>NetConnection()</code>	创建 <code>NetConnection</code> 对象, 您可以将该对象与 <code>NetStream</code> 对象一起使用来播放本地视频流 (FLV) 文件。

方法摘要

修饰符	签名	说明
	<code>connect(targetURI:String) : Boolean</code>	打开您可以通过它从 HTTP 地址或本地文件系统回放视频 (FLV) 文件的本地连接。

继承自 Object 类的方法

---

`addProperty` (Object.addProperty 方法), `hasOwnProperty` (Object.hasOwnProperty 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `registerClass` (Object.registerClass 方法), `toString` (Object.toString 方法), `unwatch` (Object.unwatch 方法), `valueOf` (Object.valueOf 方法), `watch` (Object.watch 方法)

---

## connect (NetConnection.connect 方法)

`public connect(targetURI:String) : Boolean`

打开您可以通过它从 HTTP 地址或本地文件系统回放视频 (FLV) 文件的本地连接。

使用此方法时, 请考虑 **Flash Player** 安全模型和下列安全注意事项:

- 默认值为拒绝在沙箱之间进行访问。网站可以通过跨域策略文件来启用对资源的访问。
- 通过在 **Flash Communication Server** 中添加服务器端 **ActionScript** 应用程序逻辑, 网站可以拒绝对资源的访问。
- 对于 **Flash Player 8**, 如果执行调用的 **SWF** 文件在只能与本地文件系统交互的沙箱中, 则不允许 `NetConnection.connect()`。

有关更多信息, 请参见以下部分:

- 《学习 Flash 中的 **ActionScript 2.0**》的第 17 章, “了解安全性”
- **Flash Player 8 安全性白皮书** (位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security))

- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 7

### 参数

**targetURI:String** - 对于此参数，必须传递 `null`。

### 返回

`Boolean` - 如果为 `false`，则连接将失败且无法使用。如果为 `true`，则此连接在调用 `connect()` 方法时没有失败，但不能保证会成功。

### 示例

下面的示例打开一个连接以播放 `video2.flv` 文件。从“库”面板的选项菜单中选择“新建视频”以创建一个新视频对象，并为该对象指定实例名称 `my_video`。

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video2.flv");
```

另请参见

[NetStream](#)

## NetConnection 构造函数

```
public NetConnection()
```

创建 `NetConnection` 对象，您可以将该对象与 `NetStream` 对象一起使用来播放本地视频流 (FLV) 文件。在创建 `NetConnection` 对象后，使用 `NetConnection.connect()` 进行实际连接。

与在 `Flash` 文档中嵌入视频相比，播放外部 `FLV` 文件有多个好处，例如更好的性能和内存管理以及独立的视频和 `Flash` 帧频。`NetConnection` 类提供从本地驱动器或 `HTTP` 地址播放 `FLV` 文件流的方法。

可用性：ActionScript 1.0；Flash Player 7

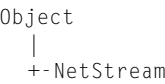
### 示例

请参见 `NetConnection.connect()` 的示例。

另请参见

[connect](#) ([NetConnection.connect](#) 方法), [attachVideo](#) ([Video.attachVideo](#) 方法), [NetStream](#)

# NetStream



```
public dynamic class NetStream
extends Object
```

**NetStream** 类提供从本地文件系统或 HTTP 地址播放 **Flash** 视频 (FLV) 文件的方法和属性。您可以使用 **NetStream** 对象通过 **NetConnection** 对象对视频进行流式处理。与在 **Flash** 文档中嵌入视频相比, 播放外部 FLV 文件有多个好处, 例如更好的性能和内存管理以及独立的视频和 **Flash** 帧频。该类提供若干方法和属性, 您可以利用这些方法和属性在一个文件加载和播放时跟踪该文件的进度, 以及便于用户控制播放 (停止或暂停等)。

可用性: **ActionScript 1.0** ; **Flash Player 7**

## 属性摘要

修饰符	属性	说明
	<code>bufferLength: Number [ 只读 ]</code>	数据当前存在于缓冲区中的秒数。
	<code>bufferTime: Number [ 只读 ]</code>	由 <code>NetStream.setBufferTime()</code> 分配给缓冲区的秒数。
	<code>bytesLoaded: Number [ 只读 ]</code>	已加载到播放器中的数据的数据的字节数。
	<code>bytesTotal: Number [ 只读 ]</code>	正加载到播放器中的文件的总大小 (以字节为单位)。
	<code>currentFps: Number [ 只读 ]</code>	每秒所显示的帧的数目。
	<code>time: Number [ 只读 ]</code>	播放头的位置 (以秒为单位)。

## 继承自 **Object** 类的属性

<code>constructor</code> ( <a href="#">Object.constructor</a> 属性), <code>__proto__</code> ( <a href="#">Object.__proto__</a> 属性), <code>prototype</code> ( <a href="#">Object.prototype</a> 属性), <code>__resolve</code> ( <a href="#">Object.__resolve</a> 属性)
--

事件摘要

事件	说明
<code>onMetaData = function(infoObject: Object) {}</code>	在 Flash Player 接收在正播放的 FLV 文件中嵌入的描述性信息时调用。
<code>onStatus = function(infoObject: Object) {}</code>	每当状态更改或发布针对 NetStream 对象的错误时调用。

构造函数摘要

签名	说明
<code>NetStream(connection: NetConnection)</code>	创建可用于通过指定的 NetConnection 对象播放 FLV 文件的流。

方法摘要

修饰符	签名	说明
	<code>close() : Void</code>	停止播放流上的所有数据，将 NetStream.time 属性设置为 0，并使该流可用于其它用途。
	<code>pause([flag:Boolean]) : Void</code>	暂停或恢复流的回放。
	<code>play(name:Object, start:Number, len:Number, reset:Object) : Void</code>	开始回放外部视频 (FLV) 文件。
	<code>seek(offset:Number) : Void</code>	从流的开始处搜寻最接近于指定秒数的关键帧。
	<code>setBufferTime(bufferTime: Number) : Void</code>	指定在开始显示流之前需要多长时间将消息存入缓冲区。

继承自 Object 类的方法

<code>addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)</code>
--

## bufferLength (NetStream.bufferLength 属性)

`public bufferLength : Number [read-only]`

数据当前存在于缓冲区中的秒数。您可以将此属性与 `NetStream.bufferTime` 一起使用以评估缓冲区将近填满的程度；例如，向正等待数据加载到缓冲区中的用户显示反馈。

可用性：ActionScript 1.0；Flash Player 7

### 示例

以下示例动态创建一个文本字段，该字段显示有关当前位于缓冲区中的秒数的信息。该文本字段还显示为视频设置的缓冲区长度以及缓冲区的填充百分比。

```
this.createTextField("buffer_txt", this.getNextHighestDepth(), 10, 10, 300,
    22);
buffer_txt.html = true;

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.setBufferTime(3);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

var buffer_interval:Number = setInterval(checkBufferTime, 100, stream_ns);
function checkBufferTime(my_ns:NetStream):Void {
    var bufferPct:Number = Math.min(Math.round(my_ns.bufferLength/
        my_ns.bufferTime 100), 100);
    var output_str:String = "<textformat tabStops='[100,200]'\>";
    output_str += "Length: "+my_ns.bufferLength+"\t"+"Time: "+my_ns.bufferTime+"\t"+"Buffer: "+bufferPct+"%";
    output_str += "</textformat>";
    buffer_txt.htmlText = output_str;
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 `DepthManager` 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[bufferTime \(NetStream.bufferTime 属性\)](#)，[bytesLoaded \(NetStream.bytesLoaded 属性\)](#)

## bufferTime (NetStream.bufferTime 属性)

public bufferTime : Number [read-only]

由 `NetStream.setBufferTime()` 分配给缓冲区的秒数。默认值是 `.1`（十分之一秒）。若要确定当前位于缓冲区中的秒数，请使用 `NetStream.bufferLength`。

可用性：ActionScript 1.0；Flash Player 7

### 示例

以下示例动态创建一个文本字段，该字段显示有关当前位于缓冲区中的秒数的信息。该文本字段还显示为视频设置的缓冲区长度的填充百分比。

```
this.createTextField("buffer_txt", this.getNextHighestDepth(), 10, 10, 300,
    22);
buffer_txt.html = true;

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.setBufferTime(3);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

var buffer_interval:Number = setInterval(checkBufferTime, 100, stream_ns);
function checkBufferTime(my_ns:NetStream):Void {
    var bufferPct:Number = Math.min(Math.round(my_ns.bufferLength/
        my_ns.bufferTime 100), 100);
    var output_str:String = "<textformat tabStops='[100,200]'\>";
    output_str += "Length: "+my_ns.bufferLength+"\t"+"Time: ";
    output_str += "+my_ns.bufferTime+"\t"+"Buffer: "+bufferPct+"%";
    output_str += "</textformat>";
    buffer_txt.htmlText = output_str;
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 `DepthManager` 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[setBufferTime \(NetStream.setBufferTime 方法\)](#) , [time \(NetStream.time 属性\)](#) ,  
[bufferLength \(NetStream.bufferLength 属性\)](#)

## bytesLoaded（NetStream.bytesLoaded 属性）

public bytesLoaded : Number [read-only]

已加载到播放器中的数据字节数。您可以将此方法与 NetStream.bytesTotal 一起使用以评估缓冲区将近填满的程度；例如，向正等待数据加载到缓冲区中的用户显示反馈。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例使用 **Drawing API** 以及 bytesLoaded 和 bytesTotal 属性创建进度栏，以显示将 **video1.flv** 加载到名为 my\_video 的视频对象实例中的进度。动态创建一个名为 loaded\_txt 的文本字段，以便也显示有关加载进度的信息。

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10, 160,
22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc",
progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
    beginFill(0xFF0000);
    moveTo(0, 0);
    lineTo(100, 0);
    lineTo(100, 10);
    lineTo(0, 10);
    lineTo(0, 0);
    endFill();
    _xscale = 0;
}
progressBar_mc.createEmptyMovieClip("stroke_mc",
progressBar_mc.getNextHighestDepth());
with (progressBar_mc.stroke_mc) {
    lineStyle(0, 0x000000);
    moveTo(0, 0);
    lineTo(100, 0);
    lineTo(100, 10);
    lineTo(0, 10);
    lineTo(0, 0);
}

var loaded_interval:Number = setInterval(checkBytesLoaded, 500, stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
    var pctLoaded:Number = Math.round(my_ns.bytesLoaded/my_ns.bytesTotal 100);
```



```

loaded_txt.text = Math.round(my_ns.bytesLoaded/1000)+" of
"+Math.round(my_ns.bytesTotal/1000)+" KB loaded (" +pctLoaded+"%");
progressBar_mc.bar_mc._xscale = pctLoaded;
if (pctLoaded>=100) {
    clearInterval(loaded_interval);
}
}

```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[bytesTotal](#) ([NetStream.bytesTotal](#) 属性), [bufferLength](#) ([NetStream.bufferLength](#) 属性)

## bytesTotal ([NetStream.bytesTotal](#) 属性)

```
public bytesTotal : Number [read-only]
```

正加载到播放器中的文件的总大小（以字节为单位）。

可用性：ActionScript 1.0；Flash Player 7

示例

下面的示例使用 **Drawing API** 以及 `bytesLoaded` 和 `bytesTotal` 属性创建进度栏，以显示将 **video1.flv** 加载到名为 `my_video` 的视频对象实例中的进度。动态创建一个名为 `loaded_txt` 的文本字段，以便也显示有关加载进度的信息。

```

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10, 160,
22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc",
    progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
    beginFill(0xFF0000);
    moveTo(0, 0);
    lineTo(100, 0);
    lineTo(100, 10);
    lineTo(0, 10);
    lineTo(0, 0);
    endFill();
    _xscale = 0;
}

```

```

}
progressBar_mc.createEmptyMovieClip("stroke_mc",
    progressBar_mc.getNextHighestDepth());
with (progressBar_mc.stroke_mc) {
    lineStyle(0, 0x000000);
    moveTo(0, 0);
    lineTo(100, 0);
    lineTo(100, 10);
    lineTo(0, 10);
    lineTo(0, 0);
}

var loaded_interval:Number = setInterval(checkBytesLoaded, 500, stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
    var pctLoaded:Number = Math.round(my_ns.bytesLoaded/my_ns.bytesTotal 100);
    loaded_txt.text = Math.round(my_ns.bytesLoaded/1000)+" of
    "+Math.round(my_ns.bytesTotal/1000)+" KB loaded (" +pctLoaded+"%");
    progressBar_mc.bar_mc._xscale = pctLoaded;
    if (pctLoaded>=100) {
        clearInterval(loaded_interval);
    }
}

```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[bytesLoaded \(NetStream.bytesLoaded 属性\)](#)，[bufferTime \(NetStream.bufferTime 属性\)](#)

## close (NetStream.close 方法)

```
public close() : Void
```

停止播放流上的所有数据，将 `NetStream.time` 属性设置为 0，并使该流可用于其它用途。此命令还删除已使用 HTTP 下载的 FLV 文件的本地副本。虽然 **Flash Player** 会删除它所创建的 FLV 文件的本地副本，但该视频的副本可以永久保存在浏览器的缓存目录中。如果需要完全阻止 FLV 文件的缓存或本地存储，请使用 **Flash Communication Server MX**。

可用性: **ActionScript 1.0** ; **Flash Player 7**

## 示例

下面的 `onDisconnect()` 函数在您单击名为 `close_btn` 的按钮时关闭一个连接并删除存储在本地磁盘上的 **video1.flv** 的临时副本：

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

close_btn.onRelease = function(){
    stream_ns.close();
};
```

另请参见

[pause \(NetStream.pause 方法\)](#)，[play \(NetStream.play 方法\)](#)

## currentFps (NetStream.currentFps 属性)

`public currentFps : Number [read-only]`

每秒所显示的帧的数目。如果您正在导出 FLV 文件以在多个系统上播放，则您可以在测试期间检查该值以帮助您确定在导出该文件时要应用多大程度的压缩。

可用性：ActionScript 1.0；Flash Player 7

## 示例

下面的示例创建一个文本字段，该字段显示 **video1.flv** 显示的当前帧频率。

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("fps_txt", this.getNextHighestDepth(), 10, 10, 50, 22);
fps_txt.autoSize = true;
var fps_interval:Number = setInterval(displayFPS, 500, stream_ns);
function displayFPS(my_ns:NetStream) {
    fps_txt.text = "currentFps (frames per second):"
    "+Math.floor(my_ns.currentFps);
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 `DepthManager` 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## NetStream 构造函数

```
public NetStream(connection:NetConnection)
```

创建可用于通过指定的 **NetConnection** 对象播放 FLV 文件的流。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**connection:**NetConnection - 一个 **NetConnection** 对象。

### 示例

以下代码首先构造新的 **NetConnection** 对象 `connection_nc`，并使用它构造名为 `stream_ns` 的新 **NetStream** 对象。从“库”选项菜单中选择“新建视频”以创建视频对象实例，并为其指定实例名称 `my_video`。

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
```

### 另请参见

[NetConnection](#), [attachVideo](#) ([Video.attachVideo](#) 方法)

## onMetaData (NetStream.onMetaData 处理函数)

```
onMetaData = function(infoObject:Object) {}
```

在 **Flash Player** 接收在正播放的 FLV 文件中嵌入的描述性信息时调用。

**Flash Video Exporter** 实用工具 (1.1 版本或更高版本) 将视频的持续时间、创建日期、数据速率及其它信息嵌入视频文件本身。不同的视频编码器嵌入不同的元数据组中。

此处理函数在调用 `NetStream.play()` 方法后而在视频播放头前进之前触发。

在多数情况下，FLV 元数据中嵌入的持续时间值接近实际持续时间，但是并不精确。换言之，当播放头在视频流的结尾处时，此值不会始终与 `NetStream.time` 属性的值相匹配。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**infoObject:Object** - 包含各个元数据项的某个属性的对象。

## 示例

此示例中的代码首先创建新的 **NetConnection** 对象和 **NetStream** 对象。然后，它为此 **NetStream** 对象定义 `onMetaData()` 处理函数。此处理函数循环遍历接收的 **infoObject** 对象中的每个命名属性，并打印属性的名称和值。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
var ns:NetStream = new NetStream(nc);

ns.onMetaData = function(infoObject:Object) {
    for (var propName:String in infoObject) {
        trace(propName + " = " + infoObject[propName]);
    }
};

ns.play("http://www.helpexamples.com/flash/video/water.flv");
```

这会导致显示以下信息：

```
canSeekToEnd = true
videocodecid = 4
framerate = 15
videodatarate = 400
height = 215
width = 320
duration = 7.347
```

根据对 **FLV** 文件进行编码所使用的软件，属性的列表会有所不同。

## 另请参见

[time \(NetStream.time 属性\)](#), [play \(NetStream.play 方法\)](#), [NetConnection](#)

## onStatus (NetStream.onStatus 处理函数)

```
onStatus = function(infoObject:Object) {}
```

每当状态更改或发布针对 **NetStream** 对象的错误时调用。如果您要对此事件处理函数做出响应，则必须创建一个函数来处理该信息对象。

信息对象具有一个代码属性（其中包含用以描述 **onStatus** 处理函数结果的字符串）和一个级别属性（其中包含 “**status**” 或 “**error**” 字符串）。

除了此 **onStatus** 处理函数外，**Flash** 还提供名为 `System.onStatus` 的“超级”函数。如果为特定对象调用了 **onStatus** 但未分配任何函数对其进行响应，则 **Flash** 将处理分配到 `System.onStatus` 的函数（如果存在）。

以下事件在发生某些 `NetStream` 活动时通知您。

代码属性	级别属性	含义
<code>NetStream.Buffer.Empty</code>	<code>status</code>	数据的接收速度不足以填充缓冲区。数据流将在缓冲区重新填充前中断，此时将发送 <code>NetStream.Buffer.Full</code> 消息，并且该流将重新开始播放。
<code>NetStream.Buffer.Full</code>	<code>status</code>	缓冲区已满并且流将开始播放。
<code>NetStream.Buffer.Flush</code>	<code>status</code>	数据已完成流式处理，剩余的缓冲区将清空。
<code>NetStream.Play.Start</code>	<code>status</code>	播放已开始。
<code>NetStream.Play.Stop</code>	<code>status</code>	播放已结束。
<code>NetStream.Play.StreamNotFound</code>	<code>error</code>	无法找到传递给 <code>play()</code> 方法的 FLV。
<code>NetStream.Seek.InvalidTime</code>	<code>error</code>	对于使用渐进式下载方式下载的视频，用户已尝试跳过到目前为止已下载的视频数据的结尾或在整个文件已下载后跳过视频的结尾进行搜寻或播放。 <code>message.details</code> 属性包含一个时间代码，该代码指出用户可以搜寻的最后一个有效位置。
<code>NetStream.Seek.Notify</code>	<code>status</code>	搜寻操作完成。

如果您不断地看到有关缓冲区的错误，您应尝试使用 `NetStream.setBufferTime()` 方法更改缓冲区。

可用性：ActionScript 1.0 ； Flash Player 6

参数

`infoObject:Object` - 根据状态消息或错误消息定义的参数。

示例

下面的示例在“输出”面板中显示有关流的数据：

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
stream_ns.onStatus = function(infoObject:Object) {
    trace("NetStream.onStatus called: (" +getTimer()+ " ms)");
    for (var prop in infoObject) {
        trace("\t"+prop+":\t"+infoObject[prop]);
    }
}
```

```
        trace("");  
    };
```

另请参见

[setBufferTime \(NetStream.setBufferTime 方法\)](#), [onStatus \(System.onStatus 处理函数\)](#)

## pause (NetStream.pause 方法)

```
public pause([flag:Boolean]) : Void
```

暂停或恢复流的回放。

您第一次调用此方法时（不发送参数），它将暂停播放；下一次调用此方法时，它将恢复播放。您可能要将此方法附加到一个按钮上，用户可以按下该按钮来暂停或恢复播放。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**flag:Boolean** [ 可选 ] - 一个布尔值，指定是暂停播放 (**true**) 还是恢复播放 (**false**)。如果您省略此参数，`NetStream.pause()` 将会在暂停播放和恢复播放之间切换：第一次对指定的流调用该方法时，该流暂停播放；下一次调用该方法时，该流恢复播放。

### 示例

以下示例阐释该方法的某些用法：

```
my_ns.pause(); // pauses play first time issued  
my_ns.pause(); // resumes play  
my_ns.pause(false); // no effect, play continues  
my_ns.pause(); // pauses play
```

另请参见

[close \(NetStream.close 方法\)](#), [play \(NetStream.play 方法\)](#)

## play (NetStream.play 方法)

```
public play(name:Object, start:Number, len:Number, reset:Object) : Void
```

开始回放外部视频 (FLV) 文件。若要查看视频数据, 您必须调用 `Video.attachVideo()` 方法; 与视频一起进行流式处理的音频或只包含音频的 FLV 文件自动播放。

如果要控制与 FLV 文件关联的音频, 您可以使用 `MovieClip.attachAudio()` 将音频路由到影片剪辑; 然后创建 **Sound** 对象来控制该音频的某些属性。有关更多信息, 请参见 `MovieClip.attachAudio()`。

如果无法找到 FLV 文件, 则调用 `NetStream.onStatus` 事件处理函数。如果要停止当前正播放的流, 请使用 `NetStream.close()`。

您可以播放与 SWF 文件存储于同一目录中或存储于其子目录中的本地 FLV 文件; 但不能导航到更高级别的目录。例如, 如果 SWF 文件位于名为 `/training` 的目录中, 并且您要播放存储于 `/training/videos` 目录中的视频, 则应使用以下语法:

```
my_ns.play("videos/videoName.flv");
```

若要播放存储于 `/training` 目录中的视频, 应使用以下语法:

```
my_ns.play("videoName.flv");
```

使用此方法时, 请考虑 **Flash Player** 安全模型。

对于 **Flash Player 8**:

- 如果执行调用的 SWF 文件在只能与本地系统文件交互的沙箱中, 而资源在非本地沙箱中, 则不允许 `NetStream.play()`。
- 从受信任的本地沙箱或只能与远程内容交互的沙箱访问网络沙箱需要通过跨域策略文件获得网站的许可。

有关更多信息, 请参见以下部分:

- 《学习 Flash 中的 **ActionScript 2.0**》的第 17 章, “了解安全性”
- **Flash Player 8** 安全性白皮书 (位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security))
- **Flash Player 8** 与安全相关的 API 白皮书 (位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis))

可用性: **ActionScript 1.0** : **Flash Player 7**

### 参数

**name**:Object - 用引号括起来要播放的 FLV 文件的名称。支持 `http://` 和 `file://` 格式; `file://` 位置始终相对于 SWF 文件的位置。

**start**:Number -

**len**:Number -

**reset**:Object -



## 示例

下面的示例说明使用 `NetStream.play()` 命令的若干方法。您可以播放位于用户的计算机上的文件。**joe\_user** 目录是存储 SWF 所在目录的子目录。您还可以播放位于服务器上的文件：

```
// Play a file that is on the user's computer.
my_ns.play("file://joe_user/flash/videos/lectureJune26.flv");

// Play a file on a server.
my_ns.play("http://someServer.someDomain.com/flash/video/orientation.flv");
```

## 另请参见

[attachAudio \(MovieClip.attachAudio 方法\)](#), [close \(NetStream.close 方法\)](#),  
[onStatus \(NetStream.onStatus 处理函数\)](#), [pause \(NetStream.pause 方法\)](#),  
[attachVideo \(Video.attachVideo 方法\)](#)

## seek (NetStream.seek 方法)

```
public seek(offset:Number) : Void
```

从流的开始处搜寻最接近于指定秒数的关键帧。该流在达到流中的指定位置时恢复播放。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**offset:**Number - 要移至 FLV 文件中的时间近似值（以秒为单位）。播放头将移至与 *numberOfSeconds* 最接近的视频的关键帧。

- 若要返回到该流的开始处，请将 **0** 传递给 *numberOfSeconds*。
- 若要从此流的开始处向前搜寻，应传递您要前进的秒数。例如，若要将播放头置于距开始处 **15** 秒的位置，请使用 `my_ns.seek(15)`。
- 若要搜寻当前位置的相对位置，请传递 `my_ns.time + n` 或 `my_ns.time - n` 来搜寻分别从当前位置前进或后退 **n** 秒的位置。例如，若要从当前位置后退 **20** 秒，请使用 `my_ns.seek(my_ns.time - 20)`。

视频搜寻的确切位置将因导出视频时的帧频率设置而异。因此，如果以 **6 fps** 和 **30 fps** 的帧频率导出同一视频，则它将搜寻您所使用的两个不同位置，例如，两个视频对象各自的 `my_ns.seek(15)` 处。

## 示例

下面的示例说明使用 `NetStream.seek()` 命令的若干方法。您可以返回到流的起始处，可以移至距流起始处 **30** 秒的位置，也可以向后移至距当前位置 **3** 分钟的位置：

```
// Return to the beginning of the stream
my_ns.seek(0);

// Move to a location 30 seconds from the beginning of the stream
my_ns.seek(30);

// Move backwards three minutes from current location
my_ns.seek(my_ns.time - 180);
```

## 另请参见

[time \(NetStream.time 属性\)](#)

## setBufferTime (NetStream.setBufferTime 方法)

```
public setBufferTime(bufferTime:Number) : Void
```

指定在开始显示流之前需要多长时间将消息存入缓冲区。例如，如果您要确保流的最初 **15** 秒无中断播放，请将 *numberOfSeconds* 设置为 **15**；**Flash** 在将 **15** 秒的数据存入缓冲区后才开始播放该流。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**bufferTime**:Number - 在 **Flash** 开始显示数据前存入缓冲区的数据的秒数。默认值是 **.1**（十分之一秒）。

## 示例

请参见 `NetStream.bufferLength` 的示例。

## 另请参见

[bufferLength \(NetStream.bufferLength 属性\)](#)，[bufferTime \(NetStream.bufferTime 属性\)](#)

## time (NetStream.time 属性)

public time : Number [read-only]

播放头的位置（以秒为单位）。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例在动态创建的名为 time\_txt 的文本字段中显示播放头的当前位置。从“库”选项菜单中选择“新建视频”以创建视频对象实例，并为其指定实例名称 my\_video。创建名为 my\_video 的新视频对象。请将以下 ActionScript 添加到 FLA 或 AS 文件：

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
//
stream_ns.onStatus = function(info:Object) {
    statusCode_txt.text = info.code;
};

this.createTextField("time_txt", this.getNextHighestDepth(), 10, 10, 100,
    22);
time_txt.text = "LOADING";

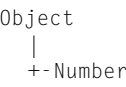
var time_interval:Number = setInterval(checkTime, 500, stream_ns);
function checkTime(my_ns:NetStream) {
    var ns_seconds:Number = my_ns.time;
    var minutes:Number = Math.floor(ns_seconds/60);
    var seconds = Math.floor(ns_seconds%60);
    if (seconds<10) {
        seconds = "0"+seconds;
    }
    time_txt.text = minutes+": "+seconds;
}
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 DepthManager 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

### 另请参见

[bufferLength \(NetStream.bufferLength 属性\)](#)，[bytesLoaded \(NetStream.bytesLoaded 属性\)](#)

# Number



```
public class Number
extends Object
```

**Number** 类是 **Number** 数据类型的简单包装对象。使用与 **Number** 类关联的方法和属性可以操作基元数值。此类与 **JavaScript** 的 **Number** 类完全相同。

但 **Number** 类的属性是静态属性，这意味着无需对象就可以使用这些属性，因此您不需要使用构造函数。

以下示例调用 **Number** 类的 `toString()` 方法，它返回字符串 1234：

```
var myNumber:Number = new Number(1234);
myNumber.toString();
```

以下示例将 `MIN_VALUE` 属性的值分配给一个无需使用构造函数进行声明的变量：

```
var smallest:Number = Number.MIN_VALUE;
```

可用性： **ActionScript 1.0** ； **Flash Player 5**

## 属性摘要

修饰符	属性	说明
static	MAX_VALUE:Number	最大可表示数（双精度 IEEE-754）。
static	MIN_VALUE:Number	最小可表示数（双精度 IEEE-754）。
static	NaN:Number	表示“非数字”（NaN）的 IEEE-754 值。
static	NEGATIVE_INFINITY: Number	指定表示负无穷大的 IEEE-754 值。
static	POSITIVE_INFINITY: Number	指定表示正无穷大的 IEEE-754 值。

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

构造函数摘要

签名	说明
<code>Number(num:Object)</code>	创建一个新的 <code>Number</code> 对象。

方法摘要

修饰符	签名	说明
	<code>toString(radix:Number) : String</code>	返回指定的 <code>Number</code> 对象 ( <code>myNumber</code> ) 的字符串表示形式。
	<code>valueOf() : Number</code>	返回指定的 <code>Number</code> 对象的基元值类型。

继承自 `Object` 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

MAX\_VALUE (`Number.MAX_VALUE` 属性)

`public static MAX_VALUE : Number`

最大可表示数（双精度 IEEE-754）。此数字大约为 1.79e+308。

可用性: `ActionScript 1.0` ; `Flash Player 5`

示例

以下 `ActionScript` 向“输出”面板显示最大可表示数和最小可表示数。

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

此代码显示下列值:

```
Number.MIN_VALUE = 4.94065645841247e-324
Number.MAX_VALUE = 1.79769313486232e+308
```

## MIN\_VALUE (Number.MIN\_VALUE 属性)

`public static MIN_VALUE : Number`

最小可表示数（双精度 IEEE-754）。此数字大约为 5e-324。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

以下 **ActionScript** 向“输出”面板显示最大可表示数和最小可表示数。

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);  
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

此代码显示下列值:

```
Number.MIN_VALUE = 4.94065645841247e-324  
Number.MAX_VALUE = 1.79769313486232e+308
```

## NaN (Number.NaN 属性)

`public static NaN : Number`

表示“非数字”(NaN)的 IEEE-754 值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

另请参见

[isNaN 函数](#)

## NEGATIVE\_INFINITY (Number.NEGATIVE\_INFINITY 属性)

`public static NEGATIVE_INFINITY : Number`

指定表示负无穷大的 IEEE-754 值。此属性的值与常数 `-Infinity` 的值相同。

负无穷大是当数学运算或函数返回的值超过可表示的负值时，返回的一个特殊数值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

此示例对除以下值的结果进行比较。

```
var posResult:Number = 1/0;  
if (posResult == Number.POSITIVE_INFINITY) {  
    trace("posResult = "+posResult); // output: posResult = Infinity  
}  
var negResult:Number = -1/0;  
if (negResult == Number.NEGATIVE_INFINITY) {  
    trace("negResult = "+negResult); // output: negResult = -Infinity
```

## Number 构造函数

```
public Number(num:Object)
```

创建一个新的 **Number** 对象。new Number 构造函数主要用作占位符。**Number** 对象与 Number() 函数不同，后者将参数转换为原始值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**num:Object** - 要创建的 **Number** 对象的数值，或者要转换为数字的值。如果未提供 value，则默认值为 0。

### 示例

下面的代码构造新的 **Number** 对象：

```
var n1:Number = new Number(3.4);  
var n2:Number = new Number(-10);
```

### 另请参见

[toString \(Number.toString 方法\)](#), [valueOf \(Number.valueOf 方法\)](#)

## POSITIVE\_INFINITY (Number.POSITIVE\_INFINITY 属性)

```
public static POSITIVE_INFINITY : Number
```

指定表示正无穷大的 **IEEE-754** 值。此属性的值与常数 *Infinity* 的值相同。

正无穷大是当数学运算或函数返回的值大于可表示的值时，返回的一个特殊数值。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

此示例对除以下值的结果进行比较。

```
var posResult:Number = 1/0;  
if (posResult == Number.POSITIVE_INFINITY) {  
    trace("posResult = "+posResult); // output: posResult = Infinity  
}  
var negResult:Number = -1/0;  
if (negResult == Number.NEGATIVE_INFINITY) {  
    trace("negResult = "+negResult); // output: negResult = -Infinity
```

## toString (Number.toString 方法)

```
public toString(radix:Number) : String
```

返回指定的 **Number** 对象 (*myNumber*) 的字符串表示形式。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**radix**:Number - 指定要用于数字到字符串的转换的基数 (从 2 到 36)。如果未指定 **radix** 参数, 则默认值为 10。

### 返回

String - 一个字符串。

### 示例

下面的示例将 2 和 8 用于 **radix** 参数, 并返回包含数字 9 的相应表示形式的字符串:

```
var myNumber:Number = new Number(9);  
trace(myNumber.toString(2)); // output: 1001  
trace(myNumber.toString(8)); // output: 11
```

下面的示例生成十六进制值。

```
var r:Number = new Number(250);  
var g:Number = new Number(128);  
var b:Number = new Number(114);  
var rgb:String = "0x"+ r.toString(16)+g.toString(16)+b.toString(16);  
trace(rgb);  
// output: rgb:0xFA8072 (Hexadecimal equivalent of the color 'salmon')
```

## valueOf (Number.valueOf 方法)

```
public valueOf() : Number
```

返回指定的 **Number** 对象的基元值类型。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个字符串。

### 示例

下面的示例生成 **numSocks** 对象的原始值。

```
var numSocks = new Number(2);  
trace(numSocks.valueOf()); // output: 2
```



# Object

Object

```
public class Object
```

**Object** 类位于 **ActionScript** 类层次结构的根处。此类包含 **JavaScript Object** 类所提供功能的一小部分。

可用性：ActionScript 1.0 ； Flash Player 5

## 属性摘要

修饰符	属性	说明
	constructor:Object	对给定的对象实例的构造函数的引用。
	__proto__:Object	引用用于创建对象的类 (ActionScript 2.0) 或构造函数 (ActionScript 1.0) 的 prototype 属性。
static	prototype:Object	对类或函数对象的超类的引用。
	__resolve:Object	对用户定义的函数的引用，该函数在 ActionScript 代码引用未定义的属性或方法时调用。

## 构造函数摘要

签名	说明
Object()	创建 <b>Object</b> 对象，并将对该对象的构造函数方法的引用存储在该对象的 <b>constructor</b> 属性中。

## 方法摘要

修饰符	签名	说明
	addProperty(name:String, getter:Function, setter:Function) : Boolean	创建一个 <b>getter/setter</b> 属性。
	hasOwnProperty(name:String) : Boolean	指示对象是否已经定义了指定的属性。
	isPropertyEnumerable(name:String) : Boolean	指示指定的属性是否存在、是否可枚举。

修饰符	签名	说明
	<code>isPrototypeOf(theClass:Object) : Boolean</code>	指示 Object 类的实例是否在指定为参数的对象的原型链中。
<code>static</code>	<code>registerClass(name:String, theClass:Function) : Boolean</code>	将影片剪辑元件与 ActionScript 对象类相关联。
	<code>toString() : String</code>	将指定对象转换为字符串然后返回它。
	<code>unwatch(name:String) : Boolean</code>	删除 Object.watch() 创建的监视点。
	<code>valueOf() : Object</code>	返回指定对象的原始值。
	<code>watch(name:String, callback:Function, [userData:Object]) : Boolean</code>	注册当 ActionScript 对象的指定属性更改时要调用的事件处理函数。

## addProperty（Object.addProperty 方法）

```
public addProperty(name:String, getter:Function, setter:Function) : Boolean
```

创建一个 **getter/setter** 属性。当 Flash 读取 **getter/setter** 属性时，它调用 `get` 函数，而该函数的返回值将成为 `name` 的值。当 Flash 写入 **getter/setter** 属性时，它调用 `set` 函数，并且将新值作为参数传递给该函数。如果具有给定名称的属性已经存在，新属性将覆盖该属性。

“**get**”函数是一个不带参数的函数。它的返回值可以为任何类型。它的类型可以在两次调用之间改变。返回值视作该属性的当前值。

“**set**”函数带一个参数，即该属性的新值。例如，如果属性 `x` 由语句 `x = 1` 进行赋值，则会将数字类型的参数 `1` 传递给 `set` 函数。忽略 `set` 函数的返回值。

可以向原型对象添加 **getter/setter** 属性。如果向一个原型对象添加 **getter/setter** 属性，则继承此原型对象的所有对象实例都将继承 **getter/setter** 属性。这样就能够在一个位置（即原型对象）中添加 **getter/setter** 属性，然后使它传播到类的所有实例（与向原型对象中添加方法相似）。如果为继承的原型对象中的 **getter/setter** 属性调用 `get/set` 函数，则传递给该 `get/set` 函数的引用是最初引用的对象，而不是该原型对象。

如果调用不正确，则 `Object.addProperty()` 可能出错失败。下表说明了可能发生的错误：

出错条件	后果
name 不是有效的属性名称；例如，一个空字符串。	返回 false，并且不添加该属性。
getter 不是有效的函数对象。	返回 false，并且不添加该属性。
setter 不是有效的函数对象。	返回 false，并且不添加该属性。

可用性：ActionScript 1.0 ； Flash Player 6

参数

**name:**String - 一个字符串；要创建的对象属性的名称。  
**getter:**Function - 被调用以检索属性值的函数；此参数是一个 **Function** 对象。  
**setter:**Function - 被调用以设置属性值的函数；此参数是一个 **Function** 对象。如果向此参数传递 null 值，则该属性为只读。

返回

Boolean - 一个布尔值：如果属性创建成功，则返回 true ； 否则返回 false。

示例

在下面的示例中，对象具有两个内部方法：`setQuantity()` 和 `getQuantity()`。当设置或检索了属性 `bookcount` 后，该属性可用于调用这些方法。第三个内部方法 `getTitle()` 返回一个与属性 `bookname` 关联的只读值。当脚本检索 `myBook.bookcount` 的值时，**ActionScript** 解释程序将自动调用 `myBook.getQuantity()`。当脚本修改 `myBook.bookcount` 的值时，该解释程序调用 `myObject.setQuantity()`。`bookname` 属性不指定 `set` 函数，因此忽略修改 `bookname` 的尝试。

```
function Book() {
    this.setQuantity = function(numBooks:Number):Void {
        this.books = numBooks;
    };
    this.getQuantity = function():Number {
        return this.books;
    };
    this.getTitle = function():String {
        return "Catcher in the Rye";
    };
    this.addProperty("bookcount", this.getQuantity, this.setQuantity);
    this.addProperty("bookname", this.getTitle, null);
}
```

```

var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
// output: You ordered 5 copies of Catcher in the Rye

```

上一个示例将正常运行，但将属性 `bookcount` 和 `bookname` 添加到 `Book` 对象的每个实例，这就要求对象的每个实例都拥有这两个属性。如果类中有许多属性（如 `bookcount` 和 `bookname`），它们可能占用大量的内存。相反，您可以将这些属性添加到 `Book.prototype`，以便 `bookcount` 和 `bookname` 属性仅存在于一个位置中。但是，它的效果与示例（直接将 `bookcount` 和 `bookname` 添加到每一个实例）中代码的效果相同。如果尝试访问 **Book** 实例中的任一属性，则在该属性不存在时，会导致原型链上升，直到遇到在 `Book.prototype` 中定义的版本。下面的示例说明如何将属性添加到 `Book.prototype`：

```

function Book() {}
Book.prototype.setQuantity = function(numBooks:Number):Void {
    this.books = numBooks;
};
Book.prototype.getQuantity = function():Number {
    return this.books;
};
Book.prototype.getTitle = function():String {
    return "Catcher in the Rye";
};
Book.prototype.addProperty("bookcount", Book.prototype.getQuantity,
    Book.prototype.setQuantity);
Book.prototype.addProperty("bookname", Book.prototype.getTitle, null);
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);

```

下面的示例说明如何使用 **ActionScript 2.0** 中提供的隐式 **getter** 和 **setter** 函数。您可以在名为 **Book.as** 的外部文件中定义 `Book` 类，而不必定义 `Book` 函数并编辑 `Book.prototype`。下面的代码必须位于名为 **Book.as** 的单独外部文件中，该文件只包含此类定义并且驻留在 **Flash** 应用程序的类路径中：

```

class Book {
    var books:Number;
    function set bookcount(numBooks:Number):Void {
        this.books = numBooks;
    }
    function get bookcount():Number {
        return this.books;
    }
    function get bookname():String {
        return "Catcher in the Rye";
    }
}

```

然后，可以将下面的代码置于 FLA 文件中，这些代码的运行方式与在前几个示例中的运行方式相同：

```
var myBook:Book = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

另请参见

[get 语句](#)，[set 语句](#)

## constructor（Object.constructor 属性）

public constructor : Object

对给定的对象实例的构造函数的引用。在使用 **Object** 类的构造函数创建对象时，**constructor** 属性会自动分配给所有对象。

可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例是对 myObject 对象的构造函数的引用。

```
var my_str:String = new String("sven");
trace(my_str.constructor == String); //output: true
```

如果使用 instanceof 运算符，还可以确定某对象是否属于指定的类：

```
var my_str:String = new String("sven");
trace(my_str instanceof String); //output: true
```

但是，在下面的示例中，Object.constructor 属性将原始数据类型（如下面列出的字符串）转换为包装对象。instanceof 运算符不执行任何转换，如下面的示例所示：

```
var my_str:String = "sven";
trace(my_str.constructor == String); //output: true
trace(my_str instanceof String); //output: false
```

另请参见

[instanceof 运算符](#)

## hasOwnProperty (Object.hasOwnProperty 方法)

```
public hasOwnProperty(name:String) : Boolean
```

指示对象是否已经定义了指定的属性。如果目标对象具有与 `name` 参数指定的字符串匹配的属性，则此方法返回 `true`；否则返回 `false`。此方法不检查对象的原型链，并且只在该属性存在于对象本身上时才返回 `true`。

可用性: **ActionScript 1.0**；Flash Player 6

### 参数

**name:String** -

### 返回

**Boolean** - 一个布尔值: 如果目标对象具有 `name` 参数指定的属性，则为 `true`；否则为 `false`。

## isPropertyEnumerable (Object.isPropertyEnumerable 方法)

```
public isPropertyEnumerable(name:String) : Boolean
```

指示指定的属性是否存在、是否可枚举。如果为 `true`，则该属性存在并且可以在 **for..in** 循环中枚举。该属性必须存在于目标对象上，原因是：该方法不检查目标对象的原型链。

您创建的属性是可枚举的，但是内置属性通常是不可枚举的。

可用性: **ActionScript 1.0**；Flash Player 6

### 参数

**name:String** -

### 返回

**Boolean** - 一个布尔值: 如果 `name` 参数指定的属性是可枚举的，则返回 `true`。

### 示例

下面的示例创建一个一般对象，向该对象添加属性，然后检查该对象是否是可枚举的。通过对比，该示例还说明内置属性 `Array.length` 是不可枚举的。

```
var myObj:Object = new Object();
myObj.prop1 = "hello";
trace(myObj.isPropertyEnumerable("prop1")); // Output: true

var myArray = new Array();
trace(myArray.isPropertyEnumerable("length")); // Output: false
```

另请参见

[for..in 语句](#)

## isPrototypeOf (Object.isPrototypeOf 方法)

```
public isPrototypeOf(theClass:Object) : Boolean
```

指示 **Object** 类的实例是否在指定为参数的对象的原型链中。如果该对象位于由 `theClass` 参数指定的对象的原型链中，则此方法返回 `true`。不仅在 `theClass` 对象的原型链中缺少目标对象时，而且当 `theClass` 参数不是对象时，此方法均返回 `false`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

`theClass:Object` -

### 返回

**Boolean** - 一个布尔值: 如果对象位于由 `theClass` 参数指定的对象的原型链中，则返回 `true` ; 否则返回 `false`。

## Object 构造函数

```
public Object()
```

创建 **Object** 对象，并将对该对象的构造函数方法的引用存储在该对象的 `constructor` 属性中。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例创建一个名为 `myObject` 的一般对象:

```
var myObject:Object = new Object();
```

## \_\_proto\_\_ (Object.\_\_proto\_\_ 属性)

```
public __proto__ : Object
```

引用用于创建对象的类 (ActionScript 2.0) 或构造函数 (ActionScript 1.0) 的 prototype 属性。在创建对象时，会自动将 \_\_proto\_\_ 属性分配给所有对象。ActionScript 解释程序使用 \_\_proto\_\_ 属性来访问对象的类或构造函数的 prototype 属性，以便弄清楚该对象从其超类中继承了什么属性和方法。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

以下示例创建一个名为 **Shape** 的类以及名为 **Circle** 的 **Shape** 的子类。

```
// Shape class defined in external file named Shape.as
class Shape {
    function Shape() {}
}

// Circle class defined in external file named Circle.as
class Circle extends Shape{
    function Circle() {}
}
```

**Circle** 类可用于创建 **Circle** 的两个实例：

```
var oneCircle:Circle = new Circle();
var twoCircle:Circle = new Circle();
```

下面的 **trace** 语句说明这两个实例的 \_\_proto\_\_ 都引用 **Circle** 类的 prototype 属性。

```
trace(Circle.prototype == oneCircle.__proto__); // Output: true
trace(Circle.prototype == twoCircle.__proto__); // Output: true
```

另请参见

[prototype \(Object.prototype 属性\)](#)



## prototype（Object.prototype 属性）

`public static prototype : Object`

对类或函数对象的超类的引用。`prototype` 属性会自动创建并附加到您所创建的任何类或函数对象。此属性是静态的，原因在于：它特定于您所创建的类或函数。例如，如果您创建自定义类，则 `prototype` 属性的值由类的所有实例共享，并且只能将其作为类属性进行访问。自定义类的实例无法直接访问 `prototype` 属性，但可通过 `__proto__` 属性访问它。

可用性：ActionScript 1.0；Flash Player 6

### 示例

以下示例创建一个名为 **Shape** 的类以及名为 **Circle** 的 **Shape** 的子类。

```
// Shape class defined in external file named Shape.as
class Shape {
    function Shape() {}
}

// Circle class defined in external file named Circle.as
class Circle extends Shape{
    function Circle() {}
}
```

**Circle** 类可用于创建 **Circle** 的两个实例：

```
var oneCircle:Circle = new Circle();
var twoCircle:Circle = new Circle();
```

下面的 **trace** 语句说明 **Circle** 类的 `prototype` 属性指向它的超类 **Shape**。标识符 `Shape` 引用 **Shape** 类的构造函数。

```
trace(Circle.prototype.constructor == Shape); // Output: true
```

下面的 **trace** 语句说明如何将 `prototype` 属性和 `__proto__` 属性一起使用以便在继承层次结构（或原型链）中上移两层。`Circle.prototype.__proto__` 属性包含对 **Shape** 类的超类的引用。

```
trace(Circle.prototype.__proto__ == Shape.prototype); // Output: true
```

另请参见

[\\_\\_proto\\_\\_（Object.\\_\\_proto\\_\\_ 属性）](#)

## registerClass (Object.registerClass 方法)

```
public static registerClass(name:String, theClass:Function) : Boolean
```

将影片剪辑元件与 **ActionScript** 对象类相关联。如果不存在元件，则 **Flash** 在字符串标识符和对象类之间创建关联。

在将指定影片剪辑元件的实例放在时间轴上时，该实例会注册到由 `theClass` 参数指定的类，而不是注册到 **MovieClip** 类。

使用 `MovieClip.attachMovie()` 或 `MovieClip.duplicateMovieClip()` 创建指定影片剪辑元件的实例时，该实例注册到由 `theClass` 指定的类，而不是注册到 **MovieClip** 类。如果 `theClass` 为 `null`，则此方法将删除所有与指定影片剪辑元件或类标识符相关联的任何 **ActionScript** 类定义。对于影片剪辑元件，该影片剪辑的任何现有实例都将保持不变，但该元件的新实例将与默认类 **MovieClip** 相关联。

如果元件已注册到某个类，则此方法会用新注册对其进行替换。

如果将影片剪辑实例通过时间轴放置或是通过使用 `attachMovie()` 或 `duplicateMovieClip()` 创建，则 **ActionScript** 会调用相应类的构造函数，并且关键字 `this` 指向该对象。此构造函数在调用时不带有参数。

如果使用此方法用 **ActionScript** 类而不是 **MovieClip** 进行影片剪辑注册，则该影片剪辑元件不会继承内置 **MovieClip** 类的方法、属性和事件，除非将 **MovieClip** 类包括在新类的原型链中。以下代码创建一个名为 `theClass` 的新 **ActionScript** 类，它可继承 **MovieClip** 类的属性：

```
theClass.prototype = new MovieClip();
```

可用性：ActionScript 1.0；Flash Player 6

### 参数

**name:String** - 字符串：影片剪辑元件的链接标识符或 **ActionScript** 类的字符串标识符。

**theClass:Function** - 对 **ActionScript** 类的构造函数的引用；如果为 `null`，则取消注册元件。

### 返回

**Boolean** - 一个布尔值：如果类注册成功，则返回 `true` 值；否则返回 `false`。

### 另请参见

[attachMovie](#) ([MovieClip.attachMovie](#) 方法)，[duplicateMovieClip](#) ([MovieClip.duplicateMovieClip](#) 方法)

## \_\_resolve (Object.\_\_resolve 属性)

public \_\_resolve : Object

对用户定义的函数的引用, 该函数在 **ActionScript** 代码引用未定义的属性或方法时调用。如果 **ActionScript** 代码引用对象的未定义的属性或方法, 则 **Flash Player** 会确定该对象的 **\_\_resolve** 属性是否已定义。如果定义了 **\_\_resolve**, 则执行它所引用的函数, 并传递未定义的属性或方法的名称。这允许您以程序化方式为未定义属性提供值并为未定义方法提供语句, 就好象实际上已经定义了这些属性或方法。此属性对启用高度透明的客户端 / 服务器通信很有用, 并且是调用服务器端方法的推荐方式。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例是基于第一个示例以渐进的方式生成的, 并说明了 **\_\_resolve** 属性的五种不同用法。为了帮助理解, 不同于上一种用法的关键语句使用粗体。

用法 1: 下面的示例使用 **\_\_resolve** 生成一个对象, 在该对象中, 每个未定义的属性均返回 值 "Hello, world!"。

```
// instantiate a new object
var myObject:Object = new Object();

// define the __resolve function
myObject.__resolve = function (name) {
    return "Hello, world!";
};
trace (myObject.property1); // output: Hello, world!
trace (myObject.property2); // output: Hello, world!
```

用法 2: 下面的示例使用 **\_\_resolve** 作为函子 (即生成函数的函数)。使用 **\_\_resolve** 将未定义的方法调用重定向到名为 **myFunction** 的一般函数。

```
// instantiate a new object
var myObject:Object = new Object();

// define a function for __resolve to call
myObject.myFunction = function (name) {
    trace("Method " + name + " was called");
};

// define the __resolve function
myObject.__resolve = function (name) {
    return function () { this.myFunction(name); };
};

// test __resolve using undefined method names
myObject.someMethod(); // output: Method someMethod was called
myObject.someOtherMethod(); //output: Method someOtherMethod was called
```

用法 3: 下面的示例建立在上一个示例的基础之上, 增加了缓存已解析的方法的功能。通过缓存方法, 对于每个相关的方法, 只调用 `__resolve` 一次。这允许对象方法的迟缓构造。迟缓构造是将方法的创建 (即构造) 推迟到第一次使用方法时进行的优化技术。

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
    trace("Method "+name+" was called");
};
// define the __resolve function
myObject.__resolve = function(name) {
    trace("Resolve called for "+name); // to check when __resolve is called
    // Not only call the function, but also save a reference to it
    var f:Function = function () {
        this.myFunction(name);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference
    return f;
};
// test __resolve using undefined method names
// __resolve will only be called once for each method name
myObject.someMethod(); // calls __resolve
myObject.someMethod(); // does not call __resolve because it is now defined
myObject.someOtherMethod(); // calls __resolve
myObject.someOtherMethod(); // does not call __resolve, no longer undefined
```

用法 4: 下面的示例建立在上一个示例的基础之上, 保留方法名称 `onStatus()` 以在本地使用, 从而使其解析方法不同于其它未定义的属性。添加的代码使用粗体。

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
    trace("Method "+name+" was called");
};
// define the __resolve function
myObject.__resolve = function(name) {
    // reserve the name "onStatus" for local use
    if (name == "onStatus") {
        return undefined;
    }
    trace("Resolve called for "+name); // to check when __resolve is called
    // Not only call the function, but also save a reference to it
    var f:Function = function () {
        this.myFunction(name);
    };
    // create a new object method and assign it the reference
    this[name] = f;
}
```

```

    // return the reference
    return f;
};
// test __resolve using the method name "onStatus"
trace(myObject.onStatus("hello"));
// output: undefined

```

用法 5：下面的示例建立在上一个示例的基础之上，创建了接受参数的函子。此示例广泛使用参数对象，并且使用了 **Array** 类的一些方法。

```

// instantiate a new object
var myObject:Object = new Object();

// define a generic function for __resolve to call
myObject.myFunction = function (name) {
    arguments.shift();
    trace("Method " + name + " was called with arguments: " +
        arguments.join(', '));
};

// define the __resolve function
myObject.__resolve = function (name) {
    // reserve the name "onStatus" for local use
    if (name == "onStatus") {
        return undefined;
    }
    var f:Function = function () {
        arguments.unshift(name);
        this.myFunction.apply(this, arguments);
    };
    // create a new object method and assign it the reference
    this[name] = f;
    // return the reference to the function
    return f;
};

// test __resolve using undefined method names with parameters
myObject.someMethod("hello");
// output: Method someMethod was called with arguments: hello

myObject.someOtherMethod("hello", "world");
// output: Method someOtherMethod was called with arguments: hello, world

```

另请参见

[arguments](#), [Array](#)

## toString (Object.toString 方法)

```
public toString() : String
```

将指定对象转换为字符串然后返回它。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

String - 一个字符串。

### 示例

此示例显示某个一般对象的 **toString()** 的返回值:

```
var myObject:Object = new Object();  
trace(myObject.toString()); // output: [object Object]
```

可以覆盖此方法以返回一个更有意义的值。下面的示例说明已经针对内置类 **Date**、**Array** 和 **Number** 覆盖了此方法:

```
// Date.toString() returns the current date and time  
var myDate:Date = new Date();  
trace(myDate.toString()); // output: [current date and time]  
  
// Array.toString() returns the array contents as a comma-delimited string  
var myArray:Array = new Array("one", "two");  
trace(myArray.toString()); // output: one,two  
  
// Number.toString() returns the number value as a string  
// Because trace() won't tell us whether the value is a string or number  
// we will also use typeof() to test whether toString() works.  
var myNumber:Number = 5;  
trace(typeof (myNumber)); // output: number  
trace(myNumber.toString()); // output: 5  
trace(typeof (myNumber.toString())); // output: string
```

下面的示例说明如何覆盖自定义类中的 **toString()**。首先创建一个名为 **Vehicle.as** 的文本文件（只包含 **Vehicle** 类定义）并将它置于 **Configuration** 文件夹中的 **Classes** 文件夹中。

```
// contents of Vehicle.as  
class Vehicle {  
    var numDoors:Number;  
    var color:String;  
    function Vehicle(param_numDoors:Number, param_color:String) {  
        this.numDoors = param_numDoors;  
        this.color = param_color;  
    }  
    function toString():String {  
        var doors:String = "door";  
        if (this.numDoors > 1) {  
            doors += "s";  
        }  
    }  
}
```

```

    }
    return ("A vehicle that is " + this.color + " and has " + this.numDoors +
        " " + doors);
    }
}

// code to place into a FLA file
var myVehicle:Vehicle = new Vehicle(2, "red");
trace(myVehicle.toString());
// output: A vehicle that is red and has 2 doors

// for comparison purposes, this is a call to valueOf()
// there is no primitive value of myVehicle, so the object is returned
// giving the same output as toString().
trace(myVehicle.valueOf());
// output: A vehicle that is red and has 2 doors

```

## unwatch (Object.unwatch 方法)

public unwatch(name:String) : Boolean

删除 Object.watch() 创建的监视点。如果删除监视点成功，则此方法返回值 true ； 否则返回值 false。

**可用性:** ActionScript 1.0 ； Flash Player 6

### 参数

**name:String** - 一个字符串； 不应再监视的对象属性的名称。

### 返回

Boolean - 一个布尔值： 如果删除监视点成功，则为 true ； 否则为 false。

### 示例

请参见 Object.watch() 的示例。

### 另请参见

[watch \(Object.watch 方法\)](#)， [addProperty \(Object.addProperty 方法\)](#)

## valueOf (Object.valueOf 方法)

`public valueOf() : Object`

返回指定对象的原始值。如果此对象没有基元值，则返回该对象。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

`Object` - 指定对象的原始值或对象本身。

### 示例

下面的示例显示某个通用对象的 `valueOf()`（不具有原始值）的返回值并将其与 `toString()` 的返回值进行比较。首先，创建一个通用对象。其次，创建一个新 `Date` 数据对象，将其设置为 2004 年 2 月 1 日上午 8:15。`toString()` 方法以可读格式返回当前时间。`valueOf()` 方法返回原始值，以毫秒表示。最后，创建一个包含两个简单元素的新 `Array` 对象。`toString()` 和 `valueOf()` 均返回相同的值: **one,two**:

```
// Create a generic object
var myObject:Object = new Object();
trace(myObject.valueOf()); // output: [object Object]
trace(myObject.toString()); // output: [object Object]
```

下面的示例显示内置类 `Date` 和 `Array` 的返回值，并将其与 `Object.toString()` 的返回值进行比较:

```
// Create a new Date object set to February 1, 2004, 8:15 AM
// The toString() method returns the current time in human-readable form
// The valueOf() method returns the primitive value in milliseconds
var myDate:Date = new Date(2004,01,01,8,15);
trace(myDate.toString()); // output: Sun Feb 1 08:15:00 GMT-0800 2004
trace(myDate.valueOf()); // output: 1075652100000

// Create a new Array object containing two simple elements
// In this case both toString() and valueOf() return the same value: one,two
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two
trace(myArray.valueOf()); // output: one,two
```

若想了解覆盖 `toString()` 的自定义类的 `Object.valueOf()` 的返回值的示例，请参见 `Object.toString()` 的示例。

### 另请参见

[toString \(Object.toString 方法\)](#)



## watch (Object.watch 方法)

```
public watch(name:String, callback:Function, [userData:Object]) : Boolean
```

注册当 **ActionScript** 对象的指定属性更改时要调用的事件处理函数。当该属性发生更改时，用 myObject 作为包含对象调用此事件处理函数。

您可以在 callback 方法定义中使用 return 语句来影响正在监视的属性的值。由 callback 方法返回的值将分配给被监视对象的属性。您选择要返回的值取决于您是想对属性进行监视、修改还是阻止更改：

- 如果您只是要监视该属性，则返回 newVal 参数。
- 如果您要修改该属性的值，则返回修改后的值。
- 如果您要阻止对该属性进行更改，则返回 oldVal 参数。

如果您定义的 callback 方法没有 return 语句，则会被监视的对象属性分配值 undefined。

通过返回已修改的 newVal（或 oldval），监视点可以过滤（或置空）分配值。如果删除已为其设置了监视点的属性，则该监视点并不消失。如果以后重新创建此属性，则该监视点依然起作用。若要删除一个监视点，请使用 Object.unwatch 方法。

在一个属性上只能注册单个监视点。在同一属性上，对 Object.watch() 的后续调用会替换原来的监视点。

Object.watch() 方法的行为类似于 JavaScript 1.2 和更高版本中的 Object.watch() 函数。主要区别在于 userData 参数，该参数是 Flash 对 Object.watch() 的增补，Netscape Navigator 不支持此参数。可将 userData 参数传递给事件处理函数，并在该事件处理函数中使用它。

Object.watch() 方法无法监视 **getter/setter** 属性。**Getter/setter** 属性以懒惰评估的方式进行操作，即直到实际查询该属性时才确定该属性的值。“懒惰评估”常常效率很高，因为该属性并不需要不断地更新；而是在需要时才进行评估。但是，Object.watch() 需要评估属性以确定是否调用 callback 函数。若要使用 **getter/setter** 属性，Object.watch() 需要不断地评估该属性，但这样做的效率很低。

通常，预定义的 **ActionScript** 属性（如 \_x、\_y、\_width 和 \_height）是 **getter/setter** 属性，且不能用 Object.watch() 进行监视。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**name:**String - 一个字符串；要监视的对象属性的名称。

**callback:**Function - 当监视的属性发生更改时要调用的函数。此参数为函数对象，而非字符串形式的函数名。callback 的格式为 callback(prop, oldVal, newVal, userData)。

**userData:**Object [ 可选 ] - 传递给 callback 方法的 **ActionScript** 数据的任意片段。如果省略 userData 参数，则将 undefined 传递给 **callback** 方法。

## 返回

Boolean - 一个布尔值：如果创建监视点成功，则为 true；否则为 false。

## 示例

下面的示例使用 watch() 检查 speed 属性是否超过了速度限制：

```
// Create a new object
var myObject:Object = new Object();

// Add a property that tracks speed
myObject.speed = 0;

// Write the callback function to be executed if the speed property changes
var speedWatcher:Function = function(prop, oldVal, newVal, speedLimit) {
    // Check whether speed is above the limit
    if (newVal > speedLimit) {
        trace ("You are speeding.");
    }
    else {
        trace ("You are not speeding.");
    }

    // Return the value of newVal.
    return newVal;
}

// Use watch() to register the event handler, passing as parameters:
// - the name of the property to watch: "speed"
// - a reference to the callback function speedWatcher
// - the speedLimit of 55 as the userData parameter
myObject.watch("speed", speedWatcher, 55);

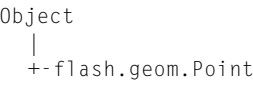
// set the speed property to 54, then to 57
myObject.speed = 54; // output: You are not speeding
myObject.speed = 57; // output: You are speeding

// unwatch the object
myObject.unwatch("speed");
myObject.speed = 54; // there should be no output
```

## 另请参见

[addProperty](#) ([Object.addProperty](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法)

# Point (flash.geom.Point)



```
public class Point
extends Object
```

**Point** 类表示二维坐标系中的一个位置，其中 **x** 表示水平轴， **y** 表示垂直轴。

下面的代码在 (0,0) 处创建一个点：

```
var myPoint:Point = new Point();
```

可用性： **ActionScript 1.0** ； **Flash Player 8**

## 属性摘要

修饰符	属性	说明
	length:Number	从 (0,0) 到此点的线段长度。
	x:Number	该点的水平坐标。
	y:Number	该点的垂直坐标。

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 构造函数摘要

签名	说明
Point(x:Number, y:Number)	创建一个新点。

方法摘要

修饰符	签名	说明
	add(v:Point) : Point	将另一个点的坐标添加到此点的坐标以创建一个新点。
	clone() : Point	创建此 Point 对象的副本。
static	distance(pt1:Point, pt2:Point) : Number	返回 pt1 和 pt2 之间的距离。
	equals(toCompare:Object) : Boolean	确定两个点是否相同。
static	interpolate(pt1:Point, pt2:Point, f:Number) : Point	确定两个指定点之间的点。
	normalize(length:Number) : Void	将 (O,O) 和当前点之间的线段缩放为设定的长度。
	offset(dx:Number, dy:Number) : Void	按指定量偏移 Point 对象。
static	polar(len:Number, angle:Number) : Point	将一对极坐标转换为笛卡尔点坐标。
	subtract(v:Point) : Point	从此点的坐标中减去另一个点的坐标以创建一个新点。
	toString() : String	返回包含 x 和 y 坐标的值的字符串。

继承自 Object 类的方法

addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)

## add (Point.add 方法)

```
public add(v:Point) : Point
```

将另一个点的坐标添加到此点的坐标以创建一个新点。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**v**:flash.geom.Point - 要添加的点。

### 返回

flash.geom.Point - 新点。

### 示例

下面的示例通过向 point\_1 添加 point\_2 来创建 **Point** 对象 resultPoint。

```
import flash.geom.Point;
var point_1:Point = new Point(4, 8);
var point_2:Point = new Point(1, 2);
var resultPoint:Point = point_1.add(point_2);
trace(resultPoint.toString()); // (x=5, y=10)
```

## clone (Point.clone 方法)

```
public clone() : Point
```

创建此 **Point** 对象的副本。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

flash.geom.Point - 新的 **Point** 对象。

### 示例

下面的示例根据 myPoint 对象中的值创建名为 clonedPoint 的 **Point** 对象的副本。

clonedPoint 对象包含 myPoint 中的所有值，但它们不是同一个对象。

```
import flash.geom.Point;
var myPoint:Point = new Point(1, 2);
var clonedPoint:Point = myPoint.clone();
trace(clonedPoint.x); // 1
trace(clonedPoint.y); // 2
trace(myPoint.equals(clonedPoint)); // true
trace(myPoint === clonedPoint); // false
```

## distance (Point.distance 方法)

`public static distance(pt1:Point, pt2:Point) : Number`

返回 **pt1** 和 **pt2** 之间的距离。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**pt1**:flash.geom.Point - 第一个点。

**pt2**:flash.geom.Point - 第二个点。

### 返回

Number - 第一个点和第二个点之间的距离。

### 示例

下面的示例创建 `point_1` 和 `point_2`，然后确定它们之间的距离 (`distanceBetween`)。

```
import flash.geom.Point;
var point_1:Point = new Point(-5, 0);
var point_2:Point = new Point(5, 0);
var distanceBetween:Number = Point.distance(point_1, point_2);
trace(distanceBetween); // 10
```

## equals (Point.equals 方法)

`public equals(toCompare:Object) : Boolean`

确定两个点是否相同。如果两个点具有相同的 **x** 和 **y** 值，则它们是相同的。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**toCompare**:Object - 要比较的点。

### 返回

Boolean - 如果该对象与此 **Point** 对象相同，则为 `true`；如果不相同，则为 `false`。

## 示例

下面的示例确定一个点的值是否等于另一个点的值。如果对象相同，则 `equals()` 返回的结果与全等运算符 (`===`) 返回的结果不相同。

```
import flash.geom.Point;
var point_1:Point = new Point(1, 2);
var point_2:Point = new Point(1, 2);
var point_3:Point = new Point(4, 8);
trace(point_1.equals(point_2)); // true
trace(point_1.equals(point_3)); // false
trace(point_1 === point_2); // false
trace(point_1 === point_3); // false
```

## interpolate (Point.interpolate 方法)

`public static interpolate(pt1:Point, pt2:Point, f:Number) : Point`

确定两个指定点之间的点。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**pt1:**flash.geom.Point - 第一个点。

**pt2:**flash.geom.Point - 第二个点。

**f:**Number - 两个点之间的内插级别。指示新点将位于 **pt1** 和 **pt2** 连成的直线上的什么位置。如果 **f=0**，则返回 **pt1**；如果 **f=1**；则返回 **pt2**。

### 返回

flash.geom.Point - 新的内插点。

## 示例

下面的示例将内插点 (`interpolatedPoint`) 放在 `point_1` 和 `point_2` 之间的一半位置处 (50%)。

```
import flash.geom.Point;
var point_1:Point = new Point(-100, -100);
var point_2:Point = new Point(50, 50);
var interpolatedPoint:Point = Point.interpolate(point_1, point_2, .5);
trace(interpolatedPoint.toString()); // (x=-25, y=-25)
```

## length (Point.length 属性)

public length : Number

从 (0,0) 到此点的线段长度。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建一个 **Point** 对象 myPoint，并确定从 (0, 0) 到该 **Point** 的直线的长度。

```
import flash.geom.Point;
var myPoint:Point = new Point(3,4);
trace(myPoint.length); // 5
```

另请参见

[polar \(Point.polar 方法\)](#)

## normalize (Point.normalize 方法)

public normalize(length:Number) : Void

将 (0,0) 和当前点之间的线段缩放为设定的长度。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**length**:Number - 缩放值。例如，如果当前点为 (0,5) 并且您将它规范化为 1，则返回的点位于 (0,1) 处。

### 示例

下面的示例将 normalizedPoint 对象的长度从 5 延长至 10。

```
import flash.geom.Point;
var normalizedPoint:Point = new Point(3, 4);
trace(normalizedPoint.length); // 5
trace(normalizedPoint.toString()); // (x=3, y=4)
normalizedPoint.normalize(10);
trace(normalizedPoint.length); // 10
trace(normalizedPoint.toString()); // (x=6, y=8)
```

另请参见

[length \(Point.length 属性\)](#)



## offset (Point.offset 方法)

```
public offset(dx:Number, dy:Number) : Void
```

按指定量偏移 **Point** 对象。dx 的值将添加到 x 的原始值中以创建新的 x 值。dy 的值将添加到 y 的原始值中以创建新的 y 值。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**dx**:Number - 偏移水平坐标 x 的量。

**dy**:Number - 偏移垂直坐标 y 的量。

### 示例

下面的示例将点的位置偏移指定的 x 和 y 的量。

```
import flash.geom.Point;
var myPoint:Point = new Point(1, 2);
trace(myPoint.toString()); // (x=1, y=2)
myPoint.offset(4, 8);
trace(myPoint.toString()); // (x=5, y=10)
```

### 另请参见

[add \(Point.add 方法\)](#)

## Point 构造函数

```
public Point(x:Number, y:Number)
```

创建一个新点。如果不向此方法传递任何参数，则在 (0,0) 处创建一个点。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**x**:Number - 水平坐标。默认值是 0。

**y**:Number - 垂直坐标。默认值是 0。

### 示例

第一个示例使用默认构造函数创建 **Point** 对象 point\_1。

```
import flash.geom.Point;
var point_1:Point = new Point();
trace(point_1.x); // 0
trace(point_1.y); // 0
```

第二个示例使用坐标  $x = 1$  且  $y = 2$  创建 **Point** 对象 point\_2。

```
import flash.geom.Point;
var point_2:Point = new Point(1, 2);
trace(point_2.x); // 1
trace(point_2.y); // 2
```

## polar（Point.polar 方法）

public static polar(len:Number, angle:Number) : Point

将一对极坐标转换为笛卡尔点坐标。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**len**:Number - 极坐标对的长度。

**angle**:Number - 极坐标对的角度（以弧度表示）。

### 返回

flash.geom.Point - 笛卡尔点。

### 示例

下面的示例根据 cartesianPoint 的值创建一个 **Point** 对象 cartesianPoint 并创建一条长度为 5 的直线。由于直角三角形的特征是三边比例为 3:4:5，因此使用等于 **Math.atan(3/4)** 的 angleInRadians 值。

```
import flash.geom.Point;
var len:Number = 5;
var angleInRadians:Number = Math.atan(3/4);
var cartesianPoint:Point = Point.polar(len, angleInRadians);
trace(cartesianPoint.toString()); // (x=4, y=3)
```

当计算机使用超越数（如 **pi**）时，由于浮点算法只有有限精度，可能会出现一些舍入错误。使用 Math.PI 时，请考虑使用 Math.round() 函数，如下面的示例所示。

```
import flash.geom.Point;
var len:Number = 10;
var angleInRadians:Number = Math.PI;
var cartesianPoint:Point = Point.polar(len, angleInRadians);
trace(cartesianPoint.toString()); // should be (x=-10, y=0), but is (x=-10,
    y=1.22460635382238e-15)
trace(Math.round(cartesianPoint.y)); // 0
```

### 另请参见

[length（Point.length 属性）](#)，[round（Math.round 方法）](#)

## subtract (Point.subtract 方法)

```
public subtract(v:Point) : Point
```

从此点的坐标中减去另一个点的坐标以创建一个新点。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 参数

**v:**flash.geom.Point - 要减去的点。

### 返回

flash.geom.Point - 新点。

### 示例

下面的示例通过从 point\_1 中减去 point\_2 来创建 point\_3。

```
import flash.geom.Point;
var point_1:Point = new Point(4, 8);
var point_2:Point = new Point(1, 2);
var resultPoint:Point = point_1.subtract(point_2);
trace(resultPoint.toString()); // (x=3, y=6)
```

## toString (Point.toString 方法)

```
public toString() : String
```

返回包含 **x** 和 **y** 坐标的值的字符串。它的格式为 (x, y)，因此 23,17 处的 Point 将报告 "(x=23, y=17)"。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 返回

String - 一个字符串。

### 示例

下面的示例创建一个点，并将它的值转换为 (x=x, y=y) 格式的字符串。

```
import flash.geom.Point;
var myPoint:Point = new Point(1, 2);
trace("myPoint: " + myPoint.toString()); // (x=1, y=2)
```

## x (Point.x 属性)

`public x : Number`

该点的水平坐标。默认值是 0。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例创建一个 **Point** 对象 `myPoint` 并设置 **x** 坐标值。

```
import flash.geom.Point;
var myPoint:Point = new Point();
trace(myPoint.x); // 0
myPoint.x = 5;
trace(myPoint.x); // 5
```

## y (Point.y 属性)

`public y : Number`

该点的垂直坐标。默认值是 0。

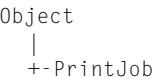
可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例创建一个 **Point** 对象 `myPoint` 并设置 **y** 坐标值。

```
import flash.geom.Point;
var myPoint:Point = new Point();
trace(myPoint.y); // 0
myPoint.y = 5;
trace(myPoint.y); // 5
```

# PrintJob



```
public class PrintJob
extends Object
```

**PrintJob** 类用于创建内容并将其打印为一页或多页。除了对 `print()` 方法提供的打印功能进行改进之外，此类还允许您呈现屏幕上未显示的动态内容、通过单个“打印”对话框提示用户，并且可以使用与内容相同的比例打印未经缩放的文档。此功能特别适用于呈现和打印动态内容，例如数据库内容和动态文本。

另外，使用由 `PrintJob.start()` 填充的属性，文档可以读取您用户的打印机设置（如页高度、宽度和方向），并且您可以配置文档以动态方式设置适用于打印机设置的 **Flash** 内容的格式。这些用户布局属性是只读的，**Flash Player** 不能对其进行更改。

可用性：ActionScript 1.0；Flash Player 7

## 属性摘要

修饰符	属性	说明
	<code>orientation:String</code> [ 只读 ]	图像的打印方向。
	<code>pageHeight:Number</code> [ 只读 ]	页面上实际可打印区域的高度（以磅为单位）。
	<code>pageWidth:Number</code> [ 只读 ]	页面上实际可打印区域的宽度（以磅为单位）。
	<code>paperHeight:Number</code> [ 只读 ]	纸张总高度（以磅为单位）。
	<code>paperWidth:Number</code> [ 只读 ]	纸张总宽度（以磅为单位）。

继承自 `Object` 类的属性

[constructor](#) (`Object.constructor` 属性), [\\_\\_proto\\_\\_](#) (`Object.__proto__` 属性), [prototype](#) (`Object.prototype` 属性), [\\_\\_resolve](#) (`Object.__resolve` 属性)

## 构造函数摘要

签名	说明
<code>PrintJob()</code>	创建一个可用于打印一页或多页的 <code>PrintJob</code> 对象。

方法摘要

修饰符	签名	说明
	<code>addPage(target:Object, [printArea:Object], [options:Object], [frameNum:Number]) : Boolean</code>	将指定的级别或影片剪辑作为单个页发送到打印后台处理程序。
	<code>send() : Void</code>	用在 <code>PrintJob.start()</code> 和 <code>PrintJob.addPage()</code> 方法之后，将经过后台处理的页发送到打印机。
	<code>start() : Boolean</code>	显示操作系统的打印对话框并开始进行后台处理。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

addPage (PrintJob.addPage 方法)

```
public addPage(target:Object, [printArea:Object], [options:Object],
               [frameNum:Number]) : Boolean
```

将指定的级别或影片剪辑作为单个页发送到打印后台处理程序。在使用此方法前，您必须使用 `PrintJob.start()`；在对打印作业调用了一次或多次 `PrintJob.addPage()` 后，您必须使用 `PrintJob.send()` 将经过后台处理的页发送到打印机。

如果此方法返回 `false`（例如，如果您尚未调用 `PrintJob.start()` 或者用户取消了打印作业），则对 `PrintJob.addPage()` 的任何后续调用都将失败。但是，如果此前对 `PrintJob.addPage()` 的调用成功，则最后的 `PrintJob.send()` 命令会将经后台成功处理的页发送到打印机。

如果您为 `printArea` 传递了值，则 `xMin` 和 `yMin` 坐标将映射到页上可打印区域的左上角（`0,0` 坐标）。用户的可打印区域通过由 `PrintJob.start()` 设置的只读 `pageHeight` 和 `pageWidth` 属性进行描述。因为打印输出与页上可打印区域的左上角对齐，所以如果 `printArea` 中定义的区域大于页上的可打印区域，则打印输出会在右边和 / 或底部被截断。如果尚未为 `printArea` 传递值并且舞台大于可打印区域，则会发生同样类型的截断操作。

如果要在打印前对影片剪辑进行缩放，可在调用此方法前设置其 `MovieClip._xscale` 和 `MovieClip._yscale` 属性，并在打印后将它们重新设置为原始值。影片剪辑的缩放与 `printArea` 无关。也就是说，如果指定打印大小为 **50 x 50** 像素的区域，则将打印 **2500** 像素。如果对影片剪辑进行了缩放，同样会打印 **2500** 像素，但是将按缩放的大小进行打印。

Flash Player 打印功能支持 **PostScript** 和非 **PostScript** 打印机。非 **PostScript** 打印机将矢量转换成位图。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**target:Object** - 一个数字或字符串；要打印的影片剪辑的级别或实例名称。传递一个数字来指定级别（例如，**0** 表示 `_root` 影片）或传递一个字符串（括在引号中 `[""]`）来指定影片剪辑的实例名称。

**printArea:Object** [ 可选 ] - 一个对象，它指定要打印的区域，采用以下格式：

```
{xMin:topLeft, xMax:topRight, yMin:bottomLeft, yMax:bottomRight}
```

为 `printArea` 指定的坐标表示屏幕像素，这些屏幕像素相对于 `_root` 影片剪辑（如果 `target = 0`）的注册点或由 `target` 指定的级别或影片剪辑的注册点。您必须提供所有四个坐标。宽度 (`xMax-xMin`) 和高度 (`yMax-yMin`) 必须均大于 **0**。

磅是打印度量单位，像素是屏幕度量单位；磅的实际大小是固定的（**1/72** 英寸），但是像素的大小取决于特定屏幕的分辨率。您可以使用以下换算公式将英寸或厘米转换为缇或磅（**1** 缇为 **1/20** 磅）：

- **1 磅 = 1/72 英寸 = 20 缇**
- **1 英寸 = 72 磅 = 1440 缇**
- **1 厘米 = 567 缇**

您无法可靠地在像素和磅之间转换；转换比率取决于屏幕及其分辨率。例如，如果屏幕设置为每英寸显示 **72** 个像素，则一磅等于一个像素。



如果以前曾使用 `print()`、`printAsBitmap()`、`printAsBitmapNum()` 或 `printNum()` 从 Flash 进行打印，则可能使用过 `#b` 帧标签来指定要打印的区域。当使用 `addPage()` 方法时，您必须使用 `printArea` 参数来指定打印区域；`#b` 帧标签将被忽略。

如果省略了 `printArea` 参数或错误地传递了该参数，则将打印 `target` 的整个舞台区域。如果您不希望指定 `printArea` 的值，但希望指定 `options` 或 `frameNumber` 的值，请为 `printArea` 传递 `null`。

**options:Object** [ 可选 ] - 一个参数，它指定打印为矢量还是打印为位图，采用以下格式：

```
{printAsBitmap:Boolean}
```

默认值为 `false`，表示请求矢量打印。若要将 `target` 打印为位图，请为 `printAsBitmap` 传递 `true`。当确定要使用的值时，请记住以下建议：

- 如果要打印的内容包括位图图像，请使用 `{printAsBitmap:true}` 以包括任何透明度和色彩效果。
- 如果内容不包括位图图像，请省略此参数或者使用 `{printAsBitmap:false}` 以较高品质的矢量格式打印内容。

如果省略或错误传递了 `options`，则使用矢量打印。如果您不希望指定 `options` 的值，但希望指定 `frameNumber` 的值，请为 `options` 传递 `null`。

**frameNum: Number [ 可选 ]** - 一个数字，用于指定要打印的帧；传递 `frameNumber` 不会导致调用该帧上的 **ActionScript**。如果省略此参数，则打印 `target` 中的当前帧。

提醒

如果以前曾使用 `print()`、`printAsBitmap()`、`printAsBitmapNum()` 或 `printNum()` 从 Flash 进行打印，则可能在多个帧上使用过 `#p` 帧标签以指定要打印哪些页。若要使用 `PrintJob.addPage()` 打印多个帧，必须对每个帧发出 `PrintJob.addPage()` 命令；`#p` 帧标签将被忽略。有关以编程方式执行此操作的方法，请参见“示例”部分。

## 返回

**Boolean** - 一个布尔值：如果页成功发送到打印后台处理程序，则返回 `true`；否则返回 `false`。

## 示例

下面的示例说明了发出 `addPage()` 命令的多种方式：

```
my_btn.onRelease = function()
{
    var pageCount:Number = 0;

    var my_pj:PrintJob = new PrintJob();

    if (my_pj.start())
    {
        // Print entire current frame of the _root movie in vector format
        if (my_pj.addPage(0)){
            pageCount++;

            // Starting at 0,0, print an area 400 pixels wide and 500 pixels high
            // of the current frame of the _root movie in vector format
            if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500})){
                pageCount++;

                // Starting at 0,0, print an area 400 pixels wide and 500 pixels high
                // of frame 1 of the _root movie in bitmap format
                if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500},
                    {printAsBitmap:true}, 1)){
```



```

    pageCount++;

    // Starting 50 pixels to the right of 0,0 and 70 pixels down,
    // print an area 500 pixels wide and 600 pixels high
    // of frame 4 of level 5 in vector format
    if (my_pj.addPage(5, {xMin:50,xMax:550,yMin:70,yMax:670},null, 4)){
        pageCount++;

        // Starting at 0,0, print an area 400 pixels wide
        // and 400 pixels high of frame 3 of the "dance_mc" movie clip
        // in bitmap format
        if (my_pj.addPage("dance_mc",
            {xMin:0,xMax:400,yMin:0,yMax:400},{printAsBitmap:true}, 3)){
            pageCount++;

            // Starting at 0,0, print an area 400 pixels wide
            // and 600 pixels high of frame 3 of the "dance_mc" movie clip
            // in vector format at 50% of its actual size
            var x:Number = dance_mc._xscale;
            var y:Number = dance_mc._yscale;
            dance_mc._xscale = 50;
            dance_mc._yscale = 50;

            if (my_pj.addPage("dance_mc",
                {xMin:0,xMax:400,yMin:0,yMax:600},null, 3)){
                pageCount++;
            }
            dance_mc._xscale = x;
            dance_mc._yscale = y;
        }
    }
}
}
}

// If addPage() was successful at least once, print the spooled pages.
if (pageCount > 0){
    my_pj.send();
}
delete my_pj;
}

```

另请参见

[send \(PrintJob.send 方法\)](#), [start \(PrintJob.start 方法\)](#)

## orientation (PrintJob.orientation 属性)

`public orientation : String [read-only]`

图像的打印方向。此属性可以为 "landscape" 或 "portrait"。请注意，只有在调用 `PrintJob.start()` 方法后，此属性才可用。

可用性: `ActionScript 1.0` ; `Flash Player 7`

## pageHeight (PrintJob.pageHeight 属性)

`public pageHeight : Number [read-only]`

页面上实际可打印区域的高度（以磅为单位）。忽略用户设置的任何边距。请注意，只有在调用 `PrintJob.start()` 方法后，此属性才可用。

可用性: `ActionScript 1.0` ; `Flash Player 7`

## pageWidth (PrintJob.pageWidth 属性)

`public pageWidth : Number [read-only]`

页面上实际可打印区域的宽度（以磅为单位）。忽略用户设置的任何边距。请注意，只有在调用 `PrintJob.start()` 方法后，此属性才可用。

可用性: `ActionScript 1.0` ; `Flash Player 7`

## paperHeight (PrintJob.paperHeight 属性)

`public paperHeight : Number [read-only]`

纸张总高度（以磅为单位）。请注意，只有在调用 `PrintJob.start()` 方法后，此属性才可用。

可用性: `ActionScript 1.0` ; `Flash Player 7`

## paperWidth (PrintJob.paperWidth 属性)

`public paperWidth : Number [read-only]`

纸张总宽度（以磅为单位）。请注意，只有在调用 `PrintJob.start()` 方法后，此属性才可用。

可用性: `ActionScript 1.0` ; `Flash Player 7`

## PrintJob 构造函数

```
public PrintJob()
```

创建一个可用于打印一页或多页的 **PrintJob** 对象。

若要实现打印作业，请按顺序使用以下方法。您必须将与特定打印作业相关的所有命令（从构造函数到 `PrintJob.send()` 和删除）放在同一帧上。将 `my_pj.addPage()` 方法调用的 `[params]` 替换为您自定义的参数。

```
// create PrintJob object
var my_pj:PrintJob = new PrintJob();

// display print dialog box, but only initiate the print job
// if start returns successfully.
if (my_pj.start()) {

    // use a variable to track successful calls to addPage
    var pagesToPrint:Number = 0;

    // add specified area to print job
    // repeat once for each page to be printed
    if (my_pj.addPage([params])) {
        pagesToPrint++;
    }
    if (my_pj.addPage([params])) {
        pagesToPrint++;
    }
    if (my_pj.addPage([params])) {
        pagesToPrint++;
    }

    // send pages from the spooler to the printer, but only if one or more
    // calls to addPage() was successful. You should always check for
    // successful
    // calls to start() and addPage() before calling send().
    if (pagesToPrint > 0) {
        my_pj.send(); // print page(s)
    }
}

// clean up
delete my_pj; // delete object
```

在第一个 **PrintJob** 对象仍处于活动状态时，您不能创建另一个对象。在第一个 **PrintJob** 对象仍处于活动状态时，不能通过调用 `new PrintJob()` 创建另一个 **PrintJob** 对象，因此不会创建第二个 **PrintJob** 对象。

可用性：ActionScript 1.0；Flash Player 7

示例

请参见 `PrintJob.addPage()`。

另请参见

[addPage\(PrintJob.addPage 方法\)](#), [send\(PrintJob.send 方法\)](#), [start\(PrintJob.start 方法\)](#)

## send (PrintJob.send 方法)

```
public send() : Void
```

用在 `PrintJob.start()` 和 `PrintJob.addPage()` 方法之后，将经过后台处理的页发送到打印机。如果对 `PrintJob.start()` 和 `PrintJob.addpage()` 的相关调用失败，则对 `PrintJob.send()` 的调用将不会成功，因此在调用 `PrintJob.send()` 之前应检查对 `PrintJob.addpage()` 和 `PrintJob.start()` 的调用是否成功：

```
var my_pj:PrintJob = new PrintJob();
if (my_pj.start()) {
    if (my_pj.addPage(this)) {
        my_pj.send();
    }
}
delete my_pj;
```

可用性：ActionScript 1.0；Flash Player 7

示例

请参见 `PrintJob.addPage()` 和 `PrintJob.start()`。

另请参见

[addPage\(PrintJob.addPage 方法\)](#), [start\(PrintJob.start 方法\)](#)

# start（PrintJob.start 方法）

public start() : Boolean

显示操作系统的打印对话框并开始进行后台处理。打印对话框允许用户更改打印设置。如果 PrintJob.start() 方法返回成功，则会填充下列只读属性，以体现用户的打印设置：

属性	类型	单位	备注
PrintJob.paperHeight	Number	磅	纸张整体高度。
PrintJob.paperWidth	Number	磅	纸张整体宽度。
PrintJob.pageHeight	Number	磅	页上实际可打印区域的高度；忽略用户设置的任何边距。
PrintJob.pageWidth	Number	磅	页上实际可打印区域的宽度；忽略用户设置的任何边距。
PrintJob.orientation	Number	磅	“纵向”或“横向”。

当用户在“打印”对话框中单击“确定”之后，播放器开始在后台将打印作业处理到操作系统。您应该发布影响打印输出的任何 **ActionScript** 命令，然后可以使用 PrintJob.addPage() 命令将页发送到后台处理程序。您可以使用此方法的只读 **height**、**width** 和 **orientation** 属性进行填充以设置打印输出格式。

由于在单击“确定”后用户会马上看到诸如“正在打印第 1 页”这样的信息，所以应该尽快调用 PrintJob.addPage() 和 PrintJob.send() 命令。

如果此方法返回 false（例如，如果用户在操作系统的“打印”对话框中单击“取消”而不是“确定”），则对 PrintJob.addPage() 和 PrintJob.send() 的任何后续调用都将失败。但是，如果在测试此返回值后不发送 PrintJob.addPage() 命令，则还应该删除 **PrintJob** 对象，以确保清除该打印后台处理程序，如下面的示例所示：

```
var my_pj:PrintJob = new PrintJob();
var myResult:Boolean = my_pj.start();
    if(myResult) {
        // addPage() and send() statements here
    }
delete my_pj;
```

可用性：ActionScript 1.0；Flash Player 7

## 返回

Boolean - 一个布尔值：如果出现“打印”对话框时，用户单击“确定”，则返回 true；如果用户单击“取消”或出现错误，则返回 false。

## 示例

下面的示例说明如何使用 **orientation** 属性的值调整打印输出：

```
// create PrintJob object
var my_pj:PrintJob = new PrintJob();

// display print dialog box
if (my_pj.start()) {
    // boolean to track whether addPage succeeded, change this to a counter
    // if more than one call to addPage is possible
    var pageAdded:Boolean = false;

    // check the user's printer orientation setting
    // and add appropriate print area to print job
    if (my_pj.orientation == "portrait") {
        // Here, the printArea measurements are appropriate for an 8.5" x 11"
        // portrait page.
        pageAdded = my_pj.addPage(this,{xMin:0,xMax:600,yMin:0,yMax:800});
    }
    else {
        // my_pj.orientation is "landscape".
        // Now, the printArea measurements are appropriate for an 11" x 8.5"
        // landscape page.
        pageAdded = my_pj.addPage(this,{xMin:0,xMax:750,yMin:0,yMax:600});
    }

    // send pages from the spooler to the printer
    if (pageAdded) {
        my_pj.send();
    }
}

// clean up
delete my_pj;
```

## 另请参见

[addPage \(PrintJob.addPage 方法\)](#), [send \(PrintJob.send 方法\)](#)

# Rectangle (flash.geom.Rectangle)



```
public class Rectangle
extends Object
```

**Rectangle** 类用于创建和修改 **Rectangle** 对象。**Rectangle** 对象是按其位置（如它左上角的点 (x, y) 所示）以及宽度和高度定义的区域。如果描述某矩形的左上角位于 0,0 处，高度为 10，宽度为 20，并且右下角位于 9,19 处，则在设计这些区域时要小心，因为宽度和高度的计数从 0,0 处开始。

**Rectangle** 类的 x、y、width 和 height 属性相互独立；更改一个属性的值不会影响其它属性。但是，right 和 bottom 属性与这四个属性是整体相关的 - 如果更改 right，要更改 width；如果更改 bottom，要更改 height，等等。而且必须在建立 left 或 x 属性后，才能设置 width 或 right 属性。

**Rectangle** 对象用于支持 **BitmapData** 类滤镜。它们还用于 MovieClip.scrollRect 属性中，以支持将 MovieClip 实例裁切特定的宽度和高度并滚动特定的偏移量的功能。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[scrollRect \(MovieClip.scrollRect 属性\)](#)

## 属性摘要

修饰符	属性	说明
	bottom:Number	y 和 height 属性的和。
	bottomRight:Point	由 x 和 y 属性的值确定的 Rectangle 对象的右下角的位置。
	height:Number	矩形的高度，以像素为单位。
	left:Number	矩形左上角的 x 坐标。
	right:Number	x 和 width 属性的和。
	size:Point	Rectangle 对象的大小，该对象表示为具有 width 和 height 属性的值的 Point 对象。
	top:Number	矩形左上角的 y 坐标。
	topLeft:Point	由该点的 x 和 y 值确定的 Rectangle 对象左上角的位置。
	width:Number	矩形的宽度，以像素为单位。

修饰符	属性	说明
	x:Number	矩形左上角的 x 坐标。
	y:Number	矩形左上角的 y 坐标。

继承自 Object 类的属性

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

### 构造函数摘要

签名	说明
<code>Rectangle(x:Number, y:Number, width:Number, height:Number)</code>	创建一个新的 Rectangle 对象，其左上角由 x 和 y 参数指定。

### 方法摘要

修饰符	签名	说明
	<code>clone() : Rectangle</code>	返回一个新的 Rectangle 对象，其 x、y、width 和 height 属性的值与原始 Rectangle 对象的对应值相同。
	<code>contains(x:Number, y:Number) : Boolean</code>	确定由此 Rectangle 对象定义的矩形区域内是否包含指定的点。
	<code>containsPoint(pt:Point) : Boolean</code>	确定由此 Rectangle 对象定义的矩形区域内是否包含指定的点。
	<code>containsRectangle(rect:Rectangle) : Boolean</code>	确定此 Rectangle 对象内是否包含由 rect 参数指定的 Rectangle 对象。
	<code>equals(toCompare:Object) : Boolean</code>	确定在 toCompare 参数中指定的对象是否等于此 Rectangle 对象。
	<code>inflate(dx:Number, dy:Number) : Void</code>	按指定量增加 Rectangle 对象的大小。
	<code>inflatePoint(pt:Point) : Void</code>	增加 Rectangle 对象的大小。
	<code>intersection(toIntersect:Rectangle) : Rectangle</code>	如果在 toIntersect 参数中指定的 Rectangle 对象与此 Rectangle 对象相交，则 intersection() 方法返回交集区域作为 Rectangle 对象。



修饰符	签名	说明
	<code>intersects(toIntersect:Rectangle) : Boolean</code>	确定在 <code>toIntersect</code> 参数中指定的对象是否与此 <code>Rectangle</code> 对象相交。
	<code>isEmpty() : Boolean</code>	确定此 <code>Rectangle</code> 对象是否为空。
	<code>offset(dx:Number, dy:Number) : Void</code>	按指定量调整 <code>Rectangle</code> 对象的位置（由其左上角确定）。
	<code>offsetPoint(pt:Point) : Void</code>	将 <code>Point</code> 对象用作参数来调整 <code>Rectangle</code> 对象的位置。
	<code>setEmpty() : Void</code>	将 <code>Rectangle</code> 对象的所有属性设置为 <code>O</code> 。
	<code>toString() : String</code>	生成并返回一个字符串，该字符串列出 <code>Rectangle</code> 对象的水平位置和垂直位置以及高度和宽度。
	<code>union(toUnion:Rectangle) : Rectangle</code>	通过填充两个矩形之间的水平和垂直空间，将这两个矩形组合在一起以创建一个新的 <code>Rectangle</code> 对象。

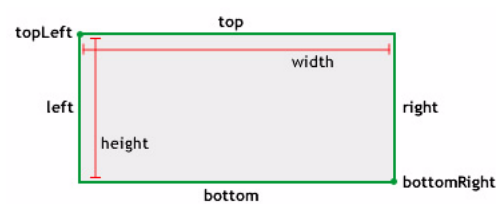
继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## bottom（`Rectangle.bottom` 属性）

`public bottom : Number`

`y` 和 `height` 属性的和。



可用性：ActionScript 1.0；Flash Player 8

## 示例

下面的示例创建一个 **Rectangle** 对象并将其 **bottom** 属性的值从 15 更改为 30。请注意，**rect.height** 的值也从 10 更改为 25。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.height); // 10
trace(rect.bottom); // 15

rect.bottom = 30;
trace(rect.height); // 25
trace(rect.bottom); // 30
```

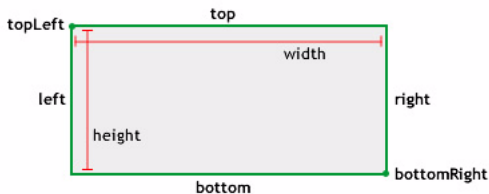
## 另请参见

[y \(Rectangle.y 属性\)](#)，[height \(Rectangle.height 属性\)](#)

## bottomRight (Rectangle.bottomRight 属性)

```
public bottomRight : Point
```

由 **x** 和 **y** 属性的值确定的 **Rectangle** 对象的右下角的位置。



可用性: ActionScript 1.0 ; Flash Player 8

## 示例

下面的示例使用 **Point** 对象的值设置 **Rectangle** 对象的 **bottomRight** 属性。请注意，**rect.width** 和 **rect.height** 发生了更改。

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.bottom); // 10
trace(rect.right); // 5
trace(rect.height); // 8
trace(rect.width); // 4

var myBottomRight:Point = new Point(16, 32);
```

```
rect.bottomRight = myBottomRight;
trace(rect.bottom); // 32
trace(rect.right); // 16
trace(rect.height); // 30
trace(rect.width); // 15
```

另请参见

[Point \(flash.geom.Point\)](#)

## clone (Rectangle.clone 方法)

```
public clone() : Rectangle
```

返回一个新的 **Rectangle** 对象，其 x、y、width 和 height 属性的值与原始 **Rectangle** 对象的对应值相同。

可用性：ActionScript 1.0；Flash Player 8

### 返回

flash.geom.Rectangle - 一个新 **Rectangle** 对象，其 x、y、width 和 height 属性的值与原始 **Rectangle** 对象的对应值相同。

### 示例

下面的示例创建三个 **Rectangle** 对象并将这三个对象进行比较。使用 **Rectangle** 构造函数创建 rect\_1。接着创建 rect\_2，方法是将它设置为等效于 rect\_1。然后通过克隆 rect\_1 创建 clonedRect。请注意，尽管 rect\_2 等效于 rect\_1，但 clonedRect 却不等效于 rect\_1，尽管它们包含相同的值。

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(1, 2, 4, 8);
var rect_2:Rectangle = rect_1;
var clonedRect:Rectangle = rect_1.clone();

trace(rect_1 == rect_2); // true
trace(rect_1 == clonedRect); // false

for(var i in rect_1) {
    trace(">> " + i + ": " + rect_1[i]);
    >> toString: [type Function]
    >> equals: [type Function]
    >> union: [type Function]
    >> intersects: [type Function]
    >> intersection: [type Function]
    >> containsRectangle: [type Function]
    >> containsPoint: [type Function]
```

```

>> contains: [type Function]
>> offsetPoint: [type Function]
>> offset: [type Function]
>> inflatePoint: [type Function]
>> inflate: [type Function]
>> size: (x=4, y=8)
>> bottomRight: (x=5, y=10)
>> topLeft: (x=1, y=2)
>> bottom: 10
>> top: 2
>> right: 5
>> left: 1
>> isEmpty: [type Function]
>> setEmpty: [type Function]
>> clone: [type Function]
>> height: 8
>> width: 4
>> y: 2
>> x: 1
}

for(var i in clonedRect) {
  trace(">> " + i + ": " + clonedRect[i]);
  >> toString: [type Function]
  >> equals: [type Function]
  >> union: [type Function]
  >> intersects: [type Function]
  >> intersection: [type Function]
  >> containsRectangle: [type Function]
  >> containsPoint: [type Function]
  >> contains: [type Function]
  >> offsetPoint: [type Function]
  >> offset: [type Function]
  >> inflatePoint: [type Function]
  >> inflate: [type Function]
  >> size: (x=4, y=8)
  >> bottomRight: (x=5, y=10)
  >> topLeft: (x=1, y=2)
  >> bottom: 10
  >> top: 2
  >> right: 5
  >> left: 1
  >> isEmpty: [type Function]
  >> setEmpty: [type Function]
  >> clone: [type Function]
  >> height: 8
  >> width: 4
  >> y: 2
  >> x: 1
}

```

为了进一步说明 `rect_1`、`rect_2` 和 `clonedRect` 之间的关系，下面的示例将修改 `rect_1` 的 `x` 属性。通过修改 `x`，说明 `clone()` 方法是根据 `rect_1` 的值而不是通过在引用中指向这些值来创建新实例的。

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(1, 2, 4, 8);
var rect_2:Rectangle = rect_1;
var clonedRect:Rectangle = rect_1.clone();

trace(rect_1.x); // 1
trace(rect_2.x); // 1
trace(clonedRect.x); // 1

rect_1.x = 10;

trace(rect_1.x); // 10
trace(rect_2.x); // 10
trace(clonedRect.x); // 1
```

另请参见

[x \(Rectangle.x 属性\)](#)，[y \(Rectangle.y 属性\)](#)，[width \(Rectangle.width 属性\)](#)，[height \(Rectangle.height 属性\)](#)

## contains (Rectangle.contains 方法)

```
public contains(x:Number, y:Number) : Boolean
```

确定由此 **Rectangle** 对象定义的矩形区域内是否包含指定的点。

可用性：ActionScript 1.0；Flash Player 8

### 参数

**x**:Number - 点的 **x** 值（水平位置）。

**y**:Number - 点的 **y** 值（垂直位置）。

### 返回

Boolean - 如果指定的点包含在 **Rectangle** 对象中，则返回 `true`；否则返回 `false`。

### 示例

下面的示例创建一个 **Rectangle** 对象并测试三个坐标对是否都处于各自的边界内。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(10, 10, 50, 50);
trace(rect.contains(59, 59)); // true
```

```
trace(rect.contains(10, 10)); // true
trace(rect.contains(60, 60)); // false
```

另请参见

[Point \(flash.geom.Point\)](#)

## containsPoint (Rectangle.containsPoint 方法)

```
public containsPoint(pt:Point) : Boolean
```

确定由此 **Rectangle** 对象定义的矩形区域内是否包含指定的点。此方法与 `Rectangle.contains()` 方法类似，只不过它采用 **Point** 对象作为参数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**pt:**flash.geom.Point - 用其 **x,y** 值表示的点。

### 返回

Boolean - 如果指定的点包含在 **Rectangle** 对象内，则返回 `true`；否则返回 `false`。

### 示例

下面的示例创建一个 **Rectangle** 对象和三个 **Point** 对象，并测试这些点是否都处于矩形的边界内。

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(10, 10, 50, 50);
trace(rect.containsPoint(new Point(10, 10))); // true
trace(rect.containsPoint(new Point(59, 59))); // true
trace(rect.containsPoint(new Point(60, 60))); // false
```

另请参见

[contains \(Rectangle.contains 方法\)](#), [Point \(flash.geom.Point\)](#)

## containsRectangle (Rectangle.containsRectangle 方法)

```
public containsRectangle(rect:Rectangle) : Boolean
```

确定此 **Rectangle** 对象内是否包含由 `rect` 参数指定的 **Rectangle** 对象。如果一个 **Rectangle** 对象完全在另一个 **Rectangle** 的边界内，我们说第二个 **Rectangle** 包含第一个 **Rectangle**。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**rect**: `flash.geom.Rectangle` - 所检查的 **Rectangle** 对象。

### 返回

**Boolean** - 如果此 **Rectangle** 对象包含您指定的 **Rectangle** 对象，则返回 `true` ; 否则返回 `false`。

### 示例

下面的示例创建四个新的 **Rectangle** 对象并确定矩形 **A** 是否包含矩形 **B**、**C** 或 **D**。

```
import flash.geom.Rectangle;

var rectA:Rectangle = new Rectangle(10, 10, 50, 50);
var rectB:Rectangle = new Rectangle(10, 10, 50, 50);
var rectC:Rectangle = new Rectangle(10, 10, 51, 51);
var rectD:Rectangle = new Rectangle(15, 15, 45, 45);

trace(rectA.containsRectangle(rectB)); // true
trace(rectA.containsRectangle(rectC)); // false
trace(rectA.containsRectangle(rectD)); // true
```

## equals (Rectangle.equals 方法)

`public equals(toCompare:Object) : Boolean`

确定在 `toCompare` 参数中指定的对象是否等于此 **Rectangle** 对象。此方法将某个对象的 `x`、`y`、`width` 和 `height` 属性与此 **Rectangle** 对象所对应的相同属性进行比较。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

`toCompare:Object` - 要与此 **Rectangle** 对象进行比较的矩形。

### 返回

**Boolean** - 如果对象具有与此 **Rectangle** 对象完全相同的 `x`、`y`、`width` 和 `height` 属性值，则返回 `true`；否则返回 `false`。

### 示例

在下面的示例中，`rect_1` 等效于 `rect_2`，但 `rect_3` 不等效于另外两个对象，因为它的 `x`、`y`、`width` 和 `height` 属性与 `rect_1` 和 `rect_2` 的对应属性不相等。

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 100);
var rect_2:Rectangle = new Rectangle(0, 0, 50, 100);
var rect_3:Rectangle = new Rectangle(10, 10, 60, 110);
```

```
trace(rect_1.equals(rect_2)); // true;
trace(rect_1.equals(rect_3)); // false;
```

虽然该方法签名只需要一个抽象对象，但只将其它 **Rectangle** 实例视为等效。

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 100);
var nonRect:Object = new Object();
nonRect.x = 0;
nonRect.y = 0;
nonRect.width = 50;
nonRect.height = 100;
trace(rect_1.equals(nonRect));
```

### 另请参见

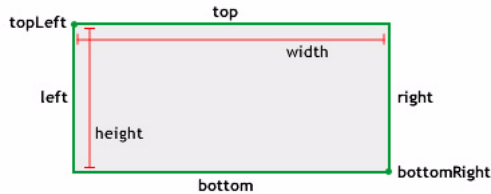
[x \(Rectangle.x 属性\)](#)，[y \(Rectangle.y 属性\)](#)，[width \(Rectangle.width 属性\)](#)，[height \(Rectangle.height 属性\)](#)



## height (Rectangle.height 属性)

public height : Number

矩形的高度，以像素为单位。更改 **Rectangle** 对象的 height 值对 x、y 和 width 属性没有影响。



可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例创建一个 **Rectangle** 对象并将其 height 属性从 10 更改为 20。请注意，rect.bottom 也发生了更改。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.height); // 10
trace(rect.bottom); // 15

rect.height = 20;
trace(rect.height); // 20
trace(rect.bottom); // 25
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#)

## inflate (Rectangle.inflate 方法)

```
public inflate(dx:Number, dy:Number) : Void
```

按指定量增加 **Rectangle** 对象的大小。保持 **Rectangle** 对象的中心点不变，使用 dx 值横向增加它的大小，使用 dy 值纵向增加它的大小。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**dx**:Number - **Rectangle** 对象横向增加的值。下列等式用于计算矩形的新宽度和位置:

```
x -= dx;  
width += 2 * dx;
```

**dy**:Number - **Rectangle** 对象纵向增加的值。下列等式用于计算矩形的新高度和位置。

```
y -= dy;  
height += 2 * dy;
```

### 示例

下面的示例创建一个 **Rectangle** 对象，并将其 width 属性的值增加  $16 * 2$  (32)，将其 height 属性的值增加  $32 * 2$  (64)

```
import flash.geom.Rectangle;  
  
var rect:Rectangle = new Rectangle(1, 2, 4, 8);  
trace(rect.toString()); // (x=1, y=2, w=4, h=8)  
  
rect.inflate(16, 32);  
trace(rect.toString()); // (x=-15, y=-30, w=36, h=72)
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#)

## inflatePoint (Rectangle.inflatePoint 方法)

```
public inflatePoint(pt:Point) : Void
```

增加 **Rectangle** 对象的大小。此方法与 `Rectangle.inflate()` 方法类似，只不过它采用 **Point** 对象作为参数。

下面的两个代码示例产生相同的结果：

```
rect1 = new flash.geom.Rectangle(0,0,2,5);
rect1.inflate(2,2)
rect1 = new flash.geom.Rectangle(0,0,2,5);
pt1 = new flash.geom.Point(2,2);
rect1.inflatePoint(pt1)
```

可用性：ActionScript 1.0 ； Flash Player 8

### 参数

**pt**:flash.geom.Point - 通过增加点的 **x** 和 **y** 坐标值来增大此矩形。

### 示例

下面的示例创建一个 **Rectangle** 对象并通过增加点的 **x**（水平）和 **y**（垂直）量来扩大它。

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(0, 0, 2, 5);
trace(rect.toString()); // (x=0, y=0, w=2, h=5)

var myPoint:Point = new Point(2, 2);
rect.inflatePoint(myPoint);
trace(rect.toString()); // (x=-2, y=-2, w=6, h=9)
```

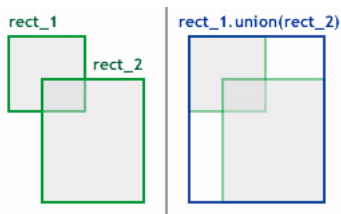
另请参见

[Point \(flash.geom.Point\)](#)

## intersection (Rectangle.intersection 方法)

`public intersection(toIntersect:Rectangle) : Rectangle`

如果在 `toIntersect` 参数中指定的 **Rectangle** 对象与此 **Rectangle** 对象相交，则 `intersection()` 方法返回交集区域作为 **Rectangle** 对象。如果矩形不相交，则此方法返回一个空的 **Rectangle** 对象，其属性设置为 0。



可用性: `ActionScript 1.0` ; `Flash Player 8`

### 参数

`toIntersect:flash.geom.Rectangle` - 要对照比较以查看其是否与此 **Rectangle** 对象相交的 **Rectangle** 对象。

### 返回

`flash.geom.Rectangle` - 一个 **Rectangle** 对象，它等于交集区域。如果该矩形不相交，则此方法返回一个空的 **Rectangle** 对象；即，其 `x`、`y`、`width` 和 `height` 属性均设置为 0 的矩形。

### 示例

下面的示例确定 `rect_1` 与 `rect_2` 相交的区域。

```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(0, 0, 50, 50);
var rect_2:Rectangle = new Rectangle(25, 25, 100, 100);
var intersectingArea:Rectangle = rect_1.intersection(rect_2);
trace(intersectingArea.toString()); // (x=25, y=25, w=25, h=25)
```

## intersects (Rectangle.intersects 方法)

`public intersects(toIntersect:Rectangle) : Boolean`

确定在 `toIntersect` 参数中指定的对象是否与此 **Rectangle** 对象相交。此方法检查指定的 **Rectangle** 对象的 `x`、`y`、`width` 和 `height` 属性，以查看它是否与此 **Rectangle** 对象相交。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

`toIntersect:flash.geom.Rectangle` - 要与此 **Rectangle** 对象比较的 **Rectangle** 对象。

### 返回

**Boolean** - 如果指定的对象与此 **Rectangle** 对象相交，则返回 `true`；否则返回 `false`。

### 示例

下面的示例确定 `rectA` 是否与 `rectB` 或 `rectC` 相交。

```
import flash.geom.Rectangle;
var rectA:Rectangle = new Rectangle(10, 10, 50, 50);
var rectB:Rectangle = new Rectangle(59, 59, 50, 50);
var rectC:Rectangle = new Rectangle(60, 60, 50, 50);
var rectAIntersectsB:Boolean = rectA.intersects(rectB);
var rectAIntersectsC:Boolean = rectA.intersects(rectC);
trace(rectAIntersectsB); // true
trace(rectAIntersectsC); // false

var firstPixel:Rectangle = new Rectangle(0, 0, 1, 1);
var adjacentPixel:Rectangle = new Rectangle(1, 1, 1, 1);
var pixelsIntersect:Boolean = firstPixel.intersects(adjacentPixel);
trace(pixelsIntersect); // false
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#), [height \(Rectangle.height 属性\)](#)

## isEmpty (Rectangle.isEmpty 方法)

public isEmpty() : Boolean

确定此 **Rectangle** 对象是否为空。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

Boolean - 如果 **Rectangle** 对象的宽度或高度小于或等于 0，则返回 true ； 否则返回 false。

### 示例

下面的示例创建一个空 **Rectangle** 对象并验证它是否为空。

```
import flash.geom.*;
var rect:Rectangle = new Rectangle(1, 2, 0, 0);
trace(rect.toString()); // (x=1, y=2, w=0, h=0)
trace(rect.isEmpty()); // true
```

下面的示例创建一个非空 **Rectangle** 并将它变为空 **Rectangle**。

```
import flash.geom.Rectangle;

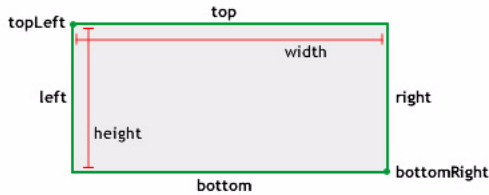
var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.isEmpty()); // false
rect.width = 0;
trace(rect.isEmpty()); // true
rect.width = 4;
trace(rect.isEmpty()); // false
rect.height = 0;
trace(rect.isEmpty()); // true
```

## left (Rectangle.left 属性)

`public left : Number`

矩形左上角的 x 坐标。更改 **Rectangle** 对象的 x 值对 y、width 和 height 属性没有影响。

left 属性等于 x 属性。



可用性: **ActionScript 1.0 ; Flash Player 8**

### 示例

下面的示例将 left 属性从 **0** 更改为 **10**。请注意，rect.x 也发生了更改。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.left); // 0
trace(rect.x); // 0

rect.left = 10;
trace(rect.left); // 10
trace(rect.x); // 10
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#), [height \(Rectangle.height 属性\)](#)

## offset (Rectangle.offset 方法)

public offset(dx:Number, dy:Number) : Void

按指定量调整 **Rectangle** 对象的位置（由其左上角确定）。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**dx**:Number - 将 **Rectangle** 对象的 **x** 值移动此数量。

**dy**:Number - 将 **Rectangle** 对象的 **y** 值移动此数量。

### 示例

下面的示例创建一个 **Rectangle** 对象并将它的 **x** 和 **y** 值分别偏移 5 和 10。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.toString()); // (x=1, y=2, w=4, h=8)

rect.offset(16, 32);
trace(rect.toString()); // (x=17, y=34, w=4, h=8)
```

## offsetPoint (Rectangle.offsetPoint 方法)

public offsetPoint(pt:Point) : Void

将 **Point** 对象用作参数来调整 **Rectangle** 对象的位置。此方法与 **Rectangle.offset()** 方法类似，只不过它采用 **Point** 对象作为参数。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 参数

**pt**:flash.geom.Point - 要用于偏移此 **Rectangle** 对象的 **Point** 对象。

### 示例

下面的示例通过使用点中的值来偏移 **Rectangle**。

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
trace(rect.toString()); // (x=1, y=2, w=4, h=8)

var myPoint:Point = new Point(16, 32);
rect.offsetPoint(myPoint);
trace(rect.toString()); // (x=17, y=34, w=4, h=8)
```



另请参见

[Point \(flash.geom.Point\)](#)

## Rectangle 构造函数

```
public Rectangle(x:Number, y:Number, width:Number, height:Number)
```

创建一个新的 **Rectangle** 对象，其左上角由 *x* 和 *y* 参数指定。如果调用此构造函数时不使用任何参数，将创建一个 *x*、*y*、*width* 和 *height* 属性均设置为 **0** 的矩形。

可用性: **ActionScript 1.0 ; Flash Player 8**

### 参数

**x**:Number - 矩形左上角的 *x* 坐标。

**y**:Number - 矩形左上角的 *y* 坐标。

**width**:Number - 矩形的宽度，以像素为单位。

**height**:Number - 矩形的高度，以像素为单位。

### 示例

下面的示例用指定的参数创建一个 **Rectangle** 对象。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 10, 50, 100);
trace(rect.toString()); // (x=5, y=10, w=50, h=100)
```

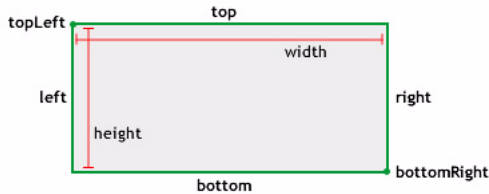
另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#),  
[height \(Rectangle.height 属性\)](#)

## right (Rectangle.right 属性)

public right : Number

x 和 width 属性的和。



可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例创建一个 **Rectangle** 对象并将其 `right` 属性从 15 更改为 30。请注意，`rect.width` 也发生了更改。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.width); // 10
trace(rect.right); // 15

rect.right = 30;
trace(rect.width); // 25
trace(rect.right); // 30
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [width \(Rectangle.width 属性\)](#)

## setEmpty (Rectangle.setEmpty 方法)

```
public setEmpty() : Void
```

将 **Rectangle** 对象的所有属性设置为 0。如果 **Rectangle** 对象的宽度或高度小于或等于 0，则它为 **空**。

此方法将 `x`、`y`、`width` 和 `height` 属性设置为 0。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建一个非空 **Rectangle** 对象并将它变为空对象。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 10, 50, 100);
trace(rect.isEmpty()); // false
rect.setEmpty();
trace(rect.isEmpty()); // true
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#),  
[height \(Rectangle.height 属性\)](#)

## size (Rectangle.size 属性)

```
public size : Point
```

**Rectangle** 对象的大小，该对象表示为具有 `width` 和 `height` 属性的值的 **Point** 对象。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建一个 **Rectangle** 对象，检索其大小 (`size`)，更改其大小 (`size`)，然后对此 **Rectangle** 对象设置新值。记住下面这一点很重要: `size` 属性所使用的 **Point** 对象使用 `x` 和 `y` 值来表示 **Rectangle** 对象的 `width` 和 `height` 属性。

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle(1, 2, 4, 8);
var size:Point = rect.size;
trace(size.x); // 4;
trace(size.y); // 8;

size.x = 16;
size.y = 32;
```

```
rect.size = size;
trace(rect.x); // 1
trace(rect.y); // 2
trace(rect.width); // 16
trace(rect.height); // 32
```

另请参见

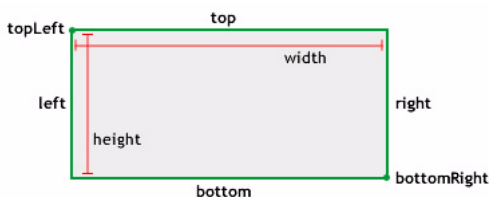
[Point \(flash.geom.Point\)](#)

## top (Rectangle.top 属性)

public top : Number

矩形左上角的 **y** 坐标。更改 **Rectangle** 对象的 **top** 属性的值对 **x**、**width** 和 **height** 属性没有影响。

**top** 属性的值等于 **y** 属性的值。



可用性: **ActionScript 1.0 ; Flash Player 8**

示例

此示例将 **top** 属性的值从 **0** 更改为 **10**。请注意，**rect.y** 也发生了更改。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.top); // 0
trace(rect.y); // 0

rect.top = 10;
trace(rect.top); // 10
trace(rect.y); // 10
```

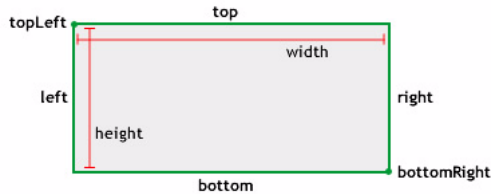
另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#), [height \(Rectangle.height 属性\)](#)

## topLeft (Rectangle.topLeft 属性)

`public topLeft : Point`

由该点的 `x` 和 `y` 值确定的 `Rectangle` 对象左上角的位置。



可用性: ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例使用 `Point` 对象中的值设置 `Rectangle` 对象的 `topLeft` 属性。请注意，`rect.x` 和 `rect.y` 发生了更改。

```
import flash.geom.Rectangle;
import flash.geom.Point;

var rect:Rectangle = new Rectangle();
trace(rect.left); // 0
trace(rect.top); // 0
trace(rect.x); // 0
trace(rect.y); // 0

var myTopLeft:Point = new Point(5, 15);
rect.topLeft = myTopLeft;
trace(rect.left); // 5
trace(rect.top); // 15
trace(rect.x); // 5
trace(rect.y); // 15
```

### 另请参见

[Point \(flash.geom.Point\)](#), [x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#)

## toString (Rectangle.toString 方法)

public toString() : String

生成并返回一个字符串，该字符串列出 **Rectangle** 对象的水平位置和垂直位置以及高度和宽度。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 返回

String - 一个字符串，它列出了 **Rectangle** 对象的下列各个属性的值: x、y、width 和 height。

### 示例

下面的示例将 rect\_1 的字符串表示形式与一些有用的调试文本连接在一起。

```
import flash.geom.Rectangle;
```

```
var rect_1:Rectangle = new Rectangle(0, 0, 50, 100);  
trace("Rectangle 1 : " + rect_1.toString()); // Rectangle 1 : (x=0, y=0,  
      w=50, h=100)
```

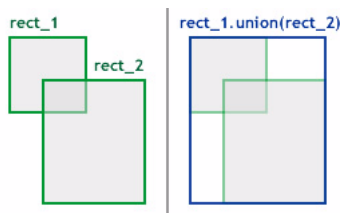
### 另请参见

[x \(Rectangle.x 属性\)](#), [y \(Rectangle.y 属性\)](#), [width \(Rectangle.width 属性\)](#), [height \(Rectangle.height 属性\)](#)

## union (Rectangle.union 方法)

public union(toUnion:Rectangle) : Rectangle

通过填充两个矩形之间的水平和垂直空间，将这两个矩形组合在一起以创建一个新的 **Rectangle** 对象。



可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

`toUnion:flash.geom.Rectangle` - 要添加到此 `Rectangle` 对象的 `Rectangle` 对象。

## 返回

`flash.geom.Rectangle` - 充当两个矩形的联合的新 `Rectangle` 对象。

## 示例

下面的示例在另外两个 `Rectangle` 对象的联合之外创建一个 `Rectangle` 对象。

例如，考虑一个属性为 `x=20`、`y=50`、`width=60` 和 `height=30` (`20, 50, 60, 30`) 的矩形和另一个属性为 (`150, 130, 50, 30`) 的矩形。这两个矩形的联合将是一个更大的矩形，其中包含两个属性为 (`20, 50, 180, 110`) 的矩形。

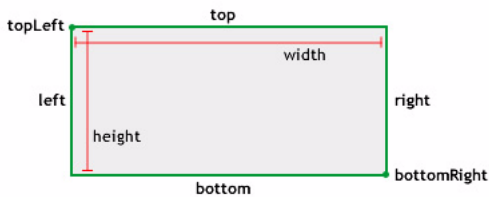
```
import flash.geom.Rectangle;

var rect_1:Rectangle = new Rectangle(20, 50, 60, 30);
var rect_2:Rectangle = new Rectangle(150, 130, 50, 30);
var combined:Rectangle = rect_1.union(rect_2);
trace(combined.toString()); // (x=20, y=50, w=180, h=110)
```

## width (`Rectangle.width` 属性)

`public width : Number`

矩形的宽度，以像素为单位。更改 `Rectangle` 对象的 `width` 属性的值对 `x`、`y` 和 `height` 属性没有影响。



可用性: `ActionScript 1.0` : `Flash Player 8`

## 示例

下面的示例创建一个 **Rectangle** 对象并将其 `width` 属性从 **10** 更改为 **20**。请注意，`rect.right` 也发生了更改。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle(5, 5, 10, 10);
trace(rect.width); // 10
trace(rect.right); // 15

rect.width = 20;
trace(rect.width); // 20
trace(rect.right); // 25
```

## 另请参见

[x \(Rectangle.x 属性\)](#)，[y \(Rectangle.y 属性\)](#)，[height \(Rectangle.height 属性\)](#)

## x (Rectangle.x 属性)

```
public x : Number
```

矩形左上角的 **x** 坐标。更改 **Rectangle** 对象的 **x** 属性的值对 `y`、`width` 和 `height` 属性没有影响。

`x` 属性等于 `left` 属性。

可用性: **ActionScript 1.0 ; Flash Player 8**

## 示例

下面的示例创建一个空 **Rectangle** 并将其 `x` 属性设置为 **10**。请注意，`rect.left` 也发生了更改。

```
import flash.geom.Rectangle;

var rect:Rectangle = new Rectangle();
trace(rect.x); // 0
trace(rect.left); // 0

rect.x = 10;
trace(rect.x); // 10
trace(rect.left); // 10
```

## 另请参见

[left \(Rectangle.left 属性\)](#)



## y (Rectangle.y 属性)

```
public y : Number
```

矩形左上角的 y 坐标。更改 **Rectangle** 对象的 y 属性的值对 x、width 和 height 属性没有影响。

y 属性等于 top 属性。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建一个空 **Rectangle** 并将其 y 属性设置为 **10**。请注意，rect.top 也发生了更改。

```
import flash.geom.Rectangle;

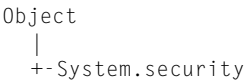
var rect:Rectangle = new Rectangle();
trace(rect.y); // 0
trace(rect.top); // 0

rect.y = 10;
trace(rect.y); // 10
trace(rect.top); // 10
```

### 另请参见

[x \(Rectangle.x 属性\)](#), [width \(Rectangle.width 属性\)](#), [height \(Rectangle.height 属性\)](#), [top \(Rectangle.top 属性\)](#)

# security (System.security)



```
public class security
extends Object
```

System.security 类包含的方法指定不同域中的 SWF 文件如何相互通讯。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 6

## 属性摘要

修饰符	属性	说明
static	sandboxType:String [ 只读 ]	指示其中正在运行执行调用的 SWF 文件的安全沙箱的类型。

继承自 Object 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 方法摘要

修饰符	签名	说明
static	allowDomain(domain1:String) : Void	允许所标识的域中的 SWF 文件和 HTML 文件访问包含 allowDomain() 调用的 SWF 文件中的对象和变量。
static	allowInsecureDomain(domain:String) : Void	允许所标识的域中的 SWF 文件和 HTML 文件访问执行调用的 SWF 文件中的对象和变量，该文件是使用 HTTPS 协议承载的。
static	loadPolicyFile(url:String) : Void	从 url 参数指定的位置加载一个跨域策略文件。

addProperty (Object.addProperty 方法), hasOwnProperty (Object.hasOwnProperty 方法), isPropertyEnumerable (Object.isPropertyEnumerable 方法), isPrototypeOf (Object.isPrototypeOf 方法), registerClass (Object.registerClass 方法), toString (Object.toString 方法), unwatch (Object.unwatch 方法), valueOf (Object.valueOf 方法), watch (Object.watch 方法)

## allowDomain (security.allowDomain 方法)

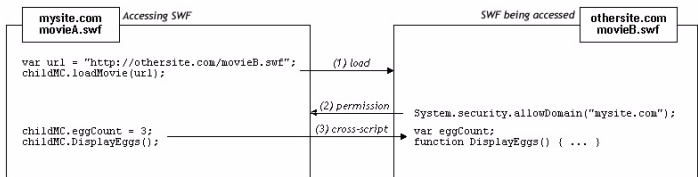
public static allowDomain(domain1:String) : Void

允许所标识的域中的 SWF 文件和 HTML 文件访问包含 allowDomain() 调用的 SWF 文件中的对象和变量。

如果从同一个域中提供两个 SWF 文件（例如，http://mysite.com/movieA.swf 和 http://mysite.com/movieB.swf），则 movieA.swf 可以检查和修改 movieB.swf 中的变量、对象、属性、方法，等等，而且 movieB.swf 可以对 movieA.swf 执行同样的操作。这被称为跨影片脚本编写 或简称跨脚本编写。

如果从不同的域提供两个 SWF 文件（例如 http://mysite.com/movieA.swf 和 http://othersite.com/movieB.swf），则在默认情况下，Flash Player 既不允许 movieA.swf 编写 movieB.swf 的脚本，也不允许 movieB.swf 编写 movieA.swf 的脚本。通过调用 System.security.allowDomain()，一个 SWF 文件可授予其它域中的 SWF 文件编写其脚本的权限。这被称为跨域脚本编写。通过调用 System.security.allowDomain("mysite.com")，movieB.swf 授予 movieA.swf 编写 movieB.swf 的脚本的权限。

在任何跨域的情况下，都会涉及两方，明确各方是很重要的。为了便于进行此讨论，我们将执行跨脚本编写的一方称为访问方（通常是执行访问的 SWF），将另一方称为被访问方（通常是被访问的 SWF）。为了继续我们的示例，当 movieA.swf 编写 movieB.swf 的脚本时，movieA.swf 是访问方，movieB.swf 是被访问方。



使用 `System.security.allowDomain()` 建立的跨域权限是不对称的。在前面的示例中，`movieA.swf` 可以编写 `movieB.swf` 的脚本，但 `movieB.swf` 无法编写 `movieA.swf` 的脚本，因为 `movieA.swf` 尚未调用 `System.security.allowDomain()` 来将编写 `movieA.swf` 的脚本的权限赋予 `othersite.com`。您可以通过让两个 SWF 文件都调用 `System.security.allowDomain()` 来设置对称权限。

除了防止 SWF 文件受到其它 SWF 文件发起的跨域脚本编写外，Flash Player 还防止 SWF 文件受到 HTML 文件发起的跨域脚本编写。可使用旧版本的 Flash 浏览器函数（例如 `SetVariable`）或通过使用 `ExternalInterface.addCallback()` 创建的回调执行 HTML 到 SWF 的脚本使用。当 HTML 到 SWF 的脚本编写跨越边界时，如果访问方恰是一个 SWF 文件，则被访问的 SWF 文件必须调用 `System.security.allowDomain()`，否则操作将失败。

如果将 IP 地址指定为 `System.security.allowDomain()` 的参数，则不允许所有源自指定 IP 地址的访问方进行访问。而是只允许通过在访问方的 URL 中显式指定该 IP 地址（而不是通过映射到该 IP 地址的域名）加载的访问方进行访问。

特定版本的差异 Flash Player 的跨域安全性规则随着版本的升级发生了演变。下表概述了这些差异。

跨脚本编写操作中所涉及的最新 SWF 版本。	是否需要 <code>allowDomain()</code> ?	是否需要 <code>allowInsecureDomain()</code> ?	哪个 SWF 必须调用 <code>allowDomain()</code> 或 <code>allowInsecureDomain()</code> ?	在 <code>allowDomain()</code> 或 <code>allowInsecureDomain()</code> 中可以指定哪些内容?
第 5 版或更早版本	否	否	暂缺	
6	是的，如果超级域不匹配		被访问的 SWF 文件，或者任何与被访问的 SWF 文件具有相同超级域的 SWF 文件	<ul style="list-style-type: none"><li>基于文本的域 (<code>mysite.com</code>)</li><li>IP 地址 (<code>192.168.1.1</code>)</li></ul>
7	是的，如果域不是完全匹配		被访问的 SWF 文件，或者任何与被访问的 SWF 文件具有完全相同域的 SWF 文件	
第 8 版或更高版本			被访问的 SWF	<ul style="list-style-type: none"><li>基于文本的域 (<code>mysite.com</code>)</li><li>IP 地址 (<code>192.168.1.1</code>)</li><li>通配符 (*)</li></ul>

控制 **Flash Player** 行为的版本是 **SWF 版本**（**SWF 文件**的发布版本），而不是 **Flash Player** 版本本身。例如，当 **Flash Player 8** 正在播放为第 7 版发布的 **SWF 文件**时，它应用与第 7 版一致的行为。这种做法可确保播放器升级不会更改已部署的 **SWF 文件**中的 `System.security.allowDomain()` 的行为。

上表中的版本列显示了跨脚本编写操作中所涉及的最新 **SWF 版本**。**Flash Player** 根据执行访问的 **SWF 文件**的版本或被访问的 **SWF 文件**的版本（以两者中的较高版本为准）来确定其行为。

下面的段落提供有关涉及 `System.security.allowDomain()` 的 **Flash Player** 安全性更改的详细信息。

第 5 版。没有跨域脚本编写限制。

第 6 版。引入了跨域脚本编写安全性。默认情况下，**Flash Player** 禁止跨域脚本编写；`System.security.allowDomain()` 可允许跨域脚本编写。为了确定两个文件是否处于同一域中，**Flash Player** 将使用每个文件的超级域（它与文件的 **URL** 中的主机名完全相同），去掉第一段，最少剩两段。例如，`www.mysite.com` 的超级域是 `mysite.com`。此示例将允许来自 `www.mysite.com` 和 `store.mysite.com` 的 **SWF 文件**相互编写脚本，而无需调用 `System.security.allowDomain()`。

第 7 版。超级域匹配更改为域完全匹配。仅在这两个文件的 **URL** 中的主机名完全相同时才允许它们相互编写脚本；否则需要调用 `System.security.allowDomain()`。默认情况下，不再允许从非 **HTTPS URL** 加载的文件编写从 **HTTPS URL** 加载的文件的脚本，即使这两个文件是从完全相同的域加载的。由于非 **HTTPS** 文件容易在下载的过程中受到修改，而经过恶意修改的非 **HTTPS** 文件能破坏 **HTTPS** 文件，此限制可防止这样的篡改，所以有助于保护 **HTTPS** 文件。引入了 `System.security.allowInsecureDomain()` 以允许被访问的 **HTTPS SWF 文件**自动禁用此限制，但 **Macromedia** 建议不要使用 `System.security.allowInsecureDomain()`。

第 8 版。有两个主要更改区域：

- 现在，只有被访问的 **SWF 文件**是调用 `System.security.allowDomain()` 的 **SWF 文件**时，调用 `System.security.allowDomain()` 才允许跨脚本编写操作。也就是说，现在，调用 `System.security.allowDomain()` 的 **SWF 文件**仅允许对其自身的访问。在以前的版本中，调用 `System.security.allowDomain()` 允许跨脚本编写操作，其中被访问的 **SWF 文件**可以是与名为 `System.security.allowDomain()` 的 **SWF 文件**在同一个域中的任何 **SWF 文件**。以前调用 `System.security.allowDomain()` 会打开执行调用的 **SWF 文件**的整个域。

- 已添加了对具有 `System.security.allowDomain("*")` 和 `System.security.allowInsecureDomain("*")` 的通配符值的支持。通配符 (\*) 值允许跨脚本编写操作，其中访问文件是从任何位置加载的任何文件。将通配符视为全局权限。通配符权限一般情况下会很有用，但是，在 **Flash Player 8** 中新的本地文件安全性规则下，要启用特定种类的操作，则尤其需要这些权限。具体来说，要使具有网络访问权限的本地 **SWF** 文件在 **Internet** 上编写某个 **SWF** 文件的脚本，被访问的 **Internet SWF** 文件必须调用 `System.security.allowDomain("*")`，从而反映本地 **SWF** 文件的来源是未知的。（如果被访问的 **Internet SWF** 文件是从 **HTTPS URL** 加载的，则 **Internet SWF** 文件必须改为调用 `System.security.allowInsecureDomain("*")`。）

有时，您可能会遇到以下情况：您从另一个域中加载一个子级 **SWF** 文件，并想让孩子级 **SWF** 文件编辑父级 **SWF** 文件的脚本，但您不知道该子级 **SWF** 文件将来自哪个最终域。例如，当您使用加载平衡重定向或第三方服务器时就可能发生这种情况。

在这种情况下，您可以使用 `MovieClip._url` 属性作为此方法的参数。例如，如果将 **SWF** 文件加载到影片剪辑 `my_mc` 中，则您可以调用

`System.security.allowDomain(my_mc._url)`。如果您这样做，请务必等待到 `my_mc` 中的 **SWF** 文件开始加载，这是因为在此时刻之前，`_url` 属性不会有最终的正确值。确定子级 **SWF** 文件开始加载的时刻的最佳方式是使用 `MovieClipLoader.onLoadStart`。

也可能出现相反的情况；即，您可能创建一个子级 **SWF** 文件，并允许其父级编辑该子级 **SWF** 文件，但不知道该父级将来自哪个域。在这种情况下，请从子级 **SWF** 调用

`System.security.allowDomain(_parent._url)`。您不必等待父级 **SWF** 文件加载；加载子级时，父级已加载完毕。

如果正在为 **Flash Player 8** 进行发布，也可以通过调用

`System.security.allowDomain("*")` 处理这些情况。不过，有时这可能是很危险的快捷方式，因为它允许来自任何域的任何其它 **SWF** 文件访问执行调用的 **SWF** 文件。通常，使用 `_url` 属性更安全。

有关更多信息，请参见以下部分：

- 《学习 **Flash** 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 6

## 参数

**domain1:String** - 一个或多个字符串，指定可以访问包含

`System.Security.allowDomain()` 调用的 SWF 文件中的对象和变量的域。可以按照以下格式指定域：

- "domain.com"
- "http://domain.com"
- "http://IPAddress"
- （仅限 **Flash Player 8**）"\*" 您可以将通配符（"\*"）传递到 `System.security.allowDomain()` 以允许所有的域（包括本地主机）可以对执行调用的 SWF 文件访问。在使用该通配符之前，请确认：对于执行调用的 SWF 文件，您的确需要提供这样广泛的访问。请参见此方法的主要说明中的讨论。

## 示例

位于 `www.macromedia.com/MovieA.swf` 的 SWF 文件包含以下行：

```
System.security.allowDomain("www.shockwave.com");  
loadMovie("http://www.shockwave.com/MovieB.swf", my_mc);
```

由于 **MovieA** 包含 `allowDomain()` 调用，所以 **MovieB** 可以访问 **MovieA** 中的对象和变量。如果 **MovieA** 不包含此调用，则 **Flash Player** 安全实施机制将阻止 **MovieB** 访问 **MovieA** 的对象和变量。

## 另请参见

`addCallback` (`ExternalInterface.addCallback` 方法)，`onLoadComplete` (`MovieClipLoader.onLoadComplete` 事件侦听器)，`_parent` (`MovieClip._parent` 属性)，`_url` (`MovieClip._url` 属性)，`allowInsecureDomain` (`security.allowInsecureDomain` 方法)

## allowInsecureDomain (security.allowInsecureDomain 方法)

```
public static allowInsecureDomain(domain:String) : Void
```

允许所标识的域中的 SWF 文件和 HTML 文件访问执行调用的 SWF 文件中的对象和变量，该文件是使用 HTTPS 协议承载的。Macromedia 不推荐使用此方法；请参见本条目后面部分的“安全性注意事项”。

此方法与 System.security.allowDomain() 的工作方式相同，但是它还允许在使用非 HTTPS 协议加载访问方并使用 HTTPS 加载被访问方的情况下执行操作。在 Flash Player 7 及更高版本中，不允许非 HTTPS 文件编写 HTTPS 文件的脚本。当被访问的 HTTPS SWF 文件使用 allowInsecureDomain() 方法时，该方法可解除此限制。

仅使用 allowInsecureDomain() 启用从非 HTTPS 文件到 HTTPS 文件的脚本编写。当从同一域提供正在访问的非 HTTPS 文件和被访问的 HTTPS 文件（例如，如果 <http://mysite.com> 上的 SWF 文件要编写 <https://mysite.com> 的 SWF 文件的脚本）时，使用它启用脚本编写。请不要使用此方法在非 HTTPS 文件之间、HTTPS 文件之间或从 HTTPS 文件到非 HTTPS 文件启用脚本编写。对于那些情况，请改用 allowDomain()。

**安全性注意事项：**Flash Player 提供 allowInsecureDomain() 以获得最优的灵活性，但 Macromedia 建议不要调用此方法。通过 HTTPS 提供文件，可以为您和您的用户提供若干保护措施，而调用 allowInsecureDomain 会削弱这些保护措施之一。下面的情形说明了在未经仔细考虑的情况下使用 allowInsecureDomain() 时，它将如何削弱安全性。

提醒

下面的信息只是一种可能的情形，意在通过一个具体的跨脚本编写示例来帮助您了解 allowInsecureDomain()。它没有涉及安全体系结构的所有问题，仅应该用于背景信息。Macromedia Developer Center 包含有关 Flash Player 和安全性的信息。有关更多信息，请参见 <http://www.macromedia.com/devnet/security/>。

设想您将要创建一个电子商务站点，它由两个组件构成：一个组件是产品目录，它不必是安全的，因为它仅包含公共信息；另一个组件是购物车 / 结帐，它必须是安全的，以保护用户的财务信息和个人信息。假定您考虑从 <http://mysite.com/catalog.swf> 提供产品目录，从 <https://mysite.com/cart.swf> 提供购物车。对您网站的一个要求是，第三方应该不能通过利用您安全体系结构中的漏洞盗取用户的信用卡号码。

请设想：“中间方”攻击者在您的服务器和您的用户之间进行干预，试图盗取您的用户在购物车申请表中输入的信用卡号码。中间方是指这样的人员：对于在您的用户和服务器之间通过公共 Internet 传输的网络数据包，他能够进行查看或更改。例如，您的某些用户所使用的不道德的 ISP，或者，在用户工作区的怀有不良企图的管理员，都可能是中间方。这种情况并不罕见。



如果 `cart.swf` 使用 HTTPS 将信用卡信息传输到服务器，则中间方攻击者无法直接从网络数据包盗取此信息，因为 HTTPS 传输已加密。但是，攻击者可以使用其它方法：在将您的 SWF 文件传递到用户时更改其中一个 SWF 文件的内容，将您的 SWF 文件替换为将用户的信息传输到攻击者所拥有的其它服务器的已更改版本。

HTTPS 协议等阻止此“修改”攻击发生作用，因为除了加密外，HTTPS 传输还是防篡改的。如果中间方攻击者更改数据包，则接收方将检测到更改并丢弃该数据包。因此在这种情况下，攻击者无法更改您的 `cart.swf`，因为它通过 HTTPS 传递的。

不过，假设您希望允许 `catalog.swf`（通过 HTTP 提供）中的按钮将项目添加到 `cart.swf`（通过 HTTPS 提供）中的购物车。若要实现此功能，`cart.swf` 应调用 `allowInsecureDomain()`，这样 `catalog.swf` 就可以编写 `cart.swf` 的脚本。此操作会出现意外结果：现在，假设的攻击者可以在用户最初下载 `catalog.swf` 时对其进行更改，因为 `catalog.swf` 是通过 HTTP 传递的且是不防篡改的。现在，攻击者更改过的 `catalog.swf` 可以编写 `cart.swf` 的脚本，因为 `cart.swf` 包含对 `allowInsecureDomain()` 的调用。已更改的 `catalog.swf` 文件可以使用 ActionScript 访问 `cart.swf` 中的变量，这样就可以读取用户的信用卡信息和其它敏感数据。然后，已更改的 `catalog.swf` 可以将此数据发送到攻击者的服务器。

显然，此实现是您所不愿看到的，但是您仍希望允许站点上两个 SWF 文件之间的跨脚本编写。要重新设计此假设的电子商务站点，以避免 `allowInsecureDomain()`，可以采用下面两种可能的方法：

- 通过 HTTPS 提供申请表中的所有 SWF 文件。这是最简单、最可靠的解决方案。在上述方案中，您将通过 HTTPS 提供 `catalog.swf` 和 `cart.swf`。当进行文件切换时（例如将 `catalog.swf` 从 HTTP 切换到 HTTPS 时），您可能发现占用的带宽和服务器 CPU 负载会稍微增大，您的用户可能发现应用程序加载时间稍长。您需要使用实际服务器进行试验，以确定这些影响的严重程度；通常它们各自不会比 10-20% 差，有时它们根本不存在。通常，可以通过使用服务器上的 HTTPS 加速硬件或软件改善效果。通过 HTTPS 提供所有协作 SWF 文件的主要好处是：您可以将 HTTPS URL 用作用户浏览器中的主 URL 而不会从浏览器生成任何混合内容的警告。此外，浏览器的挂锁图标变为可见，为您的用户提供公共的、受信任的安全性指示器。

- 使用 HTTPS 到 HTTP 的脚本编写，而不是 HTTP 到 HTTPS 的脚本编写。在所述的方案中，您可能将用户购物车的内容存储在 `catalog.swf` 中，并让 `cart.swf` 仅管理结账流程。在结账时，`cart.swf` 可以从 `catalog.swf` 中的 `ActionScript` 变量中检索购物车内容。对 HTTP 到 HTTPS 的脚本编写的限制是不对称的：虽然无法安全地允许 HTTP 发送的 `catalog.swf` 文件编写 HTTPS 发送的 `cart.swf` 文件的脚本，但 HTTPS `cart.swf` 文件可以编写 HTTP `catalog.swf` 文件的脚本。此方法比所有 HTTPS 方法都容易受到攻击；您必须谨慎，不要信任通过 HTTP 传递的任何 SWF 文件，因为 HTTP 容易被篡改。例如，当 `cart.swf` 检索说明购物车内容的 `ActionScript` 变量时，`cart.swf` 中的 `ActionScript` 代码无法信任此变量的值是所需格式的。您必须仔细地验证购物车内容不包含可能导致 `cart.swf` 执行不需要的操作的无效数据。您还必须接受以下风险：中间方可能通过更改 `catalog.swf` 为 `cart.swf` 提供有效但不准确的数据（例如，通过在用户的购物车中放入项目）。通常的结账流程通过显示购物车内容和用户最终确认的总费用，可以在某种程度上缓解此风险，但是此风险仍然存在。

多年来，Web 浏览器一直强制将 HTTPS 文件和非 HTTPS 文件分开，所描述的情形说明了进行此限制的一个很好的理由。Flash Player 为您提供了在绝对必须时避开此安全限制的功能，但在这样做之前一定要仔细考虑后果。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 7

## 参数

`domain:String` - 一个精确的域名，如 `www.myDomainName.com` 或 `store.myDomainName.com`。在 Flash Player 8 中，您可以将通配符 ("\*") 传递到 `System.security.allowInsecureDomain()`，以便允许所有域（包括本地主机）可以对执行调用的 SWF 文件访问。除非确信要允许所有域（包括本地主机）访问 HTTPS SWF 文件，否则不要使用通配符。

## 示例

在下面的示例中，在安全的域中承载数学测试，以便只有注册的学生才能进行访问。您还开发了一些 SWF 文件用于说明某些概念，您将 **这些文件** 放置在不安全的域中。您希望学生能够从包含有关概念信息的 SWF 文件访问测试。

```
// This SWF file is at https://myEducationSite.somewhere.com/mathTest.swf
// Concept files are at http://myEducationSite.somewhere.com
System.security.allowInsecureDomain("myEducationSite.somewhere.com");
```

另请参见

`allowDomain` (`security.allowDomain` 方法), `exactSettings` (`System.exactSettings` 属性)

## loadPolicyFile (security.loadPolicyFile 方法)

```
public static loadPolicyFile(url:String) : Void
```

从 `url` 参数指定的位置加载一个跨域策略文件。**Flash Player** 使用策略文件作为权限机制来允许 **Flash** 影片从它们自身以外的服务器加载数据。

**Flash Player 7.0.14.0** 只在一个位置查找策略文件：对其进行数据加载请求的服务器上的 `/crossdomain.xml`。对于 **XMLSocket** 连接尝试，**Flash Player 7.0.14.0** 在尝试 **XMLSocket** 连接的子域中的 **HTTP** 服务器的端口 **80** 上查找 `/crossdomain.xml`。**Flash Player 7.0.14.0** (和所有早期版本) 也将 **XMLSocket** 连接限制为端口 **1024** 和更高的端口。

添加 `System.security.loadPolicyFile()` 后，**Flash Player 7.0.19.0** 可以从任意位置加载策略文件，如以下示例所示：

```
System.security.loadPolicyFile("http://foo.com/sub/dir/pf.xml");
```

这使得 **Flash Player** 可以从指定 **URL** 中检索策略文件。由该位置处的策略文件授予的任何权限将适用于该服务器虚拟目录层次结构中的同一级别或更低级别中的所有内容。以下代码延续前面的示例：

```
loadVariables("http://foo.com/sub/dir/vars.txt") // allowed
loadVariables("http://foo.com/sub/dir/deep/vars2.txt") // allowed
loadVariables("http://foo.com/elsewhere/vars3.txt") // not allowed
```

您可以使用 `loadPolicyFile()` 加载任意数量的策略文件。在考虑需要策略文件的请求时，**Flash Player** 始终会等待策略文件下载完成后才会拒绝请求。如果由 `loadPolicyFile()` 指定的任何策略文件都未对请求进行授权，则 **Flash Player** 会查询原始的默认位置 `/crossdomain.xml`，这是最终的后备操作。

与特定端口号一起使用 `xmlsocket` 协议，您可以直接从 **XMLSocket** 服务器中检索策略文件，如以下示例所示：

```
System.security.loadPolicyFile("xmlsocket://foo.com:414");
```

这会导致 **Flash Player** 试图从指定的主机和端口检索策略文件。不仅可以使端口 1024 和更高端口，任何端口都可以使用。使用指定的端口建立连接后，**Flash Player** 立即传送 `<policy-file-request />`，并以 null 字节结束。**XMLSocket** 服务器可以配置为通过同一端口提供策略文件和常规 **XMLSocket** 连接，在这种情况下，服务器在传送策略文件之前，应等待 `<policy-file-request />`。服务器也可以设置成通过与标准连接相单独的端口提供策略文件，在这种情况下，服务器可以在专用策略文件端口上建立了连接之后立刻发送策略文件。服务器必须发送一个空字节来终止策略文件，并可以随后关闭该连接；如果服务器不关闭该连接，则 **Flash Player** 在收到终止 null 字节后也会这样做。

由 **XMLSocket** 服务器提供的策略文件具有与其它策略文件相同的语法，只是它还必须指定授予访问权限的端口。如果策略文件来自低于 1024 的端口，则它可以对任何端口授予访问权限；如果策略文件来自 1024 或更高的端口，则它只能对其它 1024 端口和更高的端口授予访问权限。所允许的端口在 `<allow-access-from>` 标签的 "to-ports" 属性中指定。允许使用单个端口号、端口范围和通配符。以下示例显示了一个 **XMLSocket** 策略文件：

```
<cross-domain-policy>
<allow-access-from domain="*" to-ports="507" />
<allow-access-from domain="*.foo.com" to-ports="507,516" />
<allow-access-from domain="*.bar.com" to-ports="516-523" />
<allow-access-from domain="www.foo.com" to-ports="507,516-523" />
<allow-access-from domain="www.bar.com" to-ports="*" />
</cross-domain-policy>
```

从旧的默认位置（端口 80 上的 HTTP 服务器中的 `/crossdomain.xml`）获取的策略文件将对所有 1024 端口和更高的端口隐式授予访问权限。不可能从 HTTP 服务器上的任何其它位置检索策略文件来授权 **XMLSocket** 操作，**XMLSocket** 策略文件的任何自定义位置必须位于 **XMLSocket** 服务器上。

因为连接到低于 1024 的端口是一个新增功能，因此，即使影片剪辑是连接到自己的子域，用 `loadPolicyFile()` 加载的策略文件也必须始终对这种连接进行授权。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 7,0,19,0

## 参数

`url:String` - 一个字符串；要加载的跨域策略文件所在的 URL。

## sandboxType (security.sandboxType 属性)

public static sandboxType : String [read-only]

指示其中正在运行执行调用的 SWF 文件的安全沙箱的类型。

System.security.sandboxType 具有下列值之一：

- remote: 此 SWF 文件来自于 Internet URL，并且将在基于域的沙箱规则下运行。
- localWithFile: 此 SWF 文件是本地文件，且尚未受到用户信任，它没有指定用于网络发布。此 SWF 文件可以从本地数据源读取，但不能与 Internet 进行通讯。
- localWithNetwork: 此 SWF 文件是本地文件，且尚未受到用户信任，它是使用网络标记发布的。此 SWF 可与 Internet 进行通讯，但无法从本地数据源读取。
- localTrusted: 此 SWF 文件是本地文件，并且已受到用户信任，可以使用“设置管理器”或 FlashPlayerTrust 配置文件。此 SWF 文件既可从本地数据源读取也可与 Internet 进行通讯。

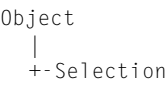
请注意，您可从任何版本的 SWF 文件检查此属性，但仅在 Flash Player 8 或更高版本中受支持。这种不常见的安排表示：您可以从诸如 Flash Player 8 中正在播放的第 7 版 SWF 文件检查此属性。这种所有版本的支持表示：如果您为低于 8 的版本进行发布，则在发布时您将无法知道在播放时是否支持此属性。因此，在第 7 版或更低版本的 SWF 文件中，您可能发现此属性有一个未定义的值；这仅在播放器版本（由 System.capabilities.version 指示）低于 8 时发生。在这种情况下，您可以根据您的 SWF 文件的 URL 是否为本地文件来确定沙箱类型。如果是本地文件，可以假定 Flash Player 会将您的 SWF 分类为 "localTrusted"（在 Flash Player 8 之前，这是对待所有本地内容的方式）。如果不是本地文件，可以假定 Flash Player 会将您的 SWF 文件分类为 "remote"。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 8

# Selection



```
public class Selection
extends Object
```

**Selection** 类可以让您设置和控制插入点所在的文本字段（即，具有焦点的字段）。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

**Selection** 类没有构造函数，这是因为一次只能有一个当前具有焦点的字段。

可用性：ActionScript 1.0 ； Flash Player 5

## 属性摘要

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 事件摘要

事件	说明
onSetFocus = function([oldfocus: Object], [newfocus:Object]) { }	当输入焦点更改时获得通知。

方法摘要

修饰符	签名	说明
static	<code>addListener(listener:Object) : Void</code>	注册一个对象，以接收键盘焦点更改通知。
static	<code>getBeginIndex() : Number</code>	返回选择范围的开始索引。
static	<code>getCaretIndex() : Number</code>	返回闪烁插入点（插入符号）位置的索引。
static	<code>getEndIndex() : Number</code>	返回当前具有焦点的选择范围的结束索引。
static	<code>getFocus() : String</code>	返回一个字符串，该字符串指定具有焦点的对象的目标路径。
static	<code>removeListener(listener:Object) : Boolean</code>	删除以前用 <code>Selection.addListener()</code> 注册的对象。
static	<code>setFocus(newFocus:Object) : Boolean</code>	使 <code>newFocus</code> 参数指定的可选择（可编辑）文本字段、按钮或影片剪辑获得焦点。
static	<code>setSelection(beginIndex:Number, endIndex:Number) : Void</code>	设置当前具有焦点的文本字段的选择范围。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## addListener (Selection.addListener 方法)

```
public static addListener(listener:Object) : Void
```

注册一个对象，以接收键盘焦点更改通知。当焦点发生更改时（例如，每当调用 `Selection.setFocus()` 时），所有用 `addListener()` 注册的侦听对象都调用它们的 `onSetFocus` 方法。可以有多个对象侦听焦点更改通知。如果指定的侦听器已经注册，则不会发生任何更改。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**listener:Object** - 一个具有 `onSetFocus` 方法的新对象。

### 示例

在下面的示例中，在运行时创建两个输入文本字段，将每个文本字段的边框设置为 `true`。此代码创建名为 `focusListener` 的新的（通用）**ActionScript** 对象。此对象为其本身定义 `onSetFocus` 属性，并为该属性分配一个函数。该函数采用两个参数：对失去焦点的文本字段的引用和对获得焦点的文本字段的引用。该函数将失去焦点的文本字段的 `border` 属性设置为 `false`，并将获得焦点的文本字段的 `border` 属性设置为 `true`：

```
this.createTextField("one_txt", 99, 10, 10, 200, 20);
this.createTextField("two_txt", 100, 10, 50, 200, 20);
one_txt.border = true;
one_txt.type = "input";
two_txt.border = true;
two_txt.type = "input";

var focusListener:Object = new Object();
focusListener.onSetFocus = function(oldFocus_txt, newFocus_txt) {
    oldFocus_txt.border = false;
    newFocus_txt.border = true;
};
Selection.addListener(focusListener);
```

测试 **SWF** 文件时，请尝试使用 **Tab** 键在两个文本字段之间移动。确保选择了“控制” > “禁用快捷键”，以便您可以使用 **Tab** 键在两个字段之间切换焦点。

另请参见

[setFocus \(Selection.setFocus 方法\)](#)



## getBeginIndex (Selection.getBeginIndex 方法)

`public static getBeginIndex() : Number`

返回选择范围的开始索引。如果不存在索引，或者当前没有具有焦点的文本字段，则该方法返回 -1。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 返回

Number - 一个整数。

### 示例

下面的示例在运行时创建一个文本字段并设置其属性。添加一个上下文菜单项，可以使用该项将当前所选文本更改为大写字符。

```
this.createTextField("output_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
output_txt.multiline = true;
output_txt.wordWrap = true;
output_txt.border = true;
output_txt.type = "input";
output_txt.text = "Enter your text here";
var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Uppercase...", doUppercase));
function doUppercase():Void {
    var startIndex:Number = Selection.getBeginIndex();
    var endIndex:Number = Selection.getEndIndex();
    var stringToUppercase:String = output_txt.text.substring(startIndex, endIndex);
    output_txt.replaceText(startIndex, endIndex, stringToUppercase.toUpperCase());
}
output_txt.menu = my_cm;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

**ActionScript** 示例文件夹的 **Strings.fla** 文件中也有一个示例。此文件夹的路径通常是：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

另请参见

[getEndIndex \(Selection.getEndIndex 方法\)](#)

## getCaretIndex (Selection.getCaretIndex 方法)

`public static getCaretIndex(): Number`

返回闪烁插入点（插入符号）位置的索引。如果没有显示闪烁插入点，则该方法返回 **-1**。选择范围索引是从零开始的（例如，第一个位置为 **0**、第二个位置为 **1**，依此类推）。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

`Number` - 一个整数。

### 示例

下面的示例在运行时创建一个文本字段并设置其属性。`getCaretIndex()` 方法用于返回插入符号的索引，并在另一个文本字段中显示其值。

```
this.createTextField("pos_txt", this.getNextHighestDepth(), 50, 20, 100, 22);
this.createTextField("content_txt", this.getNextHighestDepth(), 50, 50, 400, 300);
content_txt.border = true;
content_txt.type = "input";
content_txt.wordWrap = true;
content_txt.multiline = true;
content_txt.onChanged = getCaretPos;
```

```
var keyListener:Object = new Object();
keyListener.onKeyUp = getCaretPos;
Key.addListener(keyListener);
```

```
var mouseListener:Object = new Object();
mouseListener.onMouseUp = getCaretPos;
Mouse.addListener(mouseListener);
```

```
function getCaretPos() {
    pos_txt.text = Selection.getCaretIndex();
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

**ActionScript** 示例文件夹的 **Strings.fla** 文件中也有一个示例。此文件夹的路径通常是：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

## getEndIndex (Selection.getEndIndex 方法)

`public static getEndIndex() : Number`

返回当前具有焦点的选择范围的结束索引。如果不存在索引，或者当前没有具有焦点的选择范围，则该方法返回 -1。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数。

### 示例

此示例摘自 **ActionScript** 示例文件夹中的 **Strings.fla** 文件。

```
// define the function which converts the selected text in an instance,
// and convert the string to upper or lower case.
function convertCase(target, menuItem) {
    var beginIndex:Number = Selection.getBeginIndex();
    var endIndex:Number = Selection.getEndIndex();
    var tempString:String;
    // make sure that text is actually selected.
    if (beginIndex>-1 && endIndex>-1) {
        // set the temporary string to the text before the selected text.
        tempString = target.text.slice(0, beginIndex);
        switch (menuItem.caption) {
            case 'Uppercase...' :
                // if the user selects the "Uppercase..." context menu item,
                // convert the selected text to upper case.
                tempString += target.text.substring(beginIndex,
                    endIndex).toUpperCase();
                break;
            case 'Lowercase...' :
                tempString += target.text.substring(beginIndex,
                    endIndex).toLowerCase();
                break;
        }
        // append the text after the selected text to the temporary string.
        tempString += target.text.slice(endIndex);
        // set the text in the target text field to the contents of the temporary
        string.
        target.text = tempString;
    }
}
```

若要查看整个脚本，请参见 `Strings.fla` 文件。`ActionScript` 示例文件夹的典型路径为：

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript

另请参见

[getBeginIndex](#) ([Selection.getBeginIndex](#) 方法)

## getFocus (Selection.getFocus 方法)

`public static getFocus() : String`

返回一个字符串，该字符串指定具有焦点的对象的目标路径。

- 如果 `TextField` 对象具有焦点，并且该对象具有一个实例名称，则此方法返回 `TextField` 对象的目标路径。否则，它返回 `TextField` 的变量名。
- 如果 `Button` 对象或按钮影片剪辑具有焦点，则此方法返回 `Button` 对象或按钮影片剪辑的目标路径。
- 如果 `TextField` 对象、`Button` 对象、`Component` 实例和按钮影片剪辑均没有焦点，则此方法返回 `null`。

可用性: `ActionScript 1.0` ; `Flash Player 5`

### 返回

`String` - 一个字符串或 `null`。

### 示例

下面的示例在 `TextArea` 组件实例中显示当前获得焦点的选择的目标路径。将几个组件实例或按钮、文本字段和影片剪辑实例添加到舞台。将几个组件实例或按钮、文本字段和影片剪辑实例添加到您的 `SWF` 文件。然后，将以下 `ActionScript` 添加到您的 `AS` 或 `FLA` 文件。

```
var focus_ta:mx.controls.TextArea;
my_mc.onRelease = function() {};
my_btn.onRelease = function() {};

var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    if (Key.isDown(Key.SPACE)) {
        focus_ta.text = Selection.getFocus()+newline+focus_ta.text;
    }
};
Key.addListener(keyListener);
```

测试 SWF 文件，并使用 **Tab** 键在舞台上的实例之间移动。确保在测试环境中选择了“控制” > “禁用快捷键”。

另请参见

[onSetFocus \(Selection.onSetFocus 事件侦听器\)](#), [setFocus \(Selection.setFocus 方法\)](#)

## onSetFocus (Selection.onSetFocus 事件侦听器)

```
onSetFocus = function([oldfocus:Object], [newfocus:Object]) {}
```

当输入焦点更改时获得通知。若要使用此侦听器，您必须创建一个侦听器对象。然后，您可以为此侦听器定义函数，并使用 **Selection.addListener()** 向 **Selection** 对象注册该侦听器，如以下代码所示：

```
var someListener:Object = new Object();
someListener.onSetFocus = function () {
    // statements
}
Selection.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

**可用性：** **ActionScript 1.0** ; **Flash Player 6**

### 参数

**oldfocus:Object** [ 可选 ] - 失去焦点的对象。

**newfocus:Object** [ 可选 ] - 接收焦点的对象。

### 示例

下面的示例演示如何确定 SWF 文件中的输入焦点何时在几个动态创建的文本字段之间更改。将以下 **ActionScript** 输入到 **FLA** 或 **AS** 文件中，然后测试文档：

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
    if (this[i] instanceof TextField) {
        this[i].border = true;
        this[i].type = "input";
    }
}

this.createTextField("status_txt", this.getNextHighestDepth(), 200, 10, 300,
100);
status_txt.html = true;
```

```

status_txt.multiline = true;

var someListener:Object = new Object();
someListener.onSetFocus = function(oldFocus, newFocus) {
    status_txt.htmlText = "<b>setFocus triggered</b>";
    status_txt.htmlText += "<textformat tabStops='[20,80]'">";
    status_txt.htmlText += "&nbsp;\toldFocus:\t"+oldFocus;
    status_txt.htmlText += "&nbsp;\tnewFocus:\t"+newFocus;
    status_txt.htmlText += "&nbsp;\tgetFocus:\t"+Selection.getFocus();
    status_txt.htmlText += "</textformat>";
};
Selection.addListener(someListener);

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[addListener](#) ([Selection.addListener](#) 方法), [setFocus](#) ([Selection.setFocus](#) 方法)

## removeListener (Selection.removeListener 方法)

```
public static removeListener(listener:Object) : Boolean
```

删除以前用 `Selection.addListener()` 注册的对象。

**可用性:** **ActionScript 1.0** ; **Flash Player 6**

### 参数

**listener:Object** - 不再接收焦点通知的对象。

### 返回

**Boolean** - 如果删除 `listener` 成功，则该方法的返回值为 `true`。如果未能成功删除 `listener`（例如，如果 `listener` 不在该 **Selection** 对象的侦听器列表上），则该方法的返回值为 `false`。

### 示例

下面的 **ActionScript** 动态创建几个文本字段实例。选择文本字段时，信息将显示在“输出”面板中。单击 `remove_btn` 实例时，侦听器将被删除，信息不再显示在“输出”面板中。

```

this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {

```

```

        if (this[i] instanceof TextField) {
            this[i].border = true;
            this[i].type = "input";
        }
    }

    var selectionListener:Object = new Object();
    selectionListener.onSetFocus = function(oldFocus, newFocus) {
        trace("Focus shifted from "+oldFocus+" to "+newFocus);
    };
    Selection.addListener(selectionListener);

    remove_btn.onRelease = function() {
        trace("removeListener invoked");
        Selection.removeListener(selectionListener);
    };

```

另请参见

[addListener \(Selection.addListener 方法\)](#)

## setFocus (Selection.setFocus 方法)

```
public static setFocus(newFocus:Object) : Boolean
```

使 newFocus 参数指定的可选择（可编辑）文本字段、按钮或影片剪辑获得焦点。如果传递 null 或 undefined，则删除当前焦点。

**可用性:** **ActionScript 1.0 ; Flash Player 5**

### 参数

**newFocus:Object** - 一个对象（例如，某个按钮、影片剪辑或文本字段实例），或者是一个字符串，该字符串指定某个按钮、影片剪辑或文本字段实例的路径。如果您传递指定路径的字符串，请将该路径置于引号内 (" ")。可以使用点或斜杠记号指定路径。如果要使用 **ActionScript 2.0**，则必须使用点符号表示。可以使用相对路径或绝对路径。

### 返回

**Boolean** - 一个布尔值；如果尝试获得焦点成功，则为 true；如果失败，则为 false。

## 示例

在下面的示例中，当文本字段在浏览器窗口中运行时，它将在 username\_txt 文本字段上获得焦点。如果用户不填写必需文本字段（username\_txt 和 password\_txt）之一，则光标将自动在缺少数据的文本字段中获得焦点。例如，如果用户未在 username\_txt 文本字段中键入任何内容，则在单击“提交”按钮时，将出现一条错误消息，而且光标在 username\_txt 文本字段中获得焦点。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 100, 70, 100, 22);
this.createTextField("username_txt", this.getNextHighestDepth(), 100, 100, 100, 22);
this.createTextField("password_txt", this.getNextHighestDepth(), 100, 130, 100, 22);
this.createEmptyMovieClip("submit_mc", this.getNextHighestDepth());
submit_mc.createTextField("submit_txt", this.getNextHighestDepth(), 100, 160, 100, 22);
submit_mc.submit_txt.autoSize = "center";
submit_mc.submit_txt.text = "Submit";
submit_mc.submit_txt.border = true;
submit_mc.onRelease = checkForm;
username_txt.border = true;
password_txt.border = true;
username_txt.type = "input";
password_txt.type = "input";
password_txt.password = true;
Selection.setFocus("username_txt");
//
function checkForm():Boolean {
    if (username_txt.text.length == 0) {
        status_txt.text = "fill in username";
        Selection.setFocus("username_txt");
        return false;
    }
    if (password_txt.text.length == 0) {
        status_txt.text = "fill in password";
        Selection.setFocus("password_txt");
        return false;
    }
    status_txt.text = "success!";
    Selection.setFocus(null);
    return true;
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。



另请参见

[getFocus \(Selection.getFocus 方法\)](#)

## setSelection (Selection.setSelection 方法)

```
public static setSelection(beginIndex:Number, endIndex:Number) : Void
```

设置当前具有焦点的文本字段的选择范围。新的选择范围将从 `beginIndex` 参数中指定的索引处开始，到 `endIndex` 参数中指定的索引处结束。选择范围索引是从零开始的（例如，第一个位置为 **0**、第二个位置为 **1**，依此类推）。如果当前没有具有焦点的文本字段，则此方法无效。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**beginIndex**:Number - 选择范围的起始索引。

**endIndex**:Number - 选择范围的结束索引。

### 示例

在下面的 **ActionScript** 中，在运行时创建一个文本字段并向其添加字符串。然后，使文本字段具有焦点，并在具有焦点的文本字段中选择字符范围。

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.text = "this is my text";
this.onEnterFrame = function () {
    Selection.setFocus("myText_txt");
    Selection.setSelection(0, 3);
    delete this.onEnterFrame;
}
```

下面的示例说明 `endIndex` 参数如何将端点值排除在内。为了选择第一个字符，您必须使用值为 **1**（而不是 **0**）的 `endIndex`。如果将 `endIndex` 参数更改为 **0**，则不会选择任何内容。

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.text = "this is my text";
this.onEnterFrame = function () {
    Selection.setFocus("myText_txt");
    Selection.setSelection(0, 1);
    delete this.onEnterFrame;
}
```

# SharedObject



```
public dynamic class SharedObject
extends Object
```

**SharedObject** 类用于在用户计算机上读取和存储有限的数据量。共享对象提供永久贮存在用户计算机上的对象之间的实时数据共享。本地共享对象与浏览器 **cookie** 类似。

下面是共享对象的三种可能用法：

- 存储用户高分的游戏。游戏可以为用户提供个性化数据（如用户名和高分），而无须在服务器上专门存储。
- 可以联机或脱机工作的电话簿应用程序。作为放映文件应用程序提供的电话簿可以包含本地数据缓存，其中存有用户输入的姓名和电话号码的列表。当 **Internet** 连接可用时，应用程序将从服务器检索最新信息。当没有连接可用时，应用程序将使用在共享对象中保存的最新数据。
- 复杂网站的用户首选项或跟踪数据，如用户阅读了新闻网站上哪些文章的记录。跟踪此信息将允许您以不同方式分别显示已经阅读的文章和新的未读文章。在用户计算机上存储此信息可减小服务器负载。


本地共享对象维护本地永久性。例如，您可以调用 `SharedObject.getLocal()` 以创建一个包含游戏中高分的共享对象。因为共享对象在本地永久贮存，所以当关闭游戏时，**Flash** 会将其数据属性保存在用户的计算机上。下次打开游戏时，将显示以前会话中的高分。或者，您可以在关闭游戏之前，将共享对象的属性设置为 `null`。**SWF** 文件下次运行时，游戏将打开，但不显示以前的高分。

若要创建本地共享对象，请使用以下语法：

```
var so:SharedObject = SharedObject.getLocal("userHighScore");
so.data.highScore = new Number();
so.flush();
```

在示例中，显式刷新共享对象或将其写入磁盘。在应用程序关闭时，自动刷新共享对象；但是，在此处显示它是为了说明将数据写入磁盘的步骤。

**本地磁盘空间注意事项：**本地共享对象将非常有用，但它们也有一些限制，这些限制很重要，您在设计应用程序时要予以考虑。有时可能不允许 SWF 文件写入本地共享对象，有时可能在您不知情的情况下删除在本地共享对象中存储的数据。Flash Player 用户可以管理对单个域或所有域可用的磁盘空间。当用户降低可用磁盘空间量时，一些本地共享对象可能会被删除。Flash Player 用户还具有隐私控件，它们可防止第三方域（当前浏览器地址栏中域之外的域）读取或写入本地共享对象。

本地内容始终可以将第三方共享对象写入磁盘，即使不允许由第三方域将共享对象写入磁盘。

Macromedia 建议您检查与可用磁盘空间量和用户隐私控件相关的故障。在调用 `getLocal()` 和 `flush()` 时执行这些检查：

- `SharedObject.getLocal()` - 当用户已禁用第三方共享对象，并且您的 SWF 文件的域与浏览器地址栏中的域不匹配时，此方法返回 `null`。
- `SharedObject.flush()` - 当用户已对您的域或所有域禁用共享对象时，此方法返回 `false`。当需要额外存储空间，且用户必须以交互方式决定是否允许增加时，它返回 `"pending"`。

如果您的 SWF 文件尝试创建或修改本地共享对象，请确保 SWF 文件的宽度至少为 215 像素，高度至少为 138 像素，这是用于显示提示用户增加其本地共享对象存储限制的对话框的最小尺寸。如果您的 SWF 文件小于这些尺寸，且需要增加存储限制，则 `SharedObject.flush()` 失败，返回 `"pending"`，但此后会调用 `SharedObject.onStatus` 处理函数，结果为 `"SharedObject.Flush.Failed"`。

可用性：ActionScript 1.0；Flash Player 6

另请参见

[getLocal \(SharedObject.getLocal 方法\)](#)，[flush \(SharedObject.flush 方法\)](#)，[onStatus \(SharedObject.onStatus 处理函数\)](#)

属性摘要

修饰符	属性	说明
	<code>data:Object</code>	分配给对象的 <code>data</code> 属性的属性集合；可以共享和 / 或存储这些属性。

继承自 Object 类的属性

---

[constructor \(Object.constructor 属性\)](#)，[\\_\\_proto\\_\\_ \(Object.\\_\\_proto\\_\\_ 属性\)](#)，[prototype \(Object.prototype 属性\)](#)，[\\_\\_resolve \(Object.\\_\\_resolve 属性\)](#)

---

事件摘要

事件	说明
onStatus = function(infoOb ject:Object) {}	每次为共享对象公布错误、警告或信息性通知时调用。

方法摘要

修饰符	签名	说明
	clear() : Void	清除共享对象中的所有数据并从磁盘中删除共享对象。
	flush([minDiskSpace: Number]) : Object	将本地永久共享对象立即写入本地文件。
static	getLocal(name:String, [localPath:String], [secure:Boolean]) : SharedObject	返回对本地永久共享对象的引用，该对象只可用于当前客户端。
	getSize() : Number	获取共享对象的当前大小（以字节为单位）。

继承自 `Object` 类的方法

<a href="#">addProperty</a> ( <a href="#">Object.addProperty</a> 方法), <a href="#">hasOwnProperty</a> ( <a href="#">Object.hasOwnProperty</a> 方法), <a href="#">isPrototypeOf</a> ( <a href="#">Object.isPrototypeOf</a> 方法), <a href="#">isPropertyEnumerable</a> ( <a href="#">Object.isPropertyEnumerable</a> 方法), <a href="#">isPrototypeOf</a> ( <a href="#">Object.isPrototypeOf</a> 方法), <a href="#">registerClass</a> ( <a href="#">Object.registerClass</a> 方法), <a href="#">toString</a> ( <a href="#">Object.toString</a> 方法), <a href="#">unwatch</a> ( <a href="#">Object.unwatch</a> 方法), <a href="#">valueOf</a> ( <a href="#">Object.valueOf</a> 方法), <a href="#">watch</a> ( <a href="#">Object.watch</a> 方法)
--

clear（`SharedObject.clear` 方法）

```
public clear() : Void
```

清除共享对象中的所有数据并从磁盘中删除共享对象。对 `my_so` 的引用仍然处于活动状态，而且 `my_so` 现在是空的。

可用性：ActionScript 1.0；Flash Player 7

示例

下面的示例在共享对象中设置数据，然后从共享对象中清空所有数据。

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");  
my_so.data.name = "Hector";  
trace("before my_so.clear():");  
for (var prop in my_so.data) {  
    trace("\t"+prop);  
}
```

```

}
trace("");
my_so.clear();
trace("after my_so.clear()");
for (var prop in my_so.data) {
    trace("\t"+prop);
}

```

此 **ActionScript** 在“输出”面板中显示以下消息：

```

before my_so.clear():
    name

```

```

after my_so.clear():

```

## data（SharedObject.data 属性）

```

public data : Object

```

分配给对象的 **data** 属性的属性集合；可以共享和 / 或存储这些属性。每个属性都可以是任何基本 **ActionScript** 或 **JavaScript** 类型的对象（数组、数字、布尔值，等等）。例如，下面几行将值分配到共享对象的不同方面：

```

var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserUsername:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserUsername;

for (var prop in my_so.data) {
    trace(prop+": "+my_so.data[prop]);
}

```

如果对象是永久的，则会保存共享对象的 **data** 属性的所有属性，共享对象包含以下信息：

```

userName: Ramona
adminPrivileges: true
itemNumbers: 101,346,483

```



不要将值直接分配给共享对象（例如在 `so.data = someValue` 中）的 **data** 属性，Flash 将忽略这些分配值。

若要删除本地共享对象的属性，请使用诸如 `delete so.data.attributeName` 这样的代码；将本地共享对象的属性设置为 `null` 或 `undefined` 并不会删除该属性。

若要为共享对象创建私有值（该对象正在使用时只有客户端实例才可以使用该值，并且该值在对象关闭时不与该对象存储在一起），请创建名称不是 `data` 的属性来存储它们，如以下示例所示：

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.favoriteColor = "blue";
my_so.favoriteNightClub = "The Bluenote Tavern";
my_so.favoriteSong = "My World is Blue";

for (var prop in my_so) {
    trace(prop+": "+my_so[prop]);
}
```

共享对象包含以下数据：

```
favoriteSong: My World is Blue
favoriteNightClub: The Bluenote Tavern
favoriteColor: blue
data: [object Object]
```

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例将 `TextInput` 组件实例中的文本保存到名为 `my_so` 的共享对象（有关完整的示例，请参见 `SharedObject.getLocal()`）：

```
// Create a listener object and function for the <enter> event.
var textListener:Object = new Object();
textListener.enter = function(eventObj:Object) {
    my_so.data.myTextSaved = eventObj.target.text;
    my_so.flush();
};
```

另请参见

## flush (SharedObject.flush 方法)

```
public flush([minDiskSpace:Number]) : Object
```

将本地永久共享对象立即写入本地文件。如果您不使用此方法，则 **Flash** 会在共享对象会话结束时（也就是说，在 **SWF** 文件关闭时、在共享对象由于不再具有引用而被作为垃圾回收时、或者在您调用 `SharedObject.clear()` 时），将共享对象写入文件。

如果此方法返回 "pending"，则 **Flash Player** 将显示一个对话框，要求用户增加磁盘空间量以供此域中的对象使用。若要允许将来保存共享对象时其空间能够增长，从而避免返回值 "pending"，请为 `minimumDiskSpace` 传递一个值。当 **Flash** 尝试写入文件时，它查找传递给 `minimumDiskSpace` 的字节数，而不是查找足够的空间以保存当前大小的共享对象。

例如，如果预期共享对象增长到最大为 500 个字节，则即使它开始时要小得多，也为 `minimumDiskSpace` 传递 500。如果 **Flash** 要求用户为该共享对象分配磁盘空间，它将要求 500 个字节。在用户分配了请求的空间量之后，当以后尝试对齐该对象时（只要其大小不超过 500 个字节），**Flash** 将无需要求更多的空间。

在用户响应对话框后，再次调用此方法并返回 `true` 或 `false`；还使用

`SharedObject.Flush.Success` 或 `SharedObject.Flush.Failed` 的 `code` 属性调用 `SharedObject.onStatus`。

有关更多信息，请参见 `SharedObject` 类概述中的“本地磁盘空间注意事项”。

可用性：ActionScript 1.0；Flash Player 6

### 参数

`minDiskSpace:Number` [ 可选 ] - 一个整数，指定必须为此对象分配的字节数。默认值是 0。

### 返回

`Object` - 一个布尔值：`true` 或 `false`；或一个值为 "pending" 的字符串，如以下列表中所述：

- 如果用户允许对此域中的对象进行本地信息存储，并且所分配的空间量足够存储该对象，则此方法返回 `true`。（如果已为 `minimumDiskSpace` 传递了一个值，则所分配的空间量必须至少等于该值才能够返回 `true`）。
- 如果用户允许对此域中的对象进行本地信息存储，但所分配的空间量不足以存储该对象，则此方法返回 "pending"。
- 如果用户已永久拒绝对此域中的对象进行本地信息存储，或者如果 **Flash** 由于任何原因无法保存该对象，则此方法返回 `false`。



本地内容始终可以将第三方域（当前浏览器地址栏中域之外的域）中的共享对象写入磁盘，即使禁止将第三方共享对象写入磁盘。

## 示例

下面的函数获取共享对象 `my_so`，并通过用户提供的设置填充可写属性。最后，调用 `flush()` 以保存设置并分配最少 **1000** 字节的磁盘空间。

```
this.syncSettingsCore = function(soName:String, override:Boolean,
    settings:Object) {
    var my_so:SharedObject = SharedObject.getLocal(soName, "http://
    www.mydomain.com/app/sys");
    // settings list index
    var i;
    // For each specified value in settings:
    // If override is true, set the persistent setting to the provided value.
    // If override is false, fetch the persistent setting, unless there
    // isn't one, in which case, set it to the provided value.
    for (i in settings) {
        if (override || (my_so.data[i] == null)) {
            my_so.data[i] = settings[i];
        } else {
            settings[i] = my_so.data[i];
        }
    }
    my_so.flush(1000);
};
```

另请参见

[clear \(SharedObject.clear 方法\)](#)，[onStatus \(SharedObject.onStatus 处理函数\)](#)

## getLocal (SharedObject.getLocal 方法)

```
public static getLocal(name:String, [localPath:String], [secure:Boolean]) :
    SharedObject
```

返回对本地永久共享对象的引用，该对象只可用于当前客户端。如果尚不存在共享对象，则此方法将创建一个共享对象。此方法是 **SharedObject** 类的静态方法。若要将该对象分配给变量，请使用类似下面的语法：

```
var so:SharedObject = SharedObject.getLocal("savedData")
```



如果用户已选择了从不允许此域的本地存储，则即使指定了 `localPath` 的值，对象也不会在本地图保存。本地内容不遵循此规则。本地内容始终可以将第三方域（当前浏览器地址栏中域之外的域）中的共享对象写入磁盘，即使禁止将第三方共享对象写入磁盘。

为了避免名称冲突，**Flash** 会考虑创建共享对象的 **SWF** 文件的位置。例如，如果位于 **www.myCompany.com/apps/stockwatcher.swf** 的 **SWF** 文件创建了一个名为 `portfolio` 的共享对象，则该共享对象不会与位于 **www.yourCompany.com/photoshoot.swf** 的 **SWF** 文件所创建的另一个名为 `portfolio` 的对象冲突，这是因为这两个 **SWF** 文件源于不同的目录。



虽然 `localPath` 参数是可选的，但是您应该考虑该参数的用途，特别是其它 SWF 文件需要访问共享对象时。如果共享对象中的数据特定于一个不会移动到其它位置的 SWF 文件，则使用默认值将会解决问题。如果其它 SWF 文件需要访问共享对象，或如果创建共享对象的 SWF 文件以后将发生移动，则此参数的值对是否任何 SWF 文件都能够访问共享对象会产生影响。例如，如果在将 `localPath` 设置为 SWF 文件完整路径的默认值的情况下创建一个共享对象，则其它任何 SWF 文件都将无法访问该共享对象。如果您以后将原始的 SWF 文件移动到另一个位置，则即使该 SWF 文件也将无法访问已经存储在共享对象中的数据。

通过使用 `localPath` 参数，可以降低不经意间限制对共享对象的访问的可能性。允许级别最高的选项是将 `localPath` 参数设置为 `"/"`，这样做可使域中的所有 SWF 文件都可访问该共享对象，但会增加与域中其它共享对象发生名称冲突的可能性。限制级别较高的选项对可追加 `localPath` 参数（使用 SWF 文件的完整路径中包含的文件夹名称）的范围可用；例如，通过位于 [www.myCompany.com/apps/stockwatcher.swf](http://www.myCompany.com/apps/stockwatcher.swf) 的 SWF 文件创建的 `portfolio` 共享对象的 `localPath` 参数选项为：`"/"`、`"/apps"` 和 `"/apps/stockwatcher.swf"`。您需要确定哪个选项将为您的应用程序提供最佳的灵活性。

使用此方法时，请考虑 Flash Player 安全模型：

- 无法跨沙箱边界访问共享对象。
- 用户可通过 Flash Player 的“设置”对话框或“设置管理器”限制共享对象访问。默认情况下，共享对象最大可以创建为每域 100K 的数据。管理用户和用户还可限制写入文件系统的能力。

如果您将要回放的 SWF 文件内容发布为本地文件（本地安装的 SWF 文件或放映文件 [EXE]），而且，您需要从多个本地 SWF 文件访问某个特定的共享对象，请注意：对于本地文件，可能会使用两个不同位置来存储共享对象。所用的域取决于为创建该共享对象的本地文件授予的安全权限。本地文件可以具有三种不同级别的权限：1) 仅访问本地文件系统；2) 仅访问网络；或 3) 同时访问网络和本地文件系统。可以访问本地文件系统（1 或 3）的本地文件将其共享对象存储在一个位置。无法访问本地文件系统（2）的本地文件将其共享对象存储在另一位置中。有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 6

## 参数

**name:String** - 表示对象名称的字符串。该名称可以包含正斜杠 (/)；例如，work/addresses 是合法名称。共享对象名称中不允许使用空格，也不允许使用以下字符：  
~ % & \ ; : " ' , < > ? #

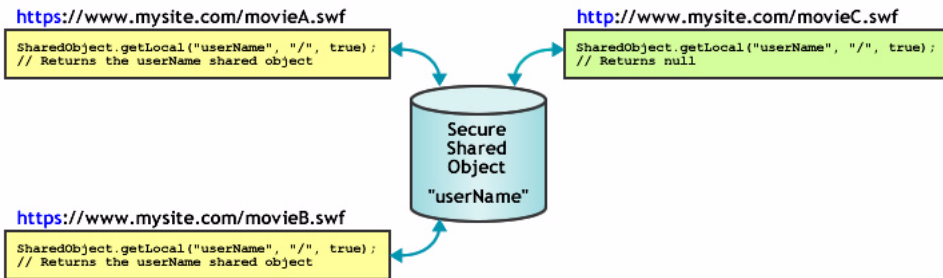
**localPath:String** [ 可选 ] - 一个字符串，指定创建共享对象的 SWF 文件的完整路径或部分路径，并确定共享对象的本地存储位置。默认值是完整路径。

**secure:Boolean** [ 可选 ] - (仅适用于 Flash Player 8) 确定对此共享对象的访问是否只限于通过 HTTPS 连接传递的 SWF 文件。假定您的 SWF 文件是通过 HTTPS 传递的：

- 如果此参数设置为 true，则 Flash Player 将创建一个新的安全共享对象或获取一个对现有安全共享对象的引用。只可由通过 HTTPS 传递的 SWF 文件对此安全共享对象进行读取或写入，而该 HTTPS 调用 SharedObject.getLocal() 并将 secure 参数设置为 true。
- 如果此参数设置为 false，则 Flash Player 将创建一个新的共享对象或获取一个对现有共享对象的引用。仅可由通过非 HTTPS 连接传递的 SWF 文件对此共享对象进行读取或写入。

如果您的 SWF 文件是通过非 HTTPS 连接传递的，并且您尝试将此参数设置为 true，将无法创建新的共享对象（或访问以前创建的安全共享对象），并且返回 null。无论此参数为何值，创建的共享对象的数量都接近域所允许的磁盘空间的总量。默认值为 false。

下面的图示说明了 secure 参数的用途：



## 返回

SharedObject - 一个对共享对象的引用，该共享对象永久贮存在本地并且只可用于当前客户端。如果 Flash Player 无法创建或找到共享对象（例如，指定 localPath 但不存在此目录时，或者错误地使用 secure 参数时），则此方法返回 null。

如果禁止第三方 Flash 内容创建或存储永久共享对象，则此方法将失败并返回 null。用户可以在设置管理器的“全局存储设置”面板上禁止第三方永久共享对象。

## 示例

下面的示例创建一个共享对象，该对象将键入的文本存储到 TextInput 组件实例中。生成的 SWF 文件在开始播放时从共享对象加载已保存的文本。用户每次按 **Enter** 键时，文本字段中的文本都将被写入共享对象。若要使用此示例，请将 TextInput 组件拖动到舞台上，并将实例命名为 myText\_ti。将以下代码复制到主时间轴中（在舞台的空白区域中单击或按 **Esc** 键以从组件上移除焦点）：

```
// Create the shared object and set localpath to server root.
var my_so:SharedObject = SharedObject.getLocal("savedText", "/");
// Load saved text from the shared object into the myText_ti TextInput
// component.
myText_ti.text = my_so.data.myTextSaved;
// Assign an empty string to myText_ti if the shared object is undefined
// to prevent the text input box from displaying "undefined" when
// this script is first run.
if (myText_ti.text == undefined) {
    myText_ti.text = "";
}
// Create a listener object and function for <enter> event
var textListener:Object = new Object();
textListener.enter = function(eventObj:Object) {
    my_so.data.myTextSaved = eventObj.target.text;
    my_so.flush();
};
// Register the listener with the TextInput component instance
myText_ti.addEventListener("enter", textListener);
```

下面的示例将用户输入的最后一帧保存到本地共享对象 kookie：

```
// Get the kookie
var my_so:SharedObject = SharedObject.getLocal("kookie");

// Get the user of the kookie and go to the frame number saved for this user.
if (my_so.data.user != undefined) {
    this.user = my_so.data.user;
    this.gotoAndStop(my_so.data.frame);
}
```

下面的代码块放置在每个 SWF 文件帧上：

```
// On each frame, call the rememberme function to save the frame number.
function rememberme() {
    my_so.data.frame=this._currentframe;
    my_so.data.user="John";
}
```

## getSize (SharedObject.getSize 方法)

`public getSize() : Number`

获取共享对象的当前大小（以字节为单位）。

**Flash** 通过逐句调试共享对象的每个数据属性计算该对象的大小；对象具有的数据属性越多，评估其大小所花的时间就越长。评估对象大小可能要耗费相当长的处理时间，因此，除非您对它具有特定的需要，否则您可能希望避免使用此方法。

**可用性:** **ActionScript 1.0**； **Flash Player 6**

### 返回

Number - 一个指定共享对象大小的数值，以字节为单位。

### 示例

下面的示例获取共享对象 my\_so 的大小：

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserString:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserString;

var soSize:Number = my_so.getSize();
trace(soSize);
```

# onStatus（SharedObject.onStatus 处理函数）

`onStatus = function(info:Object) {}`

每次为共享对象公布错误、警告或信息性通知时调用。如果要响应此事件处理函数，您必须创建一个函数来处理共享对象生成的信息对象。

信息对象具有一个代码属性（其中包含用以描述 `onStatus` 处理函数结果的字符串）和一个 `level` 属性（其中包含字符串 `"Status"` 或 `"Error"`）。

除此 `onStatus` 处理函数外，**Flash** 还提供称为 `System.onStatus` 的超级函数。如果为特定对象调用了 `onStatus` 但未分配任何函数对其进行响应，则 **Flash** 将处理分配给 `System.onStatus` 的函数（如果存在）。

下列事件在发生某些 `SharedObject` 活动时通知您：

代码属性	级别属性	含义
<code>SharedObject.Flush.Failed</code>	<code>Error</code>	返回 "pending" 的 <code>SharedObject.flush()</code> 命令已失败（当 <b>Flash Player</b> 显示“本地存储设置”对话框时，用户没有为共享对象分配额外的磁盘空间）。
<code>SharedObject.Flush.Success</code>	<code>Status</code>	返回 "pending" 的 <code>SharedObject.flush()</code> 命令已成功完成（用户为共享对象分配了额外的磁盘空间）。

可用性：ActionScript 1.0；Flash Player 6

## 参数

`info:Object` - 根据状态消息定义的参数。

## 示例

下面的示例根据用户选择允许还是拒绝 `SharedObject` 对象实例写入磁盘，显示不同的消息。

```
var message_str:String;
this.createTextField("message_txt", this.getNextHighestDepth(), 0, 0, 300,
    22);
message_txt.html = true;
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 30, 300,
    100);
status_txt.multiline = true;
status_txt.html = true;

var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserUsername:String = "Ramona";
```

```

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUser_name;

my_so.onStatus = function(infoObject:Object) {
    status_txt.htmlText = "<textformat tabStops='[50]'\>";
    for (var i in infoObject) {
        status_txt.htmlText += "<b>"+i+"</b>"+ "\t"+infoObject[i];
    }
    status_txt.htmlText += "</textformat>";
};

var flushResult = my_so.flush(1000001);
switch (flushResult) {
case 'pending' :
    message_str = "flush is pending, waiting on user interaction.";
    break;
case true :
    message_str = "flush was successful. Requested storage space approved.";
    break;
case false :
    message_str = "flush failed. User denied request for additional storage.";
    break;
}
message_txt.htmlText = "<a href=\"\"asfunction:System.showSettings,1\"><u>"+message_str+"</u></a>";

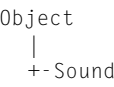
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[getLocal](#) ([SharedObject.getLocal](#) 方法), [onStatus](#) ([System.onStatus](#) 处理函数)

# Sound



```
public class Sound
extends Object
```

**Sound** 类使您可以控制影片中的声音。可以在影片正在播放时从库中向该影片剪辑添加声音，并控制这些声音。如果在创建新 **Sound** 对象时没有指定 **target**，则可以使用方法控制整个影片的声音。

必须使用构造函数 `new Sound` 创建一个 **Sound** 对象后才能调用 **Sound** 类的方法。

可用性：ActionScript 1.0；Flash Player 5

## 属性摘要

修饰符	属性	说明
	<code>duration:Number</code> [ 只读 ]	声音的持续时间（以毫秒为单位）。
	<code>id3:Object</code> [ 只读 ]	提供对作为 MP3 文件一部分的元数据的访问。
	<code>position:Number</code> [ 只读 ]	声音已播放的毫秒数。

继承自 **Object** 类的属性

[constructor](#) ([Object.constructor](#) 属性), [\\_\\_proto\\_\\_](#) ([Object.\\_\\_proto\\_\\_](#) 属性), [prototype](#) ([Object.prototype](#) 属性), [\\_\\_resolve](#) ([Object.\\_\\_resolve](#) 属性)

## 事件摘要

事件	说明
<code>onID3 = function() {}</code>	每次有新的 ID3 数据可用于使用 <code>Sound.attachSound()</code> 或 <code>Sound.loadSound()</code> 加载的 MP3 文件时调用。
<code>onLoad = function(success:Boolean) {}</code>	加载声音时自动调用。
<code>onSoundComplete = function() {}</code>	声音结束播放时自动调用。

构造函数摘要

签名	说明
Sound([target:Object])	为指定的影片剪辑创建新的 Sound 对象。

方法摘要

修饰符	签名	说明
	attachSound(id:String) : Void	将在 id 参数中指定的声音附加到指定的 Sound 对象。
	getBytesLoaded() : Number	返回为指定 Sound 对象加载（流式处理）的字节数。
	getBytesTotal() : Number	以字节为单位返回指定 Sound 对象的大小。
	getPan() : Number	返回在上一次 setPan() 调用中设置的面板级别，这是一个从 -100（左）到 +100（右）的整数。
	getTransform() : Object	返回用上一次 Sound.setTransform() 调用设置的指定 Sound 对象的声音转换信息。
	getVolume() : Number	返回音量级别，这是一个从 0 到 100 的整数，其中 0 表示关闭，100 表示最大音量。
	loadSound(url:String, isStreaming:Boolean) : Void	将 MP3 文件加载到 Sound 对象中。
	setPan(value:Number) : Void	确定声音在左右声道（扬声器）中是如何播放的。
	setTransform(transformObject:Object) : Void	设置 Sound 对象的声音转换（或均衡）信息。
	setVolume(value:Number) : Void	设置 Sound 对象的音量。
	start([secondOffset:Number], [loops:Number]) : Void	从开头开始播放（如果未指定参数）最后附加的声音，或者从由 secondOffset 参数指定的声音点处开始播放。
	stop([linkageID:String]) : Void	停止当前播放的所有声音（如果未指定参数），或者只停止播放 idName 参数中指定的声音。



继承自 Object 类的方法

---

`addProperty` (Object.addProperty 方法), `hasOwnProperty` (Object.hasOwnProperty 方法), `isPropertyEnumerable` (Object.isPropertyEnumerable 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `registerClass` (Object.registerClass 方法), `toString` (Object.toString 方法), `unwatch` (Object.unwatch 方法), `valueOf` (Object.valueOf 方法), `watch` (Object.watch 方法)

---

## attachSound (Sound.attachSound 方法)

```
public attachSound(id:String) : Void
```

将在 `id` 参数中指定的声音附加到指定的 **Sound** 对象。该声音必须位于当前 **SWF** 文件的库中, 并且必须已经在 “链接属性” 对话框中指定为导出。必须调用 `Sound.start()` 才能开始播放此声音。

为了确保从 **SWF** 文件中的任何场景都可以控制声音, 请将声音放置在 **SWF** 文件的主时间轴上。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

`id:String` - 库中导出声音的标识符。该标识符位于 “链接属性” 对话框。

### 示例

下面的示例将声音 `logoff_id` 附加到 `my_sound`。库中的声音具有链接标识符 `logoff_id`。

```
var my_sound:Sound = new Sound();
my_sound.attachSound("logoff_id");
my_sound.start();
```

## duration (Sound.duration 属性)

```
public duration : Number [read-only]
```

声音的持续时间 (以毫秒为单位)。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例加载声音, 并将声音文件的持续时间显示在 “输出” 面板中。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件。

```
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    var totalSeconds:Number = this.duration/1000;
    trace(this.duration+" ms (" +Math.round(totalSeconds)+" seconds)");
};
```

```

var minutes:Number = Math.floor(totalSeconds/60);
var seconds = Math.floor(totalSeconds)%60;
if (seconds<10) {
    seconds = "0"+seconds;
}
trace(minutes+": "+seconds);
};
my_sound.loadSound("song1.mp3", true);

```

下面的示例将几首歌曲加载到 SWF 文件中。一个使用 **Drawing API** 创建的进度栏会显示加载进度。当音乐开始加载和完成加载时，“输出”面板中会显示信息。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件。

```

var pb_height:Number = 10;
var pb_width:Number = 100;
var pb:MovieClip = this.createEmptyMovieClip("progressBar_mc",
    this.getNextHighestDepth());
pb.createEmptyMovieClip("bar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("vBar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("stroke_mc", pb.getNextHighestDepth());
pb.createTextField("pos_txt", pb.getNextHighestDepth(), 0, pb_height,
    pb_width, 22);

pb._x = 100;
pb._y = 100;

with (pb.bar_mc) {
    beginFill(0x00FF00);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
    endFill();
    _xscale = 0;
}
with (pb.vBar_mc) {
    lineStyle(1, 0x000000);
    moveTo(0, 0);
    lineTo(0, pb_height);
}
with (pb.stroke_mc) {
    lineStyle(3, 0x000000);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
}

```

```

var my_interval:Number;
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        trace("sound loaded");
    }
};
my_sound.onSoundComplete = function() {
    clearInterval(my_interval);
    trace("Cleared interval");
}
my_sound.loadSound("song3.mp3", true);
my_interval = setInterval(updateProgressBar, 100, my_sound);

function updateProgressBar(the_sound:Sound):Void {
    var pos:Number = Math.round(the_sound.position/the_sound.duration 100);
    pb.bar_mc._xscale = pos;
    pb.vBar_mc._x = pb.bar_mc._width;
    pb.pos_txt.text = pos+"%";
}

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[position \(Sound.position 属性\)](#)

## getBytesLoaded (Sound.getBytesLoaded 方法)

`public getBytesLoaded() : Number`

返回为指定 **Sound** 对象加载（流式处理）的字节数。可以将 `getBytesLoaded()` 的值与 `getBytesTotal()` 的值进行比较，以确定已加载声音的百分比。

**可用性:** **ActionScript 1.0** ; **Flash Player 6**

**返回**

**Number** - 一个整数，指示已加载的字节数。

## 示例

下面的示例动态创建两个文本字段，它们显示加载到 SWF 文件中的声音文件的已加载字节数和总字节数。当文件完成加载时，有一个文本字段还会显示消息。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
this.createTextField("message_txt", this.getNextHighestDepth(),
    10,10,300,22)
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 50, 300,
    40);
status_txt.autoSize = true;
status_txt.multiline = true;
status_txt.border = false;

var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        this.start();
        message_txt.text = "Finished loading";
    }
};
my_sound.onSoundComplete = function() {
    message_txt.text = "Clearing interval";
    clearInterval(my_interval);
};
my_sound.loadSound("song2.mp3", true);
var my_interval:Number;
my_interval = setInterval(checkProgress, 100, my_sound);
function checkProgress(the_sound:Sound):Void {
    var pct:Number = Math.round(the_sound.getBytesLoaded()/
        the_sound.getBytesTotal() 100);
    var pos:Number = Math.round(the_sound.position/the_sound.duration 100);
    status_txt.text = the_sound.getBytesLoaded()+" of
        "+the_sound.getBytesTotal()+" bytes (" +pct+"%")+newline;
    status_txt.text += the_sound.position+" of "+the_sound.duration+"
        milliseconds (" +pos+"%")+newline;
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[getBytesTotal \(Sound.getBytesTotal 方法\)](#)

## getBytesTotal (Sound.getBytesTotal 方法)

`public getBytesTotal() : Number`

以字节为单位返回指定 **Sound** 对象的大小。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 返回

**Number** - 一个整数，以字节为单位，指示指定 **Sound** 对象的总大小。

### 示例

有关此方法的示例用法，请参见 `Sound.getBytesLoaded()`。

### 另请参见

[getBytesLoaded \(Sound.getBytesLoaded 方法\)](#)

## getPan (Sound.getPan 方法)

`public getPan() : Number`

返回在上一次 `setPan()` 调用中设置的面板级别，这是一个从 **-100**（左）到 **+100**（右）的整数。（**0** 平衡地设置左右声道。）该面板设置控制 **SWF** 文件中当前和将来声音的左右均衡。

此方法对于 `setVolume()` 或 `setTransform()` 是渐增的。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

**Number** - 一个整数。

### 示例

下面的示例使用 **Drawing API** 创建一个滑块。当用户拖动滑块时，已加载声音的面板级别会发生变化。当前的面板级别显示在动态创建的文本字段中。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
var bar_width:Number = 200;
this.createEmptyMovieClip("bar_mc", this.getNextHighestDepth());
with (bar_mc) {
    lineStyle(4, 0x000000);
    moveTo(0, 0);
    lineTo(bar_width+4, 0);
    lineStyle(0, 0x000000);
    moveTo((bar_width/2)+2, -8);
    lineTo((bar_width/2)+2, 8);
}
```

```

bar_mc._x = 100;
bar_mc._y = 100;

this.createEmptyMovieClip("knob_mc", this.getNextHighestDepth());
with (knob_mc) {
    lineStyle(0, 0x000000);
    beginFill(0xCCCCCC);
    moveTo(0, 0);
    lineTo(4, 0);
    lineTo(4, 10);
    lineTo(0, 10);
    lineTo(0, 0);
    endFill();
}
knob_mc._x = bar_mc._x+(bar_width/2);
knob_mc._y = bar_mc._y-(knob_mc._height/2);

knob_mc.left = knob_mc._x-(bar_width/2);
knob_mc.right = knob_mc._x+(bar_width/2);
knob_mc.top = knob_mc._y;
knob_mc.bottom = knob_mc._y;

knob_mc.onPress = function() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
};
knob_mc.onRelease = function() {
    this.stopDrag();
    var multiplier:Number = 100/(this.right-this.left) 2;
    var pan:Number = (this._x-this.left-(bar_width/2)) multiplier;
    my_sound.setPan(pan);
    pan_txt.text = my_sound.getPan();
};

var my_sound:Sound = new Sound();
my_sound.loadSound("song2.mp3", true);
this.createTextField("pan_txt", this.getNextHighestDepth(), knob_mc._x,
    knob_mc._y+knob_mc._height, 20, 22);
pan_txt.selectable = false;
pan_txt.autoSize = "center";
pan_txt.text = my_sound.getPan();

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setPan \(Sound.setPan 方法\)](#)

## getTransform (Sound.getTransform 方法)

```
public getTransform(): Object
```

返回用上一次 `Sound.setTransform()` 调用设置的指定 **Sound** 对象的声音转换信息。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Object - 一个具有属性的对象，这些属性包含指定声音对象的声道百分比值。

### 示例

下面的示例从库中的元件附加用作滑块（或调节器）的四个影片剪辑（链接标识符：**knob\_id**），`knob_id`），以控制加载到 **SWF** 文件中的声音文件。这些滑块控制声音文件的转换对象或均衡。有关更多信息，请参见 `Sound.setTransform()` 的条目。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
var transform_obj:Object = my_sound.getTransform();

this.createEmptyMovieClip("transform_mc", this.getNextHighestDepth());
transform_mc.createTextField("transform_txt",
    transform_mc.getNextHighestDepth, 0, 8, 120, 22);
transform_mc.transform_txt.html = true;

var knob_ll:MovieClip = transform_mc.attachMovie("knob_id", "ll_mc",
    transform_mc.getNextHighestDepth(), {_x:0, _y:30});
var knob_lr:MovieClip = transform_mc.attachMovie("knob_id", "lr_mc",
    transform_mc.getNextHighestDepth(), {_x:30, _y:30});
var knob_rl:MovieClip = transform_mc.attachMovie("knob_id", "rl_mc",
    transform_mc.getNextHighestDepth(), {_x:60, _y:30});
var knob_rr:MovieClip = transform_mc.attachMovie("knob_id", "rr_mc",
    transform_mc.getNextHighestDepth(), {_x:90, _y:30});

knob_ll.top = knob_ll._y;
knob_ll.bottom = knob_ll._y+100;
knob_ll.left = knob_ll._x;
knob_ll.right = knob_ll._x;
knob_ll._y = knob_ll._y+(100-transform_obj['ll']);
knob_ll.onPress = pressKnob;
knob_ll.onRelease = releaseKnob;
knob_ll.onReleaseOutside = releaseKnob;

knob_lr.top = knob_lr._y;
knob_lr.bottom = knob_lr._y+100;
knob_lr.left = knob_lr._x;
knob_lr.right = knob_lr._x;
knob_lr._y = knob_lr._y+(100-transform_obj['lr']);
```

```

knob_lr.onPress = pressKnob;
knob_lr.onRelease = releaseKnob;
knob_lr.onReleaseOutside = releaseKnob;

knob_rl.top = knob_rl._y;
knob_rl.bottom = knob_rl._y+100;
knob_rl.left = knob_rl._x;
knob_rl.right = knob_rl._x;
knob_rl._y = knob_rl._y+(100-transform_obj['rl']);
knob_rl.onPress = pressKnob;
knob_rl.onRelease = releaseKnob;
knob_rl.onReleaseOutside = releaseKnob;

knob_rr.top = knob_rr._y;
knob_rr.bottom = knob_rr._y+100;
knob_rr.left = knob_rr._x;
knob_rr.right = knob_rr._x;
knob_rr._y = knob_rr._y+(100-transform_obj['rr']);
knob_rr.onPress = pressKnob;
knob_rr.onRelease = releaseKnob;

knob_rr.onReleaseOutside = releaseKnob;

updateTransformTxt();

function pressKnob() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
}
function releaseKnob() {
    this.stopDrag();
    updateTransformTxt();
}
function updateTransformTxt() {
    var ll_num:Number = 30+100-knob_ll._y;
    var lr_num:Number = 30+100-knob_lr._y;
    var rl_num:Number = 30+100-knob_rl._y;
    var rr_num:Number = 30+100-knob_rr._y;
    my_sound.setTransform({ll:ll_num, lr:lr_num, rl:rl_num, rr:rr_num});
    transform_mc.transform_txt.htmlText = "<textformat
    tabStops='[0,30,60,90]'\>";
    transform_mc.transform_txt.htmlText +=
    ll_num+"\t"+lr_num+"\t"+rl_num+"\t"+rr_num;
    transform_mc.transform_txt.htmlText += "</textformat>";
}

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。



另请参见

[setTransform](#) ([Sound.setTransform](#) 方法)

## getVolume (Sound.getVolume 方法)

`public getVolume() : Number`

返回音量级别，这是一个从 0 到 100 的整数，其中 0 表示关闭，100 表示最大音量。默认设置为 100。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

Number - 一个整数。

### 示例

下面的示例使用 **Drawing API** 和在运行时创建的影片剪辑创建一个滑块。一个动态创建的文本字段显示在 SWF 文件中播放的声音的当前音量级别。将以下 **ActionScript** 添加到 **AS** 或 **FLA** 文件:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song3.mp3", true);

this.createEmptyMovieClip("knob_mc", this.getNextHighestDepth());

knob_mc.left = knob_mc._x;
knob_mc.right = knob_mc.left+100;
knob_mc.top = knob_mc._y;
knob_mc.bottom = knob_mc._y;

knob_mc._x = my_sound.getVolume();

with (knob_mc) {
    lineStyle(0, 0x000000);
    beginFill(0xCCCCCC);
    moveTo(0, 0);
    lineTo(4, 0);
    lineTo(4, 18);
    lineTo(0, 18);
    lineTo(0, 0);
    endFill();
}

knob_mc.createTextField("volume_txt", knob_mc.getNextHighestDepth(),
    knob_mc._width+4, 0, 30, 22);
knob_mc.volume_txt.text = my_sound.getVolume();
```

```

knob_mc.onPress = function() {
    this.startDrag(false, this.left, this.top, this.right, this.bottom);
    this.isDragging = true;
};
knob_mc.onMouseMove = function() {
    if (this.isDragging) {
        this.volume_txt.text = this._x;
    }
}
knob_mc.onRelease = function() {
    this.stopDrag();
    this.isDragging = false;
    my_sound.setVolume(this._x);
};

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setVolume \(Sound.setVolume 方法\)](#)

## id3 (Sound.id3 属性)

```
public id3 : Object [read-only]
```

提供对作为 **MP3** 文件一部分的元数据的访问。

**MP3** 声音文件可以包含 **ID3** 标签，**ID3** 标签提供有关该文件的元数据。如果使用 `Sound.attachSound()` 或 `Sound.loadSound()` 加载的 **MP3** 声音包含 **ID3** 标签，您可以查询这些属性。只支持使用 **UTF-8** 字符集的 **ID3** 标签。

**Flash Player 6 (6.0.40.0)** 和更高版本使用 `Sound.id3` 属性支持 **ID3 1.0** 和 **ID3 1.1** 标签。

**Flash Player 7** 添加了对 **ID3 2.0** 标签的支持，特别是 **2.3** 和 **2.4**。下表列出了标准 **ID3 2.0** 标签以及这些标签表示的内容的类型；您可以以 `my_sound.id3.COMM`、`my_sound.id3.TIME` 等格式查询这些标签。**MP3** 文件可以包含此表中所列标签之外的其它标签；`Sound.id3` 也提供对这些标签的访问。

属性	说明
TFLT	文件类型
TIME	时间
TIT1	内容组描述
TIT2	标题 / 歌曲名称 / 内容描述
TIT3	副标题 / 主要描述
TKEY	最初关键字
TLAN	语言
TLEN	长度
TMED	媒体类型
TOAL	原唱片 / 影片 / 演出名称
TOFN	原文件名
TOLY	原词作者 / 乐谱作者
TOPE	原艺术家 / 演奏者
TORY	最初发行年份
TOWN	文件所有者 / 许可证持有人
TPE1	领衔演奏者 / 独唱（奏）者
TPE2	乐队 / 交响乐队 / 伴奏
TPE3	指挥 / 主要演奏者
TPE4	翻译、混录员、或修改者
TPOS	作品集的部分
TPUB	发行人
TRCK	音轨编号 / 作品集中的位置
TRDA	录制日期
TRSN	Internet 电台名称
TRSO	Internet 电台所有者
TSIZ	大小
TSRC	ISRC（国际标准音像制品编码）
TSSE	编码使用的软件 / 硬件和设置
TYER	年份
WXXX	URL 链接帧

Flash Player 6 支持若干种 ID3 1.0 标签。如果这些标签不在 MP3 文件中，而相应的 ID3 2.0 标签却在 MP3 文件中，则 ID3 2.0 标签会被复制到下表所示的 ID3 1.0 属性中。此过程提供与您已编写的、读取 ID3 1.0 属性的脚本的向后兼容性。

ID3 2.0 标签	相应的 ID3 1.0 属性
COMM	Sound.id3.comment
TALB	Sound.id3.album
TCON	Sound.id3.genre
TIT2	Sound.id3.songname
TPE1	Sound.id3.artist
TRCK	Sound.id3.track
TYER	Sound.id3.year

可用性：ActionScript 1.0 ； Flash Player 6

示例

下面的示例用 `trace` 命令将 `song.mp3` 的 ID3 属性发送到 “输出” 面板：

```
var my_sound:Sound = new Sound();
my_sound.onID3 = function(){
    for( var prop in my_sound.id3 ){
        trace( prop + " : "+ my_sound.id3[prop] );
    }
}
my_sound.loadSound("song.mp3", false);
```

另请参见

[attachSound \(Sound.attachSound 方法\)](#)， [loadSound \(Sound.loadSound 方法\)](#)

## loadSound (Sound.loadSound 方法)

```
public loadSound(url:String, isStreaming:Boolean) : Void
```

将 MP3 文件加载到 **Sound** 对象中。可以使用 `isStreaming` 参数指示该声音是事件声音还是声音流。

事件声音在完全加载后才能播放。它们由 **ActionScript Sound** 类进行管理，并且响应此类的所有方法和属性。

声音流在下载的同时播放。当接收的数据足以启动解压缩程序时，播放即开始。

使用此方法加载的所有 MP3（事件或流）都保存在用户系统上浏览器的文件缓存中。

使用此方法时，请考虑 **Flash Player** 安全模型。

对于 **Flash Player 8**：

- 如果执行调用的 **SWF** 文件在只能与本地文件系统的内容交互的沙箱中，并且声音在网络沙箱中，则不允许 `Sound.loadSound()`。
- 从受信任的本地沙箱或只能与远程内容交互的沙箱进行访问需要通过跨域策略文件获得网站的许可。

对于 **Flash Player 7** 及更高版本：

- 网站可以允许通过跨域策略文件根据不同域中的请求访问资源。

有关更多信息，请参见以下部分：

- 《学习 **Flash** 中的 **ActionScript 2.0**》的第 17 章，“了解安全性”
- **Flash Player 8** 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- **Flash Player 8** 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 6

### 参数

`url:String` - MP3 声音文件在服务器上的位置。

`isStreaming:Boolean` - 一个布尔值，指示声音是声音流 (`true`) 还是事件声音 (`false`)。

### 示例

下面的示例加载事件声音，它在完全加载前是无法播放的：

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", false);
```

下面的示例加载声音流：

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
```

另请参见

[onLoad \(Sound.onLoad 处理函数\)](#)

## onID3 (Sound.onID3 处理函数)

```
onID3 = function() {}
```

每次有新的 ID3 数据可用于使用 `Sound.attachSound()` 或 `Sound.loadSound()` 加载的 MP3 文件时调用。此处理函数提供对 ID3 数据的访问而无需轮询。如果文件中同时出现 ID3 1.0 和 ID3 2.0 标签，则此处理函数将被调用两次。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 示例

下面的示例向 **DataGrid** 组件的实例显示 song1.mp3 的 ID3 属性。将实例名称为 id3\_dg 的 **DataGrid** 添加到您的文档，然后将以下 **ActionScript** 添加到您的 FLA 或 AS 文件：

```
import mx.controls.gridclasses.DataGridColumn;
var id3_dg:mx.controls.DataGrid;
id3_dg.move(0, 0);
id3_dg.setSize(Stage.width, Stage.height);
var property_dgc:DataGridColumn = id3_dg.addColumn(new
    DataGridColumn("property"));
property_dgc.width = 100;
property_dgc.headerText = "ID3 Property";
var value_dgc:DataGridColumn = id3_dg.addColumn(new
    DataGridColumn("value"));
value_dgc.width = id3_dg._width-property_dgc.width;
value_dgc.headerText = "ID3 Value";

var my_sound:Sound = new Sound();
my_sound.onID3 = function() {
    trace("onID3 called at "+getTimer()+" ms.");
    for (var prop in this.id3) {
        id3_dg.addItem({property:prop, value:this.id3[prop]});
    }
};
my_sound.loadSound("song1.mp3", true);
```

另请参见

[attachSound \(Sound.attachSound 方法\)](#)，[id3 \(Sound.id3 属性\)](#)，[loadSound \(Sound.loadSound 方法\)](#)

## onLoad (Sound.onLoad 处理函数)

```
onLoad = function(success:Boolean) {}
```

加载声音时自动调用。必须创建在调用此处理函数时执行的函数。您可以使用匿名函数或命名函数（有关每种情况的示例，请参见 `Sound.onSoundComplete`）。在调用 `mySound.loadSound()` 之前，您应该定义此处理函数。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**success:** Boolean - 一个布尔值，如果已成功加载 `my_sound`，则为 `true`；否则为 `false`。

### 示例

以下示例创建新的 **Sound** 对象并加载声音。加载声音由 `onLoad` 处理函数处理，它允许您在成功加载歌曲后开始播放该歌曲。请创建一个新的 **FLA** 文件，并将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件。为了使此示例正常工作，名为 `song1.mp3` 的 **MP3** 必须位于 **FLA** 或 **AS** 文件所在的目录中。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// if the sound loads, play it; if not, trace failure loading
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        my_sound.start();
        status_txt.text = "Sound loaded";
    } else {
        status_txt.text = "Sound failed";
    }
};
// load the sound
my_sound.loadSound("song1.mp3", true);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[loadSound \(Sound.loadSound 方法\)](#)

## onSoundComplete（Sound.onSoundComplete 处理函数）

```
onSoundComplete = function() {}
```

声音结束播放时自动调用。您可以使用此处理函数在声音播放完毕后触发 SWF 文件中的事件。

必须创建在调用此处理函数时执行的函数。您既可以使用匿名函数，也可以使用命名函数。

可用性：ActionScript 1.0；Flash Player 6

### 示例

用法 1：下面的示例使用匿名函数：

```
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = function() {
    trace("mySoundID completed");
};
my_sound.start();
```

用法 2：下面的示例使用命名函数：

```
function callback1() {
    trace("mySoundID completed");
}
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = callback1;
my_sound.start();
```

另请参见

[onLoad（Sound.onLoad 处理函数）](#)



## position (Sound.position 属性)

`public position : Number [read-only]`

声音已播放的毫秒数。如果该声音循环播放，则在每次循环开始时，将 **position** 重置为 0。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

有关此属性的示例用法，请参见 `Sound.duration`。

### 另请参见

[duration \(Sound.duration 属性\)](#)

## setPan (Sound.setPan 方法)

`public setPan(value:Number) : Void`

确定声音在左右声道（扬声器）中是如何播放的。对于单声道声音，*pan* 决定通过哪个扬声器（左或右）播放声音。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**value:**Number - 一个整数，指定声音的左右均衡。有效值的范围为 **-100** 到 **100**，其中 **-100** 表示仅使用左声道，**100** 表示仅使用右声道，而 **0** 表示在两个声道间平均地均衡声音。

### 示例

有关此方法的示例用法，请参见 `Sound.getPan()`。

### 另请参见

[attachSound \(Sound.attachSound 方法\)](#)，[getPan \(Sound.getPan 方法\)](#)，[setTransform \(Sound.setTransform 方法\)](#)，[setVolume \(Sound.setVolume 方法\)](#)，[start \(Sound.start 方法\)](#)

## setTransform (Sound.setTransform 方法)

```
public setTransform(transformObject:Object) : Void
```

设置 **Sound** 对象的声音转换（或均衡）信息。

**soundTransformObject** 参数是一个使用通用 **Object** 类的构造函数方法创建的对象，该对象具有指定声音在左右声道（扬声器）间分布方式的参数。

声音占用相当大的磁盘空间和内存。因为立体声声音使用的数据是单声道声音的两倍，通常情况下最好使用 **22 KHz 6 位** 单声道声音。您可以使用 `setTransform()` 将单声道声音播放为立体声、将立体声声音播放为单声道声音以及向声音中添加有趣的效果。

**soundTransformObject** 的属性如下：

**ll** - 一个百分比值，指定左输入在左扬声器里播放的量（0 至 100）。

**lr** - 一个百分比值，指定右输入在左扬声器里播放的量（0 至 100）。

**rr** - 一个百分比值，指定右输入在右扬声器里播放的量（0 至 100）。

**rl** - 一个百分比值，指定左输入在右扬声器里播放的量（0 至 100）。

这些参数的净结果由以下公式表示：

```
leftOutput = left_input ~ ll + right_input ~ lr  
rightOutput = right_input ~ rr + left_input ~ rl
```

**left\_input** 或 **right\_input** 的值由 **SWF** 文件中的声音类型（立体声或单声）决定。

立体声声音将声音输入在左右扬声器之间平均分配，且在默认情况下具有以下转换设置：

```
ll = 100  
lr = 0  
rr = 100  
rl = 0
```

单声道声音在左扬声器里播放所有的声音输入，且在默认情况下具有以下转换设置：

```
ll = 100  
lr = 100  
rr = 0  
rl = 0
```

可用性：ActionScript 1.0；Flash Player 5

### 参数

**transformObject:Object** - 一个用通用 **Object** 类的构造函数创建的对象。

## 示例

下面的示例说明了这样一个设置，它可以通过使用 `setTransform()` 实现，但无法通过使用 `setVolume()` 或 `setPan()` 实现，即使将后两种方法结合起来使用也无法实现该设置。

下面的代码创建一个新的 `soundTransformObject` 对象并设置其属性，以便只在左声道中播放来自两个声道的声音。

```
var mySoundTransformObject:Object = new Object();
mySoundTransformObject.ll = 100;
mySoundTransformObject.lr = 100;
mySoundTransformObject.rr = 0;
mySoundTransformObject.rl = 0;
```

若要将 `soundTransformObject` 对象应用到 **Sound** 对象，则需要使用 `setTransform()` 将该对象传递给 **Sound** 对象，如下所示：

```
my_sound.setTransform(mySoundTransformObject);
```

下面的示例将立体声音播放为单声道声音； `soundTransformObjectMono` 对象具有下列参数：

```
var mySoundTransformObjectMono:Object = new Object();
mySoundTransformObjectMono.ll = 50;
mySoundTransformObjectMono.lr = 50;
mySoundTransformObjectMono.rr = 50;
mySoundTransformObjectMono.rl = 50;
my_sound.setTransform(mySoundTransformObjectMono);
```

此示例只播放左声道音量的一半，而将左声道音量的剩余部分添加到右声道：

`soundTransformObjectHalf` 对象具有下列参数：

```
var mySoundTransformObjectHalf:Object = new Object();
mySoundTransformObjectHalf.ll = 50;
mySoundTransformObjectHalf.lr = 0;
mySoundTransformObjectHalf.rr = 100;
mySoundTransformObjectHalf.rl = 50;
my_sound.setTransform(mySoundTransformObjectHalf);
```

```
var mySoundTransformObjectHalf:Object = {ll:50, lr:0, rr:100, rl:50};
```

另请参见 `Sound.getTransform()` 的示例。

另请参见

[Object](#), [getTransform \(Sound.getTransform 方法\)](#)

## setVolume (Sound.setVolume 方法)

```
public setVolume(value:Number) : Void
```

设置 **Sound** 对象的音量。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**value:**Number - 一个从 **0** 到 **100** 之间的数字, 表示音量级别。**100** 为最大音量, 而 **0** 为没有音量。默认设置为 **100**。

### 示例

有关此方法的示例用法, 请参见 `Sound.getVolume()`。

另请参见

[setPan \(Sound.setPan 方法\)](#), [setTransform \(Sound.setTransform 方法\)](#)

## Sound 构造函数

```
public Sound([target:Object])
```

为指定的影片剪辑创建新的 **Sound** 对象。如果没有指定 **target** 实例, 则 **Sound** 对象控制影片中的所有声音。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**target:**Object [ 可选 ] - **Sound** 对象在其上操作的影片剪辑实例。

### 示例

下面的示例创建名为 `global_sound` 的新 **Sound** 对象。示例中的第二行调用 `setVolume()` 并将影片中的所有声音的音量调整到 **50%**。

```
var global_sound:Sound = new Sound();  
global_sound.setVolume(50);
```

下面的示例创建一个新 **Sound** 对象, 将目标影片剪辑 `my_mc` 传递给它, 然后调用 `start` 方法, 该方法开始播放 `my_mc` 中的所有声音。

```
var movie_sound:Sound = new Sound(my_mc);  
movie_sound.start();
```

## start (Sound.start 方法)

```
public start([secondOffset:Number], [loops:Number]) : Void
```

从开头开始播放（如果未指定参数）最后附加的声音，或者从由 `secondOffset` 参数指定的声音点处开始播放。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**secondOffset:**Number [可选] - 一个参数，通过它可以从此特定点开始播放声音。例如，如果您有一个 **30** 秒的声音，而您希望该声音从中间开始播放，可将 `secondOffset` 参数指定为 **15**。并非声音延迟 **15** 秒，而是从 **15** 秒标记处开始播放。

**loops:**Number [可选] - 一个参数，通过它可以指定声音应该连续播放的次数。如果声音是声音流，则此参数不可用。

### 示例

以下示例创建新的 **Sound** 对象并加载声音。加载声音由 `onLoad` 处理函数处理，它允许您在成功加载歌曲后开始播放该歌曲。然后将使用 `start()` 方法开始播放声音。请创建一个新的 **FLA** 文件，并将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件。为了使此示例正常工作，**FLA** 或 **AS** 文件所在的目录必须包含名为 **song1.mp3** 的 **MP3**。

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);
```

```
// create a new Sound object
var my_sound:Sound = new Sound();
// if the sound loads, play it; if not, trace failure loading
my_sound.onLoad = function(success:Boolean) {
    if (success) {
        my_sound.start();
        status_txt.text = "Sound loaded";
    } else {
        status_txt.text = "Sound failed";
    }
};
// load the sound
my_sound.loadSound("song1.mp3", true);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 **2** 版的组件，请使用第 **2** 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[stop \(Sound.stop 方法\)](#)

## stop (Sound.stop 方法)

`public stop([linkageID:String]) : Void`

停止当前播放的所有声音（如果未指定参数），或者只停止播放放在 `idName` 参数中指定的声音。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

`linkageID:String` [ 可选 ] - 指定特定声音停止播放的参数。 `idName` 参数必须置于引号 (" ") 之中。

### 示例

下面的示例使用两个按钮（即 `stop_btn` 和 `play_btn`）控制加载到 **SWF** 文件中的声音的回放。将两个按钮添加到您的文档，并将以下 **ActionScript** 添加到您的 **FLA** 或 **AS** 文件：

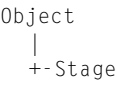
```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
```

```
stop_btn.onRelease = function() {
    trace("sound stopped");
    my_sound.stop();
};
play_btn.onRelease = function() {
    trace("sound started");
    my_sound.start();
};
```

### 另请参见

[start \(Sound.start 方法\)](#)

# Stage



```
public class Stage
extends Object
```

**Stage** 类是一个顶级类，不必使用构造函数即可访问其方法、属性和处理函数。此类的方法和属性用于访问和操作有关 **SWF** 文件边界的信息。

可用性：ActionScript 1.0 ； Flash Player 5

### 属性摘要

修饰符	属性	说明
static	align:String	指示 SWF 文件在播放器或浏览器中当前的对齐方式。
static	height:Number	属性（只读）；以像素为单位指示舞台的当前高度。
static	scaleMode:String	指示 SWF 文件在 Flash Player 内当前的缩放比例。
static	showMenu:Boolean	指定显示或隐藏 Flash Player 上下文菜单中的默认项。
static	width:Number	属性（只读）；以像素为单位指示舞台的当前宽度。

继承自 Object 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

### 事件摘要

事件	说明
onResize = function() {}	当 Stage.scaleMode 设置为 noScale 且调整 SWF 文件大小时调用。

方法摘要

修饰符	签名	说明
static	<code>addListener(listener:Object) : Void</code>	检测何时调整 SWF 文件的大小（但只有在 <code>Stage.scaleMode = "noScale"</code> 的情况下）。
static	<code>removeListener(listener:Object) : Boolean</code>	删除用 <code>addListener()</code> 创建的侦听器对象。

继承自 `Object` 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## addListener (Stage.addListener 方法)

```
public static addListener(listener:Object) : Void
```

检测何时调整 SWF 文件的大小（但只有在 `Stage.scaleMode = "noScale"` 的情况下）。  
`addListener()` 方法不能与默认的电影剪辑缩放设置 (`showAll`) 或其它缩放设置 (`exactFit` 和 `noBorder`) 一起使用。

若要使用 `addListener()`，必须先创建一个侦听器对象。舞台侦听器对象接收来自 `Stage.onResize` 的通知。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**listener:Object** - 一个对象，侦听来自 `Stage.onResize` 事件的回调通知。

### 示例

此示例创建一个名为 `stageListener` 的新侦听器对象。然后它使用 `stageListener` 来调用 `onResize`，并定义一个将在 `onResize` 受到激发时所调用的函数。最后，该代码将 `stageListener` 对象添加到 **Stage** 对象的回调列表中。侦听器对象允许多个对象侦听调整大小的通知。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```



另请参见

[onResize \(Stage.onResize 事件侦听器\)](#), [removeListener \(Stage.removeListener 方法\)](#)

## align (Stage.align 属性)

`public static align : String`

指示 SWF 文件在播放器或浏览器中当前的对齐方式。

下表列出了 align 属性的值。此处没有列出的任何值都会使 SWF 文件在 Flash Player 或浏览器区域中居中对齐，这是默认设置。

值	垂直	水平
"T"	顶对齐	居中
"B"	底对齐	居中
"L"	居中	左对齐
"R"	居中	右对齐
"TL"	顶对齐	左对齐
"TR"	顶对齐	右对齐
"BL"	底对齐	左对齐
"BR"	底对齐	右对齐

可用性：ActionScript 1.0 ； Flash Player 6

### 示例

下面的示例演示 SWF 文件的不同对齐方式。将实例名称为 stageAlign\_cb 的 ComboBox 实例添加到您的文档。请将以下 ActionScript 添加到 FLA 或 AS 文件：

```
var stageAlign_cb:mx.controls.ComboBox;
stageAlign_cb.dataProvider = ['T', 'B', 'L', 'R', 'TL', 'TR', 'BL', 'BR'];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    var align:String = evt.target.selectedItem;
    Stage.align = align;
};
stageAlign_cb.addEventListener("change", cbListener);
Stage.scaleMode = "noScale";
```

从 ComboBox 中选择不同的对齐设置。

## height (Stage.height 属性)

```
public static height : Number
```

属性（只读）：以像素为单位指示舞台的当前高度。如果 `Stage.scaleMode` 的值为 `noScale`，则 `height` 属性表示 **Flash Player** 的高度。如果 `Stage.scaleMode` 的值不为 `noScale`，则 `height` 表示 **SWF** 文件的高度。

可用性：ActionScript 1.0；Flash Player 6

### 示例

此示例创建一个名为 `stageListener` 的新侦听器对象。然后它使用 `myListener` 来调用 `onResize`，并定义一个将在 `onResize` 受到激发时所调用的函数。最后，该代码将 `myListener` 对象添加到 **Stage** 对象的回调列表中。侦听器对象允许多个对象侦听调整大小的通知。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10,
    100, 22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

### 另请参见

[align \(Stage.align 属性\)](#)，[scaleMode \(Stage.scaleMode 属性\)](#)，[width \(Stage.width 属性\)](#)

## onResize (Stage.onResize 事件侦听器)

```
onResize = function() {}
```

当 `Stage.scaleMode` 设置为 `noScale` 且调整 **SWF** 文件大小时调用。可以利用此事件处理函数编写一个函数，当调整 **SWF** 文件的大小时布置舞台上对象的布局。

```
myListener.onResize = function()[
    // your statements here
]
```

可用性：ActionScript 1.0；Flash Player 6

## 示例

当调整舞台大小时，下面的示例在“输出”面板中显示一条消息：

```
Stage.scaleMode = "noScale"
var myListener:Object = new Object();
myListener.onResize = function () {
    trace("Stage size is now " + Stage.width + " by " + Stage.height);
}
Stage.addListener(myListener);
// later, call Stage.removeListener(myListener)
```

## 另请参见

[scaleMode \(Stage.scaleMode 属性\)](#), [addListener \(Stage.addListener 方法\)](#),  
[removeListener \(Stage.removeListener 方法\)](#)

# removeListener (Stage.removeListener 方法)

```
public static removeListener(listener:Object) : Boolean
```

删除用 `addListener()` 创建的侦听器对象。

可用性：ActionScript 1.0；Flash Player 6

## 参数

**listener:Object** - 使用 `addListener()` 添加到对象的回调列表的一个对象。

## 返回

Boolean - 一个布尔值。

## 示例

下面的示例在动态创建的文本字段中显示舞台尺寸。调整舞台大小时，将更新文本字段中的值。创建一个实例名称为 `remove_btn` 的按钮。将以下 **ActionScript** 添加到时间轴的第 1 帧。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10,
    100, 22);
stageSize_txt.autoSize = true;
stageSize_txt.border = true;
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.addListener(stageListener);

remove_btn.onRelease = function() {
    stageSize_txt.text = "Removing Stage listener...";
    Stage.removeListener(stageListener);
}
```

选择“控制”>“测试影片”对此示例进行测试。在调整测试环境大小时，将更新在文本字段中看到的值。单击 remove\_btn 时，侦听器将被删除，且在文本字段中不再更新值。

另请参见

[addListener \(Stage.addListener 方法\)](#)

## scaleMode (Stage.scaleMode 属性)

```
public static scaleMode : String
```

指示 SWF 文件在 **Flash Player** 内当前的缩放比例。scaleMode 属性强制 SWF 文件设置为特定的缩放模式。默认情况下，SWF 文件使用在“发布设置”对话框中设置的 HTML 参数。

scaleMode 属性可以使用值 "exactFit"、"showAll"、"noBorder" 和 "noScale"。任何其它值都会将 scaleMode 属性设置为默认值 "showAll"。

- showAll（默认值）使整个 **Flash** 内容在指定区域中可见，且不会发生扭曲，同时保持原始高宽比。边框可能出现在应用程序的两侧。
- noBorder 对 **Flash** 内容进行缩放以填充指定区域，不会发生扭曲，它会使应用程序保持原始高宽比，但有可能会进行一些裁剪。
- exactFit 使整个 **Flash** 内容在指定区域中可见，但不尝试保持原始高宽比。可能发生扭曲。
- noScale 使 **Flash** 内容的尺寸固定，因此，即使在更改播放器窗口大小时，它仍然保持不变。如果播放器窗口比 **Flash** 内容小，则可能进行一些裁剪。



除了在测试影片模式中默认设置为 noScale 以外，默认设置均为 showAll。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例演示 SWF 文件的各种缩放设置。将实例名称为 scaleMode\_cb 的 **ComboBox** 实例添加到您的文档。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
var scaleMode_cb:mx.controls.ComboBox;
scaleMode_cb.dataProvider = ["showAll", "exactFit", "noBorder", "noScale"];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
    var scaleMode_str:String = evt.target.selectedItem;
    Stage.scaleMode = scaleMode_str;
};
scaleMode_cb.addEventListener("change", cbListener);
```

若要查看另一示例，请参见 **ActionScript** 示例文件夹中的 **stagesize fla** 文件。下面的列表提供到 **ActionScript** 示例文件夹的典型路径：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

## showMenu (Stage.showMenu 属性)

public static showMenu : Boolean

指定显示或隐藏 **Flash Player** 上下文菜单中的默认项。如果 **showMenu** 设置为 **true**（默认设置），则显示所有上下文菜单项。如果 **showMenu** 设置为 **false**，则只显示“设置”和“关于 **Macromedia Flash Player**”项。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个可单击文本链接，通过该链接用户可以启用和禁用 **Flash Player** 上下文菜单。

```
this.createTextField("showMenu_txt", this.getNextHighestDepth(), 10, 10,
    100, 22);
showMenu_txt.html = true;
showMenu_txt.autoSize = true;
showMenu_txt.htmlText = "<a href=\"asfunction:toggleMenu\"><u>Stage.showMenu
    = "+Stage.showMenu+"</u></a>";
function toggleMenu() {
    Stage.showMenu = !Stage.showMenu;
    showMenu_txt.htmlText = "<a
    href=\"asfunction:toggleMenu\"><u>Stage.showMenu = "+Stage.showMenu+"</
    u></a>";
}
```

另请参见

[ContextMenu](#), [ContextMenuItem](#)

## width (Stage.width 属性)

`public static width : Number`

属性（只读）：以像素为单位指示舞台的当前宽度。如果 `Stage.scaleMode` 的值为 `"noScale"`，则 `width` 属性表示 **Flash Player** 的宽度。这意味着 `Stage.width` 将随着您调整播放器窗口的大小而变化。如果 `Stage.scaleMode` 的值不为 `"noScale"`，则 `width` 表示进行创作时在“文本属性”对话框中设置的 **SWF** 文件的宽度。这意味着在您调整播放器窗口大小时，`width` 的值将保持不变。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

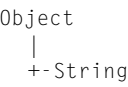
此示例创建一个名为 `stageListener` 的新侦听器对象。然后它使用 `stageListener` 来调用 `onResize`，并定义一个将在 `onResize` 受到激发时所调用的函数。最后，该代码将 `stageListener` 对象添加到 **Stage** 对象的回调列表中。侦听器对象允许多个对象侦听调整大小的通知。

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10,
    100, 22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
    stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

### 另请参见

[align \(Stage.align 属性\)](#)，[height \(Stage.height 属性\)](#)，[scaleMode \(Stage.scaleMode 属性\)](#)

# String



```
public class String
extends Object
```

**String** 类是字符串基元数据类型的包装，提供用于操作基元字符串值类型的方法和属性。您可以使用 `String()` 函数将任何对象的值转换为字符串。

**String** 类的所有方法（`concat()`、`fromCharCode()`、`slice()` 和 `substr()` 除外）都是通用方法，这意味着在对这些方法执行操作前，这些方法都将调用 `toString()`，并且可以将这些方法用于其它非 **String** 对象。

因为所有字符串索引都是从零开始的，所以任何字符串 `x` 的最后一个字符的索引都是 `x.length - 1`。

可以使用构造函数方法 `new String` 或者使用字符串值调用 **String** 类的任何方法。如果您指定了一个字符串，则 **ActionScript** 解释程序自动将其转换为一个临时 **String** 对象，再调用方法，然后放弃该临时 **String** 对象。您还可以将 `String.length` 属性与字符串一起使用。

请不要将字符串和 **String** 对象相混淆。在下面的示例中，代码的第一行创建字符串 `first_string`，代码的第二行创建 **String** 对象 `second_string`：

```
var first_string:String = "foo"
var second_string:String = new String("foo")
```

除非您确实需要使用 **String** 对象，否则请使用字符串。

可用性：ActionScript 1.0；Flash Player 5

## 属性摘要

修饰符	属性	说明
	<code>length:Number</code>	一个整数，它指定在所指定的 <b>String</b> 对象中的字符数。

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 构造函数摘要

签名	说明
<code>String(value:String)</code>	创建一个新 <b>String</b> 对象。

方法摘要

修饰符	签名	说明
	charAt(index:Number) : String	返回由参数 index 指定的位置处的字符。
	charCodeAt(index:Number) : Number	返回一个 0 到 65535 之间的 16 位整数，它表示由 index 指定的字符。
	concat(value:Object) : String	将 String 对象的值与参数合并，并返回新组成的字符串；而原始值 my_str 不变。
static	fromCharCode() : String	返回一个由参数中的 Unicode 值表示的字符组成的字符串。
	indexOf(value:String, [startIndex:Number]) : Number	搜索字符串，并返回在调用字符串内 startIndex 位置上或之后找到的 value 的第一个匹配项的位置。
	lastIndexOf(value:String, [startIndex:Number]) : Number	从右向左搜索字符串，并返回在调用字符串内 startIndex 之前找到的 value 的最后一个匹配项的索引。
	slice(start:Number, end:Number) : String	返回一个字符串，该字符串包括从 start 字符一直到 end 字符（但不包括该字符）之间的所有字符。
	split(delimiter:String, [limit:Number]) : Array	在指定的 delimiter 参数出现的所有位置断开 String 对象，将其拆分为子字符串，然后以数组形式返回子字符串。
	substr(start:Number, length:Number) : String	返回字符串中从 start 参数所指定的索引开始，直至 length 参数所指定的字符数为止的字符。
	substring(start:Number, end:Number) : String	返回一个字符串，该字符串由 start 和 end 参数指定的两点之间的字符组成。
	toLowerCase() : String	返回 String 对象的一个副本，其中的所有大写字符均转换为小写字符。
	toString() : String	无论对象的属性是否为字符串，均以字符串形式返回该属性。
	toUpperCase() : String	返回 String 对象的一个副本，其中的所有小写字符均转换为大写字符。
	valueOf() : String	返回字符串。



继承自 Object 类的方法

---

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPropertyEnumerable](#) ([Object.isPropertyEnumerable](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

---

## charAt (String.charAt 方法)

```
public charAt(index:Number) : String
```

返回由参数 `index` 指定的位置处的字符。如果 `index` 不是从 **0** 到 `string.length` 减 **1** 之间的数字, 则返回一个空字符串。

此方法与 `String.charCodeAt()` 类似, 所不同的是它返回的值是一个字符, 而不是 **16** 位整数字符代码。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**index:Number** - 一个整数, 指定字符在字符串中的位置。第一个字符由 **0** 指示, 最后一个字符由 `my_str.length-1` 指示。

### 返回

`String` - 指定索引处的字符。或者为一个空 `String` (如果指定的索引在此 `String` 的索引的范围之外)。

### 示例

在以下示例中, 此方法是在字符串 "Chris" 的首字母上调用的:

```
var my_str:String = "Chris";  
var firstChar_str:String = my_str.charAt(0);  
trace(firstChar_str); // output: C
```

### 另请参见

[charCodeAt](#) ([String.charCodeAt](#) 方法)

## charCodeAt (String.charCodeAt 方法)

`public charCodeAt(index:Number) : Number`

返回一个 0 到 65535 之间的 16 位整数，它表示由 `index` 指定的字符。如果 `index` 不是从 0 到 `string.length` 减 1 之间的数字，则返回 NaN。

此方法与 `String.charAt()` 类似，所不同的是它返回的值是 16 位整数字符代码，而不是字符。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**index:**Number - 一个整数，指定字符在字符串中的位置。第一个字符由 0 指示，最后一个字符由 `my_str.length` 减 1 指示。

### 返回

Number - 一个整数，表示由 `index` 指定的字符。

### 示例

在以下示例中，此方法是在字符串 "Chris" 的首字母上调用的：

```
var my_str:String = "Chris";  
var firstChar_num:Number = my_str.charCodeAt(0);  
trace(firstChar_num); // output: 67
```

### 另请参见

[charAt \(String.charAt 方法\)](#)

## concat (String.concat 方法)

`public concat(value:Object) : String`

将 **String** 对象的值与参数合并，并返回新组成的字符串；而原始值 `my_str` 不变。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**value:Object - value1[,...valueN]** 要连接的零个或多个值。

### 返回

String - 一个字符串。

## 示例

下面的示例创建两个字符串，并使用 `String.concat()` 将它们合并起来：

```
var stringA:String = "Hello";
var stringB:String = "World";
var combinedAB:String = stringA.concat(" ", stringB);
trace(combinedAB); // output: Hello World
```

## fromCharCode（String.fromCharCode 方法）

```
public static fromCharCode() : String
```

返回一个由参数中的 **Unicode** 值表示的字符组成的字符串。

可用性：ActionScript 1.0；Flash Player 5

## 返回

String - 指定的 **Unicode** 字符代码的字符串值。

## 示例

下面的示例使用 `fromCharCode()` 在电子邮件地址中插入一个 @ 字符：

```
var address_str:String = "dog"+String.fromCharCode(64)+"house.net";
trace(address_str); // output: dog@house.net
```

## indexOf（String.indexOf 方法）

```
public indexOf(value:String, [startIndex:Number]) : Number
```

搜索字符串，并返回在调用字符串内 `startIndex` 位置上或之后找到的 `value` 的第一个匹配项的位置。此索引从零开始，这意味着字符串中的第一个字符被视为位于索引 **0** 而不是索引 **1** 处。如果未找到 `value`，该方法会返回 **-1**。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**value**:String - 一个字符串；要搜索的子字符串。

**startIndex**:Number [可选] - 一个整数，指定搜索的开始索引。

## 返回

Number - 指定子字符串的第一个匹配项的位置，或 **-1**。

## 示例

下面的示例使用 `indexOf()` 返回字符和子字符串的索引：

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.indexOf("L");
trace(index); // output: 0

index = searchString.indexOf("l");
trace(index); // output: 14

index = searchString.indexOf("i");
trace(index); // output: 6

index = searchString.indexOf("ipsum");
trace(index); // output: 6

index = searchString.indexOf("i", 7);
trace(index); // output: 19

index = searchString.indexOf("z");
trace(index); // output: -1
```

另请参见

[lastIndexOf](#) ([String.lastIndexOf](#) 方法)

## lastIndexOf (String.lastIndexOf 方法)

```
public lastIndexOf(value:String, [startIndex:Number]) : Number
```

从右向左搜索字符串，并返回在调用字符串内 `startIndex` 之前找到的 `value` 的最后一个匹配项的索引。此索引从零开始，这意味着字符串中的第一个字符被视为位于索引 **0** 而不是索引 **1** 处。如果未找到 `value`，该方法会返回 **-1**。

可用性：ActionScript 1.0 ； Flash Player 5

### 参数

**value:String** - 要搜索的字符串。

**startIndex:Number** [ 可选 ] - 一个整数，指定搜索 `value` 的起始点。

### 返回

**Number** - 指定子字符串的最后一个匹配项的位置，或 **-1**。

## 示例

下面的示例说明如何使用 `lastIndexOf()` 返回某个字符的索引：

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.lastIndexOf("L");
trace(index); // output: 0

index = searchString.lastIndexOf("l");
trace(index); // output: 14

index = searchString.lastIndexOf("i");
trace(index); // output: 19

index = searchString.lastIndexOf("ipsum");
trace(index); // output: 6

index = searchString.lastIndexOf("i", 18);
trace(index); // output: 6

index = searchString.lastIndexOf("z");
trace(index); // output: -1
```

另请参见

[indexOf \(String.indexOf 方法\)](#)

## length (String.length 属性)

```
public length : Number
```

一个整数，它指定在所指定的 **String** 对象中的字符数。

因为所有字符串索引都是从零开始的，所以任何字符串 *x* 的最后一个字符的索引都是 *x.length - 1*。

可用性：ActionScript 1.0 ； Flash Player 5

## 示例

下面的示例创建一个新的 **String** 对象，并使用 `String.length` 统计字符数：

```
var my_str:String = "Hello world!";
trace(my_str.length); // output: 12
```

下面的示例从 0 循环到 `my_str.length`。该代码检查字符串内的字符，如果字符串包含字符 @，则 `true` 将显示在“输出”面板中。如果该字符串不包含字符 @，则 `false` 将显示在“输出”面板中。

```
function checkAtSymbol(my_str:String):Boolean {
    for (var i = 0; i<my_str.length; i++) {
        if (my_str.charAt(i) == "@") {
            return true;
        }
    }
    return false;
}
```

```
trace(checkAtSymbol("dog@house.net")); // output: true
trace(checkAtSymbol("Chris")); // output: false
```

**ActionScript** 示例文件夹的 **Strings.fla** 文件中也有一个示例。下面的列表指定到此文件夹的典型路径:

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples\ActionScript

## slice (String.slice 方法)

```
public slice(start:Number, end:Number) : String
```

返回一个字符串, 该字符串包括从 **start** 字符一直到 **end** 字符 (但不包括该字符) 之间的所有字符。不修改原始 **String** 对象。如果未指定 **end** 参数, 则子字符串的结尾就是该字符串的结尾。如果按 **start** 索引到的字符与按 **end** 索引到的字符相同或位于后者的右侧, 则该方法返回一个空字符串。

**可用性:** ActionScript 1.0 ; Flash Player 5

### 参数

**start:**Number - 片段起始点的从零开始的索引。如果 **start** 是一个负数, 则起始点从字符串的结尾开始确定, 其中 **-1** 表示最后一个字符。

**end:**Number - 一个比片段终点索引大 **1** 的整数。由 **end** 参数索引的字符未包括在已提取的字符串中。如果省略此参数, 则使用 **String.length**。如果 **end** 是一个负数, 则终点根据从字符串的结尾向后数确定, 其中 **-1** 表示最后一个字符。

### 返回

**String** - 指定字符串的子字符串。

## 示例

下面的示例创建一个变量 `my_str`，将 **String** 值赋予它，然后使用 `start` 和 `end` 参数的各种值调用 `slice()` 方法。对 `slice()` 的每个调用都包装在 `trace()` 语句中，该语句将输出显示在“输出”面板中。

```
// Index values for the string literal
// positive index: 0 1 2 3 4
// string: L o r e m
// negative index: -5 -4 -3 -2 -1

var my_str:String = "Lorem";

// slice the first character
trace("slice(0,1): "+my_str.slice(0, 1)); // output: slice(0,1): L
trace("slice(-5,1): "+my_str.slice(-5, 1)); // output: slice(-5,1): L

// slice the middle three characters
trace("slice(1,4): "+my_str.slice(1, 4)); // slice(1,4): ore
trace("slice(1,-1): "+my_str.slice(1, -1)); // slice(1,-1): ore

// slices that return empty strings because start is not to the left of end
trace("slice(1,1): "+my_str.slice(1, 1)); // slice(1,1):
trace("slice(3,2): "+my_str.slice(3, 2)); // slice(3,2):
trace("slice(-2,2): "+my_str.slice(-2, 2)); // slice(-2,2):

// slices that omit the end parameter use String.length, which equals 5
trace("slice(0): "+my_str.slice(0)); // slice(0): Lorem
trace("slice(3): "+my_str.slice(3)); // slice(3): em
```

**ActionScript** 示例文件夹的 **Strings.fla** 文件中也有一个示例。下面的列表指定到此文件夹的典型路径：

- Windows: 引导驱动器 \Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples\ActionScript

另请参见

[substr \(String.substr 方法\)](#)，[substring \(String.substring 方法\)](#)

## split (String.split 方法)

`public split(delimiter:String, [limit:Number]) : Array`

在指定的 `delimiter` 参数出现的所有位置断开 **String** 对象，将其拆分为子字符串，然后以数组形式返回子字符串。如果使用空字符串 ("" ) 作为分隔符，则该字符串中的每个字符都将作为一个元素放入到数组中。

如果未定义 `delimiter` 参数，则会将整个字符串放入返回的数组的第一个元素中。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**delimiter:String** - 一个字符串； `my_str` 拆分处的字符或字符串。

**limit:Number** [ 可选 ] - 要放入数组中的项目数。

### 返回

Array - 包含 `my_str` 的子字符串的数组。

### 示例

下面的示例返回含有五个元素的数组：

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",");
for (var i = 0; i<my_array.length; i++) {
    trace(my_array[i]);
}
// output:
P
A
T
S
Y
```

下面的示例返回含有两个元素 ("P" 和 "A") 的数组：

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",", 2);
trace(my_array); // output: P,A
```

下面的示例说明如果您将空字符串 ("" ) 用于 `delimiter` 参数，则将字符串中的每个字符作为元素放入数组：

```
var my_str:String = new String("Joe");
var my_array:Array = my_str.split("");
for (var i = 0; i<my_array.length; i++) {
    trace(my_array[i]);
}
```



```
// output:  
J  
o  
e
```

ActionScript 示例文件夹的 Strings.fla 文件中也有一个示例。下面的列表指定到此文件夹的典型路径:

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

另请参见

[join \(Array.join 方法\)](#)

## String 构造函数

```
public String(value:String)
```

创建一个新 String 对象。



因为与 String 对象相比, 使用字符串需要更低的资源, 而且通常更易于使用, 所以, 除非有充分的理由要使用 String 对象而不使用字符串, 否则您应该使用字符串而不使用 String 类的构造函数。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

**value:String** - 新 String 对象的初始值。

## substr (String.substr 方法)

```
public substr(start:Number, length:Number) : String
```

返回字符串中从 start 参数所指定的索引开始, 直至 length 参数所指定的字符数为止的字符。substr 方法不更改由 my\_str 指定的字符串; 它返回一个新字符串。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

**start:Number** - 一个整数, 指示 my\_str 中用于创建子字符串的第一个字符的位置。如果 start 为一个负数, 则起始位置从字符串的结尾开始确定, 其中 -1 表示最后一个字符。

**length:**Number - 要创建的子字符串中的字符数。如果没有指定 **length**，则子字符串包括从字符串开头到字符串结尾的所有字符。

## 返回

String - 指定字符串的子字符串。

## 示例

下面的示例创建一个新字符串 **my\_str**，并使用 **substr()** 返回该字符串中的第二个单词；首先，使用正的 **start** 参数，然后使用负的 **start** 参数：

```
var my_str:String = new String("Hello world");
var mySubstring:String = new String();
mySubstring = my_str.substr(6,5);
trace(mySubstring); // output: world
```

```
mySubstring = my_str.substr(-5,5);
trace(mySubstring); // output: world
```

**ActionScript** 示例文件夹的 **Strings fla** 文件中也有一个示例。下面的列表指定到此文件夹的典型路径：

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

## substring (String.substring 方法)

```
public substring(start:Number, end:Number) : String
```

返回一个字符串，该字符串由 **start** 和 **end** 参数指定的两点之间的字符组成。如果未指定 **end** 参数，则子字符串的结尾就是该字符串的结尾。如果 **start** 的值等于 **end** 的值，则该方法返回一个空字符串。如果 **start** 的值大于 **end** 的值，则在函数执行前两个参数将自动互换，且原始值不变。

可用性: **ActionScript 1.0 ; Flash Player 5**

## 参数

**start:**Number - 一个整数，指示 **my\_str** 的用于创建子字符串的第一个字符的位置。**start** 的有效值为 **0** 到 **String.length** 减 **1**。如果 **start** 为负值，则使用 **0**。

**end:**Number - 一个整数，值为 **my\_str** 中要提取的最后一个字符的索引加 **1**。**end** 的有效值为 **1** 到 **String.length**。由 **end** 参数索引的字符未包括在已提取的字符串中。如果省略此参数，则使用 **String.length**。如果此参数是一个负值，则使用 **0**。

## 返回

String - 指定字符串的子字符串。

## 示例

下面的示例说明如何使用 `substring()`：

```
var my_str:String = "Hello world";
var mySubString:String = my_str.substring(6,11);
trace(mySubString); // output: world
```

下面的示例说明在使用负的 `start` 参数时出现的情况：

```
var my_str:String = "Hello world";
var mySubString:String = my_str.substring(-5,5);
trace(mySubString); // output: Hello
```

**ActionScript** 示例文件夹的 **Strings fla** 文件中也有一个示例。下面的列表指定到此文件夹的典型路径：

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

## toLowerCase (String.toLowerCase 方法)

```
public toLowerCase() : String
```

返回 String 对象的一个副本，其中的所有大写字符均转换为小写字符。原始值不变。

可用性：ActionScript 1.0；Flash Player 5

## 返回

String - 一个字符串。

## 示例

下面的示例创建一个全部由大写字符组成的字符串，然后使用 `toLowerCase()` 将所有大写字符转换为小写字符来创建该字符串的副本：

```
var upperCase:String = "LOREM IPSUM DOLOR";
var lowerCase:String = upperCase.toLowerCase();
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
```

ActionScript 示例文件夹的 `Strings fla` 文件中也有一个示例。下面的列表指定到此文件夹的典型路径：

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

另请参见

[toUpperCase](#) ([String.toUpperCase](#) 方法)

## toString (String.toString 方法)

`public toString() : String`

无论对象的属性是否为字符串，均以字符串形式返回该属性。

可用性: ActionScript 1.0 ; Flash Player 5

### 返回

String - 字符串。

### 示例

下面的示例输出一个大写字符串，该字符串列出对象的所有属性，而无论该属性是否为字符串：

```
var employee:Object = new Object();
employee.name = "bob";
employee.salary = 60000;
employee.id = 284759021;

var employeeData:String = new String();
for (prop in employee)
{
    employeeData += employee[prop].toString().toUpperCase() + " ";
}
trace(employeeData);
```

如果此代码中不包括 `toString()` 方法，并且 `for` 循环中的行使用了 `employee[prop].toUpperCase()`，则输出将为 `"undefined undefined BOB"`。包括 `toString()` 方法将产生如下预期输出：`"284759021 60000 BOB"`。

## toUpperCase (String.toUpperCase 方法)

public toUpperCase() : String

返回 **String** 对象的一个副本，其中的所有小写字符均转换为大写字符。原始值不变。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

String - 一个字符串。

### 示例

下面的示例创建一个全部由小写字符组成的字符串，然后使用 toUpperCase() 创建该字符串的一个副本：

```
var lowerCase:String = "lorem ipsum dolor";
var upperCase:String = lowerCase.toUpperCase();
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
```

**ActionScript** 示例文件夹的 **Strings.fla** 文件中也能找到一个示例。下面的列表指定到此文件夹的典型路径：

- Windows: 引导驱动器 \Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples\ActionScript

### 另请参见

[toLowerCase \(String.toLowerCase 方法\)](#)

## valueOf (String.valueOf 方法)

public valueOf() : String

返回字符串。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 返回

String - 字符串的值。

示例

下面的示例创建 `String` 对象的一个新实例，然后说明 `valueOf` 方法返回对原始 值的一个引用，而不是此对象的一个实例。

```
var str:String = new String("Hello World");
var value:String = str.valueOf();
trace(str instanceof String); // true
trace(value instanceof String); // false
trace(str === value); // false
```

# StyleSheet (TextField.StyleSheet)



```
public class StyleSheet
extends Object
```

使用 `StyleSheet` 类可以创建包含文本格式设置规则（例如，字体大小、颜色和其它格式样式）的 `StyleSheet` 对象。然后，可以将样式表定义的样式应用到包含 `HTML` 或 `XML` 格式文本的 `TextField` 对象。根据 `StyleSheet` 对象定义的标签样式自动设置 `TextField` 对象中文本的格式。可以使用文本样式来定义新的格式标签，重新定义内置的 `HTML` 标签，或创建可应用到某些 `HTML` 标签的样式类。

若要对一个 `TextField` 对象应用样式，请将该 `StyleSheet` 对象分配给 `TextField` 对象的 `styleSheet` 属性。

Flash Player 支持原 `CSS1` 规范 ([www.w3.org/TR/REC-CSS1](http://www.w3.org/TR/REC-CSS1)) 中的一部分属性。下表显示受支持的层叠样式表 (`CSS`) 属性和值，以及它们对应的 `ActionScript` 属性名称。（每个 `ActionScript` 属性名称都是从对应的 `CSS` 属性名称派生的；如果名称中包含连字符，请省略连字符并将连字符后的字符变成大写。）

CSS 属性	ActionScript 属性	用法和支持的值
color	color	只支持十六进制颜色值。不支持具有指定名称的颜色（例如 blue）。颜色以下面的格式写入： #FF0000。
display	display	受支持的值为 inline、block 和 none。
font-family	fontFamily	用逗号分隔的供使用字体的列表，按需求降序排列。可以使用任何字体系列名称。如果您指定通用字体名称，请它将转换为相应的设备字体。支持以下字体转换：mono 转换为 _typewriter，sans-serif 转换为 _sans，并且 serif 转换为 _serif。

CSS 属性	ActionScript 属性	用法和支持的值
font-size	fontSize	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
font-style	fontStyle	可识别的值为 normal 和 italic。
font-weight	fontWeight	可识别的值为 normal 和 bold。
kerning	kerning	可识别的值为 true 和 false。仅嵌入字体支持字距调整。某些字体（如 Courier New）不支持字距调整。只有 Windows 中创建的 SWF 文件支持 kerning 属性，而 Macintosh 中创建的 SWF 文件不支持该属性。但是，这些 SWF 文件可以在 Flash Player 的非 Windows 版本中播放，并且仍可以应用字距调整。
letter-spacing	letterSpacing	两个字符之间统一分布的距离。该值指定在每个字符之后添加到进距的像素数。负值将压缩两个字符之间的距离。只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
margin-left	marginLeft	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
margin-right	marginRight	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
text-align	textAlign	可识别的值为 left、center、right 和 justify。
text-decoration	textDecoration	可识别的值为 none 和 underline。
text-indent	textIndent	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。

可用性：ActionScript 1.0；Flash Player 7

### 属性摘要

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

### 事件摘要

事件	说明
onload = function(success:Boolean) {}	当已完成 load() 操作时调用。

构造函数摘要

签名	说明
StyleSheet()	创建 StyleSheet 对象。

方法摘要

修饰符	签名	说明
	clear() : Void	从指定的 StyleSheet 对象中删除所有样式。
	getStyle(name:String) : Object	返回与指定样式 (name) 相关联的样式对象的一个副本。
	getStyleNames() : Array	返回一个数组，其中包含此样式表中注册的所有样式的名称（字符串形式）。
	load(url:String) : Boolean	开始将 CSS 文件加载到样式表中。
	parseCSS(cssText:String) : Boolean	分析 cssText 中的 CSS 并用它加载样式表。
	setStyle(name:String, style:Object) : Void	将具有指定名称的新样式添加到 StyleSheet 对象。
	transform(style:Object) : TextFormat	扩展 CSS 分析功能。

继承自 Object 类的方法

<a href="#">addProperty (Object.addProperty 方法)</a> , <a href="#">hasOwnProperty (Object.hasOwnProperty 方法)</a> , <a href="#">isPrototypeOf (Object.isPrototypeOf 方法)</a> , <a href="#">isPrototypeOf (Object.isPrototypeOfOf 方法)</a> , <a href="#">registerClass (Object.registerClass 方法)</a> , <a href="#">toString (Object.toString 方法)</a> , <a href="#">unwatch (Object.unwatch 方法)</a> , <a href="#">valueOf (Object.valueOf 方法)</a> , <a href="#">watch (Object.watch 方法)</a>
---



## clear (StyleSheet.clear 方法)

public clear() : Void

从指定的 **StyleSheet** 对象中删除所有样式。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 示例

下面的示例将名为 **styles.css** 的样式表加载到 **SWF** 文件中，并将已加载的样式显示在“输出”面板中。当单击 **clear\_btn** 时，将从 **my\_styleSheet** 对象中删除所有样式。

```
// Create a new StyleSheet object
import TextField.StyleSheet;
var my_styleSheet:StyleSheet = new StyleSheet();

my_styleSheet.onLoad = function(success:Boolean) {
    if (success) {
        trace("Styles loaded.");
        var styles_array:Array = my_styleSheet.getStyleNames();
        for (var i = 0; i < styles_array.length; i++) {
            trace("\t"+styles_array[i]);
        }
        trace("");
    } else {
        trace("Error loading CSS");
    }
};

// Start the loading operation
my_styleSheet.load("styles.css");

clear_btn.onRelease = function() {
    my_styleSheet.clear();
    trace("Styles cleared.");
    var styles_array:Array = my_styleSheet.getStyleNames();
    for (var i = 0; i < styles_array.length; i++) {
        trace("\t"+styles_array[i]);
    }
    trace("");
};
```

## getStyle (StyleSheet.getStyle 方法)

public getStyle(name:String) : Object

返回与指定样式 (name) 相关联的样式对象的一个副本。如果没有与 name 相关联的样式对象，则此方法返回 null。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**name:String** - 要检索的样式的名称。

### 返回

Object - 一个样式对象；否则为 null。

### 示例

下面的示例从一个 CSS 文件加载样式，分析样式表并将样式名和属性值显示在“输出”面板中。创建名为 **StyleSheetTracer.as** 的 **ActionScript** 文件，并在该文件中输入以下代码：

```
import TextField.StyleSheet;
class StyleSheetTracer {
    // StyleSheetTracer.displayFromURL
    // This method displays the CSS style sheet at
    // the specified URL in the Output panel.
    static function displayFromURL(url:String):Void {
        // Create a new StyleSheet object
        var my_styleSheet:StyleSheet = new StyleSheet();
        // The load operation is asynchronous, so set up
        // a callback function to display the loaded StyleSheet.
        my_styleSheet.onLoad = function(success:Boolean) {
            if (success) {
                StyleSheetTracer.display(this);
            } else {
                trace("Error loading style sheet "+url);
            }
        };
        // Start the loading operation.
        my_styleSheet.load(url);
    }
    static function display(my_styleSheet:StyleSheet):Void {
        var styleNames:Array = my_styleSheet.getStyleNames();
        if (!styleNames.length) {
            trace("This is an empty style sheet.");
        } else {
            for (var i = 0; i<styleNames.length; i++) {
                var styleName:String = styleNames[i];
                trace("Style "+styleName+":");
                var styleObject:Object = my_styleSheet.getStyle(styleName);
```

```

        for (var propName in styleObject) {
            var propValue = styleObject[propName];
            trace("\t"+propName+": "+propValue);
        }
        trace("");
    }
}
}
}

```

创建名为 **styles.css** 的 CSS 文档，该文档具有名为 `.heading` 和 `.mainBody` 和两个样式，它们定义了 `font-family`、`font-size` 和 `font-weight` 的属性。在 CSS 文档中输入以下代码：

```

/~ In styles.css ~/
.heading {
font-family: Arial, Helvetica, sans-serif;
font-size: 24px;
font-weight: bold;
}
.mainBody {
font-family: Arial, Helvetica, sans-serif;
font-size: 12px;
font-weight: normal;
}

```

最后，在 **FLA** 或 **ActionScript** 文件中，输入以下 **ActionScript** 以加载外部样式表 **styles.css**：

```

StyleSheetTracer.displayFromURL("styles.css");

```

这样就会将以下信息显示在“输出”面板中：

```

Style .heading:
fontWeight: bold
fontSize: 24px
fontFamily: Arial, Helvetica, sans-serif

Style .mainBody:
fontWeight: normal
fontSize: 12px
fontFamily: Arial, Helvetica, sans-serif

```

另请参见

[setStyle \(StyleSheet.setStyle 方法\)](#)

## getStyleNames (StyleSheet.getStyleNames 方法)

public getStyleNames() : Array

返回一个数组，其中包含此样式表中注册的所有样式的名称（字符串形式）。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 返回

Array - 样式名的数组（字符串形式）。

### 示例

此示例创建名为 `styleSheet` 的 **StyleSheet** 对象，该对象包含 `heading` 和 `bodyText` 两个样式。然后调用该 **StyleSheet** 对象的 `getStyleNames()` 方法，将结果赋予数组 `names_array`，并在“输出”面板中显示该数组的内容。

```
import TextField.StyleSheet;
var my_styleSheet:StyleSheet = new StyleSheet();
my_styleSheet.setStyle("heading", {fontsize:'24px'});
my_styleSheet.setStyle("bodyText", {fontsize:'12px'});
var names_array:Array = my_styleSheet.getStyleNames();
trace(names_array.join("\n"));
```

在“输出”面板中显示下面的信息：

```
bodyText
heading
```

### 另请参见

[getStyle \(StyleSheet.getStyle 方法\)](#)

## load (StyleSheet.load 方法)

public load(url:String) : Boolean

开始将 CSS 文件加载到样式表中。加载操作是异步进行的；使用 onLoad() 回调处理函数可确定该文件何时加载完毕。CSS 文件驻留的域必须与加载它的 SWF 文件的域相同。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

url:String - 要加载的 CSS 文件的 URL。该 URL 必须与 SWF 文件当前位于的 URL 在同一个域中。

### 返回

Boolean - 如果没有传递参数 (null), 则为 false ; 否则为 true。请使用 onLoad() 回调处理函数检查加载样式表是否成功。

### 示例

有关使用 **ActionScript 2.0** 来异步加载样式表的示例, 请参见 `getStyle()` 的示例。

以下示例将名为 **styles.css** 的 CSS 文件加载到 **StyleSheet** 对象 `my_styleSheet` 中。当该文件已成功完成加载时, **StyleSheet** 对象将应用于名为 `news_txt` 的 **TextField** 对象。

```
import TextField.StyleSheet;
this.createTextField("news_txt", 999, 10, 10, 320, 240);
news_txt.multiline = true;
news_txt.wordWrap = true;
news_txt.html = true;

var my_styleSheet:StyleSheet = new StyleSheet();
my_styleSheet.onLoad = function(success:Boolean) {
    if (success) {
        news_txt.styleSheet = my_styleSheet;
        news_txt.htmlText = "<p class=\"heading\">Heading goes here!</p>"
            + "<p class=\"mainBody\">Lorem ipsum dolor sit amet, consectetur "
            + "adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet "
            + "dolore magna aliquam erat volutpat.</p>";
    }
};
my_styleSheet.load("styles.css");
```

有关 **styles.css** 的完整代码, 请参见 `getStyle()` 的示例。

### 另请参见

[onLoad \(StyleSheet.onLoad 处理函数\)](#), [getStyle \(StyleSheet.getStyle 方法\)](#)

## onLoad (StyleSheet.onLoad 处理函数)

```
onLoad = function(success:Boolean) {}
```

当已完成 load() 操作时调用。如果样式表加载成功，则 success 参数为 true。如果未收到该文档，或从服务器接收响应时出现错误，则 success 参数为 false。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**success:Boolean** - 一个布尔值，指示 CSS 文件加载是成功 (true) 还是失败 (false)。

### 示例

以下示例将名为 **styles.css** 的 CSS 文件加载到 **StyleSheet** 对象 my\_styleSheet 中。当文件已成功完成加载时，**StyleSheet** 对象将应用于名为 news\_txt 的 **TextField** 对象。

```
import TextField.StyleSheet;
this.createTextField("news_txt", 999, 10, 10, 320, 240);
news_txt.multiline = true;
news_txt.wordWrap = true;
news_txt.html = true;

var my_styleSheet:StyleSheet = new StyleSheet();
my_styleSheet.onLoad = function(success:Boolean) {
    if (success) {
        news_txt.styleSheet = my_styleSheet;
        news_txt.htmlText = "<p class=\"heading\">Heading goes here!"
        + "</p><p class=\"mainBody\">Lorem ipsum dolor "
        + "sit amet, consectetur adipiscing elit, sed diam nonummy "
        + "nibh euismod tincidunt ut laoreet dolore magna aliquam "
        + "erat volutpat.</p>";
    }
};
my_styleSheet.load("styles.css");
```

有关 **styles.css** 的完整代码，请参见 `getStyle()` 的示例。有关使用 **ActionScript 2.0** 来异步加载样式表的示例，请参见 `getStyle()` 的示例。

### 另请参见

[load \(StyleSheet.load 方法\)](#), [getStyle \(StyleSheet.getStyle 方法\)](#)

## parseCSS (StyleSheet.parseCSS 方法)

public parseCSS(cssText:String) : Boolean

分析 cssText 中的 CSS 并用它加载样式表。如果样式表中已存在 cssText 中的某一样式，则样式表保留其属性，并且对于仅在 cssText 中存在的样式，则样式表会进行添加或更改。若要扩展本机 CSS 分析功能，可通过创建 **StyleSheet** 类的子类来覆盖此方法。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**cssText:String** - 要分析的 CSS 文本。

### 返回

Boolean - 一个布尔值，指示文本分析是成功 (true) 还是失败 (false)。

### 示例

下面的示例分析 css\_str 中的 CSS。脚本显示有关它是否成功分析 CSS 的信息，然后将已分析的 CSS 显示在“输出”面板中。

```
import TextField.StyleSheet;
var css_str:String = ".heading {font-family: Arial, Helvetica, sans-serif;
    font-size: 24px; font-weight: bold; }";
var my_styleSheet:StyleSheet = new StyleSheet();
if (my_styleSheet.parseCSS(css_str)) {
    trace("parsed successfully");
    dumpStyles(my_styleSheet);
} else {
    trace("unable to parse CSS");
}
//
function dumpStyles(styles:StyleSheet):Void {
    var styleNames_array:Array = styles.getStyleNames();
    for (var i = 0; i<styleNames_array.length; i++) {
        var styleName_str:String = styleNames_array[i];
        var styleObject:Object = styles.getStyle(styleName_str);
        trace(styleName_str);
        for (var prop in styleObject) {
            trace("\t"+prop+": "+styleObject[prop]);
        }
        trace("");
    }
}
```

## setStyle (StyleSheet.setStyle 方法)

```
public setStyle(name:String, style:Object) : Void
```

将具有指定名称的新样式添加到 **StyleSheet** 对象。如果该样式表中没有具有指定名称的样式，将添加该样式。如果该样式表中已经具有指定名称的样式，将替换该样式。如果 `style` 参数为 `null`，则删除指定名称的样式。

**Flash Player** 将创建传递给此方法的样式对象的一个副本。

要获取所支持样式的列表，请参见 **StyleSheet** 类说明中的表。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**name:String** - 要添加到样式表中的样式名。

**style:Object** - 一个说明样式的对象，或 `null`。

### 示例

下面的示例将名为 `emphasized` 的样式添加到样式表 `myStyleSheet`。该样式包含两个样式属性: `color` 和 `fontWeight`。该样式对象是用 `{}` 运算符定义的。

```
myStyleSheet.setStyle("emphasized", {color:'#000000',fontWeight:'bold'});
```

您也可以使用 **Object** 类的一个实例来创建样式对象，然后作为 `style` 参数传递该对象 (`styleObj`)，如下一个示例所示：

```
import TextField.StyleSheet;
var my_styleSheet:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.color = "#000000";
styleObj.fontWeight = "bold";
my_styleSheet.setStyle("emphasized", styleObj);
delete styleObj;

var styleNames_array:Array = my_styleSheet.getStyleNames();
for (var i=0;i<styleNames_array.length;i++) {
    var styleName:String = styleNames_array[i];
    var thisStyle:Object = my_styleSheet.getStyle(styleName);
    trace(styleName);
    for (var prop in thisStyle) {
        trace("\t"+prop+": "+thisStyle[prop]);
    }
    trace("");
}
```



在“输出”面板中显示下面的信息：

```
emphasized  
fontWeight: bold  
color: #000000
```



由于 Flash Player 会创建您传递给 `setStyle()` 的样式对象的一个副本，所以代码示例中的 `delete styleObj` 命令将通过删除传递给 `setStyle()` 的原始样式对象来减少内存使用。

另请参见

[{} 对象初始值设定项运算符](#)，[StyleSheet \(TextField.StyleSheet\)](#)

## StyleSheet 构造函数

```
public StyleSheet()
```

创建 `StyleSheet` 对象。

可用性：ActionScript 1.0；Flash Player 7

示例

下面的示例加载一个样式表，并跟踪加载到文档中的样式。将以下 **ActionScript** 添加到您的 **ActionScript** 或 **FLA** 文件：

```
import TextField.StyleSheet;  
var my_styleSheet:StyleSheet = new StyleSheet();  
my_styleSheet.onLoad = function(success:Boolean) {  
    if (success) {  
        trace("Styles loaded:");  
        var styles_array:Array = my_styleSheet.getStyleNames();  
        trace(styles_array.join(newline));  
    } else {  
        trace("Error loading CSS");  
    }  
};  
my_styleSheet.load("styles.css");
```

**styles.css** 文件包含名为 `.heading` 和 `.mainbody` 的两个样式，所以“输出”面板中会显示以下信息：

```
Styles loaded:  
.heading  
.mainBody
```

在 `getStyle()` 的示例中有 **styles.css** 的完整代码。

另请参见

[getStyle \(StyleSheet.getStyle 方法\)](#)

## transform (StyleSheet.transform 方法)

`public transform(style:Object) : TextFormat`

扩展 CSS 分析功能。高级开发人员可通过扩展 `StyleSheet` 类覆盖此方法。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**style:Object** - 一个说明样式的对象 (包含作为对象属性的样式规则), 或 `null`。

### 返回

`TextFormat` - 包含 CSS 规则向文本格式属性映射的结果的一个 `TextFormat` 对象。

### 示例

下面的示例扩展 `transform()` 方法:

```
import TextField.StyleSheet;
class AdvancedCSS extends StyleSheet {
    public function AdvancedCSS() {
        trace("AdvancedCSS instantiated");
    }

    public function transform(styleObject):TextFormat {
        trace("transform called");
    }
}
```

## System

Object  
|  
+-System

```
public class System
extends Object
```

**System** 类包含与发生在用户计算机上的某些操作相关的属性, 如具有共享对象的操作、摄像头和麦克风的本地设置和剪贴板的使用。下列附加属性和方法包含在 **System** 包内的特定类中: **capabilities** 类、**security** 类和 **IME** 类。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 另请参见

[capabilities \(System.capabilities\)](#), [security \(System.security\)](#), [IME \(System.IME\)](#)

属性摘要

修饰符	属性	说明
static	exactSettings:Boolean	一个布尔值，指定当访问本地设置（例如，摄像头或麦克风访问权限）或本地永久数据（共享对象）时是使用超级域匹配规则 (false) 还是使用精确域匹配规则 (true)。
static	useCodepage:Boolean	一个布尔值，它通知 Flash Player 是使用 Unicode 来解释外部文本文件，还是使用运行播放器的操作系统的传统代码页来解释外部文本文件。

继承自 Object 类的属性

<a href="#">constructor</a> (Object.constructor 属性), <a href="#">__proto__</a> (Object.__proto__ 属性), <a href="#">prototype</a> (Object.prototype 属性), <a href="#">__resolve</a> (Object.__resolve 属性)
--

事件摘要

事件	说明
<code>onStatus = function(infoObject:Object) {}</code>	事件处理函数：为特定对象提供超级事件处理函数。

方法摘要

修饰符	签名	说明
static	setClipboard(text:String) : Void	用指定的文本字符串替换剪贴板的内容。
static	showSettings([tabID:Number]) : Void	显示指定的 Flash Player “设置” 面板。

继承自 Object 类的方法

<a href="#">addProperty</a> (Object.addProperty 方法), <a href="#">hasOwnProperty</a> (Object.hasOwnProperty 方法), <a href="#">isPropertyEnumerable</a> (Object.isPropertyEnumerable 方法), <a href="#">isPrototypeOf</a> (Object.isPrototypeOf 方法), <a href="#">registerClass</a> (Object.registerClass 方法), <a href="#">toString</a> (Object.toString 方法), <a href="#">unwatch</a> (Object.unwatch 方法), <a href="#">valueOf</a> (Object.valueOf 方法), <a href="#">watch</a> (Object.watch 方法)
---

## exactSettings (System.exactSettings 属性)

`public static exactSettings : Boolean`

一个布尔值，指定当访问本地设置（例如，摄像头或麦克风访问权限）或本地永久数据（共享对象）时是使用超级域匹配规则 (`false`) 还是使用精确域匹配规则 (`true`)。对于为 **Flash Player 7** 或更高版本发布的文件，默认值为 `true`；对于为 **Flash Player 6** 发布的文件，默认值为 `false`。

如果此值为 `true`，则承载于 `here.xyz.com` 的 SWF 文件的设置和数据将存储在名为 `here.xyz.com` 的目录中，承载于 `there.xyz.com` 的 SWF 文件的设置和数据将存储在名为 `there.xyz.com` 的目录中，依此类推。如果此值为 `false`，则承载于 `here.xyz.com`、`there.xyz.com` 和 `xyz.com` 的 SWF 文件的设置和数据是共享的，且都存储在名为 `xyz.com` 的目录中。

如果某些文件将此属性设置为 `false`，而其它文件将其设置为 `true`，则将出现不同子域中的 SWF 文件共享设置和数据的情况。例如，如果在承载于 `here.xyz.com` 的某个 SWF 文件中此属性为 `false`，而在承载于 `xyz.com` 的某个 SWF 文件中此属性为 `true`，则这两个文件都将使用相同的设置和数据，即 `xyz.com` 目录中的设置和数据。如果您不希望出现这种行为，请确保您在每个文件中设置此属性，以便正确表示希望将设置和数据存储在何处。

如果您希望更改此属性的默认值，请在文档的第一帧中执行此操作。如果您希望更改此属性的默认值，请在脚本起始处附近执行此操作。无法在需要访问本地设置（例如 `System.showSettings()` 或 `SharedObject.getLocal()`）的任何活动后更改此属性。

如果使用 `loadMovie()`、`MovieClip.loadMovie()` 或 `MovieClipLoader.loadClip()` 将一个 SWF 文件加载到另一个 SWF 文件中，则为 **Flash Player 7** 发布的所有文件共享 `System.exactSettings` 的单个值，为 **Flash Player 6** 发布的所有文件共享 `System.exactSettings` 的单个值。如果您使用 `MovieClip.loadMovie()` 或 `MovieClipLoader.loadClip()` 将一个 SWF 文件加载到另一个 SWF 文件中，则所有文件都共享 `System.exactSettings` 的单个值。因此，如果在一个为某个特定 **Player** 版本发布的文件中指定此属性的值，则应该在要加载的所有文件中执行此操作。如果加载多个文件，则在最后一个加载的文件中指定的设置将覆盖所有以前指定的设置。

通常，您会发现 `System.exactSettings` 的默认值非常有用。通常，您的唯一要求就是当 SWF 文件在会话中保存共享对象时，同一个 SWF 文件可以在以后的会话中检索同一个共享对象。无论 `System.exactSettings` 的值如何，这一情况始终发生。但您可能希望更改 `System.exactSettings` 的默认值，以便为 Flash Player 7 或更高版本发布的 SWF 文件能够检索原来为 Flash Player 6 发布的 SWF 文件创建的共享对象。由于播放器已将由 Flash Player 6 SWF 文件创建的共享对象存储在特定于该 SWF 文件所在的超级域的文件夹中，因此您应该在 Flash Player 7 SWF 文件中使用超级域规则来检索共享对象。此步骤需要在 Flash Player 7 SWF 文件中指定 `System.exactSettings = false`。也可以使为 Flash Player 6 发布的 SWF 文件和 Flash Player 7 SWF 文件共享相同的共享对象数据。在这种情况下，只要为 `System.exactSettings` 选择一个值（true 或 false），并在 Flash Player 6 SWF 文件和 Flash Player 7 SWF 文件中始终使用此值即可。

可用性：ActionScript 1.0；Flash Player 7

## 示例

下面的示例演示如何指定与规则匹配的超级域：

另请参见

`loadMovie` ([MovieClip.loadMovie 方法](#))，`loadClip` ([MovieClipLoader.loadClip 方法](#))，`getLocal` ([SharedObject.getLocal 方法](#))，`exactSettings` ([System.exactSettings 属性](#))

## onStatus (System.onStatus 处理函数)

```
onStatus = function(info:Object){}
```

事件处理函数：为特定对象提供超级事件处理函数。

`LocalConnection`、`NetStream` 和 `SharedObject` 类提供 `onStatus` 事件处理函数，该事件处理函数使用信息对象提供信息、状态或错误消息。若要响应此事件处理函数，您必须创建一个函数来处理信息对象，同时还必须了解所返回的信息对象的格式和内容。

除这些特定的 `onStatus` 方法外，Flash 还提供了一个名为 `System.onStatus` 的超级函数，它作为辅助错误消息处理函数。如果 `LocalConnection`、`NetStream` 或 `SharedObject` 类的实例传递一个级别属性为“error”的信息对象，但您没有为该特定实例定义 `onStatus` 函数，则 Flash 会改用您为 `System.onStatus` 定义的函数。



`Camera` 和 `Microphone` 类也具有 `onStatus` 处理函数，但不传递级别属性为“error”的信息对象。因此，如果不为这些处理函数指定函数，将不会调用 `System.onStatus`。

可用性：ActionScript 1.0；Flash Player 6

## 参数

`infoObject:Object` - 根据状态消息定义的参数。

## 示例

下面的示例演示如何创建一个 `System.onStatus` 函数，以便在类特定的 `onStatus` 函数不存在时处理信息对象：

```
// Create generic function
System.onStatus = function(genericError:Object){
    // Your script would do something more meaningful here
    trace("An error has occurred. Please try again.");
}
```

下面的示例演示如何为 `NetStream` 类的实例创建一个 `onStatus` 函数：

```
// Create function for NetStream object

videoStream_ns.onStatus = function(infoObject:Object) {
    if (infoObject.code == "NetStream.Play.StreamNotFound") {
        trace("Could not find video file.");
    }
}
```

## 另请参见

[onStatus \(Camera.onStatus 处理函数\)](#), [onStatus \(LocalConnection.onStatus 处理函数\)](#), [onStatus \(Microphone.onStatus 处理函数\)](#), [onStatus \(NetStream.onStatus 处理函数\)](#), [onStatus \(SharedObject.onStatus 处理函数\)](#)

# setClipboard (System.setClipboard 方法)

```
public static setClipboard(text:String) : Void
```

用指定的文本字符串替换剪贴板的内容。



出于安全方面的考虑，您不能读取系统剪贴板的内容。换句话说，不存在相应的 `System.getClipboard()` 方法。

可用性：ActionScript 1.0；Flash Player 7

## 参数

`text:String` - 要放置在系统剪贴板上的纯文本字符串，替换系统剪贴板上的当前内容（如果有的话）。

## 示例

下面的示例将短语 "Hello World" 放置到系统剪贴板上：

```
System.setClipboard("Hello world");
```

下面的示例在运行时创建名为 `in_txt` 和 `out_txt` 的两个文本字段。当在 `in_txt` 字段中选择文本时，您可以单击 `copy_btn` 将数据复制到剪贴板上。然后，可以将该文本粘贴到 `out_txt` 字段中。

```
this.createTextField("in_txt", this.getNextHighestDepth(), 10, 10, 160,
    120);
in_txt.multiline = true;
in_txt.border = true;
in_txt.text = "lorum ipsum...";
this.createTextField("out_txt", this.getNextHighestDepth(), 10, 140, 160,
    120);
out_txt.multiline = true;
out_txt.border = true;
out_txt.type = "input";

copy_btn.onRelease = function() {
    System.setClipboard(in_txt.text);
    Selection.setFocus("out_txt");
};
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## showSettings（System.showSettings 方法）

```
public static showSettings([tabID:Number]) : Void
```

显示指定的 **Flash Player** “设置”面板。该面板允许用户执行以下任意操作：

- 允许或拒绝对摄像头和麦克风的访问
- 指定可供共享对象使用的本地磁盘空间
- 选择默认摄像头和麦克风
- 指定麦克风增益和回声抑制设置

例如，如果您的应用程序需要使用摄像头，则可以通知用户在“隐私设置”面板中选择“允许”，然后发出 `System.showSettings(0)` 命令。（请确保“舞台”大小至少为 215 x 138 像素）

可用性：ActionScript 1.0；Flash Player 6

参数

**tabID:**Number [ 可选 ] - 一个数字；指定要显示哪个 Flash Player “设置”面板的数字，如下表所示：

为 panel 传递的值	显示的 “设置” 面板
无 （参数被忽略）或不支持的值	用户上次关闭 Player “设置” 面板时，处于打开状态的那个面板。
0	隐私
1	本地存储
2	Microphone
3	Camera

示例

下面的示例演示如何显示 Flash Player “本地存储设置” 面板：

```
System.showSetting(1)
```

另请参见

[get \(Camera.get 方法\)](#) , [get \(Microphone.get 方法\)](#) , [getLocal \(SharedObject.getLocal 方法\)](#)

# useCodepage （System.useCodepage 属性）

```
public static useCodepage : Boolean
```

一个布尔值，它通知 Flash Player 是使用 Unicode 来解释外部文本文件，还是使用运行播放器的操作系统的传统代码页来解释外部文本文件。System.useCodepage 的默认值为 **false**。

- 当该属性设置为 **false** 时，Flash Player 按 Unicode 解释外部文本文件。（保存这些文件时，必须使用 Unicode 对其进行编码。）
- 当属性设置为 **true** 时，Flash Player 使用运行播放器的操作系统的传统代码页来解释外部文本文件。

在您保存文本文件时，作为外部文件加载（使用 loadVariables() 或 getURL() 语句，或使用 LoadVars 类或 XML 类）的文本必须按 Unicode 进行编码，以使 Flash Player 能够按 Unicode 识别它。若要使用 Unicode 对外部文件进行编码，请在支持 Unicode 的应用程序（例如，Windows 2000 上的“记事本”）中保存这些文件。

如果加载的外部文本文件没有按 Unicode 进行编码，则应将 System.useCodepage 设置为 **true**。在加载数据的 SWF 文件的第一帧中，在最前面添加以下代码（第一行代码）：

```
System.useCodepage = true;
```



如果有这一行代码，Flash Player 将使用运行 Flash Player 的操作系统的传统代码页来解释外部文本。对于英文 Windows 操作系统，该代码页通常为 CP1252；对于日文操作系统，该代码页通常为 Shift-JIS。如果将 System.useCodepage 设置为 true，则 Flash Player 6 和更高版本处理文本的方式与 Flash Player 5 相同。（Flash Player 5 将所有文本都视为按运行播放器的操作系统的传统代码页编码的文本。）

如果将 System.useCodepage 设置为 true，应注意在运行播放器的操作系统的传统代码页中必须包括您的外部文本文件中使用的字符，这样才能显示文本。例如，如果您加载了一个包含中文字符的外部文本文件，则这些字符不能显示在使用 CP1252 代码页的系统上，因为该代码页不包括中文字符。

若要确保所有平台上的用户都能查看您的 SWF 文件中使用的文本文件，应将所有外部文本文件按 Unicode 进行编码，并将 System.useCodepage 设置保留为其默认设置 false。这样，Flash Player 6 和更高版本将按 Unicode 解释文本。

可用性：ActionScript 1.0；Flash Player 6

## TextField

```
Object
|
+-TextField
```

```
public dynamic class TextField
extends Object
```

**TextField** 类用于创建区域以供文本显示和输入。SWF 文件中的所有动态文本字段和输入文本字段都是 **TextField** 类的实例。可以在属性检查器中为文本字段指定实例名称，并且可以在 **ActionScript** 中使用 **TextField** 类的方法和属性对文本字段进行操作。**TextField** 的实例名称显示在“影片管理器”中及“动作”面板的“插入目标路径”对话框中。

若要动态创建文本字段，您不必使用 new 运算符。而改用 `MovieClip.createTextField()`。

**TextField** 类的方法允许您设置、选择并操控在创作过程中或运行时创建的动态或输入文本字段中的文本。

**ActionScript** 提供了多种在运行时设置文本格式的方法。**TextFormat** 类允许您设置 **TextField** 对象的字符和段落格式。在 **Flash Player 7** 和更高版本中，您可以使用 **TextField.styleSheet** 属性和 **StyleSheet** 类将层叠样式表 (CSS) 样式应用于文本字段。您可以使用 CSS 设置内置 HTML 标签的样式、定义新的格式设置标签或应用样式。您可以将 HTML 格式的文本（该文本可以选择使用 CSS 样式）直接分配给文本字段。在 **Flash Player 7** 和更高版本中，分配给文本字段的 HTML 文本可以包含嵌入的媒体（影片剪辑、SWF 文件、JPEG 文件、GIF 文件和 PNG 文件）。文本在嵌入的媒体旁自动换行，这非常类似于 HTML 页中的文本在嵌入的媒体旁自动换行。

**Flash Player** 还支持部分 HTML 标签，可以使用这些 HTML 标签设置文本格式。

可用性: ActionScript 1.0 ; Flash Player 6

另请参见

[Object](#), [createTextField](#) ([MovieClip.createTextField](#) 方法)

属性摘要

修饰符	属性	说明
	<code>_alpha:Number</code>	设置或检索文本字段的 Alpha 透明度值。
	<code>antiAliasType:String</code>	用于此 <code>TextField</code> 实例的消除锯齿类型。
	<code>autoSize:Object</code>	控制文本字段的自动大小调整和对齐。
	<code>background:Boolean</code>	指定文本字段是否具有背景填充。
	<code>backgroundColor:Number</code>	文本字段背景的颜色。
	<code>border:Boolean</code>	指定文本字段是否具有边框。
	<code>borderColor:Number</code>	文本字段边框的颜色。
	<code>bottomScroll:Number [ 只读 ]</code>	一个整数 （从一开始的索引），指示文本字段中当前可见的最后一行。
	<code>condenseWhite:Boolean</code>	一个布尔值，指定当 HTML 文本字段在浏览器中呈现时是否删除字段中的额外空白 （空格、换行符等）。
	<code>embedFonts:Boolean</code>	指定是否使用嵌入字体轮廓进行呈现。
	<code>filters:Array</code>	一个索引数组，包含当前与文本字段相关联的每个滤镜对象。
	<code>gridFitType:String</code>	用于此 <code>TextField</code> 实例的网格固定类型。
	<code>_height:Number</code>	文本字段的高度，以像素为单位。
	<code>_highquality:Number</code>	自 Flash Player 7 后不推荐使用。不推荐使用此属性，而推荐使用 <code>TextField._quality</code> 。 指定当前 SWF 文件所应用的消除锯齿的级别。
	<code>hscroll:Number</code>	指示当前水平滚动位置。
	<code>html:Boolean</code>	指示文本字段是否包含 HTML 表示形式的标志。
	<code>htmlText:String</code>	如果文本字段为 HTML 文本字段，则此属性包含文本字段内容的 HTML 表示形式。
	<code>length:Number [ 只读 ]</code>	指示文本字段中的字符数。
	<code>maxChars:Number</code>	指示文本字段最多可容纳的字符数。
	<code>maxhscroll:Number [ 只读 ]</code>	表示 <code>TextField.hscroll</code> 的最大值。

修饰符	属性	说明
	maxscroll:Number [ 只读 ]	指示 TextField.scroll 的最大值。
	menu:ContextMenu	将 ContextMenu 对象 <i>contextMenu</i> 与文本字段 <i>my_txt</i> 关联。
	mouseWheelEnabled:Boolean	一个布尔值，指示当鼠标指针单击某个文本字段且用户滚动鼠标滚轮时，Flash Player 是否应自动滚动多个文本字段。
	multiline:Boolean	指示文本字段是否为多行文本字段。
	_name:String	文本字段的实例名称。
	_parent:MovieClip	对包含当前文本字段或对象的影片剪辑或对象的引用。
	password:Boolean	指定文本字段是否是密码文本字段。
	_quality:String	用于 SWF 文件的呈现品质。
	restrict:String	指示用户可输入到文本字段中的字符集。
	_rotation:Number	文本字段距其原始方向的旋转程度，以度为单位。
	scroll:Number	文本在文本字段中的垂直位置。
	selectable:Boolean	一个布尔值，指示文本字段是否可选。
	sharpness:Number	此 TextField 实例中字型边缘的清晰度。
	_soundbuftime:Number	在声音开始进入流之前预先缓冲的秒数。
	styleSheet:StyleSheet	将样式表附加到文本字段。
	tabEnabled:Boolean	指定文本字段是否包括在 Tab 键的自动排序中。
	tabIndex:Number	用于自定义 SWF 文件中对象的 Tab 键排序。
	_target:String [ 只读 ]	文本字段实例的目标路径。
	text:String	指示文本字段中的当前文本。
	textColor:Number	指示文本字段中文本的颜色。
	textHeight:Number	指示文本的高度。
	textWidth:Number	指示文本的宽度。
	thickness:Number	此 TextField 实例中字型边缘的粗细。
	type:String	指定文本字段的类型。
	_url:String [ 只读 ]	检索创建文本字段的 SWF 文件的 URL。
	variable:String	与文本字段关联的变量的名称。
	_visible:Boolean	一个布尔值，指示文本字段 <i>my_txt</i> 是否可见。

修饰符	属性	说明
	<code>_width:Number</code>	文本字段的宽度，以像素为单位。
	<code>wordWrap:Boolean</code>	一个布尔值，指示文本字段是否自动换行。
	<code>_x:Number</code>	一个整数，用来设置文本字段相对于父级影片剪辑的本地坐标的 x 坐标。
	<code>_xmouse:Number</code> [ 只读 ]	返回鼠标位置相对于文本字段的 x 坐标。
	<code>_xscale:Number</code>	确定从文本字段注册点开始应用的文本字段的水平缩放比例，以百分比表示。
	<code>_y:Number</code>	文本字段相对于父级影片剪辑的本地坐标的 y 坐标。
	<code>_ymouse:Number</code> [ 只读 ]	指示鼠标位置相对于文本字段的 y 坐标。
	<code>_yscale:Number</code>	从文本字段的注册点开始应用的文本字段的垂直缩放比例，以百分比表示。

继承自 `Object` 类的属性

`constructor` (`Object.constructor` 属性), `__proto__` (`Object.__proto__` 属性), `prototype` (`Object.prototype` 属性), `__resolve` (`Object.__resolve` 属性)

### 事件摘要

事件	说明
<code>onChanged = function(changedField:TextField) {}</code>	事件处理函数 / 侦听器；在文本字段的内容发生更改时调用。
<code>onKillFocus = function(newFocus:Object) {}</code>	在文本字段失去键盘焦点时调用。
<code>onScroller = function(scrolledField:TextField) {}</code>	事件处理函数 / 侦听器；在某一个文本字段的 <code>scroll</code> 属性发生更改时调用。
<code>onSetFocus = function(oldFocus:Object) {}</code>	在文本字段接收键盘焦点时调用。

方法摘要

修饰符	签名	说明
	<code>addListener(listener:Object) : Boolean</code>	注册一个对象，以接收 TextField 事件通知。
	<code>getDepth() : Number</code>	返回文本字段的深度。
<code>static</code>	<code>getFontList() : Array</code>	以数组的形式返回播放器的主机系统上的字体名称。
	<code>getNewTextFormat() : TextFormat</code>	返回一个 TextFormat 对象，该对象包含文本字段的文本格式对象的一个副本。
	<code>getTextFormat([beginIndex:Number], [endIndex:Number]) : TextFormat</code>	返回一个字符、一段字符或整个 TextField 对象的 TextFormat 对象。
	<code>removeListener(listener:Object) : Boolean</code>	删除以前使用 TextField.addListener() 注册到文本字段实例的侦听器对象。
	<code>removeTextField() : Void</code>	删除文本字段。
	<code>replaceSel(newText:String) : Void</code>	使用 newText 参数的内容替换当前所选内容。
	<code>replaceText(beginIndex:Number, endIndex:Number, newText:String) : Void</code>	在指定的文本字段中，用 newText 参数的内容替换由 beginIndex 和 endIndex 参数所指定的一段字符。
	<code>setNewTextFormat(tf:TextFormat) : Void</code>	设置文本字段的默认新文本格式。
	<code>setTextFormat([beginIndex:Number], [endIndex:Number], textFormat:TextFormat) : Void</code>	将 textFormat 参数指定的文本格式应用于文本字段中的某些文本或全部文本。

继承自 Object 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## addListener (TextField.addListener 方法)

```
public addListener(listener:Object) : Boolean
```

注册一个对象，以接收 **TextField** 事件通知。只要已调用 `onChanged` 和 `onScroller` 事件处理函数，该对象就将接收事件通知。当文本字段更改或滚动时，先调用 `TextField.onChanged` 和 `TextField.onScroller` 事件处理函数，然后调用任何注册为侦听器的对象的 `onChanged` 和 `onScroller` 事件处理函数。可将多个对象注册为侦听器。

若要删除从文本字段删除侦听器对象，可以调用 `TextField.removeListener()`。

事件源把对文本字段实例的一个引用作为参数传递给 `onScroller` 和 `onChanged` 处理函数。您可以通过将参数放入事件处理函数方法来捕获此数据。例如，以下代码使用 `txt` 作为传递给 `onScroller` 事件处理函数的参数。该参数随后用在 `trace` 语句中，将文本字段的实例名称发送到“输出”面板。

```
my_txt.onScroller = function(textfield_txt:TextField) {  
    trace(textfield_txt._name+" scrolled");  
};
```

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**listener:Object** - 一个具有 `onChanged` 或 `onScroller` 事件处理函数的对象。

### 返回

**Boolean** -

### 示例

下面的示例为输入文本字段 `my_txt` 定义一个 `onChanged` 处理函数。然后定义一个新的侦听器对象 `txtListener`，并为该对象定义一个 `onChanged` 处理函数。当文本字段 `my_txt` 发生更改时，将调用此处理函数。代码的最后一行调用 `TextField.addListener` 向文本字段 `my_txt` 注册侦听器对象 `txtListener`，以便在 `my_txt` 发生更改时它将得到通知。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);  
my_txt.border = true;  
my_txt.type = "input";  
  
my_txt.onChanged = function(textfield_txt:TextField) {  
    trace(textfield_txt._name+" changed");  
};  
var txtListener:Object = new Object();  
txtListener.onChanged = function(textfield_txt:TextField) {  
    trace(textfield_txt._name+" changed and notified myListener");  
};  
my_txt.addListener(txtListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[onChanged \(TextField.onChanged 处理函数\)](#), [onScroller \(TextField.onScroller 处理函数\)](#), [removeListener \(TextField.removeListener 方法\)](#)

## `_alpha` (TextField.\_alpha 属性)

```
public _alpha : Number
```

设置或检索文本字段的 **Alpha** 透明度值。有效值为 **0**（完全透明）到 **100**（完全不透明）。默认值为 **100**。使用设备字体的文本字段不支持透明度值。您必须使用嵌入字体才能对文本字段使用 `_alpha` 透明度属性。

**可用性:** **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的代码将名为 `my_txt` 的文本字段的 `_alpha` 属性设置为 **20%**。请从“库”选项菜单中选择“新建字型”，从而在库中创建一个新字体元件。然后将字体的链接设置为 `my font`。将字体元件的链接设置为 `my font`。请将以下 **ActionScript** 代码添加到 **FLA** 或 **AS** 文件。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";
// where 'my font' is the linkage name of a font in the Library
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.embedFonts = true;
my_txt.text = "Hello World";
my_txt.setTextFormat(my_fmt);
my_txt._alpha = 20;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[\\_alpha \(Button.\\_alpha 属性\)](#), [\\_alpha \(MovieClip.\\_alpha 属性\)](#)

# antiAliasType（TextField.antiAliasType 属性）

public antiAliasType : String

用于此 **TextField** 实例的消除锯齿类型。高级消除锯齿仅在 **Flash Player 8** 和更高版本中可用。仅在字体为嵌入（即 `embedFonts` 属性设置为 `true`）时可以控制此设置。对于 **Flash Player 8**，默认设置为 `"advanced"`。

若要设置此属性的值，请使用下列字符串值：

字符串值	说明
"normal"	应用常规文本消除锯齿。这与 <b>Flash Player</b> 在第 7 版和更早版本中使用的消除锯齿类型匹配。
"advanced"	应用高级消除锯齿将增加文本的可读性。（此功能在 <b>Flash Player 8</b> 中可用。）高级消除锯齿可以高品质的呈现小尺寸的字体。它最适合需要显示大量小字的情况。建议不要对大于 48 磅的字体使用高级消除锯齿。

可用性：ActionScript 1.0；Flash Player 8

## 示例

本示例将创建两个文本字段，且仅将高级消除锯齿应用于第一个文本字段。假定您已将字体嵌入到链接标识符设置为 `"Times-12"` 的库中。若要嵌入字体，请按照下列步骤操作：

- 打开您的库
- 单击该库右上角的“库”选项菜单
- 从下拉列表中选择“新建字体”
- 将字体命名为“Times-12”
- 从字体下拉列表中选择“Times New Roman”
- 按“确定”按钮
- 右键单击新创建的字体并选择“链接 ...”
- 选中“为 ActionScript 导出”框
- 通过按“确定”按钮接受默认标识符“Times-12”

```
var my_format:TextFormat = new TextFormat();
my_format.font = "Times-12";

var my_text1:TextField = this.createTextField("my_text1",
    this.getNextHighestDepth(), 10, 10, 300, 30);
my_text1.text = "This text uses advanced anti-aliasing.";
my_text1.antiAliasType = "advanced";
my_text1.border = true;
my_text1.embedFonts = true;
my_text1.setTextFormat(my_format);
```



```
var my_text2:TextField = this.createTextField("my_text2",
    this.getNextHighestDepth(), 10, 50, 300, 30);
my_text2.text = "This text uses normal anti-aliasing."
my_text2.antiAliasType = "normal";
my_text2.border = true;
my_text2.embedFonts = true;
my_text2.setTextFormat(my_format);
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[TextRenderer \(flash.text.TextRenderer\)](#), [gridFitType \(TextField.gridFitType 属性\)](#), [thickness \(TextField.thickness 属性\)](#), [sharpness \(TextField.sharpness 属性\)](#)

## autoSize (TextField.autoSize 属性)

```
public autoSize : Object
```

控制文本字段的自动大小调整和对齐。**autoSize** 的可接受值为 "none" (默认值)、"left"、"right" 和 "center"。当您设置 **autoSize** 属性时，true 是 "left" 的同义词，false 是 "none" 的同义词。

**autoSize** 和 `TextField.wordWrap` 的值决定文本字段是扩展还是收缩到左边、右边或底边。这些属性的默认值均为 false。

如果 **autoSize** 设置为 "none" (默认值) 或 false，则不会调整大小。

如果 **autoSize** 设置为 "left" 或 true，则文本将被视为左对齐的文本，这意味着该文本字段的左边保持固定，而在右边调整单行文本字段的大小。如果文本中包括换行符 (例如 "\n" 或 "\r")，则也会调整底边以适合文本的下一行。如果 `wordWrap` 也设置为 true，则仅调整文本字段底边的大小，而右边保持固定。

如果 **autoSize** 设置为 "right"，则文本将被视为右对齐的文本，这意味着该文本字段的右边保持固定，而在左边调整单行文本字段的大小。如果文本中包括换行符 (例如 "\n" 或 "\r")，则也会调整底边以适合文本的下一行。如果 `wordWrap` 也设置为 true，则仅调整文本字段底边的大小，而左边保持固定。

如果 **autoSize** 设置为 "center"，则文本将被视为居中对齐的文本，这意味着单行文本字段大小的调整在右边和左边均衡分布。如果文本中包括换行符 (例如 "\n" 或 "\r")，则也会调整底边以适合文本的下一行。如果 `wordWrap` 也设置为 true，则仅调整文本字段底边的大小，而左边和右边保持固定。

可用性: **ActionScript 1.0 ; Flash Player 6**

## 示例

您可以使用以下代码并为 `autoSize` 输入不同的值，以查看在这些值改变时字段大小如何调整。使用 `autoSize` 的几个不同设置，在播放 **SWF** 文件的同时单击鼠标会将每个文本字段的 "short text" 字符串替换为更长的文本。

```
this.createTextField("left_txt", 997, 10, 10, 70, 30);
this.createTextField("center_txt", 998, 10, 50, 70, 30);
this.createTextField("right_txt", 999, 10, 100, 70, 30);
this.createTextField("true_txt", 1000, 10, 150, 70, 30);
this.createTextField("false_txt", 1001, 10, 200, 70, 30);

left_txt.text = "short text";
left_txt.border = true;

center_txt.text = "short text";
center_txt.border = true;

right_txt.text = "short text";
right_txt.border = true;

true_txt.text = "short text";
true_txt.border = true;

false_txt.text = "short text";
false_txt.border = true;

// create a mouse listener object to detect mouse clicks
var myMouseListener:Object = new Object();
// define a function that executes when a user clicks the mouse
myMouseListener.onMouseDown = function() {
    left_txt.autoSize = "left";
    left_txt.text = "This is much longer text";
    center_txt.autoSize = "center";
    center_txt.text = "This is much longer text";
    right_txt.autoSize = "right";
    right_txt.text = "This is much longer text";
    true_txt.autoSize = true;
    true_txt.text = "This is much longer text";
    false_txt.autoSize = false;
    false_txt.text = "This is much longer text";
};
// register the listener object with the Mouse object
Mouse.addListener(myMouseListener);
```

## background (TextField.background 属性)

public background : Boolean

指定文本字段是否具有背景填充。如果为 true，则文本字段具有背景填充。如果为 false，则文本字段没有背景填充。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个文本字段，当您按下键盘上的几乎任何键时，将呈现和隐藏其背景颜色。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320,
    240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
my_txt.backgroundColor = 0xFF0000;
```

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
    my_txt.background = !my_txt.background;
};
Key.addListener(keyListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## backgroundColor (TextField.backgroundColor 属性)

public backgroundColor : Number

文本字段背景的颜色。默认值为 0xFFFFFF（白色）。即使当前没有背景，也可检索或设置此属性，但只有当文本字段有边框时，背景颜色才可见。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

请参见 `TextField.background` 的示例。

### 另请参见

[background \(TextField.background 属性\)](#)

## border (TextField.border 属性)

public border : Boolean

指定文本字段是否具有边框。如果为 true，则文本字段具有边框。如果为 false，则文本字段没有边框。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

以下示例创建名为 my\_txt 的文本字段，将 **border** 属性设置为 true，并显示该字段中的一些文本。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320,
    240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

## borderColor (TextField.borderColor 属性)

public borderColor : Number

文本字段边框的颜色。默认值为 0x000000（黑色）。即使当前没有边框，也可检索或设置此属性。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

以下示例创建名为 my\_txt 的文本字段，将 **border** 属性设置为 true，并显示该字段中的一些文本。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320,
    240);
my_txt.border = true;
my_txt.borderColor = 0x00FF00;
my_txt.text = "Lorum ipsum";
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

另请参见

[border \(TextField.border 属性\)](#)

## bottomScroll (TextField.bottomScroll 属性)

`public bottomScroll : Number [read-only]`

一个整数（从一开始的索引），指示文本字段中当前可见的最后一行。可将文本字段看作文本块上的一个窗口。属性 `TextField.scroll` 是此窗口中最顶端可见行的索引（从 1 开始）。文本字段中 `TextField.scroll` 行和 `TextField.bottomScroll` 行之间的所有文本当前都可见。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个文本字段并用文本填充它。您必须插入一个按钮（其实例名称为“`my_btn`”），在单击它时，将跟踪 `comment_txt` 字段的 `scroll` 和 `bottomScroll` 文本字段属性。

```
this.createTextField("comment_txt", this.getNextHighestDepth(), 0, 0, 160,
    120);
comment_txt.html = true;
comment_txt.selectable = true;
comment_txt.multiline = true;
comment_txt.wordWrap = true;
comment_txt.htmlText = "<b>What is hexadecimal?</b><br>"
    + "The hexadecimal color system uses six digits to represent color values."
    + "Each digit has sixteen possible values or characters. The characters"
    + " range"
    + " from 0 to 9 and then A to F. Black is represented by (#000000) and"
    + " white, "
    + "at the opposite end of the color system, is (#FFFFFF).";
my_btn.onRelease = function() {
    trace("scroll: "+comment_txt.scroll);
    trace("bottomScroll: "+comment_txt.bottomScroll);
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## condenseWhite (TextField.condenseWhite 属性)

public condenseWhite : Boolean

一个布尔值，指定当 **HTML** 文本字段在浏览器中呈现时是否删除字段中的额外空白（空格、换行符等）。默认值为 `false`。

如果将此值设置为 `true`，则必须使用标准 **HTML** 命令（如 `<BR>` 和 `<P>`）将换行符放在文本字段中。

如果文本字段的 `.html` 为 `false`，则忽略此属性。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建名为 `first_txt` 和 `second_txt` 的两个文本字段。第二个文本字段中的空白会被删除。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
var my_str:String = "Hello\tWorld\nHow are you?\t\t\tEnd";

this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160,
    120);
first_txt.html = true;
first_txt.multiline = true;
first_txt.wordWrap = true;
first_txt.condenseWhite = false;
first_txt.border = true;
first_txt.htmlText = my_str;

this.createTextField("second_txt", this.getNextHighestDepth(), 180, 10, 160,
    120);
second_txt.html = true;
second_txt.multiline = true;
second_txt.wordWrap = true;
second_txt.condenseWhite = true;
second_txt.border = true;
second_txt.htmlText = my_str;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[html \(TextField.html 属性\)](#)

## embedFonts (TextField.embedFonts 属性)

```
public embedFonts : Boolean
```

指定是否使用嵌入字体轮廓进行呈现。一个布尔值，当它为 true 时，使用嵌入字体轮廓呈现文本字段。如果为 false，则使用设备字体呈现文本字段。

如果将文本字段的 embedFonts 设置为 true，则必须通过应用于该文本字段的 TextFormat 对象的 font 属性，指定该文本的字体。如果库中不存在指定的字体（具有对应的链接实例名称），则将不显示文本。

可用性：ActionScript 1.0；Flash Player 6

### 示例

在此示例中，您需要创建名为 my\_txt 的动态文本字段，然后使用以下 **ActionScript** 嵌入字体并旋转该文本字段。对 my font 的引用是指库中的字体元件，且链接设置为 my font。此示例假定在库中具有名为 my font 的字体元件，且链接属性设置如下：标识符设置为 my font，而且选择了“为 **ActionScript** 导出”和“在第一帧导出”。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";
```

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160,
    120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

## filters (TextField.filters 属性)

```
public filters : Array
```

一个索引数组，包含当前与文本字段相关联的每个滤镜对象。**flash.filters** 包中的多个类定义了可使用的特定滤镜。

在设计时或者在运行时（使用 **ActionScript** 代码），可以在 **Flash** 创作工具中应用滤镜。若要使用 **ActionScript** 应用滤镜，您必须制作整个 `TextField.filters` 数组的临时副本，修改临时数组，然后将临时数组的值重新分配给 `TextField.filters` 数组。无法直接将新滤镜对象添加到 `TextField.filters` 数组。下面的代码对名为 `myTextField` 的目标文本字段不起作用：

```
myTextField.filters[0].push(myDropShadow);
```

若要使用 **ActionScript** 添加滤镜，您必须按照以下步骤操作（假定目标影片剪辑名为 `myTextField`）：

- 使用所选滤镜类的构造函数创建一个新的滤镜对象。
- 将 `myTextField.filters` 数组的值分配给临时数组，例如一个名为 `myFilters` 的数组。
- 将新的滤镜对象添加到临时数组 `myFilters`。
- 将临时数组的值分配给 `myTextField.filters` 数组。

如果 `filters` 数组为空，则无需使用临时数组。相反，您可以直接分配包含一个或多个已创建的滤镜对象的数组。

若要修改现有的滤镜对象（不管它是在设计时创建的还是在运行时创建的），您必须使用修改 `filters` 数组副本的技巧，如下所示：

- 将 `myTextField.filters` 数组的值分配给临时数组，例如一个名为 `myFilters` 的数组。
- 使用临时数组 `myFilters` 修改属性。例如，如果要设置数组中第一个滤镜的 `quality` 属性，可以使用以下代码：`myList[0].quality = 1;`
- 将临时数组的值分配给 `myTextField.filters` 数组。

若要清除文本字段的滤镜，请将 `filters` 设置为一个空数组 (`[]`)。

如果正在使用包含多个滤镜的 `filters` 数组，并且需要跟踪分配给每个数组索引的滤镜类型，则可以维护您自己的 `filters` 数组，并使用单独的数据结构跟踪与每个数组索引关联的滤镜类型。没有简单的方法可以确定与每个 `filters` 数组索引关联的滤镜类型。

可用性：ActionScript 1.0；Flash Player 8



## 示例

下面的示例将投影滤镜添加到名为 myTextField 的文本字段。

```
var myDropFilter = new flash.filters.DropShadowFilter();
var myFilters:Array = myTextField.filters;
myFilters.push(myDropFilter);
myTextField.filters = myFilters;
```

下面的示例将数组中第一个滤镜的 quality 设置更改为 15（仅当至少一个滤镜对象已经与 myTextField 文本字段关联时，此示例才起作用）。

```
var myList:Array = myTextField.filters;
myList[0].quality = 15;
myTextField.filters = myList;
```

另请参见

## getDepth（TextField.getDepth 方法）

```
public getDepth() : Number
```

返回文本字段的深度。

可用性：ActionScript 1.0；Flash Player 6

### 返回

Number - 一个表示文本字段深度的整数。

## 示例

下面的示例演示驻留在不同深度的文本字段。在舞台上创建一个动态文本字段，并将以下 **ActionScript** 代码添加到 FLA 或 AS 文件中。该代码将在运行时动态创建两个文本字段，并输出它们的深度。

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100,
    22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100,
    22);
for (var prop in this) {
    if (this[prop] instanceof TextField) {
        var this_txt:TextField = this[prop];
        trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
    }
}
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

## getFontList (TextField.getFontList 方法)

`public static getFontList() : Array`

以数组的形式返回播放器的主机系统上的字体名称。（此方法不返回当前加载的 SWF 文件中所有字体的名称。）这些名称的类型为 `String`。此方法是全局 `TextField` 类的静态方法。在调用此方法时，不能指定文本字段实例。

可用性: `ActionScript 1.0` ; `Flash Player 6`

### 返回

`Array` - 一个字体名称的数组。

### 示例

下面的代码显示 `getFontList()` 返回的字体列表:

```
var font_array:Array = TextField.getFontList();
font_array.sort();
trace("You have "+font_array.length+" fonts currently installed");
trace("-----");
for (var i = 0; i<font_array.length; i++) {
    trace("Font #"+(i+1)+":\t"+font_array[i]);
}
```

## getNewTextFormat (TextField.getNewTextFormat 方法)

`public getNewTextFormat() : TextFormat`

返回一个 `TextFormat` 对象，该对象包含文本字段的文本格式对象的一个副本。文本格式对象是新插入的文本（例如，使用 `replaceSel()` 方法插入的文本或由用户输入的文本）接收的格式。当调用 `getNewTextFormat()` 时，返回的 `TextFormat` 对象的所有属性均已定义。所有属性都不为 `null`。

可用性: `ActionScript 1.0` ; `Flash Player 6`

### 返回

`TextFormat` - 一个 `TextFormat` 对象。

### 示例

下面的示例显示指定文本字段的 (`my_txt`) 文本格式对象。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160,
    120);
var my_fmt:TextFormat = my_txt.getNewTextFormat();
trace("TextFormat has the following properties:");
```

```
for (var prop in my_fmt) {
    trace(prop+": "+my_fmt[prop]);
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## getTextFormat（TextField.getTextFormat 方法）

```
public getTextFormat([beginIndex:Number], [endIndex:Number]) : TextFormat
```

返回一个字符、一段字符或整个 **TextField** 对象的 **TextFormat** 对象。

下表描述三种可能的用法：

用法	说明
<code>my_textField.getTextFormat()</code>	返回一个 <b>TextFormat</b> 对象，该对象包含文本字段中所有文本的格式设置信息。在结果 <b>TextFormat</b> 对象中只设置文本字段中所有文本共有的属性。所有混合型属性（意味着它在文本中的不同位置有不同的值）的值都为 <code>null</code> 。
<code>my_textField.getTextFormat(beginIndex:Number)</code>	返回一个 <b>TextFormat</b> 对象，该对象包含 <code>beginIndex</code> 处文本字段的文本格式的一个副本。
<code>my_textField.getTextFormat(beginIndex:Number, endIndex:Number)</code>	返回一个 <b>TextFormat</b> 对象，该对象包含从 <code>beginIndex</code> 到 <code>endIndex</code> 范围内文本的格式设置信息。在结果 <b>TextFormat</b> 对象中只设置特定范围内所有文本共有的属性。所有混合型属性（也就是说，在该范围内的不同位置有不同的值）的值都设置为 <code>null</code> 。

可用性：ActionScript 1.0 ； Flash Player 6

### 参数

**beginIndex**:Number [ 可选 ] - 一个整数，指定字符串中的某个字符。如果您没有指定 `beginIndex` 和 `endIndex`，则返回整个 **TextField** 的 **TextFormat** 对象。

**endIndex**:Number [ 可选 ] - 一个整数，指定文本范围的结尾位置。如果指定了 `beginIndex` 而没有指定 `endIndex`，则返回由 `beginIndex` 指定的单个字符的 **TextFormat**。

### 返回

**TextFormat** - **TextFormat** 对象，它表示指定文本的格式设置属性。

示例

下面的 **ActionScript** 代码跟踪在运行时创建的文本字段的所有格式设置信息。

```
this.createTextField("dyn_txt", this.getNextHighestDepth(), 0, 0, 100, 200);
dyn_txt.text = "Frank";
dyn_txt.setTextFormat(new TextFormat());
var my_fmt:TextFormat = dyn_txt.getTextFormat();
for (var prop in my_fmt) {
    trace(prop+": "+my_fmt[prop]);
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[getNewTextFormat](#) ([TextField.getNewTextFormat](#) 方法), [setNewTextFormat](#) ([TextField.setNewTextFormat](#) 方法), [setTextFormat](#) ([TextField.setTextFormat](#) 方法)

# gridFitType (TextField.gridFitType 属性)

public gridFitType : String

用于此 **TextField** 实例的网格固定类型。仅在文本字段的 `antiAliasType` 属性设置为 "advanced" 时才应用此属性。

对于 `gridFitType` 属性，您可以使用下列字符串值：

字符串值	说明
"none"	指定无网格固定。不强制根据像素网格调整字型中的水平线和垂直线。此设置通常适合动画或大号字。
"pixel"	指定强力水平线和垂直线适合像素网格。此设置仅适用于左对齐的文本字段。若要使用此设置，文本字段的 <code>antiAliasType</code> 属性必须设置为 "advanced"。此设置通常能为左对齐的文本提供最佳可读性。
"subpixel"	指定强力水平线和垂直线适合 LCD 显示器上的子像素网格。若要使用此设置，文本字段的 <code>antiAliasType</code> 属性必须设置为 "advanced"。"subpixel" 设置通常适合右对齐或居中的动态文本，有时，为了在动画与文本品质之间达到一种平衡，也可使用该设置。

可用性：ActionScript 1.0 ； Flash Player 8

## 示例

本示例将说明使用不同 `gridFitType` 设置的三种文本字段。假定您已将字体嵌入到链接标识符设置为 "Times-12" 的库中。若要嵌入字体，请按照下列步骤操作：

- 打开您的库
- 单击该库右上角的“库”选项菜单
- 从下拉列表中选择“新建字体”
- 将字体命名为“Times-12”
- 从字体下拉列表中选择“Times New Roman”
- 按“确定”按钮
- 右键单击新创建的字体并选择“链接...”
- 选中“为 ActionScript 导出”框
- 通过按“确定”按钮接受默认标识符“Times-12”

```
var my_format:TextFormat = new TextFormat();
my_format.font = "Times-12";
```

```
var my_text1:TextField = this.createTextField("my_text1",
    this.getNextHighestDepth(), 9.5, 10, 400, 100);
my_text1.text = "this.gridFitType = none";
my_text1.embedFonts = true;
my_text1.antiAliasType = "advanced";
my_text1.gridFitType = "none";
my_text1.setTextFormat(my_format);
```

```
var my_text2:TextField = this.createTextField("my_text2",
    this.getNextHighestDepth(), 9.5, 40, 400, 100);
my_text2.text = "this.gridFitType = advanced";
my_text2.embedFonts = true;
my_text2.antiAliasType = "advanced";
my_text2.gridFitType = "pixel";
my_text2.setTextFormat(my_format);
```

```
var my_text3:TextField = this.createTextField("my_text3",
    this.getNextHighestDepth(), 9.5, 70, 400, 100);
my_text3.text = "this.gridFitType = subpixel";
my_text3.embedFonts = true;
my_text3.antiAliasType = "advanced";
my_text3.gridFitType = "subpixel";
my_text3.setTextFormat(my_format);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 `DepthManager` 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[TextRenderer \(flash.text.TextRenderer\)](#), [antiAliasType \(TextField.antiAliasType 属性\)](#), [sharpness \(TextField.sharpness 属性\)](#)

## `_height` (`TextField._height` 属性)

`public _height : Number`

文本字段的高度，以像素为单位。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的代码示例设置文本字段的高度和宽度：

```
my_txt._width = 200;  
my_txt._height = 200;
```

## `_highquality` (`TextField._highquality` 属性)

`public _highquality : Number`

自 **Flash Player 7** 后不推荐使用。不推荐使用此属性，而推荐使用 `TextField._quality`。

指定当前 **SWF** 文件所应用的消除锯齿的级别。指定 **2**（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 **1**（高品质）将应用消除锯齿；如果 **SWF** 文件中没有动画并且是默认值，指定 **1** 将对位图图像进行平滑处理。指定 **0**（低品质），则不消除锯齿。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 另请参见

[\\_quality \(TextField.\\_quality 属性\)](#)

## `hscroll` (`TextField.hscroll` 属性)

`public hscroll : Number`

指示当前水平滚动位置。如果 `hscroll` 属性为 **0**，则不能水平滚动文本。

水平滚动的单位是像素，而垂直滚动的单位是行。水平滚动以像素计量是因为您通常使用的多数字体都是按比例隔开的；这意味着字符可以有不同的宽度。**Flash** 按行执行垂直滚动是因为用户通常希望看到一整行文本，而不是一行的局部。即使一行上有多种字体，行的高度也会调整到与使用的最大字体相适合。

**注意：**`hscroll` 属性从 **0** 开始，而不像垂直滚动属性 `TextField.scroll` 那样从 **1** 开始。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 示例

下面的示例使用名为 `scrollLeft_btn` 和 `scrollRight_btn` 的两个按钮水平滚动 `my_txt` 文本字段。滚动量显示在名为 `scroll_txt` 的文本字段中。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 160, 22);
my_txt.border = true;
my_txt.multiline = false;
my_txt.wordWrap = false;
my_txt.text = "Lorem ipsum dolor sit amet, consectetur adipiscing...";

scrollLeft_btn.onRelease = function() {
    my_txt.hscroll -= 10;
    scroll_txt.text = my_txt.hscroll + " of " + my_txt.maxhscroll;
};
scrollRight_btn.onRelease = function() {
    my_txt.hscroll += 10;
    scroll_txt.text = my_txt.hscroll + " of " + my_txt.maxhscroll;
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[maxhscroll](#) ([TextField.maxhscroll](#) 属性), [scroll](#) ([TextField.scroll](#) 属性)

## html (TextField.html 属性)

public html : Boolean

指示文本字段是否包含 **HTML** 表示形式的标志。如果 `html` 属性为 `true`，则文本字段为 **HTML** 文本字段。如果 `html` 为 `false`，则文本字段为非 **HTML** 文本字段。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例创建将 `html` 属性设置为 `true` 的文本字段。**HTML** 格式的文本显示在该文本字段中。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "<b> this is bold text </b>";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[htmlText \(TextField.htmlText 属性\)](#)

# htmlText (TextField.htmlText 属性)

```
public htmlText : String
```

如果文本字段为 **HTML** 文本字段，则此属性包含文本字段内容的 **HTML** 表示形式。如果文本字段不是 **HTML** 文本字段，则其行为与 `text` 属性相同。可以在属性检查器中将某个文本字段指定为 **HTML** 文本字段，或通过将文本字段的 `html` 属性设置为 `true`。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例创建将 `html` 属性设置为 `true` 的文本字段。**HTML** 格式的文本显示在该文本字段中。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "<b> this is bold text </b>";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## 另请参见

[html \(TextField.html 属性\)](#)，[asfunction 协议](#)



## length (TextField.length 属性)

public length : Number [read-only]

指示文本字段中的字符数。此属性的返回值与 `text.length` 的返回值相同，但是它更快。如 `tab (\t)` 之类的字符视为一个字符。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例输出 `date_txt` 文本字段中的字符数，该文本字段显示当前日期。

```
var today:Date = new Date();
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100,
    22);
date_txt.autoSize = true;
date_txt.text = today.toString();
trace(date_txt.length);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## maxChars (TextField.maxChars 属性)

public maxChars : Number

指示文本字段最多可容纳的字符数。脚本可以插入比 `maxChars` 允许的字符数更多的文本；`maxChars` 属性仅指示用户可以输入多少文本。如果此属性的值为 `null`，则对用户可以输入的文本字符数没有限制。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个名为 `age_txt` 的文本字段，该字段仅允许用户在其中输入最多两个数字。

```
this.createTextField("age_txt", this.getNextHighestDepth(), 10, 10, 30, 22);
age_txt.type = "input";
age_txt.border = true;
age_txt.maxChars = 2;
age_txt.restrict = "0-9";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## maxhscroll (TextField.maxhscroll 属性)

public maxhscroll : Number [read-only]

表示 TextField.hscroll 的最大值。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

请参见 TextField.hscroll 的示例。

## maxscroll (TextField.maxscroll 属性)

public maxscroll : Number [read-only]

指示 TextField.scroll 的最大值。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例设置滚动文本字段 my\_txt 的最大值。创建两个按钮 (scrollUp\_btn 和 scrollDown\_btn) 以滚动文本字段。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件。

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh "
        + "eiusmod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

## menu (TextField.menu 属性)

```
public menu : ContextMenu
```

将 **ContextMenu** 对象 `contextMenu` 与文本字段 `my_txt` 关联。 **ContextMenu** 类用于修改当用户在 **Flash Player** 中右键单击（在 **Windows** 中）或按住 **Control** 键并单击（在 **Macintosh** 中）时显示的上下文菜单。

此属性仅在可选（可编辑）文本字段上起作用；在非可选文本字段上不起作用。

可用性：ActionScript 1.0；Flash Player 7

### 示例

下面的示例将 **ContextMenu** 对象 `menu_cm` 分配给文本字段 `news_txt`。 **ContextMenu** 对象包含一个标记为“**Resize**”的自定义菜单项（具有名为 `doResize()` 的关联回调处理函数），它可以用来添加调整大小功能（未显示）：

```
this.createTextField("news_txt", this.getNextHighestDepth(), 10, 10, 320,
    240);
news_txt.border = true;
news_txt.wordWrap = true;
news_txt.multiline = true;
news_txt.text = "To see the custom context menu item, right click (PC) or ";
news_txt.text += "control click (Mac) within the text field.";
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Resize", doResize));

function doResize(obj:TextField, item:ContextMenuItem):Void {
    // "Resize" code here
    trace("you selected: "+item.caption);
}
news_txt.menu = menu_cm;
```

在文本字段的区域内右键单击或按住 **Control** 键单击时，可看到自定义菜单项。



您不能使用 Flash 已经使用的菜单项。例如，Print...（包括三个点）被 Flash 所保留，所以您无法使用此菜单项；但是，您可以使用 Print...（包括两个点）或任何 Flash 还没有使用的菜单项。

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[Button](#), [ContextMenu](#), [ContextMenuItem](#), [MovieClip](#)

## mouseWheelEnabled (TextField.mouseWheelEnabled 属性)

public mouseWheelEnabled : Boolean

一个布尔值，指示当鼠标指针单击某个文本字段且用户滚动鼠标滚轮时，Flash Player 是否应自动滚动多个文本字段。默认情况下，此值为 true。如果您想让文本字段在用户滚动鼠标滚轮时不随之滚动，或要实现您自己的文本字段滚动方式，可以使用此属性。

可用性：ActionScript 1.0 ； Flash Player 7

### 示例

下面的示例创建两个文本字段。scrollable\_txt 字段已将 mouseWheelEnabled 属性设置为 **true**，因此在您单击该字段并滚动鼠标滚轮时，scrollable\_txt 将滚动。如果您单击 nonscrollable\_txt 字段并滚动鼠标滚轮，该字段不滚动。

```
var font_array:Array = TextField.getFontList().sort();

this.createTextField("scrollable_txt", this.getNextHighestDepth(), 10, 10,
    240, 320);
scrollable_txt.border = true;
scrollable_txt.wordWrap = true;
scrollable_txt.multiline = true;
scrollable_txt.text = font_array.join("\n");

this.createTextField("nonscrollable_txt", this.getNextHighestDepth(), 260,
    10, 240, 320);
nonscrollable_txt.border = true;
nonscrollable_txt.wordWrap = true;
nonscrollable_txt.multiline = true;
nonscrollable_txt.mouseWheelEnabled = false;
nonscrollable_txt.text = font_array.join("\n");
```

Mouse.onMouseWheel

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 DepthManager 类取代本示例中所用的 MovieClip.getNextHighestDepth() 方法。

另请参见

[mouseWheelEnabled \(TextField.mouseWheelEnabled 属性\)](#)

## multiline (TextField.multiline 属性)

public multiline : Boolean

指示文本字段是否为多行文本字段。如果值为 true，则文本字段为多行文本字段；如果值为 false，则文本字段为单行文本字段。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个名为 fontList\_txt 的多行文本字段，该字段显示字体的多行长列表。

```
var font_array:Array = TextField.getFontList().sort();

this.createTextField("fontList_txt", this.getNextHighestDepth(), 10, 10,
    240, 320);
fontList_txt.border = true;
fontList_txt.wordWrap = true;
fontList_txt.multiline = true;
fontList_txt.text = font_array.join("\n");
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

## \_name (TextField.\_name 属性)

public \_name : String

文本字段的实例名称。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

以下示例演示驻留在不同深度的文本字段。在舞台上创建一个动态文本字段。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件，这样就会在运行时动态创建两个文本字段，并在“输出”面板中显示它们的深度。

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100,
    22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100,
    22);
for (var prop in this) {
    if (this[prop] instanceof TextField) {
        var this_txt:TextField = this[prop];
        trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
    }
}
```

当您测试文档时，实例名称和深度会显示在“输出”面板中。

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## onChanged (TextField.onChanged 处理函数)

```
onChanged = function(changedField:TextField) {}
```

事件处理函数 / 侦听器；在文本字段的内容发生更改时调用。默认情况下它是未定义的；可以在脚本中定义它。

对该文本字段实例的一个引用将作为参数传递给 `onChanged` 处理函数。您可以通过将参数放入事件处理函数方法来捕获此数据。例如，以下代码使用 `textfield_txt` 作为传递给 `onChanged` 事件处理函数的参数。该参数随后用在 `trace()` 语句中，将文本字段的实例名称发送到“输出”面板。

```
this.createTextField("myInputText_txt", 99, 10, 10, 300, 20);
myInputText_txt.border = true;
myInputText_txt.type = "input";
```

```
myInputText_txt.onChanged = function(textfield_txt:TextField) {
    trace("the value of "+textfield_txt._name+" was changed. New value is:
        "+textfield_txt.text);
};
```

只有当用户交互导致更改时才调用 `onChanged` 处理函数；例如，当用户在键盘上键入内容、使用鼠标在文本字段中更改内容或选择菜单项时。以程序化方式更改该文本字段不会触发 `onChanged` 事件，这是因为代码能够识别对文本字段所做的更改。

可用性：ActionScript 1.0；Flash Player 6

### 参数

`changedField:TextField` - 触发事件的字段。

### 另请参见

[TextFormat](#), [setNewTextFormat](#) ([TextField.setNewTextFormat](#) 方法)

## onKillFocus (TextField.onKillFocus 处理函数)

```
onKillFocus = function(newFocus:Object) {}
```

在文本字段失去键盘焦点时调用。onKillFocus 方法接收一个参数 newFocus，该参数是一个对象，表示要接收焦点的新对象。如果没有对象接收焦点，则 newFocus 将包含值 null。

可用性：ActionScript 1.0；Flash Player 6

### 参数

newFocus:Object - 接收焦点的对象。

### 示例

下面的示例创建名为 first\_txt 和 second\_txt 的两个文本字段。使一个文本字段具有焦点时，将在“输出”面板中显示有关具有当前焦点的文本字段和失去焦点的文本字段的信息。

```
this.createTextField("first_txt", 1, 10, 10, 300, 20);
first_txt.border = true;
first_txt.type = "input";
this.createTextField("second_txt", 2, 10, 40, 300, 20);
second_txt.border = true;
second_txt.type = "input";
first_txt.onKillFocus = function(newFocus:Object) {
    trace(this._name+" lost focus. New focus changed to: "+newFocus._name);
};
first_txt.onSetFocus = function(oldFocus:Object) {
    trace(this._name+" gained focus. Old focus changed from: "+oldFocus._name);
}
```

### 另请参见

[onSetFocus \(TextField.onSetFocus 处理函数\)](#)

## onScroller (TextField.onScroller 处理函数)

```
onScroller = function(scrolledField:TextField) {}
```

事件处理函数 / 侦听器：在某一个文本字段的 **scroll** 属性发生更改时调用。

对该文本字段实例的一个引用将作为参数传递给 onScroller 处理函数。您可以通过将参数放入事件处理函数方法来捕获此数据。例如，以下代码使用 my\_txt 作为传递给 onScroller 事件处理函数的参数。该参数随后用在 trace() 语句中，将文本字段的实例名称发送到“输出”面板。

```
myTextField.onScroller = function (my_txt:TextField) {  
    trace (my_txt._name + " scrolled");  
};
```

TextField.onScroller 事件处理函数通常用于实现滚动条。滚动条通常有一个缩略图或其它指示器，显示文本字段中的当前水平或垂直滚动位置。使用鼠标和键盘可以浏览文本字段，这会导致滚动位置发生更改。如果由于用户交互而导致滚动条位置发生更改，滚动条代码需要获得通知，这就是使用 TextField.onScroller 的目的。

不管是由于用户与文本字段交互而导致滚动位置发生更改，还是以程序化方式使滚动位置发生更改，都将调用 onScroller。只有当用户交互导致更改时才引发 onChanged 处理函数。这两个选项必须同时使用，因为通常一段代码更改了滚动位置后，如果得不到通知，不相关的滚动条代码不会知道滚动位置已发生更改。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**scrolledField:TextField** - 指向滚动位置被更改的 TextField 对象的引用。

### 示例

下面的示例创建一个名为 my\_txt 的文本字段，并使用名为 scrollUp\_btn 和 scrollDown\_btn 的两个按钮滚动该文本字段的内容。调用 onScroller 事件处理函数时，会使用 **trace** 语句将信息显示在“输出”面板中。创建实例名称为 scrollUp\_btn 和 scrollDown\_btn 的两个按钮，并将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160,  
    20);  
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320,  
    240);  
my_txt.multiline = true;  
my_txt.wordWrap = true;  
  
for (var i = 0; i<10; i++) {  
    my_txt.text += "Lorem ipsum dolor sit amet, consectetur adipiscing elit,  
    sed diam "  
        + "nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat  
    volutpat.";
```



```

}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
};
my_txt.onScroller = function() {
    trace("onScroller called");
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[hscroll \(TextField.hscroll 属性\)](#) , [maxhscroll \(TextField.maxhscroll 属性\)](#) , [maxscroll \(TextField.maxscroll 属性\)](#) , [scroll \(TextField.scroll 属性\)](#)

## onSetFocus (TextField.onSetFocus 处理函数)

```
onSetFocus = function(oldFocus:Object) {}
```

在文本字段接收键盘焦点时调用。oldFocus 参数是失去焦点的对象。例如，如果用户按下 **Tab** 键将输入焦点从一个按钮移动到一个文本字段，则 **oldFocus** 包含该按钮的实例。如果以前没有具有焦点的对象，则 oldFocus 包含一个 **null** 值。

可用性: **ActionScript 1.0 ; Flash Player 6**

参数

oldFocus:Object - 将失去焦点的对象。

示例

请参见 `TextField.onKillFocus` 的示例。

另请参见

[onKillFocus \(TextField.onKillFocus 处理函数\)](#)

## `_parent` (`TextField._parent` 属性)

```
public _parent : MovieClip
```

对包含当前文本字段或对象的影片剪辑或对象的引用。当前对象是一个包含引用 `_parent` 的 **ActionScript** 代码的对象。

使用 `_parent` 可以指定一个指向当前文本字段之上的影片剪辑或对象的相对路径。可以使用 `_parent` 在显示列表中攀升多个级别，如下所示：

```
_parent._parent._alpha = 20;
```

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的 **ActionScript** 创建两个文本字段，并输出有关每个对象的 `_parent` 的信息。第一个文本字段 `first_txt` 是在主时间轴上创建的。第二个文本字段 `second_txt` 是在名为 `holder_mc` 的影片剪辑内创建的。

```
this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
first_txt.border = true;
trace(first_txt._name+'s _parent is: '+first_txt._parent);
```

```
this.createEmptyMovieClip("holder_mc", this.getNextHighestDepth());
holder_mc.createTextField("second_txt", holder_mc.getNextHighestDepth(), 10, 40, 160, 22);
holder_mc.second_txt.border = true;
trace(holder_mc.second_txt._name+'s _parent is: '+holder_mc.second_txt._parent);
```

“输出”面板中会显示以下信息：

```
first_txt's _parent is: _level0
second_txt's _parent is: _level0.holder_mc
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[\\_parent \(Button.\\_parent 属性\)](#)，[\\_parent \(MovieClip.\\_parent 属性\)](#)

## password (TextField.password 属性)

public password : Boolean

指定文本字段是否是密码文本字段。如果密码的值为 `true`，则文本字段为密码文本字段，并使用星号替代实际字符来隐藏输入的字符。如果为 `false`，则文本字段不是密码文本字段。启用密码模式时，剪切和复制命令及其相应的键盘快捷方式不起作用。此安全机制可防止不良用户使用快捷键在无人看管的计算机上破译密码。

可用性：ActionScript 1.0；Flash Player 6

### 示例

以下示例创建两个文本字段：username\_txt 和 password\_txt。文本被输入到这两个字段中；但是，password\_txt 的 **password** 属性会设置为 `true`。因此，字符显示为星号，而不是 password\_txt 字段中的字符。

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10,
    100, 22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";

this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40,
    100, 22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_txt.password = true;
password_txt.text = "world";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## \_quality (TextField.\_quality 属性)

```
public _quality : String
```

用于 SWF 文件的呈现品质。设备字体始终是带有锯齿的，因此不受 \_quality 属性的影响。



尽管您可以为 TextField 对象指定此属性，但它实际上是一个全局属性，因此您只能将它的值指定为 \_quality。有关更多信息，请参见 \_quality 全局属性。

\_quality 属性可设置为下列值：

- “LOW” 低呈现品质。不消除图形的锯齿，位图不进行平滑处理。
- “MEDIUM” 中等呈现品质。使用 2 x 2 像素网格消除图形锯齿，但不对位图进行平滑处理。适用于不包含文本的影片。
- “HIGH” 高呈现品质。使用 4 x 4 像素网格消除图形锯齿，如果影片是静态的，则对位图进行平滑处理。这是 Flash 使用的默认呈现品质设置。
- “BEST” 极高呈现品质。使用 4 x 4 像素网格消除图形锯齿，并且始终对位图进行平滑处理。

可用性：ActionScript 1.0；Flash Player 6

### 示例

以下示例将呈现品质设置为 LOW：

```
my_txt._quality = "LOW";
```

另请参见

[\\_quality 属性](#)

## removeListener (TextField.removeListener 方法)

```
public removeListener(listener:Object) : Boolean
```

删除以前使用 TextField.addListener() 注册到文本字段实例的侦听器对象。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**listener:Object** - 不再从 TextField.onChangeed 或 TextField.onScroller 接收通知的对象。

### 返回

Boolean - 如果成功删除了 listener，则该方法的返回值为 true。如果未能成功删除 listener（例如，如果 listener 不在该 TextField 对象的侦听器列表上），则该方法的返回值为 false。

## 示例

下面的示例创建一个名为 `my_txt` 的输入文本字段。用户向字段中键入时，将在“输出”面板中显示有关文本字段中字符数的信息。如果用户单击 `removeListener_btn` 实例，则将删除侦听器且不再显示信息。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
my_txt.border = true;
my_txt.type = "input";

var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
    trace(textfield_txt+" changed. Current length is: "+textfield_txt.length);
};
my_txt.addListener(txtListener);

removeListener_btn.onRelease = function() {
    trace("Removing listener...");
    if (!my_txt.removeListener(txtListener)) {
        trace("Error! Unable to remove listener");
    }
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## removeTextField (TextField.removeTextField 方法)

```
public removeTextField(): Void
```

删除文本字段。只能对使用 `MovieClip.createTextField()` 创建的文本字段执行此操作。当调用此方法时，将删除文本字段。此方法与 `MovieClip.removeMovieClip()` 相似。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例创建一个文本字段，单击 `remove_btn` 实例时，可以将该字段从舞台中删除。创建一个按钮，将它命名为 `remove_btn`，然后将以下 **ActionScript** 添加到您的 **FLA** 或 **AS** 文件。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 300, 22);
my_txt.text = new Date().toString();
my_txt.border = true;

remove_btn.onRelease = function() {
    my_txt.removeTextField();
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## replaceSel（TextField.replaceSel 方法）

```
public replaceSel(newText:String) : Void
```

使用 `newText` 参数的内容替换当前所选内容。使用当前默认字符格式和默认段落格式，在当前所选内容的所在位置插入文本。即使文本字段是 **HTML** 文本字段，也不会按 **HTML** 处理该文本。

可以使用 `replaceSel()` 方法插入和删除文本，而不破坏其余文本的字符和段落格式。

必须使用 `Selection.setFocus()` 将焦点放置在字段上才能发布此命令。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**newText:String** - 一个字符串。

### 示例

下面的示例代码在舞台上创建一个带文本的多行文本字段。在选择某些文本，然后在文本字段上右键单击或按住 **Control** 键并单击时，您可以从上下文菜单中选择 `Enter current date`。此选择调用一个函数，该函数将所选文本替换为当前日期。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320,
    240);
my_txt.border = true;
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.text = "Select some sample text from the text field and then right-
    click/control click "
    + "and select 'Enter current date' from the context menu to replace the
    "
    + "currently selected text with the current date.";

var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Enter current date",
    enterDate));
function enterDate(obj:Object, menuItem:ContextMenuItems) {
    var today_str:String = new Date().toString();
    var date_str:String = today_str.split(" ", 3).join(" ");
    my_txt.replaceSel(date_str);
}
my_txt.menu = my_cm;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[setFocus \(Selection.setFocus 方法\)](#)

## replaceText (TextField.replaceText 方法)

```
public replaceText(beginIndex:Number, endIndex:Number, newText:String) :  
    Void
```

在指定的文本字段中，用 `newText` 参数的内容替换由 `beginIndex` 和 `endIndex` 参数所指定的一段字符。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**beginIndex:**Number - 替换范围的开始索引值。

**endIndex:**Number - 替换范围的结束索引值。

**newText:**String - 用来替换指定字符范围的文本。

### 示例

下面的示例将创建名为 `my_txt` 的文本字段，并将文本 `dog@house.net` 分配给该字段。`indexOf()` 方法用于查找指定元件 (@) 的第一个匹配项。如果找到该元件，则将指定文本（在索引 0 和元件之间）替换为字符串 `bird`。如果未找到该元件，错误消息将显示在“输出”面板中。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 22);  
my_txt.autoSize = true;  
my_txt.text = "dog@house.net";  
  
var symbol:String = "@";  
var symbolPos:Number = my_txt.text.indexOf(symbol);  
if (symbolPos > -1) {  
    my_txt.replaceText(0, symbolPos, "bird");  
} else {  
    trace("symbol '" + symbol + "' not found.");  
}
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## restrict (TextField.restrict 属性)

public restrict : String

指示用户可输入到文本字段中的字符集。如果 restrict 属性的值为 null，则可以输入任何字符。如果 restrict 属性的值为空字符串，则不能输入任何字符。如果 restrict 属性的值为一串字符，则只能在文本字段中输入该字符串中的字符。从左向右扫描该字符串。可以使用短划线 (-) 指定一个范围。它只限制用户交互；脚本可将任何文本放入文本字段中。此属性不与属性检查器中的“嵌入字体轮廓”复选框同步。

如果此字符串以 ^ 开头，则先接受所有字符，然后从接受字符集中排除字符串中 ^ 之后的字符。如果此字符串不以 ^ 开头，则最初不接受任何字符，然后将字符串中的字符包括在接受字符集中。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例仅允许在文本字段中输入大写字母、空格和数字：

```
my_txt.restrict = "A-Z 0-9";
```

下面的示例包含除小写字母之外的所有字符：

```
my_txt.restrict = "^a-z";
```

可以使用反斜杠输入 ^ 或 - 的本义。认可的反斜杠序列为 \-、\^ 或 \\。反斜杠在字符串中必须是一个本义字符，因此在 **ActionScript** 中指定时必须使用两个反斜杠。例如，下面的代码只包含短划线 (-) 和 插入符号 (^)：

```
my_txt.restrict = "\\-\\^";
```

可在字符串中的任何地方使用 ^，以在包含字符与排除字符之间进行切换。下面的代码只包含除大写字母 Q 之外的大写字母：

```
my_txt.restrict = "A-Z^Q";
```

可以使用 \u 转义序列构造 restrict 字符串。下面的代码只包含从 ASCII 32（空格）到 ASCII 126（代字号）之间的字符。

```
my_txt.restrict = "\u0020-\u007E";
```



## `_rotation` (`TextField._rotation` 属性)

`public _rotation : Number`

文本字段距其原始方向的旋转程度，以度为单位。从 **0** 到 **180** 的值表示顺时针方向旋转；从 **0** 到 **-180** 的值表示逆时针方向旋转。对于此范围之外的值，可以通过加上或减去 **360** 获得该范围内的值。例如，`my_txt._rotation = 450` 语句与 `my_txt._rotation = 90` 是相同的。

使用设备字体的文本字段不支持旋转值。您必须使用嵌入字体才能对文本字段使用 `_rotation`。

可用性：ActionScript 1.0；Flash Player 6

### 示例

在此示例中，您需要创建名为 `my_txt` 的动态文本字段，然后使用以下 **ActionScript** 嵌入字体并旋转该文本字段。对 `my font` 的引用是指库中的字体元件，且链接设置为 `my font`。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160,
    120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

使用 `TextFormat` class 类为文本字段应用其它格式。

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[\\_rotation \(Button.\\_rotation 属性\)](#), [\\_rotation \(MovieClip.\\_rotation 属性\)](#), [TextFormat](#)

## scroll (TextField.scroll 属性)

`public scroll : Number`

文本在文本字段中的垂直位置。**scroll** 属性可用于将用户定向到长篇文章的特定段落，还可用于创建滚动文本字段。可以检索和修改此属性。

水平滚动的单位是像素，而垂直滚动的单位是行。水平滚动以像素计量是因为您通常使用的多数字体都是按比例隔开的，这意味着字符可以有不同的宽度。**Flash** 按行执行垂直滚动是因为用户通常希望看到一整行文本，而不是一行的局部。即使一行上有多种字体，行的高度也会调整到与使用的最大字体相适合。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例设置滚动文本字段 `my_txt` 的最大值。创建两个按钮 (`scrollUp_btn` 和 `scrollDown_btn`) 以在文本字段中滚动。请将以下 **ActionScript** 代码添加到 **FLA** 或 **AS** 文件。

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
    my_txt.text += "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.";
}
scrollUp_btn.onRelease = function() {
    my_txt.scroll--;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
    my_txt.scroll++;
    scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

### 另请参见

[hscroll \(TextField.hscroll 属性\)](#), [maxscroll \(TextField.maxscroll 属性\)](#)

## selectable (TextField.selectable 属性)

`public selectable : Boolean`

一个布尔值，指示文本字段是否可选。值 `true` 表示文本可选。`selectable` 属性控制文本字段是否可选，而不控制文本字段是否可编辑。动态文本字段即使不可编辑，它也可能是可选的。如果动态文本字段是不可选的，则您不能选择其中的文本。

如果 **selectable** 设置为 `false`，则文本字段中的文本不响应来自鼠标或键盘的“选择”命令，并且不能使用“复制”命令复制文本。如果 **selectable** 设置为 `true`，则可以使用鼠标或键盘选择该文本字段中的文本。即使文本字段是动态文本字段而不是输入文本字段，您也可以使用这种方式选择文本。可以使用“复制”命令复制文本。

**可用性:** **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例创建一个可选择的文本字段，该字段不断地用当前日期和时间更新。

```
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100,
    22);
date_txt.autoSize = true;
date_txt.selectable = true;

var date_interval:Number = setInterval(updateTime, 500, date_txt);
function updateTime(my_txt:TextField) {
    my_txt.text = new Date().toString();
}
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## setNewTextFormat (TextField.setNewTextFormat 方法)

```
public setNewTextFormat(tf:TextFormat) : Void
```

设置文本字段的默认新文本格式。默认新文本格式是用于新插入文本（如使用 `replaceSel()` 方法插入的文本或由用户输入的文本）的新文本格式。插入文本时，将默认的新文本格式分配给新插入的文本。

默认新文本格式由 `textFormat` 指定，它是一个 `TextFormat` 对象。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

`tf:TextFormat` - 一个 `TextFormat` 对象。

### 示例

在下面的示例中，在运行时创建一个新文本字段（名为 `my_txt`）并设置几个属性。应用了新插入文本的格式。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
my_fmt.color = 0xFF9900;

this.createTextField("my_txt", 999, 0, 0, 400, 300);
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.border = true;
my_txt.type = "input";
my_txt.setNewTextFormat(my_fmt);
my_txt.text = "Oranges are a good source of vitamin C";
```

### 另请参见

[getNewTextFormat \(TextField.getNewTextFormat 方法\)](#), [getTextFormat \(TextField.getTextFormat 方法\)](#), [setTextFormat \(TextField.setTextFormat 方法\)](#)

# setTextFormat (TextField.setTextFormat 方法)

```
public setTextFormat([beginIndex:Number], [endIndex:Number],
    textFormat:TextFormat) : Void
```

将 textFormat 参数指定的文本格式应用于文本字段中的某些文本或全部文本。textFormat 必须是一个指定需要的文本格式更改的 TextFormat 对象。只有非空的 textFormat 属性才会应用到文本字段。不会应用设置为 null 的任何 textFormat 属性。默认情况下，新创建的 TextFormat 对象的所有属性都设置为 null。

在 TextFormat 对象中，有两种类型的格式信息：字符级别格式设置和段落级别格式设置。文本字段中的每个字符都可以有其自己的字符格式设置，如字体名称、字体大小、粗体和斜体。

对于段落，检查该段落的第一个字符的格式设置以代表整个段落。段落格式设置的示例有左边距、右边距和缩进。

setTextFormat() 方法更改应用于文本字段中单个字符、一段字符或整体文本的文本格式。下表中说明了这些用法：

用法	说明
my_textField.setTextFormat(textFormat:TextFormat)	将 textFormat 的属性应用于文本字段中的所有文本。
my_textField.setTextFormat(beginIndex:Number, textFormat:TextFormat)	将 textFormat 的属性应用于 beginIndex 处的字符。
my_textField.setTextFormat(beginIndex:Number, endIndex:Number, textFormat:TextFormat)	将 textFormat 参数的属性应用于范围从 beginIndex 处到 endIndex 处的文本。

请注意，用户手动插入的任何文本或通过 TextField.replaceSel() 替换的任何文本都会接收该文本字段的默认格式（而不是为文本插入点处指定的格式）作为新文本格式。若要为新文本设置文本字段的默认格式，请使用 TextField.setNewTextFormat()。

可用性：ActionScript 1.0 ； Flash Player 6

## 参数

**beginIndex:**Number [ 可选 ] - 一个整数，指定所需文本范围的第一个字符。如果您没有指定 **beginIndex** 和 **endIndex**，则 **TextFormat** 将应用于整个 **TextField**。

**endIndex:**Number [ 可选 ] - 一个整数，指定所需文本范围后的第一个字符。如果指定了 **beginIndex** 而没有指定 **endIndex**，则 **TextFormat** 将应用于 **beginIndex** 指定的单个字符。

**textFormat:**TextFormat - 一个包含字符和段落格式设置信息的 **TextFormat** 对象。

## 示例

下面的示例为两个不同的文本字符串设置文本格式。调用 **setTextFormat()** 方法并将其应用于 **my\_txt** 文本字段。

```
var format1_fmt:TextFormat = new TextFormat();
format1_fmt.font = "Arial";
var format2_fmt:TextFormat = new TextFormat();
format2_fmt.font = "Courier";

var string1:String = "Sample string number one."+newline;
var string2:String = "Sample string number two."+newline;

this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.text = string1;
var firstIndex:Number = my_txt.length;
my_txt.text += string2;
var secondIndex:Number = my_txt.length;

my_txt.setTextFormat(0, firstIndex, format1_fmt);
my_txt.setTextFormat(firstIndex, secondIndex, format2_fmt);
```

此示例中使用的 **MovieClip.getNextHighestDepth()** 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 **MovieClip.getNextHighestDepth()** 方法。

## 另请参见

[TextFormat](#), [setNewTextFormat \(TextField.setNewTextFormat 方法\)](#)

## sharpness (TextField.sharpness 属性)

public sharpness : Number

此 **TextField** 实例中字型边缘的清晰度。仅在文本字段的 `antiAliasType` 属性设置为 "advanced" 时才应用此属性。sharpness 的范围是从 -400 到 400 的数字。如果您尝试将 sharpness 设置为该范围外的值，则 **Flash** 会将该属性设置为范围内最接近的值（-400 或 400）。

可用性：ActionScript 1.0；Flash Player 8

### 示例

本示例创建三个文本字段，其 sharpness 分别设置为 400、0 和 -400。假定您已将字体嵌入到链接标识符设置为 "Times-12" 的库中。若要嵌入字体，请按照下列步骤操作：

- 打开您的库
- 单击该库右上角的“库”选项菜单
- 从下拉列表中选择“新建字体”
- 将字体命名为“Times-12”
- 从字体下拉列表中选择“Times New Roman”
- 按“确定”按钮
- 右键单击新创建的字体并选择“链接...”
- 选中“为 ActionScript 导出”框
- 通过按“确定”按钮接受默认标识符“Times-12”

```
var my_format:TextFormat = new TextFormat();  
my_format.font = "Times-12";
```

```
var my_text1:TextField = this.createTextField("my_text1",  
    this.getNextHighestDepth(), 10, 10, 400, 100);  
my_text1.text = "This text has sharpness set to 400."  
my_text1.embedFonts = true;  
my_text1.antiAliasType = "advanced";  
my_text1.gridFitType = "pixel";  
my_text1.sharpness = 400;  
my_text1.setTextFormat(my_format);
```

```
var my_text2:TextField = this.createTextField("my_text2",  
    this.getNextHighestDepth(), 10, 40, 400, 100);  
my_text2.text = "This text has sharpness set to 0."  
my_text2.embedFonts = true;  
my_text2.antiAliasType = "advanced";  
my_text2.gridFitType = "pixel";  
my_text2.sharpness = 0;  
my_text2.setTextFormat(my_format);
```

```
var my_text3:TextField = this.createTextField("my_text3",
    this.getNextHighestDepth(), 10, 70, 400, 100);
my_text3.text = "This text has sharpness set to -400."
my_text3.embedFonts = true;
my_text3.antiAliasType = "advanced";
my_text3.gridFitType = "pixel";
my_text3.sharpness = -400;
my_text3.setTextFormat(my_format);
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[gridFitType \(TextField.gridFitType 属性\)](#)，[antiAliasType \(TextField.antiAliasType 属性\)](#)

## **\_soundbuftime** (TextField.\_soundbuftime 属性)

`public _soundbuftime : Number`

在声音开始进入流之前预先缓冲的秒数。

**提醒**

尽管您可以为 `TextField` 对象指定此属性，但它实际上是一个应用于所有已加载声音的全局属性，因此您只能将它的值指定为 `_soundbuftime`。为 `TextField` 对象设置此属性实际上是设置全局属性。

有关更多信息和示例，请参见 `_soundbuftime` 全局属性。

可用性：ActionScript 1.0；Flash Player 6

另请参见

[\\_soundbuftime 属性](#)



## styleSheet (TextField.styleSheet 属性)

```
public styleSheet : StyleSheet
```

将样式表附加到文本字段。有关创建样式表的信息，请参见 [TextField.StyleSheet](#) 类条目。

可以随时更改与文本字段关联的样式表。如果正在使用的样式表发生更改，则会用新样式表重绘文本字段。可以将样式表设置为 `null` 或 `undefined` 来删除样式表。如果删除正在使用的样式表，则不使用样式表重绘文本字段。如果样式表被删除，则不会保留用该样式表设置的格式。

可用性: **ActionScript 1.0 ; Flash Player 7**

### 示例

下面的示例在运行时创建一个名为 `news_txt` 的新文本字段。舞台上的三个按钮 `css1_btn`、`css2_btn` 和 `clearCss_btn` 用于更改应用于 `news_txt` 的样式表，或从文本字段中清除样式表。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
this.createTextField("news_txt", this.getNextHighestDepth(), 0, 0, 300,
    200);
news_txt.wordWrap = true;
news_txt.multiline = true;
news_txt.html = true;
var newsText:String = "<p class='headline'>Description</p> Method; "
    + "starts loading the CSS file into styleSheet. The load operation is
    asynchronous; "
    + "use the <span class='bold'>TextField.StyleSheet.onLoad</span> "
    + "callback handler to determine when the file has finished loading. "
    + "<span class='important'>The CSS file must reside in exactly the same "
    + "domain as the SWF file that is loading it.</span> For more information
    about "
    + "restrictions on loading data across domains, see Flash Player security
    features.";

news_txt.htmlText = newsText;

css1_btn.onRelease = function() {
    var styleObj:TextField.StyleSheet = new TextField.StyleSheet();
    styleObj.onLoad = function(success:Boolean) {
        if (success) {
            news_txt.styleSheet = styleObj;
            news_txt.htmlText = newsText;
        }
    };
    styleObj.load("styles.css");
};

css2_btn.onRelease = function() {
    var styleObj:TextField.StyleSheet = new TextField.StyleSheet();
```

```

styleObj.onload = function(success:Boolean) {
    if (success) {
        news_txt.styleSheet = styleObj;
        news_txt.htmlText = newsText;
    }
};
styleObj.load("styles2.css");
};

clearCss_btn.onRelease = function() {
    news_txt.styleSheet = undefined;
    news_txt.htmlText = newsText;
};

```

下面的样式已应用于文本字段。将以下两个 **CSS** 文件保存在与以前创建的 **FLA** 或 **AS** 文件相同的目录中：

```

// in styles.css
.important {
    color: #FF0000;
}
.bold {
    font-weight: bold;
}
.headline {
    color: #000000;
    font-family: Arial,Helvetica,sans-serif;
    font-size: 18px;
    font-weight: bold;
    display: block;
}

// in styles2.css
.important {
    color: #FF00FF;
}
.bold {
    font-weight: bold;
}
.headline {
    color: #00FF00;
    font-family: Arial,Helvetica,sans-serif;
    font-size: 18px;
    font-weight: bold;
    display: block;
}

```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[StyleSheet \(TextField.StyleSheet\)](#)

## tabEnabled (TextField.tabEnabled 属性)

```
public tabEnabled : Boolean
```

指定文本字段是否包括在 **Tab** 键的自动排序中。它默认情况下为 `undefined`。

如果 `tabEnabled` 属性为 `undefined` 或 `true`，则该对象包括在 **Tab** 键的自动排序中。如果 `tabIndex` 属性也设置为某个值，则该对象也包括在 **Tab** 键的自定义排序中。如果 `tabEnabled` 为 `false`，则即使设置了 `tabIndex` 属性，该对象也不包括在 **Tab** 键的自动或自定义排序中。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例创建名为 `one_txt`、`two_txt`、`three_txt` 和 `four_txt` 的多个文本字段。`three_txt` 文本字段已将 `tabEnabled` 属性设置为 `false`，因此已从 **Tab** 键的自动排序中排除它。

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100,
    22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100,
    22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100,
    22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100,
    22);
four_txt.border = true;
four_txt.type = "input";

three_txt.tabEnabled = false;
three_txt.text = "tabEnabled = false;";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 `DepthManager` 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[tabEnabled \(Button.tabEnabled 属性\)](#)，[tabEnabled \(MovieClip.tabEnabled 属性\)](#)

## tabIndex (TextField.tabIndex 属性)

```
public tabIndex : Number
```

用于自定义 SWF 文件中对象的 **Tab** 键排序。可以设置按钮、影片剪辑或文本字段实例的 tabIndex 属性；默认情况下该属性为 undefined。

如果 SWF 文件中当前显示的任何对象包含 tabIndex 属性，则禁用 **Tab** 键的自动排序，而使用该 SWF 文件中对象的 tabIndex 属性来计算 **Tab** 键排序。这个自定义的 **Tab** 键排序仅包括具有 tabIndex 属性的对象。

tabIndex 属性必须是一个正整数。这些对象按照其 tabIndex 属性按升序进行排序。tabIndex 值为 1 的对象在 tabIndex 值为 2 的对象的前面。如果两个对象的 tabIndex 值相同，则在 **Tab** 键排序中在另一个对象前面的对象为 undefined。

由 tabIndex 属性定义的 **Tab** 键的自定义排序是平构的。这意味着不考虑 SWF 文件中对象的层次结构关系。SWF 文件中具有 tabIndex 属性的所有对象都排入 **Tab** 键顺序中，而 **Tab** 键顺序由 tabIndex 值的顺序确定。如果两个对象具有相同的 tabIndex 值，则先出现的对象为 undefined。多个对象不应使用相同的 tabIndex 值。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的 **ActionScript** 动态创建四个文本字段，并将它们分配给自定义 **Tab** 键顺序。请将以下 **ActionScript** 添加到 FLA 或 AS 文件：

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100,
    22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100,
    22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100,
    22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100,
    22);
four_txt.border = true;
four_txt.type = "input";

one_txt.tabIndex = 3;
two_txt.tabIndex = 1;
three_txt.tabIndex = 2;
four_txt.tabIndex = 4;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

`tabIndex` (`Button.tabIndex` 属性), `tabIndex` (`MovieClip.tabIndex` 属性)

## `_target` (`TextField._target` 属性)

```
public _target : String [read-only]
```

文本字段实例的目标路径。`_self` 目标指定当前窗口中的当前帧，`_blank` 指定一个新窗口，`_parent` 指定当前帧的父级，而 `_top` 指定当前窗口中的顶级帧。

可用性: **ActionScript 1.0** ; **Flash Player 6**

示例

下面的 **ActionScript** 创建一个名为 `my_txt` 的文本字段，并同时以斜杠记号和点记号输出新字段的目标路径。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);  
trace(my_txt._target); // output: /my_txt  
trace(eval(my_txt._target)); // output: _level0.my_txt
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## `text` (`TextField.text` 属性)

```
public text : String
```

指示文本字段中的当前文本。行用回车符（“`\r`”，即 **ASCII 13**）分隔。此属性包含文本字段中普通的无格式文本，不包含 **HTML** 标签，即使该文本字段为 **HTML**。

可用性: **ActionScript 1.0** ; **Flash Player 6**

示例

下面的示例创建一个名为 `my_txt` 的 **HTML** 文本字段，并将 **HTML** 格式的文本字符串分配给该字段。当您跟踪 `htmlText` 属性时，“输出”面板将显示 **HTML** 格式的字符串。当您跟踪 `text` 属性的值时，未经过格式化并带有 **HTML** 标记的字符串将显示在“输出”面板中。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 400, 22);  
my_txt.html = true;  
my_txt.htmlText = "<B>Lorem ipsum dolor sit amet.</B>";
```

```
trace("htmlText: "+my_txt.htmlText);
trace("text: "+my_txt.text);
```

它生成以下输出:

```
htmlText: <P ALIGN="LEFT"><FONT FACE="Times New Roman" SIZE="12"
  COLOR="#000000" KERNING="0">
  <B>Lorem ipsum dolor sit amet.</B></FONT></P>
text: Lorem ipsum dolor sit amet.
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[htmlText \(TextField.htmlText 属性\)](#)

## textColor (TextField.textColor 属性)

```
public textColor : Number
```

指示文本字段中文本的颜色。十六进制颜色系统使用六位数表示颜色值。每位数有十六个可能的值或字符。字符的范围从 0 到 9, 然后从 A 到 F。黑色用 (#000000) 表示, 而白色用颜色系统的相对末端值 (#FFFFFF) 表示。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的 **ActionScript** 创建一个文本字段, 并将其 **color** 属性更改为 **red**。

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);
my_txt.text = "this will be red text";
my_txt.textColor = 0xFF0000;
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## textHeight (TextField.textHeight 属性)

public textHeight : Number

指示文本的高度。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例创建一个文本字段，并将文本字符串分配给该字段。**trace** 语句用于在“输出”面板中显示文本高度和宽度。`autoSize` 属性随后用于调整文本字段的大小，而且新的高度和宽度也将显示在“输出”面板中。

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);
my_txt.text = "Sample text";
trace("textHeight: "+my_txt.textHeight+", textWidth: "+my_txt.textWidth);
trace("_height: "+my_txt._height+", _width: "+my_txt._width+"\n");
my_txt.autoSize = true;
trace("after my_txt.autoSize = true;");
trace("_height: "+my_txt._height+", _width: "+my_txt._width);
```

它输出以下信息:

```
textHeight: 15, textWidth: 56
_height: 300, _width: 100
```

```
after my_txt.autoSize = true;
_height: 19, _width: 60
```

### 另请参见

[textWidth \(TextField.textWidth 属性\)](#)

## textWidth (TextField.textWidth 属性)

public textWidth : Number

指示文本的宽度。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

请参见 `TextField.textHeight` 的示例。

### 另请参见

[textHeight \(TextField.textHeight 属性\)](#)

## thickness (TextField.thickness 属性)

public thickness : Number

此 **TextField** 实例中字型边缘的粗细。仅在 `antiAliasType()` 设置为 "advanced" 时才可应用此属性。

thickness 的范围是从 -200 到 200 的数字。如果您尝试将 thickness 设置为该范围外的值，则该属性会设置为范围内最接近的值（-200 或 200）。

可用性：ActionScript 1.0；Flash Player 8

### 示例

本示例创建两个文本字段，并将一个字段的 thickness 设置为 -200，而将另一个字段的设置为 200。假定您已将字体嵌入到链接标识符设置为 "Times-12" 的库中。若要嵌入字体，请按照下列步骤操作：

- 打开您的库
- 单击该库右上角的“库”选项菜单
- 从下拉列表中选择“新建字体”
- 将字体命名为“Times-12”
- 从字体下拉列表中选择“Times New Roman”
- 按“确定”按钮
- 右键单击新创建的字体并选择“链接...”
- 选中“为 ActionScript 导出”框
- 通过按“确定”按钮接受默认标识符“Times-12”

```
var my_format:TextFormat = new TextFormat();  
my_format.font = "Times-12";
```

```
var my_text1:TextField = this.createTextField("my_text1",  
    this.getNextHighestDepth(), 10, 10, 300, 30);  
my_text1.text = "thickness = 200";  
my_text1.antiAliasType = "advanced";  
my_text1.border = true;  
my_text1.thickness = 200;  
my_text1.embedFonts = true;  
my_text1.setTextFormat(my_format);
```

```
var my_text2:TextField = this.createTextField("my_text2",  
    this.getNextHighestDepth(), 10, 50, 300, 30);  
my_text2.text = "thickness = -200.";  
my_text2.antiAliasType = "advanced";  
my_text2.thickness = -200;  
my_text2.border = true;  
my_text2.embedFonts = true;
```



```
my_text2.setTextFormat(my_format);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[antiAliasType](#) (`TextField.antiAliasType` 属性)

## type (TextField.type 属性)

```
public type : String
```

指定文本字段的类型。有两种值: "dynamic", (指定不能由用户对其进行编辑的动态文本字段) 和 "input" (指定输入文本字段)。

可用性: **ActionScript 1.0** ; **ActionScript 1.0** ; **Flash Player 6**

### 示例

以下示例创建两个文本字段: `username_txt` 和 `password_txt`。文本被输入到这两个字段文本中; 但是, `password_txt` 的 `password` 属性会设置为 `true`。因此, 字符显示为星号, 而不是 `password_txt` 字段中的字符。

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10,
    100, 22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";
```

```
this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40,
    100, 22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_txt.password = true;
password_txt.text = "world";
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

## \_url (TextField.\_url 属性)

public \_url : String [read-only]

检索创建文本字段的 SWF 文件的 URL。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例检索创建文本字段的 SWF 文件的 URL，以及加载到其中的 SWF 文件。

```
this.createTextField("my_txt", 1, 10, 10, 100, 22);
trace(my_txt._url);

var mcListener:Object = new Object();
mcListener.onLoadInit = function(target_mc:MovieClip) {
    trace(target_mc._url);
};
var holder_mc1:MovieClipLoader = new MovieClipLoader();
holder_mc1.addListener(mcListener);
holder_mc1.loadClip("best_flash_ever.swf",
    this.createEmptyMovieClip("holder_mc", 2));
```

当您测试此示例时，您正在测试的 SWF 文件的 URL 和名为 `best_flash_ever.swf` 的文件会显示在“输出”面板中。

此示例中使用的 `MovieClipLoader` 类需要 **Flash Player 7** 或更高版本。

## variable (TextField.variable 属性)

public variable : String

与文本字段关联的变量的名称。此属性的类型为 `String`。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例创建一个名为 `my_txt` 的文本字段，并使变量 `today_date` 与该文本字段相关联。

当您更改变量 `today_date` 时，将更新 `my_txt` 中显示的文本。

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.variable = "today_date";
var today_date:Date = new Date();

var date_interval:Number = setInterval(updateDate, 500);
function updateDate():Void {
    today_date = new Date();
}
```

## `_visible` (TextField.`_visible` 属性)

`public _visible : Boolean`

一个布尔值，指示文本字段 `my_txt` 是否可见。禁用不可见的文本字段 (`_visible` 属性设置为 `false`)。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个名为 `my_txt` 的文本字段。名为 `visible_btn` 的按钮切换 `my_txt` 的可见性。

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.background = true;
my_txt.backgroundColor = 0xDFDFDF;
my_txt.border = true;
my_txt.type = "input";

visible_btn.onRelease = function() {
    my_txt._visible = !my_txt._visible;
};
```

### 另请参见

[\\_visible \(Button.\\_visible 属性\)](#), [\\_visible \(MovieClip.\\_visible 属性\)](#)

## `_width` (TextField.`_width` 属性)

`public _width : Number`

文本字段的宽度，以像素为单位。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建两个文本字段，您可以使用这两个字段更改舞台上第三个文本字段的宽度和高度。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件。

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 40, 160,
    120);
my_txt.background = true;
my_txt.backgroundColor = 0xFF0000;
my_txt.border = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.wordWrap = true;
```

```

this.createTextField("width_txt", this.getNextHighestDepth(), 10, 10, 30,
    20);
width_txt.border = true;
width_txt.maxChars = 3;
width_txt.restrict = "0-9";
width_txt.type = "input";
width_txt.text = my_txt._width;
width_txt.onChanged = function() {
    my_txt._width = this.text;
}

this.createTextField("height_txt", this.getNextHighestDepth(), 70, 10, 30,
    20);
height_txt.border = true;
height_txt.maxChars = 3;
height_txt.restrict = "0-9";
height_txt.type = "input";
height_txt.text = my_txt._height;
height_txt.onChanged = function() {
    my_txt._height = this.text;
}

```

当您测试示例时，请尝试向 width\_txt 和 height\_txt 输入新值以更改 my\_txt 的尺寸。

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 MovieClip.getNextHighestDepth() 方法。

另请参见

[\\_height \(TextField.\\_height 属性\)](#)

## wordWrap (TextField.wordWrap 属性)

```
public wordWrap : Boolean
```

一个布尔值，指示文本字段是否自动换行。如果 wordWrap 的值为 true，则该文本字段自动换行；如果值为 false，则该文本字段不自动换行。

可用性：ActionScript 1.0；Flash Player 6

示例

下面的示例演示 wordWrap 如何影响在运行时创建的文本字段中的长文本。

```

this.createTextField("my_txt", 99, 10, 10, 100, 200);
my_txt.text = "This is very long text that will certainly extend beyond the
    width of this text field";
my_txt.border = true;

```

通过在 **Flash Player** 中选择“控制”>“测试影片”对 **SWF** 文件进行测试。然后返回到 **ActionScript**，将以下行添加到代码并再次测试 **SWF** 文件：

```
my_txt.wordWrap = true;
```

## **\_x** (**TextField.\_x** 属性)

```
public _x : Number
```

一个整数，用来设置文本字段相对于父级影片剪辑的本地坐标的 **x** 坐标。如果文本字段在时间轴上，则其坐标系统将舞台的左上角作为 **(0, 0)**。如果文本字段在具有变形的影片剪辑内，则该文本字段位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 **90** 度的影片剪辑，其中的文本字段将继承逆时针旋转 **90** 度的坐标系统。文本字段的坐标指的是注册点的位置。

可用性：**ActionScript 1.0**；**Flash Player 6**

### 示例

下面的示例当您在任意位置单击鼠标时创建一个文本字段。它创建文本字段时，该字段显示文本字段的当前 **x** 和 **y** 坐标。

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 60,
    22);
coords_txt.autoSize = true;
coords_txt.selectable = false;
coords_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
    coords_txt.text = "X:"+Math.round(_xmouse)+"", Y:"+Math.round(_ymouse);
    coords_txt._x = _xmouse;
    coords_txt._y = _ymouse;
};
Mouse.addListener(mouseListener);
```

此示例中使用的 **MovieClip.getNextHighestDepth()** 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 **MovieClip.getNextHighestDepth()** 方法。

### 另请参见

[\\_xscale](#) ([TextField.\\_xscale](#) 属性)，[\\_y](#) ([TextField.\\_y](#) 属性)，[\\_yscale](#) ([TextField.\\_yscale](#) 属性)

## `_xmouse` (`TextField._xmouse` 属性)

`public _xmouse : Number [read-only]`

返回鼠标位置相对于文本字段的 `x` 坐标。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例在舞台上创建三个文本字段。`mouse_txt` 实例显示鼠标相对于舞台的当前位置。`textfield_txt` 实例显示鼠标指针相对于 `my_txt` 实例的当前位置。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件:

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 10, 10, 200, 22);
mouse_txt.border = true;
this.createTextField("textfield_txt", this.getNextHighestDepth(), 220, 10, 200, 22);
textfield_txt.border = true;
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100, 160, 120);
my_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
    mouse_txt.text = "MOUSE ... X:" + Math.round(_xmouse) + ",\tY:" + Math.round(_ymouse);
    textfield_txt.text = "TEXTFIELD ... X:" + Math.round(my_txt._xmouse) + ",\tY:" + Math.round(my_txt._ymouse);
}
```

```
MovieClip.addListener(mouseListener);
```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件, 请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[\\_ymouse](#) (`TextField._ymouse` 属性)

## `_xscale` (TextField.\_xscale 属性)

public `_xscale` : Number

确定从文本字段注册点开始应用的文本字段的水平缩放比例，以百分比表示。默认注册点为 (0,0)。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例在您单击 `scaleUp_btn` 和 `scaleDown_btn` 实例时缩放 `my_txt` 实例。

```
this.createTextField("my_txt", 99, 10, 40, 100, 22);
my_txt.autoSize = true;
my_txt.border = true;
my_txt.selectable = false;
my_txt.text = "Sample text goes here.";
```

```
scaleUp_btn.onRelease = function() {
    my_txt._xscale = 2;
    my_txt._yscale = 2;
}
scaleDown_btn.onRelease = function() {
    my_txt._xscale /= 2;
    my_txt._yscale /= 2;
}
```

### 另请参见

[\\_x](#) (TextField.\_x 属性), [\\_y](#) (TextField.\_y 属性), [\\_yscale](#) (TextField.\_yscale 属性)

## `_y` (TextField.\_y 属性)

`public _y : Number`

文本字段相对于父级影片剪辑的本地坐标的 `y` 坐标。如果文本字段在主时间轴中，则其坐标系将舞台的左上角作为 (0, 0)。如果文本字段在具有变形的影片剪辑内，则该文本字段位于包含它的影片剪辑的本地坐标系中。因此，对于逆时针旋转 90 度的影片剪辑，其中的文本字段将继承逆时针旋转 90 度的坐标系。文本字段的坐标指的是注册点的位置。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

请参见 `TextField._x` 的示例。

### 另请参见

[\\_x](#) (TextField.\_x 属性), [\\_xscale](#) (TextField.\_xscale 属性), [\\_yscale](#) (TextField.\_yscale 属性)

## `_ymouse` (TextField.\_ymouse 属性)

`public _ymouse : Number [read-only]`

指示鼠标位置相对于文本字段的 `y` 坐标。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

请参见 `TextField._xmouse` 的示例。

### 另请参见

[\\_xmouse](#) (TextField.\_xmouse 属性)



## `_yscale` (`TextField._yscale` 属性)

```
public _yscale : Number
```

从文本字段的注册点开始应用的文本字段的垂直缩放比例，以百分比表示。默认注册点为 (0,0)。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

请参见 `TextField._xscale` 的示例。

### 另请参见

[\\_x \(TextField.\\_x 属性\)](#), [\\_xscale \(TextField.\\_xscale 属性\)](#), [\\_y \(TextField.\\_y 属性\)](#)

# TextFormat

```
Object
|
+-TextFormat
```

```
public class TextFormat
extends Object
```

**TextFormat** 类描述字符格式设置信息。使用 **TextFormat** 类可以为文本字段创建特定的文本格式。您可以将文本格式应用于静态文本字段和动态文本字段。**TextFormat** 类的一些属性并不是对于嵌入字体和设备字体均可用。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 另请参见

[setTextFormat \(TextField.setTextFormat 方法\)](#), [getTextFormat \(TextField.getTextFormat 方法\)](#)

属性摘要

修饰符	属性	说明
	align:String	指示段落的对齐方式的字符串。
	blockIndent:Number	以磅为单位指示块缩进的数字。
	bold:Boolean	一个布尔值，指示文本是否为粗体字。
	bullet:Boolean	一个布尔值，指示文本为带项目符号的列表的一部分。
	color:Number	指示文本的颜色。
	font:String	使用此文本格式的文本的字体名称，以字符串形式表示。
	indent:Number	指示从左边距到段落中第一个字符的缩进的整数。
	italic:Boolean	一个布尔值，指示使用此文本格式的文本是否为斜体。
	kerning:Boolean	一个布尔值，指示是启用还是禁用字距调整。
	leading:Number	一个整数，表示以像素为单位的行间垂直距离（称为行距）。
	leftMargin:Number	段落的左边距，以磅为单位。
	letterSpacing:Number	两个字符之间统一分布的距离。
	rightMargin:Number	段落的右边距，以磅为单位。
	size:Number	使用此文本格式的文本的磅值。
	tabStops:Array	将自定义 Tab 键停靠位指定为一个非负整数的数组。
	target:String	指示显示超链接的目标窗口。
	underline:Boolean	一个布尔值，指示使用此文本格式的文本有下划线 (true) 还是没有下划线 (false)。
	url:String	指示使用此文本格式的文本链接所至的 URL。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

构造函数摘要

签名	说明
<code>TextFormat([font:String], [size:Number], [color:Number], [bold:Boolean], [italic:Boolean], [underline:Boolean], [url:String], [target:String], [align:String], [leftMargin:Number], [rightMargin:Number], [indent:Number], [leading:Number])</code>	创建一个具有指定属性的 TextFormat 对象。

方法摘要

修饰符	签名	说明
	<code>getTextExtent(text:String, [width:Number]) : Object</code>	自 Flash Player 8 后不推荐使用。没有其它替换方法。返回格式由 my_fmt 指定的文本字符串 text 的文本度量信息。

继承自 Object 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## align (TextFormat.align 属性)

public align : String

指示段落的对齐方式的字符串。您可以将此属性应用于静态文本和动态文本。下面的列表显示此属性的可能值：

- "left" - 段落为左对齐。
- "center" - 段落居中。
- "right" - 段落为右对齐。
- "justify" - 段落为右对齐。(Flash Player 8 中添加了此值。)

默认值为 null，它指示该属性未定义。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例说明 align 属性将设置为两端对齐，这使得每行中的字符可以展开，以便文本在水平上看起来分布更均匀。

```
var format:TextFormat = new TextFormat();
format.align = "justify";

var txtField:TextField = this.createTextField("txtField",
    this.getNextHighestDepth(), 100, 100, 300, 100);
txtField.multiline = true;
txtField.wordWrap = true;
txtField.border = true;
txtField.text = "When this text is justified, it will be "
    + "spread out to more cleanly fill the horizontal "
    + "space for each line. This can be considered an "
    + "improvement over regular left-aligned text that "
    + "will simply wrap and do no more.";
txtField.setTextFormat(format);
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 Flash Player 7 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

## blockIndent (TextFormat.blockIndent 属性)

public blockIndent : Number

以磅为单位指示块缩进的数字。块缩进应用于整个文本块，即文本的所有行。而普通缩进 (TextFormat.indent) 只影响各段的第一行。如果此属性为 null，则 TextFormat 对象不指定块缩进。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

此示例创建一个带边框的文本字段并将 blockIndent 设置为 20。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.blockIndent = 20;

mytext.text = "This is my first test field object text";
mytext.setTextFormat(myformat);
```

## bold (TextFormat.bold 属性)

public bold : Boolean

一个布尔值，指示文本是否为粗体字。默认值为 null，它指示该属性未定义。如果值为 true，则文本为粗体字。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

下面的示例创建一个包括粗体字符的文本字段。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "This is my test field object text";
my_txt.setTextFormat(my_fmt);
```

## bullet (TextFormat.bullet 属性)

public bullet : Boolean

一个布尔值，指示文本为带项目符号的列表的一部分。在带项目符号的列表中，文本的各段都是缩进的。项目符号显示在各段第一行的左侧。默认值为 null。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

下面的示例在运行时创建一个新的文本字段，并向该字段输入包含换行符的字符串。

**TextFormat** 类用于通过向文本字段中的每行添加项目符号来设置字符的格式。以下

**ActionScript** 中演示了这一点：

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bullet = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my text"+newline;
my_txt.text += "this is more text"+newline;
my_txt.setTextFormat(my_fmt);
```

## color (TextFormat.color 属性)

public color : Number

指示文本的颜色。包含三个 8 位 RGB 颜色成分的数字；例如，0xFF0000 为红色，0x00FF00 为绿色。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

下面的示例创建一个文本字段并将文本颜色设置为红色。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.blockIndent = 20;
my_fmt.color = 0xFF0000; // hex value for red

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first test field object text";
my_txt.setTextFormat(my_fmt);
```

## font (TextFormat.font 属性)

```
public font : String
```

使用此文本格式的文本的字体名称，以字符串形式表示。默认值为 `null`，它指示该属性未定义。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个文本字段并将字体设置为 **Courier**。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.font = "Courier";

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

## getTextExtent (TextFormat.getTextExtent 方法)

```
public getTextExtent(text:String, [width:Number]) : Object
```

自 **Flash Player 8** 后不推荐使用。没有其它替换方法。

返回格式由 `my_fmt` 指定的文本字符串 `text` 的文本度量信息。文本字符串被视为纯文本（而不是 **HTML**）。

此方法返回一个具有以下六个属性的对象：`ascent`、`descent`、`width`、`height`、`textFieldHeight` 和 `textFieldWidth`。所有度量值均以像素为单位。

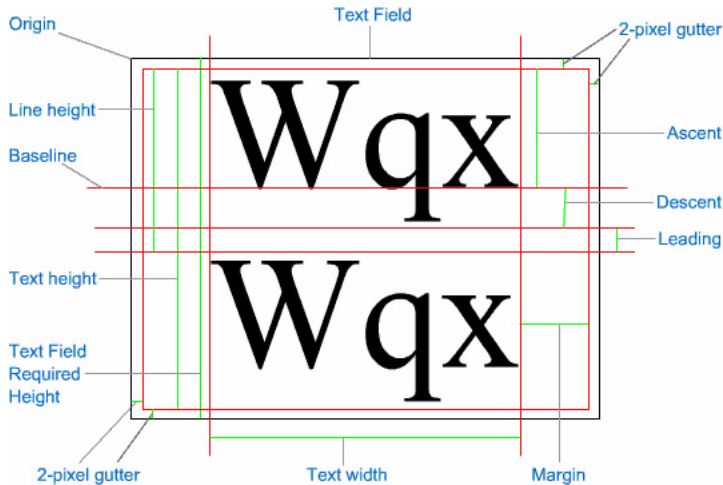
如果指定了 `width` 参数，则指定的文本将自动换行。这样，您就可以确定文本框在显示所有指定的文本时，它的高度是多少。

`ascent` 和 `descent` 度量值分别提供一行文本的基线上面和下面的距离。第一行文本的基线位于该文本字段的原点加上其 `ascent` 度量值的位置。

`width` 和 `height` 度量值提供文本字符串的宽度和高度。`textFieldHeight` 和 `textFieldWidth` 度量值提供文本字段对象显示整个文本字符串所需要具有的高度和宽度。文本字段的周围具有 2 个像素宽的装订线，所以 `textFieldHeight` 的值等于 `height` 的值加 4；同理，`textFieldWidth` 的值始终等于 `width` 的值加 4。

如果您基于文本度量创建文本字段，请使用 `textFieldHeight` 而非 `height`，并使用 `textFieldWidth` 而非 `width`。

下图中介绍这些度量值。



设置 `TextFormat` 对象时，所有属性设置必须与创建文本字段时所设置的属性完全一样，包括字体名称、字体大小和前导间距量。前导间距量的默认值为 2。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**text:**String - 一个字符串。

**width:**Number [ 可选 ] - 一个表示宽度的数字（以像素为单位），指定的文本应在该处换行。

### 返回

Object - 一个对象，具有以下属性：width、height、ascent、descent、textFieldHeight、textFieldWidth。

### 示例

此示例创建一个单行文本字段，其大小足够使用指定格式显示一个文本字符串。

```
var my_str:String = "Small string";

// Create a TextFormat object,
// and apply its properties.
var my_fmt:TextFormat = new TextFormat();
with (my_fmt) {
    font = "Arial";
    bold = true;
}
```



```
// Obtain metrics information for the text string
// with the specified formatting.
var metrics:Object = my_fmt.getTextExtent(my_str);

// Create a text field just large enough to display the text.
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100,
    metrics.textFieldWidth,
    metrics.textFieldHeight);
my_txt.border = true;
my_txt.wordWrap = true;
// Assign the same text string and TextFormat object to the my_txt object.
my_txt.text = my_str;
my_txt.setTextFormat(my_fmt);
```

下面的示例创建一个 **100** 像素宽的多行文本字段，其高度足够使用指定的格式显示一个字符串。

```
// Create a TextFormat object.
var my_fmt:TextFormat = new TextFormat();
// Specify formatting properties for the TextFormat object:
my_fmt.font = "Arial";
my_fmt.bold = true;
my_fmt.leading = 4;

// The string of text to be displayed
var textToDisplay:String = "Macromedia Flash Player 7, now with improved text
    metrics.";

// Obtain text measurement information for the string,
// wrapped at 100 pixels.
var metrics:Object = my_fmt.getTextExtent(textToDisplay, 100);

// Create a new TextField object using the metric
// information just obtained.
this.createTextField("my_txt", this.getNextHighestDepth(), 50, 50-
    metrics.ascent, 100,
    metrics.textFieldHeight);
my_txt.wordWrap = true;
my_txt.border = true;
// Assign the text and the TextFormat object to the TextObject:
my_txt.text = textToDisplay;
my_txt.setTextFormat(my_fmt);
```

## indent (TextFormat.indent 属性)

public indent : Number

指示从左边距到段落中第一个字符的缩进的整数。正值指示普通缩进。您可以使用负值，但只可在左边距大于 0 时进行负缩进。若要将边距设置为大于 0，请使用 indent 属性或 TextFormat 对象的 blockIndent 属性。默认值为 null，它指示该属性未定义。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个文本字段并将缩进设置为 10:

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.indent = 10;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

另请参见

[blockIndent \(TextFormat.blockIndent 属性\)](#)

## italic (TextFormat.italic 属性)

public italic : Boolean

一个布尔值，指示使用此文本格式的文本是否为斜体。默认值为 null，它指示该属性未定义。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个文本字段并将文本样式设置为斜体。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.italic = true;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

## Kerning (TextFormat.kerning 属性)

public kerning : Boolean

一个布尔值，指示是启用还是禁用字距调整。字距调整在某些字符对之间留出预定的距离以提高可读性。默认值为 false，它指示禁用字距调整。

仅嵌入字体支持字距调整。某些字体（如 **Courier New**）不支持字距调整。

只有在 **Windows** 中创建的 **SWF** 文件中支持 kerning 属性，而在 **Macintosh** 上创建的 **SWF** 文件中不支持此属性。但是，**Windows SWF** 文件可以在 **Flash Player** 的非 **Windows** 版本中播放，而且仍可以应用字距调整。

请只在必要时（例如使用大字体标题时）使用字距调整。

**可用性:** ActionScript 1.0 ; Flash Player 8

### 示例

下面的示例说明两个文本字段：第一个文本字段使用的格式为红色文本且 kerning 设置为 false，而第二个文本字段使用的格式为蓝色文本且 kerning 设置为 true。若要使用此示例，您可以将字体元件添加到“库”中，然后选择“**Arial**”作为字体。在该字体的“链接属性”对话框中，将“标识符”名称设置为“Font 1”，选中“为 **ActionScript** 导出”，然后选择“在第一帧导出”。

```
var fmt1:TextFormat = new TextFormat();
fmt1.font = "Font 1";
fmt1.size = 50;
fmt1.color = 0xFF0000;
fmt1.kerning = false;
```

```
var fmt2:TextFormat = new TextFormat();
fmt2.font = "Font 1";
fmt2.size = 50;
fmt2.color = 0x0000FF;
fmt2.kerning = true;
```

```
this.createTextField("tf1", this.getNextHighestDepth(), 10, 10, 400, 100);
tf1.embedFonts = true;
tf1.text = "Text 7AVA-7AVA";
tf1.setTextFormat(fmt1);
```

```
this.createTextField("tf2", this.getNextHighestDepth(), 10, 40, 400, 100);
tf2.embedFonts = true;
tf2.text = tf1.text;
tf2.setTextFormat(fmt2);
```

如果您的 **SWF** 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 **MovieClip.getNextHighestDepth()** 方法。

## leading（TextFormat.leading 属性）

public leading : Number

一个整数，表示以像素为单位的行间垂直距离（称为行距）。默认值为 null，它指示该属性未定义。

**Flash Player 8** 支持负行距，表示行间距离小于文本高度。在希望将文本行（如标题）非常近地放置在一起时，负行距会很有用。若要防止文本重叠，请对不包含下行字母的文本（如全部为大写的文本）行使用负行距。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个文本字段并将行距设置为 10。

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.leading = 10;

this.createTextField("my_txt", 1, 100, 100, 100, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "This is my first text field object text";
my_txt.setTextFormat(my_fmt);
```

## leftMargin（TextFormat.leftMargin 属性）

public leftMargin : Number

段落的左边距，以磅为单位。默认值为 null，它指示该属性未定义。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个文本字段并将左边距设置为 20 磅。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.leftMargin = 20;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

## letterSpacing (TextFormat.letterSpacing 属性)

public letterSpacing : Number

两个字符之间统一分布的距离。该数值指定添加到每个字符后的间隔中的像素数。负值将压缩两个字符之间的距离。

系统字体仅支持整数值；但是，您可以为嵌入字体指定浮点（非整数）值（如 2.6）。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例使用两个 **TextFormat** 对象将 letterSpacing 的正值和负值分别应用于文本字段中不同范围的文本。

```
this.createTextField("mytext", this.getNextHighestDepth(), 10, 10, 200,
    100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var format1:TextFormat = new TextFormat();
format1.letterSpacing = -1;

var format2:TextFormat = new TextFormat();
format2.letterSpacing = 10;

mytext.text = "Eat at \nJOE'S.";
mytext.setTextFormat(0, 7, format1);
mytext.setTextFormat(8, 12, format2);
```

如果您的 SWF 文件包括第 2 版的组件，则请使用第 2 版的组件 **DepthManager** 类取代本示例中所用的 `MovieClip.getNextHighestDepth()` 方法。

## rightMargin（TextFormat.rightMargin 属性）

public rightMargin : Number

段落的右边距，以磅为单位。默认值为 null，它指示该属性未定义。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个文本字段并将右边距设置为 20 磅。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.rightMargin = 20;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

## size（TextFormat.size 属性）

public size : Number

使用此文本格式的文本的磅值。默认值为 null，它指示该属性未定义。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例创建一个文本字段并将文本大小设置为 20 磅。

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.size = 20;

mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

## tabStops (TextFormat.tabStops 属性)

public tabStops : Array

将自定义 **Tab** 键停靠位指定为一个非负整数的数组。以像素为单位指定每个 **Tab** 键的停靠位。如果没有指定自定义 **Tab** 键停靠位 (null)，则默认的 **Tab** 键停靠位为 **4** (平均字符宽度)。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例创建两个文本字段，一个文本字段每隔 **40** 像素有一个 **Tab** 键停靠位，而另一个每隔 **75** 像素有一个 **Tab** 键停靠位。

```
this.createTextField("mytext",1,100,100,400,100);
mytext.border = true;
var myformat:TextFormat = new TextFormat();
myformat.tabStops = [40,80,120,160];
mytext.text = "A\tB\tC\tD"; // \t is the tab stop character
mytext.setTextFormat(myformat);
```

```
this.createTextField("mytext2",2,100,220,400,100);
mytext2.border = true;
var myformat2:TextFormat = new TextFormat();
myformat2.tabStops = [75,150,225,300];
mytext2.text = "A\tB\tC\tD";
mytext2.setTextFormat(myformat2);
```

## target (TextFormat.target 属性)

public target : String

指示显示超链接的目标窗口。如果目标窗口为空字符串，则文本显示在默认目标窗口 `_self` 中。您可以选择一个自定义名称或下面四个名称中的一个: `_self` 指定当前窗口中的当前帧，`_blank` 指定一个新窗口，`_parent` 指定当前帧的父级，而 `_top` 指定当前窗口中的顶级帧。如果 `TextFormat.url` 属性是空字符串或 `null`，则虽然您可以获取或设置此属性，但该属性不起作用。

可用性: **ActionScript 1.0 ; Flash Player 6**

### 示例

下面的示例创建一个带有指向 **Macromedia** 网站的超链接的文本字段。该示例使用 `TextFormat.target` 在新浏览器窗口中显示 **Macromedia** 网站。

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.macromedia.com";
myformat.target = "_blank";
```

```
this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Macromedia.com";
mytext.setTextFormat(myformat);
```

另请参见

[url](#) ([TextFormat.url](#) 属性)

## TextFormat 构造函数

```
public TextFormat([font:String], [size:Number], [color:Number],
    [bold:Boolean], [italic:Boolean], [underline:Boolean], [url:String],
    [target:String], [align:String], [leftMargin:Number],
    [rightMargin:Number], [indent:Number], [leading:Number])
```

创建一个具有指定属性的 **TextFormat** 对象。然后可更改 **TextFormat** 对象的属性以更改文本字段的格式设置。

任何参数都可设置为 `null` 以指示该参数未定义。所有参数都是可选的；任何省略的参数都被视为 `null`。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 参数

**font:String** [ 可选 ] - 以字符串形式表示的文本字体名称。

**size:Number** [ 可选 ] - 指示磅值的整数。

**color:Number** [ 可选 ] - 使用此文本格式的文本的颜色。包含三个 8 位 RGB 颜色成分的数字；例如，**0xFF0000** 为红色，**0x00FF00** 为绿色。

**bold:Boolean** [ 可选 ] - 指示文本是否为粗体的布尔值。

**italic:Boolean** [ 可选 ] - 指示文本是否为斜体的布尔值。

**underline:Boolean** [ 可选 ] - 指示文本是否带有下划线的布尔值。



**url:**String [ 可选 ] - 使用此文本格式的文本超链接到的 URL。如果 url 为空字符串, 则表示文本没有超链接。

**target:**String [ 可选 ] - 显示超链接的目标窗口。如果目标窗口为空字符串, 则文本显示在默认目标窗口 `_self` 中。如果 url 参数设置为空字符串或值 `null`, 虽然您可以获取或设置此属性, 但该属性不起作用。

**align:**String [ 可选 ] - 以字符串形式表示的段落的对齐方式。如果为 "left", 则段落为左对齐。如果为 "center", 则段落居中。如果为 "right", 则段落为右对齐。

**leftMargin:**Number [ 可选 ] - 以磅为单位指示段落的左边距。

**rightMargin:**Number [ 可选 ] - 以磅为单位指示段落的右边距。

**indent:**Number [ 可选 ] - 指示从左边距到段落中第一个字符的缩进的整数。

**leading:**Number [ 可选 ] - 一个数字, 指示行与行之间的前导垂直间距。

## 示例

下面的示例创建一个 **TextFormat** 对象, 设置 stats\_txt 文本字段的格式, 并创建一个在其中显示文本的新文本字段:

```
// Define a TextFormat which is used to format the stats_txt text field.
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
my_fmt.size = 12;
my_fmt.color = 0xFF0000;
// Create a text field to display the player's statistics.
this.createTextField("stats_txt", 5000, 10, 0, 530, 22);
// Apply the TextFormat to the text field.
stats_txt.setNewTextFormat(my_fmt);
stats_txt.selectable = false;
stats_txt.text = "Lorem ipsum dolor sit amet...";
```

若要查看另一示例, 请参见 **ActionScript** 示例文件夹中的 **animations.fl** 文件。下面的列表提供到 **ActionScript** 示例文件夹的典型路径:

- Windows: 引导驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript
- Macintosh: Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples\ActionScript

## underline (TextFormat.underline 属性)

public underline : Boolean

一个布尔值，指示使用此文本格式的文本有下划线 (true) 还是没有下划线 (false)。此下划线类似于用 <u> 标签生成的下划线，但后者不是真正的下划线，因为它不能正确地跳下行字符。默认值为 null，它指示该属性未定义。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例创建一个文本字段并将文本样式设置为带下划线。

```
this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.underline = true;
mytext.text = "This is my first text field object text";
mytext.setTextFormat(myformat);
```

## url (TextFormat.url 属性)

public url : String

指示使用此文本格式的文本链接所至的 URL。如果 url 属性为空字符串，则文本没有超链接。默认值为 null，它指示该属性未定义。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

此示例创建一个文本字段，该文本字段是指向 **Macromedia** 网站的超链接。

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.macromedia.com";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Macromedia.com";
mytext.setTextFormat(myformat);
```

# TextRenderer (flash.text.TextRenderer)



```
public class TextRenderer
extends Object
```

**TextRenderer** 类提供了嵌入字体的高级消除锯齿功能。使用高级消除锯齿时，即使字号很小，也能使字型达到极高的呈现品质。对需要显示大量小字的应用程序使用高级消除锯齿。**Macromedia** 建议不要对非常大（大于 48 磅）的字体使用高级消除锯齿。只有在 **Flash Player 8** 中才能设置高级消除锯齿。

若要在文本字段上设置高级消除锯齿，请设置 **TextField** 实例的 `antiAliasType` 属性。下面的示例需要库中链接标识符的名称为 "CustomFont" 的共享字体。

```
var txtFormat:TextFormat = new TextFormat();
txtFormat.font = "CustomFont";

var label:TextField = this.createTextField("label",
    this.getNextHighestDepth(), 10, 10, 200, 20);
label.setNewTextFormat(txtFormat);
label.text = "Hello World";
label.embedFonts = true;
label.antiAliasType = "advanced";
```

高级消除锯齿提供连续笔触调制 (CSM)，它是笔触粗细和边缘清晰度的连续调制。作为一种高级功能，您可以使用 `setAdvancedAntialiasingTable()` 方法定义特定字体和字体大小的设置。

可用性：ActionScript 1.0；Flash Player 8

另请参见

[antiAliasType \(TextField.antiAliasType 属性\)](#)

## 属性摘要

修饰符	属性	说明
static	<code>maxLevel:Number</code>	用于高级消除锯齿的自适应采样距离字段 (ADF) 的品质级别。

继承自 **Object** 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

方法摘要

修饰符	签名	说明
static	setAdvancedAntialiasingTable(fontName:String, fontStyle:String, colorType:String, advancedAntialiasingTable:Array) : Void	设置字体的自定义连续笔触调制 (CSM) 查找表。

继承自 Object 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

maxLevel (TextRenderer.maxLevel 属性)

public static maxLevel : Number

用于高级消除锯齿的自适应采样距离字段 (ADF) 的品质级别。可接受的值只有 3、4 和 7。高级消除锯齿使用 ADF 表示确定字型的轮廓。品质越高，ADF 结构所需的缓存空间越多。值 3 需要的内存量最小，提供的品质也最低。字体越大，所需的缓存空间越多；对于大小为 64 像素的字体，除非品质级别已设置为 7，否则品质级别将可从 3 增加到 4 或从 4 增加到 7。

可用性: **ActionScript 1.0** ; **Flash Player 8**

示例

下面的示例指定整个 SWF 文件的 maxLevel 值，然后显示具有此值设置的文本字段。为了使本示例中的文本能够正确显示，必须有一个链接标识符为 "CustomFont" 的字体元件。

```
import flash.text.TextRenderer;
TextRenderer.maxLevel = 3;

var txtFormat:TextFormat = new TextFormat();
txtFormat.font = "CustomFont";
txtFormat.size = 64;

var label:TextField = this.createTextField("label",
    this.getNextHighestDepth(), 10, 10, 500, 100);
label.setNewTextFormat(txtFormat);
```

```
label.text = "Hello World";
label.embedFonts = true;
trace("TextRenderer.maxLevel: " + TextRenderer.maxLevel);
```

## setAdvancedAntialiasingTable (TextRenderer.setAdvancedAntialiasingTable 方法)

```
public static setAdvancedAntialiasingTable(fontName:String,
    fontStyle:String, colorType:String, advancedAntialiasingTable:Array) :
    Void
```

设置字体的自定义连续笔触调制 (CSM) 查找表。这是一种高级方法。

**Flash Player** 仅包括 10 种基本字体的高级消除锯齿设置；而且对于这些字体，仅为从 6 到 20 的字体大小提供了高级消除锯齿设置。对于这些字体，6 以下的所有大小的字体将使用 6 的设置；20 以上的所有大小的字体都使用 20 的设置。其它字体映射到提供的字体数据。通过 `setAdvancedAntialiasingTable()` 方法，可以为其它字体和字体大小设置自定义消除锯齿数据，或覆盖所提供字体的默认设置。

可用性：ActionScript 1.0 ； Flash Player 8

### 参数

**fontName:String** - 要为其应用设置的字体的名称。

**fontStyle:String** - 字体样式可以为 "bold"、"bolditalic"、"italic" 和 "none"。

**colorType:String** - 此值可以为 "dark" 或 "light"。

**advancedAntialiasingTable:Array** - 指定的字体的 CSM 设置的数组。每个设置都是具有以下属性的对象：

- **fontSize**
- **insideCutOff**
- **outsideCutOff**

**advancedAntialiasingTable** 数组可以包含指定不同字体大小的 CSM 设置的多个条目。（请参见示例。）

**fontSize** 是设置所应用的大小，以像素为单位。

高级消除锯齿使用自适应采样距离字段 (ADF) 表示确定字型的轮廓。**Macromedia Flash** 使用外侧截止值 **outsideCutOff**（在该值之下密度设置为 0）和内侧截止值 **insideCutOff**（在该值之上密度设置为最大密度值（如 255））。在这两个截止值之间，映射函数是其范围从外侧截止处的 0 到内侧截止处最大密度的线性曲线。

调整外侧截止值和内侧截止值会影响笔触粗细和边缘清晰度。这两个参数之间的间距相当于典型消除锯齿方法的滤镜半径的两倍；较窄的间距提供的边缘更清晰，而较宽的间距提供更柔滑、经过更多过滤的边缘。当间距为 0 时，相应的密度图像为双层位图。在间距非常宽时，生成的密度图像具有类似水彩画的边缘。

通常，对于小磅值，用户首选清晰的、高对比边缘，对于动画文本和较大的磅值，用户首选较柔滑的边缘。

外侧截止通常具有负值，内侧截止通常具有正值，而它们的中点通常在 0 附近。通过调整这些参数以使中点向负无穷大移动，将增大笔触粗细；将中点向正无穷大移动将减小笔触粗细。请确保外侧截止值始终小于或等于内侧截止值。

在大多数情况下，灰度系数指数等于 1 是足够的。但是，当子像素呈现 [ 液晶显示器模式 (LCD)] 时，请使用灰度系数指数柔化彩色边缘效应（此现象会在呈现细笔触（如 Times Roman）和小磅值的字体时出现）。还可以使用灰度系数指数提高阴极射线管 (CRT) 模式和 LCD 模式的对比度。

## 示例

下面的示例创建两个消除锯齿条目和两个说明这些条目的文本字段。要使用此示例，SWF 文件中必须嵌入链接标识符为 "myArial" 的共享字体。若要嵌入字体，请按照下列步骤操作：

- 打开您的库。
- 单击该库右上角的“库”选项菜单。
- 从弹出菜单中选择“新建字体”。
- 将字体命名为 myArial。
- 从“字体”弹出菜单中选择“Arial”。
- 单击“确定”。
- 右键单击新建的字体，并选择“链接”。
- 选中“为 ActionScript 导出”复选框。
- 单击“确定”以接受默认标识符 myArial。

```
import flash.text.TextRenderer;
```

```
var antiAliasEntry_1 = {fontSize:24, insideCutoff:1.61, outsideCutoff:-3.43};  
var antiAliasEntry_2 = {fontSize:48, insideCutoff:0.8, outsideCutoff:-0.8};  
var arialTable:Array = new Array(antiAliasEntry_1, antiAliasEntry_2);
```

```
var lbl_1:TextField = createLabel(0, 0, 300, 100, 24);  
var lbl_2:TextField = createLabel(0, 100, 300, 100, 48);
```

```
TextRenderer.setAdvancedAntialiasingTable("Arial", "none", "dark",  
    arialTable);
```

```

function createLabel(x:Number, y:Number, width:Number, height:Number,
    fontSize:Number):TextField {
    var depth:Number = this.getNextHighestDepth();

    var tmpTxt = this.createTextField("txt_" + depth, depth, x, y, width,
        height);
    tmpTxt.antiAliasType = "advanced";
    tmpTxt.gridFitType = "pixel";
    tmpTxt.border = true;
    tmpTxt.text = "Hello World";
    tmpTxt.embedFonts = true;
    tmpTxt.setTextFormat(getTextFormat(fontSize));
    return tmpTxt;
}

function getTextFormat(fontSize:Number):TextFormat {
    var tf:TextFormat = new TextFormat();
    tf.align = "center";
    tf.size = fontSize;
    tf.font = "myArial";
    return tf;
}

```

## TextSnapshot

```

Object
|
+-TextSnapshot

```

```

public class TextSnapshot
extends Object

```

**TextSnapshot** 对象可用于处理影片剪辑中的静态文本。例如，您可以使用 **TextSnapshot** 对象用比动态文本所允许的更高的精度对文本进行布局，但仍以只读方式访问该文本。

您不必使用构造函数即可创建 **TextSnapshot** 对象；它由 `MovieClip.getTextSnapshot()` 返回。

可用性：ActionScript 1.0；Flash Player 7

另请参见

[getTextSnapshot](#) ([MovieClip.getTextSnapshot](#) 方法)

属性摘要

继承自 Object 类的属性

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

方法摘要

修饰符	签名	说明
	<code>findText(startIndex:Number, textToFind:String, caseSensitive:Boolean) : Number</code>	搜索指定的 TextSnapshot 对象，并返回在 startIndex 位置或其后找到的 textToFind 的第一个匹配项的位置。
	<code>getCount() : Number</code>	返回 TextSnapshot 对象中的字符数。
	<code>getSelected(start:Number, [end:Number]) : Boolean</code>	返回一个布尔值，该值指定 TextSnapshot 对象在指定范围内是否包含所选的文本。
	<code>getSelectedText([includeLineEndings:Boolean]) : String</code>	返回一个字符串，它包含对应的 TextSnapshot.setSelected() 方法指定的所有字符。
	<code>getText(start:Number, end:Number, [includeLineEndings:Boolean]) : String</code>	返回一个字符串，它包含 start 和 end 参数指定的所有字符。
	<code>getTextRunInfo(beginIndex:Number, endIndex:Number) : Array</code>	返回包含关于文本运行信息的对象的数组。
	<code>hitTestTextNearPos(x:Number, y:Number, [closeDist:Number]) : Number</code>	用于确定 TextSnapshot 对象中哪个字符位于包含 TextSnapshot 对象中文本的影片剪辑指定的 x, y 坐标上或位于该坐标对附近。
	<code>setSelectColor(color:Number) : Void</code>	指定当突出显示使用 TextSnapshot.setSelected() 方法选择的字符时要使用的颜色。
	<code>setSelected(start:Number, end:Number, select:Boolean) : Void</code>	指定 TextSnapshot 对象中要选择或取消选择的字符范围。



继承自 Object 类的方法

---

[addProperty](#) ([Object.addProperty](#) 方法), [hasOwnProperty](#) ([Object.hasOwnProperty](#) 方法), [isPropertyEnumerable](#) ([Object.isPropertyEnumerable](#) 方法), [isPrototypeOf](#) ([Object.isPrototypeOf](#) 方法), [registerClass](#) ([Object.registerClass](#) 方法), [toString](#) ([Object.toString](#) 方法), [unwatch](#) ([Object.unwatch](#) 方法), [valueOf](#) ([Object.valueOf](#) 方法), [watch](#) ([Object.watch](#) 方法)

---

## findText (TextSnapshot.findText 方法)

```
public findText(startIndex:Number, textToFind:String, caseSensitive:Boolean)
: Number
```

搜索指定的 **TextSnapshot** 对象，并返回在 **startIndex** 位置或其后找到的 **textToFind** 的第一个匹配项的位置。如果未找到 **textToFind**，则该方法返回 -1。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**startIndex:Number** - 指定 **TextSnapshot** 文本中的起始索引点，从该位置搜索指定文本。

**textToFind:String** - 要搜索的文本。指定字符串（括在引号内）或变量。

**caseSensitive:Boolean** - 布尔值，指定找到的文本是否必须与 **textToFind** 中的字符串大小写匹配，如果是，则该值为 **true**；否则为 **false**。

### 返回

**Number** - 指定文本的第一个匹配项从零开始的索引位置，如果没有匹配的文本，则返回 -1。

### 示例

以下示例说明如何使用此方法。要使用此代码，请将包含文本 “**TextSnapshot Example**” 的静态文本字段放在舞台上。

```
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var index1:Number = my_snap.findText(0, "Snap", true);
var index2:Number = my_snap.findText(0, "snap", true);
var index3:Number = my_snap.findText(0, "snap", false);
trace(index1); // 4
trace(index2); // -1
trace(index3); // 4
```

### 另请参见

[getText](#) ([TextSnapshot.getText](#) 方法)

## getCount (TextSnapshot.getCount 方法)

public getCount() : Number

返回 **TextSnapshot** 对象中的字符数。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 返回

Number - **TextSnapshot** 对象中的字符数。

### 示例

下面的示例说明如何返回 **TextSnapshot** 对象中的字符数。若要使用此代码，请将一个包含文本 “**TextSnapshot Example**”（且仅包含该文本）的静态文本字段放在舞台上。

```
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var count:Number = my_snap.getCount();
var theText:String = my_snap.getText(0, count, false);
trace(count); // 20
trace(theText); // TextSnapshot Example
```

### 另请参见

[getText \(TextSnapshot.getText 方法\)](#)

## getSelected (TextSnapshot.getSelected 方法)

public getSelected(start:Number, [end:Number]) : Boolean

返回一个布尔值，该值指定 **TextSnapshot** 对象在指定范围内是否包含所选的文本。

若要搜索全部字符，请将值 0 传递给 start，并将 **TextSnapshot.getCount()**（或任何非常大的数字）传递给 end。若要搜索一个字符，请传递给 end 参数一个比 start 参数大 1 的值。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**start**:Number - 要检查的第一个字符的索引位置。start 的有效值从 0 到 **TextSnapshot.getCount()** - 1。如果 start 是一个负值，则使用 0。

**end**:Number [ 可选 ] - 比最后一个要检查字符大 1 的索引位置。end 的有效值从 0 到 **TextSnapshot.getCount()**。由 end 参数索引的字符未包括在已提取的字符串中。如果您省略此参数，请使用 **TextSnapshot.getCount()**。如果 end 的值小于或等于 start 的值，请使用 start + 1。

## 返回

Boolean - 一个布尔值，指示相应的 `TextSnapshot.setSelected()` 方法是否选择了给定范围内的至少一个字符，如果选择了字符，则为 `true`；否则为 `false`。

## 示例

以下示例说明如何使用此方法。要使用此代码，请将包含文本“**TextSnapshot Example**”的静态文本字段放在舞台上。在包括静态文本字段使用的字体的“库”中，在字体的“链接”选项中，选择“为 **ActionScript** 导出”。将下面的 **ActionScript** 添加到时间轴的第 1 帧：

```
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelected(0, 4, true);
my_snap.setSelected(1, 2, false);

var firstCharIsSelected:Boolean = my_snap.getSelected(0, 1);
var secondCharIsSelected:Boolean = my_snap.getSelected(1, 2);
trace(firstCharIsSelected); // true
trace(secondCharIsSelected); // false
```

## 另请参见

[getCount \(TextSnapshot.getCount 方法\)](#), [getText \(TextSnapshot.getText 方法\)](#), [getSelectedText \(TextSnapshot.getSelectedText 方法\)](#), [setSelected \(TextSnapshot.setSelected 方法\)](#)

# getSelectedText (TextSnapshot.getSelectedText 方法)

```
public getSelectedText([includeLineEndings:Boolean]) : String
```

返回一个字符串，它包含对应的 `TextSnapshot.setSelected()` 方法指定的所有字符。如果没有指定任何字符（由 `TextSnapshot.setSelected()` 方法指定），则返回一个空字符串。

如果将 `true` 传递给 `includeLineEndings`，则换行符将插入到返回字符串中，因此该返回字符串可能比输入范围长。如果 `includeLineEndings` 为 `false` 或被省略，则该方法返回没有添加任何字符的选定文本。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**includeLineEndings:Boolean** [ 可选 ] - 一个布尔值，指定换行符是否插入到返回的字符串中，如果插入到返回的字符串中，则该值为 `true`，否则为 `false`。默认值为 `false`。

## 返回

String - 一个字符串，该字符串包含对应的 `TextSnapshot.setSelected()` 方法指定的所有字符。

## 示例

以下示例说明如何使用此方法。要使用此代码，请将包含文本 “TextSnapshot Example” 的静态文本字段放在舞台上。然后在包括静态文本字段使用的字体的“库”中，在字体的“链接”选项中，选择“为 **ActionScript** 导出”。将下面的 **ActionScript** 添加到时间轴的第 1 帧：

```
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelected(0, 4, true);
my_snap.setSelected(1, 2, false);

var theText:String = my_snap.getSelectedText(false);
trace(theText); // Text
```

当您测试 SWF 文件时，一个彩色矩形将出现在指定字符的周围。

## 另请参见

[getSelected \(TextSnapshot.getSelected 方法\)](#)，[setSelected \(TextSnapshot.setSelected 方法\)](#)

## getText (TextSnapshot.getText 方法)

```
public getText(start:Number, end:Number, [includeLineEndings:Boolean]) :
    String
```

返回一个字符串，它包含 `start` 和 `end` 参数指定的所有字符。如果没有指定任何字符，该方法将返回一个空字符串。

若要返回全部字符，请将值 0 传递给 `start`，并将 `TextSnapshot.getCount()`（或任何较大数字）传递给 `end`。若要返回单个字符，请为 `end` 传递值 `start + 1`。

如果将 `true` 传递给 `includeLineEndings`，则换行符将插入到返回字符串中被视为合适的位置，因此该返回字符串可能比输入范围长。如果 `includeLineEndings` 为 `false` 或被省略，则该方法返回没有添加任何字符的选定文本。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**start:**Number - 一个整数，指示要包括在返回的字符串中的第一个字符的位置。start 的有效值从 0 到 TextSnapshot.getCount() - 1。如果 start 是一个负值，则使用 0。

**end:**Number - 一个整数，它等于要在 TextSnapshot 对象中检查的最后一个字符的索引加 1。end 的有效值从 0 到 TextSnapshot.getCount()。由 end 参数索引的字符未包括在已提取的字符串中。如果您省略此参数，请使用 TextSnapshot.getCount()。如果 end 的值小于或等于 start 的值，请使用 start + 1。

**includeLineEndings:**Boolean [可选] - 一个布尔值，指定换行符是否插入到返回的字符串中，如果插入到返回的字符串中，则该值为 true，否则为 false。默认值为 false。

## 返回

String - 一个字符串，该字符串包含指定范围内的字符，如果在指定范围内未找到任何字符，则返回一个空字符串。

## 示例

下面的示例说明如何返回指定 TextSnapshot 对象中的字符数。要使用此代码，请将包含文本 “TextSnapshot Example” 的静态文本字段放在舞台上。

```
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var count:Number = my_snap.getCount();
var theText:String = my_snap.getText(0, count, false);
trace(count); // 20
trace(theText); // TextSnapshot Example
```

## 另请参见

[getCount \(TextSnapshot.getCount 方法\)](#), [getSelectedText \(TextSnapshot.getSelectedText 方法\)](#)

## getTextRunInfo (TextSnapshot.getTextRunInfo 方法)

```
public getTextRunInfo(beginIndex:Number, endIndex:Number) : Array
```

返回包含关于文本运行信息的对象的数组。每个对象都与两个方法参数指定的字符范围中的一个字符相对应。



对一个大范围的文本使用 `getTextRunInfo()` 方法可以返回一个较大的对象。Macromedia 建议限制由 `beginIndex` 参数和 `endIndex` 参数定义的文本范围。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**beginIndex:**Number - 在字符范围中的第一个字符的索引值。

**endIndex:**Number - 在字符范围中的最后一个字符的索引值。

### 返回

Array - 对象的数组，该数组中的每个对象都包含关于指定范围内的特定字符的信息。每个对象都包含下列属性：

- `indexInRun` 字符相对于整个字符串（而不是相对于所选运行文本）的从零开始的整数索引。
- `selected` 一个布尔值，指示字符是否被选定，如果被选定，则该值为 `true`；否则为 `false`。
- `font` 字符的字体名称。
- `color` 字符的 **Alpha** 和颜色的合并值。前两个十六进制数字表示 **alpha** 值，而其余数字表示颜色值。（该示例包括将十进制值转换为十六进制值的方法。）
- `height` 字符的高度，以像素为单位。
- `matrix_a`、`matrix_b`、`matrix_c`、`matrix_d`、`matrix_tx` 和 `matrix_ty` 矩阵的值，该矩阵对字符定义几何转换。通常，立式文本始终具有形式为 `[1 0 0 1 x y]` 的矩阵，其中 `x` 和 `y` 是字符在父级影片剪辑内的位置，与文本的高度无关。该矩阵位于父级影片剪辑坐标系统中，并且不包括对影片剪辑本身（或其父级）的任何转换。
- `corner0x`、`corner0y`、`corner1x`、`corner1y`、`corner2x`、`corner2y`、`corner3x` 和 `corner3y` 字符边框的各个角，基于父级影片剪辑的坐标系统。这些值仅在字符使用的字体嵌入到 **SWF** 文件时可用。

## 示例

以下示例说明如何使用此方法。若要使用此代码，请在舞台上创建包含文本“AB”的静态文本字段。将文本字段 45 旋转度，并将第二个字符设置为颜色为 0xFFFFF 且 alpha 为 50% 的上标，如下图所示：



下面的脚本列出文本字段中每个字符的 `getTextRunInfo()` 属性：

```
var myTS:TextSnapshot = this.getTextSnapshot();
var myArray:Array = myTS["getTextRunInfo"](0, myTS.getCount());
for (var i = 0; i < myTS.getCount(); i++) {
    trace("indexInRun: " + myArray[i].indexInRun);
    trace("selected: " + myArray[i].selected);
    trace("font: " + myArray[i].font);
    trace("color: " + decToHex(myArray[i].color));
    trace("height: " + myArray[i].height);
    trace("matrix_a: " + myArray[i].matrix_a);
    trace("matrix_b: " + myArray[i].matrix_b);
    trace("matrix_c: " + myArray[i].matrix_c);
    trace("matrix_d: " + myArray[i].matrix_d);
    trace("matrix_ty: " + myArray[i].matrix_ty);
    trace("matrix_tx: " + myArray[i].matrix_tx);
    trace(" ");
}

function decToHex(dec:Number) {
    var hexString:String = "";
    if (dec > 15) {
        hexString = decToHex(Math.floor(dec / 16));
    }
    var hexDigit = dec - 16 * (Math.floor(dec / 16));
    if (hexDigit > 9) {
        hexDigit = String.fromCharCode(hexDigit + 55);
    }
    hexString = hexString + hexDigit;
    return hexString;
}
```

它创建以下输出：

```
indexInRun: 0
selected: false
font: Times New Roman
color: FF000000
height: 28.6
matrix_a: 0.0316612236983293
matrix_b: 0.0385940558426864
matrix_c: -0.0385940558426864
matrix_d: 0.0316612236983293
matrix_ty: 22.75
```

```
matrix_tx: 40.35

indexInRun: 0
selected: false
font: Times New Roman
color: 80000000
height: 28.6
matrix_a: 0.0316612236983293
matrix_b: 0.0385940558426864
matrix_c: -0.0385940558426864
matrix_d: 0.0316612236983293
matrix_ty: 49
matrix_tx: 45.5
```

此示例使用 `decToHex()` 方法将 `color` 属性的十进制值转换为十六进制值。

另请参见

[Matrix \(flash.geom.Matrix\)](#)

## hitTestTextNearPos (TextSnapshot.hitTestTextNearPos 方法)

```
public hitTestTextNearPos(x:Number, y:Number, [closeDist:Number]) : Number
```

用于确定 **TextSnapshot** 对象中哪个字符位于包含 **TextSnapshot** 对象中文本的影片剪辑指定的 **x, y** 坐标上或位于该坐标附近。

如果省略 `closeDist` 或为其传递值 0，则由 **x, y** 坐标指定的位置必须位于 **TextSnapshot** 对象的边框内。

此方法仅对于包括字符度量信息的字体正常起作用；但是，默认情况下，**Macromedia Flash** 不为静态文本字段包含此信息。因此，该方法可能返回 -1 而不是索引值。若要确保返回索引值，您可以强制使 **Flash** 创作工具包括字体的字符度量信息。为此，请添加一个使用此字体的动态文本字段，为该动态文本字段选择“字符选项”，然后指定至少为一个字符嵌入该字体轮廓。（指定哪个（些）字符并没有关系，它们是否为此静态文本字段中使用的字符也没有关系。）

可用性：ActionScript 1.0；Flash Player 7

### 参数

**x**:Number - 包含 **TextSnapshot** 对象中的文本的影片剪辑的 **x** 坐标。

**y**:Number - 包含 **TextSnapshot** 对象中的文本的影片剪辑的 **y** 坐标。

**closeDist**:Number [ 可选 ] - 可用于搜索文本的以 **x, y** 为起点的距离的最大值。此距离是从每个字符的中心点开始测量的。默认值为 0。



## 返回

Number - **TextSnapshot** 对象中的字符的索引值，该对象距离指定的 *x, y* 坐标最近。如果未找到任何字符，或者字体不包含字符度量信息，则该方法返回 -1。

## 示例

以下示例说明如何使用此方法。要使用此代码，请将包含文本 “**TextSnapshot Example**” 的静态文本字段放在舞台上。在包括静态文本字段使用的字体的 “库” 中，在字体的 “链接” 选项中，选择 “为 **ActionScript** 导出”。若要测试代码，请运行 **SWF** 文件并将鼠标指针指向屏幕上的文本。

```
var my_ts:TextSnapshot = getTextSnapshot();
this.onMouseMove = function() {
    var hitIndex:Number = my_ts.hitTestTextNearPos(_xmouse, _ymouse, 0);
    my_ts.setSelected(0, my_ts.getCount(), false);
    if (hitIndex >= 0) {
        my_ts.setSelected(hitIndex, hitIndex + 1, true);
    }
};
```

## 另请参见

[getTextSnapshot \(MovieClip.getTextSnapshot 方法\)](#), [\\_x \(MovieClip.\\_x 属性\)](#), [\\_y \(MovieClip.\\_y 属性\)](#)

## setSelectColor (TextSnapshot.setSelectColor 方法)

```
public setSelectColor(color:Number) : Void
```

指定当突出显示使用 `TextSnapshot.setSelected()` 方法选择的字符时要使用的颜色。该颜色始终是不透明的；您不能指定透明值。

此方法仅对于包括字符度量信息的字体正常起作用；但是，默认情况下，**Macromedia Flash** 不为静态文本字段包含此信息。因此，该方法可能返回 -1 而不是索引值。若要确保返回索引值，您可以强制使 **Flash** 创作工具包括字体的字符度量信息。为此，请添加一个使用此字体的动态文本字段，为该动态文本字段选择 “字符选项”，然后指定至少为一个字符嵌入该字体轮廓。（指定哪个（些）字符并没有关系，它们是否为此静态文本字段中使用的字符也没有关系。）

可用性：ActionScript 1.0；Flash Player 7

## 参数

**color**:Number - 用于字符（这些字符已被对应的 `TextSnapshot.setSelected()` 方法选中）旁边边框的颜色，以 **0xRRGGBB** 格式表示。

## 示例

以下示例说明如何使用此方法。要使用此代码，请将包含文本“TextSnapshot Example”的静态文本字段放在舞台上。在包括静态文本字段使用的字体的“库”中，在字体的“链接”选项中，选择“为 ActionScript 导出”。将下面的 ActionScript 添加到时间轴的第 1 帧：

```
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelectColor(0xFF0000);
my_snap.setSelected(0, 4, true);
my_snap.setSelected(1, 2, false);

var theText:String = my_snap.getSelectedText(false); // get the selected text
trace(theText); // Text
```

当您测试 SWF 文件时，可在指定字符的周围看到一个彩色矩形。

## 另请参见

[setSelected \(TextSnapshot.setSelected 方法\)](#)

## setSelected (TextSnapshot.setSelected 方法)

```
public setSelected(start:Number, end:Number, select:Boolean) : Void
```

指定 TextSnapshot 对象中要选择或取消选择的字符范围。被选中的字符的后面绘制有一个带颜色的矩形，与字符的边框匹配。边框的颜色由 TextSnapshot.setSelectColor() 确定。

若要选择或取消选择所有字符，请为 start 传递值 0，并为 end 传递

TextSnapshot.getCount()（或任何一个非常大的数字）。若要指定单个字符，请为 end 传递值 start + 1。

因为在选择时，字符是单个地进行标记的，所以您可以多次调用此方法以选择多个字符；也就是说，使用此方法不会取消选择已由此方法设置的其它字符。

此方法仅对于包括字符度量信息的字体正常起作用；默认情况下，Flash 不为静态文本字段包括此信息。因此，选定的文本可能不显示在所选的屏幕上。若要确保所有选中的文本都显示为已选中，您可以强制使 Flash 创作工具包括字体的字符度量信息。为此，在包括静态文本字段使用的字体的“库”中，在字体的“链接”选项中，选择“为 ActionScript 导出”。

可用性：ActionScript 1.0；Flash Player 7

## 参数

**start:**Number - 要选择的第一个字符的位置。start 的有效值从 0 到 TextSnapshot.getCount() - 1。如果 start 是一个负值，则使用 0。

**end:**Number - 一个整数，它等于要检查的最后一个字符的索引加 1。end 的有效值从 0 到 TextSnapshot.getCount()。由 end 参数索引的字符未包括在已提取的字符串中。如果您省略此参数，请使用 TextSnapshot.getCount()。如果 end 的值小于或等于 start 的值，请使用 start + 1。

**select:**Boolean - 一个布尔值，指示文本是否应该被选择，如果应选择文本，则该值为 true，否则为 false。

## 示例

以下示例说明如何使用此方法。要使用此代码，请将包含文本“TextSnapshot Example”的静态文本字段放在舞台上。在包括静态文本字段使用的字体的“库”中，在字体的“链接”选项中，选择“为 ActionScript 导出”。将下面的 ActionScript 添加到时间轴的第 1 帧：

```
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelected(0, 4, true);
my_snap.setSelected(1, 2, false);

var theText:String = my_snap.getSelectedText(false);
trace(theText); // Text
```

## 另请参见

[getCount \(TextSnapshot.getCount 方法\)](#)

# Transform (flash.geom.Transform)

```
Object
|
+- flash.geom.Transform
```

```
public class Transform
extends Object
```

Transform 类收集有关应用于 MovieClip 对象的颜色转换和坐标处理的数据。

Transform 对象通常是通过从 MovieClip 对象获取 transform 属性的值获得的。

可用性: ActionScript 1.0 ; Flash Player 8

## 另请参见

[transform \(MovieClip.transform 属性\)](#), [ColorTransform \(flash.geom.ColorTransform\)](#), [Matrix \(flash.geom.Matrix\)](#)

属性摘要

修饰符	属性	说明
	colorTransform:ColorTransform	一个 ColorTransform 对象，包含普遍调整影片剪辑中颜色的值。
	concatenatedColorTransform:ColorTransform [ 只读 ]	一个 ColorTransform 对象，表示应用于此对象及其所有父级对象的组合颜色转换，回到根级别。
	concatenatedMatrix:Matrix [ 只读 ]	一个 Matrix 对象，表示此对象及其所有父级对象的组合转换矩阵，回到根级别。
	matrix:Matrix	一个转换 Matrix 对象，包含影响影片剪辑的缩放、旋转和平移的值。
	pixelBounds:Rectangle	一个 Rectangle 对象，定义舞台上 MovieClip 对象的边框矩形。

继承自 Object 类的属性

[constructor](#) (Object.constructor 属性), [\\_\\_proto\\_\\_](#) (Object.\_\_proto\_\_ 属性), [prototype](#) (Object.prototype 属性), [\\_\\_resolve](#) (Object.\_\_resolve 属性)

构造函数摘要

签名	说明
Transform(mc:MovieClip)	创建附加到给定 MovieClip 对象的 Transform 对象。

方法摘要

继承自 Object 类的方法

[addProperty](#) (Object.addProperty 方法), [hasOwnProperty](#) (Object.hasOwnProperty 方法), [isPropertyEnumerable](#) (Object.isPropertyEnumerable 方法), [isPrototypeOf](#) (Object.isPrototypeOf 方法), [registerClass](#) (Object.registerClass 方法), [toString](#) (Object.toString 方法), [unwatch](#) (Object.unwatch 方法), [valueOf](#) (Object.valueOf 方法), [watch](#) (Object.watch 方法)

## colorTransform (Transform.colorTransform 属性)

public colorTransform : ColorTransform

一个 **ColorTransform** 对象，包含普遍调整影片剪辑中颜色的值。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将 **ColorTransform** 对象 `blueColorTransform` 应用于 **Transform** 对象 `trans`。此 **ColorTransform** 将 **MovieClip** `rect` 的颜色从红色转换为蓝色。

```
import flash.geom.Transform;
import flash.geom.ColorTransform;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var trans:Transform = new Transform(rect);
trace(trans.colorTransform);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,
   redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)

var blueColorTransform:ColorTransform = new ColorTransform(0, 1, 1, 1, 0, 0,
   255, 0);

rect.onPress = function() {
    trans.colorTransform = blueColorTransform;
    trace(trans.colorTransform);
    // (redMultiplier=0, greenMultiplier=1, blueMultiplier=1,
       alphaMultiplier=1, redOffset=0, greenOffset=0, blueOffset=255,
       alphaOffset=0)
}

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

另请参见

[ColorTransform \(flash.geom.ColorTransform\)](#)

## concatenatedColorTransform

### (Transform.concatenatedColorTransform 属性)

public concatenatedColorTransform : ColorTransform [read-only]

一个 **ColorTransform** 对象，表示应用于此对象及其所有父级对象的组合颜色转换，回到根级别。如果在不同级别上应用了不同的颜色转换，则将其每个转换连接成此属性的一个 **ColorTransform** 对象。

可用性：ActionScript 1.0；Flash Player 8

#### 示例

下面的示例将两个 **Transform** 对象同时应用于父级和子级 **MovieClip** 对象。

blueColorTransform 变量随后应用于 **Transform** 对象 parentTrans，这将父级和子级 **MovieClip** 对象的颜色都调整为蓝色。您可以看到 child.concatenatedColorTransform 是如何合并 parentTrans 和 childTrans 的。

```
import flash.geom.Transform;
import flash.geom.ColorTransform;

var parentRect:MovieClip = createRectangle(20, 80, 0xFF0000);
var childRect:MovieClip = createRectangle(10, 40, 0x00FF00, parentRect);

var parentTrans:Transform = new Transform(parentRect);
var childTrans:Transform = new Transform(childRect);

var blueColorTransform:ColorTransform = new ColorTransform(0, 1, 1, 1, 0, 0,
    255, 0);

parentTrans.colorTransform = blueColorTransform;

trace(childTrans.concatenatedColorTransform);
// (redMultiplier=0, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,
//   redOffset=0, greenOffset=0, blueOffset=255, alphaOffset=0)
trace(childTrans.colorTransform);
// (redMultiplier=1, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,
//   redOffset=0, greenOffset=0, blueOffset=0, alphaOffset=0)
trace(parentTrans.concatenatedColorTransform);
// (redMultiplier=0, greenMultiplier=1, blueMultiplier=1, alphaMultiplier=1,
//   redOffset=0, greenOffset=0, blueOffset=255, alphaOffset=0)

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
```

```

        mc.lineTo(width, height);
        mc.lineTo(width, 0);
        mc.lineTo(0, 0);
        return mc;
    }

```

另请参见

[ColorTransform \(flash.geom.ColorTransform\)](#)

## concatenatedMatrix (Transform.concatenatedMatrix 属性)

public concatenatedMatrix : Matrix [read-only]

一个 **Matrix** 对象，表示此对象及其所有父级对象的组合转换矩阵，回到根级别。如果在不同级别上应用了不同的转换矩阵，则将其每个矩阵连接成此属性的一个矩阵。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例将两个 **Transform** 对象同时应用于子级和父级 **MovieClip** 对象。scaleMatrix 随后应用于 **Transform** 对象 parentTrans，这将同时缩放父级和子级 **MovieClip** 对象。您可以看到 child.concatenatedMatrix 是如何合并 parentTrans 和 childTrans 的。

```

import flash.geom.Transform;
import flash.geom.Matrix;

var parentRect:MovieClip = createRectangle(20, 80, 0xFF0000);
var childRect:MovieClip = createRectangle(10, 40, 0x00FF00, parentRect);

var parentTrans:Transform = new Transform(parentRect);
var childTrans:Transform = new Transform(childRect);

var scaleMatrix:Matrix = new Matrix();
scaleMatrix.scale(2, 2);

parentTrans.matrix = scaleMatrix;

trace(childTrans.concatenatedMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
trace(childTrans.matrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)
trace(parentTrans.concatenatedMatrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);

```

```

mc.beginFill(color);
mc.lineTo(0, height);
mc.lineTo(width, height);
mc.lineTo(width, 0);
mc.lineTo(0, 0);
return mc;
}

```

## matrix（Transform.matrix 属性）

public matrix : Matrix

一个转换 **Matrix** 对象，包含影响影片剪辑的缩放、旋转和平移的值。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例将 **Matrix** 对象 `scaleMatrix` 应用于 **Transform** 对象 `trans`。此 **Matrix** 使用系数 2 缩放 **MovieClip** `rect`。

```

import flash.geom.Transform;
import flash.geom.Matrix;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var trans:Transform = new Transform(rect);
trace(trans.matrix); // (a=1, b=0, c=0, d=1, tx=0, ty=0)

var scaleMatrix:Matrix = new Matrix();
scaleMatrix.scale(2, 2);

rect.onPress() = function() {
    trans.matrix = scaleMatrix;
    trace(trans.matrix); // (a=2, b=0, c=0, d=2, tx=0, ty=0)
}

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}

```

### 另请参见

[Matrix \(flash.geom.Matrix\)](#)



## pixelBounds (Transform.pixelBounds 属性)

public pixelBounds : Rectangle

一个 **Rectangle** 对象，定义舞台上 **MovieClip** 对象的边框矩形。

可用性: **ActionScript 1.0** ; **Flash Player 8**

### 示例

下面的示例创建 **Transform** 对象 **trans**，并输出其 **pixelBounds** 属性。请注意，**pixelBounds** 返回一个边框，其值与 **MovieClip** 对象的 **getBounds()** 和 **getRect()** 方法的值相等。

```
import flash.geom.Transform;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);
var trans:Transform = new Transform(rect);
trace(trans.pixelBounds); // (x=0, y=0, w=20, h=80)

var boundsObj:Object = rect.getBounds();
trace(boundsObj.xMin); // 0
trace(boundsObj.yMin); // 0
trace(boundsObj.xMax); // 20
trace(boundsObj.yMax); // 80

var rectObj:Object = rect.getRect();
trace(rectObj.xMin); // 0
trace(rectObj.yMin); // 0
trace(rectObj.xMax); // 20
trace(rectObj.yMax); // 80

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

# Transform 构造函数

```
public Transform(mc:MovieClip)
```

创建附加到给定 **MovieClip** 对象的 **Transform** 对象。

在创建新 **Transform** 对象后，就可以通过获取给定 **MovieClip** 对象的 `transform` 属性来检索它。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**mc:MovieClip** - 将新 **Transform** 对象应用到的 **MovieClip** 对象。

## 示例

下面的示例创建 **Transform** `trans`，并将其应用于 **MovieClip** 矩形。您可以看到 **Transform** 对象的 `trans` 和 `rect.transform` 的计算结果不相等，即使它们包含相同的值。

```
import flash.geom.Transform;

var rect:MovieClip = createRectangle(20, 80, 0xFF0000);

var trans:Transform = new Transform(rect);

trace(rect.transform == trans); // false

for(var i in trans) {
    trace(">> " + i + ": " + trans[i]);
    // >> pixelBounds: (x=0, y=0, w=20, h=80)
    // >> concatenatedColorTransform: (redMultiplier=1, greenMultiplier=1,
    blueMultiplier=1, alphaMultiplier=1, redOffset=0, greenOffset=0,
    blueOffset=0, alphaOffset=0)
    // >> colorTransform: (redMultiplier=1, greenMultiplier=1,
    blueMultiplier=1, alphaMultiplier=1, redOffset=0, greenOffset=0,
    blueOffset=0, alphaOffset=0)
    // >> concatenatedMatrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
    // >> matrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
}

for(var i in rect.transform) {
    trace(">> " + i + ": " + rect.transform[i]);
    // >> pixelBounds: (x=0, y=0, w=20, h=80)
    // >> concatenatedColorTransform: (redMultiplier=1, greenMultiplier=1,
    blueMultiplier=1, alphaMultiplier=1, redOffset=0, greenOffset=0,
    blueOffset=0, alphaOffset=0)
    // >> colorTransform: (redMultiplier=1, greenMultiplier=1,
    blueMultiplier=1, alphaMultiplier=1, redOffset=0, greenOffset=0,
    blueOffset=0, alphaOffset=0)
    // >> concatenatedMatrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
```

```
// >> matrix: (a=1, b=0, c=0, d=1, tx=0, ty=0)
}

function createRectangle(width:Number, height:Number, color:Number,
    scope:MovieClip):MovieClip {
    scope = (scope == undefined) ? this : scope;
    var depth:Number = scope.getNextHighestDepth();
    var mc:MovieClip = scope.createEmptyMovieClip("mc_" + depth, depth);
    mc.beginFill(color);
    mc.lineTo(0, height);
    mc.lineTo(width, height);
    mc.lineTo(width, 0);
    mc.lineTo(0, 0);
    return mc;
}
```

## Video

```
Object
|
+-Video
```

```
public class Video
extends Object
```

使用 **Video** 类可以直接在舞台上显示实时视频流，而无需将其嵌入您的 SWF 文件中。可以使用 `Camera.get()` 来捕获视频。在为 **Flash Player 7** 和更高版本发布的文件中，您还可以使用 **Video** 类通过 HTTP 或在本地文件系统中播放 Flash 视频 (FLV) 文件。有关更多信息，请参见 **NetConnection** 类和 **NetStream** 类条目。

**Flash Player 7** 支持用 Sorenson Spark 视频编解码器编码的 Flash video (FLV)。**Flash Player 8** 支持用 Sorenson 或 On2 VP6 编解码器编码的 Flash video (FLV)，也支持 Alpha 通道。On2 VP6 视频编解码器比早期技术占用的带宽少，并且另外提供了解块过滤器和 **Deringing** 过滤器。

如果您的 Flash 内容动态装载 Flash 视频（使用渐进式下载或 **Flash Communication Server**），您不必为 **Flash Player 8** 重新发布 SWF 就可以使用 On2 VP6 视频，只要用户通过 **Flash Player 8** 来观看您的内容。通过对 On2 VP6 视频进行流式处理或将其下载到 Flash SWF 第 6 版或第 7 版中，并使用 **Flash Player 8** 播放内容，就可以避免重新创建与 **Flash Player 8** 一起使用的 SWF 文件。

Codec	内容 (SWF) 版本 （发布版本）	Flash Player 版本 （进行回放所需要的版本）
Sorenson Spark	6	6、7、8
	7	7、8
On2 VP6	6	8*
	7	8
	8	8

\* 如果您的 Flash 内容动态装载 Flash 视频 (FLV)，您不必为 Flash Player 8 重新发布您的 SWF 就可以使用 On2 VP6 视频，只有用户使用 Flash Player 8 来观看您的内容。只有 Flash Player 8 既支持发布又支持回放 On2 VP6 视频。

Video 对象的使用方式类似于影片剪辑。和放置在舞台上的其它对象一样，您可以控制 Video 对象的多种属性。例如，可以通过使用 Video 对象的 `_x` 和 `_y` 属性在舞台上移动该对象，可以通过使用其 `_height` 和 `_width` 属性更改其大小，等等。

若要显示视频流，请首先将 Video 对象放置在舞台上。然后使用 `Video.attachVideo()` 将视频流附加到 Video 对象。

- 如果看不到“库”面板，请选择“窗口” > “库”，让该面板显示出来。
- 单击“库”面板标题栏右侧的“选项”菜单，然后选择“新建视频”，在库中添加一个嵌入式 Video 对象。
- 将该 Video 对象拖放到舞台上，然后使用属性检查器为它指定一个唯一的实例名称，如 `my_video`。（不要将其命名为“Video”。）

可用性：ActionScript 1.0；Flash Player 6

另请参见

[NetConnection](#)，[NetStream](#)

属性摘要

修饰符	属性	说明
	<code>_alpha: Number</code>	指示指定的 Video 对象的 Alpha 透明度值。
	<code>deblocking: Number</code>	指示作为后处理的一部分应用于已解码视频的解块过滤器的类型。
	<code>_height: Number</code>	指示 Video 对象的高度，以像素为单位。
	<code>height: Number [ 只读 ]</code>	一个整数，以像素为单位指定视频流的高度。
	<code>_name: String</code>	指示指定的 Video 对象的实例名称。
	<code>_parent: MovieClip</code>	指示包含当前 Video 对象的影片剪辑或对象。
	<code>_rotation: Number</code>	指示 Video 对象距其原始方向的旋转程度，以度为单位。
	<code>smoothing: Boolean</code>	指定在缩放视频时是否应进行平滑处理（插补数据）。
	<code>_visible: Boolean</code>	指示由 <code>my_video</code> 指定的 Video 对象是否可见。
	<code>_width: Number</code>	指示 Video 对象的宽度，以像素为单位。
	<code>width: Number [ 只读 ]</code>	一个整数，以像素为单位指定视频流的宽度。
	<code>_x: Number</code>	指示 Video 对象相对于父级影片剪辑的本地坐标的 x 坐标。
	<code>_xmouse: Number [ 只读 ]</code>	指示鼠标位置的 x 坐标。
	<code>_xscale: Number</code>	指示从 Video 对象注册点开始应用的 Video 对象的水平缩放比例 (percentage)。
	<code>_y: Number</code>	指示 Video 对象相对于父级影片剪辑的本地坐标的 y 坐标。
	<code>_ymouse: Number [ 只读 ]</code>	指示鼠标位置的 y 坐标。
	<code>_yscale: Number</code>	指示从 Video 对象注册点开始应用的 Video 对象的垂直缩放比例 (percentage)。

继承自 Object 类的属性

`constructor` (Object.constructor 属性), `__proto__` (Object.\_\_proto\_\_ 属性), `prototype` (Object.prototype 属性), `__resolve` (Object.\_\_resolve 属性)

方法摘要

修饰符	签名	说明
	<code>attachVideo(source:Object) : Void</code>	指定将在舞台上的 Video 对象的边界内显示的视频流 (source)。
	<code>clear() : Void</code>	清除该 Video 对象中当前显示的图像。

继承自 Object 类的方法

[addProperty](#) (Object.addProperty 方法), [hasOwnProperty](#) (Object.hasOwnProperty 方法), [isPropertyEnumerable](#) (Object.isPropertyEnumerable 方法), [isPrototypeOf](#) (Object.isPrototypeOf 方法), [registerClass](#) (Object.registerClass 方法), [toString](#) (Object.toString 方法), [unwatch](#) (Object.unwatch 方法), [valueOf](#) (Object.valueOf 方法), [watch](#) (Object.watch 方法)

\_alpha (Video.\_alpha 属性)

`public _alpha : Number`

指示指定的 Video 对象的 Alpha 透明度值。有效值为 0（完全透明）到 100（完全不透明）。默认值为 100。\_alpha 设置为 0 的影片剪辑中的对象即使不可见，也将处于活动状态。

可用性: ActionScript 1.0 ; Flash Player 8

另请参见

[\\_visible](#) (Video.\_visible 属性)

attachVideo (Video.attachVideo 方法)

`public attachVideo(source:Object) : Void`

指定将在舞台上的 Video 对象的边界内显示的视频流 (source)。视频流或者是通过 NetStream.play() 命令显示的 FLV 文件（即 Camera 对象），或者是 null。如果 source 为 null，则 Video 对象中将不再播放视频。

如果 FLV 文件只包含音频，则您无需使用此方法；当发出 NetStream.play() 命令时，FLV 文件的音频部分会自动播放。

如果要控制与 FLV 文件关联的音频，您可以使用 MovieClip.attachAudio() 将音频路由到影片剪辑；然后创建 Sound 对象来控制该音频的某些属性。有关更多信息，请参见 MovieClip.attachAudio()。

可用性: ActionScript 1.0 ; Flash Player 6

## 参数

**source:Object** - 正在捕获视频数据的 **Camera** 对象，或 **NetStream** 对象。要切断与该 **Video** 对象的连接，请为 **source** 传递 **null**。

## 示例

下面的示例在本地播放实时视频：

```
var my_video:Video; //my_video is a Video object on the Stage
var active_cam:Camera = Camera.get();
my_video.attachVideo(active_cam);
```

下面的示例播放一个以前录制的名为 **myVideo.flv** 的文件，该文件与 **SWF** 文件存储在同一个目录中。

```
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
```

## 另请参见

[Camera](#)，[NetStream](#)

## clear（Video.clear 方法）

```
public clear() : Void
```

清除该 **Video** 对象中当前显示的图像。例如，如果要显示待机信息，则可以使用此方法。这样就不必隐藏该 **Video** 对象了。

可用性：ActionScript 1.0；Flash Player 6

## 示例

下面的示例在用户单击 **pause\_btn** 实例时暂停并清除正在一个 **Video** 对象（名为 **my\_video**）中播放的 **video1.flv**。

```
var pause_btn:Button;
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
pause_btn.onRelease = function() {
    my_ns.pause();
    my_video.clear();
};
```

另请参见

[attachVideo](#) ([Video.attachVideo](#) 方法)

## deblocking (Video.deblocking 属性)

`public deblocking : Number`

指示作为后处理的一部分应用于已解码视频的解块过滤器的类型。有两个解块过滤器可供使用：一个在 **Sorenson** 编解码器中，另一个在 **On2 VP6** 编解码器中。下面是可接受的值：

- 0（默认值）- 允许视频压缩程序根据需要应用解块过滤器。
- 1 - 不使用任何解块过滤器。
- 2 - 使用 **Sorenson** 解块过滤器。
- 3 - 使用 **On2** 解块过滤器，不使用 **Deringing** 过滤器。
- 4 - 使用 **On2** 解块过滤器和快速 **On2 Deringing** 过滤器。
- 5 - 使用 **On2** 解块过滤器和更好的 **On2 Deringing** 过滤器。
- 6 - 与 5 相同。
- 7 - 与 5 相同。

如果在使用 **Sorenson** 编解码器时为视频选择 2 之后的某个模式，**Sorenson** 解码器会在内部默认采用模式 2。

使用解块过滤器会影响整体播放性能，而且对于高带宽视频通常不必要。如果您的系统不够强大，则在启用此过滤器的情况下播放视频可能会有困难。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例在 `my_video` 视频对象中播放 `video1.flv`，并且允许用户在 `video1.flv` 上更改解块过滤器行为。向您的文件添加名为 `my_video` 的视频对象和名为 `deblocking_cb` 的 **ComboBox** 实例，然后将以下 **ActionScript** 添加到您的 **FLA** 或 **AS** 文件。

```
var deblocking_cb:mx.controls.ComboBox;
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
```

```
deblocking_cb.addItem({data:0, label:'Auto'});
deblocking_cb.addItem({data:1, label:'No'});
deblocking_cb.addItem({data:2, label:'Yes'});
```

```
var cbListener:Object = new Object();
```



```
cbListener.change = function(evt:Object) {  
    my_video.deblocking = evt.target.selectedItem.data;  
};  
deblocking_cb.addEventListener("change", cbListener);
```

使用该 **ComboBox** 实例在 **video1.flv** 上更改解块过滤器行为。

## **\_height** (**Video.\_height** 属性)

```
public _height : Number
```

指示 **Video** 对象的高度，以像素为单位。

可用性: **ActionScript 1.0** ; **Flash Player 8**

另请参见

[\\_width \(Video.\\_width 属性\)](#)

## **height** (**Video.height** 属性)

```
public height : Number [read-only]
```

一个整数，以像素为单位指定视频流的高度。对于实时流，此值与正在捕获该视频流的 **Camera** 对象的 **Camera.height** 属性相同。对于 **FLV** 文件，此值是以 **FLV** 形式导出的文件的高度。

例如，您可能需要使用此属性来确保用户以捕获时的相同大小观看视频，而无论 **Video** 对象在舞台上的实际大小是什么。

可用性: **ActionScript 1.0** ; **Flash Player 6**

### 示例

下面的示例将 **Video** 元件实例的 **\_height** 和 **\_width** 属性设置为等于已加载的 **FLV** 文件的 **height** 和 **width**。

若要使用此实例，请首先使用实例名 “**myVideo**” 创建新的 **Video** 元件，然后将其放在与此脚本相同的上下文中。当触发 **NetStream.Play.Start** 状态代码时，**height** 和 **width** 属性将为零。通过在收到 **NetStream.Buffer.Full** 的状态后调整视频的大小，可以确保在显示视频的第一帧之前视频的大小是正确的。

```
var netConn:NetConnection = new NetConnection();  
netConn.connect(null);  
var netStrm:NetStream = new NetStream(netConn);
```

```
myVideo.attachVideo(netStrm);  
netStrm.play("Video.flv");
```

```
netStrm.onStatus = function(infoObject:Object) {
```

```

switch (infoObject.code) {
    case 'NetStream.Play.Start' :
    case 'NetStream.Buffer.Full' :
        myVideo._width = myVideo.width;
        myVideo._height = myVideo.height;
        break;
}
}

```

另请参见

[\\_height \(MovieClip.\\_height 属性\)](#), [width \(Video.width 属性\)](#)

## **\_name (Video.\_name 属性)**

```
public _name : String
```

指示指定的 **Video** 对象的实例名称。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## **\_parent (Video.\_parent 属性)**

```
public _parent : MovieClip
```

指示包含当前 **Video** 对象的影片剪辑或对象。当前对象是包含引用 `_parent` 的 **ActionScript** 代码的对象。使用 `_parent` 属性可指定一个相对路径,用以指向当前对象上级的影片剪辑或对象。

可以使用 `_parent` 在显示列表中上移多个级别,如下所示:

```
this._parent._parent._alpha = 20;
```

可用性: **ActionScript 1.0** ; **Flash Player 8**

另请参见

[\\_root 属性](#), [\\_target \(MovieClip.\\_target 属性\)](#)

## **\_rotation (Video.\_rotation 属性)**

```
public _rotation : Number
```

指示 **Video** 对象距其原始方向的旋转程度,以度为单位。从 **0** 到 **180** 的值表示顺时针方向旋转;从 **0** 到 **-180** 的值表示逆时针方向旋转。对于此范围之外的值,可以通过加上或减去 **360** 获得该范围内的值。例如,语句 `my_video._rotation = 450` 与 `my_video._rotation = 90` 是相同的。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## smoothing（Video.smoothing 属性）

public smoothing : Boolean

指定在缩放视频时是否应进行平滑处理（插补数据）。播放器必须处于高品质模式，平滑处理才起作用。默认值为 false（不进行平滑处理）。

可用性：ActionScript 1.0；Flash Player 6

### 示例

下面的示例使用一个按钮（名为 smoothing\_btn），当视频 my\_video 在 SWF 文件中播放时，切换应用于它的 **smoothing** 属性。创建一个名为 smoothing\_btn 的按钮，并将以下 **ActionScript** 添加到 FLA 或 AS 文件：

```
this.createTextField("smoothing_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
smoothing_txt.autoSize = true;

var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
my_ns.onStatus = function(info:Object:Object) {
    updateSmoothing();
};
smoothing_btn.onRelease = function() {
    my_video.smoothing = !my_video.smoothing;
    updateSmoothing();
};
function updateSmoothing():Void {
    smoothing_txt.text = "smoothing = "+my_video.smoothing;
}
```

此示例中使用的 MovieClip.getNextHighestDepth() 方法要求 **Flash Player 7** 或更高版本。如果您的 SWF 文件包括第 2 版的组件，请使用第 2 版的组件的 DepthManager 类而不是 MovieClip.getNextHighestDepth() 方法。

## \_visible（Video.\_visible 属性）

public \_visible : Boolean

指示由 my\_video 指定的 Video 对象是否可见。

可用性：ActionScript 1.0；Flash Player 8

## `_width` (`Video._width` 属性)

`public _width : Number`

指示 **Video** 对象的宽度，以像素为单位。

可用性: **ActionScript 1.0** ; **Flash Player 8**

另请参见

[\\_height](#) (`Video._height` 属性)

## `width` (`Video.width` 属性)

`public width : Number [read-only]`

一个整数，以像素为单位指定视频流的宽度。对于实时流，此值与正在捕获该视频流的 **Camera** 对象的 `Camera.width` 属性相同。对于 **FLV** 文件，此值是以 **FLV** 文件形式导出的文件的宽度。

例如，您可能需要使用此属性来确保用户以捕获时的相同大小观看视频，而无论 **Video** 对象在舞台上的实际大小是什么。

可用性: **ActionScript 1.0** ; **Flash Player 6**

示例

请参见 `Video.height` 的示例。

## `_x` (`Video._x` 属性)

`public _x : Number`

指示 **Video** 对象相对于父级影片剪辑的本地坐标的 `x` 坐标。如果 **Video** 对象在主时间轴中，则其坐标系统将舞台的左上角作为 (0, 0)。如果 **Video** 对象在具有变形的影片剪辑内，则该 **Video** 对象位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 90° 的影片剪辑，该影片剪辑的子级将继续继承逆时针旋转 90° 的坐标系统。**Video** 对象的坐标指的是注册点的位置。

可用性: **ActionScript 1.0** ; **Flash Player 8**

另请参见

[\\_xscale](#) (`Video._xscale` 属性), [\\_y](#) (`Video._y` 属性), [\\_yscale](#) (`Video._yscale` 属性)

## `_xmouse` (`Video._xmouse` 属性)

`public _xmouse : Number [read-only]`

指示鼠标位置的 `x` 坐标。

可用性: `ActionScript 1.0` ; `Flash Player 8`

另请参见

[Mouse](#), [\\_ymouse](#) (`Video._ymouse` 属性)

## `_xscale` (`Video._xscale` 属性)

`public _xscale : Number`

指示从 `Video` 对象注册点开始应用的 `Video` 对象的水平缩放比例 (percentage)。默认注册点为 (0,0)。

缩放本地坐标系将影响 `_x` 和 `_y` 属性设置，这些设置是以整像素定义的。

可用性: `ActionScript 1.0` ; `Flash Player 8`

另请参见

[\\_x](#) (`Video._x` 属性), [\\_y](#) (`Video._y` 属性), [\\_yscale](#) (`Video._yscale` 属性), [\\_width](#) (`Video._width` 属性)

## `_y` (`Video._y` 属性)

`public _y : Number`

指示 `Video` 对象相对于父级影片剪辑的本地坐标的 `y` 坐标。如果 `Video` 对象在主时间轴中，则其坐标系将舞台的左上角作为 (0, 0)。如果 `Video` 对象在具有变形的影片剪辑内，则该 `Video` 对象位于包含它的影片剪辑的本地坐标系中。因此，对于逆时针旋转 90° 的影片剪辑，该影片剪辑的子级将继承逆时针旋转 90° 的坐标系。`Video` 对象的坐标指的是注册点的位置。

可用性: `ActionScript 1.0` ; `Flash Player 8`

另请参见

[\\_x](#) (`Video._x` 属性), [\\_xscale](#) (`Video._xscale` 属性), [\\_yscale](#) (`Video._yscale` 属性)

## `_ymouse` (`Video._ymouse` 属性)

`public _ymouse : Number [read-only]`

指示鼠标位置的 `y` 坐标。

可用性: **ActionScript 1.0** ; **Flash Player 8**

另请参见

[Mouse](#), [\\_xmouse](#) (`Video._xmouse` 属性)

## `_yscale` (`Video._yscale` 属性)

`public _yscale : Number`

指示从 **Video** 对象注册点开始应用的 **Video** 对象的垂直缩放比例 (**percentage**)。默认注册点为 (0,0)。

缩放本地坐标系将影响 `_x` 和 `_y` 属性设置，这些设置是以整像素定义的。

可用性: **ActionScript 1.0** ; **Flash Player 8**

另请参见

[\\_x](#) (`Video._x` 属性), [\\_xscale](#) (`Video._xscale` 属性), [\\_y](#) (`Video._y` 属性), [\\_height](#) (`Video._height` 属性)

# XML

```
Object
|
+-XMLNode
|
+-XML
```

```
public class XML
extends XMLNode
```

使用 **XML** 类的方法和属性可以加载、分析、发送、生成和操作 **XML** 文档树。

您必须使用构造函数 `new XML()` 创建 **XML** 对象才能调用 **XML** 类的任何方法。

在 **Flash** 中, **XML** 文档用 **XML** 类来表示。层次结构文档的每个元素都由一个 **XMLNode** 对象来表示。

有关以下方法和属性的信息, 可参见 **XMLNode** 类: 特别是 `appendChild()`、`attributes`、`childNodes`、`cloneNode()`、`firstChild`、`hasChildNodes()`、`insertBefore()`、`lastChild`、`nextSibling`、`nodeName`、`nodeType`、`nodeValue`、`parentNode`、`previousSibling`、`removeNode()` 和 `toString()`。

在较早版本的“**ActionScript 语言参考**”中，上面这些方法和属性是记录在 **XML** 类中的。现在，它们记录在 **XMLNode** 类中。

注意：按照 **W3C DOM Level 1** 推荐方法，可以对 **XML** 和 **XMLNode** 对象建模，详情请见：<http://www.w3.org/tr/1998/REC-DOM-Level-1-19981001/level-one-core.html> 对 **XML** 和 **XMLNode** 对象建模。该推荐方法指定节点接口和文档接口。文档接口继承自节点接口，并添加了诸如 `createElement()` 和 `createTextNode()` 这样的方法。在 **ActionScript** 中，**XML** 和 **XMLNode** 对象旨在区分类似功能。

可用性：**ActionScript 1.0** ； **Flash Player 5**

另请参见

`appendChild` (`XMLNode.appendChild` 方法), `attributes` (`XMLNode.attributes` 属性), `childNodes` (`XMLNode.childNodes` 属性), `cloneNode` (`XMLNode.cloneNode` 方法), `firstChild` (`XMLNode.firstChild` 属性), `hasChildNodes` (`XMLNode.hasChildNodes` 方法), `insertBefore` (`XMLNode.insertBefore` 方法), `lastChild` (`XMLNode.lastChild` 属性), `nextSibling` (`XMLNode.nextSibling` 属性), `nodeName` (`XMLNode.nodeName` 属性), `nodeType` (`XMLNode.nodeType` 属性), `nodeValue` (`XMLNode.nodeValue` 属性), `parentNode` (`XMLNode.parentNode` 属性), `previousSibling` (`XMLNode.previousSibling` 属性), `removeNode` (`XMLNode.removeNode` 方法), `toString` (`XMLNode.toString` 方法)

属性摘要

修饰符	属性	说明
	<code>contentType:String</code>	调用 <code>XML.send()</code> 或 <code>XML.sendAndLoad()</code> 方法时发送到服务器的 <b>MIME</b> 内容类型。
	<code>docTypeDecl:String</code>	指定有关 <b>XML</b> 文档的 <b>DOCTYPE</b> 声明的信息。
	<code>idMap:Object</code>	包含分配了 <code>id</code> 属性的 <b>XML</b> 文件的节点的对象。
	<code>ignoreWhite:Boolean</code>	默认设置为 <code>false</code> 。
	<code>loaded:Boolean</code>	指示 <b>XML</b> 文档是否已成功加载的属性。
	<code>status:Number</code>	自动设置并返回一个数值，该数值指示 <b>XML</b> 文档是否被成功地分析为 <b>XML</b> 对象。
	<code>xmlDecl:String</code>	一个字符串，指定有关文档的 <b>XML</b> 声明的信息。

继承自 XMLNode 类的属性

attributes (XMLNode.attributes 属性), childNodes (XMLNode.childNodes 属性), firstChild (XMLNode.firstChild 属性), lastChild (XMLNode.lastChild 属性), localName (XMLNode.localName 属性), namespaceURI (XMLNode.namespaceURI 属性), nextSibling (XMLNode.nextSibling 属性), nodeName (XMLNode.nodeName 属性), nodeType (XMLNode.nodeType 属性), nodeValue (XMLNode.nodeValue 属性), parentNode (XMLNode.parentNode 属性), prefix (XMLNode.prefix 属性), previousSibling (XMLNode.previousSibling 属性)

继承自 Object 类的属性

constructor (Object.constructor 属性), \_\_proto\_\_ (Object.\_\_proto\_\_ 属性), prototype (Object.prototype 属性), \_\_resolve (Object.\_\_resolve 属性)

事件摘要

事件	说明
onData = function(src:String) {}	当 XML 文本从服务器完全下载后, 或当从服务器下载 XML 文本的过程中出现错误时调用。
onHTTPStatus = function(httpStatus:Number) { }	当 Flash Player 接收来自服务器的 HTTP 状态代码时调用。
onLoad = function(success:Boolean) { }	接收到来自服务器的 XML 文档时由 Flash Player 调用。

构造函数摘要

签名	说明
XML(text:String)	创建一个新的 XML 对象。



方法摘要

修饰符	签名	说明
	<code>addRequestHeader(header:Object, headerValue:String) : Void</code>	添加或更改与 POST 动作一起发送的 HTTP 请求标题（如 Content-Type 或 SOAPAction）。
	<code>createElement(name:String) : XMLNode</code>	使用参数中指定的名称创建一个新的 XML 元素。
	<code>createTextNode(value:String) : XMLNode</code>	使用指定的文本创建一个新的 XML 文本节点。
	<code>getBytesLoaded() : Number</code>	返回为 XML 文档加载（流式处理）的字节数。
	<code>getBytesTotal() : Number</code>	以字节为单位返回 XML 文档的大小。
	<code>load(url:String) : Boolean</code>	从指定的 URL 中加载 XML 文档，并使用下载的 XML 数据替换指定 XML 对象的内容。
	<code>parseXML(value:String) : Void</code>	分析 value 参数中指定的 XML 文本，并使用结果 XML 树填充指定的 XML 对象。
	<code>send(url:String, [target:String], [method:String]) : Boolean</code>	将指定的 XML 对象编码为 XML 文档并将其发送到指定的 target URL。
	<code>sendAndLoad(url:String, resultXML:XML) : Void</code>	将指定的 XML 对象编码为 XML 文档，使用 POST 方法将其发送到指定的 URL，下载服务器的响应，并将其加载到参数中指定的 resultXMLObject 中。

继承自 XMLNode 类的方法

`appendChild` (`XMLNode.appendChild` 方法), `cloneNode` (`XMLNode.cloneNode` 方法), `getNamespaceForPrefix` (`XMLNode.getNamespaceForPrefix` 方法), `getPrefixForNamespace` (`XMLNode.getPrefixForNamespace` 方法), `hasChildNodes` (`XMLNode.hasChildNodes` 方法), `insertBefore` (`XMLNode.insertBefore` 方法), `removeNode` (`XMLNode.removeNode` 方法), `toString` (`XMLNode.toString` 方法)

继承自 Object 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## addRequestHeader (XML.addRequestHeader 方法)

```
public addRequestHeader(header:Object, headerValue:String) : Void
```

添加或更改与 POST 动作一起发送的 **HTTP** 请求标题（如 Content-Type 或 SOAPAction）。在第一种用法中，向该方法传递了两个字符串：header 和 headerValue。在第二种用法中，传递了字符串、替代标头名称和标头值的数组。

如果通过多次调用来设置相同的标头名称，则每个后继值将替换在上一次调用中设置的值。

您不能使用此方法添加或更改下列标准 **HTTP** 标头：Accept-Ranges、Age、Allow、Allowed、Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、Vary、Via、Warning 和 WWW-Authenticate。

可用性：ActionScript 1.0；Flash Player 6

### 参数

**header:**Object - 一个字符串，表示 **HTTP** 请求标头名称。

**headerValue:**String - 一个字符串，表示与 header 关联的值。

### 示例

下面的示例将一个名为 SOAPAction 且值为 Foo 的自定义 **HTTP** 标头添加到名为 my\_xml 的 XML 对象：

```
my_xml.addRequestHeader("SOAPAction", "'Foo'");
```

下面的示例创建名为 headers 的数组，该数组包含两个可交替的 **HTTP** 标题及其关联值。该数组将作为参数被传递到 addRequestHeader() 方法。

```
var headers:Array = new Array("Content-Type", "text/plain", "X-ClientAppVersion", "2.0");
my_xml.addRequestHeader(headers);
```

### 另请参见

[addRequestHeader \(LoadVars.addRequestHeader 方法\)](#)

## contentType (XML.contentType 属性)

public contentType : String

调用 XML.send() 或 XML.sendAndLoad() 方法时发送到服务器的 MIME 内容类型。默认值为 application/x-www-form-urlencoded，它是用于大多数 HTML 格式的标准 MIME 内容类型。

可用性: ActionScript 1.0 ; Flash Player 6

### 示例

下面的示例创建一个新的 XML 文档并检查其默认内容类型:

```
// create a new XML document
var doc:XML = new XML();

// trace the default content type
trace(doc.contentType); // output: application/x-www-form-urlencoded
```

下面的示例定义一个 XML 包，并且为该 XML 对象设置内容类型。然后将数据发送到服务器，并在浏览器窗口中显示结果。

```
var my_xml:XML = new XML("<highscore><name>Ernie</name><score>13045</score></highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

按 F12 在浏览器中测试此示例。

### 另请参见

[send \(XML.send 方法\)](#)，[sendAndLoad \(XML.sendAndLoad 方法\)](#)

## createElement (XML.createElement 方法)

public createElement(name:String) : XMLNode

使用参数中指定的名称创建一个新的 XML 元素。新元素开始时没有父级、子级和同级。该方法返回一个对新创建的表示该元素的 XML 对象的引用。此方法和 XML.createTextNode() 方法都是为 XML 对象创建节点的构造函数方法。

可用性: ActionScript 1.0 ; Flash Player 5

### 参数

**name:String** - 要创建的 XML 元素的标签名。

### 返回

XMLNode - 一个 XMLNode 对象；一个 XML 元素。

## 示例

下面的示例使用 `createElement()` 方法创建三个 XML 节点：

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

trace(doc);
// output: <element1><element2 /><element3 /></element1>
```

另请参见

[createTextNode](#) ([XML.createTextNode](#) 方法)

## createTextNode (XML.createTextNode 方法)

```
public createTextNode(value:String) : XMLNode
```

使用指定的文本创建一个新的 XML 文本节点。该新节点开始时没有父级，且文本节点不能有子级或同级。此方法返回对表示新文本节点的 XML 对象的引用。此方法和 `XML.createElement()` 方法都是为 XML 对象创建节点的构造函数方法。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**value:**String - 一个字符串；用于创建新文本节点的文本。

### 返回

XMLNode - 一个 XMLNode 对象。

## 示例

下面的示例使用 `createTextNode()` 方法创建两个 XML 文本节点，并将它们放入现有的 XML 节点中：

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

// create two XML text nodes using createTextNode()
var textNode1:XMLNode = doc.createTextNode("textNode1 String value");
var textNode2:XMLNode = doc.createTextNode("textNode2 String value");

// place the new nodes into the XML tree
element2.appendChild(textNode1);
element3.appendChild(textNode2);

trace(doc);
// output (with line breaks added between tags):
// <element1>
// <element2>textNode1 String value</element2>
// <element3>textNode2 String value</element3>
// </element1>
```

## 另请参见

[createElement](#) (XML.createElement 方法)

## docTypeDecl (XML.docTypeDecl 属性)

```
public docTypeDecl : String
```

指定有关 XML 文档的 DOCTYPE 声明的信息。在已经将 XML 文本分析为 XML 对象后，该 XML 对象的 XML.docTypeDecl 属性就会被设置为该 XML 文档的 DOCTYPE 声明的文本（例如，<!DOCTYPE greeting SYSTEM "hello.dtd">）。使用 DOCTYPE 声明的字符串表示形式而不是 XML 节点对象设置该属性。

ActionScript 的 XML 分析程序不是具有验证功能的分析程序。分析程序读取 DOCTYPE 声明，并将其存储在 XML.docTypeDecl 属性中，但不执行 Dtd 验证。

如果在分析操作过程中未遇到 DOCTYPE 声明，则 XML.docTypeDecl 属性被设置为 undefined。XML.toString() 方法将在 XML 声明存储在 XML.xmlDecl 中后，并在输出该 XML 对象中的任何其它文本之前，立即输出 XML.docTypeDecl 的内容。如果未定义 XML.docTypeDecl，则不输出 DOCTYPE 声明。

可用性: ActionScript 1.0 ; Flash Player 5

### 示例

下面的示例使用 XML.docTypeDecl 属性来设置 XML 对象的 DOCTYPE 声明：

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

另请参见

[xmlDecl \(XML.xmlDecl 属性\)](#)

## getBytesLoaded (XML.getBytesLoaded 方法)

```
public getBytesLoaded() : Number
```

返回为 XML 文档加载（流式处理）的字节数。您可以将 getBytesLoaded() 的值与 getBytesTotal() 的值进行比较，以确定 XML 文档已加载的百分比。

可用性: ActionScript 1.0 ; Flash Player 6

### 返回

Number - 一个指示已加载的字节数的整数。

## 示例

下面的示例说明如何使用 `XML.getBytesLoaded()` 方法配合 `XML.getBytesTotal()` 方法来跟踪 `XML.load()` 命令的进度。您必须替换 `XML.load()` 命令的 **URL** 参数，以便该参数使用 **HTTP** 引用有效的 **XML** 文件。如果尝试用此示例加载驻留在硬盘上的本地文件，它可能无法正常运行，原因是 **Flash Player** 在测试影片模式下加载本地文件的所有内容。

```
// create a new XML document
var doc:XML = new XML();

var checkProgress = function(xmlObj:XML) {
    var bytesLoaded:Number = xmlObj.getBytesLoaded();
    var bytesTotal:Number = xmlObj.getBytesTotal();
    var percentLoaded:Number = Math.floor((bytesLoaded / bytesTotal ) 100);
    trace ("milliseconds elapsed: " + getTimer());
    trace ("bytesLoaded: " + bytesLoaded);
    trace ("bytesTotal: " + bytesTotal);
    trace ("percent loaded: " + percentLoaded);
    trace ("-----");
}

doc.onLoad = function(success:Boolean) {
    clearInterval(intervalID);
    trace("intervalID: " + intervalID);
}

doc.load("[place a valid URL pointing to an XML file here]");
var intervalID:Number = setInterval(checkProgress, 100, doc);
```

## 另请参见

[getBytesTotal](#) ([XML.getBytesTotal](#) 方法)

# getBytesTotal (XML.getBytesTotal 方法)

`public getBytesTotal() : Number`

以字节为单位返回 XML 文档的大小。

可用性: **ActionScript 1.0** ; **Flash Player 6**

## 返回

Number - 一个整数。

## 示例

请参见 `XML.getBytesLoaded()` 的示例。

## 另请参见

[getBytesLoaded](#) ([XML.getBytesLoaded](#) 方法)

## idMap (XML.idMap 属性)

`public idMap : Object`

包含分配了 `id` 属性的 XML 文件的节点的对象。对象（每个对象包含一个节点）的属性名称与 `id` 属性的值匹配。

请考虑下面的 XML 对象：

```
<employee id='41'>
  <name>
    John Doe
  </name>
  <address>
    601 Townsend St.
  </address>
</employee>

<employee id='42'>
  <name>
    Jane Q. Public
  </name>
</employee>
<department id="IT">
  Information Technology
</department>
```

在此示例中，该 XML 对象的 `idMap` 属性是具有以下三个属性的对象：41、42 和 IT。其中每个属性都是具有匹配的 `id` 值的 **XMLNode**。例如，`idMap` 对象的 `IT` 属性为下面的节点：

```
<department id="IT">
  Information Technology
</department>
```

您必须对此 XML 对象使用 `parse()` 方法才能对 `idMap` 属性进行实例化。

如果多个 **XMLNode** 具有相同的 `id` 值，则 `idNode` 对象的匹配属性是分析的最后一个节点的属性，如下所示：

```
var x1:XML = new XML("<a id='1'><b id='2' /><c id='1' /></a>");
x2 = new XML();
x2.parseXML(x1);
trace (x2.idMap['1']);
```

以下文本将输出 `<c>` 节点：

```
<c id='1' />
```

可用性：ActionScript 1.0；Flash Player 8



## 示例

您可以创建包含以下文本的名为 idMapTest.xml 的文本文件。

```
<?xml version="1.0"?>
<doc xml:base="http://example.org/today/" xmlns:xlink="http://www.w3.org/
  1999/xlink">
<head>
<title>Virtual Library</title>
</head>
<body>
<paragraph id="linkP1">See <link xlink:type="simple"
  xlink:href="new.xml">what's
new</link>!</paragraph>
<paragraph>Check out the hot picks of the day!</paragraph>
<olist xml:base="/hotpicks/">
<item>
<link id="foo" xlink:type="simple" xlink:href="pick1.xml">Hot Pick #1</
  link>
</item>
<item>
<link id="bar" xlink:type="simple" xlink:href="pick2.xml">Hot Pick #2</
  link>
</item>
<item>
<link xlink:type="simple" xlink:href="pick3.xml">Hot Pick #3</link>
</item>
</olist>
</body>
</doc>
```

然后，可以在 XML 文件所在的同一目录中创建一个 SWF 文件。您可以在该 SWF 文件中包含以下脚本。

```
var readXML = new XML();
readXML.load("idMapTest.xml");
readXML.onLoad = function(success) {
  myXML = new XML();
  myXML.parseXML(readXML);
  for (var x in myXML.idMap){
    trace('idMap.' + x + " = " + newline + myXML.idMap[x]);
    trace('_____ ' + newline);
  }
}
```

测试此 SWF 文件时，将生成以下输出。

```
idMap.bar =
<link id="bar" xlink:type="simple" xlink:href="pick2.xml">Hot Pick #2</
  link>
```

---

```
idMap.foo =
<link id="foo" xlink:type="simple" xlink:href="pick1.xml">Hot Pick #1</
  link>

idMap.linkP1 =
<paragraph id="linkP1">See <link xlink:type="simple"
  xlink:href="new.xml">what's
new</link>!</paragraph>
```

---

## ignoreWhite（XML.ignoreWhite 属性）

public ignoreWhite : Boolean

默认设置为 false。当设置为 true 时，在分析过程中将放弃仅包含空白的文本节点。带有前导或尾随空白的文本节点不受影响。

用法 1：可以为单个 XML 对象设置 ignoreWhite 属性，如下代码所示：

```
my_xml.ignoreWhite = true;
```

用法 2：可以为 XML 对象设置默认 ignoreWhite 属性，如下代码所示：

```
XML.prototype.ignoreWhite = true;
```

可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例加载一个 XML 文件，该文件带有一个只包含空白的文本节点：foyer 标签包含十四个空格字符。要运行此示例，请创建一个名为 **flooring.xml** 的文本文件，然后将以下标签复制到其中：

```
<house>
<kitchen> ceramic tile </kitchen>
<bathroom>linoleum</bathroom>
<foyer> </foyer>
</house>
```

创建一个名为 **flooring fla** 的新的 Flash 文档，然后将它保存到 XML 文件所在的同一目录。将以下代码放入主时间轴：

```
// Create a new XML object.
var flooring:XML = new XML();

// Set the ignoreWhite property to true (default value is false)
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object.
```

```

flooring.onload = function(success:Boolean) {
    trace(flooring);
}

// Load the XML into the flooring object.
flooring.load("flooring.xml");

// Output (line breaks added for clarity):
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer />
</house>

```

如果您以后将 `flooring.ignoreWhite` 的设置更改为 `false`，或者，只是整个删除该代码行，`foyer` 标签中的十四个空格字符将被保留：

```

...
// Set the ignoreWhite property to false (default value).
flooring.ignoreWhite = false;
...
// Output (line breaks added for clarity):
<house>
    <kitchen> ceramic tile </kitchen>
    <bathroom>linoleum</bathroom>
    <foyer> </foyer>
</house>

```

**ActionScript** 示例文件夹中的 `XML_blogTracker fla` 和 `XML_languagePicker fla` 文件也包含一个代码示例。此文件夹的典型路径如下：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples\ActionScript

## load (XML.load 方法)

```
public load(url:String) : Boolean
```

从指定的 URL 中加载 XML 文档，并使用下载的 XML 数据替换指定 XML 对象的内容。该 URL 是相对 URL，并使用 HTTP 进行调用。加载过程是异步的；它不会在执行 load() 方法后立即结束。

执行 load() 方法时，XML 对象的 loaded 属性被设置为 false。在 XML 数据下载完毕后，loaded 属性被设置为 true，并调用 onLoad 事件处理函数。直到 XML 数据完全下载后，才开始分析。如果该 XML 对象以前包含任何 XML 树，它们将被放弃。

您可以定义一个在调用 XML 对象的 onLoad 事件处理函数时执行的自定义函数。

注意：如果要下载的文件包含非 ASCII 字符（在许多非英语的语言中出现），建议您使用 UTF-8 或 UTF-16 编码（而不是诸如 ASCII 的非 Unicode 格式）来保存文件。

使用此方法时，请考虑 Flash Player 安全模型：

对于 Flash Player 8：

- 如果执行调用的 SWF 文件在只能与本地文件系统内容交互的沙箱中，而目标资源来自网络沙箱，则不允许进行数据加载。
- 如果执行调用的 SWF 文件来自网络沙箱而目标资源在本地，也不允许进行数据加载。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

对于 Flash Player 7 及更高版本，网站可通过跨域策略文件允许对资源进行跨域访问。在运行于 Flash Player 7 及更高版本中的任何版本的 SWF 文件中，url 参数必须正好在同一个域中。例如，位于 [www.someDomain.com](http://www.someDomain.com) 的 SWF 文件只能从也位于 [www.someDomain.com](http://www.someDomain.com) 的源中加载数据。

如果 SWF 文件运行于 Flash Player 7 以前版本的播放器中，则 url 参数必须与发出此调用的 SWF 文件位于同一个超级域中。超级域可以通过删除某一文件的 URL 最左侧的组件而得到。例如，位于 [www.someDomain.com](http://www.someDomain.com) 的 SWF 文件可以从位于 [store.someDomain.com](http://store.someDomain.com) 的源中加载数据，这是因为这两个文件都在同一个名为 [someDomain.com](http://someDomain.com) 的超级域中。

可用性：ActionScript 1.0；Flash Player 5

### 参数

url:String - 一个字符串，表示要加载的 XML 文档所在位置的 URL。如果发出此调用的 SWF 文件正在 Web 浏览器中运行，则 url 必须与 SWF 文件位于同一个域中。

## 返回

Boolean - 如果没有传递任何参数 (**null**)，则返回布尔值 false ； 否则返回 true。使用 onLoad() 事件处理函数可查看是否已成功加载 XML 文档。

## 示例

下面的代码示例使用 XML.load() 方法：

```
// Create a new XML object.
var flooring:XML = new XML();

// Set the ignoreWhite property to true (default value is false).
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object.
flooring.onLoad = function(success) {
    trace(flooring);
};

// Load the XML into the flooring object.
flooring.load("flooring.xml");
```

有关 **flooring.xml** 文件的内容，以及此示例生成的输出结果，请参见 XML.ignoreWhite 属性的示例。

## 另请参见

[ignoreWhite \(XML.ignoreWhite 属性\)](#)，[loaded \(XML.loaded 属性\)](#)，[onLoad \(XML.onLoad 处理函数\)](#)，[useCodepage \(System.useCodepage 属性\)](#)

## loaded (XML.loaded 属性)

public loaded : Boolean

指示 XML 文档是否已成功加载的属性。如果没有为 XML 对象定义自定义 onLoad() 事件处理函数，则当 XML.load() 调用启动的文档加载过程已成功完成时，**Flash Player** 将此属性设置为 true ； 否则，此属性为 false。不过，如果您为该 XML 对象的 onLoad() 事件处理函数定义一个自定义行为，则必须确保在该函数中设置 onload。

可用性：ActionScript 1.0 ； Flash Player 5

## 示例

下面的示例在一个简单脚本中使用 XML.loaded 属性。

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
    trace("success: "+success);
    trace("loaded: "+my_xml.loaded);
    trace("status: "+my_xml.status);
};
my_xml.load("http://www.flash-mx.com/mm/problems/products.xml");
```

当 **Flash** 调用 onLoad() 处理函数时，“输出”面板中将显示信息。如果该调用成功完成，“输出”面板中会为 loaded 的状态显示 true。

```
success: true
loaded: true
status: 0
```

另请参见

[load \(XML.load 方法\)](#), [onLoad \(XML.onLoad 处理函数\)](#)

## onData (XML.onData 处理函数)

当 XML 文本从服务器完全下载后，或当从服务器下载 XML 文本的过程中出现错误时调用。在分析 XML 之前调用此处理函数，因此您可以用它调用一个自定义分析例程，从而不必使用 **Flash XML** 分析程序。src 参数是一个包含从服务器下载的 XML 文本的字符串，除非下载过程出现错误，否则 src 参数为 undefined。

默认情况下，XML.onData 事件处理函数调用 XML.onLoad。您可以用自定义行为覆盖 XML.onData 事件处理函数，但是，除非您在 XML.onData 实现过程中调用 XML.onLoad，否则不会对其进行调用。

可用性: **ActionScript 1.0** ; **Flash Player 5**

## 参数

**src:String** - 一个字符串或 undefined；服务器发送的原始数据，通常为 XML 格式。

## 示例

下面的示例显示 XML.OnData 事件处理函数的默认外观：

```
XML.prototype.onData = function (src:String) {  
    if (src == undefined) {  
        this.onLoad(false);  
    } else {  
        this.parseXML(src);  
        this.loaded = true;  
        this.onLoad(true);  
    }  
}
```

您可以覆盖 XML.onData 事件处理函数，以截获 XML 文本而不对它进行分析。

## 另请参见

[onLoad \(XML.onLoad 处理函数\)](#)

## onHTTPStatus (XML.onHTTPStatus 处理函数)

当 Flash Player 接收来自服务器的 HTTP 状态代码时调用。使用此处理函数，您可以捕获 HTTP 状态代码并根据该状态代码进行操作。

onHTTPStatus 处理函数在 onData 前调用，它在加载失败时触发对值为 undefined 的 onLoad 的调用。要特别注意的是，在触发 onHTTPStatus 之后，随后始终触发 onData，而不管是否会覆盖 onHTTPStatus。若要充分地使用 onHTTPStatus 处理函数，请编写结果函数以捕获 onHTTPStatus 调用的结果；然后可以在 onData 或 onLoad 处理函数中使用该结果。如果未调用 onHTTPStatus，则表示播放器没有尝试发出 URL 请求。发生这种情况的原因是此请求违反了 SWF 的安全沙箱规则。

如果 Flash Player 无法从服务器获取状态代码或者 Flash Player 无法与服务器进行通讯，则将默认值 0 传递到您的 ActionScript 代码。在任何播放器中都可生成值 0（例如当请求了格式不正确的 URL 时），并且当 Flash Player 插件在以下浏览器（这些浏览器不会将 HTTP 状态代码传递到播放器）中运行时，值 0 将始终由 Flash Player 插件生成：用于 Macintosh 的 Netscape、Mozilla、Safari、Opera 或 Internet Explorer。

可用性：ActionScript 1.0；Flash Player 8

## 参数

httpStatus:Number - 由服务器返回的 HTTP 状态代码。例如，值 404 表示服务器尚未找到任何与请求的 URI 匹配的内容。在 <ftp://ftp.isi.edu/in-notes/rfc2616.txt> 处的 HTTP 规范的 10.4 和 10.5 节中，可以找到 HTTP 状态代码。

## 示例

下面的示例说明如何使用 `onHTTPStatus` 方法帮助调试。此示例收集 **HTTP** 状态代码并将它们的值和类型分配给 XML 对象的实例（请注意，本示例在运行时创建实例成员 `this.httpStatus` 和 `this.httpStatusType`）。`onData` 方法使用这些实例成员跟踪有关在调试时非常有用的 **HTTP** 响应的信息。

```
var myXml:XML = new XML();

myXml.onHTTPStatus = function(httpStatus:Number) {
    this.httpStatus = httpStatus;
    if(httpStatus < 100) {
        this.httpStatusType = "flashError";
    }
    else if(httpStatus < 200) {
        this.httpStatusType = "informational";
    }
    else if(httpStatus < 300) {
        this.httpStatusType = "successful";
    }
    else if(httpStatus < 400) {
        this.httpStatusType = "redirection";
    }
    else if(httpStatus < 500) {
        this.httpStatusType = "clientError";
    }
    else if(httpStatus < 600) {
        this.httpStatusType = "serverError";
    }
}

myXml.onData = function(src:String) {
    trace(">> " + this.httpStatusType + ": " + this.httpStatus);
    if(src != undefined) {
        this.parseXML(src);
        this.loaded = true;
        this.onLoad(true);
    }
    else {
        this.onLoad(false);
    }
}

myXml.onLoad = function(success:Boolean) {
}

myXml.load("http://weblogs.macromedia.com/mxna/xml/
    rss.cfm?query=byMostRecent&languages=1");
```



另请参见

[onHTTPStatus \(LoadVars.onHTTPStatus 处理函数\)](#), [load \(XML.load 方法\)](#), [sendAndLoad \(XML.sendAndLoad 方法\)](#)

## onLoad (XML.onLoad 处理函数)

接收到来自服务器的 XML 文档时由 **Flash Player** 调用。如果成功接收了 XML 文档, 则 `success` 参数为 `true`。如果未收到该文档, 或从服务器接收响应时出现错误, 则 `success` 参数为 `false`。默认情况下, 此方法的实现不处于活动状态。若要覆盖默认实现, 必须指定一个包含自定义动作的函数。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**success:** Boolean - 一个布尔值, 如果使用 `XML.load()` 或 `XML.sendAndLoad()` 操作成功加载了 XML 对象, 则值为 `true`; 否则为 `false`。

### 示例

下面的示例包括用于简单电子商务店面应用程序的 **ActionScript**。 `sendAndLoad()` 方法传输一个包含用户名和密码的 XML 元素, 并使用 `XML.onLoad` 处理函数来处理来自服务器的答复。

```
var login_str:String = "<login username=\""+username_txt.text+"\"
    password=\""+password_txt.text+"\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();

myLoginReply_xml.ignoreWhite = true;

myLoginReply_xml.onLoad = function(success:Boolean){

    if (success) {

        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
            (myLoginReply_xml.firstChild.attributes.success == "true")) {
            gotoAndStop("loggedIn");
        } else {
            gotoAndStop("loginFailed");
        }

    } else {
        gotoAndStop("connectionFailed");
    }

};
```

```
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",  
    myLoginReply_xml);
```

另请参见

[load \(XML.load 方法\)](#), [sendAndLoad \(XML.sendAndLoad 方法\)](#),

## parseXML (XML.parseXML 方法)

```
public parseXML(value:String) : Void
```

分析 value 参数中指定的 XML 文本, 并使用结果 XML 树填充指定的 XML 对象。XML 对象中任何现有的树将被放弃。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**value:**String - 一个字符串, 表示要分析并传递到指定的 XML 对象的 XML 文本。

### 示例

下面的示例创建并分析 XML 包:

```
var xml_str:String = "<state name=\"California\">  
<city>San Francisco</city></state>"  
  
// defining the XML source within the XML constructor:  
var my1_xml:XML = new XML(xml_str);  
trace(my1_xml.firstChild.attributes.name); // output: California  
  
// defining the XML source using the XML.parseXML method:  
var my2_xml:XML = new XML();  
my2_xml.parseXML(xml_str);  
trace(my2_xml.firstChild.attributes.name); // output: California
```

## send (XML.send 方法)

```
public send(url:String, [target:String], [method:String]) : Boolean
```

将指定的 XML 对象编码为 XML 文档并将其发送到指定的 target URL。

使用此方法时, 请考虑 **Flash Player** 安全模型:

- 对于 **Flash Player 8**, 如果执行调用的 SWF 文件在不受信任的本地沙箱中, 则不允许使用此方法。
- 对于 **Flash Player 7** 及更高版本, 如果执行调用的 SWF 文件是本地文件, 则不允许使用此方法。

有关更多信息, 请参见以下部分:

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章, “了解安全性”
- Flash Player 8 安全性白皮书 (位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security))
- Flash Player 8 与安全相关的 API 白皮书 (位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis))

可用性: ActionScript 1.0 ; Flash Player 5

## 参数

**url:String** - 指定 XML 对象的目标 URL。

**target:String** [ 可选 ] - 显示服务器返回的数据的浏览器窗口:

- `_self` 指定当前窗口中的当前帧。
- `_blank` 指定一个新窗口。
- `_parent` 指定当前帧的父级。
- `_top` 指定当前窗口中的顶级帧。

如果不指定 `target` 参数, 则与指定 `_self` 的效果相同。

**method:String** [ 可选 ] - HTTP 协议所使用的方法: "GET" 或 "POST"。在浏览器中, 默认值为 "POST"。在 Flash 测试环境中, 默认值为 "GET"。

## 返回

Boolean - 如果未指定任何参数, 将返回 `false`, 否则返回 `true`。

## 示例

下面的示例定义一个 XML 包并设置 XML 对象的内容类型。然后将数据发送到服务器, 并在浏览器窗口中显示结果。

```
var my_xml:XML = new XML("<highscore><name>Ernie</name><score>13045</score></highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

按 F12 在浏览器中测试此示例。

## 另请参见

[sendAndLoad \(XML.sendAndLoad 方法\)](#)

## sendAndLoad (XML.sendAndLoad 方法)

```
public sendAndLoad(url:String, resultXML:XML) : Void
```

将指定的 XML 对象编码为 XML 文档，使用 POST 方法将其发送到指定的 URL，下载服务器的响应，并将其加载到参数中指定的 resultXMLObject 中。服务器响应加载的方式与 XML.load() 方法使用的方式相同。

执行 sendAndLoad() 方法时，XML 对象的 loaded 属性被设置为 false。在 XML 数据下载完毕后，如果成功加载数据，则 loaded 属性被设置为 true，并调用 onLoad 事件处理函数。直到 XML 数据完全下载后，才开始分析。如果该 XML 对象以前包含任何 XML 树，这些树将被放弃。

使用此方法时，请考虑 Flash Player 安全模型：

对于 Flash Player 8：

- 如果执行调用的 SWF 文件在只能与本地文件系统内容交互的沙箱中，而目标资源来自网络沙箱，则不允许进行数据加载。
- 如果执行调用的 SWF 文件来自网络沙箱而目标资源在本地，也不允许进行数据加载。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

对于 Flash Player 7 及更高版本，网站可通过跨域策略文件允许对资源进行跨域访问。在运行于 Flash Player 7 及更高版本中的任何版本的 SWF 文件中，url 参数必须正好在同一个域中。例如，位于 [www.someDomain.com](http://www.someDomain.com) 的 SWF 文件只能从也位于 [www.someDomain.com](http://www.someDomain.com) 的源中加载数据。

如果 SWF 文件运行于 Flash Player 7 以前版本的播放器中，则 url 必须与正在发出此调用的 SWF 文件位于同一个超级域中。超级域可以通过删除某一文件的 URL 最左侧的组件而得到。例如，位于 [www.someDomain.com](http://www.someDomain.com) 的 SWF 文件可以从位于 [store.someDomain.com](http://store.someDomain.com) 的源中加载数据，这是因为这两个文件都在同一个名为 [someDomain.com](http://someDomain.com) 的超级域中。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**url:String** - 一个字符串；指定的 XML 对象的目标 URL。如果发出此调用的 SWF 文件正在 Web 浏览器上运行，则 url 必须与 SWF 文件位于同一个域中；有关详细信息，请参见“说明”部分。

**resultXML:XML** - 一个用 XML 构造函数方法创建的目标 XML 对象，它将接收来自服务器的返回信息。

## 示例

下面的示例包括用于简单电子商务店面应用程序的 **ActionScript**。XML.sendAndLoad() 方法传输一个包含用户名和密码的 XML 元素，并使用 onLoad 处理函数来处理来自服务器的答复。

```
var login_str:String = "<login username=\""+username_txt.text+"\"  
    password=\""+password_txt.text+"\" />";  
var my_xml:XML = new XML(login_str);  
var myLoginReply_xml:XML = new XML();  
myLoginReply_xml.ignoreWhite = true;  
myLoginReply_xml.onLoad = myOnLoad;  
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",  
    myLoginReply_xml);  
function myOnLoad(success:Boolean) {  
    if (success) {  
        if ((myLoginReply_xml.firstChild.nodeName == "packet") &&  
            (myLoginReply_xml.firstChild.attributes.success == "true")) {  
            gotoAndStop("loggedIn");  
        } else {  
            gotoAndStop("loginFailed");  
        }  
    } else {  
        gotoAndStop("connectionFailed");  
    }  
}
```

## 另请参见

[send \(XML.send 方法\)](#)，[load \(XML.load 方法\)](#)，[loaded \(XML.loaded 属性\)](#)，[onLoad \(XML.onLoad 处理函数\)](#)

## status (XML.status 属性)

public status : Number

自动设置并返回一个数值，该数值指示 XML 文档是否被成功地分析为 XML 对象。以下是数字状态代码和说明：

- 0 没有错误；成功地完成了分析。
- -2 一个 CDATA 部分没有正确结束。
- -3 XML 声明没有正确结束。
- -4 DOCTYPE 声明没有正确结束。
- -5 一个注释没有正确结束。
- -6 一个 XML 元素有格式错误。
- -7 内存不足。
- -8 一个属性值没有正确结束。
- -9 一个开始标签和结束标签不匹配。
- -10 遇到一个没有匹配的开始标签的结束标签。

可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例将一个 XML 包加载到 SWF 文件中。这时会显示一条状态消息，内容视 XML 的加载和分析成功与否而定。请将以下 **ActionScript** 添加到 FLA 或 AS 文件：

```
var my_xml:XML = new XML();
my_xml.onLoad = function(success:Boolean) {
    if (success) {
        if (my_xml.status == 0) {
            trace("XML was loaded and parsed successfully");
        } else {
            trace("XML was loaded successfully, but was unable to be parsed.");
        }
    }
    var errorMessage:String;
    switch (my_xml.status) {
        case 0 :
            errorMessage = "No error; parse was completed successfully.";
            break;
        case -2 :
            errorMessage = "A CDATA section was not properly terminated.";
            break;
        case -3 :
            errorMessage = "The XML declaration was not properly terminated.";
            break;
        case -4 :
            errorMessage = "The DOCTYPE declaration was not properly terminated.";
            break;
```

```

case -5 :
    errorMessage = "A comment was not properly terminated.";
    break;
case -6 :
    errorMessage = "An XML element was malformed.";
    break;
case -7 :
    errorMessage = "Out of memory.";
    break;
case -8 :
    errorMessage = "An attribute value was not properly terminated.";
    break;
case -9 :
    errorMessage = "A start-tag was not matched with an end-tag.";
    break;
case -10 :
    errorMessage = "An end-tag was encountered without a matching
        start-tag.";
    break;
default :
    errorMessage = "An unknown error has occurred.";
    break;
}
trace("status: "+my_xml.status+" (" +errorMessage+"");
} else {
    trace("Unable to load/parse XML. (status: "+my_xml.status+"");
}
};
my_xml.load("http://www.helpexamples.com/flash/badxml.xml");

```

## XML 构造函数

```
public XML(text:String)
```

创建一个新的 **XML** 对象。调用 **XML** 类的方法之前，必须使用该构造函数创建一个 **XML** 对象。

注意：使用 `createElement()` 和 `createTextNode()` 方法可以将元素和文本节点添加到 **XML** 文档树中。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**text:String** - 一个字符串；XML 文本分析它以创建新 **XML** 对象。

## 示例

下面的示例创建一个新的空 XML 对象：

```
var my_xml:XML = new XML();
```

下面的示例通过分析 source 参数中指定的 XML 文本来创建一个 XML 对象，并且使用得到的 XML 文档树来填充新创建的 XML 对象：

```
var other_xml:XML = new XML("<state name=\"California\"><city>San  
    Francisco</city></state>");
```

## 另请参见

[createElement](#) (XML.createElement 方法), [createTextNode](#) (XML.createTextNode 方法)

## xmlDecl (XML.xmlDecl 属性)

```
public xmlDecl : String
```

一个字符串，指定有关文档的 XML 声明的信息。将 XML 文档分析为 XML 对象之后，此属性被设置为文档的 XML 声明的文本。使用 XML 声明的字符串表示形式而不是 XML 节点对象设置该属性。如果在分析操作过程中未遇到 XML 声明，则该属性被设置为 undefined。XML.toString() 方法在输出 XML 对象中的其它文本之前输出 XML.xmlDecl 属性的内容。如果 XML.xmlDecl 属性包含 undefined 类型，则不输出 XML 声明。

可用性：ActionScript 1.0；Flash Player 5

## 示例

下面的示例创建一个名为 my\_txt 的文本字段，该字段的尺寸与舞台的尺寸相同。该文本字段显示加载到 SWF 文件中的 XML 包的属性。my\_txt 中会显示 doc 类型声明。请将以下 **ActionScript** 添加到 FLA 或 AS 文件：

```
var my_fmt:TextFormat = new TextFormat();  
my_fmt.font = "_typewriter";  
my_fmt.size = 12;  
my_fmt.leftMargin = 10;
```

```
this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0,  
    Stage.width, Stage.height);  
my_txt.border = true;  
my_txt.multiline = true;  
my_txt.wordWrap = true;  
my_txt.setTextFormat(my_fmt);
```

```
var my_xml:XML = new XML();  
my_xml.ignoreWhite = true;  
my_xml.onLoad = function(success:Boolean) {
```



```

var endTime:Number = getTimer();
var elapsedTime:Number = endTime-startTime;
if (success) {
    my_txt.text = "xmlDecl:"+newline+my_xml.xmlDecl+newline+newline;
    my_txt.text +=
"contentType:"+newline+my_xml.contentType+newline+newline;
    my_txt.text +=
"docTypeDecl:"+newline+my_xml.docTypeDecl+newline+newline;
    my_txt.text += "packet:"+newline+my_xml.toString()+newline+newline;
} else {
    my_txt.text = "Unable to load remote XML."+newline+newline;
}
my_txt.text += "loaded in: "+elapsedTime+" ms.";
};
my_xml.load("http://www.helpexamples.com/crossdomain.xml");
var startTime:Number = getTimer();

```

此示例中使用的 `MovieClip.getNextHighestDepth()` 方法要求 **Flash Player 7** 或更高版本。如果您的 **SWF** 文件包括第 2 版的组件，请使用第 2 版的组件的 **DepthManager** 类而不是 `MovieClip.getNextHighestDepth()` 方法。

另请参见

[docTypeDecl \(XML.docTypeDecl 属性\)](#)

## XMLNode

```

Object
|
+- XMLNode

```

```

public class XMLNode
extends Object

```

在 **Flash** 中，**XML** 文档用 **XML** 类来表示。层次结构文档的每个元素都由一个 **XMLNode** 对象来表示。

**可用性:** ActionScript 1.0 ; Flash Player 5

另请参见

[XML](#)

属性摘要

修饰符	属性	说明
	attributes:Object	一个对象，其中包含指定的 XML 实例的所有属性。
	childNodes:Array [ 只读 ]	指定 XML 对象的子级的数组。
	firstChild:XMLNode [ 只读 ]	计算指定的 XML 对象，并引用父级节点的子级列表中的第一个子级。
	lastChild:XMLNode [ 只读 ]	一个 XMLNode 值，它引用节点的子级列表中的最后一个子级。
	localName:String [ 只读 ]	XML 节点名称的本地名称部分。
	namespaceURI:String [ 只读 ]	如果 XML 节点具有前缀，则 namespaceURI 为该前缀 (URI) 的 xmlns 声明的值，通常称为命名空间 URI。
	nextSibling:XMLNode [ 只读 ]	一个 XMLNode 值，它引用父级节点的子级列表中的下一个同级。
	nodeName:String	一个字符串，它表示 XML 对象的节点名称。
	nodeType:Number [ 只读 ]	一个 nodeType 值，对于 XML 元素为 1，对于文本节点为 3。
	nodeValue:String	XML 对象的节点值。
	parentNode:XMLNode [ 只读 ]	一个 XMLNode 值，它引用指定 XML 对象的父级节点；如果该节点没有父级，则返回 null。
	prefix:String [ 只读 ]	XML 节点名称的前缀部分。
	previousSibling:XMLNode [ 只读 ]	一个 XMLNode 值，它引用父级节点的子级列表中的前一个同级。

继承自 Object 类的属性

`constructor` (`Object.constructor` 属性), `__proto__` (`Object.__proto__` 属性), `prototype` (`Object.prototype` 属性), `__resolve` (`Object.__resolve` 属性)

构造函数摘要

签名	说明
<code>XMLNode(type:Number, value:String)</code>	使用 XMLNode 构造函数，您可以根据指定 XML 节点内容的字符串和表示其节点类型的数字对该节点进行实例化。

方法摘要

修饰符	签名	说明
	<code>appendChild(newChild:XMLNode) : Void</code>	将指定的节点追加到 XML 对象的子级列表中。
	<code>cloneNode(deep:Boolean) : XMLNode</code>	构造并返回一个类型、名称、值和属性与指定的 XML 对象均相同的新 XML 节点。
	<code>getNamespaceForPrefix(prefix:String) : String</code>	返回与节点的指定前缀相关联的命名空间 URI。
	<code>getPrefixForNamespace(nsURI:String) : String</code>	返回与节点的指定命名空间 URI 相关联的前缀。
	<code>hasChildNodes() : Boolean</code>	指定 XML 对象是否具有子级节点。
	<code>insertBefore(newChild:XMLNode, insertPoint:XMLNode) : Void</code>	将 newChild 节点插入到 XML 对象的子级列表中，且在 insertPoint 节点之前。
	<code>removeNode() : Void</code>	从指定 XML 对象的父级中删除该对象。
	<code>toString() : String</code>	计算指定的 XML 对象，构造一个包括节点、子级和属性的 XML 结构的文本表示形式，并以字符串形式返回结果。

继承自 Object 类的方法

`addProperty` (`Object.addProperty` 方法), `hasOwnProperty` (`Object.hasOwnProperty` 方法), `isPropertyEnumerable` (`Object.isPropertyEnumerable` 方法), `isPrototypeOf` (`Object.isPrototypeOf` 方法), `registerClass` (`Object.registerClass` 方法), `toString` (`Object.toString` 方法), `unwatch` (`Object.unwatch` 方法), `valueOf` (`Object.valueOf` 方法), `watch` (`Object.watch` 方法)

## appendChild (XMLNode.appendChild 方法)

```
public appendChild(newChild:XMLNode) : Void
```

将指定的节点追加到 XML 对象的子级列表中。此方法直接在 childNode 参数引用的节点上操作；它并不追加该节点的副本。如果要追加的节点已经存在于另一个树结构中，则向新位置追加该节点会删除当前位置的该节点。如果 childNode 参数引用的节点已经存在于另一个 XML 树结构中，则在追加的子节点从现有父级节点删除后，它会被放在新的树结构中。

可用性：ActionScript 1.0；Flash Player 5

### 参数

**newChild:**XMLNode - 一个 XMLNode，表示要从其当前位置移动到 my\_xml 对象的子列表的节点。

### 示例

此示例按所显示的顺序进行以下操作：

- 创建两个空 XML 文档，doc1 和 doc2。
- 使用 createElement() 方法创建一个新节点，并使用 appendChild() 方法将该节点附加到名为 doc1 的 XML 文档。
- 说明如何使用 appendChild() 方法通过将根节点从 doc1 移动到 doc2 来移动节点。
- 从 doc2 克隆根节点并将它附加到 doc1。
- 创建一个新节点并将它附加到 XML 文档 doc1 的根节点。

```
var doc1:XML = new XML();
var doc2:XML = new XML();

// create a root node and add it to doc1
var rootnode:XMLNode = doc1.createElement("root");
doc1.appendChild(rootnode);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2:

// move the root node to doc2
doc2.appendChild(rootnode);
trace ("doc1: " + doc1); // output: doc1:
trace ("doc2: " + doc2); // output: doc2: <root />

// clone the root node and append it to doc1
var clone:XMLNode = doc2.firstChild.cloneNode(true);
doc1.appendChild(clone);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2: <root />

// create a new node to append to root node (named clone) of doc1
var newNode:XMLNode = doc1.createElement("newbie");
clone.appendChild(newNode);
trace ("doc1: " + doc1); // output: doc1: <root><newbie /></root>
```

## attributes (XMLNode.attributes 属性)

`public attributes : Object`

一个对象，其中包含指定的 XML 实例的所有属性。XML.attributes 对象为 XML 实例的每个属性包含一个变量。因为这些变量定义为该对象的一部分，所以通常将它们称为该对象的属性。每个属性的值以字符串形式保存在相应的属性中。例如，如果您有一个名为 **color** 的属性，则可以通过将 **color** 指定为属性名称来检索该属性的值，如下代码所示：

```
var myColor:String = doc.firstChild.attributes.color;
```

可用性：ActionScript 1.0 ； Flash Player 5

### 示例

下面的示例说明如何阅读和编写 XML 节点的属性：

```
var doc:XML = new XML("<mytag name='Val'> item </mytag>");
trace(doc.firstChild.attributes.name); // Val

doc.firstChild.attributes.order = "first";
trace (doc.firstChild); // <mytag order="first" name="Val"> item </mytag>

for (attr in doc.firstChild.attributes) {
    trace (attr + " = " + doc.firstChild.attributes[attr]);
}

// order = first
// name = Val
```

## childNodes (XMLNode.childNodes 属性)

`public childNodes : Array [read-only]`

指定 XML 对象的子级的数组。数组中的每个元素都是对表示子级节点的 XML 对象的引用。这是一个只读属性，不能用于操作子级节点。请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 来操作子节点。

对于文本节点 (`nodeType == 3`)，此属性未定义。

可用性：ActionScript 1.0 ； Flash Player 5

## 示例

下面的示例显示如何使用 `XML.childNodes` 属性来返回子节点的数组：

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// create an array and use rootNode to populate it
var firstArray:Array = doc.childNodes;
trace (firstArray);
// output: <rootNode><oldest /><middle /><youngest /></rootNode>

// create another array and use the child nodes to populate it
var secondArray:Array = rootNode.childNodes;
trace(secondArray);
// output: <oldest />,<middle />,<youngest />
```

## 另请参见

[nodeType \(XMLNode.nodeType 属性\)](#) , [appendChild \(XMLNode.appendChild 方法\)](#) ,  
[insertBefore \(XMLNode.insertBefore 方法\)](#) , [removeNode \(XMLNode.removeNode 方法\)](#)

## cloneNode (XMLNode.cloneNode 方法)

public cloneNode(deep:Boolean) : XMLNode

构造并返回一个类型、名称、值和属性与指定的 XML 对象均相同的新 XML 节点。如果将 deep 设置为 true, 则递归克隆所有子节点, 这将得到一个与原始对象文档树完全相同的副本。

返回的克隆节点与被克隆项目的树不再相关联。因此, nextSibling、parentNode 和 previousSibling 值都为 null。如果 deep 参数设置为 false, 或者 my\_xml 节点没有子节点, 则 firstChild 和 lastChild 同样为空。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 参数

**deep:Boolean** - 一个布尔值; 如果设置为 true, 将会以递归方式克隆指定 XML 对象的子级。

### 返回

XMLNode - 一个 XMLNode 对象。

### 示例

下面的示例说明如何使用 XML.cloneNode() 方法来创建节点的副本:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// create a copy of the middle node using cloneNode()
var middle2:XMLNode = middle.cloneNode(false);

// insert the clone node into rootNode between the middle and youngest nodes
rootNode.insertBefore(middle2, youngest);
trace(rootNode);
```

```

// output (with line breaks added):
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// </rootNode>

// create a copy of rootNode using cloneNode() to demonstrate a deep copy
var rootClone:XMLNode = rootNode.cloneNode(true);

// insert the clone, which contains all child nodes, to rootNode
rootNode.appendChild(rootClone);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// <rootNode>
// <oldest />
// <middle />
// <middle />
// <youngest />
// </rootNode>
// </rootNode>

```

## firstChild (XMLNode.firstChild 属性)

public firstChild : XMLNode [read-only]

计算指定的 XML 对象，并引用父级节点的子级列表中的第一个子级。如果该节点没有子级，则此属性为 null。如果该节点为文本节点，则此属性为 undefined。这是一个只读属性，不能用于操作子节点；请使用 appendChild()、insertBefore() 和 removeNode() 方法来操作子节点。

**可用性:** ActionScript 1.0 ; Flash Player 5

### 示例

下面的示例说明如何使用 XML.firstChild 遍历节点的子节点:

```

// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes

```



```

var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use firstChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode =
    aNode.nextSibling) {
    trace(aNode);
}

// output:
// <oldest />
// <middle />
// <youngest />

```

下面的示例来自 **Examples** 目录中的 **XML\_languagePicker.FLA** 文件，可在 **languageXML.onLoad** 事件处理函数功能定义中找到该示例：

```

// loop through the strings in each language node
// adding each string as a new element in the language array
for (var stringNode:XMLNode = childNode.firstChild; stringNode != null;
    stringNode = stringNode.nextSibling, j++) {
    masterArray[i][j] = stringNode.firstChild.nodeValue;
}

```

要查看整个脚本，请参见 **ActionScript** 示例文件夹中的 **XML\_languagePicker fla**：

- **Windows:** 引导驱动器 \Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\ActionScript
- **Macintosh:** Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples\ActionScript

另请参见

[appendChild \(XMLNode.appendChild 方法\)](#)，[insertBefore \(XMLNode.insertBefore 方法\)](#)，[removeNode \(XMLNode.removeNode 方法\)](#)

## getNamespaceForPrefix (XMLNode.getNamespaceForPrefix 方法)

public getNamespaceForPrefix(prefix:String) : String

返回与节点的指定前缀相关联的命名空间 URI。要确定该 URI，getPrefixForNamespace() 将根据需要从节点开始搜索 XML 层次结构，并返回给定 prefix 的第一个 xmlns 声明的命名空间 URI。

如果没有为指定的前缀定义命名空间，该方法将返回 null。

如果您指定了一个空字符串("") 作为 prefix，并且为该节点定义了默认命名空间（如 xmlns="http://www.example.com/" 中所示），该方法将返回该默认命名空间 URI。

可用性：ActionScript 1.0；Flash Player 8

### 参数

prefix:String - 该方法为其返回相关联的命名空间的前缀。

### 返回

String - 与指定的前缀相关联的命名空间。

### 示例

下面的示例创建一个非常简单的 XML 对象，并输出调用 getNamespaceForPrefix() 的结果。

```
function createXML():XMLNode {
    var str:String = "<Outer xmlns:exu=\"http://www.example.com/util\">"
        + "<exu:Child id='1' />"
        + "<exu:Child id='2' />"
        + "<exu:Child id='3' />"
        + "</Outer>";
    return new XML(str).firstChild;
}

var xml:XMLNode = createXML();
trace(xml.getNamespaceForPrefix("exu")); // output: http://www.example.com/
util
trace(xml.getNamespaceForPrefix("")); // output: null
```

### 另请参见

[getPrefixForNamespace](#) (XMLNode.getPrefixForNamespace 方法)，[namespaceURI](#) (XMLNode.namespaceURI 属性)

## getPrefixForNamespace (XMLNode.getPrefixForNamespace 方法)

```
public getPrefixForNamespace(nsURI:String) : String
```

返回与节点的指定命名空间 **URI** 相关联的前缀。为了确定前缀，`getPrefixForNamespace()` 根据需从节点开始搜索 **XML** 层次结构，并返回具有与 `nsURI` 匹配的命名空间 **URI** 的第一个 `xmlns` 声明的前缀。

如果未对给定 **URI** 进行 `xmlns` 赋值，该方法将返回 `null`。如果对给定 **URI** 进行了 `xmlns` 赋值但没有与该赋值相关联的前缀，该方法将返回一个空字符串 ("")。

可用性：ActionScript 1.0；Flash Player 8

### 参数

`nsURI:String` - 该方法为其返回相关联前缀的命名空间 **URI**。

### 返回

`String` - 与指定的命名空间相关联的前缀。

### 示例

下面的示例创建一个非常简单的 **XML** 对象，并输出调用 `getPrefixForNamespace()` 方法的结果。Outer **XML** 节点由 `xmlDoc` 变量表示，它定义一个命名空间 **URI** 并将其分配给 `exu` 前缀。使用已定义的命名空间 **URI** ("`http://www.example.com/util`") 调用 `getPrefixForNamespace()` 方法将返回前缀 `exu`，但使用未定义的 **URI** ("`http://www.example.com/other`") 调用此方法将返回 `null`。第一个 `exu:Child` 节点由 `child1` 变量表示，它也定义一个命名空间 **URI** ("`http://www.example.com/child`")，但是不会将其分配给一个前缀。对已定义但未分配的命名空间 **URI** 调用此方法将返回空字符串。

```
function createXML():XMLNode {
    var str:String = "<Outer xmlns:exu=\"http://www.example.com/util\">"
        + "<exu:Child id='1' xmlns=\"http://www.example.com/child\"/>"
        + "<exu:Child id='2' />"
        + "<exu:Child id='3' />"
        + "</Outer>";
    return new XML(str).firstChild;
}

var xmlDoc:XMLNode = createXML();
trace(xmlDoc.getPrefixForNamespace("http://www.example.com/util")); //
    output: exu
trace(xmlDoc.getPrefixForNamespace("http://www.example.com/other")); //
    output: null

var child1:XMLNode = xmlDoc.firstChild;
```

```
trace(child1.getPrefixForNamespace("http://www.example.com/child")); //
    output: [empty string]
trace(child1.getPrefixForNamespace("http://www.example.com/other")); //
    output: null
```

另请参见

[getNamespaceForPrefix \(XMLNode.getNamespaceForPrefix 方法\)](#) , [namespaceURI \(XMLNode.namespaceURI 属性\)](#)

## hasChildNodes (XMLNode.hasChildNodes 方法)

```
public hasChildNodes() : Boolean
```

指定 XML 对象是否具有子级节点。

可用性: **ActionScript 1.0** ; **Flash Player 5**

返回

Boolean - 如果指定的 **XMLNode** 具有一个或多个子节点, 则为 true ; 否则为 false。

示例

下面的示例创建新的 XML 包。如果根节点具有子级节点, 则代码将遍历每个子级节点以显示节点的名称和值。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件:

```
var my_xml:XML = new XML("hankrudolph");
if (my_xml.firstChild.hasChildNodes()) {
    // use firstChild to iterate through the child nodes of rootNode
    for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
        aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

下面是在 “输出” 面板中显示的内容:

```
output:
username: hank
password: rudolph
```

## insertBefore (XMLNode.insertBefore 方法)

```
public insertBefore(newChild:XMLNode, insertPoint:XMLNode) : Void
```

将 newChild 节点插入到 XML 对象的子级列表中，且在 insertPoint 节点之前。如果 insertPoint 不是 XMLNode 对象的子级，插入将失败。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 参数

**newChild**:XMLNode - 要插入的 XMLNode 对象。

**insertPoint**:XMLNode - 在调用该方法之后将要跟在 newChild 节点后面的 XMLNode 对象。

### 示例

以下代码在两个现有节点之间插入一个新的 XML 节点：

```
var my_xml:XML = new XML("<a>1</a>\n<c>3</c>");
var insertPoint:XMLNode = my_xml.lastChild;
var newNode:XML = new XML("<b>2</b>\n");
my_xml.insertBefore(newNode, insertPoint);
trace(my_xml);
```

### 另请参见

[XML](#), [cloneNode](#) ([XMLNode.cloneNode](#) 方法)

## lastChild (XMLNode.lastChild 属性)

```
public lastChild : XMLNode [read-only]
```

一个 XMLNode 值，它引用节点的子级列表中的最后一个子级。如果该节点没有子级，则 XML.lastChild 属性为 null。此属性不能用于处理子节点；请使用 appendChild()、insertBefore() 和 removeNode() 方法来处理子节点。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例使用 XML.lastChild 属性来循环访问 XML 节点的子节点，访问时从节点的子级列表中的最后一项开始，到节点的子级列表的第一个子级结束：

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
```

```

var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode =
    aNode.previousSibling) {
    trace(aNode);
}

// output:
// <youngest />
// <middle />
// <oldest />

```

下面的示例创建一个新的 XML 包，并使用 XML.lastChild 属性来循环访问根节点的子节点：

```

// create a new XML document
var doc:XML = new XML("");

var rootNode:XMLNode = doc.firstChild;

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null;
    aNode=aNode.previousSibling) {
    trace(aNode);
}

// output:
// <youngest />
// <middle />
// <oldest />

```

另请参见

[appendChild](#) (XMLNode.appendChild 方法), [insertBefore](#) (XMLNode.insertBefore 方法), [removeNode](#) (XMLNode.removeNode 方法), [XML](#)

## localName (XMLNode.localName 属性)

```
public localName : String [read-only]
```

XML 节点名称的本地名称部分。这是没有命名空间前缀的元素名称。例如，节点 `<contact:mailbox/>bob@example.com</contact:mailbox>` 具有本地名称 “**mailbox**” 和前缀 “**contact**”，两者组成完整的元素名称 “**contact.mailbox**”。

您可以通过 XML 节点对象的 `prefix` 属性来访问命名空间前缀。`nodeName` 属性会返回完整的名称（包括前缀和本地名称）。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例使用位于同一目录中的 SWF 文件和 XML 文件。名为 “`SoapSample.xml`” 的 XML 文件包含以下内容：

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
<soap:Body xmlns:w="http://www.example.com/weather">
<w:GetTemperature>
<w:City>San Francisco</w:City>
</w:GetTemperature>
</soap:Body>
</soap:Envelope>
```

包含以下脚本的 SWF 文件的源（请注意 **Output** 字符串的注释）：

```
var xmlDoc:XML = new XML()
xmlDoc.ignoreWhite = true;
xmlDoc.load("SoapSample.xml")
xmlDoc.onLoad = function(success:Boolean)
{
    var tempNode:XMLNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("w:GetTemperature localname: " + tempNode.localName); // Output: ...
    GetTemperature
    var soapEnvNode:XMLNode = xmlDoc.childNodes[0];
    trace("soap:Envelope localname: " + soapEnvNode.localName); // Output: ...
    Envelope
}
```

## namespaceURI (XMLNode.namespaceURI 属性)

```
public namespaceURI : String [read-only]
```

如果 XML 节点具有前缀，则 namespaceURI 为该前缀 (URI) 的 xmlns 声明的值，通常称为命名空间 URI。xmlns 声明位于当前节点中或 XML 层次结构中较高层次的节点中。

如果 XML 节点没有前缀，namespaceURI 属性的值将取决于是否定义了默认的命名空间（如 xmlns="http://www.example.com/" 中所示）。如果有默认命名空间，则 namespaceURI 属性的值即为默认命名空间的值。如果没有默认命名空间，则该节点的 namespaceURI 属性为空字符串 ("")。

您可以使用 getNamespaceForPrefix() 方法标识与特定前缀相关联的命名空间。

namespaceURI 属性会返回与节点名称相关联的前缀。

可用性：ActionScript 1.0；Flash Player 8

### 示例

下面的示例说明 namespaceURI 属性如何受使用前缀的影响。包含 SWF 文件和 XML 文件的目录。名为 SoapSample.xml 的 XML 文件包含以下标签。

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body xmlns:w="http://www.example.com/weather">
    <w:GetTemperature>
      <w:City>San Francisco</w:City>
    </w:GetTemperature>
  </soap:Body>
</soap:Envelope>
```

包含以下脚本的 SWF 文件的源（请注意 **Output** 字符串的注释）。对于表示 w:GetTemperature 节点的 tempNode，namespaceURI 的值在 soap:Body 标签中定义。对于表示 soap:Body 节点的 soapBodyNode，namespaceURI 的值由上一级节点中的 soap 前缀的定义确定，而不是由 soap:Body 节点包含的 w 前缀的定义确定。

```
var xmlDoc:XML = new XML();
xmlDoc.load("SoapSample.xml");
xmlDoc.ignoreWhite = true;
xmlDoc.onLoad = function(success:Boolean)
{
    var tempNode:XMLNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("w:GetTemperature namespaceURI: " + tempNode.namespaceURI);
    // Output: ... http://www.example.com/weather

    trace("w:GetTemperature soap namespace: " +
        tempNode.getNamespaceForPrefix("soap"));
    // Output: ... http://www.w3.org/2001/12/soap-envelope

    var soapBodyNode:XMLNode = xmlDoc.childNodes[0].childNodes[0];
```



```

        trace("soap:Envelope namespaceURI: " + soapBodyNode.namespaceURI);
        // Output: ... http://www.w3.org/2001/12/soap-envelope
    }

```

下面的示例使用不带前缀的 XML 标签。它使用位于同一目录中的 SWF 文件和 XML 文件。名为 NoPrefix.xml 的 XML 文件包含以下标签。

```

<?xml version="1.0"?>
<rootnode>
<simplenode xmlns="http://www.w3.org/2001/12/soap-envelope">
<innernode />
</simplenode>
</rootnode>

```

包含以下脚本的 SWF 文件的源（请注意 **Output** 字符串的注释）。rootNode 没有默认的命名空间，因此它的 namespaceURI 值是一个空字符串。simpleNode 定义一个默认命名空间，因此它的 namespaceURI 值是该默认命名空间。innerNode 不定义默认的命名空间，但是使用由 simpleNode 定义的默认命名空间，因此它的 namespaceURI 值与 simpleNode 的值相同。

```

var xmlDoc:XML = new XML();
xmlDoc.load("NoPrefix.xml");
xmlDoc.ignoreWhite = true;
xmlDoc.onLoad = function(success:Boolean)
{
    var rootNode:XMLNode = xmlDoc.childNodes[0];
    trace("rootNode Node namespaceURI: " + rootNode.namespaceURI);
    // Output: [empty string]

    var simpleNode:XMLNode = xmlDoc.childNodes[0].childNodes[0];
    trace("simpleNode Node namespaceURI: " + simpleNode.namespaceURI);
    // Output: ... http://www.w3.org/2001/12/soap-envelope

    var innerNode:XMLNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("innerNode Node namespaceURI: " + innerNode.namespaceURI);
    // Output: ... http://www.w3.org/2001/12/soap-envelope
}

```

另请参见

[getNamespaceForPrefix \(XMLNode.getNamespaceForPrefix 方法\)](#),  
[getPrefixForNamespace \(XMLNode.getPrefixForNamespace 方法\)](#)

## nextSibling (XMLNode.nextSibling 属性)

public nextSibling : XMLNode [read-only]

一个 **XMLNode** 值，它引用父级节点的子级列表中的下一个同级。如果该节点没有下一个同级节点，则此属性为 `null`。此属性不能用于处理子节点；请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 方法来处理子节点。

可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例摘自 `XML.firstChild` 属性的示例，并且说明如何使用 `XML.nextSibling` 属性来遍历 XML 节点的子节点：

```
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode =
    aNode.nextSibling) {
    trace(aNode);
}
```

### 另请参见

[firstChild \(XMLNode.firstChild 属性\)](#)，[appendChild \(XMLNode.appendChild 方法\)](#)，[insertBefore \(XMLNode.insertBefore 方法\)](#)，[removeNode \(XMLNode.removeNode 方法\)](#)，[XML](#)

## nodeName (XMLNode.nodeName 属性)

public nodeName : String

一个字符串，它表示 XML 对象的节点名称。如果 XML 对象是一个 XML 元素 (`nodeType == 1`)，则 `nodeName` 是 XML 文件中表示节点的标签的名称。例如，`TITLE` 是 HTML `TITLE` 标签的 `nodeName`。如果 XML 对象是一个文本节点 (`nodeType == 3`)，则 `nodeName` 为 `null`。

可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例创建一个元素节点和一个文本节点，并且检查每个节点的节点名称：

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);
```

```
// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeName);
trace(myTextNode.nodeName);

// output:
// rootNode
// null
```

下面的示例创建新的 **XML** 包。如果根节点具有子级节点，则代码将遍历每个子级节点以显示节点的名称和值。请将以下 **ActionScript** 添加到 **FLA** 或 **AS** 文件：

```
var my_xml:XML = new XML("hankrudolph");
if (my_xml.firstChild.hasChildNodes()) {
    // use firstChild to iterate through the child nodes of rootNode
    for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
        aNode=aNode.nextSibling) {
        if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

“输出”面板中会显示以下节点名称：

```
output:
username: hank
password: rudolph
```

另请参见

[nodeType \(XMLNode.nodeType 属性\)](#)

# nodeType (XMLNode.nodeType 属性)

public nodeType : Number [read-only]

一个 nodeType 值，对于 XML 元素为 1，对于文本节点为 3。

nodeType 是来自 W3C DOM Level 1 推荐方法: [www.w3.org/tr/1998/REC-DOM-Level-1-19981001/level-one-core.html](http://www.w3.org/tr/1998/REC-DOM-Level-1-19981001/level-one-core.html) 中 NodeType 枚举的一个数值。下表列出了这些值:

整数值	已定义的常量
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	ENTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

在 Flash Player 中，内置 XML 类仅支持 1 (ELEMENT\_NODE) 和 3 (TEXT\_NODE)。

可用性: ActionScript 1.0 ; Flash Player 5

## 示例

下面的示例创建一个元素节点和一个文本节点，并且检查各个节点的节点类型:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
```

```
myNode.appendChild(myTextNode);
```

```
trace(myNode.nodeType);  
trace(myTextNode.nodeType);
```

```
// output:  
// 1  
// 3
```

另请参见

[nodeValue](#) ([XMLNode.nodeValue](#) 属性)

## nodeValue (XMLNode.nodeValue 属性)

```
public nodeValue : String
```

**XML** 对象的节点值。如果 **XML** 对象是一个文本节点，则 `nodeType` 为 **3**，`nodeValue` 是节点的文本。如果 **XML** 对象是一个 **XML** 元素 (`nodeType` 为 **1**)，则 `nodeValue` 为 `null` 且只读。

可用性: **ActionScript 1.0** ; **Flash Player 5**

### 示例

下面的示例创建一个元素节点和一个文本节点，并且检查各个节点的节点值：

```
// create an XML document  
var doc:XML = new XML();  
  
// create an XML node using createElement()  
var myNode:XMLNode = doc.createElement("rootNode");  
  
// place the new node into the XML tree  
doc.appendChild(myNode);  
  
// create an XML text node using createTextNode()  
var myTextNode:XMLNode = doc.createTextNode("textNode");  
  
// place the new node into the XML tree  
myNode.appendChild(myTextNode);  
  
trace(myNode.nodeValue);  
trace(myTextNode.nodeValue);  
  
// output:  
// null  
// myTextNode
```

下面的示例创建一个 **XML** 包并对它进行分析。该代码将遍历每个子节点，并使用 `firstChild` 属性和 `firstChild.nodeValue` 显示节点值。当使用 `firstChild` 显示节点内容时，它将保持 `&#` 实体。不过，当您显式使用 `nodeValue` 时，它会转换为与号字符 (`&`)。

```
var my_xml:XML = new XML("mortongood&evil");
trace("using firstChild:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
    trace("\t"+my_xml.firstChild.childNodes[i].firstChild);
}
trace("");
trace("using firstChild.nodeValue:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
    trace("\t"+my_xml.firstChild.childNodes[i].firstChild.nodeValue);
}
```

“输出”面板中会显示以下信息：

```
using firstChild:
morton
good&evil

using firstChild.nodeValue:
morton
good&evil
```

另请参见

[nodeType](#) ([XMLNode.nodeType](#) 属性)

## parentNode (XMLNode.parentNode 属性)

`public parentNode : XMLNode [read-only]`

一个 **XMLNode** 值，它引用指定 **XML** 对象的父级节点；如果该节点没有父级，则返回 `null`。这是一个只读属性，不能用于操作子节点；请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 方法来操作子节点。

可用性：ActionScript 1.0 ； Flash Player 5

## 示例

下面的示例创建一个 XML 包并在“输出”面板中显示 **username** 节点的父节点：

```
var my_xml:XML = new XML("mortongood&evil");

// first child is the <login /> node
var rootNode:XMLNode = my_xml.firstChild;

// first child of the root is the <username /> node
var targetNode:XMLNode = rootNode.firstChild;
trace("the parent node of '"+targetNode.nodeName+"' is:
      '"+targetNode.parentNode.nodeName);
trace("contents of the parent node are:\n"+targetNode.parentNode);

输出（为清楚起见，对输出内容进行了换行）：
the parent node of 'username' is: login
contents of the parent node are:
morton
good&evil
```

## 另请参见

[appendChild \(XMLNode.appendChild 方法\)](#)，[insertBefore \(XMLNode.insertBefore 方法\)](#)，[removeNode \(XMLNode.removeNode 方法\)](#)，[XML](#)

## prefix (XMLNode.prefix 属性)

public prefix : String [read-only]

XML 节点名称的前缀部分。例如，节点 `<contact:mailbox/>bob@example.com</contact:mailbox>` 具有前缀“**contact**”和本地名称“**mailbox**”，两者组成完整的元素名称“**contact.mailbox**”。

XML 节点对象的 `nodeName` 属性会返回完整名称（包括前缀和本地名称）。您可以通过 `localName` 属性来访问元素名称的本地名称部分。

可用性：ActionScript 1.0；Flash Player 8

## 示例

包含 SWF 文件和 XML 文件的目录。名为 “SoapSample.xml” 的 XML 文件包含以下内容：

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
<soap:Body xmlns:w="http://www.example.com/weather">
<w:GetTemperature>
<w:City>San Francisco</w:City>
</w:GetTemperature>
</soap:Body>
</soap:Envelope>
```

包含以下脚本的 SWF 文件的源（请注意 **Output** 字符串的注释）：

```
var xmlDoc:XML = new XML();
xmlDoc.ignoreWhite = true;
xmlDoc.load("SoapSample.xml");
xmlDoc.onLoad = function(success:Boolean)
{
    var tempNode:XMLNode = xmlDoc.childNodes[0].childNodes[0].childNodes[0];
    trace("w:GetTemperature prefix: " + tempNode.prefix); // Output: ... w
    var soapEnvNode:XMLNode = xmlDoc.childNodes[0];
    trace("soap:Envelope prefix: " + soapEnvNode.prefix); // Output: ... soap
}
```

## previousSibling（XMLNode.previousSibling 属性）

public previousSibling : XMLNode [read-only]

一个 **XMLNode** 值，它引用父级节点的子级列表中的前一个同级。如果该节点没有前一个同级节点，则此属性的值为 **null**。此属性不能用于处理子节点；请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 方法来处理子节点。

可用性：ActionScript 1.0；Flash Player 5

## 示例

下面的示例摘自 `XML.lastChild` 属性的示例，并且说明如何使用 `XML.previousSibling` 属性来遍历 XML 节点的子节点：

```
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode =
    aNode.previousSibling) {
    trace(aNode);
}
```



另请参见

[lastChild](#) (XMLNode.lastChild 属性) , [appendChild](#) (XMLNode.appendChild 方法) , [insertBefore](#) (XMLNode.insertBefore 方法) , [removeNode](#) (XMLNode.removeNode 方法) , [XML](#)

## removeNode (XMLNode.removeNode 方法)

```
public removeNode() : Void
```

从指定 XML 对象的父级中删除该对象。还将删除此节点的所有子级节点。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 示例

下面的示例创建一个 XML 包，然后删除指定的 XML 对象及其子级节点：

```
var xml_str:String = "<state name=\"California\"><city>San Francisco</city></state>";

var my_xml:XML = new XML(xml_str);
var cityNode:XMLNode = my_xml.firstChild.firstChild;
trace("before XML.removeNode():\n"+my_xml);
cityNode.removeNode();
trace("");
trace("after XML.removeNode():\n"+my_xml);

// output (line breaks added for clarity):
//
// before XML.removeNode():
// <state name="California">
// <city>San Francisco</city>
// </state>
//
// after XML.removeNode():
// <state name="California" />
```

## toString (XMLNode.toString 方法)

```
public toString() : String
```

计算指定的 XML 对象，构造一个包括节点、子级和属性的 XML 结构的文本表示形式，并以字符串形式返回结果。

对于顶级 XML 对象（那些用构造函数创建的对象），XML.toString() 方法首先输出文档的 XML 声明（存储在 XML.xmlDecl 属性中），随后输出文档的 DOCTYPE 声明（存储在 XML.docTypeDecl 属性中），然后输出该对象中所有 XML 节点的文本表示形式。如果未定义 XML.xmlDecl 属性，则不输出 XML 声明。如果 XML.docTypeDecl 属性为 undefined，则不输出 DOCTYPE 声明。

可用性: **ActionScript 1.0 ; Flash Player 5**

### 返回

String - 字符串。

### 示例

以下代码使用 toString() 方法将 XMLNode 对象转换为 String，然后使用 String 类的 toUpperCase() 方法：

```
var xString = "<first>Mary</first>"
            + "<last>Ng</last>"
var my_xml:XML = new XML(xString);
var my_node:XMLNode = my_xml.childNodes[1];
trace(my_node.toString().toUpperCase());
// output: <LAST>NG</LAST>
```

### 另请参见

[docTypeDecl](#) (XML.docTypeDecl 属性)，[xmlDecl](#) (XML.xmlDecl 属性)

# XMLNode 构造函数

```
public XMLNode(type:Number, value:String)
```

使用 **XMLNode** 构造函数，您可以根据指定 **XML** 节点内容的字符串和表示其节点类型的数字对该节点进行实例化。

可用性: **ActionScript 1.0** ; **Flash Player 8**

## 参数

**type:**Number - 一个表示节点类型的整数:

整数值	已定义的常量
1	ELEMENT_NODE
2	ATTRIBUTE_NODE
3	TEXT_NODE
4	CDATA_SECTION_NODE
5	NTITY_REFERENCE_NODE
6	ENTITY_NODE
7	PROCESSING_INSTRUCTION_NODE
8	COMMENT_NODE
9	DOCUMENT_NODE
10	DOCUMENT_TYPE_NODE
11	DOCUMENT_FRAGMENT_NODE
12	NOTATION_NODE

在 **Flash Player** 中，**XML** 类只支持节点类型 **1** (**ELEMENT\_NODE**) 和 **3** (**TEXT\_NODE**)。

**value:**String - 对于文本节点，这是节点的文本；对于元素节点，这是标签的内容。

## 示例

```
var ELEMENT_NODE:Number = 1;
var node1:XMLNode = new XMLNode(ELEMENT_NODE, "fullName");

var TEXT_NODE:Number = 3;
var node2:XMLNode = new XMLNode(TEXT_NODE, "Justin Case");

// Create a new XML document
var doc:XML = new XML();

// Create a root node
var rootNode:XMLNode = doc.createElement("root");

// Add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// Build the rest of the document:
rootNode.appendChild(node1);
node1.appendChild(node2);

trace(doc);

// Output: Justin Case
```

# XMLSocket

```
Object
|
+-XMLSocket
```

```
public class XMLSocket
extends Object
```

**XMLSocket** 类可实现客户端套接字，这使得运行 **Flash Player** 的计算机可以与由 **IP** 地址或域名标识的服务器计算机进行通讯。对于要求滞后时间较短的客户端 / 服务器应用程序，如实时聊天系统，**XMLSocket** 类非常有用。传统的基于 **HTTP** 的聊天解决方案频繁轮询服务器，并使用 **HTTP** 请求来下载新的消息。与此相对照，**XMLSocket** 聊天解决方案保持与服务器的开放连接，这一连接允许服务器即时发送传入的消息，而无需客户端发出请求。若要使用 **XMLSocket** 类，服务器计算机必须运行可识别 **XMLSocket** 类使用的协议的守护程序。下面的列表说明了该协议：

- **XML** 消息通过全双工 **TCP/IP** 流套接字连接发送。
- 每个 **XML** 消息都是一个完整的 **XML** 文档，以一个零 (0) 字节结束。
- 通过一个 **XMLSocket** 连接发送和接收的 **XML** 消息的数量没有限制。

XMLSocket 对象连接到服务器的方式和位置受以下限制：

- XMLSocket.connect() 方法只能连接到端口号大于或等于 1024 的 TCP 端口。这种限制的一个后果是，与 XMLSocket 对象通讯的服务器守护程序也必须分配到端口号大于或等于 1024 的端口。端口号小于 1024 的端口通常用于系统服务（如 FTP、Telnet 和 HTTP），因此，出于安全方面的考虑，禁止 XMLSocket 对象使用这些端口。这种端口号方面的限制可以减少不恰当地访问和滥用这些资源的可能性。
- XMLSocket.connect() 方法只能连接到 SWF 文件所在的同一个域中的计算机。这一限制不适用于在本地磁盘外运行的 SWF 文件。（此限制与 loadVariables()、XML.sendAndLoad() 和 XML.load() 的安全规则相同。）若要连接到在 SWF 所在域之外的其它域中运行的服务器守护程序，可以在该服务器上创建一个允许从特定域进行访问的安全策略文件。

将服务器设置为与 XMLSocket 对象进行通讯可能会遇到一些困难。如果您的应用程序不需要进行实时交互，请使用 loadVariables() 函数或 Flash 的基于 HTTP 的 XML 服务器连接（XML.load()、XML.sendAndLoad()、XML.send()），而不要使用 XMLSocket 类。若要使用 XMLSocket 类的方法，您必须首先使用构造函数 new XMLSocket 创建一个 XMLSocket 对象。

可用性：ActionScript 1.0；Flash Player 5

属性摘要

继承自 Object 类的属性

---

constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性), prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
--

---

事件摘要

事件	说明
onClose = function() {}	仅在服务器关闭某个打开的连接时调用。
onConnect = function(success:Boolean) {}	在通过 XMLSocket.connect() 启动的连接请求成功或失败后，由 Flash Player 调用。
onData = function(src:String) {}	当某个消息已从服务器下载并以零 (0) 字节结束时调用。
onXML = function(src:XML) {}	在包含 XML 文档的指定 XML 对象通过打开的 XMLSocket 连接到达时，由 Flash Player 调用。

构造函数摘要

签名	说明
XMLSocket()	创建一个新的 XMLSocket 对象。

方法摘要

修饰符	签名	说明
	<code>close() : Void</code>	关闭由 <code>XMLSocket</code> 对象指定的连接。
	<code>connect(url:String, port:Number) : Boolean</code>	使用指定的 TCP 端口（必须为 1024 或更高的端口）建立一个到指定 Internet 主机的连接，并根据是否成功建立了连接返回 <code>true</code> 或 <code>false</code> 。
	<code>send(data:Object) : Void</code>	将在 <code>object</code> 参数中指定的 XML 对象或数据转换成一个字符串，并将其传输到服务器，后面跟有一个零 (0) 字节。

继承自 `Object` 类的方法

[addProperty \(Object.addProperty 方法\)](#), [hasOwnProperty \(Object.hasOwnProperty 方法\)](#), [isPropertyEnumerable \(Object.isPropertyEnumerable 方法\)](#), [isPrototypeOf \(Object.isPrototypeOf 方法\)](#), [registerClass \(Object.registerClass 方法\)](#), [toString \(Object.toString 方法\)](#), [unwatch \(Object.unwatch 方法\)](#), [valueOf \(Object.valueOf 方法\)](#), [watch \(Object.watch 方法\)](#)

## close（XMLSocket.close 方法）

```
public close() : Void
```

关闭由 `XMLSocket` 对象指定的连接。

可用性: `ActionScript 1.0`；`Flash Player 5`

### 示例

以下简单示例创建一个 `XMLSocket` 对象，尝试连接到服务器，然后关闭连接。

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.close();
```

### 另请参见

[connect \(XMLSocket.connect 方法\)](#)

## connect (XMLSocket.connect 方法)

```
public connect(url:String, port:Number) : Boolean
```

使用指定的 TCP 端口（必须为 1024 或更高的端口）建立一个到指定 Internet 主机的连接，并根据是否成功建立了连接返回 true 或 false。如果您不知道 Internet 主机的端口号，请与您的网络管理员联系。

如果您为 host 参数指定 null，则与调用 XMLSocket.connect() 的 SWF 文件所在的主机连接。例如，如果 SWF 文件是从 **www.yoursite.com** 下载的，则将 host 参数指定为 null 与输入 **www.yoursite.com** 的 IP 地址效果相同。

如果 SWF 文件运行于 Flash Player 7 以前版本的播放器中，则 host 必须与发出此调用的 SWF 文件位于同一个超级域中。例如，位于 **www.someDomain.com** 的 SWF 文件可以从位于 **store.someDomain.com** 的源中加载数据，这是因为这两个文件都在同一个名为 **someDomain.com** 的超级域中。

如果任何版本的 SWF 文件运行于 Flash Player 7 或更高版本中，则 host 必须位于完全相同的域中。例如，位于 **www.someDomain.com** 中的为 Flash Player 5 发布但运行于 Flash Player 7 或更高版本中的 SWF 文件只能从同样位于 **www.someDomain.com** 中的 SWF 文件中加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。

执行 load() 时，XML 对象的 loaded 属性被设置为 false。在 XML 数据下载完毕后，loaded 属性被设置为 true，并调用 onLoad 事件处理函数。直到 XML 数据完全下载后，才开始分析。如果该 XML 对象以前包含任何 XML 树，它们将被放弃。

如果 XMLSocket.connect() 返回 true 值，则表示连接过程的初始阶段是成功的；随后将调用 XMLSocket.onConnect 方法以确定最终连接是成功还是失败。如果 XMLSocket.connect() 返回 false，则无法建立连接。

使用此方法时，请考虑 Flash Player 安全模型。

- 对于 Flash Player 8，如果执行调用的 SWF 文件在只能与本地文件系统的内容交互的沙箱中，则不允许使用 XMLSocket.connect() 方法。
- 对于 Flash Player 7 及更高版本，网站可通过跨域策略文件允许对资源进行跨域访问。

有关更多信息，请参见以下部分：

- 《学习 Flash 中的 ActionScript 2.0》的第 17 章，“了解安全性”
- Flash Player 8 安全性白皮书（位于 [http://www.macromedia.com/go/fp8\\_security](http://www.macromedia.com/go/fp8_security)）
- Flash Player 8 与安全相关的 API 白皮书（位于 [http://www.macromedia.com/go/fp8\\_security\\_apis](http://www.macromedia.com/go/fp8_security_apis)）

可用性：ActionScript 1.0；Flash Player 5

## 参数

**url:String** - 字符串；一个完全限定 DNS 域名，或一个 `aaa.bbb.ccc.ddd` 形式的 IP 地址。也可指定 `null` 以连接到 SWF 文件所在的主机服务器。如果发出此调用的 SWF 文件正在 Web 浏览器中运行，则 `host` 必须与 SWF 文件位于同一个域中；有关详细信息，请参见此方法的主要说明中有关 SWF 文件的域限制的信息。

**port:Number** - 一个编号；用于建立连接的主机上的 TCP 端口号。该端口号必须为 1024 或更高。

## 返回

Boolean - 如果连接成功，则为 `true`；否则为 `false`。

## 示例

下面的示例使用 `XMLSocket.connect()` 来连接 SWF 文件所在的主机，然后使用 `trace` 来显示表明连接是否成功的返回值：

```
var socket:XMLSocket = new XMLSocket()
socket.onConnect = function (success:Boolean) {
    if (success) {
        trace ("Connection succeeded!")
    } else {
        trace ("Connection failed!")
    }
}
if (!socket.connect(null, 2000)) {
    trace ("Connection failed!")
}
```

## 另请参见

[onConnect \(XMLSocket.onConnect 处理函数\)](#)，[function 语句](#)



## onClose (XMLSocket.onClose 处理函数)

**onClose = function() {}**

仅在服务器关闭某个打开的连接时调用。此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含自定义动作的函数。

可用性：ActionScript 1.0 ； Flash Player 5

### 示例

下面的示例在服务器关闭一个打开的连接时执行 **trace** 语句：

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.onClose = function () {
    trace("Connection to server lost.");
}
```

另请参见

[onConnect \(XMLSocket.onConnect 处理函数\)](#)，[function 语句](#)

## onConnect (XMLSocket.onConnect 处理函数)

**onConnect = function(success:Boolean) {}**

在通过 XMLSocket.connect() 启动的连接请求成功或失败后，由 **Flash Player** 调用。如果连接成功，则 success 参数为 true ； 否则 success 参数为 false。

此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含自定义动作的函数。

可用性：ActionScript 1.0 ； Flash Player 5

### 参数

**success:Boolean** - 一个布尔值，指示是否成功建立了套接字连接（true 或 false）。

## 示例

下面的示例说明在一个简单的聊天应用程序中为 `onConnect` 方法指定替换函数的过程。

在使用 `constructor` 方法创建 `XMLSocket` 对象之后，脚本会定义在调用 `onConnect` 事件处理函数时要执行的自定义功能。该函数根据是否成功建立了连接来控制向用户显示哪一屏幕。如果连接成功建立，则向用户显示标记为 `startChat` 的帧上的主聊天屏幕。如果连接不成功，则向用户显示标记为 `connectionFailed` 的帧上的带有疑难解答信息的屏幕。

```
var socket:XMLSocket = new XMLSocket();
socket.onConnect = function (success) {
    if (success) {
        gotoAndPlay("startChat");
    } else {
        gotoAndStop("connectionFailed");
    }
}
```

最后，启动连接。如果 `connect()` 返回 `false`，则 `SWF` 文件将直接被发送给标记为 `connectionFailed` 的帧，并且从不调用 `onConnect`。如果 `connect()` 返回 `true`，`SWF` 文件就会跳转到标记为 `waitForConnection` 的帧，即“请稍候”屏幕。`SWF` 文件会停留在 `waitForConnection` 帧上，直到调用 `onConnect` 处理函数，调用该函数会在将来某个时刻发生，具体取决于网络的滞后时间。

```
if (!socket.connect(null, 2000)) {
    gotoAndStop("connectionFailed");
} else {
    gotoAndStop("waitForConnection");
}
```

另请参见

[connect \(XMLSocket.connect 方法\)](#)，[function 语句](#)

## onData (XMLSocket.onData 处理函数)

**onData = function(src:String) {}**

当某个消息已从服务器下载并以零 (0) 字节结束时调用。您可以覆盖 `XMLSocket.onData` 以截获服务器发送的数据，而不将其分析为 `XML`。如果您传输的是任意格式的数据包，而且希望在数据到达时直接操纵这些数据，而不让 `Flash Player` 将数据分析为 `XML`，则此方法很有用。

默认情况下，`XMLSocket.onData` 方法调用 `XMLSocket.onXML` 方法。如果您用自定义行为覆盖 `XMLSocket.onData`，除非您在 `XMLSocket.onData` 实现过程中调用 `XMLSocket.onXML`，否则不会对其进行调用。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**src:String** - 一个字符串，包含服务器发送的数据。

## 示例

在此示例中，src 参数是一个字符串，其中包含从服务器下载的 XML 文本。零 (0) 字节终止符不包含在该字符串中。

```
XMLSocket.prototype.onData = function (src) {  
    this.onXML(new XML(src));  
}
```

## onXML (XMLSocket.onXML 处理函数)

在包含 XML 文档的指定 XML 对象通过打开的 XMLSocket 连接到达时，由 Flash Player 调用。XMLSocket 连接可用于在客户端与服务器之间传输无限量的 XML 文档。每个文档都以零 (0) 字节结束。当 Flash Player 收到 0 字节时，它分析自从前一个 0 字节以来收到的所有 XML；如果这是收到的第一条消息，则分析建立连接以来收到的所有 XML。每一批经过分析的 XML 都将被视为单个 XML 文档，并传递给 onXML 方法。

此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含您定义的动作的函数。

可用性：ActionScript 1.0；Flash Player 5

## 参数

**src:XML** - 一个 XML 对象，它包含从服务器上接收到的经过分析的 XML 文档。

## 示例

下面的函数在一个简单的聊天应用程序中覆盖 onXML 方法的默认实现。函数 myOnXML 指示该聊天应用程序识别一个 XML 元素 (MESSAGE)，该元素的格式如下所示。

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.
```

假定下面的函数 displayMessage() 是一个用户定义的函数，该函数显示用户收到的消息：

```
var socket:XMLSocket = new XMLSocket();  
socket.onXML = function (doc) {  
    var e = doc.firstChild;  
    if (e != null && e.nodeName == "MESSAGE") {  
        displayMessage(e.attributes.user, e.attributes.text);  
    }  
}
```

另请参见

[function 语句](#)

## send (XMLSocket.send 方法)

```
public send(data:Object) : Void
```

将在 `object` 参数中指定的 XML 对象或数据转换成一个字符串，并将其传输到服务器，后面跟有一个零 (0) 字节。如果 `object` 是一个 XML 对象，则该字符串是此 XML 对象的 XML 文本表示形式。发送操作是异步的；它将立即返回，但数据可能会以后传输。  
`XMLSocket.send()` 方法不返回指示数据是否成功传输的值。

如果 `myXMLSocket` 对象未连接到服务器（使用 `XMLSocket.connect()`），则 `XMLSocket.send()` 操作将失败。

可用性：ActionScript 1.0；Flash Player 5

### 参数

`data:Object` - 一个要传输到服务器的 XML 对象或其它数据。

### 示例

下面的示例说明如何指定用户名和密码，以将 XML 对象 `my_xml` 发送到服务器：

```
var myXMLSocket:XMLSocket = new XMLSocket();
var my_xml:XML = new XML();
var myLogin:XMLNode = my_xml.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
my_xml.appendChild(myLogin);
myXMLSocket.send(my_xml);
```

### 另请参见

[connect \(XMLSocket.connect 方法\)](#)

## XMLSocket 构造函数

```
public XMLSocket()
```

创建一个新的 `XMLSocket` 对象。`XMLSocket` 对象开始时不与任何服务器连接。必须调用 `XMLSocket.connect()` 将该对象连接到服务器。

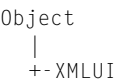
可用性：ActionScript 1.0；Flash Player 5

### 示例

下面的示例创建一个 `XMLSocket` 对象：

```
var socket:XMLSocket = new XMLSocket();
```

# XMLUI



```
public class XMLUI
extends Object
```

XMLUI 对象可以与用作 Flash 创作工具的扩展功能（比如“行为”、“命令”、“效果”和“工具”）的自定义用户界面的 SWF 文件进行通讯。

Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 附带多种扩展功能，包括“行为”、“命令” (JavaScript API)、“效果”和“工具”。有了这些功能，高级用户可以实现创作工具功能的扩展或自动化。XML 到 UI 的引擎可以使用这些扩展功能创建用户可以看到的对话框（如果扩展功能需要或接受参数）。可以通过使用 XML 标签或创建要显示的 SWF 文件来定义对话框。XMLUI 对象提供了一种机制，高级用户可以利用这一机制与以此方式使用的 SWF 文件通讯。

可用性: ActionScript 1.0 ; Flash Player 7

## 属性摘要

继承自 Object 类的属性

```
constructor (Object.constructor 属性), __proto__ (Object.__proto__ 属性),
prototype (Object.prototype 属性), __resolve (Object.__resolve 属性)
```

## 方法摘要

修饰符	签名	说明
static	accept() : Void	使当前的 XMLUI 对话框以“accept”状态退出。
static	cancel() : Void	使当前的 XMLUI 对话框以“cancel”状态退出。
static	get(name:String) : String	检索当前 XMLUI 对话框的指定属性的值。
static	set(name:String, value:String) : Void	修改当前 XMLUI 对话框的指定属性的值。

继承自 Object 类的方法

---

`addProperty` (Object.addProperty 方法), `hasOwnProperty` (Object.hasOwnProperty 方法), `isPropertyEnumerable` (Object.isPropertyEnumerable 方法), `isPrototypeOf` (Object.isPrototypeOf 方法), `registerClass` (Object.registerClass 方法), `toString` (Object.toString 方法), `unwatch` (Object.unwatch 方法), `valueOf` (Object.valueOf 方法), `watch` (Object.watch 方法)

---

## accept (XMLUI.accept 方法)

`public static accept() : Void`

使当前的 XMLUI 对话框以 “accept” 状态退出。相当于用户单击 “确定” 按钮。

可用性: ActionScript 1.0 ; Flash Player 7

## cancel (XMLUI.cancel 方法)

`public static cancel() : Void`

使当前的 XMLUI 对话框以 “cancel” 状态退出。相当于用户单击 “取消” 按钮。

可用性: ActionScript 1.0 ; Flash Player 7

## get (XMLUI.get 方法)

`public static get(name:String) : String`

检索当前 XMLUI 对话框的指定属性的值。

可用性: ActionScript 1.0 ; Flash Player 7

### 参数

**name:String** - 要检索的 XMLUI 属性的名称。

### 返回

String - 以字符串形式返回属性的值。

## set (XMLUI.set 方法)

```
public static set(name:String, value:String) : Void
```

修改当前 XMLUI 对话框的指定属性的值。

可用性: **ActionScript 1.0** ; **Flash Player 7**

### 参数

**name:**String - 要修改的 XMLUI 属性的名称。

**value:**String - 指定的属性将要被设置为的值。





# ActionScript 不推荐使用的內容

随着 ActionScript 的发展，该语言的许多元素已经不推荐使用。本节列出了不推荐使用的项并建议了替代方法（如果可用）。虽然不推荐使用的元素在 Flash Player 8 中仍然有效，但 Macromedia 建议您不要在代码中继续使用这些元素。因为不能保证这些元素在将来仍受支持。

## 不推荐使用的类摘要

修饰符	类名称	说明
	Color	从 Flash Player 8 后不推荐使用。不推荐使用 Color 类，而推荐使用 flash.geom.ColorTransform 类。

## 不推荐使用的函数摘要

修饰符	函数名称	说明
	call(frame:Object)	自 Flash Player 5 后不推荐使用。不推荐使用此动作，而推荐使用 function 语句。
	chr(number:Number)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String.fromCharCode()。
	TextFormat.getTextExtent(text:String, [width:Number])	自 Flash Player 8 后不推荐使用。没有其它替换方法。
	ifFrameLoaded([scene:String], frame:Object)	自 Flash Player 5 后不推荐使用。此函数已不推荐使用。Macromedia 建议您使用 MovieClip._framesloaded 属性。
	int(value:Number)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 Math.round()。

修饰符	函数名称	说明
	length(expression:String, variable:Object)	自 Flash Player 5 后不推荐使用。此函数及所有字符串函数已不推荐使用。Macromedia 建议您使用 String 类的方法和 String.length 属性来执行相同的操作。
	mbchr(number:Number)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String.fromCharCode() 方法。
	mblength(string:String)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String 类的方法和属性。
	mbord(character:String)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String.charCodeAt()。
	mbsubstring(value:String, index:Number, count:Number)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String.substr()。
	ord(character:String)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String 类的方法和属性。
	random(value:Number)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 Math.random()。
	substring(string:String, index:Number, count:Number)	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 String.substr()。
	tellTarget(target:String, statement(s))	自 Flash Player 5 后不推荐使用。Macromedia 建议使用点 (.) 记号和 with 语句。
	toggleHighQuality()	自 Flash Player 5 后不推荐使用。不推荐使用此函数，而推荐使用 _quality。

## 不推荐使用的属性摘要

修饰符	属性名称	说明
	Button._highquality	自 Flash Player 7 后不推荐使用。不推荐使用此属性，而推荐使用 Button._quality。
	MovieClip._highquality	自 Flash Player 7 后不推荐使用。不推荐使用此属性，而推荐使用 MovieClip._quality。
	TextField._highquality	自 Flash Player 7 后不推荐使用。不推荐使用此属性，而推荐使用 TextField._quality。
	_highquality	自 Flash Player 5 后不推荐使用。不推荐使用此属性，而推荐使用 _quality。

修饰符	属性名称	说明
	maxscroll	自 Flash Player 5 后不推荐使用。不推荐使用此属性，而推荐使用 <code>TextField.maxscroll</code> 。
	scroll	自 Flash Player 5 后不推荐使用。不推荐使用此属性，而推荐使用 <code>TextField.scroll</code> 。

# 不推荐使用的运算符摘要

运算符	说明
◇ （不等于）	自 Flash Player 5 后不推荐使用。此运算符已不推荐使用。Macromedia 建议您使用 <code>!=</code> (inequality) 运算符。
add （连接 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。Macromedia 建议在为 Flash Player 5 或更高版本创建内容时使用加运算符 (+)。此运算符在 Flash Player 8 或更高版本中不受支持。
and （逻辑 AND）	自 Flash Player 5 后不推荐使用。Macromedia 建议使用逻辑 AND 运算符 (&&)。
eq （等于 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>==</code> (equality) 运算符。
ge （大于或等于 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>&gt;=</code> （大于或等于）运算符。
gt （大于 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>&gt;</code> （大于）运算符。
le （小于或等于 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。在 Flash 5 中不推荐使用此运算符，而推荐使用 <code>&lt;=</code> （小于或等于）运算符。
lt （小于 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>&lt;</code> （小于）运算符。
ne （不等 [ 字符串 ]）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>!=</code> (inequality) 运算符。
not （逻辑 NOT）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>!</code> (logical NOT) 运算符。
or （逻辑 OR）	自 Flash Player 5 后不推荐使用。不推荐使用此运算符，而推荐使用 <code>  </code> (logical OR) 运算符。



# 索引

type 运算符 177  
不等于运算符 156  
132, 133, 158, 159  
! 逻辑 NOT 运算符 162  
!= 不等于运算符 155  
!== 不全等运算符 174  
" 字符串分隔符运算符 175  
#endinitclip 指令 31  
#include 指令 32  
#initclip 指令 33  
% 模运算符 165  
%= 模赋值运算符 166  
& 按位 AND 运算符 130  
&& 逻辑 AND 运算符 161  
&= 按位 AND 赋值运算符 131  
() 括号运算符 171  
\* 乘法运算符 167  
\*= 乘法赋值运算符 167  
+ 加法运算符 125  
++ 递增运算符 153  
+= 加法赋值运算符 126  
, 逗号运算符 143  
- 减法运算符 175  
-- 递减运算符 146  
-= 减法赋值运算符 176  
-Infinity 常数 36

## 符号

. 点运算符 148  
/ 除法运算符 146  
/\*..\*/ 注释块分隔符运算符 142  
// 注释行分隔符运算符 160  
/= 除法赋值运算符 147  
= 赋值运算符 129  
== 等于运算符 149

=== 全等运算符 172  
> 大于运算符 151  
>= 大于或等于运算符 152  
>> 按位向右移位运算符 137  
>>= 按位向右移位并赋值运算符 138  
>>> 按位无符号向右移位运算符 139  
>>>= 按位无符号向右移位并赋值运算符 140  
?  
条件运算符 145  
^ 按位 XOR 运算符 140  
^= 按位 XOR 赋值运算符 141  
\_\_proto\_\_ 属性 932  
\_\_resolve 属性 935  
\_accProps 属性 109  
\_alpha 属性 324, 772, 1099, 1202  
\_currentframe 属性 795  
\_droptarget 属性 798  
\_focusrect 属性 112, 332, 804  
\_framesloaded 属性 805  
\_global 属性 113  
\_height 属性 333, 821, 1114, 1205  
\_highquality 属性 114, 334, 821, 1114  
\_level 属性 114  
\_listeners 属性 252, 640  
\_lockroot 属性 837  
\_name 属性 335, 841, 1121, 1206  
\_parent 属性 115, 341, 854, 1126, 1206  
\_quality 属性 116, 342, 856, 1128  
\_root 属性 117  
\_rotation 属性 342, 859, 1133, 1206  
\_soundbuftime 属性 118, 344, 865, 1140  
\_target 属性 346, 871, 1145  
\_totalframes 属性 872  
\_url 属性 348, 876, 1150  
\_visible 属性 349, 877, 1151, 1207  
\_width 属性 350, 878, 1151, 1208  
\_x 属性 350, 879, 1153, 1208

\_xmouse 属性 351, 880, 1154, 1209  
\_xscale 属性 351, 881, 1155, 1209  
\_y 属性 352, 882, 1156, 1209  
\_ymouse 属性 352, 883, 1156, 1210  
\_yscale 属性 353, 884, 1157, 1210  
{} 对象初始值设定项运算符 169  
| 按位 OR 运算符 135  
|= 按位 OR 赋值运算符 136  
|| 逻辑 OR 运算符 164  
~ 按位 NOT 运算符 134

## 数字

a 属性 715

## 英文

abs() 方法 696  
accept() 方法 1274  
Accessibility  
    isActive() 方法 224  
    updateProperties() 方法 225  
Accessibility 类 223  
acos() 方法 697  
activityLevel 属性 356, 734  
add() 方法 945  
addCallback() 方法 515  
addDelayedInstance() 方法 683  
addListener() 方法 249, 523, 548, 612, 626, 754,  
    887, 1004, 1052, 1098  
addPage() 方法 954  
addProperty() 方法 926  
addRequestHeader() 方法 649, 1214  
addXMLPath() 方法 684  
align 属性 1053, 1160  
allowDomain 事件 666  
allowDomain() 方法 991  
allowInsecureDomain 事件 669  
allowInsecureDomain() 方法 996  
alpha 属性 428, 473, 496, 560  
alphaMultiplier 属性 400  
ALPHANUMERIC\_FULL 属性 613  
ALPHANUMERIC\_HALF 属性 613  
alphaOffset 属性 401  
alphas 属性 573, 592  
and 逻辑 AND 运算符 162  
angle 属性 257, 497, 575, 593  
antiAliasType 属性 1100  
appendChild() 方法 1240

apply() 方法 555  
applyFilter() 方法 277  
arguments  
    callee 属性 227  
    caller 属性 227  
    length 属性 227  
arguments 类 226  
Array  
    Array() 构造函数 230  
    CASEINSENSITIVE 属性 231  
    concat() 方法 232  
    DESCENDING 属性 233  
    join() 方法 233  
    length 属性 234  
    NUMERIC 属性 235  
    pop() 方法 235  
    push() 方法 236  
    RETURNINDEXEDARRAY 属性 236  
    reverse() 方法 237  
    shift() 方法 237  
    slice() 方法 238  
    sort() 方法 239  
    sortOn() 方法 241  
    splice() 方法 245  
    toString() 方法 246  
    UNIQUESORT 属性 246  
    unshift() 方法 247  
Array 函数 45  
Array 类 227  
Array() 构造函数 230  
AsBroadcaster  
    \_listeners 属性 252  
    addListener() 方法 249  
    broadcastMessage() 方法 250  
    initialize() 方法 251  
    removeListener() 方法 253  
AsBroadcaster 类 247  
asfunction 协议 46  
asin() 方法 697  
atan() 方法 698  
atan2() 方法 699  
attachAudio() 方法 773  
attachBitmap() 方法 774  
attachMovie() 方法 775  
attachSound() 方法 1029  
attachVideo() 方法 1202  
attributes 属性 1241  
autoReplace 属性 684  
autoSize 属性 1101  
available 属性 517

- avHardwareDisable 属性 378
- b 属性 716
- background 属性 1103
- backgroundColor 属性 1103
- BACKSPACE 属性 628
- bandwidth 属性 357
- beginBitmapFill() 方法 776
- beginFill() 方法 778
- beginGradientFill() 方法 779
- BevelFilter
  - angle 属性 257
  - BevelFilter() 构造函数 258
  - blurX 属性 260
  - blurY 属性 261
  - clone() 方法 262
  - distance 属性 264
  - highlightAlpha 属性 265
  - highlightColor 属性 266
  - knockout 属性 267
  - quality 属性 268
  - shadowAlpha 属性 269
  - shadowColor 属性 270
  - strength 属性 271
  - type 属性 272
- BevelFilter 类 254
- BevelFilter() 构造函数 258
- bias 属性 429
- BitmapData
  - applyFilter() 方法 277
  - BitmapData() 构造函数 279
  - clone() 方法 280
  - colorTransform() 方法 282
  - copyChannel() 方法 283
  - copyPixels() 方法 284
  - dispose() 方法 286
  - draw() 方法 286
  - fillRect() 方法 288
  - floodFill() 方法 289
  - generateFilterRect() 方法 290
  - getColorBoundsRect() 方法 291
  - getPixel() 方法 292
  - getPixel32() 方法 293
  - height 属性 294
  - hitTest() 方法 294
  - loadBitmap() 方法 296
  - merge() 方法 296
  - noise() 方法 297
  - paletteMap() 方法 299
  - perlinNoise() 方法 300
  - pixelDissolve() 方法 302
  - rectangle 属性 304
  - scroll() 方法 304
  - setPixel() 方法 305
  - setPixel32() 方法 306
  - threshold() 方法 307
  - transparent 属性 309
  - width 属性 309
- BitmapData 类 273
- BitmapData() 构造函数 279
- BitmapFilter
  - clone() 方法 311
- BitmapFilter 类 310
- blendMode 属性 324, 785
- blockIndent 属性 1161
- blueMultiplier 属性 402
- blueOffset 属性 403
- BlurFilter
  - BlurFilter() 构造函数 313
  - blurX 属性 314
  - blurY 属性 315
  - clone() 方法 316
  - quality 属性 317
- BlurFilter 类 311
- BlurFilter() 构造函数 313
- blurX 属性 260, 314, 498, 561, 576, 594
- blurY 属性 261, 315, 499, 562, 577, 595
- bold 属性 1161
- Boolean
  - Boolean() 构造函数 319
  - toString() 方法 320
  - valueOf() 方法 320
- Boolean 函数 47
- Boolean 类 318
- Boolean() 构造函数 319
- border 属性 1104
- borderColor 属性 1104
- bottom 属性 965
- bottomRight 属性 966
- bottomScroll 属性 1105
- break 语句 181
- broadcastMessage() 方法 250
- browse() 方法 524, 548
- bufferLength 属性 906
- bufferTime 属性 907
- builtInItems 属性 413
- bullet 属性 1162
- Button
  - \_alpha 属性 324
  - \_focusrect 属性 332
  - \_height 属性 333

- \_highquality 属性 334
  - \_name 属性 335
  - \_parent 属性 341
  - \_quality 属性 342
  - \_rotation 属性 342
  - \_soundbuftime 属性 344
  - \_target 属性 346
  - \_url 属性 348
  - \_visible 属性 349
  - \_width 属性 350
  - \_x 属性 350
  - \_xmouse 属性 351
  - \_xscale 属性 351
  - \_y 属性 352
  - \_ymouse 属性 352
  - \_yscale 属性 353
- blendMode 属性 324
- cacheAsBitmap 属性 328
- enabled 属性 329
- filters 属性 330
- getDepth() 方法 332
- menu 属性 334
- scale9Grid 属性 343
- tabEnabled 属性 344
- tabIndex 属性 345
- trackAsMenu 属性 347
- useHandCursor 属性 348
- Button 类 321
  - onDragOut 事件 335
  - onDragOver 事件 336
  - onKeyDown 事件 336
  - onKeyUp 事件 337
  - onKillFocus 事件 338
  - onPress 事件 339
  - onRelease 事件 339
  - onReleaseOutside 事件 339
  - onRollOut 事件 340
  - onRollOver 事件 340
  - onSetFocus 事件 340
- bytesLoaded 属性 908
- bytesTotal 属性 909
- c 属性 716
- cacheAsBitmap 属性 328, 790
- call 函数 49
- call() 方法 517, 557
- callee 属性 227
- caller 属性 227
- Camera
  - activityLevel 属性 356
  - bandwidth 属性 357
  - currentFps 属性 358
  - fps 属性 359
  - get() 方法 360
  - height 属性 362
  - index 属性 362
  - motionLevel 属性 363
  - motionTimeOut 属性 365
  - muted 属性 366
  - name 属性 367
  - names 属性 367
  - quality 属性 370
  - setMode() 方法 371
  - setMotionLevel() 方法 372
  - setQuality() 方法 373
  - width 属性 375
- Camera 类 354
  - onActivity 事件 368
  - onStatus 事件 369
- cancel() 方法 526, 1274
- capabilities
  - avHardwareDisable 属性 378
  - hasAccessibility 属性 379
  - hasAudio 属性 379
  - hasAudioEncoder 属性 379
  - hasEmbeddedVideo 属性 380
  - hasIME 属性 380
  - hasMP3 属性 380
  - hasPrinting 属性 381
  - hasScreenBroadcast 属性 381
  - hasScreenPlayback 属性 381
  - hasStreamingAudio 属性 382
  - hasStreamingVideo 属性 382
  - hasVideoEncoder 属性 382
  - isDebugger 属性 383
  - language 属性 383
  - localFileReadDisable 属性 384
  - manufacturer 属性 385
  - os 属性 385
  - pixelAspectRatio 属性 385
  - playerType 属性 386
  - screenColor 属性 386
  - screenDPI 属性 386
  - screenResolutionX 属性 387
  - screenResolutionY 属性 387
  - serverString 属性 387
  - version 属性 388
- capabilities 类 375
- CAPSLOCK 属性 628
- caption 属性 420
- case 语句 182



CASEINSENSITIVE 属性 231

ceil() 方法 699

charAt() 方法 1061

charCodeAt() 方法 1062

checkXMLStatus() 方法 685

childNodes 属性 1241

CHINESE 属性 614

chr 函数 49

clamp 属性 429

class

- 由 Button 发送 335, 336, 337, 338, 339, 340
- 由 Camera 发送 368, 369
- 由 ContextMenu 发送 417
- 由 ContextMenuItem 发送 423
- 由 FileReference 发送 532, 533, 535, 536, 537, 539
- 由 FileReferenceList 发送 552
- 由 IME 调度 619
- 由 Key 发送 640, 641
- 由 LoadVars 发送 656, 657, 659
- 由 LocalConnection 发送 666, 669, 678
- 由 Microphone 发送 741, 742
- 由 Mouse 发送 757, 758, 759, 760
- 由 MovieClip 发送 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853
- 由 MovieClipLoader 发送 892, 894, 895, 897, 898
- 由 NetStream 发送 912, 913
- 由 Selection 发送 1009
- 由 SharedObject 发送 1025
- 由 Sound 发送 1042, 1043, 1044
- 由 Stage 发送 1054
- 由 StyleSheet 发送 1082
- 由 System 发送 1089
- 由 TextField 发送 1122, 1123, 1124, 1125
- 由 XML 发送 1226, 1227, 1229
- 由 XMLSocket 发送 1269, 1270, 1271

class 语句 183

clear() 方法 791, 1016, 1077, 1203

clearInterval 函数 50

clone() 方法 262, 280, 311, 316, 396, 430, 475, 500, 563, 578, 596, 717, 945, 967

cloneNode() 方法 1243

close() 方法 671, 910, 1266

Color

- Color() 构造函数 389
- getRGB() 方法 390
- getTransform() 方法 390
- setRGB() 方法 391
- setTransform() 方法 392

Color 类 388

color 属性 432, 478, 501, 564, 1162

Color() 构造函数 389

ColorMatrixFilter

clone() 方法 396

ColorMatrixFilter() 构造函数 397

matrix 属性 397

ColorMatrixFilter 类 393

ColorMatrixFilter() 构造函数 397

colors 属性 579, 598

ColorTransform

alphaMultiplier 属性 400

alphaOffset 属性 401

blueMultiplier 属性 402

blueOffset 属性 403

ColorTransform() 构造函数 404

concat() 方法 405

greenMultiplier 属性 406

greenOffset 属性 407

redMultiplier 属性 408

redOffset 属性 409

rgb 属性 410

toString() 方法 411

ColorTransform 类 398

colorTransform 属性 1193

colorTransform() 方法 282

ColorTransform() 构造函数 404

componentX 属性 479

componentY 属性 481

concat() 方法 232, 405, 717, 1062

concatenatedColorTransform 属性 1194

concatenatedMatrix 属性 1195

condenseWhite 属性 1106

connect() 方法 672, 902, 1267

constructor 属性 929

contains() 方法 969

containsPoint() 方法 970

containsRectangle() 方法 971

contentType 属性 650, 1215

ContextMenu

builtInItems 属性 413

ContextMenu() 构造函数 414

copy() 方法 415

customItems 属性 416

hideBuiltInItems() 方法 417

ContextMenu 类 411

onSelect 事件 417

ContextMenu() 构造函数 414

ContextMenuItem

caption 属性 420

- ContextMenuItem() 构造函数 420
- copy() 方法 421
- enabled 属性 422
- separatorBefore 属性 424
- visible 属性 425
- ContextMenuItem 类 418
  - onSelect 事件 423
- ContextMenuitem() 构造函数 420
- continue 语句 185
- CONTROL 属性 629
- ConvolutionFilter
  - alpha 属性 428
  - bias 属性 429
  - clamp 属性 429
  - clone() 方法 430
  - color 属性 432
- ConvolutionFilter() 构造函数 432
- divisor 属性 434
- matrix 属性 434
- matrixX 属性 435
- matrixY 属性 436
- preserveAlpha 属性 436
- ConvolutionFilter 类 426
- ConvolutionFilter() 构造函数 432
- copy() 方法 415, 421
- copyChannel() 方法 283
- copyPixels() 方法 284
- cos() 方法 700
- createBox() 方法 719
- createElement() 方法 1215
- createEmptyMovieClip() 方法 792
- createGradientBox() 方法 720
- createTextField() 方法 793
- createTextNode() 方法 1216
- creationDate 属性 526
- creator 属性 527
- currentFps 属性 358, 911
- curveTo() 方法 796
- CustomActions
  - get() 方法 438
  - install() 方法 439
  - list() 方法 440
  - uninstall() 方法 441
- CustomActions 类 437
- customItems 属性 416
- d 属性 721
- data 属性 1017
- Date
  - Date() 构造函数 446
  - getDate() 方法 447

- getDay() 方法 448
- getFullYear() 方法 448
- getHours() 方法 449
- getMilliseconds() 方法 450
- getMinutes() 方法 450
- getMonth() 方法 451
- getSeconds() 方法 451
- getTime() 方法 452
- getTimezoneOffset() 方法 452
- getUTCDate() 方法 453
- getUTCDay() 方法 453
- getUTCFullYear() 方法 454
- getUTCHours() 方法 454
- getUTCMilliseconds() 方法 455
- getUTCMinutes() 方法 455
- getUTCMonth() 方法 456
- getUTCSeconds() 方法 456
- getUTCYear() 方法 457
- getYear() 方法 457
- setDate() 方法 458
- setFullYear() 方法 458
- setHours() 方法 459
- setMilliseconds() 方法 460
- setMinutes() 方法 460
- setMonth() 方法 461
- setSeconds() 方法 462
- setTime() 方法 462
- setUTCDate() 方法 463
- setUTCFullYear() 方法 464
- setUTCHours() 方法 465
- setUTCMilliseconds() 方法 466
- setUTCMinutes() 方法 466
- setUTCMonth() 方法 467
- setUTCSeconds() 方法 468
- setYear() 方法 468
- toString() 方法 469
- UTC() 方法 470
- valueOf() 方法 470
- Date 类 442
- Date() 构造函数 446
- deblocking 属性 1204
- decode() 方法 650
- default 语句 186
- delete 语句 187
- DELETEKEY 属性 629
- deltaTransformPoint() 方法 721
- DESCENDING 属性 233
- DisplacementMapFilter
  - alpha 属性 473
  - clone() 方法 475

- color 属性 478
- componentX 属性 479
- componentY 属性 481
- DisplacementMapFilter() 构造函数 483
- mapBitmap 属性 485
- mapPoint 属性 487
- mode 属性 488
- scaleX 属性 490
- scaleY 属性 492
- DisplacementMapFilter 类 471
- DisplacementMapFilter() 构造函数 483
- dispose() 方法 286
- distance 属性 264, 502, 580, 599
- distance() 方法 946
- divisor 属性 434
- do..while 语句 189
- doConversion() 方法 614
- docTypeDecl 属性 1218
- domain() 方法 675
- DOWN 属性 630
- download() 方法 527
- draw() 方法 286
- DropShadowFilter
  - alpha 属性 496
  - angle 属性 497
  - blurX 属性 498
  - blurY 属性 499
  - clone() 方法 500
  - color 属性 501
  - distance 属性 502
  - DropShadowFilter() 构造函数 503
  - hideObject 属性 505
  - inner 属性 506
  - knockout 属性 507
  - quality 属性 508
  - strength 属性 509
- DropShadowFilter 类 493
- DropShadowFilter() 构造函数 503
- duplicateMovieClip 函数 50
- duplicateMovieClip() 方法 799
- duration 属性 1029
- dynamic 语句 190
- E 属性 701
- else if 语句 192
- else 语句 191
- embedFonts 属性 1107
- enabled 属性 329, 422, 801
- END 属性 631
- endFill() 方法 801
- ENTER 属性 631
- eq 等于 (字符串) 运算符 150
- equals() 方法 946, 972
- Error
  - Error() 构造函数 511
  - message 属性 511
  - name 属性 512
  - toString() 方法 513
- Error 类 510
- Error() 构造函数 511
- escape 函数 51
- ESCAPE 属性 632
- eval 函数 52
- exactSettings 属性 1088
- exp() 方法 701
- extends 语句 193
- ExternalInterface
  - addCallback() 方法 515
  - available 属性 517
  - call() 方法 517
- ExternalInterface 类 514
- false 常数 35
- fileList 属性 550
- FileReference
  - addListener() 方法 523
  - browse() 方法 524
  - cancel() 方法 526
  - creationDate 属性 526
  - creator 属性 527
  - download() 方法 527
  - FileReference() 构造函数 530
  - modificationDate 属性 530
  - name 属性 531
  - removeListener() 方法 540
  - size 属性 541
  - type 属性 541
  - upload() 方法 542
- FileReference 类 519
  - onCancel 事件 532
  - onComplete 事件 533
  - onHTTPError 事件 533
  - onIOError 事件 535
  - onOpen 事件 536
  - onProgress 事件 536
  - onSecurityError 事件 537
  - onSelect 事件 539
- FileReference() 构造函数 530
- FileReferenceList
  - addListener() 方法 548
  - browse() 方法 548
  - fileList 属性 550

FileReferenceList() 构造函数 551  
removeListener() 方法 553  
FileReferenceList 类 545  
onCancel 事件 552  
onSelect 事件 552  
FileReferenceList() 构造函数 551  
fillRect() 方法 288  
filters 属性 330, 802, 1108  
findText() 方法 1181  
firstChild 属性 1244  
floodFill() 方法 289  
floor() 方法 702  
flush() 方法 1019  
focusEnabled 属性 803  
font 属性 1163  
for 语句 195  
for..in 语句 196  
fps 属性 359  
fromCharCode() 方法 1063  
fscommand 函数 53  
Function  
    apply() 方法 555  
    call() 方法 557  
Function 类 554  
function 语句 197  
gain 属性 735  
ge 大于或等于（字符串）运算符 152  
generateFilterRect() 方法 290  
get 语句 198  
get() 方法 360, 438, 736, 1274  
getAscii() 方法 633  
getBeginIndex() 方法 1005  
getBounds() 方法 806  
getBytesLoaded() 方法 651, 807, 1031, 1218  
getBytesTotal() 方法 652, 808, 1033, 1219  
getCaretIndex() 方法 1006  
getCode() 方法 634  
getColorBoundsRect() 方法 291  
getConversionMode() 方法 615  
getCount() 方法 1182  
getDate() 方法 447  
getDay() 方法 448  
getDefaultLang() 方法 686  
getDepth() 方法 332, 809, 1109  
getEnabled() 方法 616  
getEndIndex() 方法 1007  
getFocus() 方法 1008  
getFontList() 方法 1110  
getFullYear() 方法 448  
getHours() 方法 449  
getInstanceAtDepth() 方法 810  
getLocal() 方法 1020  
getMilliseconds() 方法 450  
getMinutes() 方法 450  
getMonth() 方法 451  
getNamespaceForPrefix() 方法 1246  
getNewTextFormat() 方法 1110  
getNextHighestDepth() 方法 811  
getPan() 方法 1033  
getPixel() 方法 292  
getPixel32() 方法 293  
getPrefixForNamespace() 方法 1247  
getProgress() 方法 888  
getProperty 函数 57  
getRect() 方法 812  
getRGB() 方法 390  
getSeconds() 方法 451  
getSelected() 方法 1182  
getSelectedText() 方法 1183  
getSize() 方法 1024  
getStyle() 方法 1078  
getStyleNames() 方法 1080  
getSWFVersion() 方法 813  
getText() 方法 1184  
getTextExtent() 方法 1163  
getTextFormat() 方法 1111  
getTextRunInfo() 方法 1186  
getTextSnapshot() 方法 814  
getTime() 方法 452  
getTimer 函数 57  
getTimezoneOffset() 方法 452  
getTransform() 方法 390, 1035  
getURL 函数 58  
getURL() 方法 815  
getUTCDate() 方法 453  
getUTCDay() 方法 453  
getUTCFullYear() 方法 454  
getUTCHours() 方法 454  
getUTCMilliseconds() 方法 455  
getUTCMinutes() 方法 455  
getUTCMonth() 方法 456  
getUTCSeconds() 方法 456  
getUTCYear() 方法 457  
getVersion 函数 59  
getVolume() 方法 1037  
getYear() 方法 457  
globalToLocal() 方法 817  
GlowFilter  
    alpha 属性 560  
    blurX 属性 561

- blurY 属性 562
- clone() 方法 563
- color 属性 564
- GlowFilter() 构造函数 565
- inner 属性 567
- knockout 属性 568
- quality 属性 569
- strength 属性 570
- GlowFilter 类 558
- GlowFilter() 构造函数 565
- gotoAndPlay 函数 60
- gotoAndPlay() 方法 819
- gotoAndStop 函数 61
- gotoAndStop() 方法 820
- GradientBevelFilter
  - alphas 属性 573
  - angle 属性 575
  - blurX 属性 576
  - blurY 属性 577
  - clone() 方法 578
  - colors 属性 579
  - distance 属性 580
  - GradientBevelFilter() 构造函数 581
  - knockout 属性 583
  - quality 属性 584
  - ratios 属性 585
  - strength 属性 587
  - type 属性 588
- GradientBevelFilter 类 571
- GradientBevelFilter() 构造函数 581
- GradientGlowFilter
  - alphas 属性 592
  - angle 属性 593
  - blurX 属性 594
  - blurY 属性 595
  - clone() 方法 596
  - colors 属性 598
  - distance 属性 599
  - GradientGlowFilter() 构造函数 600
  - knockout 属性 602
  - quality 属性 603
  - ratios 属性 604
  - strength 属性 606
  - type 属性 607
- GradientGlowFilter 类 589
- GradientGlowFilter() 构造函数 600
- greenMultiplier 属性 406
- greenOffset 属性 407
- gridFitType 属性 1112
- gt 大于 (字符串) 运算符 151
- hasAccessibility 属性 379
- hasAudio 属性 379
- hasAudioEncoder 属性 379
- hasChildNodes() 方法 1248
- hasEmbeddedVideo 属性 380
- hasIME 属性 380
- hasMP3 属性 380
- hasOwnProperty() 方法 930
- hasPrinting 属性 381
- hasScreenBroadcast 属性 381
- hasScreenPlayback 属性 381
- hasStreamingAudio 属性 382
- hasStreamingVideo 属性 382
- hasVideoEncoder 属性 382
- height 属性 294, 362, 973, 1054, 1205
- hide() 方法 756
- hideBuiltInItems() 方法 417
- hideObject 属性 505
- highlightAlpha 属性 265
- highlightColor 属性 266
- hitArea 属性 822
- hitTest() 方法 294, 823
- hitTestTextNearPos() 方法 1188
- HOME 属性 635
- hscroll 属性 1114
- html 属性 1115
- htmlText 属性 1116
- id3 属性 1038
- identity() 方法 723
- idMap 属性 1220
- if 语句 200
- iffFrameLoaded 函数 62
- ignoreWhite 属性 1222
- IME
  - addListener() 方法 612
  - ALPHANUMERIC\_FULL 属性 613
  - ALPHANUMERIC\_HALF 属性 613
  - CHINESE 属性 614
  - doConversion() 方法 614
  - getConversionMode() 方法 615
  - getEnabled() 方法 616
  - JAPANESE\_HIRAGANA 属性 616
  - JAPANESE\_KATAKANA\_FULL 属性 617
  - JAPANESE\_KATAKANA\_HALF 属性 618
  - KOREAN 属性 618
  - removeListener() 方法 620
  - setCompositionString() 方法 621
  - setConversionMode() 方法 622
  - setEnabled() 方法 623
  - UNKNOWN 属性 624

- IME 类 608
  - onIMEComposition 事件 619
- implements 语句 201
- import 语句 201
- indent 属性 1166
- index 属性 362, 738
- indexOf() 方法 1063
- Infinity 常数 36
- inflate() 方法 974
- inflatePoint() 方法 975
- initialize() 方法 251, 687
- inner 属性 506, 567
- INSERT 属性 636
- insertBefore() 方法 1249
- install() 方法 439
- instanceof 运算符 157
- int 函数 62
- interface 语句 202
- interpolate() 方法 947
- intersection() 方法 976
- intersects() 方法 977
- intrinsic 语句 204
- invert() 方法 724
- isAccessible() 方法 636
- isActive() 方法 224
- isDebugger 属性 383
- isDown() 方法 637
- isEmpty() 方法 978
- isFinite 函数 63
- isNaN 函数 63
- isPropertyEnumerable() 方法 930
- isPrototypeOf() 方法 931
- isToggled() 方法 637
- italic 属性 1166
- JAPANESE\_HIRAGANA 属性 616
- JAPANESE\_KATAKANA\_FULL 属性 617
- JAPANESE\_KATAKANA\_HALF 属性 618
- join() 方法 233
- kerning 属性 1167
- Key
  - \_listeners 属性 640
  - addListener() 方法 626
  - BACKSPACE 属性 628
  - CAPSLOCK 属性 628
  - CONTROL 属性 629
  - DELETEKEY 属性 629
  - DOWN 属性 630
  - END 属性 631
  - ENTER 属性 631
  - ESCAPE 属性 632
  - getAscii() 方法 633
  - getCode() 方法 634
  - HOME 属性 635
  - INSERT 属性 636
  - isAccessible() 方法 636
  - isDown() 方法 637
  - isToggled() 方法 637
  - LEFT 属性 639
  - PGDN 属性 641
  - PGUP 属性 642
  - removeListener() 方法 642
  - RIGHT 属性 643
  - SHIFT 属性 644
  - SPACE 属性 644
  - TAB 属性 645
  - UP 属性 646
- Key 类 624
  - onKeyDown 事件 640
  - onKeyUp 事件 641
- knockout 属性 267, 507, 568, 583, 602
- KOREAN 属性 618
- language 属性 383
- languageCodeArray 属性 687
- lastChild 属性 1249
- lastIndexOf() 方法 1064
- le 小于或等于 (字符串) 运算符 160
- leading 属性 1168
- LEFT 属性 639
- left 属性 979
- leftMargin 属性 1168
- length 函数 64
- length 属性 227, 234, 948, 1065, 1117
- letterSpacing 属性 1169
- lineGradientStyle() 方法 824
- lineStyle() 方法 828
- lineTo() 方法 830
- list() 方法 440
- LN10 属性 702
- LN2 属性 703
- load () 方法 653, 1081, 1224
- loadBitmap() 方法 296
- loadClip() 方法 889
- loaded 属性 655, 1225
- loadLanguageXML() 方法 688
- loadMovie 函数 65
- loadMovie() 方法 831
- loadMovieNum 函数 67
- loadPolicyFile() 方法 999
- loadSound() 方法 1041
- loadString() 方法 689

loadStringEx() 方法 690  
loadVariables 函数 69  
loadVariables() 方法 834  
loadVariablesNum 函数 71  
LoadVars  
    addRequestHeader() 方法 649  
    contentType 属性 650  
    decode() 方法 650  
    getBytesLoaded() 方法 651  
    getBytesTotal() 方法 652  
    load() 方法 653  
    loaded 属性 655  
    LoadVars() 构造函数 656  
    send() 方法 660  
    sendAndLoad() 方法 662  
    toString() 方法 664  
LoadVars 类 647  
    onData 事件 656  
    onHTTPStatus 事件 657  
    onLoad 事件 659  
LoadVars() 构造函数 656  
LocalConnection  
    close() 方法 671  
    connect() 方法 672  
    domain() 方法 675  
    LocalConnection() 构造函数 677  
    send() 方法 679  
LocalConnection 类 664  
    allowDomain 事件 666  
    allowInsecureDomain 事件 669  
    onStatus 事件 678  
LocalConnection() 构造函数 677  
Locale 类 681  
localFileReadDisable 属性 384  
localName 属性 1251  
localToGlobal() 方法 835  
log() 方法 703  
LOG10E 属性 703  
LOG2E 属性 704  
lt 小于 (字符串) 运算符 158  
manufacturer 属性 385  
mapBitmap 属性 485  
mapPoint 属性 487  
Math  
    abs() 方法 696  
    acos() 方法 697  
    asin() 方法 697  
    atan() 方法 698  
    atan2() 方法 699  
    ceil() 方法 699  
    cos() 方法 700  
    E 属性 701  
    exp() 方法 701  
    floor() 方法 702  
    LN10 属性 702  
    LN2 属性 703  
    log() 方法 703  
    LOG10E 属性 703  
    LOG2E 属性 704  
    max() 方法 704  
    min() 方法 705  
    PI 属性 705  
    pow() 方法 706  
    random() 方法 707  
    round() 方法 707  
    sin() 方法 708  
    sqrt() 方法 709  
    SQRT1\_2 属性 710  
    SQRT2 属性 710  
    tan() 方法 710  
Math 类 694  
Matrix  
    clone() 方法 717  
    concat() 方法 717  
    createBox() 方法 719  
    createGradientBox() 方法 720  
    deltaTransformPoint() 方法 721  
    identity() 方法 723  
    invert() 方法 724  
    rotate() 方法 726  
    scale() 方法 728  
    toString() 方法 729  
    transformPoint() 方法 730  
    translate() 方法 731  
Matrix 类 711  
matrix 属性 397, 434, 1196  
Matrix() 构造函数 725  
matrixX 属性 435  
matrixY 属性 436  
max() 方法 704  
MAX\_VALUE 属性 921  
maxChars 属性 1117  
maxhscroll 属性 1118  
maxLevel 属性 1176  
maxscroll 属性 115, 1118  
mbchr 函数 72  
mblength 函数 73  
mbord 函数 73  
mbsubstring 函数 74  
menu 属性 334, 840, 1119

merge() 方法 296  
message 属性 511  
Microphone  
    activityLevel 属性 734  
    gain 属性 735  
    get() 方法 736  
    index 属性 738  
    muted 属性 739  
    name 属性 739  
    names 属性 740  
    rate 属性 743  
    setGain() 方法 744  
    setRate() 方法 745  
    setSilenceLevel() 方法 746  
    setUseEchoSuppression() 方法 748  
    silenceLevel 属性 749  
    silenceTimeOut 属性 750  
    useEchoSuppression 属性 752  
Microphone 类 732  
    onActivity 事件 741  
    onStatus 事件 742  
min() 方法 705  
MIN\_VALUE 属性 922  
MMExecute 函数 74  
mode 属性 488  
modificationDate 属性 530  
motionLevel 属性 363  
motionTimeOut 属性 365  
Mouse  
    addListener() 方法 754  
    hide() 方法 756  
    removeListener() 方法 762  
    show() 方法 763  
Mouse 类 753  
    onMouseDown 事件 757  
    onMouseMove 事件 758  
    onMouseUp 事件 759  
    onMouseWheel 事件 760  
mouseWheelEnabled 属性 1120  
moveTo() 方法 841  
MovieClip  
    \_alpha 属性 772  
    \_currentframe 属性 795  
    \_droptarget 属性 798  
    \_focusrect 属性 804  
    \_framesloaded 属性 805  
    \_height 属性 821  
    \_highquality 属性 821  
    \_lockroot 属性 837  
    \_name 属性 841

    \_parent 属性 854  
    \_quality 属性 856  
    \_rotation 属性 859  
    \_soundbuftime 属性 865  
    \_target 属性 871  
    \_totalframes 属性 872  
    \_url 属性 876  
    \_visible 属性 877  
    \_width 属性 878  
    \_x 属性 879  
    \_xmouse 属性 880  
    \_xscale 属性 881  
    \_y 属性 882  
    \_ymouse 属性 883  
    \_yscale 属性 884  
attachAudio() 方法 773  
attachBitmap() 方法 774  
attachMovie() 方法 775  
beginBitmapFill() 方法 776  
beginFill() 方法 778  
beginGradientFill() 方法 779  
blendMode 属性 785  
cacheAsBitmap 属性 790  
clear() 方法 791  
createEmptyMovieClip() 方法 792  
createTextField() 方法 793  
curveTo() 方法 796  
duplicateMovieClip() 方法 799  
enabled 属性 801  
endFill() 方法 801  
filters 属性 802  
focusEnabled 属性 803  
getBounds() 方法 806  
getBytesLoaded() 方法 807  
getBytesTotal() 方法 808  
getDepth() 方法 809  
getInstanceAtDepth() 方法 810  
getNextHighestDepth() 方法 811  
getRect() 方法 812  
getSWFVersion() 方法 813  
getTextSnapshot() 方法 814  
getURL() 方法 815  
globalToLocal() 方法 817  
gotoAndPlay() 方法 819  
gotoAndStop() 方法 820  
hitArea 属性 822  
hitTest() 方法 823  
lineGradientStyle() 方法 824  
lineStyle() 方法 828  
lineTo() 方法 830



- loadMovie() 方法 831
- loadVariables() 方法 834
- localToGlobal() 方法 835
- menu 属性 840
- moveTo() 方法 841
- nextFrame() 方法 842
- opaqueBackground 属性 853
- play() 方法 855
- prevFrame() 方法 855
- removeMovieClip() 方法 858
- scale9Grid 属性 860
- scrollRect 属性 863
- setMask() 方法 864
- startDrag() 方法 865
- stop() 方法 866
- stopDrag() 方法 867
- swapDepths() 方法 868
- tabChildren 属性 869
- tabEnabled 属性 870
- tabIndex 属性 870
- trackAsMenu 属性 872
- transform 属性 873
- unloadMovie() 方法 875
- useHandCursor 属性 877
- MovieClip 类 764
  - onData 事件 843
  - onDragOut 事件 844
  - onDragOver 事件 844
  - onEnterFrame 事件 845
  - onKeyDown 事件 845
  - onKeyUp 事件 846
  - onKillFocus 事件 847
  - onLoad 事件 848
  - onMouseDown 事件 849
  - onMouseMove 事件 849
  - onMouseUp 事件 849
  - onPress 事件 850
  - onRelease 事件 850
  - onReleaseOutside 事件 851
  - onRollOut 事件 851
  - onRollOver 事件 852
  - onSetFocus 事件 852
  - onUnload 事件 853
- MovieClipLoader
  - addListener() 方法 887
  - getProgress() 方法 888
  - loadClip() 方法 889
  - MovieClipLoader() 构造函数 891
  - removeListener() 方法 899
  - unloadClip() 方法 900
- MovieClipLoader 类 885
  - onLoadComplete 事件 892
  - onLoadError 事件 894
  - onLoadInit 事件 895
  - onLoadProgress 事件 897
  - onLoadStart 事件 898
- MovieClipLoader() 构造函数 891
- multiline 属性 1121
- muted 属性 366, 739
- name 属性 367, 512, 531, 739
- names 属性 367, 740
- namespaceURI 属性 1252
- NaN 常数 36
- NaN 属性 922
- ne 不等于 (字符串) 运算符 169
- NEGATIVE\_INFINITY 属性 922
- NetConnection
  - connect() 方法 902
  - NetConnection() 构造函数 903
- NetConnection 类 901
- NetConnection() 构造函数 903
- NetStream
  - bufferLength 属性 906
  - bufferTime 属性 907
  - bytesLoaded 属性 908
  - bytesTotal 属性 909
  - close() 方法 910
  - currentFps 属性 911
  - NetStream() 构造函数 912
  - pause () 方法 915
  - play() 方法 916
  - seek () 方法 917
  - setBufferTime() 方法 918
  - time 属性 919
- NetStream 类 904
  - onMetaData 事件 912
  - onStatus 事件 913
- NetStream() 构造函数 912
- new 运算符 168
- newline 常数 36
- nextFrame 函数 75
- nextFrame() 方法 842
- nextScene 函数 76
- nextSibling 属性 1254
- nodeName 属性 1254
- nodeType 属性 1256
- nodeValue 属性 1257
- noise() 方法 297
- normalize() 方法 948
- not 逻辑 NOT 运算符 163

null 常数 37  
Number  
    MAX\_VALUE 属性 921  
    MIN\_VALUE 属性 922  
    NaN 属性 922  
    NEGATIVE\_INFINITY 属性 922  
    Number() 构造函数 923  
    POSITIVE\_INFINITY 属性 923  
    toString() 方法 924  
    valueOf() 方法 924  
Number 函数 77  
Number 类 920  
Number() 构造函数 923  
NUMERIC 属性 235  
Object  
    \_\_proto\_\_ 属性 932  
    \_\_resolve 属性 935  
    addProperty() 方法 926  
    constructor 属性 929  
    hasOwnProperty() 方法 930  
    isPrototypeOf() 方法 930  
    isPrototypeOf() 方法 931  
    Object() 构造函数 931  
    prototype 属性 933  
    registerClass() 方法 934  
    toString() 方法 938  
    unwatch() 方法 939  
    valueOf() 方法 940  
    watch() 方法 941  
Object 函数 78  
Object 类 925  
Object() 构造函数 931  
offset() 方法 949, 980  
offsetPoint() 方法 980  
on 处理函数 79  
onActivity 事件 368, 741  
onCancel 事件 532, 552  
onChanged 事件 1122  
onClipEvent 处理函数 80  
onClose 事件 1269  
onComplete 事件 533  
onConnect 事件 1269  
onData 事件 656, 843, 1226, 1270  
onDragOut 事件 335, 844  
onDragOver 事件 336, 844  
onEnterFrame 事件 845  
onHTTPError 事件 533  
onHTTPStatus 事件 657, 1227  
onID3 事件 1042  
onIMEComposition 事件 619  
onIOError 事件 535  
onKeyDown 事件 336, 640, 845  
onKeyUp 事件 337, 641, 846  
onKillFocus 事件 338, 847, 1123  
onLoad 事件 659, 848, 1043, 1082, 1229  
onLoadComplete 事件 892  
onLoadError 事件 894  
onLoadInit 事件 895  
onLoadProgress 事件 897  
onLoadStart 事件 898  
onMetaData 事件 912  
onMouseDown 事件 757, 849  
onMouseMove 事件 758, 849  
onMouseUp 事件 759, 849  
onMouseWheel 事件 760  
onOpen 事件 536  
onPress 事件 339, 850  
onProgress 事件 536  
onRelease 事件 339, 850  
onReleaseOutside 事件 339, 851  
onResize 事件 1054  
onRollOut 事件 340, 851  
onRollOver 事件 340, 852  
onScroller 事件 1124  
onSecurityError 事件 537  
onSelect 事件 417, 423, 539, 552  
onSetFocus 事件 340, 852, 1009, 1125  
onSoundComplete 事件 1044  
onStatus 事件 369, 678, 742, 913, 1025, 1089  
onUnload 事件 853  
onXML 事件 1271  
opaqueBackground 属性 853  
or 逻辑 OR 运算符 165  
ord 函数 81  
orientation 属性 958  
os 属性 385  
pageHeight 属性 958  
pageWidth 属性 958  
paletteMap() 方法 299  
paperHeight 属性 958  
paperWidth 属性 958  
parentNode 属性 1258  
parseCSS() 方法 1083  
parseFloat 函数 82  
parseInt 函数 82  
parseXML() 方法 1230  
password 属性 1127  
pause() 方法 915  
perlinNoise() 方法 300  
PGDN 属性 641

PGUP 属性 642  
PI 属性 705  
pixelAspectRatio 属性 385  
pixelBounds 属性 1197  
pixelDissolve() 方法 302  
play 函数 83  
play() 方法 855, 916  
playerType 属性 386  
Point  
    add() 方法 945  
    clone() 方法 945  
    distance() 方法 946  
    equals() 方法 946  
    interpolate() 方法 947  
    normalize() 方法 948  
    offset() 方法 949  
    polar() 方法 950  
    subtract() 方法 951  
    toString() 方法 951  
Point 类 943  
Point() 构造函数 949  
polar() 方法 950  
pop() 方法 235  
position 属性 1045  
POSITIVE\_INFINITY 属性 923  
pow() 方法 706  
prefix 属性 1259  
preserveAlpha 属性 436  
prevFrame 函数 84  
prevFrame() 方法 855  
previousSibling 属性 1260  
prevScene 函数 84  
print 函数 85  
printAsBitmap 函数 86  
printAsBitmapNum 函数 87  
PrintJob  
    addPage() 方法 954  
    orientation 属性 958  
    pageHeight 属性 958  
    pageWidth 属性 958  
    paperHeight 属性 958  
    paperWidth 属性 958  
    PrintJob() 构造函数 959  
    send() 方法 960  
    start() 方法 961  
PrintJob 类 953  
PrintJob() 构造函数 959  
printNum 函数 88  
private 语句 205  
prototype 属性 933  
public 语句 206  
push() 方法 236  
quality 属性 268, 317, 370, 508, 569, 584, 603  
random 函数 89  
random() 方法 707  
rate 属性 743  
ratios 属性 585, 604  
Rectangle  
    clone() 方法 967  
    contains() 方法 969  
    containsPoint() 方法 970  
    containsRectangle() 方法 971  
    equals() 方法 972  
    inflate() 方法 974  
    inflatePoint() 方法 975  
    intersection() 方法 976  
    intersects() 方法 977  
    isEmpty() 方法 978  
    offset() 方法 980  
    offsetPoint() 方法 980  
    setEmpty() 方法 983  
    toString() 方法 986  
    union() 方法 986  
Rectangle 类 963  
rectangle 属性 304  
Rectangle() 构造函数 981  
redMultiplier 属性 408  
redOffset 属性 409  
registerClass() 方法 934  
removeListener() 方法 253, 540, 553, 620, 642, 762, 899, 1010, 1055, 1128  
removeMovieClip 函数 90  
removeMovieClip() 方法 858  
removeNode() 方法 1261  
removeTextField() 方法 1129  
replaceSel() 方法 1130  
replaceText() 方法 1131  
restrict 属性 1132  
return 语句 207  
RETURNINDEXEDARRAY 属性 236  
reverse() 方法 237  
rgb 属性 410  
RIGHT 属性 643  
right 属性 982  
rightMargin 属性 1170  
rotate() 方法 726  
round() 方法 707  
sandboxType 属性 1001  
scale() 方法 728  
scale9Grid 属性 343, 860

- scaleMode 属性 1056
- scaleX 属性 490
- scaleY 属性 492
- screenColor 属性 386
- screenDPI 属性 386
- screenResolutionX 属性 387
- screenResolutionY 属性 387
- scroll 属性 118, 1134
- scroll() 方法 304
- scrollRect 属性 863
- security 类 990
- seek () 方法 917
- selectable 属性 1135
- Selection
  - addListener() 方法 1004
  - getBeginIndex() 方法 1005
  - getCaretIndex() 方法 1006
  - getEndIndex() 方法 1007
  - getFocus() 方法 1008
  - removeListener() 方法 1010
  - setFocus() 方法 1011
  - setSelection() 方法 1013
- Selection 类 1002
  - onSetFocus 事件 1009
- send() 方法 660, 679, 960, 1230, 1272
- sendAndLoad() 方法 662, 1232
- separatorBefore 属性 424
- serverString 属性 387
- set variable 语句 209
- set 语句 208
- set() 方法 1275
- setAdvancedAntialiasingTable() 方法 1177
- setBufferTime() 方法 918
- setClipboard() 方法 1090
- setCompositionString() 方法 621
- setConversionMode() 方法 622
- setDate() 方法 458
- setDefaultLang() 方法 691
- setEmpty() 方法 983
- setEnabled() 方法 623
- setFocus() 方法 1011
- setFullYear() 方法 458
- setGain() 方法 744
- setHours() 方法 459
- setInterval 函数 91
- setLoadCallback() 方法 692
- setMask() 方法 864
- setMilliseconds() 方法 460
- setMinutes() 方法 460
- setMode() 方法 371

- setMonth() 方法 461
- setMotionLevel() 方法 372
- setNewTextFormat() 方法 1136
- setPan() 方法 1045
- setPixel() 方法 305
- setPixel32() 方法 306
- setProperty 函数 95
- setQuality() 方法 373
- setRate() 方法 745
- setRGB() 方法 391
- setSeconds() 方法 462
- setSelectColor() 方法 1189
- setSelected() 方法 1190
- setSelection() 方法 1013
- setSilenceLevel() 方法 746
- setString() 方法 692
- setStyle() 方法 1084
- setTextFormat() 方法 1137
- setTime() 方法 462
- setTransform() 方法 392, 1046
- setUseEchoSuppression() 方法 748
- setUTCDate() 方法 463
- setUTCFullYear() 方法 464
- setUTCHours() 方法 465
- setUTCMilliseconds() 方法 466
- setUTCMinutes() 方法 466
- setUTCMonth() 方法 467
- setUTCSeconds() 方法 468
- setVolume() 方法 1048
- setYear() 方法 468
- shadowAlpha 属性 269
- shadowColor 属性 270
- SharedObject
  - clear() 方法 1016
  - data 属性 1017
  - flush() 方法 1019
  - getLocal() 方法 1020
  - getSize() 方法 1024
- SharedObject 类 1014
  - onStatus 事件 1025
- sharpness 属性 1139
- SHIFT 属性 644
- shift() 方法 237
- show() 方法 763
- showMenu 属性 1057
- showRedrawRegions 函数 96
- showSettings() 方法 1091
- silenceLevel 属性 749
- silenceTimeOut 属性 750
- sin() 方法 708

- size 属性 541, 983, 1170
- slice() 方法 238, 1066
- smoothing 属性 1207
- sort() 方法 239
- sortOn() 方法 241
- Sound
  - attachSound() 方法 1029
  - duration 属性 1029
  - getBytesLoaded() 方法 1031
  - getBytesTotal() 方法 1033
  - getPan() 方法 1033
  - getTransform() 方法 1035
  - getVolume() 方法 1037
  - id3 属性 1038
  - loadSound() 方法 1041
  - position 属性 1045
  - setPan() 方法 1045
  - setTransform() 方法 1046
  - setVolume() 方法 1048
  - Sound() 构造函数 1048
  - start() 方法 1049
  - stop() 方法 1050
- Sound 类 1027
  - onID3 事件 1042
  - onLoad 事件 1043
  - onSoundComplete 事件 1044
- Sound() 构造函数 1048
- SPACE 属性 644
- splice() 方法 245
- split() 方法 1068
- sqrt() 方法 709
- SQRT1\_2 属性 710
- SQRT2 属性 710
- Stage
  - addListener() 方法 1052
  - align 属性 1053
  - height 属性 1054
  - removeListener() 方法 1055
  - scaleMode 属性 1056
  - showMenu 属性 1057
  - width 属性 1058
- Stage 类 1051
  - onResize 事件 1054
- start() 方法 961, 1049
- startDrag 函数 97
- startDrag() 方法 865
- static 语句 210
- status 属性 1234
- stop 函数 98
- stop() 方法 866, 1050
- stopAllSounds 函数 98
- stopDrag 函数 99
- stopDrag() 方法 867
- strength 属性 271, 509, 570, 587, 606
- String
  - charAt() 方法 1061
  - charCodeAt() 方法 1062
  - concat() 方法 1062
  - fromCharCode() 方法 1063
  - indexOf() 方法 1063
  - lastIndexOf() 方法 1064
  - length 属性 1065
  - slice() 方法 1066
  - split() 方法 1068
  - String() 构造函数 1069
  - substr() 方法 1069
  - substring() 方法 1070
  - toLowerCase() 方法 1071
  - toString() 方法 1072
  - toUpperCase() 方法 1073
  - valueOf() 方法 1073
- String 函数 99
- String 类 1059
- String() 构造函数 1069
- stringIDArray 属性 693
- StyleSheet
  - clear() 方法 1077
  - getStyle() 方法 1078
  - getStyleNames() 方法 1080
  - load() 方法 1081
  - parseCSS() 方法 1083
  - setStyle() 方法 1084
  - StyleSheet() 构造函数 1085
  - transform() 方法 1086
- StyleSheet 类 1074
  - onLoad 事件 1082
- styleSheet 属性 1141
- StyleSheet() 构造函数 1085
- substr() 方法 1069
- substring 函数 100
- substring() 方法 1070
- subtract() 方法 951
- super 语句 211
- swapDepths() 方法 868
- switch 语句 212
- System
  - exactSettings 属性 1088
  - setClipboard() 方法 1090
  - showSettings() 方法 1091
  - useCodepage 属性 1092

System 类 1086  
    onStatus 事件 1089  
TAB 属性 645  
tabChildren 属性 869  
tabEnabled 属性 344, 870, 1143  
tabIndex 属性 345, 870, 1144  
tabStops 属性 1171  
tan() 方法 710  
target 属性 1171  
targetPath 函数 101  
tellTarget 函数 102  
text 属性 1145  
textColor 属性 1146  
TextField  
    \_alpha 属性 1099  
    \_height 属性 1114  
    \_highquality 属性 1114  
    \_name 属性 1121  
    \_parent 属性 1126  
    \_quality 属性 1128  
    \_rotation 属性 1133  
    \_soundbuftime 属性 1140  
    \_target 属性 1145  
    \_url 属性 1150  
    \_visible 属性 1151  
    \_width 属性 1151  
    \_x 属性 1153  
    \_xmouse 属性 1154  
    \_xscale 属性 1155  
    \_y 属性 1156  
    \_ymouse 属性 1156  
    \_yscale 属性 1157  
    addListener() 方法 1098  
    antiAliasType 属性 1100  
    autoSize 属性 1101  
    background 属性 1103  
    backgroundColor 属性 1103  
    border 属性 1104  
    borderColor 属性 1104  
    bottomScroll 属性 1105  
    condenseWhite 属性 1106  
    embedFonts 属性 1107  
    filters 属性 1108  
    getDepth() 方法 1109  
    getFontList() 方法 1110  
    getNewTextFormat() 方法 1110  
    getTextFormat() 方法 1111  
    gridFitType 属性 1112  
    hscroll 属性 1114  
    html 属性 1115

    htmlText 属性 1116  
    length 属性 1117  
    maxChars 属性 1117  
    maxhscroll 属性 1118  
    maxscroll 属性 1118  
    menu 属性 1119  
    mouseWheelEnabled 属性 1120  
    multiline 属性 1121  
    password 属性 1127  
    removeListener() 方法 1128  
    removeTextField() 方法 1129  
    replaceSel() 方法 1130  
    replaceText() 方法 1131  
    restrict 属性 1132  
    scroll 属性 1134  
    selectable 属性 1135  
    setNewTextFormat() 方法 1136  
    setTextFormat() 方法 1137  
    sharpness 属性 1139  
    styleSheet 属性 1141  
    tabEnabled 属性 1143  
    tabIndex 属性 1144  
    text 属性 1145  
    textColor 属性 1146  
    textHeight 属性 1147  
    textWidth 属性 1147  
    thickness 属性 1148  
    type 属性 1149  
    variable 属性 1150  
    wordWrap 属性 1152  
TextField 类 1093  
    onChanged 事件 1122  
    onKillFocus 事件 1123  
    onScroller 事件 1124  
    onSetFocus 事件 1125  
TextFormat  
    align 属性 1160  
    blockIndent 属性 1161  
    bold 属性 1161  
    bullet 属性 1162  
    color 属性 1162  
    font 属性 1163  
    getTextExtent() 方法 1163  
    indent 属性 1166  
    italic 属性 1166  
    kerning 属性 1167  
    leading 属性 1168  
    leftMargin 属性 1168  
    letterSpacing 属性 1169  
    rightMargin 属性 1170

- size 属性 1170
- tabStops 属性 1171
- target 属性 1171
- TextFormat() 构造函数 1172
- underline 属性 1174
- url 属性 1174
- TextFormat 类 1157
- TextFormat() 构造函数 1172
- textHeight 属性 1147
- TextRenderer
  - maxLevel 属性 1176
  - setAdvancedAntialiasingTable() 方法 1177
- TextRenderer 类 1175
- TextSnapshot
  - findText() 方法 1181
  - getCount() 方法 1182
  - getSelected() 方法 1182
  - getSelectedText() 方法 1183
  - getText() 方法 1184
  - getTextRunInfo() 方法 1186
  - hitTestTextNearPos() 方法 1188
  - setSelectColor() 方法 1189
  - setSelected() 方法 1190
- TextSnapshot 类 1179
- textWidth 属性 1147
- thickness 属性 1148
- this 属性 119
- threshold() 方法 307
- throw 语句 213
- time 属性 919
- toggleHighQuality 函数 103
- toLowerCase() 方法 1071
- top 属性 984
- topLeft 属性 985
- toString() 方法 246, 320, 411, 469, 513, 664, 729, 924, 938, 951, 986, 1072, 1262
- toUpperCase() 方法 1073
- trace 函数 104
- trackAsMenu 属性 347, 872
- Transform 类 1191
- transform 属性 873
- transform() 方法 1086
- Transform() 构造函数 1198
- transformPoint() 方法 730
- translate() 方法 731
- transparent 属性 309
- true 常数 38
- try..catch..finally 语句 214
- tx 属性 731
- ty 属性 732
- type 属性 272, 541, 588, 607, 1149
- typeof 运算符 178
- undefined 常数 39
- underline 属性 1174
- unescape 函数 105
- uninstall() 方法 441
- union() 方法 986
- UNIQUESORT 属性 246
- UNKNOWN 属性 624
- unloadClip() 方法 900
- unloadMovie 函数 105
- unloadMovie() 方法 875
- unloadMovieNum 函数 106
- unshift() 方法 247
- unwatch() 方法 939
- UP 属性 646
- updateAfterEvent 函数 107
- updateProperties() 方法 225
- upload() 方法 542
- url 属性 1174
- useCodepage 属性 1092
- useEchoSuppression 属性 752
- useHandCursor 属性 348, 877
- UTC() 方法 470
- valueOf() 方法 320, 470, 924, 940, 1073
- var 语句 218
- variable 属性 1150
- version 属性 388
- Video
  - \_alpha 属性 1202
  - \_height 属性 1205
  - \_name 属性 1206
  - \_parent 属性 1206
  - \_rotation 属性 1206
  - \_visible 属性 1207
  - \_width 属性 1208
  - \_x 属性 1208
  - \_xmouse 属性 1209
  - \_xscale 属性 1209
  - \_y 属性 1209
  - \_ymouse 属性 1210
  - \_yscale 属性 1210
  - attachVideo() 方法 1202
  - clear() 方法 1203
  - deblocking 属性 1204
  - height 属性 1205
  - smoothing 属性 1207
  - width 属性 1208
- Video 类 1199
- visible 属性 425

void 运算符 179  
watch() 方法 941  
while 语句 219  
width 属性 309, 375, 987, 1058, 1208  
with 语句 220  
wordWrap 属性 1152  
x 属性 952, 988  
XML

addRequestHeader() 方法 1214  
contentType 属性 1215  
createElement() 方法 1215  
createTextNode() 方法 1216  
docTypeDecl 属性 1218  
getBytesLoaded() 方法 1218  
getBytesTotal() 方法 1219  
idMap 属性 1220  
ignoreWhite 属性 1222  
load () 方法 1224  
loaded 属性 1225  
parseXML() 方法 1230  
send() 方法 1230  
sendAndLoad() 方法 1232  
status 属性 1234  
XML() 构造函数 1235  
xmlDecl 属性 1236

XML 类 1210

onData 事件 1226  
onHTTPStatus 事件 1227  
onLoad 事件 1229

XML() 构造函数 1235

xmlDecl 属性 1236

XMLNode

appendChild() 方法 1240  
attributes 属性 1241  
childNodes 属性 1241  
cloneNode() 方法 1243  
firstChild 属性 1244  
getNamespaceForPrefix() 方法 1246  
getPrefixForNamespace() 方法 1247  
hasChildNodes() 方法 1248  
insertBefore() 方法 1249  
lastChild 属性 1249  
localName 属性 1251  
namespaceURI 属性 1252  
nextSibling 属性 1254  
nodeName 属性 1254  
nodeType 属性 1256  
nodeValue 属性 1257  
parentNode 属性 1258  
prefix 属性 1259

previousSibling 属性 1260  
removeNode() 方法 1261  
toString() 方法 1262  
XMLNode() 构造函数 1263  
XMLNode 类 1237  
XMLNode() 构造函数 1263  
XMLSocket

close() 方法 1266  
connect() 方法 1267  
send() 方法 1272  
XMLSocket() 构造函数 1272

XMLSocket 类 1264

onClose 事件 1269  
onConnect 事件 1269  
onData 事件 1270  
onXML 事件 1271

XMLSocket() 构造函数 1272

XMLUI

accept() 方法 1274  
cancel() 方法 1274  
get() 方法 1274  
set() 方法 1275

XMLUI 类 1273

y 属性 952, 989

## A

安全性

allowDomain() 方法 991  
allowInsecureDomain() 方法 996  
loadPolicyFile() 方法 999  
sandboxType 属性 1001

## B

编译器指令 31

变形

colorTransform 属性 1193  
concatenatedColorTransform 属性 1194  
concatenatedMatrix 属性 1195  
matrix 属性 1196  
pixelBounds 属性 1197  
Transform() 构造函数 1198

## C

常数 35



## D

点

- length 属性 948
- Point() 构造函数 949
- x 属性 952
- y 属性 952

## J

矩形

- bottom 属性 965
- bottomRight 属性 966
- height 属性 973
- left 属性 979
- Rectangle() 构造函数 981
- right 属性 982
- size 属性 983
- top 属性 984
- topLeft 属性 985
- width 属性 987
- x 属性 988
- y 属性 989

矩阵

- a 属性 715
- b 属性 716
- c 属性 716
- d 属性 721
- Matrix() 构造函数 725
- tx 属性 731
- ty 属性 732

## Q

区域设置

- addDelayedInstance() 方法 683
- addXMLPath() 方法 684
- autoReplace 属性 684
- checkXMLStatus() 方法 685
- getDefaultLang() 方法 686
- initialize() 方法 687
- languageCodeArray 属性 687
- loadLanguageXML() 方法 688
- loadString() 方法 689
- loadStringEx() 方法 690
- setDefaultLang() 方法 691
- setLoadCallback() 方法 692
- setString() 方法 692
- stringIDArray 属性 693

全局函数 40

全局属性 108

## S

数组访问运算符 127

## T

添加连接（字符串）运算符 144

## Y

语句 179

运算符 121

