



组件语言参考

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甚至世界范围内注册。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。如果要访问本指南提到的第三方 Web 站点，您应自己承担因此而带来的风险。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已得到 Nellymoser, Inc. (www.nellymoser.com) 的许可。



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

Opera® 浏览器的版权归 © 1995-2002 Opera Software ASA 及其提供商所有。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>。

Visual SourceSafe 是 Microsoft Corporation 在美国和 / 或其它国家 / 地区的注册商标或商标。

版权所有 © 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允许拷贝、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：Sheila McGinn

撰稿：Bob Berry、Jen deHaan、Peter deHaan、David Jacowitz、Wade Pickett

主编：Rosana Francescato

主编：Lisa Stanziano

编辑：Evelyn Eldridge、Mary Ferguson、Mary Kraemer、Jessie Wood

生产管理：Patrice O'Neill、Kristin Conradi、Yuko Yagi

媒体设计和制作：Adam Barnett、Aaron Begley、Paul Benkman、John Francis、Geeta Karmarkar、Masayo Noda、Paul Rangel、Arena Reed、Mario Reynoso

特别感谢 Jody Bleyle、Mary Burger、Lisa Friendly、Stephanie Gowin、Bonnie Loo、Nivesh Rajbhandari、Mary Ann Walsh、Erick Vera、Yi Tan、测试版测试人员以及 Flash 和 Flash Player 工程小组和质量保证小组的全体成员。

第一版：2005 年 9 月

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

目 录

第 1 章：组件字典	29
组件类型	30
本章中的其它列表	33
 第 2 章：Accordion 组件（仅限 Flash Professional）	35
使用 Accordion 组件（仅限 Flash Professional）	36
自定义 Accordion 组件（仅限 Flash Professional）	40
Accordion 类（仅限 Flash Professional）	45
Accordion.change	49
Accordion.createChild()	50
Accordion.createSegment()	52
Accordion.destroyChildAt()	54
Accordion.getChildAt()	55
Accordion.getHeaderAt()	56
Accordion.numChildren	57
Accordion.selectedChild	58
Accordion.selectedIndex	59
 第 3 章：Alert 组件（仅限 Flash Professional）	61
使用 Alert 组件（仅限 Flash Professional）	62
自定义 Alert 组件（仅限 Flash Professional）	63
Alert 类（仅限 Flash Professional）	67
Alert.buttonHeight	71
Alert.buttonWidth	72
Alert.CANCEL	73
Alert.cancelLabel	73
Alert.click	74
Alert.NO	75
Alert.noLabel	76
Alert.NONMODAL	76
Alert.OK	78
Alert.okLabel	78
Alert.show()	79

Alert.YES.....	81
Alert.yesLabel	82
第 4 章：Button 组件	83
使用 Button 组件	84
自定义 Button 组件	87
Button 类	94
Button.icon	98
Button.label	99
Button.labelPlacement	100
第 5 章：CellRenderer API	101
了解 List 类	101
使用 CellRenderer API	102
CellRenderer.getCellIndex()	109
CellRenderer.getDataLabel()	110
CellRenderer.getPreferredHeight()	111
CellRenderer.getPreferredWidth()	112
CellRenderer.listOwner	113
CellRenderer.owner	113
CellRenderer.setSize()	114
CellRenderer.setValue()	115
第 6 章：CheckBox 组件	119
使用 CheckBox 组件	120
自定义 CheckBox 组件	122
CheckBox 类	124
CheckBox.click	129
CheckBox.label	130
CheckBox.labelPlacement	131
CheckBox.selected	133
第 7 章：Collection 接口（仅限 Flash Professional）	135
Collection 类（仅限 Flash Professional）	135
Collection.addItem()	136
Collection.contains()	137
Collection.clear()	138
Collection.getItemAt()	139
Collection.getIterator()	140
Collection.getLength()	141

Collection.isEmpty()	141
Collection.removeItem()	142

第 8 章：ComboBox 组件 145

使用 ComboBox 组件	147
自定义 ComboBox 组件	149
ComboBox 类	152
ComboBox.addItem()	157
ComboBox.addItemAt()	158
ComboBox.change	159
ComboBox.close()	160
ComboBox.close	161
ComboBox.dataProvider	162
ComboBox.dropdown	163
ComboBox.dropdownWidth	164
ComboBox.editable	165
ComboBox.enter	166
ComboBox.getItemAt()	168
ComboBox.itemRollOut	169
ComboBox.itemRollOver	170
ComboBox.labelField	171
ComboBox.labelFunction	172
ComboBox.length	173
ComboBox.open()	173
ComboBox.open	174
ComboBox.removeAll()	175
ComboBox.removeItemAt()	176
ComboBox.replaceItemAt()	177
ComboBox.restrict	179
ComboBox.rowCount	180
ComboBox.scroll	181
ComboBox.selectedIndex	182
ComboBox.selectedItem	183
ComboBox.sortItems()	184
ComboBox.sortItemsBy()	186
ComboBox.text	188
ComboBox.textField	188
ComboBox.value	189

第 9 章：数据绑定类（仅限 Flash Professional）	191
使数据绑定类在运行时可用（仅限 Flash Professional）	191
mx.data.binding 包中的类（仅限 Flash Professional）	192
Binding 类（仅限 Flash Professional）	192
Binding 类的构造函数	193
Binding.execute()	195
CustomFormatter 类（仅限 Flash Professional）	196
CustomFormatter.format()	198
CustomFormatter.unformat()	199
CustomValidator 类（仅限 Flash Professional）	200
CustomValidator.validate()	201
CustomValidator.validationError()	203
EndPoint 类（仅限 Flash Professional）	204
EndPoint 类的构造函数	205
EndPoint.component	206
EndPoint.constant	206
EndPoint.event	207
EndPoint.location	208
EndPoint.property	209
ComponentMixins 类（仅限 Flash Professional）	210
ComponentMixins.getField()	210
ComponentMixins.initComponent()	212
ComponentMixins.refreshDestinations()	212
ComponentMixins.refreshFromSources()	213
ComponentMixins.validateProperty()	214
DataType 类（仅限 Flash Professional）	216
DataType.encoder	217
DataType.formatter	218
DataType.getAnyTypedValue()	219
DataType.getAsBoolean()	220
DataType.getAsNumber()	220
DataType.getAsString()	221
DataType.getTypedValue()	222
DataType.kind	223
DataType.setAnyTypedValue()	223
DataType.setAsBoolean()	224
DataType.setAsNumber()	225
DataType.setAsString()	226
DataType.setTypedValue()	226
TypedValue 类（仅限 Flash Professional）	228
TypedValue 类的构造函数	228
TypedValue.type	229

TypedValue.typeName.....	229
TypedValue.value.....	230

第 10 章：DataGrid 组件（仅限 Flash Professional）..... 231

与 DataGrid 组件进行交互（仅限 Flash Professional）.....	232
使用 DataGrid 组件（仅限 Flash Professional）.....	233
DataGrid 性能策略.....	238
自定义 DataGrid 组件（仅限 Flash Professional）.....	240
DataGrid 类（仅限 Flash Professional）.....	243
DataGrid.addColumn().....	249
DataGrid.addColumnAt().....	250
DataGrid.addItem().....	251
DataGrid.addItemAt().....	252
DataGrid.cellEdit.....	253
DataGrid.cellFocusIn.....	255
DataGrid.cellFocusOut.....	257
DataGrid.cellPress.....	259
DataGrid.change.....	260
DataGrid.columnCount.....	261
DataGrid.columnNames.....	262
DataGrid.columnStretch.....	262
DataGrid.dataProvider.....	264
DataGrid.editable.....	265
DataGrid.editField().....	266
DataGrid.focusedCell.....	267
DataGrid.getColumnAt().....	268
DataGrid.getColumnIndex().....	269
DataGrid.headerHeight.....	270
DataGrid.headerRelease.....	271
DataGrid.hScrollPolicy.....	272
DataGrid.removeAllColumns().....	273
DataGrid.removeColumnAt().....	274
DataGrid.replaceItemAt().....	275
DataGrid.resizableColumns.....	276
DataGrid.selectable.....	277
DataGrid.showHeaders.....	278
DataGrid.sortableColumns.....	278
DataGrid.spaceColumnsEqually().....	279
DataGridColumn 类（仅限 Flash Professional）.....	280
DataGridColumn 类的构造函数.....	282
DataGridColumn.cellRenderer.....	282
DataGridColumn.columnName.....	283

DataGridColumn.editable.....	284
DataGridColumn.headerRenderer.....	285
DataGridColumn.headerText.....	285
DataGridColumn.labelFunction.....	286
DataGridColumn.resizable.....	287
DataGridColumn.sortable.....	288
DataGridColumn.sortOnHeaderRelease.....	289
DataGridColumn.width.....	290

第 11 章：DataHolder 组件（仅限 Flash Professional）..... 291

创建具有 DataHolder 组件的应用程序（仅限 Flash Professional）...	292
DataHolder 类.....	293
DataHolder.data.....	294

第 12 章：DataProvider API..... 295

DataProvider 类.....	295
DataProvider.addItem().....	297
DataProvider.addItemAt().....	297
DataProvider.editField().....	298
DataProvider.getEditingData().....	299
DataProvider.getItemAt().....	299
DataProvider.getItemID().....	300
DataProvider.length.....	301
DataProvider.modelChanged.....	301
DataProvider.removeAll().....	302
DataProvider.removeItemAt().....	303
DataProvider.replaceItemAt().....	304
DataProvider.sortItems().....	304
DataProvider.sortItemsBy().....	306

第 13 章：DataSet 组件（仅限 Flash Professional）..... 309

使用 DataSet 组件.....	309
DataSet 类（仅限 Flash Professional）.....	313
DataSet.addItem.....	315
DataSet.addItem().....	317
DataSet.addItemAt().....	318
DataSet.addSort().....	319
DataSet.afterLoaded.....	322
DataSet.applyUpdates().....	323
DataSet.calcFields.....	324
DataSet.changesPending().....	325

DataSet.clear()	326
DataSet.createItem()	327
DataSet.currentItem	328
DataSet.dataProvider	329
DataSet.deltaPacket	329
DataSet.deltaPacketChanged	330
DataSet.disableEvents()	331
DataSet.enableEvents()	332
DataSet.filtered	333
DataSet.filterFunc	335
DataSet.find()	338
DataSet.findFirst()	339
DataSet.findLast()	341
DataSet.first()	342
DataSet.getItemId()	343
DataSet.getIterator()	344
DataSet.getLength()	345
DataSet.hasNext()	346
DataSet.hasPrevious()	347
DataSet.hasSort()	348
DataSet.isEmpty()	349
DataSet.items	349
DataSet.itemClassName	350
DataSet.iteratorScrolled	351
DataSet.last()	352
DataSet.length	353
DataSet.loadFromSharedObj()	354
DataSet.locateById()	355
DataSet.logChanges	356
DataSet.modelChanged	357
DataSet.newItem	360
DataSet.next()	361
DataSet.previous()	362
DataSet.properties	363
DataSet.readOnly	363
DataSet.removeAll()	364
DataSet.removeItem	365
DataSet.removeItem()	367
DataSet.removeItemAt()	368
DataSet.removeRange()	369
DataSet.removeSort()	370
DataSet.resolveDelta	371

DataSet.saveToSharedObj()	373
DataSet.schema	374
DataSet.selectedIndex	375
DataSet.setIterator()	375
DataSet.setRange()	376
DataSet.skip()	377
DataSet.useSort()	378

第 14 章：DateChooser 组件（仅限 Flash Professional） 381

使用 DateChooser 组件（仅限 Flash Professional）	381
自定义 DateChooser 组件（仅限 Flash Professional）	383
DateChooser 类（仅限 Flash Professional）	386
DateChooser.change	390
DateChooser.dayNames	392
DateChooser.disabledDays	392
DateChooser.disabledRanges	393
DateChooser.displayedMonth	394
DateChooser.displayedYear	394
DateChooser.firstDayOfWeek	395
DateChooser.monthNames	396
DateChooser.scroll	396
DateChooser.selectableRange	398
DateChooser.selectedDate	399
DateChooser.showToday	399

第 15 章：DateField 组件（仅限 Flash Professional） 401

使用 DateField 组件（仅限 Flash Professional）	401
自定义 DateField 组件（仅限 Flash Professional）	403
DateField 类（仅限 Flash Professional）	406
DateField.change	409
DateField.close()	411
DateField.close	411
DateField.dateFormatter	413
DateField.dayNames	413
DateField.disabledDays	414
DateField.disabledRanges	415
DateField.displayedMonth	416
DateField.displayedYear	416
DateField.firstDayOfWeek	417
DateField.monthNames	417
DateField.open()	418

DateField.open	418
DateField.pullDown	420
DateField.scroll	420
DateField.selectableRange	422
DateField.selectedDate	423
DateField.showToday	423
 第 16 章：Delegate 类	425
Delegate.create()	426
 第 17 章：Deltaltem 类（仅限 Flash Professional）	427
Deltaltem.argList	428
Deltaltem.curValue	428
Deltaltem.delta	429
Deltaltem.kind	429
Deltaltem.message	430
Deltaltem.name	430
Deltaltem.newValue	431
Deltaltem.oldValue	431
 第 18 章：Delta 接口（仅限 Flash Professional）	433
Delta.addDeltaltem()	434
Delta.getChangeList()	435
Delta.getDeltaPacket()	436
Delta.getId()	436
Delta.getItemByName()	437
Delta.getMessage()	438
Delta.getOperation()	439
Delta.getSource()	440
 第 19 章：DeltaPacket 接口（仅限 Flash Professional）	443
DeltaPacket.getConfigInfo()	444
DeltaPacket.getIterator()	445
DeltaPacket.getSource()	446
DeltaPacket.getTimestamp()	446
DeltaPacket.getTransactionId()	447
DeltaPacket.logChanges()	448
 第 20 章：DepthManager 类	449
DepthManager.createChildAtDepth()	450
DepthManager.createClassChildAtDepth()	451

DepthManager.createClassObjectAtDepth()	452
DepthManager.createObjectAtDepth()	453
DepthManager.kBottom	454
DepthManager.kCursor	454
DepthManager.kNotopmost	455
DepthManager.kTooltip	455
DepthManager.kTop	456
DepthManager.kTopmost	456
DepthManager.setDepthAbove()	457
DepthManager.setDepthBelow()	457
DepthManager.setDepthTo()	458
第 21 章：EventDispatcher 类	461
事件对象	461
EventDispatcher 类 (API)	462
EventDispatcher.addEventListener()	462
EventDispatcher.dispatchEvent()	464
EventDispatcher.removeEventListener()	465
第 22 章：FLVPlayback 组件（仅限 Flash Professional）	467
使用 FLVPlayback 组件	469
使用提示点	474
播放多个 FLV 文件	481
从 FCS 流式加载 FLV 文件	483
自定义 FLVPlayback 组件	484
FLVPlayback 类	497
VideoError 类	642
VideoPlayer 类	649
使用 SMIL 文件	653
第 23 章：FocusManager 类	663
使用焦点管理器	664
自定义焦点管理器	666
FocusManager 类 (API)	667
FocusManager.defaultPushButton	670
FocusManager.defaultPushButtonEnabled	671
FocusManager.enabled	671
FocusManager.getFocus()	672
FocusManager.nextTabIndex	673
FocusManager.sendDefaultPushButtonEvent()	673
FocusManager.setFocus()	675

第 24 章：Form 类（仅限 Flash Professional）	677
使用 Form 类（仅限 Flash Professional）	677
Form 类（仅限 Flash Professional）	678
Form.currentFocusedForm	684
Form.getChildForm()	684
Form.indexInParentForm	685
Form.numChildForms	686
Form.parentIsForm	686
Form.parentForm	687
Form.rootForm	688
Form.visible	688
 第 25 章：Iterator 接口（仅限 Flash Professional）	 691
Iterator.hasNext()	691
Iterator.next()	692
 第 26 章：Label 组件	 693
使用 Label 组件	693
自定义 Label 组件	695
Label 类	696
Label.autoSize	699
Label.html	700
Label.text	701
 第 27 章：List 组件	 703
使用 List 组件	704
自定义 List 组件	708
List 类	711
List.addItem()	717
List.addItemAt()	718
List.cellRenderer	719
List.change	719
List.dataProvider	721
List.getItemAt()	722
List.hPosition	723
List.hScrollPolicy	724
List.iconField	725
List.iconFunction	726
List.itemRollOut	727
List.itemRollOver	729
List.labelField	730

List.labelFunction	731
List.length	732
List.maxHPosition	732
List.multipleSelection	733
List.removeAll()	734
List.removeItemAt()	735
List.replaceItemAt()	736
List.rowCount	737
List.rowHeight	738
List.scroll	739
List.selectable	740
List.selectedIndex	741
List.selectedIndices	742
List.selectedItem	743
List.selectedItems	744
List.setPropertiesAt()	745
List.sortItems()	746
List.sortItemsBy()	747
List.vPosition	748
List.vScrollPolicy	749
第 28 章：Loader 组件	751
使用 Loader 组件	751
自定义 Loader 组件	754
Loader 类	755
Loader.autoLoad	759
Loader.bytesLoaded	759
Loader.bytesTotal	760
Loader.complete	761
Loader.content	763
Loader.contentPath	764
Loader.load()	764
Loader.percentLoaded	766
Loader.progress	767
Loader.scaleContent	768
第 29 章：媒体组件（仅限 Flash Professional）	769
与媒体组件进行交互（仅限 Flash Professional）	770
了解媒体组件（仅限 Flash Professional）	771
使用媒体组件（仅限 Flash Professional）	773
媒体组件参数（仅限 Flash Professional）	779

使用媒体组件创建应用程序（仅限 Flash Professional）	781
自定义媒体组件（仅限 Flash Professional）	782
Media 类（仅限 Flash Professional）	782
Media.activePlayControl	786
Media.addCuePoint()	787
Media.aspectRatio	788
Media.associateController()	788
Media.associateDisplay()	789
Media.autoPlay	790
Media.autoSize	791
Media.backgroundColor	792
Media.bytesLoaded	793
Media.bytesTotal	793
Media.change	794
Media.click	795
Media.complete	796
Media.contentPath	797
Media.controllerPolicy	797
Media.controlPlacement	798
Media.cuePoint	799
Media.cuePoints	800
Media.displayFull()	800
Media.displayNormal()	801
Media.getCuePoint()	802
Media.horizontal	803
Media.mediaType	803
Media.pause()	804
Media.play()	805
Media.playheadChange	805
Media.playheadTime	806
Media.playing	807
Media.preferredHeight	808
Media.preferredWidth	808
Media.progress	809
Media.scrubbing	810
Media.removeAllCuePoints()	811
Media.removeCuePoint()	812
Media.setMedia()	812
Media.stop()	813
Media.totalTime	814
Media.volume	815
Media.volume	815

第 30 章：Menu 组件（仅限 Flash Professional）	817
与 Menu 组件进行交互（仅限 Flash Professional）	818
使用 Menu 组件（仅限 Flash Professional）	819
关于菜单项类型（仅限 Flash Professional）	821
关于初始化对象属性（仅限 Flash Professional）	824
Menu 参数（仅限 Flash Professional）	825
创建具有 Menu 组件的应用程序（仅限 Flash Professional）	825
自定义 Menu 组件（仅限 Flash Professional）	829
Menu 类（仅限 Flash Professional）	833
Menu.addItem()	837
Menu.addItemAt()	838
Menu.change	840
Menu.createMenu()	842
Menu.dataProvider	843
Menu.getItemAt()	844
Menu.hide()	845
Menu.indexOf()	847
Menu.menuHide	848
Menu.menuShow	850
Menu.removeAll()	852
Menu.removeItem()	854
Menu.removeItemAt()	855
Menu.rollOut	857
Menu.rollOver	859
Menu.setItemEnabled()	860
Menu.setItemSelected()	862
Menu.show()	863
MenuDataProvider 类	864
MenuDataProvider.addItem()	865
MenuDataProvider.addItemAt()	867
MenuDataProvider.getItemAt()	868
MenuDataProvider.indexOf()	870
MenuDataProvider.removeItem()	871
MenuDataProvider.removeItemAt()	872
 第 31 章：MenuBar 组件（仅限 Flash Professional）	 875
与 MenuBar 组件进行交互（仅限 Flash Professional）	876
使用 MenuBar 组件（仅限 Flash Professional）	876
自定义 MenuBar 组件（仅限 Flash Professional）	878
MenuBar 类（仅限 Flash Professional）	880
MenuBar.addMenu()	884

MenuBar.addMenuAt()	886
MenuBar.dataProvider	887
MenuBar.getMenuAt()	889
MenuBar.getMenuEnabledAt()	890
MenuBar.labelField	891
MenuBar.labelFunction	892
MenuBar.removeAll()	893
MenuBar.removeMenuAt()	894
MenuBar.setMenuEnabledAt()	895
第 32 章：NumericStepper 组件	897
使用 NumericStepper 组件	898
自定义 NumericStepper 组件	899
NumericStepper 类	902
NumericStepper.change	906
NumericStepper.maximum	908
NumericStepper.minimum	909
NumericStepper.nextValue	910
NumericStepper.previousValue	911
NumericStepper.stepSize	912
NumericStepper.value	913
第 33 章：PopUpManager 类	915
PopUpManager.createPopUp()	916
PopUpManager.deletePopUp()	917
第 34 章：ProgressBar 组件	919
使用 ProgressBar 组件	919
自定义 ProgressBar 组件	924
ProgressBar 类	926
ProgressBar.complete	929
ProgressBar.conversion	931
ProgressBar.direction	932
ProgressBar.indeterminate	933
ProgressBar.label	934
ProgressBar.labelPlacement	935
ProgressBar.maximum	936
ProgressBar.minimum	937
ProgressBar.mode	939
ProgressBar.percentComplete	940
ProgressBar.progress	941

ProgressBar.setProgress().	943
ProgressBar.source	945
ProgressBar.value	946
第 35 章：RadioButton 组件	949
使用 RadioButton 组件.	950
自定义 RadioButton 组件.	951
RadioButton 类	954
RadioButton.click	959
RadioButton.data	960
RadioButton.groupName	961
RadioButton.label	962
RadioButton.labelPlacement.	964
RadioButton.selected	965
RadioButton.selectedData	966
RadioButton.selection.	967
第 36 章：RadioButtonGroup 组件	969
第 37 章：RDBMSResolver 组件（仅限 Flash Professional）.	971
使用 RDBMSResolver 组件（仅限 Flash Professional）.	972
RDBMSResolver 类（仅限 Flash Professional）.	974
RDBMSResolver.addFieldInfo()	976
RDBMSResolver.beforeApplyUpdates.	977
RDBMSResolver.deltaPacket	978
RDBMSResolver.fieldInfo	978
RDBMSResolver.nullValue	979
RDBMSResolver.reconcileResults.	980
RDBMSResolver.reconcileUpdates.	981
RDBMSResolver.tableName	982
RDBMSResolver.updateMode	982
RDBMSResolver.updatePacket	983
RDBMSResolver.updateResults.	984
第 38 章：RectBorder 类.	985
对 RectBorder 类使用样式.	986
创建自定义的 RectBorder 实现.	988

第 39 章：Screen 类（仅限 Flash Professional）	991
将外部内容加载到屏幕（仅限 Flash Professional）	992
Screen 类 (API)（仅限 Flash Professional）	994
Screen.allTransitionsInDone	999
Screen.allTransitionsOutDone	1000
Screen.currentFocusedScreen	1000
Screen.getChildScreen()	1001
Screen.indexInParent	1002
Screen.mouseDown	1003
Screen.mouseDownSomewhere	1004
Screen.mouseMove	1004
Screen.mouseOut	1005
Screen.mouseOver	1006
Screen.mouseUp	1006
Screen.mouseUpSomewhere	1007
Screen.numChildScreens	1008
Screen.parentIsScreen	1008
Screen.parentScreen	1009
Screen.rootScreen	1010
 第 40 章：ScrollPane 组件	 1011
使用 ScrollPane 组件	1012
自定义 ScrollPane 组件	1014
ScrollPane 类	1015
ScrollPane.complete	1019
ScrollPane.content	1021
ScrollPane.contentPath	1022
ScrollPane.getBytesLoaded()	1023
ScrollPane.getBytesTotal()	1024
ScrollPane.hLineScrollSize	1025
ScrollPane.hPageScrollSize	1026
ScrollPane.hPosition	1027
ScrollPane.hScrollPolicy	1028
ScrollPane.progress	1029
ScrollPane.refreshPane()	1030
ScrollPane.scroll	1032
ScrollPane.scrollDrag	1034
ScrollPane.vLineScrollSize	1035
ScrollPane.vPageScrollSize	1036
ScrollPane.vPosition	1037
ScrollPane.vScrollPolicy	1038

第 41 章：SimpleButton 类	1039
SimpleButton.click	1042
SimpleButton.emphasized	1044
SimpleButton.emphasizedStyleDeclaration	1045
SimpleButton.selected	1045
SimpleButton.toggle	1046
 第 42 章：Slide 类（仅限 Flash Professional）	 1047
使用 Slide 类（仅限 Flash Professional）	1048
Slide 类 (API)（仅限 Flash Professional）	1049
Slide.autoKeyNav	1055
Slide.currentChildSlide	1056
Slide.currentFocusedSlide	1057
Slide.currentSlide	1057
Slide.defaultKeydownHandler	1058
Slide.firstSlide	1059
Slide.getChildSlide()	1060
Slide.gotoFirstSlide()	1061
Slide.gotoLastSlide()	1062
Slide.gotoNextSlide()	1064
Slide.gotoPreviousSlide()	1066
Slide.gotoSlide()	1068
Slide.hideChild	1069
Slide.indexInParentSlide	1070
Slide.lastSlide	1071
Slide.nextSlide	1072
Slide.numChildSlides	1072
Slide.overlayChildren	1073
Slide.parentIsSlide	1074
Slide.parentSlide	1075
Slide.playHidden	1075
Slide.previousSlide	1076
Slide.revealChild	1077
Slide.rootSlide	1077
 第 43 章：StyleManager 类	 1079
StyleManager.registerColorName()	1079
StyleManager.registerColorStyle()	1080
StyleManager.registerInheritingStyle()	1081

第 44 章：SystemManager 类	1083
SystemManager.screen	1083
第 45 章：TextArea 组件	1085
使用 TextArea 组件	1086
自定义 TextArea 组件	1088
TextArea 类	1090
TextArea.change	1094
TextArea.editable	1096
TextArea.hPosition	1097
TextArea.hScrollPolicy	1098
TextArea.html	1099
TextArea.length	1100
TextArea.maxChars	1101
TextArea.maxHPosition	1102
TextArea.maxVPosition	1103
TextArea.password	1104
TextArea.restrict	1105
TextArea.scroll	1106
TextArea.styleSheet	1108
TextArea.text	1109
TextArea.vPosition	1110
TextArea.vScrollPolicy	1111
TextArea.wordWrap	1112
第 46 章：TextInput 组件	1115
使用 TextInput 组件	1116
自定义 TextInput 组件	1118
TextInput 类	1119
TextInput.change	1123
TextInput.editable	1125
TextInput.enter	1125
TextInput.hPosition	1127
TextInput.length	1128
TextInput.maxChars	1129
TextInput.maxHPosition	1129
TextInput.password	1130
TextInput.restrict	1131
TextInput.text	1133

第 47 章：TransferObject 接口	1135
TransferObject.clone()	1135
TransferObject.getPropertyData()	1136
TransferObject.setPropertyData()	1137
 第 48 章：TransitionManager 类	 1139
使用 TransitionManager 类	1139
TransitionManager 类摘要	1140
TransitionManager.allTransitionsInDone	1141
TransitionManager.allTransitionsOutDone	1143
TransitionManager.content	1144
TransitionManager.contentAppearance	1144
TransitionManager.start()	1145
TransitionManager.startTransition()	1147
TransitionManager.toString()	1148
基于过渡的类	1149
 第 49 章：TreeDataProvider 接口（仅限 Flash Professional）	 1157
TreeDataProvider.addTreeNode()	1158
TreeDataProvider.addTreeNodeAt()	1159
TreeDataProvider.attributes.data	1160
TreeDataProvider.attributes.label	1161
TreeDataProvider.getTreeNodeAt()	1161
TreeDataProvider.removeTreeNode()	1162
TreeDataProvider.removeTreeNodeAt()	1163
 第 50 章：Tree 组件（仅限 Flash Professional）	 1165
使用 Tree 组件（仅限 Flash Professional）	1166
自定义 Tree 组件（仅限 Flash Professional）	1173
Tree 类（仅限 Flash Professional）	1177
Tree.addTreeNode()	1182
Tree.addTreeNodeAt()	1184
Tree.dataProvider	1186
Tree.firstVisibleNode	1187
Tree.getDisplayIndex()	1188
Tree.getIsBranch()	1189
Tree.getIsOpen()	1190
Tree.getNodeDisplayedAt()	1191
Tree.getTreeNodeAt()	1193
Tree.nodeClose	1194
Tree.nodeOpen	1195

Tree.refresh()	1197
Tree.removeAll()	1198
Tree.removeTreeNodeAt()	1199
Tree.selectedNode	1200
Tree.selectedNodes	1201
Tree.setIcon()	1202
Tree.setIsBranch()	1203
Tree.setIsOpen()	1204

第 51 章：Tween 类1207

使用 Tween 类	1208
将缓动方法应用于组件	1210
Tween.continueTo()	1214
Tween.duration	1215
Tween.fforward()	1215
Tween.finish	1216
Tween.FPS	1217
Tween.nextFrame()	1218
Tween.onMotionChanged	1219
Tween.onMotionFinished	1219
Tween.onMotionResumed	1220
Tween.onMotionStarted	1221
Tween.onMotionStopped	1222
Tween.position	1223
Tween.prevFrame()	1224
Tween.resume()	1225
Tween.rewind()	1226
Tween.start()	1227
Tween.stop()	1228
Tween.time	1229
Tween.toString()	1230
Tween.yoyo()	1230

第 52 章：UIComponent 类1233

UIComponent 类 (API)	1233
UIComponent.enabled	1236
UIComponent.focusIn	1237
UIComponent.focusOut	1238
UIComponent.getFocus()	1240
UIComponent.keyDown	1240
UIComponent.keyUp	1241

UIComponent.setFocus()	1242
UIComponent.tabIndex	1243
第 53 章: UIEventDispatcher 类	1245
UIEventDispatcher.keyDown	1246
UIEventDispatcher.keyUp	1247
UIEventDispatcher.load	1247
UIEventDispatcher.mouseDown	1248
UIEventDispatcher.mouseOut	1248
UIEventDispatcher.mouseOver	1249
UIEventDispatcher.mouseUp	1249
UIEventDispatcher.removeEventListener()	1250
UIEventDispatcher.unload	1250
第 54 章: UIObject 类	1251
UIObject.bottom	1253
UIObject.createClassObject()	1254
UIObject.createLabel()	1255
UIObject.createObject()	1256
UIObject.destroyObject()	1257
UIObject.doLater()	1258
UIObject.draw	1259
UIObject.getStyle()	1260
UIObject.height	1261
UIObject.hide	1261
UIObject.invalidate()	1262
UIObject.left	1263
UIObject.load	1263
UIObject.move	1264
UIObject.move()	1265
UIObject.redraw()	1266
UIObject.resize	1267
UIObject.reveal	1269
UIObject.right	1270
UIObject.scaleX	1270
UIObject.scaleY	1271
UIObject.setSize()	1271
UIObject.setSkin()	1272
UIObject.setStyle()	1273
UIObject.top	1275
UIObject.unload	1275

UIObject.visible	1276
UIObject.width	1277
UIObject.x	1277
UIObject.y	1278

第 55 章：UIScrollBar 组件1279

使用 UIScrollBar 组件	1279
自定义 UIScrollBar 组件	1282
UIScrollBar 类	1284
UIScrollBar.horizontal	1288
UIScrollBar.lineScrollSize	1289
UIScrollBar.pageScrollSize	1290
UIScrollBar.scroll	1291
UIScrollBar.scrollPosition	1293
UIScrollBar.setScrollProperties()	1295
UIScrollBar.setScrollTarget()	1296
UIScrollBar._targetInstanceName	1297

第 56 章：Web 服务类（仅限 Flash Professional）1299

使 Web 服务类在运行时可用（仅限 Flash Professional）	1300
Log 类（仅限 Flash Professional）	1300
Log 类的构造函数	1301
Log.getDateString()	1302
Log.logInfo()	1303
Log.logDebug()	1304
Log.level	1305
Log.name	1306
Log.onLog()	1307
PendingCall 类（仅限 Flash Professional）	1308
PendingCall.getOutputParameter()	1310
PendingCall.getOutputParameterByName()	1311
PendingCall.getOutputParameters()	1312
PendingCall.getOutputValue()	1312
PendingCall.getOutputValues()	1313
PendingCall.myCall	1314
PendingCall.onFault	1315
PendingCall.onResult	1316
PendingCall.request	1317
PendingCall.response	1317
SOAPCall 类（仅限 Flash Professional）	1318
SOAPCall.concurrency	1319

SOAPCall.doDecoding	1319
SOAPCall.doLazyDecoding	1320
WebService 类 (仅限 Flash Professional)	1320
支持的类型 (仅限 Flash Professional)	1322
WebService 安全性 (仅限 Flash Professional)	1324
WebService 类的构造函数	1324
WebService.getCall()	1326
WebService.myMethodName()	1327
WebService.onFault	1328
WebService.onLoad	1329

第 57 章: WebServiceConnector 组件 (仅限 Flash Professional) 1331

使用 WebServiceConnector 组件 (仅限 Flash Professional)	1331
WebServiceConnector 类 (仅限 Flash Professional)	1333
WebServiceConnector.multipleSimultaneousAllowed	1335
WebServiceConnector.operation	1335
WebServiceConnector.params	1336
WebServiceConnector.result	1337
WebServiceConnector.results	1338
WebServiceConnector.send	1339
WebServiceConnector.status	1339
WebServiceConnector.suppressInvalidCalls	1342
WebServiceConnector.trigger()	1343
WebServiceConnector.WSDLURL	1344

第 58 章: Window 组件 1347

使用 Window 组件	1347
自定义 Window 组件	1350
Window 类	1352
Window.click	1357
Window.closeButton	1358
Window.complete	1359
Window.content	1361
Window.contentPath	1362
Window.deletePopUp()	1362
Window.mouseDownOutside	1364
Window.title	1365
Window.titleStyleDeclaration	1366

第 59 章：XMLConnector 组件（仅限 Flash Professional）	1369
使用 XMLConnector 组件（仅限 Flash Professional）	1369
XMLConnector 类（仅限 Flash Professional）	1371
XMLConnector.direction	1372
XMLConnector.ignoreWhite	1373
XMLConnector.multipleSimultaneousAllowed	1373
XMLConnector.params	1374
XMLConnector.result	1375
XMLConnector.results	1376
XMLConnector.send	1376
XMLConnector.status	1377
XMLConnector.suppressInvalidCalls	1379
XMLConnector.trigger()	1380
XMLConnector.URL	1381
 第 60 章：XPathAPI 类	 1383
 第 61 章：XUpdateResolver 组件 （仅限 Flash Professional）	 1385
使用 XUpdateResolver 组件（仅限 Flash Professional）	1386
XUpdateResolver 类（仅限 Flash Professional）	1388
XUpdateResolver.beforeApplyUpdates	1389
XUpdateResolver.deltaPacket	1390
XUpdateResolver.includeDeltaPacketInfo	1390
XUpdateResolver.reconcileResults	1391
XUpdateResolver.updateResults	1392
XUpdateResolver.xupdatePacket	1392
 索引	 1393

组件字典

《组件语言参考》一书描述每个组件及其应用程序编程接口 (API)。若要了解如何使用、自定义和创建组件，请参见《使用组件》。在《组件语言参考》中，每项组件描述都包含有关下列内容的信息：

- 键盘交互
- 实时预览
- 辅助功能
- 设置组件参数
- 在应用程序中使用组件
- 自定义组件的样式和外观
- **ActionScript** 方法、属性和事件

组件是按字母顺序排列的。您也可以在下表中找到按类别排列的组件。

本章包含以下各节：

组件类型	30
本章中的其它列表	33

组件类型

下表按类别列出了 Macromedia Component Architecture 第 2 版中的不同组件。

用户界面 (UI) 组件

组件	说明
Accordion 组件 （仅限 Flash Professional）	一组垂直的互相重叠的视图，视图顶部有一些按钮，用户利用这些按钮可以在视图之间进行切换。
Alert 组件 （仅限 Flash Professional）	一个窗口，用于显示消息并提供捕获用户响应的按钮。
Button 组件	一个大小可调整的按钮，可使用自定义图标来自定义。
CheckBox 组件	允许用户进行布尔值选择（真或假）。
ComboBox 组件	允许用户从滚动的选择列表选择一个选项。该组件可以在列表顶部有一个可选择的文本字段，以允许用户搜索此列表。
DataGrid 组件 （仅限 Flash Professional）	允许用户显示和操作多列数据。
DateChooser 组件 （仅限 Flash Professional）	允许用户从日历中选择一个或多个日期。
DateField 组件 （仅限 Flash Professional）	一个不可选择的文本字段，并带有日历图标。当用户在组件的边框内单击时，Macromedia Flash 会显示一个 DateChooser 组件。
Label 组件	一个不可编辑的单行文本字段。
List 组件	允许用户从滚动列表选择一个或多个选项。
Loader 组件	一个包含已载入的 SWF 或 JPEG 文件的区块。
Menu 组件 （仅限 Flash Professional）	一个标准的桌面应用程序菜单，允许用户从列表选择一个命令。
MenuBar 组件 （仅限 Flash Professional）	水平的菜单栏。
NumericStepper 组件	一个带有可单击箭头的文本框，单击箭头可增大或减小数字的值。
ProgressBar 组件	显示一个过程（例如加载操作）的进度。
RadioButton 组件	允许用户在相互排斥的选项之间进行选择。
ScrollPane 组件	使用自动滚动条在有限的区域内显示影片剪辑、位图和 SWF 文件。
TextArea 组件	一个可随意编辑的多行文本字段。
TextInput 组件	一个可以随意编辑的单行文本输入字段。

组件	说明
Tree 组件（仅限 Flash Professional）	允许用户处理分级信息。
Window 组件	一个可拖动的窗口，带有标题栏、题注、边框、“关闭”按钮和内容显示区域。
UIScrollBar 组件	允许将滚动条添加至文本字段。

数据处理

组件	说明
数据绑定类（仅限 Flash Professional）	可实现 Flash 的运行时数据绑定功能的类。
DataHolder 组件（仅限 Flash Professional）	保存数据，并可用作组件之间的连接器。
DataProvider API	数据线性访问列表的模型，它提供简单的用于广播数据更改的数组操作功能。
DataSet 组件（仅限 Flash Professional）	一个构造块，用于创建数据驱动的应用程序。
RDBMSResolver 组件（仅限 Flash Professional）	用于将数据保存回任何支持的数据源。此组件对 Web 服务、JavaBean、servlet 或 ASP 页可接收并分析的 XML 进行翻译。
Web 服务类（仅限 Flash Professional）	允许访问使用简单对象访问协议 (SOAP) 的 Web 服务的类。这些类位于 mx.services 包中。
WebServiceConnector 组件（仅限 Flash Professional）	提供对 Web 服务方法调用的无脚本访问。
XMLConnector 组件（仅限 Flash Professional）	使用 HTTP GET 和 POST 方法来读写 XML 文档。
XUpdateResolver 组件（仅限 Flash Professional）	用于将数据保存回任何支持的数据源。此组件将增量数据包翻译为 XUpdate。

媒体组件

组件	说明
FLVPlayback 组件 （仅限 Flash Professional）	使您可以容易地将视频播放器包括在 Flash 应用程序中，以便通过 HTTP 从 Flash Video Streaming Service (FVSS) 或从 Flash Communication Server (FCS) 播放渐进式视频流。
MediaController 组件（仅限 Flash Professional）	在应用程序中控制媒体流的播放（请参见 第 769 页 的“ 媒体组件 （仅限 Flash Professional）”）。
MediaDisplay 组件（仅限 Flash Professional）	在应用程序中显示流媒体（请参见 第 769 页 的“ 媒体组件 （仅限 Flash Professional）”）。
MediaPlayback 组件（仅限 Flash Professional）	MediaDisplay 和 MediaController 组件的结合（请参见 第 769 页 的“ 媒体组件 （仅限 Flash Professional）”）。

管理器

类	说明
DepthManager 类	管理对象的堆叠深度。
FocusManager 类	处理组件间的 Tab 键导航。还处理当用户在应用程序中单击时的焦点变化。
PopUpManager 类	允许您创建和删除弹出式窗口。
StyleManager 类	允许您注册样式和管理继承的样式。
SystemManager 类	允许您管理激活哪个顶层窗口。
TransitionManager 类	使您可以管理幻灯片和影片剪辑的动画效果。

屏幕

类	说明
Form 类 （仅限 Flash Professional）	用于在运行时操作表单应用程序屏幕。
Screen 类 （仅限 Flash Professional）	Slide 和 Form 类的基类。
Slide 类 （仅限 Flash Professional）	用于在运行时操作幻灯片演示屏幕。

本章中的其它列表

本书还描述了前一部分所列的组件类别中未包含的几个类和 API。这些类和 API 列在下表中。

项	说明
CellRenderer API	一组属性和方法，基于列表的组件（List、DataGrid、Tree、Menu 和 ComboBox）使用它们来处理和显示每一行的自定义单元格内容。
Collection 接口（仅限 Flash Professional）	允许您管理一组相关项目（称为集合项目）。此集合中的每个集合项目都具有在集合项目类定义的元数据中描述的属性。
DataGridColumn 类（仅限 Flash Professional）	允许创建对象以用作数据网格的列。
Delegate 类	允许从一个对象传递到另一个对象的函数在第一个对象的上下文中运行。
Delta 接口（仅限 Flash Professional）	提供对传输对象、集合以及传输对象级别的更改的访问。
DeltaItem 类（仅限 Flash Professional）	提供有关对传输对象所执行的单个操作的信息。
DeltaPacket 接口（仅限 Flash Professional）	提供 Delta 接口和 DeltaItem 类，并允许您管理对数据的更改。
EventDispatcher 类	允许添加和删除事件侦听器，以便代码可以对事件做出适当的反应。
Iterator 接口（仅限 Flash Professional）	允许遍历集合所包含的对象。
MenuDataProvider 类	使指定给 Menu.dataProvider 属性的 XML 实例使用方法和属性来处理自己的数据以及相关的菜单视图。
RectBorder 类	描述用于控制组件边框的样式。
SimpleButton 类	允许控制按钮的某些外观和行为。
TransferObject 接口	定义 DataSet 组件所管理的项目必须实现的一组方法。
TreeDataProvider 接口（仅限 Flash Professional）	一组属性和方法，用于为 Tree.dataProvider 属性创建 XML。
Tween 类	使您可以轻松地在舞台上移动、淡化影片剪辑及调整其大小。
UIComponent 类	提供可让组件共享某些常用行为的方法、属性和事件。
UIEventDispatcher 类	允许组件发出某些事件。此类混合在 UIComponent 类中。
UIObject 类	所有第 2 版组件的基类。

Accordion 组件（仅限 Flash Professional）

Accordion 组件是包含一系列子项（一次显示一个）的浏览器。子项必须是从 **UIObject** 类继承的对象（包括使用 **Macromedia Component Architecture** 第 2 版构建的所有组件和屏幕）；但最常见的情况下，子项是 **View** 类的子类。这包括分配给 **mx.core.View** 类的影片剪辑。要保持 **accordion** 的子项的 **Tab** 键顺序，子项必须同时是 **View** 类的实例。

Accordion 创建并管理标题按钮，用户可以单击这些按钮在 **accordion** 的子项之间浏览。**Accordion** 呈纵向布局，其标题按钮横跨整个组件。一个子项与一个标题关联，而每个标题均从属于 **accordion**，而不从属于子项。当用户单击某个标题时，关联的子项即会显示在该标题下方。过渡到新的子项的过程使用过渡动画。

当带有子项的 **accordion** 接受焦点时，其标题的外观将发生变化以显示焦点。当用户使用 **Tab** 键切换到 **accordion** 中时，选定的标题将显示焦点指示符。不带子项的 **Accordion** 不会接受焦点。如果在选定子项内单击可接受焦点的组件，焦点将会出现在这些子项上。当 **Accordion** 实例具有焦点时，您可以使用下列按键对其进行控制：

键	说明
向下箭头，向右箭头	将焦点移到下一个子标题。在不更改选定子项的情况下，焦点将从最后一个标题回绕到第一个标题。
向上箭头，向左箭头	将焦点移到前一个子标题。在不更改选定子项的情况下，焦点将从第一个标题回绕到最后一个标题。
End	选择最后一个子项。
Enter 键 / 空格键	选择与具有焦点的标题关联的子项。
Home	选择第一个子项。
Page Down	选择下一个子项。选区将从最后一个子项回绕到第一个子项。
Page Up	选择前一个子项。选区将从第一个子项回绕到最后一个子项。
Shift+Tab	将焦点移到前一个组件。此组件可以在所选的子项内，也可以在 accordion 外；但它永远不会是同一 accordion 中的其它标题。
Tab	将焦点移到下一个组件。此组件可以在所选的子项内，也可以在 accordion 外；但它永远不会是同一 accordion 中的其它标题。

无法使用屏幕阅读器来访问 **Accordion** 组件。

使用 Accordion 组件（仅限 Flash Professional）

可以使用 **Accordion** 组件来显示多部分表单。例如，由三个子项组成的 **accordion** 可以呈现用户在其中填写发运地址、开单地址和付款信息（用于电子商务交易）的表单。使用 **accordion**（而不是多个网页）可以最大程度地减少服务器通信量，并让用户能够对应用程序中的进度和上下文有一个更好的认识。

Accordion 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 **Accordion** 组件实例设置以下创作参数：

childIcons 是一个数组，它指定要用作 **Accordion** 标题中的图标的库元件的链接标识符。默认值为 []（空数组）。

childLabels 是一个数组，它指定要在 **Accordion** 的标题中使用的文本标签。默认值为 []（空数组）。

childNames 是一个数组，它指定 **Accordion** 子项的实例名称。您输入的值将成为在 **childSymbols** 参数中指定的子元件的实例名称。默认值为 []（空数组）。

childSymbols 是一个数组，它指定要用于创建 **Accordion** 子项的库元件的链接标识符。默认值为 []（空数组）。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **Accordion** 组件实例设置以下创作参数：

enabled 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 `true`。

visible 是一个布尔值，它指示对象是可见 (`true`) 还是不可见 (`false`)。默认值为 `true`。

提醒

minHeight 和 **minWidth** 属性由内部的大小调整例程使用。这两个属性在 **UIObject** 中定义，并可以根据需要被不同的组件覆盖。在为应用程序创建自定义布局管理器时，可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不具有明显效果。

您可以编写 **ActionScript**，以便使用 **Accordion** 组件的属性、方法和事件来控制该组件的其它选项。有关更多信息，请参见第 45 页的“**Accordion 类（仅限 Flash Professional）**”。

创建具有 Accordion 组件的应用程序

在本例中，应用程序开发人员正在构建在线商店的“结帐”部分。设计要求具有一个包含三个表单的 **accordion**，用户可在其中输入他们的发运地址、开单地址和付款信息。发运地址和开单地址表单相同。

使用屏幕将一个 **Accordion** 组件添加到应用程序：

1. 在 **Flash** 中，选择“文件”>“新建”，然后选择“Flash 表单应用程序”。

2. 双击文本表单 1 并输入名称 **addressForm**。

尽管 **addressForm** 屏幕未显示在库中，但它是 **Screen** 类的一个元件。因为 **Screen** 类是 **View** 类的子类，所以 **accordion** 可使用它作为子项。

3. 在表单处于选定状态的情况下，在“属性”检查器中将其 **visible** 属性设置为 **false**。

这将在应用程序中隐藏表单的内容；表单只会出现在 **accordion** 中。

4. 将诸如 **Label** 和 **TextInput** 等组件从“组件”面板拖到表单上，以便创建一个模拟地址表单；对这些组件进行排列，然后在“组件”检查器的“参数”窗格中设置它们的属性。

将表单元素定位在表单的左上角。表单的此角位于 **Accordion** 的左上角中。

5. 重复步骤 2-4，以便创建一个名为 **checkoutForm** 的屏幕。

6. 创建一个名为 **accordionForm** 的新屏幕。

7. 将一个 **Accordion** 组件从“组件”面板拖到 **accordionForm** 表单中，并将其命名为 **my_acc**。

8. 在 **my_acc** 处于选定状态的情况下，在“属性”检查器中执行以下操作：

- 对于 **childSymbols** 属性，输入 **addressForm**、**addressForm** 和 **checkoutForm**。
这些字符串指定用于创建 **accordion** 子项的屏幕的名称。



前两个子项是同一屏幕的实例，因为发运地址表单和开单地址表单相同。

- 对于 **childNames** 属性，输入 **shippingAddress**、**billingAddress** 和 **checkout**。
这些字符串是该 **Accordion** 子项的 **ActionScript** 名称。

- 对于 **childLabels** 属性，输入 **Shipping Address**（发运地址）、**Billing Address**（开单地址）和 **Checkout**（结帐）。

这些字符串是 **Accordion** 标题上的文本标签。

9. 选择“控制”>“测试影片”。

将 Accordion 组件添加到应用程序：

1. 选择“文件”>“新建”，然后创建新的 Flash 文档。
2. 选择“插入”>“新建元件”，并将其命名为 **AddressForm**。
3. 在“创建新元件”对话框中，单击“高级”按钮并选择“为 ActionScript 导出”。在“AS 2.0 类”字段中，输入 **mx.core.View**。

若要保持某个 accordion 子项中的 Tab 键顺序，该子项必须同时是 View 类的实例。

4. 将诸如 Label 和 TextInput 等组件从“组件”面板拖到舞台上，以便创建一个模拟地址表单；对这些组件进行排列，然后在“组件”检查器的“参数”窗格中设置它们的属性。

将表单元素定位在舞台上相对于 0,0（中心）的位置。影片剪辑的 0,0 坐标位于 accordion 的左上角。

5. 选择“编辑”>“编辑文档”返回到主时间轴。
6. 重复步骤 2-5，以便创建一个名为 **CheckoutForm** 的影片剪辑。
7. 将一个 Accordion 组件从“组件”面板拖到主时间轴上的舞台上。
8. 在“属性”检查器中，执行以下操作：

- 输入实例名称 **my_acc**。
- 对于 childSymbols 属性，输入 **AddressForm**、**AddressForm** 和 **CheckoutForm**。

这些字符串指定用于创建 accordion 子项的影片剪辑的名称。



前两个子项是同一影片剪辑的实例，因为发运地址表单和开单地址表单相同。

- 对于 childNames 属性，输入 **shippingAddress**、**billingAddress** 和 **checkout**。
这些字符串是该 Accordion 子项的 ActionScript 名称。
 - 对于 childLabels 属性，输入 **Shipping Address**（发运地址）、**Billing Address**（开单地址）和 **Checkout**（结帐）。
这些字符串是 Accordion 标题上的文本标签。
 - 对于 childIcons 属性，输入 **AddressIcon**、**AddressIcon** 和 **CheckoutIcon**。
这些字符串指定用作 accordion 标题上的图标的影片剪辑元件的链接标识符。如果希望图标出现在标题中，您必须创建这些影片剪辑元件。
9. 选择“控制”>“测试影片”。

使用 ActionScript 将子项添加到 Accordion 组件:

1. 选择“文件”>“新建”，然后创建 Flash 文档。
2. 将 Accordion 组件从“组件”面板拖到舞台上。
3. 在“属性”检查器中，输入实例名称 **my_acc**。
4. 将 TextInput 组件拖动到库中。

此操作会将该组件添加到库中，以便能够在步骤 6 中将其动态地实例化。

5. 在时间轴第一帧中的“动作”面板中，输入以下代码（这段代码调用 createChild() 方法来创建其子视图）：

```
import mx.core.View;
```

```
// 为要显示在 my_acc 对象中的每个表单创建子面板。  
my_acc.createChild(View, "shippingAddress", {label: "Shipping Address"});  
my_acc.createChild(View, "billingAddress", {label: "Billing Address"});  
my_acc.createChild(View, "payment", {label: "Payment"});
```

6. 在第一帧中的“动作”面板中，在步骤 5 中输入的代码下，输入以下代码（此代码向 accordion 子项添加两个 TextInput 组件实例）：

```
// 为 shippingAddress 面板创建子项文本输入。  
var firstNameChild_obj:Object =  
    my_acc.shippingAddress.createChild("TextInput", "firstName", {text:  
        "First Name"});  
  
// 设置文本输入的位置。  
firstNameChild_obj.move(10, 38);  
firstNameChild_obj.setSize(110, 20);  
  
// 创建另一个子项输入。  
var lastNameChild_obj:Object =  
    my_acc.shippingAddress.createChild("TextInput", "lastName", {text:  
        "Last Name"});  
  
// 设置文本输入的位置。  
lastNameChild_obj.move(150, 38);  
lastNameChild_obj.setSize(140, 20);
```

自定义 Accordion 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平和垂直方向上使 **Accordion** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参见 `UIObject.setSize()`）。

`setSize()` 方法和“变形”工具只会更改 **Accordion** 标题的宽度以及其内容区域的宽度和高度。标题的高度以及子项的宽度和高度不受影响。只有调用 `setSize()` 方法才能更改 **Accordion** 的边框矩形。

如果标题太小而无法包含标签文本，则标签将会被裁剪。如果 **accordion** 的内容区域比子项小，则子项将会被裁剪。

对 Accordion 组件使用样式

您可以设置样式属性，以便更改 **Accordion** 组件的边框和背景的外观。

Accordion 组件使用下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。这是唯一不继承样式值的颜色样式。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。
<code>backgroundColor</code>	光晕和范例	背景色。默认颜色为白色。
<code>borderStyle</code>	光晕和范例	Accordion 组件使用 <code>RectBorder</code> 实例作为其边框并对在该类上定义的样式做出响应。有关更多信息，请参见第 985 页的“ <code>RectBorder</code> 类”。 Accordion 组件的默认边框样式值为 "solid"。
<code>headerHeight</code>	光晕和范例	标题按钮的高度（以像素为单位）。默认值为 22。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题的默认值为 <code>0x0B333C</code> ，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>0x848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	标题标签的字体名称。默认值为 "_sans"。

样式	主题	说明
fontSize	光晕和范例	标题标签字体的磅值。默认值为 10。
fontStyle	光晕和范例	标题标签的字体样式；"normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	标题标签的字体粗细；"none" 或 "bold"。默认值为 "none"。 在调用 <code>setStyle()</code> 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textDecoration	光晕和范例	文本修饰；"none" 或 "underline"。
openDuration	光晕和范例	过渡动画的持续时间（以毫秒为单位）。
openEasing	光晕和范例	对控制动画的补间函数的引用。默认为正弦输入 / 输出。有关更多信息，请参见《使用组件》中的“自定义组件动画”。

例如，以下代码将名为 **my_acc** 的 **Accordion** 实例中的字体样式外观设置为斜体：

```
my_acc.setStyle("fontStyle", "italic");
```

如果样式属性的名称以“Color”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

对 Accordion 组件使用外观

Accordion 组件使用外观来表示其标题按钮的可视状态。要在创作时为按钮和标题栏设置外观，请在其中一个主题 FLA 文件的库中修改 **Flash UI Components 2/Themes/MMDefault/Accordion Assets skins states** 文件夹中的外观元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。

Accordion 组件由边框、背景、标题按钮和子项组成。默认情况下，边框和背景由 **RectBorder** 类提供。有关设置 **RectBorder** 类外观的信息，请参见第 985 页的“**RectBorder** 类”。您可以使用下列外观设置标题的外观。

属性	说明	默认值
falseUpSkin	所有折叠子项上标题的弹起（正常）状态。	accordionHeaderSkin
falseDownSkin	所有折叠子项上标题的按下状态。	accordionHeaderSkin
falseOverSkin	所有折叠子项上标题的滑过状态。	accordionHeaderSkin
falseDisabled	所有折叠子项上标题的禁用状态。	accordionHeaderSkin
trueUpSkin	展开子项上标题的弹起（正常）状态。	accordionHeaderSkin

属性	说明	默认值
trueDownSkin	展开子项上标题的按下状态。	accordionHeaderSkin
trueOverSkin	展开子项上标题的滑过状态。	accordionHeaderSkin
trueDisabledSkin	展开子项上标题的禁用状态。	accordionHeaderSkin

使用 ActionScript 绘制 Accordion 标题

“光晕”主题和“范例”主题中的默认标题针对所有状态均使用同一外观元素，并通过 **ActionScript** 绘制实际图形。“光晕”实现使用 **RectBorder** 类的扩展及自定义绘制 API 代码绘制状态。“范例”实现使用相同的外观和相同的 **ActionScript** 类作为 **Button** 外观。

若要创建用作外观并提供不同状态的 **ActionScript** 类，外观可以读取其 `borderStyle` 样式属性以确定状态。下表显示了为每种外观设置的边框样式：

属性	边框样式
falseUpSkin	falseup
falseDownSkin	falsedown
falseOverSkin	falserollover
falseDisabled	falsedisabled
trueUpSkin	trueup
trueDownSkin	truedown
trueOverSkin	truerollover
trueDisabledSkin	truedisabled

创建 ActionScript 自定义 Accordion 标题外观：

1. 新建一个 ActionScript 类文件。

对于本示例，将该文件命名为 **RedGreenBlueHeader.as**。

2. 将以下 ActionScript 复制到文件中：

```
import mx.skins.RectBorder;
import mx.core.ext.UIObjectExtensions;

class RedGreenBlueHeader extends RectBorder
{
    static var symbolName_str:String = "RedGreenBlueHeader";
    static var symbolOwner_obj:Object = RedGreenBlueHeader;

    function size():Void
    {
        var color_num:Number; // 颜色
        var borderStyle_str:String = getStyle("borderStyle"); // Accordion 的属性
```

```

// 定义每种选项卡状态下 Accordion 中的每个选项卡的颜色。
switch (borderStyle_str) {
    case "falseup":
    case "falsesrollover":
    case "falsedisabled":
        color_num = 0x7777FF;
        break;
    case "falsedown":
        color_num = 0x77FF77;
        break;
    case "trueup":
    case "truedown":
    case "truerollover":
    case "truedisabled":
        color_num = 0xFF7777;
        break;
}

// 清除默认样式并绘制自定义样式。
clear();
lineStyle(0, 0, 100);
beginFill(color_num, 100);
drawRect(0, 0, __width, __height);
endFill();
}

// 外观所需内容
static function classConstruct():Boolean
{
    UIObjectExtensions.Extensions();
    _global.skinRegistry["AccordionHeaderSkin"] = true;
    return true;
}
static var classConstructed_b1:Boolean = classConstruct();
static var UIObjectExtensionsDependency_obj:Object = UIObjectExtensions;
}

```

此类根据边框样式创建方框：为假弹起、滑过及禁用状态创建蓝色框；为正常按下状态创建绿色框；为展开子项创建红色框。

3. 保存该文件。
4. 创建一个新的 FLA 文件并将其与 AS 文件保存在同一个文件夹中。
5. 通过选择“插入” > “新建元件”创建一个新元件。
6. 将名称设置为 AccordionHeaderSkin。
7. 如果高级视图未显示出来，则单击“高级”按钮。
8. 选择“为 ActionScript 导出”。

标识符将自动填写为 AccordionHeaderSkin。

9. 将 AS 2.0 类设置为 RedGreenBlueHeader。
10. 确保 “在第一帧导出” 处于选中状态，然后单击 “确定”。
11. 在 “场景 1” 中，将一个 **Accordion** 组件拖到舞台上。
12. 设置 **Accordion** 属性，使其显示多个子项。
例如，将 childLabels 设置为 [One,Two,Three] 数组并将 childNames 设置为 [one,two,three] 数组。
13. 选择 “控制” > “测试影片”。

使用影片剪辑自定义 Accordion 标题外观

上面的示例演示了如何使用 **ActionScript** 类自定义 **Accordion** 标题外观，“光晕”主题和“范例”主题中提供的外观都使用此方法。但是，因为该示例使用简单的着色框，所以在本例中将不同影片剪辑元件用作标题外观较为简单。

创建 Accordion 标题外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 通过选择 “插入” > “新建元件” 创建一个新元件。
3. 将名称设置为 RedAccordionHeaderSkin。
4. 如果高级视图未显示出来，则单击 “高级” 按钮。
5. 选择 “为 ActionScript 导出”。
- 标识符将自动填写为 RedAccordionHeaderSkin。
6. 将 AS 2.0 类文本框保留为空。
7. 确保 “在第一帧导出” 处于选中状态，然后单击 “确定”。
8. 打开新元件以进行编辑。
9. 使用绘画工具创建一个黑色线条和红色填充的方框。
10. 将边框样式设置为极细。
11. 设置方框（包括边框），使它位于 (0,0) 并且宽度和高度皆为 100。
ActionScript 代码将根据需要调整外观大小。
12. 重复步骤 2-11 创建绿色和蓝色外观，根据颜色相应命名。
13. 单击 “返回” 按钮返回主时间轴。
14. 将 **Accordion** 组件拖到舞台上。
15. 设置 **Accordion** 属性，使其显示多个子项。

例如，将 childLabels 设置为 [One,Two,Three] 数组并将 childNames 设置为 [one,two,three] 数组。

16. 将以下 **ActionScript** 代码复制到 “动作” 面板中，同时选中 **Accordion** 实例：

```
onClipEvent(initialize) {  
    falseUpSkin = "RedAccordionHeaderSkin";  
    falseDownSkin = "GreenAccordionHeaderSkin";  
    falseOverSkin = "RedAccordionHeaderSkin";  
    falseDisabled = "RedAccordionHeaderSkin";  
    trueUpSkin = "BlueAccordionHeaderSkin";  
    trueDownSkin = "BlueAccordionHeaderSkin";  
    trueOverSkin = "BlueAccordionHeaderSkin";  
    trueDisabledSkin = "BlueAccordionHeaderSkin";  
}
```

17. 选择 “控制” > “测试影片”。

Accordion 类（仅限 Flash Professional）

继承 **MovieClip** > **UIObject** 类 > **UIComponent** 类 > **View** > **Accordion**

ActionScript 类名称 **mx.containers.Accordion**

Accordion 组件包含一次显示一个的子项。每个子项都有在创建子项时创建的对应标题按钮。子项必须是 **UIObject** 的实例。

当影片剪辑元件成为 **accordion** 的子项时，它将自动成为 **UIObject** 类的实例。但是，要保持某个 **accordion** 子项中的 **Tab** 键顺序，该子项必须同时是 **View** 类的实例。如果使用某个影片剪辑元件作为子项，请将其 “AS 2.0 类” 字段设置为 **mx.core.View**，以使其继承自 **View** 类。

使用 **ActionScript** 设置 **Accordion** 类的属性将会覆盖在 “属性” 检查器或 “组件” 检查器中设置的同名参数。

每个组件类都有一个 **version** 属性，该属性是一个类属性。类属性只能用于该类本身。**version** 属性会返回一个字符串，该字符串指示组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.containers.Accordion.version);
```



代码 `trace(my_accInstance.version);` 返回 `undefined`。

Accordion 类的方法摘要

下表列出了 Accordion 类的方法。

方法	说明
<code>Accordion.createChild()</code>	创建 Accordion 实例的子项。
<code>Accordion.createSegment()</code>	创建 Accordion 实例的子项。此方法的参数与 <code>createChild()</code> 方法的参数不同。
<code>Accordion.destroyChildAt()</code>	破坏位于指定索引位置的子项。
<code>Accordion.getChildAt()</code>	获取对位于指定索引位置的子项的引用。
<code>Accordion.getHeaderAt()</code>	获取对位于指定索引位置的标题对象的引用。

从 UIObject 类继承的方法

下表列出了 Accordion 类从 UIObject 类继承的方法。从 Accordion 对象调用这些方法时，请使用 `accordionInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

从 UIComponent 类继承的方法

下表列出了 **Accordion** 类从 **UIComponent** 类继承的方法。从 **Accordion** 对象调用这些方法时，请使用 *accordionInstance.methodName* 的形式。

方法	说明
UIComponent.setFocus()	返回对具有焦点的对象的引用。
UIComponent.setFocus()	将焦点设置到组件实例中。

Accordion 类的属性摘要

下表列出了 **Accordion** 类的属性。

属性	说明
Accordion.numChildren	Accordion 实例的子项数。
Accordion.selectedChild	指向选定子项的引用。
Accordion.selectedIndex	选定子项的索引位置。

从 UIObject 类继承的属性

下表列出了 **Accordion** 类从 **UIObject** 类继承的属性。访问这些属性时，请使用 *accordionInstance.propertyName* 的形式。

属性	说明
UIObject.bottom	只读；对象的底边缘位置（相对于其父对象的底边缘）。
UIObject.height	只读；对象的高度，以像素为单位。
UIObject.left	只读；对象的左边缘（以像素为单位）。
UIObject.right	只读；对象的右边缘位置（相对于其父对象的右边缘）。
UIObject.scaleX	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
UIObject.scaleY	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
UIObject.top	只读；对象上边缘的位置（相对于其父对象）。
UIObject.visible	一个布尔值，它指示对象是 (true) 否 (false) 可见。
UIObject.width	只读；对象的宽度，以像素为单位。
UIObject.x	只读；对象的左边缘（以像素为单位）。
UIObject.y	只读；对象的上边缘（以像素为单位）。

从 UIComponent 类继承的属性

下表列出了 **Accordion** 类从 **UIComponent** 类继承的属性。访问这些属性时，请使用 *accordionInstance.propertyName* 的形式。

属性	说明
<code>UIComponent.enabled</code>	指示组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指示文档中组件的 Tab 键顺序。

Accordion 类的事件摘要

下表列出了 **Accordion** 类的一个事件。

事件	说明
<code>Accordion.change</code>	当 Accordion 的 <code>selectedIndex</code> 和 <code>selectedChild</code> 属性由于用户单击鼠标或按键而更改时，广播到所有注册的侦听器。

从 UIObject 类继承的事件

下表列出了 **Accordion** 类从 **UIObject** 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

从 UIComponent 类继承的事件

下表列出了 **Accordion** 类从 **UIComponent** 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

Accordion.change

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
    // 在此处插入您的代码。
};
accordionInstance.addEventListener("change", listenerObject);
```

用法 2:

```
on (change) {
    // 在此处插入您的代码。
}
```

说明

事件；在 **Accordion** 的 `selectedIndex` 和 `selectedChild` 属性更改时广播到所有注册的侦听器。只有在用户的鼠标单击或按键动作更改了 `selectedChild` 或 `selectedIndex` 值时才会广播该事件，而当 **ActionScript** 的值更改时不会广播。在过渡动画出现之前广播该事件。

第 2 版组件使用调度程序 / 事件侦听器模型。**Accordion** 组件会在它的其中一个按钮被单击时调度 `change` 事件，该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作“处理函数”）进行处理。您需要调用 `addEventListener()` 方法并将对处理函数的引用作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

Accordion change 事件同时包含两个独有的事件对象属性：

- `newValue` 数字；将要选择的子项的索引。
- `prevValue` 数字；以前选中的子项的索引。

示例

以下示例使用一个名为 `my_acc` 的 **Accordion** 实例，其中包含三个子面板，分别标记为“Shipping Address”（发运地址）、“Billing Address”（开单地址）和“Payment”（付款）。代码定义了一个名为 `my_accListener` 的处理函数，并将该处理函数作为第二个参数传递给 `my_acc.addEventListener()` 方法。`change` 处理函数在 `eventObject` 参数中捕获该事件对象。在广播 `change` 事件时，将一条 `trace` 语句发送到“输出”面板。

```
// 创建新的侦听器对象。
var my_accListener:Object = new Object();
my_accListener.change = function() {
    trace("Changed to different view");
    // 将子面板的标签分配给变量。
    var selectedChild_str:String = my_acc.selectedChild.label;
    // 基于所选的子项执行操作。
    switch (selectedChild_str) {
        case "Shipping Address":
            trace("One was selected");
            break;
        case "Billing Address":
            trace("Two was selected");
            break;
        case "Payment":
            trace("Three was selected");
            break;
    }
};
my_acc.addEventListener("change", my_accListener);
```

Accordion.createChild()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
accordionInstance.createChild(className, instanceName[,
    initialProperties])
```

参数

classOrSymbolName 待实例化的 **UIObject** 的类的构造函数或链接名称（即指向待实例化的元件的引用）。类必须是 **UIObject** 或 **UIObject** 的子类，但大多数情况下是 **View** 对象或 **View** 的子类。

instanceName 新实例的实例名称。

initialProperties 指定新实例初始属性的可选参数。可使用以下属性：

- *label* 一个字符串，它指定新的子实例在其标题上使用的文本标签。
- *icon* 一个字符串，它指定库元件的链接标识符，子项将该元件用作其标题上的图标。

返回

指向 **UIObject**（新创建子项）的实例的引用。

说明

方法（从 **View** 继承）：为 **accordion** 创建子项。新创建的子项被添加到 **Accordion** 所拥有的子项列表的末尾。使用此方法可以将视图放置于 **Accordion** 内。创建的子项是 *classOrSymbolName* 参数中指定的类或影片剪辑元件的实例。可以使用 *label* 和 *icon* 属性为 *initialProperties* 参数中每个子项所关联的 **Accordion** 标题指定文本标签和图标。

在创建每一子项时，按创建顺序给各子项分配索引编号，并将 *numChildren* 属性加 1。

示例

首先在舞台上创建一个名为 *my_acc* 的 **Accordion** 实例。向库中添加一个元件，其链接标识符 *payIcon* 为子标题的图标。以下代码创建一个名为 *billing* 的子项（标签为“Payment”（付款）），它是 **View** 类的一个实例：

```
var child_obj:Object = my_acc.createChild(mx.core.View, "billing", {label:
    "Payment", icon: "payIcon"});
```

以下代码也创建作为 **View** 类的实例的子项，但它使用 *import* 引用该 **View** 类的构造函数：

```
import mx.core.View;
var child_obj:Object = my_acc.createChild(View, "billing", {label:
    "Payment", icon: "payIcon"});
```

或者，向库中添加一个影片剪辑元件，其链接标识符 *PaymentForm* 为 **Accordion** 子项，并且使用以下代码创建 *PaymentForm* 的实例（命名为 *billing*），作为 *my_acc* 的子项（这种方法适用于动态内容：向影片剪辑元件中加载动态内容，然后使该元件成为 **Accordion** 实例的子项）：

```
var child_obj:Object = my_acc.createChild("PaymentForm", "billing", {label:
    "Payment", icon: "payIcon"});
```

一个更复杂的示例情形为，在舞台上保留 **Accordion** 实例 `my_acc`。然后将一个 **Label** 组件和一个 **TextInput** 组件从“组件”面板拖到当前文档的库中（以使库中既具有 **TextInput** 元件，又具有 **Label** 元件）。将以下代码粘贴到主时间轴的第一帧中（替换以前示例的所有代码）。以下代码创建作为 **View** 类的实例的子项，命名为 `billing`，再将各子项添加到 `billing` 以便为表单提供标签和文本输入字段：

```
import mx.core.View;
import mx.controls.Label;
import mx.controls.TextInput;
var child_obj:Object = my_acc.createChild(View, "billing", {label:"Payment",
    icon: "payIcon"});
// 创建作为视图实例子项的标签。
var cardType_label:Object = child_obj.createChild(Label, "CardType_label",
    {_x:10, _y:50});
var cardNumber_label:Object = child_obj.createChild(Label,
    "CardNumber_label", {_x:10, _y:100});
// 创建作为视图实例子项的文本输入。
var cardTypeInput_ti:Object = child_obj.createChild(TextInput,
    "CardType_ti", {_x:150, _y:50});
var cardNumberInput_ti:Object = child_obj.createChild(TextInput,
    "CardNumber_ti", {_x:150, _y:100});
// 填充标签。
cardType_label.text = "Card Type";
cardNumber_label.text = "Card Number";
```

Accordion.createSegment()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
accordionInstance.createSegment(classOrSymbolName, instanceName[, label[,  
    icon]])
```


参数

classOrSymbolName 指向待实例化的 **UIObject** 的类构造函数的引用或待实例化的元件的链接名称。类必须是 **UIObject** 或 **UIObject** 的子类，但大多数情况下是 **View** 或 **View** 的子类。

instanceName 新实例的名称。

label 一个字符串，它指定新的子实例在其标题上使用的文本标签。此参数是可选的。

icon 一个字符串，它引用子项用作其标题上的图标的库元件的链接标识符。此参数是可选的。

返回

指向新创建的 **UIObject** 实例的引用。

说明

方法：为 **accordion** 创建子项。新创建的子项被添加到 **Accordion** 所拥有的子项列表的末尾。使用此方法可以将视图放置于 **Accordion** 内。创建的子项是 *classOrSymbolName* 参数中指定的类或影片剪辑元件的实例。可以使用 *label* 和 *icon* 参数为每个子项的关联 **accordion** 标题指定文本标签和图标。

`createSegment()` 方法与 `addChild()` 方法的不同之处在于：*label* 和 *icon* 是作为参数（而不是作为 *initialProperties* 参数的属性）直接传递的。

在创建每一子项时，按创建顺序给各子项分配索引编号，并将 `numChildren` 属性加 1。

示例

首先在舞台上创建一个名为 `my_acc` 的 **Accordion** 实例。向库中添加一个影片剪辑元件，其链接标识符 `PaymentForm` 为 **Accordion** 子项。然后，向库中添加一个元件，其链接标识符 `payIcon` 为子项标题的图标。以下示例创建 `PaymentForm` 影片剪辑元件的实例，命名为 `billing`，作为 `my_acc` 的最后一个子项。它在库中具有标题标签“**Payment**”（付款）和图标：

```
var child_obj:Object = my_acc.createSegment("PaymentForm", "billing",  
    "Payment", "payIcon");
```

以下代码创建作为 **View** 类的实例的子项：

```
var child_obj:Object = my_acc.createSegment(mx.core.View, "billing",  
    "Payment", "payIcon");
```

以下代码也创建作为 **View** 类的实例的子项，但它使用 `import` 引用该 **View** 类的构造函数：

```
import mx.core.View;  
var child_obj:Object = my_acc.createSegment(View, "billing", "Payment",  
    "payIcon");
```

将一个 **Label** 组件和一个 **TextInput** 组件从“组件”面板拖到当前文档的库中（以使库中既具有 **TextInput** 元件，又具有 **Label** 元件）。以下代码创建作为 **View** 类的实例的子项，命名为 **billing**，再将各子项添加到 **billing** 以便为表单提供标签和文本输入字段：

```
import mx.core.View;
import mx.controls.Label;
import mx.controls.TextInput;
var child_obj:Object = my_acc.createSegment(View, "billing", "Payment",
    "payIcon");
// 创建作为视图实例子项的标签。
var cardType_label:Object = child_obj.createChild(Label, "CardType_label",
    {_x:10, _y:50});
var cardNumber_label:Object = child_obj.createChild(Label,
    "CardNumber_label", {_x:10, _y:100});
// 创建作为视图实例子项的文本输入。
var cardTypeInput_ti:Object = child_obj.createChild(TextInput,
    "CardType_ti", {_x:150, _y:50});
var cardNumberInput_ti:Object = child_obj.createChild(TextInput,
    "CardNumber_ti", {_x:150, _y:100});
// 填充标签。
cardType_label.text = "Card Type";
cardNumber_label.text = "Card Number";
```

Accordion.destroyChildAt()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
accordionInstance.destroyChildAt(index)
```

参数

index 要破坏的 **accordion** 子项的索引编号。**accordion** 的每个子项均按其创建顺序获得了一个从零开始的索引编号。

返回

无。

说明

方法（从 **View** 继承）；破坏其中一个 **accordion** 子项。要破坏的子项是由其索引指定的，该索引将被传递到 *index* 参数中的方法。调用此方法将同时破坏相应的标题。

如果已破坏的子项处于选定状态，则会选择一个新的选定子项。如果有下一个子项，则会将其选定。如果没有下一个子项，则会选定前一个子项。如果没有前一个子项，则该选择是未定义的。



调用 `destroyChildAt()` 会将 `numChildren` 属性减 1。

示例

在选择第三个子项时，以下代码破坏 `my_acc` 的第一个子项：

```
import mx.core.View;

// 创建包含 View 类实例的子面板。
my_acc.createSegment(View, "myMainItem1", "Menu Item 1");
my_acc.createSegment(View, "myMainItem2", "Menu Item 2");
my_acc.createSegment(View, "myMainItem3", "Menu Item 3");

// 创建新的侦听器对象。
my_accListener = new Object();
my_accListener.change = function() {
    if ("myMainItem3"){
        my_acc.destroyChildAt(0);
    }
};

my_acc.addEventListener("change", my_accListener);
```

另请参见

[Accordion.createChild\(\)](#)

Accordion.getChildAt()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`accordionInstance.getChildAt(index)`

参数

index **Accordion** 子项的索引编号。**Accordion** 的每个子项均按其创建顺序获得了一个从零开始的索引。

返回

对位于指定索引位置的 **UIObject** 实例的引用。

说明

方法：返回指向位于指定索引位置的子项的引用。为每个 **Accordion** 子项都提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个子项为 **0**，第二个子项为 **1**，依此类推。

示例

以下代码获取对 `my_acc` 最后一个子项的引用，并将标签更改为 “**Last Child**”（最后一个子项）：

```
import mx.core.View;

// 创建包含 View 类实例的子面板。
my_acc.createSegment(View, "myMainItem1", "Menu Item 1");
my_acc.createSegment(View, "myMainItem2", "Menu Item 2");
my_acc.createSegment(View, "myMainItem3", "Menu Item 3");

// 获取对最后一个子对象的引用。
var lastChild_obj:Object = my_acc.getChildAt(my_acc.numChildren - 1);
// 更改对象的标签。
lastChild_obj.label = "Last Child";
```

Accordion.getHeaderAt()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
accordionInstance.getHeaderAt(index)
```

参数

index **Accordion** 标题的索引编号。**accordion** 的每个标题均会按其创建顺序获得一个从零开始的索引。

返回

对位于指定索引位置的 **UIObject** 实例的引用。

说明

方法；返回指向位于指定索引位置的标题的引用。为每一 **accordion** 标题都提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个标题为 **0**，第二个标题为 **1**，依此类推。

示例

以下代码获取对 `my_acc` 最后一个标题的引用，并在“输出”面板中显示标签：

```
import mx.core.View;

// 为要显示在 my_acc 对象中的每个表单创建子面板。
my_acc.createChild(View, "shippingAddress", {label: "Shipping Address"});
my_acc.createChild(View, "billingAddress", {label: "Billing Address"});
my_acc.createChild(View, "payment", {label: "Payment"});

var head3:Object = my_acc.getHeaderAt(2);
trace(head3.label);
```

Accordion.numChildren

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`accordionInstance.numChildren`

说明

属性（从 **View** 继承）；指示 **Accordion** 实例中的子项数（**UIObject** 类型）。标题不会被计作子项。

为每个 **Accordion** 子项提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个子项为 **0**，第二个子项为 **1**，依此类推。代码 `my_acc.numChild - 1` 始终引用添加到 **Accordion** 的最后一个子项。例如，如果某个 **Accordion** 中有 7 个子项，则最后一个子项的索引为 **6**。`numChildren` 属性不是从零开始的，因此 `my_acc.numChildren` 的值将为 7。7 减 1 的结果是 **6**，即最后一个子项的索引编号。

示例

以下代码使用 `numChildren` 来获取对 `my_acc` 最后一个子项的引用，并将标签更改为 “**Last Child**”（最后一个子项）：

```
import mx.core.View;

// 创建包含 View 类实例的子面板。
my_acc.createSegment(View, "myMainItem1", "Menu Item 1");
my_acc.createSegment(View, "myMainItem2", "Menu Item 2");
my_acc.createSegment(View, "myMainItem3", "Menu Item 3");

// 获取对最后一个子对象的引用。
var lastChild_obj:Object = my_acc.getChildAt(my_acc.numChildren - 1);
// 更改对象的标签。
lastChild_obj.label = "Last Child";
```

Accordion.selectedChild

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`accordionInstance.selectedChild`

说明

属性；如果存在一个或多个子项，则为选定子项（属于 `UIObject` 类型）；如果不存在子项，则为 `undefined`。

如果 **Accordion** 有子项，则代码 `accordionInstance.selectedChild` 与代码 `accordionInstance.getChildAt(accordionInstance.selectedIndex)` 等效。

将此属性设置为一个子项将导致 **Accordion** 开始转换动画以显示指定的子项。

如果更改 `selectedChild` 的值，则会同时更改 `selectedIndex` 的值。

如果 **Accordion** 有子项，则默认值为 `accordionInstance.getChildAt(0)`。如果 **Accordion** 没有子项，则默认值为 `undefined`。

示例

以下示例检测何时选定某个子项，并在每次选定某个标题时在“输出”面板中显示该子项的顺序：

```
// 创建新的侦听器对象。
var my_accListener:Object = new Object();
my_accListener.change = function() {
    trace("Changed to different view");
    // 将子面板的标签分配给变量
    var selectedChild_str:String = my_acc.selectedChild.label;
    // 基于所选的子项执行操作
    switch (selectedChild_str) {
        case "Shipping Address":
            trace("One was selected");
            break;
        case "Billing Address":
            trace("Two was selected");
            break;
        case "Payment":
            trace("Three was selected");
            break;
    }
};
my_acc.addEventListener("change", my_accListener);
```

另请参见

[Accordion.selectedIndex](#)

Accordion.selectedIndex

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`accordionInstance.selectedIndex`

说明

属性：在具有一个或多个子项的 **Accordion** 中，为选定子项从零开始的索引。对于没有子项视图的 **Accordion**，唯一的有效值为 `undefined`。

为每个 **accordion** 子项提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个子项为 **0**，第二个子项为 **1**，依此类推。`selectedIndex` 的有效值为 **0**、**1**、**2...**、**n - 1**，其中 **n** 是子项的数量。

将此属性设置为一个子项将导致 **Accordion** 开始转换动画以显示指定的子项。

如果更改 `selectedIndex` 的值，则会同时更改 `selectedChild` 的值。

示例

以下示例检测何时选定某个子项，并在每次选定某个标题时在“输出”面板中显示该子项的顺序：

```
// 创建新的侦听器对象。
var my_accListener:Object = new Object();
my_accListener.change = function() {
    trace("Changed to different view");
    // 将子面板的标签分配给变量。
    var selectedChild_num:Number = my_acc.selectedIndex;
    // 基于所选的子项执行操作。
    switch (selectedChild_num) {
        case 0:
            trace("One was selected");
            break;
        case 1:
            trace("Two was selected");
            break;
        case 2:
            trace("Three was selected");
            break;
    }
};
my_acc.addEventListener("change", my_accListener);
```

另请参见

[Accordion.numChildren](#)、[Accordion.selectedChild](#)

Alert 组件（仅限 Flash Professional）

Alert 组件使您能够显示一个窗口，该窗口向用户呈现一条消息和响应按钮。该窗口包含一个可填充文本的标题栏、一个可自定义的消息和若干可更改标签的按钮。**Alert** 窗口可以包含“是”、“否”、“确定”和“取消”按钮的任意组合，而且可以通过使用 `Alert.okLabel`、`Alert.yesLabel`、`Alert.noLabel` 和 `Alert.cancelLabel` 属性更改按钮的标签。您无法更改 **Alert** 窗口中按钮的顺序；按钮顺序始终为“确定”、“是”、“否”、“取消”。**Alert** 窗口在用户单击其中的任何一个按钮时关闭。

若要显示 **Alert** 窗口，请调用 `Alert.show()` 方法。为了成功调用该方法，**Alert** 组件必须位于库中。通过将 **Alert** 组件从“组件”面板拖到舞台上并将其删除，可将该组件添加到库中，但不在文档中显示。

Alert 组件的实时预览是一个空窗口。

将 **Alert** 组件添加到应用程序时，可以使用“辅助功能”面板使该组件的文本和按钮可由屏幕读取器访问。首先，添加以下代码行来启用辅助功能：

```
mx.accessibility.AlertAccImpl.enableAccessibility();
```



无论组件有多少实例，都只对组件启用一次辅助功能。

使用 Alert 组件（仅限 Flash Professional）

可在需要向用户通告某些内容时随时使用 **Alert** 组件。例如，您可能要在用户未正确填写表单、股票达到某个价位或用户在未保存会话的情况下退出应用程序时显示警告。

Alert 参数

Alert 组件没有创作参数。必须调用 **ActionScript** 的 `Alert.show()` 方法来显示 **Alert** 窗口。可以使用其它 **ActionScript** 属性来修改应用程序中的 **Alert** 窗口。有关更多信息，请参见第 67 页的“**Alert** 类（仅限 **Flash Professional**）”。

创建具有 Alert 组件的应用程序

以下过程解释了如何在创作时将 **Alert** 组件添加到应用程序。在本例中，**Alert** 组件将在股票达到某个价位时显示。

若要创建具有 **Alert** 组件的应用程序，请执行以下操作：

1. 将 **Alert** 组件从“组件”面板拖到当前文档的库中。

此操作将组件添加到库中，但不会在应用程序中显示。

2. 在“动作”面板中，在时间轴的第一帧中输入以下代码，以便为 `click` 事件定义事件处理函数：

```
import mx.controls.Alert;

// 定义警告确认后的动作。
var myClickHandler:Function = function (evt_obj:Object) {
    if (evt_obj.detail == Alert.OK) {
        trace("start stock app");
    }
};

// 显示警告对话框。
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |
    Alert.CANCEL, this, myClickHandler, "stockIcon", Alert.OK);
```

这段代码将创建带有“确定”和“取消”按钮的 **Alert** 窗口。当用户单击任一按钮时，**Flash** 将调用 `myClickHandler` 函数。在单击“确定”按钮时，`myClickHandler` 函数会指示 **Flash** 显示“start stock app”（启动 stock 应用程序）。

碎
碎

`Alert.show()` 方法包含一个可在 **Alert** 窗口中显示图标的可选参数（在本例中，为带有链接标识符“stockIcon”的图标）。要在您的测试示例中包含此图标，请创建一个名为 `stockIcon` 的元件，并在“链接属性”对话框或“创建新元件”对话框中将其设置为“为 **ActionScript** 导出”。`stockIcon` 元件的图形应当在元件的坐标系统中对齐到 (0,0)。

3. 选择“控制” > “测试影片”。

自定义 Alert 组件（仅限 Flash Professional）

Alert 组件将自己置于作为其“父”参数传递的组件的中心。该父项必须是 `UIComponent` 对象。如果父项是影片剪辑，您可以将剪辑注册为 `mx.core.View`，以使其继承自 `UIComponent`。

Alert 窗口会自动沿水平方向伸展，以便适合显示的消息文本或任何按钮。如果要显示大量的文本，请在文本中包含换行符。

Alert 组件不会响应 `setSize()` 方法。

对 Alert 组件使用样式

您可以设置样式属性以更改 Alert 组件的外观。如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

Alert 组件支持下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
<code>backgroundColor</code>	光晕和范例	背景色。“光晕”主题的默认颜色为白色，“范例”主题的默认颜色为 <code>OxEFEDEF</code> （浅灰）。
<code>borderStyle</code>	光晕和范例	Alert 组件使用 <code>RectBorder</code> 实例作为其边框并对在该类上定义的样式做出响应。有关更多信息，请参见第 985 页的“ RectBorder 类 ”。Alert 组件具有一个特定于组件的 <code>borderStyle</code> 设置，该设置在“光晕”主题中为“alert”，在“范例”主题中为“outset”。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题的默认值为 <code>Ox0B333C</code> ，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>Ox848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。

样式	主题	说明
fontFamily	光晕和 范例	文本的字体名称。默认值为 "_sans"。
fontSize	光晕和 范例	字体的磅值。默认值为 10。
fontStyle	光晕和 范例	字体样式: "normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和 范例	字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textAlign	光晕和 范例	文本对齐方式: "left"、"right" 或 "center"。默认值为 "left"。
textDecoration	光晕和 范例	文本修饰: "none" 或 "underline"。默认值为 "none"。
textIndent	光晕和 范例	表示文本缩进的数字。默认值为 0。

Alert 组件包含三种不同的文本类别。设置 **Alert** 组件自身的文本属性会为所有这三种类别提供默认值, 如下所示:

```
import mx.controls.Alert;
_global.styles.Alert.setStyle("color", 0x000099);
Alert.show("This is a test alert", "Title");
```

Alert 组件提供静态属性 (对 **CSSStyleDeclaration** 实例的引用) 用以单独设置一个类别的文本样式。

静态属性	影响的文本
buttonStyleDeclaration	按钮
messageStyleDeclaration	消息
titleStyleDeclaration	标题

以下示例演示如何将 **Alert** 组件的标题设置为斜体：


```
import mx.controls.Alert;
import mx.styles.CSSStyleDeclaration;

var titleStyles = new CSSStyleDeclaration();
titleStyles.setStyle("fontWeight", "bold");
titleStyles.setStyle("fontStyle", "italic");

Alert.titleStyleDeclaration = titleStyles;

Alert.show("Name is a required field", "Validation Error");
```

默认的标题样式声明将 `fontWeight` 设置为 `"bold"`。覆盖 `titleStyleDeclaration` 属性时，此默认设置也被覆盖。因此如果需要此设置，必须将 `fontWeight` 显式设置为 `"bold"`。

 在 **Alert** 组件上设置的文本样式通过样式继承为其组件提供默认的文本样式。有关更多信息，请参见《使用组件》中的“设置容器上的继承样式”。

对 **Alert** 组件使用外观

Alert 组件扩展了 **Window** 组件，并使用其标题背景的外观作为标题背景、使用 **RectBorder** 类的实例作为边框，使用 **Button** 的外观作为按钮的可视状态。要在创作时为按钮和标题栏设置外观，请修改 **Flash UI Components 2/Themes/MMDefault/Window Assets/Elements/TitleBackground** 和 **Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin** 中的元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。默认情况下，边框和背景由 **RectBorder** 类提供。有关设置 **RectBorder** 类外观的信息，请参见第 985 页的“**RectBorder** 类”。

Alert 组件使用以下外观属性动态地设置按钮和标题栏的外观：

属性	说明	默认值
buttonUp	按钮的弹起状态。	ButtonSkin
buttonUpEmphasized	默认按钮的弹起状态。	ButtonSkin
buttonDown	按钮的按下状态。	ButtonSkin
buttonDownEmphasized	默认按钮的按下状态。	ButtonSkin
buttonOver	按钮的滑过状态。	ButtonSkin
buttonOverEmphasized	默认按钮的滑过状态。	ButtonSkin
titleBackground	窗口的标题栏。	TitleBackground

将 **Alert** 组件的标题设置为自定义影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 通过选择 “插入” > “新建元件” 创建一个新元件。
3. 将名称设置为 `TitleBackground`。
4. 如果高级视图未显示出来，则单击 “高级” 按钮。
5. 选择 “为 **ActionScript** 导出”。
6. 标识符将自动填写为 `TitleBackground`。
7. 将 **AS 2.0** 类设置为 `mx.skins.SkinElement`。

`SkinElement` 是一个简单类，可用于所有自身不提供 **ActionScript** 实现方法的外观元素。该类为 **Macromedia Component Architecture** 第 2 版的组件提供移动和调整大小功能。

8. 确保 “在第一帧导出” 处于选中状态。
9. 单击 “确定”。
10. 打开新元件以进行编辑。
11. 使用绘画工具创建一个黑色线条和红色填充的方框。
12. 将边框样式设置为极细。
13. 设置此方框（包括边框），使它位于 (0,0) 并且宽度为 100，高度为 22。

Alert 组件将根据需要设置适当的外观宽度，但会使用现有的高度作为标题的高度。

14. 单击 “返回” 按钮返回主时间轴。
15. 将 **Alert** 组件拖到舞台上，然后将其删除。

此操作会将 **Alert** 组件添加到库中，并使其在运行时可用。

16. 向主时间轴添加 **ActionScript** 代码，创建一个范例 **Alert** 实例。

```
import mx.controls.Alert;  
Alert.show("This is a skinned Alert component","Title");
```

17. 选择 “控制” > “测试影片”。

Alert 类（仅限 Flash Professional）

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > View > ScrollView > [Window 组件](#) > Alert

ActionScript 类名称 mx.controls.Alert

若要使用 Alert 组件，您需要将 Alert 组件拖到舞台上并将其删除，以便组件位于文档库中但不会在应用程序中显示。然后，您需要调用 Alert.show() 来显示 Alert 窗口。可以将参数传递到 Alert.show()，这些参数将消息、标题栏和按钮添加到 Alert 窗口中。

由于 ActionScript 是异步的，因此 Alert 组件不是模块化的，也就是说 ActionScript 的代码行会在调用 Alert.show() 之后立即运行。必须添加侦听器来处理在用户单击按钮时广播的 click 事件，然后在播放事件后继续执行代码。

提醒

在模块化的操作环境（如 Microsoft Windows）中，用户采取操作（如单击按钮）后，对 Alert.show() 的调用才会返回。

若要了解有关 Alert 类的更多信息，请参见第 1347 页的“Window 组件”和第 915 页的“PopUpManager 类”。

Alert 类的方法摘要

下表列出了 Alert 类的方法。

方法	说明
Alert.show()	使用可选参数创建 Alert 窗口。

从 UIObject 类继承的方法

下表列出了 Alert 类从 UIObject 类继承的方法。

方法	说明
UIObject.createClassObject()	创建指定类的对象。
UIObject.createObject()	创建对象的子对象。
UIObject.destroyObject()	破坏组件实例。
UIObject.doLater()	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
UIObject.getStyle()	从样式声明或对象获取样式属性。
UIObject.invalidate()	标记对象使其在到达下一个帧间隔时进行重绘。
UIObject.move()	将对象移动到要求的位置。

方法	说明
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

从 UIComponent 类继承的方法

下表列出了 `Alert` 类从 `UIComponent` 类继承的方法。

方法	说明
<code>UIComponent.setFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

从 Window 类继承的方法

下表列出了 `Alert` 类从 `Window` 类继承的方法。

方法	说明
<code>Window.deletePopUp()</code>	删除由 <code>PopUpManager.createPopUp()</code> 创建的窗口实例。

Alert 类的属性摘要

下表列出了 `Alert` 类的属性。

属性	说明
<code>Alert.buttonHeight</code>	每个按钮的高度（以像素为单位）。默认值是 22。
<code>Alert.buttonWidth</code>	每个按钮的宽度（以像素为单位）。默认值是 100。
<code>Alert.CANCEL</code>	一个十六进制常数值，它指示是否应在 <code>Alert</code> 窗口中显示“取消”按钮。
<code>Alert.cancelLabel</code>	“取消”按钮的标签文本。
<code>Alert.NO</code>	一个十六进制常数值，它指示是否应在 <code>Alert</code> 窗口中显示“否”按钮。
<code>Alert.noLabel</code>	“否”按钮的标签文本。
<code>Alert.OK</code>	一个十六进制常数值，它指示是否应在 <code>Alert</code> 窗口中显示“确定”按钮。
<code>Alert.okLabel</code>	“确定”按钮的标签文本。

属性	说明
<code>Alert.YES</code>	一个十六进制常数值，它指示是否应在 <code>Alert</code> 窗口中显示“是”按钮。
<code>Alert.yesLabel</code>	“是”按钮的标签文本。

从 `UIObject` 类继承的属性

下表列出了 `Alert` 类从 `UIObject` 类继承的属性。从 `Alert` 对象调用这些属性时，请使用 `Alert.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	只读。对象的底边缘位置（相对于其父对象的底边缘）。
<code>UIObject.height</code>	只读；对象的高度，以像素为单位。
<code>UIObject.left</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.right</code>	只读；对象的右边缘位置（相对于其父对象的右边缘）。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 <code>x</code> 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 <code>y</code> 方向上的缩放因子。
<code>UIObject.top</code>	只读；对象上边缘的位置（相对于其父对象）。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (<code>true</code>) 还是不可见的 (<code>false</code>)。
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

从 `UIComponent` 类继承的属性

下表列出了 `Alert` 类从 `UIComponent` 类继承的属性。从 `Alert` 对象调用这些属性时，请使用 `Alert.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 <code>Tab</code> 键顺序。

从 Window 类继承的属性

下表列出了 `Alert` 类从 `Window` 类继承的属性。

属性	说明
<code>Window.closeButton</code>	指示标题栏上是 (true) 否 (false) 包含 “关闭” 按钮。
<code>Window.content</code>	对窗口的内容（根影片剪辑）的引用。
<code>Window.contentPath</code>	设置要在窗口中显示的内容的名称。
<code>Window.title</code>	标题栏中显示的文本。
<code>Window.titleStyleDeclaration</code>	设置标题栏中文本格式的样式声明。

Alert 类的事件摘要

下表列出了 `Alert` 类的一个事件。

事件	说明
<code>Alert.click</code>	在单击 <code>Alert</code> 窗口中的某个按钮时广播。

从 UIObject 类继承的事件

下表列出了 `Alert` 类从 `UIObject` 类继承的事件。从 `Alert` 对象调用这些事件时，请使用 `Alert.eventName` 的形式。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

从 UIComponent 类继承的事件

下表列出了 `Alert` 类从 `UIComponent` 类继承的事件。从 `Alert` 对象调用这些事件时，请使用 `Alert.eventName` 的形式。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

从 Window 类继承的事件

下表列出了 `Alert` 类从 `Window` 类继承的事件。

事件	说明
<code>Window.click</code>	单击（松开）关闭按钮时广播。
<code>Window.complete</code>	创建窗口时广播。

Alert.buttonHeight

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`Alert.buttonHeight`

说明

属性（类）；更改按钮高度的类（静态）属性。默认值是 22。

示例

在库中已具有一个 `Alert` 组件的情况下，本示例调整按钮的大小：

```
import mx.controls.Alert;

// 调整按钮大小。
Alert.buttonHeight = 50;
Alert.buttonWidth = 150;
```

```
// 显示警告对话框。  
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |  
    Alert.CANCEL);
```

另请参见

[Alert.buttonWidth](#)

Alert.buttonWidth

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`Alert.buttonWidth`

说明

属性（类）：更改按钮宽度的类（静态）属性。默认值是 100。

示例

在库中已具有 **Alert** 组件的情况下，向主时间轴的第一帧中添加此 **ActionScript** 代码以调整按钮的大小：

```
import mx.controls.Alert;  
  
// 调整按钮大小。  
Alert.buttonHeight = 50;  
Alert.buttonWidth = 150;  
  
// 显示警告对话框。  
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |  
    Alert.CANCEL);
```

另请参见

[Alert.buttonHeight](#)

Alert.CANCEL

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

Alert.CANCEL

说明

属性（常数）；十六进制常数值为 **0x8** 的属性。此属性可用于 `Alert.show()` 方法的 *flags* 或 *defaultButton* 参数。用作 *flags* 参数的值时，此属性指示应在 **Alert** 窗口中显示“取消”按钮。用作 *defaultButton* 参数的值时，“取消”按钮具有初始焦点，并在用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时触发。如果用户按 **Tab** 键切换到另一个按钮，则切换到的按钮在用户按下 **Enter** 键时触发。

示例

以下示例使用 `Alert.CANCEL` 和 `Alert.OK` 作为 *flags* 参数的值，并显示一个带有“确定”按钮和“取消”按钮的 **Alert** 组件：

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.OK |
    Alert.CANCEL, this);
```

Alert.cancelLabel

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

Alert.cancelLabel

说明

属性（类）：指明“取消”按钮上的标签文本的类（静态）属性。

示例

以下范例将“取消”按钮的标签设置为“cancellation”：

```
Alert.cancelLabel = "cancellation";
```

Alert.click

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
var clickHandler:Object = function(eventObject:Object) {  
    // 此处插入您的代码。  
}  
Alert.show(message[, title[, flags[, parent[, clickHandler[, icon[,  
    defaultButton]]]]]])
```

说明

事件：在单击“确定”、“是”、“否”或“取消”按钮时广播至注册的侦听器。

第 2 版组件使用调度程序 / 侦听器事件模型。**Alert** 组件会在它的其中一个按钮被单击时发出 `click` 事件，该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）进行处理。您需要调用 `Alert.show()` 方法并将处理函数的名称作为参数传递给它。在单击 **Alert** 窗口中的某个按钮时，即会调用侦听器。

该事件发生时，它会自动将一个事件对象 (*eventObject*) 传递给处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Alert.click` 事件的事件对象具有一个附加的 `detail` 属性，根据所单击的按钮的不同，其值为 `Alert.OK`、`Alert.CANCEL`、`Alert.YES` 或 `Alert.NO`。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

示例

在库中已具有 **Alert** 组件的情况下，向主时间轴的第一帧中添加此 **ActionScript** 代码以创建名为 `myClickHandler` 的事件处理函数。该事件处理函数作为其第五个参数传递给 `Alert.show()` 方法。`myClickHandler` 在 `evt` 参数中捕获该事件对象。然后，事件对象的 `detail` 属性将用在 `trace` 语句内，以便将所单击按钮的名称（`Alert.OK` 或 `Alert.CANCEL`）发送到“输出”面板。

```
import mx.controls.Alert;

// 定义按钮动作。
var myClickHandler:Function = function (evt_obj:Object) {
    switch (evt_obj.detail) {
        case Alert.OK :
            trace("You clicked: " + Alert.okLabel);
            break;
        case Alert.CANCEL :
            trace("You clicked: " + Alert.cancelLabel);
            break;
    }
};

// 显示对话框。
Alert.show("This is a test of errors", "Error", Alert.OK | Alert.CANCEL,
    this, myClickHandler);
```

Alert.NO

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

`Alert.NO`

说明

属性（常数）：十六进制常数值为 `0x2` 的属性。此属性可用于 `Alert.show()` 方法的 `flags` 或 `defaultButton` 参数。用作 `flags` 参数的值时，此属性指示应在 **Alert** 窗口中显示“否”按钮。用作 `defaultButton` 参数的值时，“取消”按钮具有初始焦点，并在用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时触发。如果用户按 **Tab** 键切换到另一个按钮，则切换到的按钮在用户按下 **Enter** 键时触发。

示例

以下示例使用 `Alert.NO` 和 `Alert.YES` 作为 *flags* 参数的值，并显示一个带有“否”按钮和“是”按钮的 **Alert** 组件：

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.NO |
    Alert.YES, this);
```

Alert.noLabel

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
Alert.noLabel
```

说明

属性（类）；指明“否”按钮上的标签文本的类（静态）属性。

示例

以下范例将“否”按钮的标签设置为“nyet”：

```
Alert.noLabel = "nyet";
```

Alert.NONMODAL

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
Alert.NONMODAL
```


说明

属性（常数）：十六进制常数值为 **0x8000** 的属性。此属性可用于 `Alert.show()` 方法的 *flags* 参数。此属性指示 **Alert** 窗口应该为非模态，从而允许用户与所显示的窗口下面的按钮和实例交互。默认情况下，用 `Alert.show()` 生成的窗口为模态，这意味着除了当前打开的窗口外，用户不能进行任何单击操作。

示例

以下示例在舞台上显示两个 **Button** 组件实例。单击其中一个按钮可以打开一个模态窗口，这可防止关闭 **Alert** 窗口前用户单击其它按钮。第二个按钮可以打开一个非模态窗口，从而允许用户继续单击当前打开的非模态 **Alert** 窗口下面的按钮。若要测试此示例，请将 **Alert** 组件和 **Button** 组件的实例都添加到当前文档的库中，然后向主时间轴的第一帧添加以下代码：

```
import mx.controls.Alert;

this.createClassObject(mx.controls.Button, "modal_button", 10, {_x:10,
    _y:10});
this.createClassObject(mx.controls.Button, "nonmodal_button", 20, {_x:120,
    _y:10});

modal_button.label = "modal";
modal_button.addEventListener("click", modalListener);
function modalListener(evt_obj:Object):Void {
    var a:Alert = Alert.show("This is a modal Alert window", "Alert Test",
        Alert.OK, this);
    a.move(100, 100);
}

nonmodal_button.label = "nonmodal";
nonmodal_button.addEventListener("click", nonmodalListener);
function nonmodalListener(evt_obj:Object):Void {
    var a:MovieClip = Alert.show("This is a nonmodal Alert window", "Alert
        Test", Alert.OK | Alert.NONMODAL, this);
    a.move(100, 100);
}
```

Alert.OK

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

Alert.OK

说明

属性（常数）；十六进制常数值为 **0x4** 的属性。此属性可用于 `Alert.show()` 方法的 *flags* 或 *defaultButton* 参数。用作 *flags* 参数的值时，此属性指示应在 **Alert** 窗口中显示“确定”按钮。用作 *defaultButton* 参数的值时，“确定”按钮具有初始焦点，并在用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时触发。如果用户按 **Tab** 键切换到另一个按钮，则切换到的按钮在用户按下 **Enter** 键时触发。

示例

以下示例使用 `Alert.OK` 和 `Alert.CANCEL` 作为 *flags* 参数的值，并显示一个带有“确定”按钮和“取消”按钮的 **Alert** 组件：

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.OK |
    Alert.CANCEL, this);
```

Alert.okLabel

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

Alert.okLabel

说明

属性（类）；指明“确定”按钮上的标签文本的类（静态）属性。

示例

以下示例将“确定”按钮的标签设置为“okay”：

```
Alert.okLabel = "okay";
```

Alert.show()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
Alert.show(message[, title[, flags[, parent[, clickHandler[, icon[,  
    defaultButton]]]]]])
```

参数

message 要显示的消息。

title **Alert** 标题栏中的文本。此参数可选；如果省略此参数，则标题栏为空。

flags 可选参数，指示要显示在 **Alert** 窗口中的按钮。默认值为 `Alert.OK`，它将显示“确定”按钮。在使用多个值时，请用 `|` 字符分隔各个值。使用以下的一个或多个值：`Alert.OK`、`Alert.CANCEL`、`Alert.YES`、`Alert.NO`。

您还可以使用 `Alert.NONMODAL` 指明 **Alert** 窗口为非模态。非模式窗口允许用户与应用程序中的其它窗口交互。

parent **Alert** 组件的父窗口。**Alert** 窗口会将自己置于父窗口的中心。使用值 `null` 或 `undefined` 来指定 `_root` 时间轴。父窗口必须是 **UIComponent** 类的子类，即作为 **UIComponent** 子类的另一个 **Flash** 组件，或者是作为 **UIComponent** 子类的自定义窗口（有关更多信息，请参见《学习 Flash 中的 ActionScript 2.0》中的“关于继承”。此参数是可选的。

clickHandler 单击按钮时广播的 `click` 事件的处理函数。除了标准的 `click` 事件对象属性外，还有另一个 `detail` 属性，该属性包含所单击按钮的标志值（`Alert.OK`、`Alert.CANCEL`、`Alert.YES`、`Alert.NO`）。此处理函数可以是函数或对象。有关更多信息，请参见《使用组件》中的“使用侦听器处理事件”。

icon 表示要用作图标（显示在警告文本的左边）的库元件的链接标识符的字符串。此参数是可选的。

defaultButton 指示具有初始焦点并在用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时被单击的按钮。如果用户按 **Tab** 键切换到另一个按钮，则切换到的按钮在按下 **Enter** 键时触发。

此参数可以是下列值之一：Alert.OK、Alert.CANCEL、Alert.YES、Alert.NO。

返回

所创建的 **Alert** 实例。

说明

方法（类）：显示带有消息、可选标题、可选按钮和可选图标的 **Alert** 窗口的类（静态）方法。**Alert** 的标题显示在窗口的顶部，并且靠左对齐。图标显示在消息文本的左边。按钮显示在消息文本和图标下方的中间位置。

示例

以下代码显示一个带有“确定”按钮的 **Alert** 模式窗口的简单范例。

```
mx.controls.Alert.show("Hello, world!");
```

以下代码定义一个 **click** 处理函数，它将有关所单击按钮的消息发送到“输出”面板。（要使此代码显示警告，库中必须具有一个 **Alert** 组件；要向库中添加该组件，请将其拖到舞台上，然后将其删除）：

```
import mx.controls.Alert;

// 定义按钮动作。
var myClickHandler:Function = function (evt_obj:Object) {
    if (evt_obj.detail == Alert.OK) {
        trace(Alert.okLabel);
    } else if (evt_obj.detail == Alert.CANCEL) {
        trace(Alert.cancelLabel);
    }
};

// 显示对话框。
var dialog_obj:Object = Alert.show("Test Alert", "Test", Alert.OK |
    Alert.CANCEL, null, myClickHandler, "testIcon", Alert.OK);
```

Alert.YES

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

Alert.YES

说明

属性（常数）；十六进制常数值为 `0x1` 的属性。此属性可用于 `Alert.show()` 方法的 *flags* 或 *defaultButton* 参数。用作 *flags* 参数的值时，此属性指示应在 **Alert** 窗口中显示“是”按钮。用作 *defaultButton* 参数的值时，“是”按钮具有初始焦点，并在用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时触发。如果用户按 **Tab** 键切换到另一个按钮，则切换到的按钮在用户按下 **Enter** 键时触发。

示例

以下示例使用 `Alert.NO` 和 `Alert.YES` 作为 *flags* 参数的值，并显示一个带有“否”按钮和“是”按钮的 **Alert** 组件：

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.NO |
    Alert.YES, this);
```

Alert.yesLabel

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
Alert.yesLabel
```

说明

属性（类）；指示“是”按钮上的标签文本的类（静态）属性。

示例

以下范例将“确定”按钮的标签设置为“da”：

```
Alert.yesLabel = "da";
```

Button 组件

Button 组件是一个可调整大小的矩形用户界面按钮。可以给按钮添加一个自定义图标。也可以将按钮的行为从按下改为切换。在单击切换按钮后，它将保持按下状态，直到再次单击时才会返回到弹起状态。

可以在应用程序中启用或者禁用按钮。在禁用状态下，按钮不接收鼠标或键盘输入。如果单击或者切换到某个按钮，处于启用状态的就会接收焦点。当一个 **Button** 实例具有焦点时，您可以使用下列按键来控制它：

键	说明
Shift+Tab	将焦点移到前一个对象。
空格键	按下或释放组件并触发 click 事件。
Tab	将焦点移到下一个对象。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“[FocusManager 类](#)”或“创建自定义焦点导航”。

每个 **Button** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。然而，在实时预览中，自定义图标在舞台上由一个灰色方块表示。

在将 **Button** 组件添加到应用程序时，您可以使用“辅助功能”面板，以使其可由屏幕阅读器访问。首先，您必须添加以下代码行：

```
mx.accessibility.ButtonAccImpl.enableAccessibility();
```

不管一个组件有多少实例，都只对其启用一次辅助功能。

使用 Button 组件

按钮是任何表单或 Web 应用程序的一个基础部分。每当您需要让用户启动一个事件时，都可以使用按钮实现。例如，大多数表单都有“提交”按钮。您也可以给演示文稿添加“前一个”和“后一个”按钮。

要给按钮添加一个图标，您需要选择或创建一个影片剪辑或图形元件以用作图标。元件应注册在 (0,0) 以在按钮上获得适当的布局。在“库”面板中选择图标元件，并从“库”弹出菜单中打开“链接”对话框，然后输入一个链接标识符。该值是为“属性”检查器或“组件”检查器中的图标参数输入的值。也可以为 `Button.icon` `ActionScript` 属性输入此值。



如果图标比按钮大，它将会延伸到按钮的边框外。

若要将应用程序中的某个按钮指定为默认普通按钮（当用户按 **Enter** 键时接收单击事件的按钮），请使用 `FocusManager.defaultPushButton`。

Button 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 **Button** 组件实例设置以下创作参数：

icon 为按钮添加自定义图标。该值是库中影片剪辑或图形元件的链接标识符；没有默认值。

label 设置按钮上文本的值；默认值是“Button”。

labelPlacement 确定按钮上的标签文本相对于图标的方向。该参数可以是下列四个值之一：left、right、top 或 bottom；默认值为 right。有关更多信息，请参见 `Button.labelPlacement`。

selected 如果 **toggle** 参数的值是 true，则该参数指定按钮是处于按下状态 (true) 还是释放状态 (false)。默认值为 false。

toggle 将按钮转变为切换开关。如果值为 true，则按钮在单击后保持按下状态，并在再次单击时返回到弹起状态。如果值为 false，则按钮行为与一般按钮相同。默认值为 false。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **Button** 组件实例设置以下附加参数：

enabled 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 true。

visible 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。



`minHeight` 和 `minWidth` 属性由内部的大小调整例程使用。这些属性在 `UIObject` 中定义，并可以根据需要被不同的组件覆盖。如果您为应用程序创建自定义布局管理器，则可以使用这些属性。否则，在“组件”检查器中设置这些属性将不具有明显效果。

您可以编写 `ActionScript`，以便使用 **Button** 组件的属性、方法和事件来控制该组件的这些选项和其它选项。有关更多信息，请参见第 94 页的“**Button 类**”。

创建具有 Button 组件的应用程序

以下过程解释了如何在创作时将 **Button** 组件添加到应用程序。在本示例中，单击按钮时会显示一条消息。

创建具有 Button 组件的应用程序：

1. 将 **Button** 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，执行以下操作：
 - 输入实例名称 **“my_button”**。
 - 为 **label** 参数输入 **Click me**。
 - 为 **icon** 参数输入 **BtnIcon**。
要使用图标，必须将库中一个带有链接标识符的影片剪辑或图形元件用作图标参数。
在本示例中，链接标识符是 **BtnIcon**。
 - 将 **toggle** 属性设置为 **true**。
3. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
function clicked(){
    trace("You clicked the button!");
}
my_button.addEventListener("click", clicked);
```

最后一行代码调用“click”事件的 **clicked** 事件处理函数。此操作会一起使用[第 462 页](#)的“**EventDispatcher.addEventListener()**”方法和自定义的函数来处理该事件。
4. 选择“控制”>“测试影片”。
5. 单击该按钮时，Flash 会显示消息“**You clicked the button!**”（您单击了该按钮！）。

使用 ActionScript 代码创建 Button：

1. 将 **Button** 组件从“组件”面板拖到当前文档的库中。
此操作将组件添加到库中，但不会在应用程序中显示它。
2. 在主时间轴的第一帧中，向“动作”面板中添加以下 **ActionScript** 代码以创建一个 **Button** 实例：

```
this.createClassObject(mx.controls.Button, "my_button", 10, {label:"Click me"});
my_button.move(20, 20);
```

使用 **UIObject.createClassObject()** 方法创建名为 **my_button** 的 **Button** 实例，并指定一个标签属性。然后，代码使用 **UIObject.move()** 方法来定位该按钮。

3. 现在，添加以下 **ActionScript** 来创建一个事件侦听器和一个事件处理函数：

```
function clicked() {  
    trace("You clicked the button!");  
}  
my_button.addEventListener("click", clicked);
```

此操作会一起使用第 462 页的 **“[EventDispatcher.addEventListener\(\)](#)”** 方法和自定义的函数来处理该事件。

4. 选择“控制” > “测试影片”。
5. 单击该按钮时，Flash 会显示消息 **“You clicked the button!”**（您单击了该按钮！）。
在与其它组件一起使用 **Button** 组件时，可以创建更为复杂的事件处理函数。在本示例中，“click”事件导致 **Accordion** 组件更改面板的显示。

与另一个组件一起使用 **Button** 事件：

1. 将 **Button** 组件从“组件”面板拖到当前文档的库中。
此操作将组件添加到库中，但不会在应用程序中显示它。
2. 将 **Accordion** 组件从“组件”面板拖到当前文档的库中。
3. 在主时间轴的第一帧中，向“动作”面板中添加以下 **ActionScript** 代码以创建一个 **Button** 实例：

```
this.createClassObject(mx.containers.Accordion, "my_acc", 0);  
my_acc.move(10, 40);  
my_acc.createChild(mx.core.View, "panelOne", {label: "Panel One"});  
my_acc.createChild(mx.core.View, "panelTwo", {label: "Panel Two"});  
  
this.createClassObject(mx.controls.Button, "panelOne_button", 10,  
    {label: "Panel One"});  
panelOne_button.move(10, 10);  
this.createClassObject(mx.controls.Button, "panelTwo_button", 20,  
    {label: "Panel Two"});  
panelTwo_button.move(120, 10);
```

此过程使用 **UIObject.createClassObject()** 方法来创建 **Button** 和 **Accordion** 实例。然后，代码使用 **UIObject.move()** 方法来定位这些实例。

4. 现在，添加以下 **ActionScript** 来创建事件侦听器 and 事件处理函数：

```
// 创建按钮事件的处理函数。  
function changePanel(evt_obj:Object):Void {  
    // 根据所选按钮更改 Accordion 视图。  
    switch (evt_obj.target._name) {  
        case "panelOne_button" :  
            my_acc.selectedIndex = 0;  
            break;  
        case "panelTwo_button" :  
            my_acc.selectedIndex = 1;  
            break;
```

- ```
 }
 }

 // 为按钮添加侦听器。
 panelOne_button.addEventListener("click", changePanel);
 panelTwo_button.addEventListener("click", changePanel);

 此过程会一起使用 EventDispatcher.addEventListener() 方法和自定义的函数来处理
 该事件。
```
5. 选择 “控制” > “测试影片”。
  6. 单击按钮时，`Accordion` 会更改当前面板。

## 自定义 Button 组件

在创作过程中和运行时，可以在水平和垂直方向上改变 `Button` 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，应使用 `setSize()` 方法（请参见 `UIObject.setSize()`）或 `Button` 类的任何适用的属性和方法（请参见第 94 页的“`Button` 类”）。调整按钮大小不会更改图标或标签的大小。

`Button` 实例的边框是不可见的，它同时也指定了该实例的点击区域。如果您增加实例的大小，也就增加了点击区的大小。如果边框太小而无法容纳标签，标签会被裁剪以适合边框。如果图标比按钮大，它将会延伸到按钮的边框外。

## 对 Button 组件使用样式

您可以设置样式属性来更改按钮实例的外观。如果样式属性的名称以“`Color`”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

`Button` 组件支持下列样式：

| 样式                           | 主题 | 说明                                                                                                                                                                                                            |
|------------------------------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>themeColor</code>      | 光晕 | 组件的基本配色方案。可能的值包括“ <code>haloGreen</code> ”、“ <code>haloBlue</code> ”和“ <code>haloOrange</code> ”。默认值为“ <code>haloGreen</code> ”。                                                                              |
| <code>backgroundColor</code> | 范例 | 背景色。默认值为 <code>OxEFEFEBEF</code> （浅灰）。<br>在“光晕”主题中，按钮弹起时使用 <code>OxF8F8F8</code> （极浅灰）作为其背景颜色，按钮按下时使用 <code>themeColor</code> 作为其背景颜色。在“光晕”主题中设置按钮的外观只能修改按钮弹起时的背景颜色。请参见第 89 页的“对 <code>Button</code> 组件使用外观”。 |

| 样式             | 主题    | 说明                                                                                                                                                             |
|----------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| borderStyle    | 范例    | 在“范例”主题中，Button 组件使用 RectBorder 实例作为其边框并对在该类上定义的样式做出响应。请参见 <a href="#">第 985 页的“RectBorder 类”</a> 。<br>在“光晕”主题中，Button 组件使用自定义的圆角边框，该边框的颜色除 themeColor 外都不能修改。 |
| color          | 光晕和范例 | 文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。                                                                                                                       |
| disabledColor  | 光晕和范例 | 组件禁用时的文本颜色。默认值为 0x848384（深灰）。                                                                                                                                  |
| embedFonts     | 光晕和范例 | 一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。                |
| fontFamily     | 光晕和范例 | 文本的字体名称。默认值为 “_sans”。                                                                                                                                          |
| fontSize       | 光晕和范例 | 字体的磅值。默认值为 10。                                                                                                                                                 |
| fontStyle      | 光晕和范例 | 字体样式：“normal”或“italic”。默认值为“normal”。                                                                                                                           |
| fontWeight     | 光晕和范例 | 字体粗细：“none”或“bold”。默认值为“none”。在调用 setStyle() 期间，所有组件还可以接受值“normal”来代替“none”，但随后对 getStyle() 的调用将返回“none”。                                                      |
| textDecoration | 光晕和范例 | 文本修饰：“none”或“underline”。默认值为“none”。                                                                                                                            |

## 对 Button 组件使用外观

Button 组件包含 32 种不同的外观，可以将这些外观自定义为与处于 16 种不同状态的边框和图标相对应。要在创作时设置 Button 组件的外观，请用所需的图形创建新的影片剪辑元件，并使用 `ActionScript` 设置该元件的链接标识符。（有关更多信息，请参见第 91 页的“使用 `ActionScript` 绘制 Button 外观”。）

“光晕”和“范例”主题提供的 Button 外观的默认实现使用 `ActionScript` 绘图 API 来绘制按钮的状态，并使用与一个 `ActionScript` 类关联的单个影片剪辑元件来提供 Button 组件的所有外观。

因为按钮有很多状态，并且每种状态都有一个边框和图标，所以 Button 组件有许多外观。Button 实例的状态由四个属性和用户交互进行控制。以下属性影响外观：

| 属性                      | 说明                                                                                                                                                                                          |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>emphasized</code> | 提供 Button 实例的两种不同外观，通常用于加亮显示某个按钮（如表单中的默认按钮）。                                                                                                                                                |
| <code>enabled</code>    | 显示按钮是否允许用户交互。                                                                                                                                                                               |
| <code>toggle</code>     | 提供选中和未选中值，并使用不同的外观表示当前值。对于 <code>toggle</code> 属性设置为 <code>false</code> 的 Button 实例，将使用 <code>false</code> 外观。当 <code>toggle</code> 属性为 <code>true</code> 时，外观取决于 <code>selected</code> 属性。 |
| <code>selected</code>   | 当 <code>toggle</code> 属性设置为 <code>true</code> 时，此属性确定 Button 是否被选中（ <code>true</code> 或 <code>false</code> ）。该值通过使用不同的外观进行标识，这也是默认情况下在屏幕上描绘此值的唯一方式。                                         |

如果按钮已启用，当鼠标指针在它上方移动时，它会显示其“指针经过”状态。在按下按钮时，按钮将接收输入焦点并显示其“按下”状态。当松开鼠标后，按钮又返回其“指针经过”状态。如果在按下鼠标时指针移离按钮，按钮就会返回到其初始状态并保留输入焦点。如果 `toggle` 参数设置为 `true`，则在鼠标在它上方松开前，按钮的状态不会改变。

如果按钮被禁用，不管用户进行什么交互操作，它都会显示其禁用状态。

Button 组件支持以下外观属性：

| 属性                             | 说明        |
|--------------------------------|-----------|
| <code>falseUpSkin</code>       | 弹起（正常）状态。 |
| <code>falseDownSkin</code>     | 按下状态。     |
| <code>falseOverSkin</code>     | 指针经过状态。   |
| <code>falseDisabledSkin</code> | 禁用状态。     |
| <code>trueUpSkin</code>        | 切换状态。     |
| <code>trueDownSkin</code>      | 按下切换状态。   |

| 属性                          | 说明               |
|-----------------------------|------------------|
| trueOverSkin                | 指针经过切换状态。        |
| trueDisabledSkin            | 禁用切换状态。          |
| falseUpSkinEmphasized       | 强调按钮的弹起（正常）状态。   |
| falseDownSkinEmphasized     | 强调按钮的按下状态。       |
| falseOverSkinEmphasized     | 强调按钮的指针经过状态。     |
| falseDisabledSkinEmphasized | 强调按钮的禁用状态。       |
| trueUpSkinEmphasized        | 强调按钮的切换状态。       |
| trueDownSkinEmphasized      | 强调按钮的按下切换状态。     |
| trueOverSkinEmphasized      | 强调按钮的指针经过切换状态。   |
| trueDisabledSkinEmphasized  | 强调按钮的禁用切换状态。     |
| falseUpIcon                 | 图标弹起状态。          |
| falseDownIcon               | 图标按下状态。          |
| falseOverIcon               | 图标指针经过状态。        |
| falseDisabledIcon           | 图标禁用状态。          |
| trueUpIcon                  | 图标切换状态。          |
| trueOverIcon                | 图标指针经过切换状态。      |
| trueDownIcon                | 图标按下切换状态。        |
| trueDisabledIcon            | 图标禁用切换状态。        |
| falseUpIconEmphasized       | 强调按钮的图标弹起状态。     |
| falseDownIconEmphasized     | 强调按钮的图标按下状态。     |
| falseOverIconEmphasized     | 强调按钮的图标指针经过状态。   |
| falseDisabledIconEmphasized | 强调按钮的图标禁用状态。     |
| trueUpIconEmphasized        | 强调按钮的图标切换状态。     |
| trueOverIconEmphasized      | 强调按钮的图标指针经过切换状态。 |
| trueDownIconEmphasized      | 强调按钮的图标按下切换状态。   |
| trueDisabledIconEmphasized  | 强调按钮的图标禁用切换状态。   |

所有以“Skin”结尾的外观属性的默认值均为 ButtonSkin，所有以“Icon”结尾的外观属性的默认值均为 undefined。以“Skin”为后缀的属性提供背景和边框，而以“Icon”为后缀的属性提供小图标。

除图标外观外，**Button** 组件还支持标准的 `icon` 属性。标准属性与样式属性之间的区别在于通过样式属性可以为单个状态设置图标，而使用标准属性只能设置一个图标，而且该图标会应用到所有状态。如果同时设置了一个 **Button** 实例的 `icon` 属性和图标样式属性，则该实例可能不会按预期的方式运行。

若要查看说明每种外观应何时使用的交互式演示影片，请参见《使用组件》帮助。

## 使用 ActionScript 绘制 Button 外观

“光晕”主题和“范例”主题中的默认外观针对所有状态均使用同一外观元素，并通过 **ActionScript** 绘制实际图形。“光晕”实现使用 **RectBorder** 类的扩展以及某些自定义绘图代码来绘制状态。“范例”实现使用与“光晕”主题相同的外观和 **ActionScript** 类，并使用 **ActionScript** 代码设置不同的属性来更改 **Button** 的外观。

要创建用作外观并提供不同状态的 **ActionScript** 类，外观可以读取外观的 `borderStyle` 样式属性和父级的 `emphasized` 属性以确定状态。下表显示了为每种外观设置的边框样式：

| 属性                            | 边框样式                       |
|-------------------------------|----------------------------|
| <code>falseUpSkin</code>      | <code>falseup</code>       |
| <code>falseDownSkin</code>    | <code>falsedown</code>     |
| <code>falseOverSkin</code>    | <code>falserollover</code> |
| <code>falseDisabled</code>    | <code>falsedisabled</code> |
| <code>trueUpSkin</code>       | <code>trueup</code>        |
| <code>trueDownSkin</code>     | <code>truedown</code>      |
| <code>trueOverSkin</code>     | <code>truerollover</code>  |
| <code>trueDisabledSkin</code> | <code>truedisabled</code>  |

### 创建 ActionScript 自定义的 Button 外观：

**1.** 新建一个 **ActionScript** 类文件。

对于本示例，将文件命名为 `RedGreenBlueSkin.as`。

**2.** 将以下 **ActionScript** 复制到文件中：

```
import mx.skins.RectBorder;
import mx.core.ext.UIObjectExtensions;

class RedGreenBlueSkin extends RectBorder
{
 static var symbolName:String = "RedGreenBlueSkin";
 static var symbolOwner:Object = RedGreenBlueSkin;

 function size():Void
 {
 var c:Number; // 颜色
```

```

var borderStyle:String = getStyle("borderStyle");

switch (borderStyle) {
 case "falseup":
 case "falserollover":
 case "falsedisabled":
 c = 0x7777FF;
 break;
 case "falsedown":
 c = 0x77FF77;
 break;
 case "trueup":
 case "truedown":
 case "truerollover":
 case "truedisabled":
 c = 0xFF7777;
 break;
}

clear();
var thickness = _parent.emphasized ? 2 : 0;
lineStyle(thickness, 0, 100);
beginFill(c, 100);
drawRect(0, 0, __width, __height);
endFill();
}

// 外观所需内容。
static function classConstruct():Boolean
{
 UIObjectExtensions.Extensions();
 _global.skinRegistry["ButtonSkin"] = true;
 return true;
}
static var classConstructed:Boolean = classConstruct();
static var UIObjectExtensionsDependency = UIObjectExtensions;
}

```

此类根据边框样式创建方框：为假弹起、滑过及禁用状态创建蓝色框；为正常按下状态创建绿色框；为展开子项创建红色框。正常情况下将绘制极细边框；若是强调按钮，则绘制粗边框。

3. 保存该文件。
4. 创建一个新的 FLA 文件并将其与 AS 文件保存在同一个文件夹中。
5. 通过选择“插入”>“新建元件”创建一个新元件。
6. 将名称设置为 ButtonSkin。
7. 如果高级视图未显示出来，则单击“高级”按钮。



8. 选择“为 ActionScript 导出”。
- 标识符将自动填写为 `ButtonSkin`。
9. 将 AS 2.0 类设置为 `RedGreenBlueSkin`。
10. 确保“在第一帧导出”处于选中状态，然后单击“确定”。
11. 将 `Button` 组件拖动到舞台上。
12. 选择“控制”>“测试影片”。

## 使用影片剪辑自定义 `Button` 外观

上面的示例演示了如何使用 `ActionScript` 类自定义 `Button` 外观，“光晕”主题和“范例”主题中提供的外观都使用此方法。但是，因为该示例使用简单的着色框，所以在本例中将不同影片剪辑元件用作外观较为简单。

### 创建 `Button` 外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 通过选择“插入”>“新建元件”创建一个新元件。
3. 将名称设置为 `RedButtonSkin`。
4. 如果高级视图未显示出来，则单击“高级”按钮。
5. 选择“为 ActionScript 导出”。
- 标识符将自动填写为 `RedButtonSkin`。
6. 将 AS 2.0 类设置为 `mx.skins.SkinElement`。
7. 确保“在第一帧导出”处于选中状态，然后单击“确定”。
8. 打开新元件以进行编辑。
9. 使用绘画工具创建一个黑色线条和红色填充的方框。
10. 将边框样式设置为极细。
11. 设置方框（包括边框），使它位于 (0,0) 并且宽度和高度皆为 100。
- `SkinElement` 类将适当调整内容的大小。
12. 重复步骤 2-11 创建绿色和蓝色外观，根据颜色相应命名。
13. 单击“返回”按钮返回主时间轴。
14. 将一个 `Button` 组件拖到舞台上。
15. 将 `toggled` 属性值设置为 `true`，以便看到所有三个外观。

16. 将以下 **ActionScript** 代码复制到 “动作” 面板中，同时选中 **Button** 实例。

```
onClipEvent(initialize) {
 falseUpSkin = "BlueButtonSkin";
 falseDownSkin = "GreenButtonSkin";
 falseOverSkin = "BlueButtonSkin";
 falseDisabledSkin = "BlueButtonSkin";
 trueUpSkin = "RedButtonSkin";
 trueDownSkin = "RedButtonSkin";
 trueOverSkin = "RedButtonSkin";
 trueDisabledSkin = "RedButtonSkin";
}
```

17. 选择 “控制” > “测试影片”。

## Button 类

继承 **MovieClip** > **UIObject** 类 > **UIComponent** 类 > **SimpleButton** 类 > **Button**

**ActionScript** 类名称 **mx.controls.Button**

**Button** 类的属性使您可以在运行时执行以下操作：给按钮添加图标、创建文本标签以及指示按钮用作普通按钮还是切换开关。

使用 **ActionScript** 设置 **Button** 类的属性将会覆盖在 “属性” 检查器或 “组件” 检查器中设置的同名参数。

**Button** 组件使用焦点管理器覆盖默认的 **Flash Player** 焦点矩形，并绘制一个带圆角的自定义焦点矩形。有关更多信息，请参见 《使用组件》中的 “创建自定义焦点导航”。

每个组件类都有一个 **version** 属性，该属性是一个类属性。类属性只能用于该类本身。**version** 属性会返回一个字符串，该字符串指示组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.controls.Button.version);
```



代码 `trace(myButtonInstance.version);` 返回 `undefined`。

**Button** 组件类与 **ActionScript** 内置 **Button** 对象不同。

# Button 类的方法摘要

没有 Button 类专用的方法。

## 从 UIObject 类继承的方法

下表列出了 Button 类从 UIObject 类继承的方法。从 Button 对象调用这些方法时，请使用 `buttonInstance.methodName` 的形式。

| 方法                                        | 说明                               |
|-------------------------------------------|----------------------------------|
| <code>UIObject.createClassObject()</code> | 创建指定类的对象。                        |
| <code>UIObject.createObject()</code>      | 创建对象的子对象。                        |
| <code>UIObject.destroyObject()</code>     | 破坏组件实例。                          |
| <code>UIObject.doLater()</code>           | 在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。 |
| <code>UIObject.getStyle()</code>          | 从样式声明或对象获取样式属性。                  |
| <code>UIObject.invalidate()</code>        | 标记对象使其在到达下一个帧间隔时进行重绘。            |
| <code>UIObject.move()</code>              | 将对象移动到要求的位置。                     |
| <code>UIObject.redraw()</code>            | 迫使对象有效以便能在当前帧中绘制。                |
| <code>UIObject.setSize()</code>           | 将对象调整为所要求的大小。                    |
| <code>UIObject.setSkin()</code>           | 设置对象的外观。                         |
| <code>UIObject.setStyle()</code>          | 设置样式声明或对象的样式属性。                  |

## 从 UIComponent 类继承的方法

下表列出了 Button 类从 UIComponent 类继承的方法。从 Button 对象调用这些方法时，请使用 `buttonInstance.methodName` 的形式。

| 方法                                  | 说明             |
|-------------------------------------|----------------|
| <code>UIComponent.getFocus()</code> | 返回对具有焦点的对象的引用。 |
| <code>UIComponent.setFocus()</code> | 将焦点设置到组件实例中。   |

# Button 类的属性摘要

下表列出了 Button 类的属性。

| 属性                                 | 说明              |
|------------------------------------|-----------------|
| <code>Button.icon</code>           | 指定按钮实例的图标。      |
| <code>Button.label</code>          | 指定在按钮上显示的文本。    |
| <code>Button.labelPlacement</code> | 指定标签文本相对于图标的方向。 |

## 从 SimpleButton 类继承的属性

下表列出了 Button 类从 SimpleButton 类继承的属性。访问这些属性时，请使用 `buttonInstance.propertyName` 的形式。

| 属性                                                   | 说明                                                                                                 |
|------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <code>SimpleButton.emphasized</code>                 | 指明按钮是否具有默认普通按钮外观。                                                                                  |
| <code>SimpleButton.emphasizedStyleDeclaration</code> | 当 <code>emphasized</code> 属性设置为 <code>true</code> 时的样式声明。                                          |
| <code>SimpleButton.selected</code>                   | 一个布尔值，它指示按钮处于选中状态 ( <code>true</code> ) 还是未处于选中状态 ( <code>false</code> )。默认值为 <code>false</code> 。 |
| <code>SimpleButton.toggle</code>                     | 一个布尔值，它指示按钮的行为与切换开关相同 ( <code>true</code> ) 还是不同 ( <code>false</code> )。默认值为 <code>false</code> 。  |

## 从 UIObject 类继承的属性

下表列出了 Button 类从 UIObject 类继承的属性。从 Button 对象访问这些属性时，请使用 `buttonInstance.propertyName` 的形式。

| 属性                            | 说明                                                                   |
|-------------------------------|----------------------------------------------------------------------|
| <code>UIObject.bottom</code>  | 只读；对象的底边缘位置（相对于其父对象的底边缘）。                                            |
| <code>UIObject.height</code>  | 只读；对象的高度，以像素为单位。                                                     |
| <code>UIObject.left</code>    | 只读；对象的左边缘（以像素为单位）。                                                   |
| <code>UIObject.right</code>   | 只读；对象的右边缘位置（相对于其父对象的右边缘）。                                            |
| <code>UIObject.scaleX</code>  | 一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。                                       |
| <code>UIObject.scaleY</code>  | 一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。                                       |
| <code>UIObject.top</code>     | 只读；对象上边缘的位置（相对于其父对象）。                                                |
| <code>UIObject.visible</code> | 一个布尔值，它指示对象是可见的 ( <code>true</code> ) 还是不可见的 ( <code>false</code> )。 |
| <code>UIObject.width</code>   | 只读；对象的宽度，以像素为单位。                                                     |

| 属性                      | 说明                 |
|-------------------------|--------------------|
| <code>UIObject.x</code> | 只读；对象的左边缘（以像素为单位）。 |
| <code>UIObject.y</code> | 只读；对象的上边缘（以像素为单位）。 |

## 从 UIComponent 类继承的属性

下表列出了 `Button` 类从 `UIComponent` 类继承的属性。从 `Button` 对象访问这些属性时，请使用 `buttonInstance.propertyName` 的形式。

| 属性                                | 说明                     |
|-----------------------------------|------------------------|
| <code>UIComponent.enabled</code>  | 指示组件是否可以接收焦点和输入。       |
| <code>UIComponent.tabIndex</code> | 一个数字，指示文档中组件的 Tab 键顺序。 |

## Button 类的事件摘要

没有 `Button` 类专用的事件。

## 从 SimpleButton 类继承的事件

下表列出了 `Button` 类从 `SimpleButton` 类继承的事件。

| 属性                              | 说明         |
|---------------------------------|------------|
| <code>SimpleButton.click</code> | 单击一个按钮时广播。 |

## 从 UIObject 类继承的事件

下表列出了 `Button` 类从 `UIObject` 类继承的事件。

| 事件                           | 说明                 |
|------------------------------|--------------------|
| <code>UIObject.draw</code>   | 当对象将要绘制它的图形时进行广播。  |
| <code>UIObject.hide</code>   | 在对象的状态从可见变为不可见时广播。 |
| <code>UIObject.load</code>   | 创建子对象时广播。          |
| <code>UIObject.move</code>   | 移动了对象时广播。          |
| <code>UIObject.resize</code> | 在调整对象大小后广播。        |
| <code>UIObject.reveal</code> | 在对象的状态从不可见变为可见时广播。 |
| <code>UIObject.unload</code> | 卸载子对象时广播。          |

## 从 UIComponent 类继承的事件

下表列出了 **Button** 类从 **UIComponent** 类继承的事件。

| 事件                                | 说明            |
|-----------------------------------|---------------|
| <code>UIComponent.focusIn</code>  | 当对象收到焦点时进行广播。 |
| <code>UIComponent.focusOut</code> | 当对象失去焦点时进行广播。 |
| <code>UIComponent.keyDown</code>  | 当按下按键时进行广播。   |
| <code>UIComponent.keyUp</code>    | 当松开按键时进行广播。   |

# Button.icon

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

`buttonInstance.icon`


### 说明

属性：一个字符串，指定库中要用作按钮实例图标元件的链接标识符。图标可以是一个影片剪辑元件，也可以是具有左上角注册点的图形元件。如果图标太大而超出了按钮的大小，您必须调整按钮的大小；按钮和图标的大小都不会自动调整。如果图标比按钮大，图标会延展到按钮的边界外。

要创建一个自定义图标，请创建一个影片剪辑或者图形元件。在元件编辑模式下，选择舞台上的该元件，并在“属性”检查器中的“X”和“Y”框内都输入 0。在“库”面板中，选择影片剪辑并从“库”弹出菜单中选择“链接”。选择“为 **ActionScript** 导出”，并在“标识符”文本框内输入一个标识符。

默认值是一个空字符串 ("")，指示没有图标。

使用 `labelPlacement` 属性来设置图标相对于按钮的位置。

 图标不显示在 Flash 的舞台上。必须选择“控制”>“测试影片”才能看到图标。

## 示例

在舞台上具有一个实例名称为 `my_button` 的按钮的情况下，以下代码将“库”面板中链接标识符为 `happiness` 的影片剪辑分配给 **Button** 实例作为图标：

```
my_button.icon = "happiness";
```

也可以完全在 **ActionScript** 中使用 `UIObject.createClassObject()` 方法来创建按钮并分配该图标（仍然必须已经为按钮创建了链接标识符为 `happiness` 的图标）。首先将 **Button** 组件从“组件”面板拖到当前文档的库中，这样该组件将出现在库中，而不显示在舞台上。然后，在主时间轴的第一帧中，添加以下 **ActionScript** 代码：

```
this.createClassObject(mx.controls.Button, "my_button", 1, {icon:
 "happiness"});
```

## 另请参见

[Button.labelPlacement](#)

# Button.label

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*buttonInstance.label*

## 说明

属性；指定按钮实例的文本标签。默认情况下，标签显示在按钮的中央。调用此方法将会覆盖在“属性”检查器或“组件”检查器中指定的标签创作参数。默认值为 `"Button"`。

## 示例

在舞台上具有一个实例名称为 `my_button` 的按钮的情况下，以下代码将标签设置为 **“Test Button”**：

```
my_button.label = "Test Button";
```

也可以完全在 **ActionScript** 中使用 `UIObject.createClassObject()` 方法来创建按钮并分配该标签。首先将 **Button** 组件从“组件”面板拖到当前文档的库中，这样该组件将出现在库中，而不显示在舞台上。然后，在主时间轴的第一帧中，添加以下 **ActionScript** 代码：

```
this.createClassObject(mx.controls.Button, "my_button", 1, {label: "Test
 Button"});
```

另请参见

[Button.labelPlacement](#)

# Button.labelPlacement

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*buttonInstance*.labelPlacement

## 说明

属性；设置标签相对于图标的位置。默认值为 "right"。下面是四种可能的值，图标和标签始终在按钮的边界区域内垂直居中和水平居中：

- "right" 标签设在图标的右侧。
- "left" 标签设在图标的左侧。
- "bottom" 标签设在图标的下方。
- "top" 标签设在图标的上方。

## 示例

在舞台上具有一个实例名称为 my\_button 的按钮，并且“库”面板中具有一个链接标识符为 happiness 的元件的情况下，以下代码将标签对齐方式设置为图标的左侧：

```
my_button.icon = "happiness";
my_button.label = "Test Button";
my_button.labelPlacement = "left";
```

也可以完全在 **ActionScript** 中使用 `UIObject.createClassObject()` 方法来创建按钮并设置对齐方式。首先将 **Button** 组件从“组件”面板拖到当前文档的库中，使该组件出现在库中，而不显示在舞台上。然后，在主时间轴的第一帧中，添加以下 **ActionScript** 代码：

```
this.createClassObject(mx.controls.Button, "my_button", 1, {label: "Test
Button", icon: "happiness", labelPlacement: "left"});
```



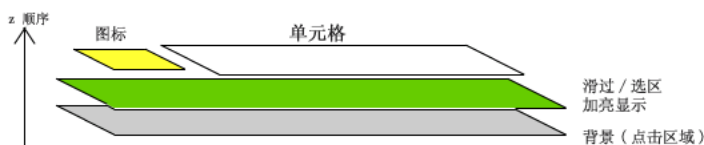
CellRendererer API 是一组属性和方法，基于列表的组件（List、DataGrid、Tree、Menu 和 ComboBox）使用它们来处理和显示每一行的自定义单元格内容。此自定义单元格可以包含预先建立的组件（如 CheckBox 组件）或您创建的任何类。

## 了解 List 类

要使用 CellRendererer API，您需要深入了解 List 类。DataGrid、Tree、Menu 和 ComboBox 组件是 List 类的扩展，因此，了解 List 类使您能够同时了解这些组件。

### 关于 List 组件的构成

List 组件由行构成。这些行显示滑过和选区加亮显示，用作行选区的点击状态，并在滚动中扮演重要的角色。除了选区加亮显示和图标（如节点图标和 Tree 组件的展开箭头）之外，行还包含一个单元格（或者，如果是 DataGrid 组件，则包含多个单元格）。在默认情况下，这些单元格是实现 CellRendererer API 的 TextField 对象。但是，您可以让 List 组件使用不同的组件类作为每一行的单元格。唯一的要求是该类必须实现 List 组件用于与单元格通信的 CellRendererer API。



List 或 DataGrid 组件中行的堆叠顺序

提醒

如果单元格具有按钮事件处理函数（onPress 等），则背景点击区域可能不会接收触发事件所必需的输入。

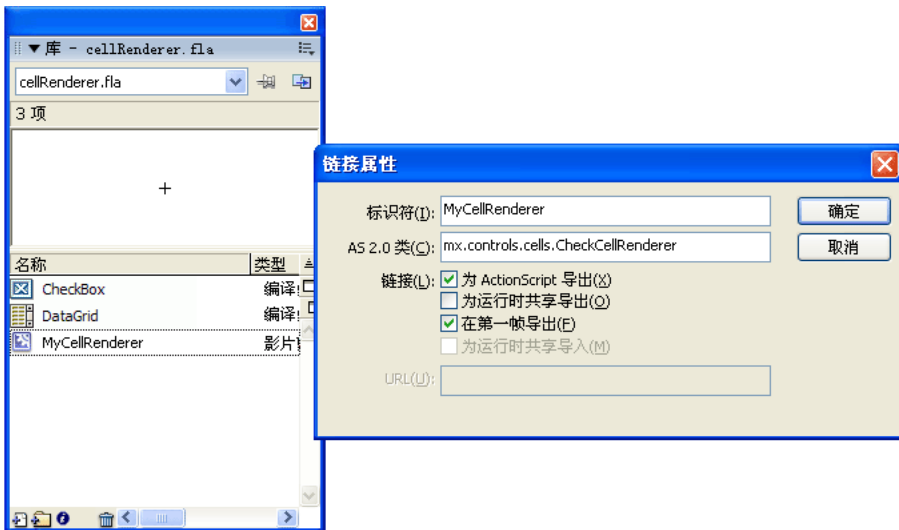
## 关于 List 组件的滚动行为

List 类使用一种相当复杂的算法进行滚动。列表只会列出它一次能显示的最多行数，超出 rowCount 属性值的项根本不会获得行。在列表滚动时，它会将所有行上下移动（取决于滚动方向）。然后，列表将回收滚出视图的行；它会重新初始化这些行，并将它们用作正在滚动到视图的新行。为此，列表将旧行的值设置为视图中的新项目，然后将旧行移动到新项目滚动到视图中时所在的位置。

考虑到这种滚动行为，您无法期望只使用单元格代表一个值。重复使用行意味着单元格渲染器必须知道当它被设置为新值时如何完全重置其状态。例如，如果单元格渲染器创建了图标来显示某个项目，在使用它渲染另一项目时，可能需要移除该图标。假设单元格渲染器是一种随着时间的改变将填充许多项目值的容器，并且它必须知道如何使本身从显示某个值完全转换到显示另一个值。事实上，单元格甚至还应该知道如何正确地渲染未定义的项目，这可能意味着删除单元格中的所有旧内容。

## 使用 CellRenderer API

必须编写一个包含四个方法（`CellRenderer.getPreferredHeight()`、`CellRenderer.getPreferredWidth()`、`CellRenderer.setSize()` 和 `CellRenderer.setValue()`）的类，基于列表的组件使用该类与单元格通信（如果该类是 `UIObject` 的扩展类，则可以使用 `size()` 来代替 `CellRenderer.setSize()`）。在 Flash 应用程序中，必须在影片剪辑元件的“链接属性”对话框中的“AS 2.0 类”文本框中指定该类。



您可以在一个示例文件中查看实现单元格渲染器 API 的 `CheckCellRenderer` 类：该文件位于 `First Run/classes/mx/controls/cells` 中。此外，还可以查看 `DataGrid` 组件文档（包括第 238 页的“`DataGrid` 性能策略”）来获取有关 `CellRenderer` 的信息。

系统将为单元格自动指定两个方法和一个属性（`CellRenderer.getCellIndex()`、`CellRenderer.getDataLabel()` 和 `CellRenderer.listOwner`），以便允许它与基于列表的组件通信。例如，假设单元格包含一个复选框，选中该复选框时会有一行被选中。单元格渲染器需要引用包含它的基于列表的组件，才能调用组件的 `selectedIndex` 属性。同时，单元格需要知道它当前正在呈现的项索引，以便能够将 `selectedIndex` 设置为正确的编号；单元格可以使用 `CellRenderer.listOwner` 和 `CellRenderer.getCellIndex()` 达到此目的。您不需要实现这些 `ActionScript` 元素；在将单元格放到基于列表的组件内时，单元格将自动接收这些元素。

## 单元格渲染器简单示例

本节介绍在一个单元格中显示多行文本的单元格渲染器的示例。

以下教程演示如何创建一个在 `DataGrid` 组件的单元格中显示多行文本的单元格渲染器类。

完成的文件 `MultiLineCell.as` 和 `CellRenderer_tutorial.fla` 位于 [www.macromedia.com/go/component\\_samples](http://www.macromedia.com/go/component_samples) 中。

### 创建 `MultiLineCell` 单元格渲染器类

单元格渲染器类必须实现以下方法：

- `CellRenderer.getPreferredHeight()`
- `CellRenderer.getPreferredWidth()`

仅 `Menu` 组件或 `DataGrid` 标题需要 `CellRenderer.getPreferredWidth()` 方法；其它情况下，请按照本示例所示的方法从代码中注释掉该方法。

- `CellRenderer.setSize()`

如果单元格渲染器类是 `UIObject` 的扩展类，请按照本示例所示改为使用 `implement size()`。

- `CellRenderer.setValue()`

单元格渲染器类还必须声明从 `List` 类接收的方法和属性：

- `CellRenderer.getCellIndex()`
- `CellRenderer.getDataLabel()`
- `CellRenderer.listOwner`

下列步骤说明如何创建一个名为 **MultiLineCell.as** 的 ActionScript 2.0 单元格渲染器类文件，并将该文件链接到新 Flash 文档中的新影片剪辑元件。这样，可以将 DataGrid 组件添加到 Flash 文档库中。在第一帧上，添加 ActionScript，以动态创建 DataGrid 并将 MultiLineCell 类指定为该网格某一列的单元格渲染器：

若要创建 multiLineCell 单元格渲染器类，请执行以下操作：

1. 在 Flash 中，选择“文件”>“新建”>“ActionScript 文件”（不是“Flash 文档”）。将该文档另存为 **MultiLineCell.as**。
2. 在 MultiLineCell.as 中输入以下代码：

```
// ActionScript 2.0 class.
class MultiLineCell extends mx.core.UIComponent
{
 private var multiLineLabel; // 用于显示文本的标签。
 private var owner; // 包含此单元格的行。
 private var listOwner; // 包含此单元格的列表、数据网格或树。

 // 单元格高度相对于行总高度的偏移量以及首选的单元格宽度。
 private static var PREFERRED_HEIGHT_OFFSET = 4;
 private static var PREFERRED_WIDTH = 100;
 // 起始深度。
 private var startDepth:Number = 1;

 // 构造函数。应为空。
 public function MultiLineCell()
 {
 }

 /* UIObject 期望您通过实例化初始化期间可能需要的所有影片剪辑资源来填充
 createChildren。在本例中，我们创建的是一个标签 */
 public function createChildren():Void
 {
 // createLabel 方法是 UIObject 的一种很有用的方法，并且是一种
 // 在组件中创建标签的便利方法。
 var c = multiLineLabel = this.createLabel("multiLineLabel",
startDepth);
 // 将该标签的样式链接到网格的样式
 c.styleName = listOwner;
 c.selectable = false;
 c.tabEnabled = false;
 c.background = false;
 c.border = false;
 c.multiline = true;
 c.wordWrap = true;
 }

 public function size():Void
 {
 }
```

```

/* 通过扩展用于导入 UIObject 的 UICComponent, 将自动获取 setSize, 但 UICComponent
期望您实现 size()。假设现在为您设置了 __width 和 __height。您将要扩展单元格以适应整
个 rowHeight。rowHeight 本身是要在其中呈现单元格的列表类型组件的属性。由于我们需要
rowHeight 适合两行的高度, 因此在使用此 cellRenderer 类创建该列表类型组件时, 请确保
将其 rowHeight 属性设置为足够大, 以便在该组件中可以呈现两行文本。*/

/*__width 和 __height 是 getter/setter .width 和 .height 的基础变量。*/
var c = multiLineLabel;
c.setSize(__width, __height);
}

// 提供单元格的首选高度。继承的方法。
public function getPreferredHeight():Number
{
/* 为单元格赋予一个引用行的属性 “owner”。首选的做法总是让单元格占用行的大部分高度。在本例
中, 我们将使单元格略小一些。*/
return owner.__height - PREFERRED_HEIGHT_OFFSET;
}

// 由 owner 调用以设置单元格中的值。继承的方法。
public function setValue(suggestedValue:String, item:Object,
selected:Boolean):Void
{
/* 如果未定义 item 参数, 则不应在单元格中呈现任何内容, 因此将标签设置为不可见。注意: 对于
滚动列表类型组件 (如滚动数据网格), 在单元格刚滚动出视线之外时就使其为空, 然后再次重用该
单元格, 并为其设置产生滚动动画效果的新值。出于此原因, 您不能指望任一单元格始终包含要显示
的数据或包含同一值。*/
if (item!=undefined){
multiLineLabel.text._visible = false;
}
multiLineLabel.text = suggestedValue;
}
// 函数 getPreferredWidth :: 仅用于菜单和数据网格标题
// 函数 getCellIndex :: 在此单元格渲染器中不使用
// 函数 getDataLabel :: 在此单元格渲染器中不使用
}

```

## 创建应用程序以测试 MultiLineCell 单元格渲染器类

在下列步骤中，将创建 DataGrid 实例并实现 MultiLineCell 类。

创建一个应用程序，该应用程序包含一个使用 MultiLineCell 单元格渲染器类的 DataGrid 组件：

1. 在 Flash 中，选择“文件”>“新建”>“Flash 文档”。
2. 选择“文件”>“另存为”，将该文件命名为 **cellRender\_tutorial.fla**，然后将其保存到 MultiLineCell.as 文件所在的同一文件夹中。
3. 若要创建新的 movieClip 元件以链接到 MultiLineCell 类，请选择“插入”>“新建元件”。
4. 单击“创建新元件”对话框右下角的“高级”按钮以启用更多选项。

当您处于“创建新元件”对话框的基本模式时，可以看到“高级”按钮。如果看不到“高级”按钮，您很可能已经在该对话框的“高级”视图下。

5. 在“名称”文本框中键入 **MultiLineCell**。  
“类型”的默认值为“影片剪辑”。保留“影片剪辑”的选中状态。
6. 在“链接”部分单击“为 ActionScript 导出”复选框。  
启用此选项将允许您在运行期间将此元件的实例动态地附加到 Flash 文档。“标识符”文本框将自动显示 MultiLineCell。
7. 将“ActionScript 2.0 类”设置为 MultiLineCell（以匹配先前创建的 MultiLineCell 单元格渲染器类的名称）。
8. 最后，启用“在第一帧导出”复选框，然后单击“确定”以应用更改并关闭对话框。



如果以后需要修改“MultiLineCell 影片剪辑”元件的链接属性，可以在文档库中右击该元件，并从所显示的菜单中选择“属性”或“链接”。

9. 将 DataGrid 组件从“组件”面板拖到库中。  
在下一步中将通过 ActionScript 动态创建 DataGrid 实例。
10. 选择主时间轴上的第一帧（确保您尚未处于 MultiLineCell 影片剪辑编辑模式下）。
11. 在第一帧的“动作”面板中，输入以下代码，以动态创建 DataGrid、向 DataGrid 分配数据并分配新的单元格渲染器类：

```
// 创建一个新的 DataGrid 组件实例
this.createClassObject(mx.controls.DataGrid, "myGrid_dg", 1);

// 为包含四列数据的数据网格构建一个数据提供程序。
myDP = new Array();
var aLongString:String = "An example of a cell renderer class that
 displays a multiple line TextField";
myDP.addItem({firstName:"Winston", lastName:"Elstad", note:aLongString,
 item:100});
```

```

myDP.addItem({firstName:"Ric", lastName:"Dietrich", note:aLongString,
 item:101});
myDP.addItem({firstName:"Ewing", lastName:"Canepa", note:aLongString,
 item:102});
myDP.addItem({firstName:"Kevin", lastName:"Wade", note:aLongString,
 item:103});
myDP.addItem({firstName:"Kimberly", lastName:"Dietrich",
 note:aLongString, item:104});
myDP.addItem({firstName:"AJ", lastName:"Bilow", note:aLongString,
 item:105});
myDP.addItem({firstName:"Chuck", lastName:"Yushan", note:aLongString,
 item:106});
myDP.addItem({firstName:"John", lastName:"Roo", note:aLongString,
 item:107});

/* 将该数据提供程序分配到 DataGrid 以填充该网格。注意：必须在应用 cellRenderer 之前执
 行此操作。*/
myGrid_dg.dataProvider = myDP;

/* 设置某些基本的网格属性。注意：该数据网格的行高应反映您期望在 MultiLineCell 单元格渲
 染器中显示的行数。单元格渲染器将调整为该行高度。对于默认文本大小，该行高度大约是两行文本
 的行高 40 或三行文本的行高 60。*/
myGrid_dg.setSize(430,200);
myGrid_dg.move(40,40);
myGrid_dg.rowHeight = 40; // 对于默认文本大小，使用两行文本的行高。
myGrid_dg.getColumnAt(0).width = 70;
myGrid_dg.getColumnAt(1).width = 70;
myGrid_dg.getColumnAt(2).width = 220;
myGrid_dg.resizableColumns = true;
myGrid_dg.vScrollPolicy = "auto";
myGrid_dg.setStyle("backgroundColor", 0xD5D5FF);

// 分配 cellRenderer。
myGrid_dg.getColumnAt(2).cellRenderer = "MultiLineCell";

```

## 12. 保存 Flash 文档，并选择“控制”>“测试影片”。

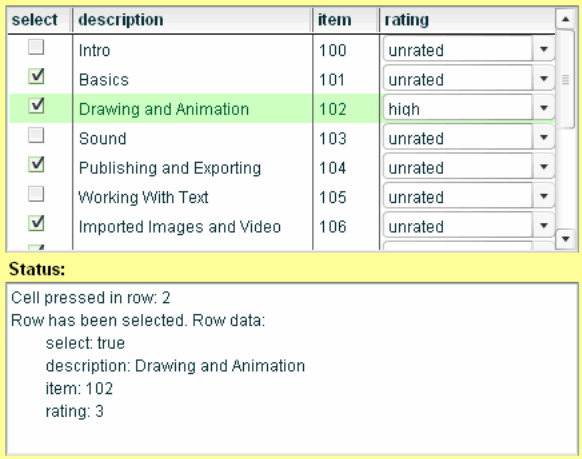
将显示一个数据网格。该数据网格的第三列包含一个多行单元格。

| firstName | lastName | note                                                                        | item |
|-----------|----------|-----------------------------------------------------------------------------|------|
| Kimberly  | Dietrich | An example of a cell renderer class that displays a multiple line TextField | 104  |
| AJ        | Bilow    | An example of a cell renderer class that displays a multiple line TextField | 105  |
| Chuck     | Yushan   | An example of a cell renderer class that displays a multiple line TextField | 106  |
| John      | Roo      | An example of a cell renderer class that displays a multiple line TextField | 107  |

完成的 MultiLineCell 单元格渲染器示例。

## 其它单元格渲染器示例

此外，还提供了显示 `ComboBox` 和 `CheckCell` 组件的其它单元格渲染器类示例。这些文件位于 `CellRenderers_sample` 文件夹中，该文件夹包含在硬盘上的 `Samples and Tutorials` 文件夹中。也可从以下网址下载这些文件：[www.macromedia.com/go/component\\_samples](http://www.macromedia.com/go/component_samples)。



另一个显示 `ComboBox` 和 `CheckBox` 的名为 `CellRenderers_Sample` 的已安装范例。

## 要为 CellRenderer API 实现的方法

您必须编写包含以下方法的类，以便 `List`、`DataGrid`、`Tree` 或 `Menu` 组件可以与单元格通信。

| 方法                                             | 说明             |
|------------------------------------------------|----------------|
| <code>CellRenderer.getPreferredHeight()</code> | 返回单元格的首选高度。    |
| <code>CellRenderer.getPreferredWidth()</code>  | 单元格的首选宽度。      |
| <code>CellRenderer.setSize()</code>            | 设置单元格的宽度和高度。   |
| <code>CellRenderer.setValue()</code>           | 设置要显示在单元格中的内容。 |



# CellRenderer API 提供的方法

以下是在组件内创建单元格时 `List`、`DataGrid`、`Tree` 和 `Menu` 组件为单元格指定的方法。您无需实现这些方法。

| 方法                                       | 说明                                                                              |
|------------------------------------------|---------------------------------------------------------------------------------|
| <code>CellRenderer.getCellIndex()</code> | 返回包含两个字段（ <code>columnIndex</code> 和 <code>itemIndex</code> ）的对象，这两个字段指示单元格的位置。 |
| <code>CellRenderer.getDataLabel()</code> | 返回包含单元格渲染器数据字段的名称的字符串。                                                          |

# CellRenderer API 提供的属性

以下是在组件内创建单元格时 `List`、`DataGrid`、`Tree` 和 `Menu` 组件为单元格指定的属性。您无需实现这些属性。

| 属性                                  | 说明                                |
|-------------------------------------|-----------------------------------|
| <code>CellRenderer.listOwner</code> | 指向包含单元格的 <code>List</code> 组件的引用。 |
| <code>CellRenderer.owner</code>     | 指向包含单元格的行的引用。                     |

# CellRenderer.getCellIndex()

**可用性**  
Flash Player 6 (6.0.79.0)。

**版本**  
Flash MX 2004。

**用法**  
`componentInstance.getCellIndex()`

**参数**  
无。

**返回**  
包含以下两个字段的对象：`columnIndex` 和 `itemIndex`。

**说明**  
方法；返回包含两个字段（`columnIndex` 和 `itemIndex`）的对象，这两个字段确定单元格在组件中的位置。每个字段都是一个整数，它指明单元格的列位置和项目位置。对于除 `DataGrid` 组件外的任何组件，`columnIndex` 的值始终为 0。

此方法由 **List** 类提供；您不一定必须实现该方法。按如下方式在单元格渲染器类中声明该方法，并在单元格渲染器的函数中使用该方法：

```
var getCellIndex:Function;
```

### 示例

此示例从单元格内编辑一个 **DataGrid** 组件的数据提供程序：

```
var index = getCellIndex();
var colName = listOwner.getColumnAt(index.columnIndex).columnName;
listOwner.dataProvider.editField(index.itemIndex, colName, someVal);
```

## CellRenderer.getDataLabel()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
componentInstance.getDataLabel()
```

### 参数

无。

### 返回

字符串。

### 说明

方法；返回包含单元格渲染器数据字段的名称的字符串。对于 **DataGrid** 组件，此方法返回当前单元格的列名称。

此方法由 **List** 类提供；您不一定必须实现该方法。按如下方式在单元格渲染器类中声明该方法，并在单元格渲染器的函数中使用该方法：

```
var getDataLabel:Function;
```

### 示例

以下代码向单元格指示它所呈现的数据字段的名称。例如，如果单元格当前所呈现的数据字段的名称为 "Price"，则变量 p 目前就等于 "Price"：

```
var p = getDataLabel();
```

# CellRenderer.getPreferredHeight()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance*.getPreferredHeight()

## 参数

无。

## 返回

单元格的正确高度。

## 说明

方法：返回单元格的首选高度。此高度对于获取单元格内文本的正确高度特别重要。如果将此值设置为大于组件的 `rowHeight` 属性，单元格将超出行的上下边缘。

此方法不是由 **List** 类提供的；您必须实现此方法。该方法指示列表的行如何使单元格居中以及如何必要时调整单元格的高度。如有必要，您可以返回一个常数（例如 22），否则可以测量并返回内容的高度。您也可以返回表示行高度的 `owner.height`。

## 示例

此示例返回值 20，指示单元格的高度应为 20 个像素：

```
function getPreferredHeight(Void) :Number
{
 return 20;
}
```

此示例返回一个比行的高度小 4 个像素的值：

```
function getPreferredHeight():Number
{
 /* 已知为单元格赋予了表示行的属性 “owner”。首选的做法总是让单元格占用行的大部分高度。
 */
 return owner.__height - 4;
}
```

# CellRenderer.getPreferredWidth()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
componentInstance.getPreferredWidth()
```

## 参数

无。

## 返回

一个数字类型的值，它指示单元格的正确宽度。

## 说明

方法；单元格的首选宽度。如果指定的宽度超出组件的宽度，则单元格可能会被截断。

为 **Menu** 组件实现此方法。无论行的宽度为多少，单元格都会调整为行的宽度，除非是在菜单中（这种情况下必须通过测量文本来确定行的宽度）。对于由其标题渲染器检查是否显示排序箭头的 **DataGrid** 组件，您也可以实现此方法。

## 示例

此示例返回该值乘以 3，指示单元格应该比它所渲染的字符串的长度大三倍：

```
function getPreferredWidth():Number
{
 return myString.length*3;
}
```

此示例注释掉 `getPreferredWidth()` 方法：

```
// 函数 getPreferredWidth :: 仅用于菜单或数据网格
```

# CellRenderer.listOwner

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.listOwner*

## 说明

属性；指向拥有单元格的列表的引用。该列表可以为 **DataGrid**、**Tree**、**List** 或 **Menu** 组件。

此方法由 **List** 类提供；您并非必须实现该方法。按如下方式在单元格渲染器类中声明该方法，并将其用作对列表（或者树、菜单或网格）的引用：

```
var listOwner:MovieClip; // 或 UIObject 等
```

## 示例

此范例查找列表在某个单元格中的选定项目：

```
var s = listOwner.selectedItem;
```

# CellRenderer.owner

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.owner*

## 说明

属性；指向包含单元格的行的引用。

此方法由 **List** 类提供；您并非必须实现该方法。在单元格渲染器类中声明该方法，并将其用作引用：

```
var owner:MovieClip; // 或 UIObject 等
```

# CellRenderer.setSize()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.setSize(width, height)*

## 参数

*width* 指示布置组件所依据宽度的数字。

*height* 指示布置组件所依据高度的数字。

## 返回

无。

## 说明

方法：允许列表告知其单元格应按什么大小安排自己的位置。单元格渲染器应进行布局以便适合指定的区域，否则单元格可能会溢出到列表的其它部分，从而看起来不完整。

如果单元格渲染器是 **UIObject** 类的扩展类，则应改为实现 `size()` 方法。编写与为 `setSize()` 编写的函数相同的函数，但是请使用 `width` 和 `height` 属性代替参数。

## 示例

以下示例将调整单元格内图像的大小，以适合列表所指定的边框：

```
function setSize(w:Number, h:Number):Void
{
 image._width = w-2;
 image._height = h-2;
 image._x = image._y = 1;
}
```

此示例位于一个扩展 **UIComponent** 的单元格渲染器类（该类为 **UIObject** 的扩展类）中，因此必须实现 `size()` 而不是 `setSize()`，如下所示：

```
// 通过扩展 UIComponent，您可自由获取 setSize；
// 但是，UIComponent 期望您实现 size()。
// 假设现在为您设置了 __width 和 __height。
// 您要扩展单元格以适合整个 rowHeight。
```

```
function size():Void
{
 // __width 和 __height 是
```

```
// getters/setters .width 和 .height 的基础变量。
var c = multiLineLabel;
c._width = __width;
c._height = __height;
}
```

## CellRenderer.setValue()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
componentInstance.setValue(suggested, item, selected)
```

### 参数

*suggested* 用于表示单元格渲染器的文本的值（如果需要）。

*item* 表示要呈现的整个项的对象。单元格渲染器可以使用此对象的属性进行渲染。

*selected* 一个字符串，可能具有以下值："normal"、"highlighted" 和 "selected"。

### 返回

无。

### 说明

方法：采用指定的值，并在单元格内将其显示出来。这将使单元格中已显示内容以及要为新项目在单元格中显示的内容不会有任何不同。（请记住，任何单元格处于列表中时都可能显示多个值。）这是最重要的 **CellRenderer** 方法，您必须在每个单元格渲染器中实现该方法。

`setValue()` 方法经常被调用（如发生滑过、选择、调整列的大小或滚动时）。必须记住的一点是单元格可能不在舞台上，因此在调用 `setValue()` 时不应始终用数据更新单元格。例如，特定的单元格可能会随时滚动出显示区域，或者可能被重用以呈现其它值。出于此原因，不能直接引用网格中的特定单元格渲染器实例，而应在 `setValue()` 体内编写 `if` 语句，使得只有在定义了 `item` 参数并发生了更改的情况下才允许运行代码。如果未定义 `item` 参数，则表示单元格在视觉上为空，并且应为单元格中的所有项分配一个值为 `false` 的 `_visible` 属性。单元格可能需要暂时在视觉上为空，如在 **DataGrid** 中发生滚动时。

如果一行被选中且鼠标指针位于该行上，则 **selected** 参数的值为 "highlighted"，而不是 "selected"。如果试图根据行的选中状态而调整单元格渲染器的行为，这可能会产生问题。若要测试当前行是否处于选中状态，请使用下面的代码：

```
var reallySelected:Boolean = selected != "normal" && listOwner.selectedNode
 == item;
```

## 示例

以下示例说明如何使用 setValue() 和 editField() 来引用网格中的单元格渲染器实例。

由于单元格可能会随时不在舞台上（可能会滚动出显示区域，或者可能被重用呈现其它值），因此您不能直接引用网格中的特定单元格渲染器实例。

可以改用数据提供程序与网格内的特定单元格通信。数据提供程序保存关于网格的所有状态信息。为了将给定单元格显示为已启用或已选择，数据提供程序中应该存在相应字段以保存该信息。单元格渲染器的 setValue() 方法将数据提供程序状态的更改传达给该单元格。下面是一个在单元格中呈现复选框的理论上的单元格渲染器的 setValue() 实现：

```
function setValue(str, itm, sel)
{
 /* 假设数据提供程序具有两个与此单元格相关的字段：checked 和 enabled。
 这种数据提供程序的形式可能如下所示：
 [
 {field1:"DisplayMe", field2:"SomeString", checked:true, enabled:false}
 {field1:"DisplayMe", field2:"SomeString", checked:false, enabled:true}
 {field1:"DisplayMe", field2:"SomeString", checked:true, enabled:true}
]
 */

 /* 如果未定义 item 参数，则隐藏通常在单元格中呈现的所有内容。否则，将用新数据更新单元格内容。
 */
 if (itm == undefined){
 myCheck._visible = false;
 }else{

 // 冗余检查
 if (myCheck.selected!=itm.checked){
 myCheck.selected = itm.checked;
 }
 if (myCheck.enabled!=itm.enabled){
 myCheck.enabled = itm.enabled;
 }
 }
}
```



如果您要在第二行上启用复选框，则可以通过数据提供程序进行通信。对数据提供程序所做的任何更改（当通过如 `DataProvider.editField()` 的 **DataProvider** 方法进行更改时）都调用 `setValue()` 以刷新网格的显示。此代码在 **Flash** 应用程序中编写，无论是在帧上、对象上，还是在其它类文件中（但不能在单元格渲染器类文件中）：

```
// 再次调用 setValue()
myGrid.editField(1, "enabled", true);
```

以下示例会将图像加载到单元格内的加载器组件中，具体取决于传递的值：

```
function setValue(suggested, item, selected) : Void
{
 /* 如果未定义 item 参数，则隐藏通常在单元格中呈现的所有内容。否则，将用新数据更新单元格内容。 */
 if (item == undefined){
 loader._visible = false;
 }else{
 // 清除加载器
 loader.contentPath = undefined;
 // 列表数据提供程序中的不同图像有不同的 URL
 if (suggested!=undefined){
 loader.contentPath = suggested;
 }
 }
}
```

以下示例摘自一个多行文本单元格渲染器：

```
function setValue(suggested:String, item:Object, selected:Boolean):Void
{
 /* 如果未定义 item 参数，则隐藏通常在单元格中呈现的所有内容。否则，将用新数据更新单元格内容。 */
 if (item == undefined){
 multiLineLabel._visible = false;
 }else{
 // 向标签添加文本
 multiLineLabel.text = suggested;
 }
}
```

以下示例摘自一个单选按钮渲染器。如果未定义 **item** 参数，则单元格可能滚出显示区域并应在视觉上为空。可以使用 `if` 语句确定是否未定义 `item` 参数。如果未定义 `item` 参数，则可以通过将单选按钮的 `_visible` 属性设置为 `false` 来隐藏它；否则用新数据更新该单选按钮并显示它。

```
function setValue(str:String, item:Object, sel:String) : Void {
 /* 如果未定义 item 参数，则隐藏通常在单元格中呈现的所有内容。否则，将用新数据更新单元格内容。 */
 if (item == undefined) {
 radio._visible = false; }
 else {
```

```
 trace(item.data + " " + item.label + " " + item.state + " " + sel);
 radio.label = item.label;
 radio.data = item.data;
 radio.selected = item.state;
 radio._visible = true;
 }
}
```

# CheckBox 组件

复选框是一个可以选中或取消选中的方框。当它被选中后，框中会出现一个复选标记。您可以为复选框添加一个文本标签，并可以将它放在左侧、右侧、顶部或底部。

可以在应用程序中启用或者禁用复选框。如果复选框已启用，并且用户单击它或者它的标签，复选框会接收输入焦点并显示为按下状态。如果用户在按下鼠标按钮时将指针移到复选框或其标签的边界区域之外，则组件的外观会返回到其最初状态，并保持输入焦点。在组件上释放鼠标之前，复选框的状态不会发生变化。另外，复选框有两种禁用状态：选中和取消选中，这两种状态不允许鼠标或键盘的交互操作。

如果复选框被禁用，它会显示其禁用外观，无论用户的交互操作如何。在禁用状态下，按钮不接收鼠标或键盘输入。

如果用户单击 **CheckBox** 实例或者用 **Tab** 按键切换到它时，**CheckBox** 实例将接收焦点。当一个 **CheckBox** 实例有焦点时，您可以使用下列按键来控制它：

| 键         | 说明                       |
|-----------|--------------------------|
| Shift+Tab | 将焦点移到前一个元素。              |
| 空格键       | 选中或者取消选中组件，并触发 click 事件。 |
| Tab       | 将焦点移到下一个元素。              |

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

每个 **CheckBox** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。

在将 **CheckBox** 组件添加到应用程序时，您可以使用“辅助功能”面板使其可由屏幕读取器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.CheckBoxAccImpl.enableAccessibility();
```

不管组件有多少实例，都只对组件启用一次辅助功能。有关更多信息，请参见《使用 Flash》中的第 19 章“创建辅助内容”。

# 使用 CheckBox 组件

复选框是任何表单或 Web 应用程序中的一个基础部分。每当需要收集一组非相互排斥的 true 或 false 值时，都可以使用复选框。例如，一个收集客户个人信息的表单可能有一个爱好列表供客户选择；每个爱好的旁边都有一个复选框。

## CheckBox 参数

您可以在“属性”检查器或“组件”检查器中为每个 CheckBox 组件实例设置以下创作参数：

**label** 设置复选框上文本的值；默认值为 `CheckBox`。

**labelPlacement** 确定复选框上标签文本的方向。该参数可以是下列四个值之一：`left`、`right`、`top` 或 `bottom`；默认值为 `right`。有关更多信息，请参见 [CheckBox.labelPlacement](#)。

**selected** 将复选框的初始值设置为选中 (`true`) 或取消选中 (`false`)。默认值为 `false`。

您可以编写 `ActionScript`，以便使用 `CheckBox` 组件的属性、方法和事件来控制该组件的这些选项和其它选项。有关更多信息，请参见第 124 页的“[CheckBox 类](#)”。

## 创建具有 CheckBox 组件的应用程序

以下过程解释了如何在创作时将 `CheckBox` 组件添加到应用程序。下面的范例是一个用于联机约会应用程序的表单，该表单是一个查询，它搜索与客户相匹配的可能约会。该查询表单必须有一个标签为“**Restrict Age**”的复选框，以允许客户将其搜索限定在一个指定的年龄组。选中“**Restrict Age**”复选框后，客户就可以在两个文本字段中输入最小年龄和最大年龄。（这两个文本字段仅在该复选框被选中时才启用。）

创建具有 `CheckBox` 组件的应用程序：

1. 将两个 `TextInput` 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，输入实例名称 **minimumAge** 和 **maximumAge**。
3. 将 `CheckBox` 组件从“组件”面板拖到舞台上。
4. 在“属性”检查器中，执行以下操作：
  - 输入 **restrictAge** 作为实例名称。
  - 输入 **Restrict Age**（限制年龄）作为标签参数。

5. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
var restrictAgeListener:Object = new Object();
restrictAgeListener.click = function (evt:Object) {
 minimumAge.enabled = evt.target.selected;
 maximumAge.enabled = evt.target.selected;
};
restrictAge.addEventListener("click", restrictAgeListener);
```

这段代码创建一个 click 事件处理函数，该处理函数可启用和禁用已放到舞台上的 minimumAge 和 maximumAge 文本字段组件。有关更多信息，请参见 [CheckBox.click](#) 和第 1115 页的“[TextInput 组件](#)”。

### 使用 ActionScript 创建复选框：

1. 将 **CheckBox** 组件从“组件”面板拖到当前文档的库中。  
此操作将组件添加到库中，但不会在应用程序中显示。
2. 将 **TextInput** 组件从“组件”面板拖到当前文档的库中。
3. 在主时间轴的第一帧中，向“动作”面板添加以下 **ActionScript** 代码以创建组件实例并定位这些组件实例：

```
this.createClassObject(mx.controls.CheckBox, "testAge_ch", 1, {label:'Age
Range', selected:true});
this.createClassObject(mx.controls.TextInput, "minimumAge_ti", 2,
 {restrict:[0-9], text:18, maxChars:2});
minimumAge_ti.move(20, 30);
this.createClassObject(mx.controls.TextInput, "maximumAge_ti", 3,
 {restrict:[0-9], text:55, maxChars:2});
maximumAge_ti.move(20, 60);
```

此脚本使用第 1254 页的“[UIObject.createClassObject\(\)](#)”方法创建名为 **restrictAge** 的 **CheckBox** 实例，并指定一个标签属性。然后，代码使用第 1265 页的“[UIObject.move\(\)](#)”方法来定位按钮。

4. 现在，添加以下 **ActionScript** 来创建一个事件侦听器和一个事件处理函数：

```
// 创建 CheckBox 事件的处理函数。
function checkboxHandler(evt_obj:Object) {
 minimumAge_ti.enabled = evt_obj.target.selected;
 maximumAge_ti.enabled = evt_obj.target.selected;
}
// 添加侦听器。
testAge_ch.addEventListener("click", checkboxHandler);
```

这段代码创建一个 click 事件处理函数，它可启用和禁用 minimumAge 和 maximumAge 文本字段组件。有关更多信息，请参见 [CheckBox.click](#)、第 462 页的“[EventDispatcher.addEventListener\(\)](#)”和第 1115 页的“[TextInput 组件](#)”。

# 自定义 CheckBox 组件

在创作过程中和运行时，可以在水平和垂直方向上改变 **CheckBox** 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，可以使用 `setSize()` 方法 (`UIObject.setSize()`) 或 **CheckBox** 类的任何适用的属性和方法。调整复选框的大小不会改变标签或复选框图标的大小；它只会改变边框的大小。

**CheckBox** 实例的边框是不可见的，它同时也指定了该实例的点击区域。如果您增加实例的大小，也就增加了点击区的大小。如果边框太小而无法容纳标签，标签会被裁剪以适合边框。

## 对 CheckBox 组件使用样式

您可以设置样式属性以更改 **CheckBox** 实例的外观。如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

**CheckBox** 组件支持下列样式：

| 样式            | 主题    | 说明                                                                                                                                              |
|---------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| themeColor    | 光晕    | 组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。                                                                        |
| color         | 光晕和范例 | 文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。                                                                                                        |
| disabledColor | 光晕和范例 | 组件禁用时的文本颜色。默认值为 0x848384（深灰）。                                                                                                                   |
| embedFonts    | 光晕和范例 | 一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。 |
| fontFamily    | 光晕和范例 | 文本的字体名称。默认值为 "_sans"。                                                                                                                           |
| fontSize      | 光晕和范例 | 字体的磅值。默认值为 10。                                                                                                                                  |
| fontStyle     | 光晕和范例 | 字体样式："normal" 或 "italic"。默认值为 "normal"。                                                                                                         |
| fontWeight    | 光晕和范例 | 字体粗细："none" 或 "bold"。默认值为 "none"。在调用 setStyle() 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 getStyle() 的调用将返回 "none"。                                |

| 样式                            | 主题    | 说明                                      |
|-------------------------------|-------|-----------------------------------------|
| textDecoration                | 光晕和范例 | 文本修饰: "none" 或 "underline"。默认值为 "none"。 |
| symbolBackgroundColor         | 范例    | 复选框的背景颜色。默认值为 0xFFFFFF (白色)。            |
| symbolBackgroundDisabledColor | 范例    | 复选框在禁用时的背景颜色。默认值为 0xEFEFEF (浅灰)。        |
| symbolBackgroundPressedColor  | 范例    | 复选框在按下时的背景颜色。默认值为 0xFFFFFF (白色)。        |
| symbolColor                   | 范例    | 复选标记的颜色。默认值为 0x000000 (黑色)。             |
| symbolDisabledColor           | 范例    | 禁用的复选标记的颜色。默认值为 0x848384 (深灰)。          |

## 对 CheckBox 组件使用外观

CheckBox 组件使用库中的元件来表示按钮的状态。若要在创作时设计 CheckBox 组件的外观, 请修改“库”面板中的元件。CheckBox 组件外观位于 HaloTheme.fla 文件或者 SampleTheme.fla 文件的库中的 Flash UI Components 2/Themes/MMDefault/CheckBox Assets/states 文件夹下。有关更多信息, 请参见《使用组件》中的“关于设置组件外观”。

CheckBox 组件使用下列外观属性:

| 属性                | 说明                                 |
|-------------------|------------------------------------|
| falseUpSkin       | 向上 (正常) 且未选中的状态。默认值为 CheckFalseUp。 |
| falseDownSkin     | 按下而未选中的状态。默认值为 CheckFalseDown。     |
| falseOverSkin     | 指针经过而未选中的状态。默认值为 CheckFalseOver。   |
| falseDisabledSkin | 禁用而未选中的状态。默认值为 CheckFalseDisabled。 |
| trueUpSkin        | 切换且选中的状态。默认值为 CheckTrueUp。         |
| trueDownSkin      | 按下且选中的状态。默认值为 CheckTrueDown。       |
| trueOverSkin      | 指针经过且选中的状态。默认值为 CheckTrueOver。     |
| trueDisabledSkin  | 禁用且选中的状态。默认值为 CheckTrueDisabled。   |

这些外观中的每一个都对应于指示 CheckBox 状态的图标。CheckBox 组件不具有边框或背景颜色。

### 创建 CheckBox 外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 HaloTheme.flc 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在您的操作系统上的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 Flash UI Components 2/Themes/MMDefault 文件夹并将 CheckBox Assets 文件夹拖动到文档的库中。
4. 在文档的库中展开 CheckBox Assets/States 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如，打开 CheckFalseDisabled 元件。
6. 按需要自定义元件。  
例如，将内部白色方框更改为浅灰色。
7. 对所有要自定义的元件重复步骤 5-6。  
例如，重复更改 CheckTrueDisabled 元件的内部框的颜色。
8. 单击“返回”按钮返回主时间轴。
9. 将 CheckBox 组件拖动到舞台上。  
对于此示例，拖动两个实例以显示两个新的外观元件。
10. 根据需要设置 CheckBox 实例的属性。  
对于此示例，将一个 CheckBox 实例设置为 true，并使用 ActionScript 代码将两个 CheckBox 实例都设置为禁用。
11. 选择“控制”>“测试影片”。

## CheckBox 类

继承 MovieClip > UIObject 类 > UIComponent 类 > SimpleButton 类 > Button 组件 > CheckBox

ActionScript 类名称 mx.controls.CheckBox

CheckBox 类的属性允许您在运行时创建一个文本标签，并将其放置在复选框的左部、右部、上部或下部。


使用 ActionScript 设置 CheckBox 类的属性将会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

CheckBox 组件使用焦点管理器覆盖默认的 Flash Player 焦点矩形，并绘制一个带圆角的自定义焦点矩形。有关更多信息，请参见《使用组件》中的“创建自定义焦点导航”。



每个组件类都有一个 `version` 属性，该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.CheckBox.version);
```



代码 `trace(myCheckBoxInstance.version);` 返回 `undefined`。

## CheckBox 类的方法摘要

没有 `CheckBox` 类专用的方法。

### 从 `UIObject` 类继承的方法

下表列出了 `CheckBox` 类从 `UIObject` 类继承的方法。从 `CheckBox` 对象调用这些方法时，请使用 `checkBoxInstance.methodName` 的形式。

| 方法                                        | 说明                               |
|-------------------------------------------|----------------------------------|
| <code>UIObject.createClassObject()</code> | 创建指定类的对象。                        |
| <code>UIObject.createObject()</code>      | 创建对象的子对象。                        |
| <code>UIObject.destroyObject()</code>     | 销毁组件实例。                          |
| <code>UIObject.doLater()</code>           | 在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。 |
| <code>UIObject.getStyle()</code>          | 从样式声明或对象中获取样式属性。                 |
| <code>UIObject.invalidate()</code>        | 标记对象使其在到达下一个帧间隔时进行重绘。            |
| <code>UIObject.move()</code>              | 将对象移动到要求的位置。                     |
| <code>UIObject.redraw()</code>            | 迫使对象有效以便能在当前帧中绘制。                |
| <code>UIObject.setSize()</code>           | 将对象调整为所要求的大小。                    |
| <code>UIObject.setSkin()</code>           | 设置对象的外观。                         |
| <code>UIObject.setStyle()</code>          | 设置样式声明或对象内的样式属性。                 |

## 从 UIComponent 类继承的方法

下表列出了 `CheckBox` 类从 `UIComponent` 类继承的方法。从 `CheckBox` 对象调用这些方法时，请使用 `checkBoxInstance.methodName` 的形式。

| 方法                                  | 说明             |
|-------------------------------------|----------------|
| <code>UIComponent.getFocus()</code> | 返回对具有焦点的对象的引用。 |
| <code>UIComponent.setFocus()</code> | 将焦点设置到组件实例中。   |

## CheckBox 类的属性摘要

下表列出了 `CheckBox` 类的属性。

| 属性                                   | 说明                                                                    |
|--------------------------------------|-----------------------------------------------------------------------|
| <code>CheckBox.label</code>          | 指定在复选框旁边出现的文本。                                                        |
| <code>CheckBox.labelPlacement</code> | 指定标签文本相对于复选框的方向。                                                      |
| <code>CheckBox.selected</code>       | 指定复选框是处于选中状态 ( <code>true</code> ) 还是处于取消选中状态 ( <code>false</code> )。 |

## 从 UIObject 类继承的属性

下表列出了 `CheckBox` 类从 `UIObject` 类继承的属性。从 `CheckBox` 对象访问这些属性时，请使用 `checkBoxInstance.propertyName` 的形式。

| 属性                            | 说明                                                                   |
|-------------------------------|----------------------------------------------------------------------|
| <code>UIObject.bottom</code>  | 只读；对象的底边缘位置（相对于其父对象的底边缘）。                                            |
| <code>UIObject.height</code>  | 只读；对象的高度，以像素为单位。                                                     |
| <code>UIObject.left</code>    | 只读；对象的左边缘（以像素为单位）。                                                   |
| <code>UIObject.right</code>   | 只读；对象的右边缘位置（相对于其父对象的右边缘）。                                            |
| <code>UIObject.scaleX</code>  | 一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。                                       |
| <code>UIObject.scaleY</code>  | 一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。                                       |
| <code>UIObject.top</code>     | 只读；对象上边缘的位置（相对于其父对象）。                                                |
| <code>UIObject.visible</code> | 一个布尔值，它指示对象是可见的 ( <code>true</code> ) 还是不可见的 ( <code>false</code> )。 |
| <code>UIObject.width</code>   | 只读；对象的宽度，以像素为单位。                                                     |
| <code>UIObject.x</code>       | 只读；对象的左边缘（以像素为单位）。                                                   |
| <code>UIObject.y</code>       | 只读；对象的上边缘（以像素为单位）。                                                   |

## 从 UIComponent 类继承的属性

下表列出了 `CheckBox` 类从 `UIComponent` 类继承的属性。从 `CheckBox` 对象访问这些属性时，请使用 `checkBoxInstance.propertyName` 的形式。

| 属性                                | 说明                     |
|-----------------------------------|------------------------|
| <code>UIComponent.enabled</code>  | 指明组件是否可以接收焦点和输入。       |
| <code>UIComponent.tabIndex</code> | 一个数字，指明文档中组件的 Tab 键顺序。 |

## 从 SimpleButton 类继承的属性

下表列出了 `CheckBox` 类从 `SimpleButton` 类继承的属性。从 `CheckBox` 对象访问这些属性时，请使用 `checkBoxInstance.propertyName` 的形式。

| 属性                                                   | 说明                                                                                                 |
|------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <code>SimpleButton.emphasized</code>                 | 指示按钮是否具有默认普通按钮的外观。                                                                                 |
| <code>SimpleButton.emphasizedStyleDeclaration</code> | 当 <code>emphasized</code> 属性设置为 <code>true</code> 时的样式声明。                                          |
| <code>SimpleButton.selected</code>                   | 一个布尔值，它指示按钮处于选中状态 ( <code>true</code> ) 还是未处于选中状态 ( <code>false</code> )。默认值为 <code>false</code> 。 |
| <code>SimpleButton.toggle</code>                     | 一个布尔值，它指示按钮的行为与切换开关相同 ( <code>true</code> ) 还是不同 ( <code>false</code> )。默认值为 <code>false</code> 。  |

## 从 Button 类继承的属性

下表列出了 `CheckBox` 类从 `Button` 类继承的属性。从 `CheckBox` 对象访问这些属性时，请使用 `checkBoxInstance.propertyName` 的形式。

| 属性                                 | 说明              |
|------------------------------------|-----------------|
| <code>Button.label</code>          | 指定在按钮上显示的文本。    |
| <code>Button.labelPlacement</code> | 指定标签文本相对于图标的方向。 |

# CheckBox 类的事件摘要

下表列出了 CheckBox 类的一个事件。

| 事件                          | 说明                                   |
|-----------------------------|--------------------------------------|
| <code>CheckBox.click</code> | 在复选框上单击（松开）鼠标时，或者如果复选框具有焦点并按下空格键时触发。 |

## 从 UIObject 类继承的事件

下表列出了 CheckBox 类从 UIObject 类继承的事件。

| 事件                           | 说明                 |
|------------------------------|--------------------|
| <code>UIObject.draw</code>   | 当对象将要绘制它的图形时进行广播。  |
| <code>UIObject.hide</code>   | 在对象的状态从可见变为不可见时广播。 |
| <code>UIObject.load</code>   | 创建子对象时广播。          |
| <code>UIObject.move</code>   | 移动了对象时广播。          |
| <code>UIObject.resize</code> | 在调整对象大小后广播。        |
| <code>UIObject.reveal</code> | 在对象的状态从不可见变为可见时广播。 |
| <code>UIObject.unload</code> | 卸载子对象时广播。          |

## 从 UIComponent 类继承的事件

下表列出了 CheckBox 类从 UIComponent 类继承的事件。

| 事件                                | 说明            |
|-----------------------------------|---------------|
| <code>UIComponent.focusIn</code>  | 当对象收到焦点时进行广播。 |
| <code>UIComponent.focusOut</code> | 当对象失去焦点时进行广播。 |
| <code>UIComponent.keyDown</code>  | 当按下按键时进行广播。   |
| <code>UIComponent.keyUp</code>    | 当松开按键时进行广播。   |

## 从 SimpleButton 类继承的事件

下表列出了 CheckBox 类从 SimpleButton 类继承的事件。

| 事件                              | 说明         |
|---------------------------------|------------|
| <code>SimpleButton.click</code> | 单击一个按钮时广播。 |

# CheckBox.click

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObject:Object) {
 // ...
};
checkBoxInstance.addEventListener("click", listenerObject);
```

用法 2:

```
on (click) {
 // ...
}
```

## 说明

事件：在复选框上单击（松开）鼠标时，或者如果复选框具有焦点并按下了空格键时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*checkBoxInstance*) 调度一个事件（在本例中为 `click`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递给侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()` 方法（请参见 [EventDispatcher.addEventListener\(\)](#)），以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用一个 `on()` 处理函数，并且必须直接附加到一个 **CheckBox** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到复选框 `myCheckBox`，它将“`_level0.myCheckBox`”发送到“输出”面板：

```
on (click) {
 trace(this);
}
```

## 示例

以下示例在复选框被选中时会启用按钮。此示例假定舞台上具有一个名为 `submit_button` 的 **Button** 组件实例以及一个名为 `agree_ch` 的 **CheckBox** 组件实例。向主时间轴的第一帧中添加以下代码：

```
agree_ch.label = "I agree";
submit_button.enabled = false;

// 创建侦听器对象。
var form_obj:Object = new Object();

// 为侦听器对象分配函数。
form_obj.click = function(event_obj:Object) {
 submit_button.enabled = event_obj.target.selected;
};

// 添加侦听器。
agree_ch.addEventListener("click", form_obj);
```

以下代码也会在 `checkBoxInstance` 被单击时向“输出”面板发送消息。`on()` 处理函数必须直接附加到 `checkBoxInstance`：

```
on (click) {
 trace("check box component was clicked");
}
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# CheckBox.label

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`checkBoxInstance.label`

## 说明

属性：显示复选框的文本标签。默认情况下，标签出现在复选框的右侧。对此属性的设置会覆盖在“组件”检查器的“参数”选项卡中指定的标签参数。

**CheckBox** 组件不允许具有多行标签。

## 示例

以下代码设置显示在 **CheckBox** 组件旁的文本，并向 “输出” 面板发送该值：

```
checkBox.label = "Remove from list";
trace(checkBox.label)
```

此示例在 **ActionScript** 中创建复选框，并在被选中时调整标签的大小。对于本示例，将一个 **CheckBox** 组件从 “组件” 面板拖到当前文档的库中（使该 **CheckBox** 组件显示在库中，但不显示在舞台上）。然后向主时间轴的第一帧中添加以下 **ActionScript** 代码：

```
this.createClassObject(mx.controls.CheckBox, "my_ch", 10, {label:"Resize
CheckBox instance"});
```

```
function checkBoxHandler(evt_obj:Object):Void {
 trace("before: " + evt_obj.target.width + "px wide");
 evt_obj.target.setSize(200, evt_obj.target.height);
 trace("after: " + evt_obj.target.width + "px wide");
}
my_ch.addEventListener("click", checkBoxHandler);
```

## 另请参见

[CheckBox.labelPlacement](#)

# CheckBox.labelPlacement

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

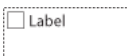
## 用法

*checkBoxInstance*.labelPlacement

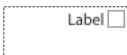
## 说明

属性；一个字符串，它指明标签相对于复选框的位置。下面是四种可能的值（虚线表示组件的边界区域，它们在文档中不可见）：

- "right" 复选框被固定在边界区域的左上角。标签设置在复选框的右边。这是默认值。



- "left" 复选框被固定在边界区域的右上角。标签设置在复选框的左边。



- "bottom" 标签设置在复选框的下面，并且复选框和标签水平、垂直居中。



- "top" 标签放置在复选框的上面，并且复选框和标签水平、垂直居中。



在创作时可以使用“变形”命令来更改组件的边界区域，或者在运行时使用 `UIObject.setSize()` 属性来更改组件的边界区域。有关更多信息，请参见第 122 页的“自定义 **CheckBox** 组件”。

## 示例

下面的范例将标签的位置设置在复选框的左侧。

```
checkBox_mc.labelPlacement = "left";
```

以下示例使用 **ActionScript** 来创建复选框实例。复选框实例 `right_ch` 的 `label` 和 `labelPlacement` 属性在第 1254 页的“`UIObject.createClassObject()`”方法中设置。复选框实例 `left_ch` 的 `label` 和 `labelPlacement` 属性在单独的声明中设置。将 **CheckBox** 组件从“组件”面板拖到当前文档的库中（因而该组件显示在库中，但不显示在舞台上）。然后向主时间轴的第一帧中添加以下 **ActionScript** 代码：

```
this.createClassObject(mx.controls.CheckBox, "right_ch", 1, {label:"Right",
 labelPlacement:"right"});
right_ch.move(10, 10);
this.createClassObject(mx.controls.CheckBox, "left_ch", 2);
left_ch.label= "Left";
left_ch.labelPlacement = "left";
left_ch.move(10, 30);
```

## 另请参见

[CheckBox.label](#)



# CheckBox.selected

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*checkBoxInstance.selected*

## 说明

属性；一个布尔值，它指示选中 (true) 或取消选中 (false) 复选框。

## 示例

以下示例显示一个在默认情况下其 **selected** 属性设置为 true 的复选框，然后在事件处理函数内使用 **selected** 属性响应用户单击该复选框的操作。将 **CheckBox** 组件拖动到舞台上。并将组件实例名称指定为 my\_ch。然后，向主时间轴第一帧的“动作”面板中添加以下代码：

```
my_ch.selected = true;
```

```
var checkBoxListener:Object = new Object();
checkBoxListener.click = function(evt_obj:Object) {
 if (evt_obj.target.selected) {
 evt_obj.target.label = "Selected!";
 } else {
 evt_obj.target.label = "Unselected!";
 }
};
my_ch.addEventListener("click", checkBoxListener);
```



# Collection 接口（仅限 Flash Professional）

集合类在公用类库中作为编译剪辑元件提供。要访问此类，请选择“窗口”>“公用库”>“类”，其中包含编译剪辑 `UtilsClasses`。

## Collection 类（仅限 Flash Professional）

ActionScript 类名称 `mx.utils.Collection`

**Collection** 接口使您可以使用编程方式管理一组相关项（称为“集合项”）。此集合中的每个集合项目都具有在集合项目类定义的元数据中描述的属性。

组件可以将属性公开为集合，您可以在创作过程中通过使用“组件”检查器的“值”对话框处理这些集合。使用此对话框，您可以添加项目、删除项目、更改项目的属性以及更改项目在集合中的位置。有关集合和集合项的更多信息，请参见《使用组件》中的“关于 **Collection** 标记”。

通常将集合接口用于使用 **Collection** 元数据标记来创建集合属性的组件。虽然可以使用编程方式创建、访问和删除 **Collection** 实例，但集合在大多数情况下与组件一起使用。**Flash MX Professional 2004** 为两种与集合相关的接口均提供了实现（适用于 **Collection** 的 **CollectionImpl** 以及适用于 **Iterator** 的 **IteratorImpl**）。

# Collection 接口的方法摘要

下表列出了 **Collection** 接口的方法。

| 方法                                    | 说明                 |
|---------------------------------------|--------------------|
| <code>Collection.addItem()</code>     | 在集合的结尾添加新项目。       |
| <code>Collection.contains()</code>    | 指示集合是否包含指定项目。      |
| <code>Collection.clear()</code>       | 删除集合中的所有元素。        |
| <code>Collection.getItemAt()</code>   | 通过使用集合内项目的索引返回该项目。 |
| <code>Collection.getIterator()</code> | 返回集合中元素的一个重复值。     |
| <code>Collection.getLength()</code>   | 返回集合中的项目数。         |
| <code>Collection.isEmpty()</code>     | 指示集合是否为空。          |
| <code>Collection.removeItem()</code>  | 从集合中删除指定项目。        |

## Collection.addItem()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

`collection.addItem(item)`

### 参数

*item* 要添加到集合中的对象。如果 *item* 为 `null`，则它未添加到集合中。

### 返回

一个表示操作结果的布尔值，如果集合被更改，则该值为 `true`。

### 说明

方法：在集合的结尾添加新项。

## 示例

以下示例调用 `addItem()`:

```
on (click) {
 import CompactDisc;

 var myColl:mx.utils.Collection;
 myColl = _parent.thisShelf.MyCompactDiscs;
 myCD = new CompactDisc();
 myCD.Artist = "John Coltrane";
 myCD.Title = "Giant Steps";

 var wasAdded:Boolean = myColl.addItem(myCD);
}
```

# Collection.contains()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*collection*.contains(*item*)

## 参数

*item* 要测试其是否包含在集合中的对象。

## 返回

一个布尔值，如果集合包含 *item*，则该值为 `true`。

## 说明

方法：指示集合是否包含指定的项目。要让 **Flash** 将多个对象视为等同，这些对象必须引用同一对象。如果 *item* 为不同的对象，则即使该对象的所有属性都相同，`Collection.contains()` 也会返回 `false`。

## 示例

以下示例调用 `contains()`:

```
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;

var itr:mx.utils.Iterator = myColl.getIterator();
while (itr.hasNext()) {
 var cd:CompactDisc = CompactDisc(itr.next());
 var title:String = cd.Title;
 var artist:String = cd.Artist;

 if(myColl.contains(cd)) {
 trace("myColl contains " + title);
 }
 else {
 trace("myColl does not contain " + title);
 }
}
```

# Collection.clear()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
collection.clear()
```

## 返回

无。

## 说明

方法：删除集合中的所有元素。

## 示例

以下示例调用 `clear()`:

```
on (click) {
 var myColl:mx.utils.Collection;
 myColl = _parent.thisShelf.MyCompactDiscs;
 myColl.clear();
}
```

# Collection.getItemAt()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
collection.getItemAt(index)
```

## 参数

*index* 一个数字，指示 *item* 在集合中的位置。这是一个从零开始的索引，因此 0 检索第一个项目，1 检索第二个项目，依此类推。

## 返回

一个包含对指定集合项的引用的对象，如果 *index* 超出边框，则返回 null。

## 说明

方法；通过使用集合内的一个项目的索引返回该项目。

## 示例

以下示例调用 getItemAt()：

```
//...
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;
var myCD = CompactDisc(myColl.getItemAt(0));
if (myCD !=null) {
 trace("Retrieved " + myCD.Title);
}
//...
```

# Collection.iterator()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
collection.iterator()
```

## 返回

一个可用来遍历集合的 **Iterator** 对象。

## 说明

方法：返回集合中的元素的一个重复值。对元素的返回顺序不作任何保证（除非此集合是提供了保证的类的实例）。

## 示例

以下示例调用 `iterator()`：

```
on (click) {
 var myColl:mx.utils.Collection;
 myColl = _parent.thisShelf.MyCompactDiscs;

 var itr:mx.utils.Iterator = myColl.iterator();
 while (itr.hasNext()) {
 var cd:CompactDisk = CompactDisc(itr.next());
 var title:String = cd.Title;
 var artist:String = cd.Artist;

 trace("Title: " + title + " - Artist: " + artist);
 }
}
```



# Collection.getLength()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
collection.getLength()
```

## 返回

集合中的项数。

## 说明

方法：返回集合中的项目数。

## 示例

以下示例调用 `getLength()`：

```
//...
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;
trace ("Collection size is: " + myColl.getLength());
//...
```

# Collection.isEmpty()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
collection.isEmpty()
```

## 返回

一个布尔值，如果集合为空，则该值为 `true`。

## 说明

方法：指明集合是否为空。

## 示例

以下示例调用 `isEmpty()`：

```
on (click) {
 var myColl:mx.utils.Collection;
 myColl = _parent.thisShelf.MyCompactDiscs;
 if (myColl.isEmpty()) {
 trace("No CDs in the collection");
 }
}
//...
```

# Collection.removeItem()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*collection*.removeItem(*item*)

## 参数

*item* 要从集合中删除的对象。

## 返回

一个布尔值，如果成功删除了 *item*，则该值为 `true`。

## 说明

方法：从集合中删除指定项目。由于 `Collection.removeItem()` 动态地减小集合的大小，因此循环访问迭代器时不要调用此方法。

## 示例

以下示例调用 `removeItem()`:

```
var myColl:mx.utils.Collection;
myColl = _parent.thisShelf.MyCompactDiscs;

// 从一个文本输入框获取此值
var removeArtist:String = _parent.tArtistToRemove.text;
var removeSize:Number = 0;

if (myColl.isEmpty()) {
 trace("No CDs in the collection");
}
else {
 var toRemove:Array = new Array();
 var itr:mx.utils.Iterator = myColl.getIterator();
 var cd:CompactDisc = new CompactDisc();
 var title:String = "";
 var artist:String = "";
 while (itr.hasNext()) {
 cd = CompactDisc(itr.next());
 title = cd.Title;
 artist = cd.Artist;
 if(artist == removeArtist) {
 // 标记此艺术家以便删除
 removeSize = toRemove.push(cd);
 trace("*** Marked for deletion: " + artist + "|" + title);
 }
 }
 // 在 while 循环后删除错误的项
 var removeCD:CompactDisc = new CompactDisc();
 for(i = 0; i < removeSize; i++) {
 removeCD = toRemove[i];
 trace("Removing: " + removeCD.Artist + "|" + removeCD.Title);
 myColl.removeItem(removeCD);
 }
}
```



# ComboBox 组件

通过使用组合框，用户可以从下拉列表中选出一项选择。组合框可以是静态的，也可以是可编辑的。通过使用可编辑的组合框，用户可以在列表顶部的文本字段中直接输入文本，也可以从下拉列表中选择一项。如果下拉列表超出文档底部，该列表将会向上打开，而不是向下。组合框由三个子组件组成，它们是：**Button** 组件、**TextInput** 组件和 **List** 组件。

当在列表中进行选择后，所选内容的标签被复制到组合框顶部的文本字段中。进行选择时既可以使用鼠标也可以使用键盘。

如果单击文本框或按钮，**ComboBox** 组件就会获取焦点。当 **ComboBox** 组件拥有焦点并且可编辑时，所有键盘输入都会传递到文本框并根据 **TextInput** 组件（请参见第 1115 页的“**TextInput** 组件”）的规则进行处理，但以下按键除外：

| 键          | 说明             |
|------------|----------------|
| Ctrl+ 向下箭头 | 打开下拉列表并给它设置焦点。 |
| Shift+Tab  | 将焦点移到前一个对象。    |
| Tab        | 将焦点移到下一个对象。    |

如果一个 **ComboBox** 组件具有焦点，并且是静态的，按字母数字键就会沿下拉列表将选区上移和下移到下一个首字符相同的项目。您也可以使用下面的按键来控制静态组合框：

| 键          | 说明                                      |
|------------|-----------------------------------------|
| Ctrl+ 向下箭头 | 打开下拉列表并给它设置焦点。                          |
| Ctrl+ 向上箭头 | 关闭下拉列表（如果已在 Flash Player 的独立和浏览器版本中打开）。 |
| 向下箭头       | 将选区下移一项。                                |
| End        | 选区会移动到列表底端。                             |
| Esc        | 关闭下拉列表，并在测试模式下将焦点返回到组合框。                |
| Enter      | 关闭下拉列表，并将焦点返回到组合框。                      |
| Home       | 将选区移到列表顶端。                              |
| Page Down  | 将选区向下移动一页。                              |

| 键         | 说明          |
|-----------|-------------|
| Page Up   | 将选区向上移动一页。  |
| Shift+Tab | 将焦点移到前一个对象。 |
| Tab       | 将焦点移到下一个对象。 |

如果组合框的下拉列表具有焦点，按字母数字键就会沿下拉列表将选区上移和下移到下一个首字符相同的项目。您也可以使用下面的按键来控制下拉列表：

| 键          | 说明                                                          |
|------------|-------------------------------------------------------------|
| Ctrl+ 向上箭头 | 如果下拉列表处于打开状态，焦点就会返回到文本框，并且下拉列表将在独立和浏览器版本的 Flash Player 中关闭。 |
| 向下箭头       | 将选区下移一项。                                                    |
| End        | 将插入点移动到文本框的末尾。                                              |
| Enter      | 如果下拉列表处于打开状态，焦点就会返回到文本框，并且下拉列表会关闭。                          |
| Esc        | 如果下拉列表处于打开状态，焦点就会返回到文本框，并且下拉列表会在测试模式下关闭。                    |
| Home       | 将插入点移动到文本框的开头。                                              |
| Page Down  | 将选区向下移动一页。                                                  |
| Page Up    | 将选区向上移动一页。                                                  |
| Tab        | 将焦点移到下一个对象。                                                 |
| Shift+End  | 选中从插入点到末尾位置的文本。                                             |
| Shift+Home | 选择从插入点到开始位置的文本。                                             |
| Shift+Tab  | 将焦点移到前一个对象。                                                 |
| 向上箭头       | 将选区上移一个项目。                                                  |

提醒

Page Up 键和 Page Down 键使用的页的大小比可以显示的项数少一项。例如，在一个十行的下拉列表中向下翻页，将会依次显示第 0-9 项、第 9-18 项、第 18-27 项等等，每页都会有一个重叠项。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“FocusManager 类”或“创建自定义焦点导航”。

每个 **ComboBox** 组件实例在舞台上的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。然而，在实时预览中下拉列表并不打开，并且第一个项目会显示为选中项目。

在将 **ComboBox** 组件添加到应用程序时，您可以使用“辅助功能”面板，使其可由屏幕读取器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.ComboBoxAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。有关更多信息，请参见《使用 Flash》中的第 19 章“创建辅助内容”。

## 使用 ComboBox 组件

在任何需要从列表中选择一项的表单或应用程序中，您都可以使用 **ComboBox** 组件。例如，您可以在客户地址表单中提供一个州 / 省的下拉列表。对于比较复杂的情况，您可以使用可编辑的组合框。例如，在提供驾驶方向的应用程序中，您可以使用一个可编辑的组合框来让用户输入出发地址和目标地址。下拉列表可以包含用户以前输入过的地址。

### ComboBox 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 **ComboBox** 组件实例设置以下创作参数：

**data** 将一个数据值与 **ComboBox** 组件中的每一项相关联。该数据参数是一个数组。

**editable** 确定 **ComboBox** 组件是可编辑的 (*true*) 还是只是可选择的 (*false*)。默认值为 *false*。

**labels** 用一个文本值数组填充 **ComboBox** 组件。

**rowCount** 设置列表中最多可以显示的项数。默认值为 5。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **ComboBox** 组件实例设置以下附加参数：

**restrict** 指示用户可在组合框的文本字段中输入的字符集。默认值为 *undefined*。请参见第 179 页的“[ComboBox.restrict](#)”。

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 *true*。

**visible** 是一个布尔值，它指示对象是 (*true*) 否 (*false*) 可见。默认值为 *true*。



**minHeight** 和 **minWidth** 属性由内部的大小调整例程使用。它们在 **UIObject** 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以编写 **ActionScript**，通过利用 **ComboBox** 类的方法、属性和事件来设置 **ComboBox** 实例的其它选项。有关更多信息，请参见第 152 页的“[ComboBox 类](#)”。

## 创建具有 ComboBox 组件的应用程序

以下过程解释了如何在创作时将 **ComboBox** 组件添加到应用程序。在此示例中，组合框在其下拉列表内呈现一个从中选择城市的列表。

### 创建具有 ComboBox 组件的应用程序：

1. 将 **ComboBox** 组件从“组件”面板拖到舞台上。
2. 选择“变形”工具，并在舞台上调整该组件的大小。  
在创作过程中，组合框只能在舞台上调整大小。通常，您只需改变组合框的宽度以适应其条目。
3. 选择组合框，并在“属性”检查器中输入实例名称 **comboBox**。
4. 在“组件”检查器或“属性”检查器中，执行以下操作：
  - 输入 **Minneapolis**、**Portland** 和 **Keene** 作为标签参数。双击标签参数字段以打开“值”对话框。然后单击加号（+）以添加项目。
  - 输入 **MN.swf**、**OR.swf** 和 **NH.swf** 作为数据参数。  
这些是假想的 SWF 文件。例如，当用户在组合框中选择了个城市时，您就可以加载这些文件。

5. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
function change(evt){
 trace(evt.target.selectedItem.label);
}
comboBox.addEventListener("change", this);
```

最后一行代码将 `change` 事件处理函数添加到 **ComboBox** 实例。有关更多信息，请参见 [ComboBox.change](#)。

### 使用 ActionScript 创建 ComboBox 组件：

1. 将 **ComboBox** 组件从“组件”面板拖到当前文档的库中。

此操作将组件添加到库中，但不会在应用程序中显示。

2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
this.createClassObject(mx.controls.ComboBox, "my_cb", 10);

my_cb.addItem({data:1, label:"One"});
my_cb.addItem({data:2, label:"Two"});
```

此脚本使用第 1254 页的“[UIObject.createClassObject\(\)](#)”方法来创建 **ComboBox** 实例，然后使用第 157 页的“[ComboBox.addItem\(\)](#)”将列表项添加到 **ComboBox** 中。



3. 现在添加一个事件侦听器 and 事件处理函数，以在选取 **ComboBox** 项目时做出响应：

```
// 创建侦听器对象。
var cbListener:Object = new Object();
// 创建事件处理函数。
cbListener.change = function (evt_obj:Object) {
 trace("Currently selected item is: " +
 evt_obj.target.selectedItem.label);
}
// 添加事件侦听器。
my_cb.addEventListener("change", cbListener);
```

4. 选择 “控制” > “测试影片”，然后单击组合框中的某一项以在 “输出” 面板中显示一条消息。

## 自定义 ComboBox 组件

在创作时，您可以在水平和垂直方向调整 **ComboBox** 组件。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。

如果文本太长而不能在组合框中完全显示，文本将会被裁剪以适合组合框。您必须在创作时调整组合框的大小以适合标签文本。

在可编辑的组合框中，只有按钮是单击区，文本框不是。对于静态组合框，按钮和文本框一起组成单击区。单击区通过打开或关闭下拉列表来做出响应。

## 对 ComboBox 组件使用样式

您可以设置样式属性来更改 **ComboBox** 组件的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的 “使用样式自定义组件的颜色和文本”。

组合框有两个独有的样式：openDuration 和 openEasing。其它样式通过各自组件传递到组合框的按钮、文本框和下拉列表，如下所示：

- 该按钮是一个 **Button** 实例，并使用它自己的样式。（请参见第 87 页的 “对 **Button** 组件使用样式”。）
- 文本是一个 **TextInput** 实例，它使用自己的样式。（请参见第 1118 页的 “对 **TextInput** 组件使用样式”。）
- 下拉列表是一个 **List** 实例并使用它自己的样式。（请参见第 708 页的 “对 **List** 组件使用样式”。）

ComboBox 组件使用下列样式：

| 样式              | 主题    | 说明                                                                                                                                                                                        |
|-----------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| themeColor      | 光晕    | 组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。                                                                                                                  |
| backgroundColor | 光晕和范例 | 背景色。默认颜色为白色。                                                                                                                                                                              |
| borderStyle     | 光晕和范例 | Button 子组件将两个 RectBorder 实例用作其边框，并对在该类上定义的样式做出响应。请参见第 985 页的“RectBorder 类”。<br>在“光晕”主题中，ComboBox 组件将一个自定义圆角边框用于 ComboBox 的折叠部分。此部分 ComboBox 的颜色只能通过外观进行修改。请参见第 151 页的“对 ComboBox 组件使用外观”。 |
| color           | 光晕和范例 | 文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。                                                                                                                                                  |
| disabledColor   | 光晕和范例 | 组件禁用时的文本颜色。默认值为 0x848384（深灰）。                                                                                                                                                             |
| embedFonts      | 光晕和范例 | 一个布尔值，指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。                                            |
| fontFamily      | 光晕和范例 | 文本的字体名称。默认值为 "_sans"。                                                                                                                                                                     |
| fontSize        | 光晕和范例 | 字体的磅值。默认值为 10。                                                                                                                                                                            |
| fontStyle       | 光晕和范例 | 字体样式："normal" 或 "italic"。默认值为 "normal"。                                                                                                                                                   |
| fontWeight      | 光晕和范例 | 字体粗细："none" 或 "bold"。默认值为 "none"。在调用 setStyle() 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 getStyle() 的调用将返回 "none"。                                                                          |
| textAlign       | 光晕和范例 | 文本对齐方式："left"、"right" 或 "center"。默认值为 "left"。                                                                                                                                             |
| textDecoration  | 光晕和范例 | 文本修饰："none" 或 "underline"。默认值为 "none"。                                                                                                                                                    |
| openDuration    | 光晕和范例 | 过渡动画的持续时间（以毫秒为单位）。默认值为 250。                                                                                                                                                               |
| openEasing      | 光晕和范例 | 对控制动画的补间函数的引用。默认为正弦输入 / 输出。有关更多信息，请参见《使用组件》中的“自定义组件动画”。                                                                                                                                   |

以下示例演示如何使用 List 样式来控制 ComboBox 组件的下拉列表部分的行为。

```
// comboBox 为舞台上的 ComboBox 组件的一个实例。
comboBox.setStyle("alternatingRowColors", [0xFFFFFFFF, 0xBF8FBF]);
```

# 对 ComboBox 组件使用外观

ComboBox 组件使用库中的元件来表示按钮状态，并具有向下箭头的外观变量。这些外观位于 HaloTheme.fla 和 SampleTheme.fla 文件的 Flash UI Components 2/Themes/MMDefault/ComboBox Assets/States 文件夹中。下面的信息描述这些外观并提供对外观进行自定义的步骤。

ComboBox 组件还将滚动条外观用于下拉列表的滚动条，并将两个 RectBorder 类实例用于文本输入和下拉列表周围的边框。有关自定义这些外观的信息，请参见第 1283 页的“对 UI ScrollBar 组件使用外观”和第 985 页的“RectBorder 类”。有关可用于外观组件的方法的更多信息，请参见《使用组件》中的“关于设置组件外观”。

ComboBox 组件使用下列外观属性：

| 属性                         | 说明                                     |
|----------------------------|----------------------------------------|
| ComboDownArrowDisabledName | 向下箭头的禁用状态。默认值为 ComboDownArrowDisabled。 |
| ComboDownArrowDownName     | 向下箭头的按下状态。默认值为 ComboDownArrowDown。     |
| ComboDownArrowUpName       | 向下箭头的弹起状态。默认值为 ComboDownArrowOver。     |
| ComboDownArrowOverName     | 向下箭头的悬停状态。默认值为 ComboDownArrowUp。       |

## 创建 ComboBox 外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 HaloTheme.fla 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在操作系统上的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 Flash UI Components 2/Themes/MMDefault 文件夹并将 ComboBox Assets 文件夹拖动到文档的库中。
4. 在文档的库中展开 ComboBox Assets/States 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如，打开 ComboDownArrowDisabled 元件。
6. 按需要自定义元件。  
例如，将内部白色方框更改为浅灰色。
7. 对所有要自定义的元件重复步骤 5-6。
8. 单击“返回”按钮返回主时间轴。
9. 将 ComboBox 组件拖动到舞台上。
10. 根据需要设置 ComboBox 实例属性。  
对于本示例，使用 ActionScript 将 ComboBox 设置为禁用。
11. 选择“控制”>“测试影片”。

# ComboBox 类

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > ComboBoxBase > ComboBox

ActionScript 类名称 mx.controls.ComboBox

ComboBox 组件结合了三个单独的子组件：Button、TextInput 和 List。每个子组件的大多数方法、属性和事件都可以直接从 ComboBox 组件获得，并在 ComboBox 类的摘要表中列出。

组合框中的下拉列表是作为数组或数据提供程序提供的。如果您使用数据提供程序，列表会在运行时更改。通过切换到新的数组或数据提供程序，可以动态更改 ComboBox 数据的源。

组合框列表中的项目是按位置从数字 0 开始编排索引的。一个项目可以是以下内容之一：

- 原始数据类型。
- 一个对象，其中包含 label 属性和 data 属性



对象可能使用 `ComboBox.labelFunction` 或 `ComboBox.labelField` 属性来确定 label 属性。

如果该项目是原始数据类型而不是字符串，则会转换为字符串。如果某一项是一个对象，则其 label 属性必须是字符串，而 data 属性可以是任何 ActionScript 值。

您向其提供项的 `ComboBox` 方法有两个参数：label 和 data，它们指的就是上述属性。返回项目的方法会将该项目作为对象返回。

组合框会延迟其下拉列表的实例化，直到用户与其进行交互。因此，组合框在第一次使用时可能响应较慢。

使用下面的代码以编程方式访问 ComboBox 组件的下拉列表并覆盖延迟：

```
var foo = myComboBox.dropdown;
```

访问下拉列表可能会导致应用程序停顿。当用户第一次与组合框进行交互，或者当上面的代码在运行时，可能会发生这种情况。

# ComboBox 类的方法摘要

下表列出了 **ComboBox** 类的方法。

| 方法                                    | 说明                  |
|---------------------------------------|---------------------|
| <code>ComboBox.addItem()</code>       | 向列表的结尾添加项目。         |
| <code>ComboBox.addItemAt()</code>     | 向列表的结尾在指定的索引处添加项目。  |
| <code>ComboBox.close()</code>         | 关闭下拉列表。             |
| <code>ComboBox.getItemAt()</code>     | 返回指定索引处的项目。         |
| <code>ComboBox.open()</code>          | 打开下拉列表。             |
| <code>ComboBox.removeAll()</code>     | 删除列表中的所有项目。         |
| <code>ComboBox.removeItemAt()</code>  | 删除位于列表中指定位置的项目。     |
| <code>ComboBox.replaceItemAt()</code> | 替换位于指定索引处的项目的内容。    |
| <code>ComboBox.sortItems()</code>     | 使用比较函数对列表进行排序。      |
| <code>ComboBox.sortItemsBy()</code>   | 使用每个项目的一个字段对列表进行排序。 |

## 从 UIObject 类继承的方法

下表列出了 **ComboBox** 类从 **UIObject** 类继承的方法。从 **ComboBox** 对象调用这些方法时，请使用 `comboBoxInstance.methodName` 的形式。

| 方法                                        | 说明                               |
|-------------------------------------------|----------------------------------|
| <code>UIObject.createClassObject()</code> | 创建指定类的对象。                        |
| <code>UIObject.createObject()</code>      | 创建对象的子对象。                        |
| <code>UIObject.destroyObject()</code>     | 破坏组件实例。                          |
| <code>UIObject.doLater()</code>           | 在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。 |
| <code>UIObject.getStyle()</code>          | 从样式声明或对象获取样式属性。                  |
| <code>UIObject.invalidate()</code>        | 标记对象使其在到达下一个帧间隔时进行重绘。            |
| <code>UIObject.move()</code>              | 将对象移动到要求的位置。                     |
| <code>UIObject.redraw()</code>            | 强制验证对象，以便在当前帧中绘制该对象。             |
| <code>UIObject.setSize()</code>           | 将对象调整为所要求的大小。                    |
| <code>UIObject.setSkin()</code>           | 设置对象的外观。                         |
| <code>UIObject.setStyle()</code>          | 设置样式声明或对象的样式属性。                  |

## 从 UIComponent 类继承的方法

下表列出了 `ComboBox` 类从 `UIComponent` 类继承的方法。从 `ComboBox` 对象调用这些方法时，请使用 `comboBoxInstance.methodName` 的形式。

| 方法                                  | 说明             |
|-------------------------------------|----------------|
| <code>UIComponent.setFocus()</code> | 返回对具有焦点的对象的引用。 |
| <code>UIComponent.setFocus()</code> | 将焦点设置到组件实例中。   |

## ComboBox 类的属性摘要

下表列出了 `ComboBox` 类的属性。

| 属性                                  | 说明                                    |
|-------------------------------------|---------------------------------------|
| <code>ComboBox.dataProvider</code>  | 列表中项目的数据模型。                           |
| <code>ComboBox.dropdown</code>      | 返回一个对组合框所包含的 <code>List</code> 组件的引用。 |
| <code>ComboBox.dropdownWidth</code> | 下拉列表的宽度（以像素为单位）。                      |
| <code>ComboBox.editable</code>      | 指明组合框是否可以编辑。                          |
| <code>ComboBox.labelField</code>    | 指明使用哪个数据字段作为下拉列表的标签。                  |
| <code>ComboBox.labelFunction</code> | 指定一个用于计算下拉列表标签字段的函数。                  |
| <code>ComboBox.length</code>        | 只读；下拉列表的长度。                           |
| <code>ComboBox.restrict</code>      | 用户可在组合框的文本字段中输入的字符集。                  |
| <code>ComboBox.rowCount</code>      | 列表一次可以显示的最大项目数。                       |
| <code>ComboBox.selectedIndex</code> | 下拉列表中所选项目的索引。                         |
| <code>ComboBox.selectedItem</code>  | 下拉列表中所选项目的值。                          |
| <code>ComboBox.text</code>          | 文本框中文本的字符串。                           |
| <code>ComboBox.textField</code>     | 对组合框中 <code>TextInput</code> 组件的引用。   |
| <code>ComboBox.value</code>         | 文本框（可编辑）或下拉列表（静态）的值。                  |

## 从 UIObject 类继承的属性

下表列出了 `ComboBox` 类从 `UIObject` 类继承的属性。从 `ComboBox` 对象访问这些属性时，请使用 `comboBoxInstance.propertyName` 的形式。

| 属性                            | 说明                                     |
|-------------------------------|----------------------------------------|
| <code>UIObject.bottom</code>  | 只读；对象的底边缘位置（相对于其父对象的底边缘）。              |
| <code>UIObject.height</code>  | 只读；对象的高度，以像素为单位。                       |
| <code>UIObject.left</code>    | 只读；对象的左边缘（以像素为单位）。                     |
| <code>UIObject.right</code>   | 只读。对象的右边缘位置（相对于其父对象的右边缘）。              |
| <code>UIObject.scaleX</code>  | 一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。         |
| <code>UIObject.scaleY</code>  | 一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。         |
| <code>UIObject.top</code>     | 只读；对象上边缘的位置（相对于其父对象）。                  |
| <code>UIObject.visible</code> | 一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。 |
| <code>UIObject.width</code>   | 只读；对象的宽度，以像素为单位。                       |
| <code>UIObject.x</code>       | 只读；对象的左边缘（以像素为单位）。                     |
| <code>UIObject.y</code>       | 只读；对象的上边缘（以像素为单位）。                     |

## 从 UIComponent 类继承的属性

下表列出了 `ComboBox` 类从 `UIComponent` 类继承的属性。从 `ComboBox` 对象访问这些属性时，请使用 `comboBoxInstance.propertyName` 的形式。

| 属性                                | 说明                     |
|-----------------------------------|------------------------|
| <code>UIComponent.enabled</code>  | 指示组件是否可以接收焦点和输入。       |
| <code>UIComponent.tabIndex</code> | 一个数字，指示文档中组件的 Tab 键顺序。 |

# ComboBox 类的事件摘要

下表列出了 ComboBox 类的事件。

| 事件                                    | 说明                   |
|---------------------------------------|----------------------|
| <a href="#">ComboBox.change</a>       | 当组合框的值因用户交互操作而改变时广播。 |
| <a href="#">ComboBox.close</a>        | 当组合框的列表开始回缩时广播。      |
| <a href="#">ComboBox.enter</a>        | 当按下 Enter 键时广播。      |
| <a href="#">ComboBox.itemRollOut</a>  | 当指针滑离一个下拉列表项时广播。     |
| <a href="#">ComboBox.itemRollOver</a> | 当滑过下拉列表的一个项目时广播。     |
| <a href="#">ComboBox.open</a>         | 当下拉列表开始打开时广播。        |
| <a href="#">ComboBox.scroll</a>       | 当滚动下拉列表时广播。          |

## 从 UIObject 类继承的事件

下表列出了 ComboBox 类从 UIObject 类继承的事件。

| 事件                              | 说明                 |
|---------------------------------|--------------------|
| <a href="#">UIObject.draw</a>   | 当对象将要绘制它的图形时进行广播。  |
| <a href="#">UIObject.hide</a>   | 在对象的状态从可见变为不可见时广播。 |
| <a href="#">UIObject.load</a>   | 创建子对象时广播。          |
| <a href="#">UIObject.move</a>   | 移动了对象时广播。          |
| <a href="#">UIObject.resize</a> | 在调整对象大小后广播。        |
| <a href="#">UIObject.reveal</a> | 在对象的状态从不可见变为可见时广播。 |
| <a href="#">UIObject.unload</a> | 卸载子对象时广播。          |

## 从 UIComponent 类继承的事件

下表列出了 ComboBox 类从 UIComponent 类继承的事件。

| 事件                                   | 说明            |
|--------------------------------------|---------------|
| <a href="#">UIComponent.focusIn</a>  | 当对象收到焦点时进行广播。 |
| <a href="#">UIComponent.focusOut</a> | 当对象失去焦点时进行广播。 |
| <a href="#">UIComponent.keyDown</a>  | 当按下按键时进行广播。   |
| <a href="#">UIComponent.keyUp</a>    | 当松开按键时进行广播。   |



# ComboBox.addItem()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
comboBoxInstance.addItem(label[, data])
comboBoxInstance.addItem({label:label[, data:data]})
comboBoxInstance.addItem(obj);
```

## 参数

*label* 一个字符串，它指示新项的标签。

*data* 该项的数据，可以是任何数据类型。此参数是可选的。

*obj* 具有 *label* 属性和可选的 *data* 属性的对象。

## 返回

在其位置添加了项的索引。

## 说明

方法：在列表的结尾添加新项。

## 示例

在具有名为 `my_cb` 的 **ComboBox** 组件实例的情况下，将以下 **ActionScript** 添加到主时间轴的第一帧的“动作”面板。此 **ActionScript** 创建一个带有三项的 **ComboBox**；每一项有一个数据值和一个标签字符串。当您测试该 **SWF** 文件时，如果单击其中的某一项，“输出”面板会显示“目标”标识、数据值和标签：

```
// 将项添加到组合框。
my_cb.addItem("this is an Item");
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

// 添加事件侦听器 and 事件处理函数。
var cbListener:Object = new Object();
cbListener.change = function(evt_obj:Object):Void {
 var currentlySelected:Object = evt_obj.target.selectedItem;
 trace(evt_obj.target);
 trace("data: "+currentlySelected.data);
 trace("label: "+currentlySelected.label);
};
my_cb.addEventListener("change", cbListener);
```

# ComboBox.addItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
comboBoxInstance.addItemAt(index, label[, data])
comboBoxInstance.addItemAt(index, {label:label[, data:data]})
comboBoxInstance.addItemAt(index, obj);
```

## 参数

*index* 一个等于或大于 0 的数字，指示项的插入位置（新项的索引）。

*label* 一个字符串，它指示新项的标签。

*data* 该项的数据，可以是任何数据类型。此参数是可选的。

*obj* 一个具有 *label* 和 *data* 属性的对象。

## 返回

在其位置添加了项的索引。

## 说明

方法：在位于列表结尾的由 *index* 参数指定的索引处添加新项。大于 `ComboBox.length` 的索引将被忽略。

## 示例

有名为 `my_cb` 的 **ComboBox** 组件实例和名为 `my_btn` 的 **Button** 组件实例。在主时间轴的第一帧的“动作”面板中添加以下 **ActionScript**。当您测试该 SWF 文件时，单击组合框会看见其中有两项。然后单击按钮，下一次单击组合框时，会看到它添加了标签为“**first value**”的另一项：

```
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

var btnListener:Object = new Object();
btnListener.click = function() {
 my_cb.addItemAt(0, {data:1, label:"first value"});
};
my_btn.addEventListener("click", btnListener);
```

# ComboBox.change

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("change", listenerObject)
```

## 说明

事件：当 `ComboBox.selectedIndex` 或 `ComboBox.selectedItem` 属性因用户交互操作而改变时向所有已注册的侦听器广播。

组件实例 (*comboBoxInstance*) 使用调度程序 / 侦听器事件模型来调度一个事件（在本例中为 `change`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法：当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()`（请参见 [EventDispatcher.addEventListener\(\)](#)），以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下示例将生成了 `change` 事件的组件的实例名称发送到“输出”面板：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// 创建侦听器对象。
var cbListener:Object = new Object();

// 为侦听器对象分配函数。
cbListener.change = function(event_obj:Object) {
 trace("Value changed to: "+event_obj.target.selectedItem.label);
};
```

```
// 添加侦听器。
my_cb.addEventListener("change", cbListener);
```

#### 另请参见

[EventDispatcher.addEventListener\(\)](#)

## ComboBox.close()

#### 可用性

Flash Player 6 (6.0.79.0)。

#### 版本

Flash MX 2004。

#### 用法

```
comboBoxInstance.close()
```

#### 参数

无。

#### 返回

无。

#### 说明

方法；关闭下拉列表。

#### 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 和 **Button** 组件实例 `my_button` 的情况下，以下示例在单击 `my_button` 按钮时关闭 `my_cb` 组合框的下拉列表：

```
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

var btnListener:Object = new Object();
btnListener.click = function() {
 my_cb.close();
};
my_button.addEventListener("click", btnListener);
```

#### 另请参见

[ComboBox.open\(\)](#)

# ComboBox.close

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.close = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("close", listenerObject)
```

## 说明

事件：当组合框的下拉列表完全回缩时，向所有已注册的侦听器广播。

组件实例 (*comboBoxInstance*) 使用调度程序 / 侦听器事件模型调度一个事件（在本例中为 *close*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 *addEventListener()* 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在舞台上具有 **ComboBox** 组件实例 *my\_cb* 的情况下，以下示例在打开或关闭下拉列表时将消息发送到“输出”面板中：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// 创建侦听器对象。
var cbListener:Object = new Object();
cbListener.open = function(evt_obj:Object) {
 trace("The ComboBox has opened.");
}
cbListener.close = function(evt_obj:Object){
 trace("The ComboBox has closed.");
}

// 添加侦听器。
```

```
my_cb.addEventListener("open", cbListener);
my_cb.addEventListener("close", cbListener);
```

```
// 打开组合框。
my_cb.open();
```

另请参见

[EventDispatcher.addEventListener\(\)](#)

## ComboBox.dataProvider

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*comboBoxInstance.dataProvider*

### 说明

属性；在列表中查看的项目的数据模型。此属性的值可以是数组或任何实现 **DataProvider API** 的对象。默认值为 []。**List** 组件与 **ComboBox** 组件共享 `dataProvider` 属性，并且对此属性所做的更改立即可以用于这两个组件。

和其它支持数据的组件一样，**List** 组件会将方法添加到 **Array** 对象的原型中，以使它们符合 **DataProvider API**（有关更多信息，请参见 **DataProvider.as**）。因此，任何同时作为列表存在的数组都会自动具有作为列表的模型所需的所有方法（`addItem()`、`getItemAt()` 等），并可用于向多个组件广播模型更改。

如果数组包含对象，则会访问 `labelField` 或 `labelFunction` 属性以确定要显示项的哪些部分。默认值是 "label"，因此，如果存在这样的字段，就会选择它来进行显示；如果不存在，就会显示用逗号分隔的所有字段的列表。



如果该数组在每个索引处都包含字符串，而不包含对象，该列表就无法对项目进行排序和保持选定状态。进行任何排序都会导致丢失所做的选择。

任何实现 **DataProvider API** 的实例都可以作为 **List** 组件的数据提供程序。这包括 **Flash Remoting RecordSet** 对象、**Firefly DataSet** 组件等等。

## 示例

此示例使用字符串数组来填充 **ComboBox** 组件实例 `my_cb` 的下拉列表：

```
my_cb.dataProvider = [{data:1, label:"First Item"}, {data:2, label:"Second Item"}];
/* 等同于
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
*/
```

此示例创建一个数据提供程序数组，并将其指定给 `dataProvider` 属性：

```
var myDP:Array = new Array();
list.dataProvider = myDP;

for (var i:Number = 0; i < accounts.length; i++) {
 // 对 DataProvider 的这些更改将广播到列表。
 myDP.addItem({label: accounts[i].name,
 data: accounts[i].accountID});
}
```

# ComboBox.dropdown

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.dropdown

## 说明

属性（只读）：返回对组合框所包含的列表的引用。组合框中的 **List** 子组件在需要显示前不实例化。但是，当访问 `dropdown` 属性时，将创建该列表。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 并且库中具有两个影片剪辑元件（其链接 ID 值分别设置为 `dw_id` 和 `fl_id`）的情况下，以下 **ActionScript** 使用 `dropdown` 属性为下拉列表中的每一项添加图标：

```
// 设置下拉列表宽度以容纳标签大小。
my_cb.dropdownWidth = 200;

// 设置 ComboBox 的 dropdown 属性中的 iconField 样式。
// dropdown 属性是对 ComboBox 中的 List 组件的引用。
```

```
// 因此可以为 CB 设置 List 样式。
my_cb.dropdown.setStyle("iconField", "pIcon");

// 将项添加到列表。
my_cb.addItem({label:"Dreamweaver 1", pIcon:"dw_id"});
my_cb.addItem({label:"Flash 1", pIcon:"fl_id"});
my_cb.addItem({label:"Flash 2", pIcon:"fl_id"});
```

另请参见

[ComboBox.dropdownWidth](#)

## ComboBox.dropdownWidth

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*comboBoxInstance*.dropdownWidth

说明

属性：下拉列表的宽度限制（以像素为单位）。默认值是 **ComboBox** 组件（**TextInput** 实例加上 **SimpleButton** 实例）的宽度。

示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下 **ActionScript** 设置下拉列表宽度以容纳标签：

```
// 设置下拉列表宽度以容纳标签大小。
my_cb.dropdownWidth = 200;

// 将项添加到列表。
my_cb.addItem("ComboBox");
my_cb.addItem({data:2, label:"This is a long label"});
my_cb.addItem({data:3, label:"This has an even longer label"});
```

另请参见

[ComboBox.dropdown](#)



# ComboBox.editable

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.editable

## 说明

属性；指示组合框是 (true) 否 (false) 可编辑。在可编辑组合框中，用户可以在文本框中输入值，这些值不显示在下拉列表中。如果组合框是不可编辑的，则无法在文本框中输入文本。文本框显示列表中的项目的文本。默认值为 false。

将组合框设置为可编辑会清除组合框的文本字段。还会将所选的索引（以及项）设置为 undefined。若要使组合框可编辑且仍保留所选项，请使用下面的代码：

```
var ix:Number = myComboBox.selectedIndex;
myComboBox.editable = true; // 清除文本字段。
myComboBox.selectedIndex = ix; // 将标签复制回文本字段中。
```

## 示例

在舞台上具有 **ComboBox** 组件实例 my\_cb 的情况下，以下 **ActionScript** 创建一个组合框列表和两个侦听器。第一个侦听器处理的是对“Add new item”（添加新项）标签的单击操作，以使组合框字段可编辑。第二个侦听器处理的是用户按 **Enter** 键的操作，以向组合框列表中添加项：

```
// 将项添加到组合框列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:-1, label:"Add new item..."});

// 对用户单击“Add new item”（添加新项）做出响应。
function changeListener(evt_obj:Object) {
 if (evt_obj.target.selectedItem.data == -1) {
 evt_obj.target.editable = true;
 } else if (evt_obj.target.selectedIndex != undefined) {
 evt_obj.target.editable = false;
 evt_obj.target.setFocus();
 }
}
my_cb.addEventListener("change", changeListener);

// 对用户添加一个新项名称后按 Enter 键做出响应。
```

```
function enterListener(evt_obj:Object) {
 if (evt_obj.target.value != '') {
 evt_obj.target.addItem({data:'', label:evt_obj.target.value});
 }
 evt_obj.target.editable = false;
 evt_obj.target.selectedIndex = evt_obj.target.dataProvider.length-1;
 evt_obj.target.setFocus();
}
my_cb.addEventListener("enter", enterListener);
```

## ComboBox.enter

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.enter = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("enter", listenerObject)
```

### 说明

事件：当用户在文本框中按下 **Enter** 键时，向所有已注册的侦听器广播。此事件是一个仅从可编辑组合框广播的 **TextInput** 事件。有关更多信息，请参见 [TextInput.enter](#)。

组件实例 (*comboBoxInstance*) 使用调度程序 / 侦听器事件模型发送一个事件（在本例中为 *enter*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 *addEventListener()* 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下 **ActionScript** 创建一个组合框列表和两个侦听器。第一个侦听器处理的是对 “Add new item”（添加新项）标签的单击操作，以使组合框字段可编辑。第二个侦听器处理的是用户按 **Enter** 键的操作以向组合框列表中添加项：

```
// 将项添加到组合框列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:-1, label:"Add new item..."});

// 对用户单击 “Add new item”（添加新项）做出响应。
function changeListener(evt_obj:Object) {
 if (evt_obj.target.selectedItem.data == -1) {
 evt_obj.target.editable = true;
 } else if (evt_obj.target.selectedIndex != undefined) {
 evt_obj.target.editable = false;
 evt_obj.target.setFocus();
 }
}
my_cb.addEventListener("change", changeListener);

// 对用户添加一个新项名称后按 Enter 键做出响应。
function enterListener(evt_obj:Object) {
 if (evt_obj.target.value != '') {
 evt_obj.target.addItem({data:'', label:evt_obj.target.value});
 }
 evt_obj.target.editable = false;
 evt_obj.target.selectedIndex = evt_obj.target.dataProvider.length-1;
 evt_obj.target.setFocus();
}
my_cb.addEventListener("enter", enterListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# ComboBox.getItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
comboBoxInstance.getItemAt(index)
```

## 参数

*index* 要检索的项的索引。该索引必须是大于或等于 0 并且小于 `ComboBox.length` 的值的编号。

## 返回

被索引的项目对象或值。如果索引超出范围，该值就会是未定义的。

## 说明

方法；检索所指定索引处的项目。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下 **ActionScript** 在“输出”面板中显示第一个组合框项的标签：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

trace(my_cb.getItemAt(1).label);
```

# ComboBox.itemRollOut

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.itemRollOut = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("itemRollOut", listenerObject)
```

## 事件对象

除了事件对象的标准属性外，itemRollOut 事件还有一个 index 属性。该索引为指针已滑过的项的编号。

## 说明

事件：当指针滑离下拉列表项时，向所有已注册的侦听器广播。这是一个从组合框广播的 List 事件。有关更多信息，请参见 [List.itemRollOut](#)。

组件实例 (comboBoxInstance) 使用调度程序 / 侦听器事件模型调度一个事件（在本例中为 itemRollOut），而该事件由您创建的侦听器对象 (listenerObject) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (eventObject) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“EventDispatcher 类”。

最后，对广播该事件的组件实例调用 addEventListener() 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

## 示例

在舞台上具有 ComboBox 实例 my\_cb 的情况下，以下 **ActionScript** 在指针滑入或滑出某一项时将指示该项索引和事件的消息发送到“输出”面板：

```
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// 创建侦听器对象。
var cbListener:Object = new Object();
cbListener.itemRollOver = function(evt_obj:Object) {
 trace("index: "+evt_obj.index+", event: "+evt_obj.type);
};
```

```
cbListener.itemRollOut = function(evt_obj:Object) {
 trace("index: "+evt_obj.index+", event: "+evt_obj.type);
};
```

// 添加侦听器。

```
my_cb.addEventListener("itemRollOver", cbListener);
my_cb.addEventListener("itemRollOut", cbListener);
```

### 另请参见

[ComboBox.itemRollOver](#)、[EventDispatcher.addEventListener\(\)](#)

## ComboBox.itemRollOver

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.itemRollOver = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("itemRollOver", listenerObject)
```

### 事件对象

除了事件对象的标准属性外，`itemRollOver` 事件还有一个 `index` 属性。该索引为指针已滑过的项的数量。

### 说明

事件：当鼠标指针滑过下拉列表项时，向所有已注册的侦听器广播。这是一个从组合框广播的 `List` 事件。有关更多信息，请参见 [List.itemRollOver](#)。

组件实例 (`comboBoxInstance`) 使用调度程序 / 侦听器事件模型调度一个事件（在本例中为 `itemRollOver`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

## 示例

在舞台上具有 **ComboBox** 实例 `my_cb` 的情况下，以下 **ActionScript** 在指针滑入或滑出某一项时将指示该项索引和事件的消息发送到“输出”面板：

```
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// 创建侦听器对象。
var cbListener:Object = new Object();
cbListener.itemRollOver = function(evt_obj:Object) {
 trace("index: " + evt_obj.index + ", event: " + evt_obj.type);
};
cbListener.itemRollOut = function(evt_obj:Object) {
 trace("index: " + evt_obj.index + ", event: " + evt_obj.type);
};

// 添加侦听器。
my_cb.addEventListener("itemRollOver", cbListener);
my_cb.addEventListener("itemRollOut", cbListener);
```

## 另请参见

[ComboBox.itemRollOut](#)、[EventDispatcher.addEventListener\(\)](#)

# ComboBox.labelField

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.labelField

## 说明

属性；dataProvider 数组对象中要用作标签字段的字段名称。这是 **List** 组件的一个属性，可以从 **ComboBox** 组件实例中使用。有关更多信息，请参见 [List.labelField](#)。

默认值为 undefined。

## 示例

以下示例将 `dataProvider` 属性设置为一个字符串数组并设置 `labelField` 属性，以指示应将 `name` 字段用作下拉列表的标签：

```
my_cb.dataProvider = [
 {name:"Gary", gender:"male"},
 {name:"Susan", gender:"female"}];

my_cb.labelField = "name";
```

## 另请参见

[List.labelFunction](#)

# ComboBox.labelFunction

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.labelFunction

## 说明

属性：一个计算数据提供程序项目的标签的函数。必须定义该函数。默认值为 `undefined`。

## 示例

下面的范例创建了一个数据提供程序，然后定义一个函数以指定用作下拉列表标签的内容：

```
myComboBox.dataProvider = [
 {firstName:"Nigel", lastName:"Pegg", age:"really young"},
 {firstName:"Gary", lastName:"Grossman", age:"young"},
 {firstName:"Chris", lastName:"Walcott", age:"old"},
 {firstName:"Greg", lastName:"Yachuk", age:"really old"}];

myComboBox.labelFunction = function(itemObj){
 return (itemObj.lastName + ", " + itemObj.firstName);
}
```

## 另请参见

[List.labelField](#)



# ComboBox.length

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance.length*

## 说明

属性（只读）：下拉列表的长度。这是 **List** 组件的一个属性，可以从 **ComboBox** 实例中使用。有关更多信息，请参见 [List.length](#)。默认值为 0。

## 示例

以下示例将 `length` 的值存储到一个变量中：

```
var dropdownItemCount:Number = myComboBox.length;
```

# ComboBox.open()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance.open()*

## 参数

无。

## 返回

无。

## 说明

方法：打开下拉列表。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 和 **Button** 组件实例 `my_button` 的情况下，以下示例在单击 `my_button` 按钮时打开 `my_cb` 组合框的下拉列表：

```
my_cb.addItem({data:2, label:"second value"});
my_cb.addItem({data:3, label:"third value"});

var btnListener:Object = new Object();
btnListener.click = function() {
 my_cb.open();
};
my_button.addEventListener("click", btnListener);
```

## 另请参见

[ComboBox.close\(\)](#)

# ComboBox.open

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.open = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("open", listenerObject)
```

## 说明

事件：当完全打开下拉列表时，向所有已注册的侦听器广播。

组件实例 (*comboBoxInstance*) 使用调度程序 / 侦听器事件模型调度一个事件（在本例中为 `open`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下示例在打开或关闭下拉列表时将消息发送到“输出”面板中：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// 创建侦听器对象。
var cbListener:Object = new Object();
cbListener.open = function(evt_obj:Object) {
 trace("The ComboBox has opened.");
}
cbListener.close = function(evt_obj:Object){
 trace("The ComboBox has closed.");
}

// 添加侦听器。
my_cb.addEventListener("open", cbListener);
my_cb.addEventListener("close", cbListener);

// 打开组合框。
my_cb.open();
```

## 另请参见

[ComboBox.close](#)、[EventDispatcher.addEventListener\(\)](#)

# ComboBox.removeAll()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
comboBoxInstance.removeAll()
```

## 参数

无。

## 返回

无。

## 说明

方法：删除列表中的所有项。这是 **List** 组件的一个方法，可以从 **ComboBox** 组件的实例中获得。

## 示例

在舞台上具有 **ComboBox** 实例 `my_cb` 和 **Button** 组件实例 `clear_button` 的情况下，以下 **ActionScript** 将组合框和按钮并排放置。当单击组合框时，会看到一个项目列表。当单击按钮时，会清除组合框的项目：

```
my_cb.move(10, 10);
clear_button.move(120, 10);

// 创建 dataprovider。
var myDP_array:Array = new Array();
myDP_array.push({data:1, label:"First Item"});
myDP_array.push({data:2, label:"Second Item"});

my_cb.dataProvider = myDP_array;

// 定义事件侦听器对象。
var clearListener:Object = new Object();
clearListener.click = function(evt_obj:Object){
 my_cb.removeAll();
}

// 添加侦听器。
clear_button.addEventListener("click", clearListener);
```

## 另请参见

[ComboBox.removeItemAt\(\)](#)、[ComboBox.replaceItemAt\(\)](#)

# ComboBox.removeItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
comboBoxInstance.removeItemAt(index)
```

## 参数

*index* 一个数字，指示要删除的项的位置。该索引是从零开始的。

## 返回

一个对象；已删除的项目（如果项不存在，则它为 `undefined`）。

## 说明

方法：删除指定索引位置处的项目。`index` 参数所指示的索引后面的列表索引会折叠一项。这是 **List** 组件的一个方法，可以从 **ComboBox** 组件的实例中获得。

## 示例

在舞台上具有 **ComboBox** 实例 `my_cb` 和 **Button** 组件实例 `clear_button` 的情况下，以下 **ActionScript** 将组合框和按钮并排放置。当单击组合框时，会看到一个有两个项目的列表。当单击按钮时，会清除组合框的第二项（位于索引位置 1，因为索引值是从零开始的）：

```
my_cb.move(10, 10);
clear_button.move(120, 10);

// 创建 dataprovider。
var myDP_array:Array = new Array();
myDP_array.push({data:1, label:"First Item"});
myDP_array.push({data:2, label:"Second Item"});

my_cb.dataProvider = myDP_array;

// 定义事件侦听器对象。
var clearListener:Object = new Object();
clearListener.click = function(evt_obj:Object){
 my_cb.removeItemAt(1);
}

// 添加侦听器。
clear_button.addEventListener("click", clearListener);
```

## 另请参见

[ComboBox.removeAll\(\)](#)、[ComboBox.replaceItemAt\(\)](#)

# ComboBox.replaceItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
comboBoxInstance.replaceItemAt(index, label[, data])
comboBoxInstance.replaceItemAt(index, {label:label[, data:data]})
comboBoxInstance.replaceItemAt(index, obj);
```

## 参数

*index* 一个等于或大于 0 的数字，指示项的插入位置（新项的索引）。

*label* 一个字符串，它指示新项的标签。

*data* 项的数据。此参数是可选的。

*obj* 一个具有 *label* 和 *data* 属性的对象。

## 返回

无。

## 说明

方法：替换位于指定索引处的项目的内容。它是 **List** 组件的一个方法，可以从 **ComboBox** 组件中获得。

## 示例

在舞台上具有 **ComboBox** 组件实例 *my\_cb* 和 **TextInput** 组件实例 *label\_ti* 的情况下，以下 **ActionScript** 代码在用户按 **Enter** 键时将用户输入添加到组合框中：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});

// 创建用户在“文本输入”字段上按 Enter 键的操作的侦听器。
var tiListener:Object = new Object();
tiListener.enter = function(evt_obj:Object) {
 my_cb.replaceItemAt(my_cb.selectedIndex, {label:evt_obj.target.text});
 // 刷新最近修改的 ComboBox 项所需的代码
 my_cb.selectedIndex = my_cb.selectedIndex;
};
label_ti.addEventListener("enter", tiListener);
```

## 另请参见

[ComboBox.removeAll\(\)](#)、[ComboBox.removeItemAt\(\)](#)

# ComboBox.restrict

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance.restrict*

## 说明

属性；指明用户可在组合框的文本字段中输入的字符集。默认值为 `undefined`。如果此属性为 `null` 或空字符串 (`"`)，则用户可以输入任何字符。如果此属性为一个字符串，则用户只能输入该字符串中的字符；该字符串是从左向右扫描的。可以使用短划线 (`-`) 指定一个范围。

如果字符串以插入符号 (`^`) 开头，则该插入符号后跟随的所有字符都将被视为不可接受的字符。如果字符串不以插入符号开头，则该字符串中的字符都将视为可接受。

可以使用反斜杠 (`\`) 来输入连字符 (`-`)、插入符号 (`^`) 或反斜杠 (`\`) 字符，如下所示：

```
\^
\-
\\
```

在“动作”面板中，当在双引号内输入反斜杠字符时，对于“动作”面板的双引号解释器，该字符有特殊的含义。它表示反斜杠后面的字符应视为“原样”。例如，可以使用以下代码来输入一个单引号：

```
var leftQuote = "\'";
```

“动作”面板的 **restrict** 解释器也将反斜杠用作转义符。因此，您可能会认为下列代码应该起作用：

```
myText.restrict = "0-9\-\^\\";
```

然而，由于此表达式包含在双引号中，因此值 `0-9-\^\` 会发送到 **restrict** 解释器，但 **restrict** 解释器无法识别此值。

因为必须在双引号中输入此表达式，所以不仅必须为 **restrict** 解释器提供该表达式，还必须转义该表达式，以便该表达式能够由“动作”面板中双引号的内置解释器正确阅读。若要将值 `0-9\-\^\` 发送到 **restrict** 解释器，您必须输入以下代码：

```
myCombo.restrict = "0-9\\-\\^\\\\\";
```

**restrict** 属性只限制用户交互；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。

## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下 **ActionScript** 将字符条目限制为数字 **0-9**、短划线和点：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
```

```
// 启用组合框的编辑。
my_cb.editable = true;
```

```
// 限制可以在组合框中输入的字符。
my_cb.restrict = "0-9\\-\\.\\ \\ ";
```

在以下示例中，第一行代码将文本字段限定为大写字母、数字和空格。第二行代码允许除小写字母之外的所有字符。

```
my_combo.restrict = "A-Z 0-9";
my_combo.restrict = "^a-z";
```

以下代码允许用户在实例 `myCombo` 中输入字符 `"0 1 2 3 4 5 6 7 8 9 - ^ \"`。您必须使用双反斜杠来转义字符 `-`、`^` 和 `\`。第一个 `\` 转义双引号，而第二个 `\` 指示解释器不应将下一个字符视为特殊字符。

```
myCombo.restrict = "0-9\\-\\.\\^\\\\\\\"";
```

# ComboBox.rowCount

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.rowCount

## 说明

属性：组合框插入滚动条前，在下拉列表中可见的最大行数。默认值为 **5**。

如果下拉列表中的项数大于 `rowCount` 属性，则该列表会调整大小，并根据需要显示滚动条。

如果下拉列表中包含的项数小于 `rowCount` 属性，则它会调整为列表中的项数。

这种行为与 **List** 组件的行为不同，**List** 组件会始终显示 `rowCount` 属性所指定的行数，即使有些地方出现空白也是如此。

如果该值是负的或小数，则该行为是未定义的。



## 示例

在舞台上具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下 **ActionScript** 设置组合框以显示前三项，然后添加一个滚动条以显示第四项：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});
```

```
// 如果 ComboBox 有 3 个以上的项，则显示滚动条。
my_cb.rowCount = 3;
```

# ComboBox.scroll

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
 // 在此处插入您的代码。
};
comboBoxInstance.addEventListener("scroll", listenerObject);
```

## 事件对象

除了标准的事件对象属性之外，滚动事件还有另一个 `direction` 属性。它是一个字符串，有两个可能的值："horizontal" 和 "vertical"。对于 **ComboBox scroll** 事件，该值始终是 "vertical"。

## 说明

事件；当滚动下拉列表时，向所有已注册的侦听器广播。这是一个 **List** 组件事件，可以用于 **ComboBox** 组件。

组件实例 (*comboBoxInstance*) 使用调度程序 / 侦听器事件模型调度一个事件（在本例中为 `scroll`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“**EventDispatcher** 类”。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

### 示例

在具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下示例将指示列表滚动到的项索引的消息发送到“输出”面板中：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});

// 如果 ComboBox 有 2 个以上的项，则显示滚动条。
my_cb.rowCount = 3;

// 创建侦听器对象。
var cbListener:Object = new Object();
cbListener.scroll = function(evt_obj:Object) {
 trace("The list had been scrolled to item # "+evt_obj.position);
};

// 添加侦听器。
my_cb.addEventListener("scroll", cbListener);
```

### 另请参见

[EventDispatcher.addEventListener\(\)](#)

## ComboBox.selectedIndex

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*comboBoxInstance*.selectedIndex

### 说明

属性；下拉列表中所选项目的索引号。默认值为 **0**。给此属性赋值会清除当前的选择，而选择指定项目，并在组合框的文本框中显示该项目的标签。

如果为此属性分配一个超出范围的值，则 **Flash** 会将其忽略。在可编辑组合框的文本字段中输入文本会将 `selectedIndex` 设置为 `undefined`。

## 示例

在具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下代码选择列表的最后一项（否则，它在默认情况下将显示第一项）：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});

// 选择列表上的最后一项。
my_cb.selectedIndex = my_cb.length-1;
```

## 另请参见

[ComboBox.selectedItem](#)

# ComboBox.selectedItem

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.selectedItem

## 说明

属性；下拉列表中所选项目的值。

如果组合框是可编辑的，则用户在文本框中输入任何文本时，`selectedItem` 都将返回 `undefined`。如果您从下拉列表中选择一个项目，或者使用 **ActionScript** 设置该值，则该属性只会有一个值。如果组合框是静态的，则 `selectedItem` 的值始终有效；如果列表中不存在任何项，则该属性返回 `undefined`。

## 示例

在具有 **ComboBox** 组件实例 `my_cb` 的情况下，以下示例显示 `selectedItem` 数据值和标签属性：

```
// 将项添加到列表。
my_cb.addItem({data:1, label:"First Item"});
my_cb.addItem({data:2, label:"Second Item"});
my_cb.addItem({data:3, label:"Third Item"});
my_cb.addItem({data:4, label:"Fourth Item"});
```

```

var cbListener:Object = new Object();
cbListener.change = function(evt_obj:Object) {
 var item_obj:Object = my_cb.selectedItem;
 var i:String;
 for (i in item_obj) {
 trace(i + ":\t" + item_obj[i]);
 }
 trace("");
};
my_cb.addEventListener("change", cbListener);

```

### 另请参见

[ComboBox.dataProvider](#)、[ComboBox.selectedIndex](#)

## ComboBox.sortItems()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*comboBoxInstance.sortItems([compareFunc], [optionsFlag])*

### 参数

*compareFunc* 一个指向函数的引用，该函数比较两个项以确定它们的排序顺序。有关详细信息，请参见《ActionScript 2.0 语言参考》中的 `Array.sort()`。此参数是可选的。

*optionsFlag* 使您不必复制整个数组或者反复对其进行重新排序即可对单个数组执行不同类型的多次排序。此参数是可选的。

以下是 *optionsFlag* 的可能值：

- `Array.DESENDING`，按从高到低的顺序排序。
- `Array.CASEINSENSITIVE`，忽略大小写进行排序。
- `Array.NUMERIC`，如果进行比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（如果指定该标志，字符串比较可以不区分大小写）。
- `Array.UNIQUESORT`，如果数组中有两个对象相同或具有相同的排序字段，则返回错误代码 (0)，而不是排序后的数组。

- `Array.RETURNINDEXEDARRAY`，返回作为排序结果的整数索引数组。例如，对于下面的数组，将返回第二行代码，并且该数组保持不变：

```
["a", "d", "c", "b"]
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 **3** 和 **1**：

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

## 返回

无。

## 说明

方法；依据指定的比较函数或指定的排序选项对组合框中的项目进行排序。

## 示例

此示例根据大写标签进行排序。项 `a` 和 `b` 被传递到该函数，并包含 `label` 和 `data` 字段：

```
myComboBox.sortItems(upperCaseFunc);
function upperCaseFunc(a,b){
 return a.label.toUpperCase() > b.label.toUpperCase();
}
```

以下示例使用上面定义的 `upperCaseFunc()` 函数以及 `optionsFlag` 参数对名为 `myComboBox` 的 **ComboBox** 实例的元素进行排序：

```
myComboBox.addItem("Mercury");
myComboBox.addItem("Venus");
myComboBox.addItem("Earth");
myComboBox.addItem("planet");
myComboBox.sortItems(upperCaseFunc, Array.DESENDING);
// myComboBox 的进行排序后的顺序将为：
// Venus
// planet
// Mercury
// Earth
```

# ComboBox.sortItemsBy()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
comboBoxInstance.sortItemsBy(fieldName, order [optionsFlag])
```

## 参数

*fieldName* 一个字符串，它指定要用于排序的字段的名称。通常情况下，此值为 "label" 或 "data"。

*order* 一个字符串，它指定是以升序 ("ASC") 还是以降序 ("DESC") 对项进行排序。

*optionsFlag* 使您不必复制整个数组或者反复对其进行重新排序即可对单个数组执行不同类型的多次排序。此参数是可选的，但如果使用了此参数，则应替换 *order* 参数。

以下是 *optionsFlag* 的可能值：

- `Array.DECENDING`，按从高到低的顺序排序。
- `Array.CASEINSENSITIVE`，忽略大小写进行排序。
- `Array.NUMERIC`，如果进行比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（如果指定该标志，字符串比较可以不区分大小写）。
- `Array.UNIQUESORT`，如果数组中有两个对象相同或具有相同的排序字段，则返回错误代码 (0)，而不是排序后的数组。
- `Array.RETURNINDEXEDARRAY`，返回作为排序结果的整数索引数组。例如，对于下面的数组，将返回第二行代码，并且该数组保持不变：

```
["a", "d", "c", "b"]
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 3 和 1：

```
array.sort (Array.NUMERIC | Array.DECENDING)
```

## 返回

无。

## 说明

方法：按字母或数值顺序、按指定的顺序、使用指定的字段名对组合框中的项目进行排序。如果 *fieldName* 项既包括文本字符串也包括整数，则首先列出整数项。*fieldName* 参数通常为 "label" 或 "data"，但高级编程人员可能会指定任何原始值。您可以根据需要使用 *optionsFlag* 参数来指定排序样式。

## 示例

以下示例基于一个名为 myComboBox 的 **ComboBox** 实例，该实例包含四个分别标记为 "Apples"、"Bananas"、"cherries" 和 "Grapes" 的元素：

```
// 首先，用这些元素填充 ComboBox。
myComboBox.addItem("Bananas");
myComboBox.addItem("Apples");
myComboBox.addItem("cherries");
myComboBox.addItem("Grapes");

// 以下语句使用设置为“ASC”的 order 参数进行排序，
// 并生成一个将“cherries”置于列表底端的排序，
// 因为该排序区分大小写。
myComboBox.sortItemsBy("label", "ASC");
// 排序后的顺序：Apples、Bananas、Grapes、cherries

// 以下示例使用设置为“DESC”的 order 参数进行排序，
// 并生成一个将“cherries”置于列表顶端的排序，
// 因为该排序区分大小写。
myComboBox.sortItemsBy("label", "DESC");
// 排序后的顺序：cherries、Grapes、Bananas、Apples

// 以下语句使用设置为 Array.CASEINSENSITIVE 的 optionsFlag 参数进行排序。
// 注意，升序排序为默认设置。
myComboBox.sortItemsBy("label", Array.CASEINSENSITIVE);
// 排序后的顺序：Apples、Bananas、cherries、Grapes

// 以下语句使用设置为 Array.CASEINSENSITIVE | Array.DECENDING 的 optionsFlag 参
// 数
// 进行排序。
myComboBox.sortItemsBy("label", Array.CASEINSENSITIVE | Array.DECENDING);
// 排序后的顺序：Grapes、cherries、Bananas、Apples
```

# ComboBox.text

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.text

## 说明

属性；文本框中的文本。您可以为可编辑的组合框获取和设定该值。对于静态的组合框，该值是只读的。

## 示例

以下示例设置了可编辑组合框的当前 text 值：

```
my_cb.addItem("Arkansas");
my_cb.addItem("Georgia");
```

```
my_cb.editable = true;
my_cb.text = "California";
```

# ComboBox.textField

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance*.textField

## 说明

属性（只读）；对 **ComboBox** 组件包含的 **TextInput** 组件的引用。

使用该属性，您可以访问基础 **TextInput** 组件，这样您就可以操作该组件。例如，您可能需要更改文本框的选定内容或者限制可以输入到文本框中的字符。



## 示例

以下代码会限制 myComboBox 的文本框，使其最多只接受 6 个字符的数字：

```
// 将项添加到列表。
my_cb.addItem({data:0xFFFFFFFF, label:"white"});
my_cb.addItem({data:0x000000, label:"black"});

my_cb.editable = true;

// 限制只能在文本字段中输入 0-9。
my_cb.restrict = "0-9";

// 限制最多只能输入 6 个字符。
my_cb.textField.maxChars = 6;
```

# ComboBox.value

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*comboBoxInstance.value*

## 说明

只读属性；如果组合框是可编辑的，则 value 返回项标签。如果组合框是静态的，则 value 返回项数据。

## 示例

以下示例通过设置 dataProvider 属性，将数据输入组合框。然后，它会在“输出”面板中显示 value。最后，它选择 "California" 并在组合框可编辑时在“输出”面板中显示它，然后在组合框不可编辑时显示 "CA"。

```
my_cb.dataProvider = [
 {label:"Alaska", data:"AK"},
 {label:"California", data:"CA"},
 {label:"Washington", data:"WA"}];
my_cb.editable = true;
my_cb.selectedIndex = 1;
trace('Editable value is "California": ' + my_cb.value);
my_cb.editable = false;
my_cb.selectedIndex = 1;
trace('Non-editable value is "CA": ' + my_cb.value);
```



# 数据绑定类（仅限 Flash Professional）

数据绑定类为 Flash Professional 8 中的数据绑定功能提供了运行时功能。您可以使用“组件”检查器中的“绑定”选项卡在 Flash 创作环境中直观地创建和配置数据绑定，也可以使用 `mx.data.binding` 包中的类以编程方式创建和配置绑定。

有关数据绑定的概述以及如何在 Flash 创作工具中直观地创建数据绑定的说明，请参见《使用 Flash》中的“数据绑定（仅限 Flash Professional）”。

## 使数据绑定类在运行时可用 （仅限 Flash Professional）

要编译 SWF 文件，您的库必须包含 SWC 文件，并且这些文件中包含数据绑定类和 Web 服务类的字节代码。如果创作时在 Flash 中创建数据绑定，则相关的组件类会自动添加到库中。如果在运行时使用数据绑定和 Web 服务，则必须将这些类添加到 FLA 文件的库中。您可以从“类”公用库获取这些 SWC 文件。

**要将 SWC 文件添加到您的库中：**

1. 选择“类”库（“窗口” > “公用库” > “类”）。
2. 打开您的文档的库（“窗口” > “库”）。
3. 将相应的 SWC 文件（DataBindingClasses、WebServiceClasses 或两者）从“类”库拖动到文档的库中。

有关这些类的更多信息，请参见第 192 页的“Binding 类（仅限 Flash Professional）”和第 1299 页的“Web 服务类（仅限 Flash Professional）”。

# mx.data.binding 包中的类（仅限 Flash Professional）

下表列出了 mx.data.binding 包中的类：

| 类                                                         | 说明                   |
|-----------------------------------------------------------|----------------------|
| <a href="#">Binding 类</a> （仅限 Flash Professional）         | 在两个端点之间创建绑定。         |
| <a href="#">ComponentMixins 类</a> （仅限 Flash Professional） | 将数据绑定的功能添加到组件中。      |
| <a href="#">CustomFormatter 类</a> （仅限 Flash Professional） | 用于创建自定义格式程序类的基类。     |
| <a href="#">CustomValidator 类</a> （仅限 Flash Professional） | 用于创建自定义验证程序类的基类。     |
| <a href="#">DataType 类</a> （仅限 Flash Professional）        | 提供对组件属性的数据字段的读写访问权限。 |
| <a href="#">EndPoint 类</a> （仅限 Flash Professional）        | 定义绑定的来源或目标。          |
| <a href="#">TypedValue 类</a> （仅限 Flash Professional）      | 包含数据值和有关值的数据类型的信息。   |

## Binding 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.Binding`

**Binding** 类定义两个端点（来源和目标）之间的关联。它侦听来源端点的更改，并在每次来源更改时将更改后的数据复制到目标端点。

您可以使用 **Binding** 类（和提供支持的类）编写自定义绑定，也可以使用“组件”检查器中的“绑定”选项卡。

提醒

若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。有关更多信息，请参见第 191 页的“使数据绑定类在运行时可用（仅限 Flash Professional）”。

有关 `mx.data.binding` 包中类的概述，请参见第 192 页的“`mx.data.binding` 包中的类（仅限 Flash Professional）”。

# Binding 类的方法摘要

下表列出了 Binding 类的方法。

| 方法                             | 说明                                |
|--------------------------------|-----------------------------------|
| <code>Binding.execute()</code> | 从来源组件中获取数据，对数据进行格式化，然后将数据分配给目标组件。 |

# Binding 类的构造函数

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

`new Binding(source, destination, [format], [isTwoWay])`

### 参数

*source* 绑定的来源端点。此参数在名义上属于 `mx.data.binding.EndPoint` 类型，但可以是具有所需 `Endpoint` 字段的任何 `ActionScript` 对象（请参见第 204 页的“[EndPoint 类（仅限 Flash Professional）](#)”）。

*destination* 绑定的目标端点。此参数在名义上属于 `mx.data.binding.EndPoint` 类型，但可以是具有所需 `Endpoint` 字段的任何 `ActionScript` 对象。

*format* 包含格式设置信息的可选对象。该对象必须具有以下属性：

- `cls` 扩展 `mx.data.binding.DataAccessor` 类的 `ActionScript` 类。
- `settings` 一个对象，其属性为 `cls` 所指定的格式程序类提供可选设置。

*isTwoWay* 一个可选布尔值，它指定新的 `Binding` 对象是 (`true`) 否 (`false`) 为双向。默认值为 `false`。

### 返回

无。

## 说明

构造函数：创建新的 **Binding** 对象。可以将数据绑定到具有属性并发出事件的任何 **ActionScript** 对象（包括但不限于组件）。

只要处于最里层的影片剪辑同时包含来源和目标组件，就会存在绑定对象。例如，如果名为 **A** 的影片剪辑包含组件 **X** 和 **Y**，并且 **X** 和 **Y** 之间存在绑定，则只要影片剪辑 **A** 存在，绑定就有效。

提醒

没有必要保留对新的 **Binding** 对象的引用。**Binding** 对象一经创建就会立即开始侦听任一端点发出的“已更改”事件。但是，在某些情况下，您可能需要保存对新 **Binding** 对象的引用，以便稍后能够调用它的 `execute()` 方法（请参见 [Binding.execute\(\)](#)）。

## 示例

在此示例中，一个 **TextInput** 组件 (`src_txt`) 的 `text` 属性被绑定到另一个 **TextInput** 组件 (`dest_txt`) 的 `text` 属性。在 `src_txt` 文本字段失去焦点时（也就是说，在生成 `focusOut` 事件时），它的 `text` 属性值将被复制到 `dest_txt.text` 中。

```
import mx.data.binding.*;
var src = new EndPoint();
src.component = src_txt;
src.property = "text";
src.event = "focusOut";
```

```
var dest = new EndPoint();
dest.component = dest_txt;
dest.property = "text";
```

```
new Binding(src, dest);
```

以下示例演示如何创建使用自定义格式程序类的 **Binding** 对象。有关更多信息，请参见第 196 页的“**CustomFormatter** 类（仅限 **Flash Professional**）”。

```
import mx.data.binding.*;
var src = new EndPoint();
src.component = src_txt;
src.property = "text";
src.event = "focusOut";
```

```
var dest = new EndPoint();
dest.component = text_dest;
dest.property = "text";
```

```
new Binding(src, dest, {cls: mx.data.formatters.Custom, settings:
 {classname: "com.mycompany.SpecialFormatter"}});
```

# Binding.execute()

## 可用性

Flash Player 6。

## 版本

Flash MX Professional 2004。

## 用法

```
myBinding.execute([reverse])
```

## 参数

*reverse* 布尔值，指定是同时执行从目标到来源的绑定 (true)，还是只执行从来源到目标的绑定 (false)。默认情况下，此值为 false。

## 返回

如果绑定成功执行，则该值为 null；否则，该方法返回一个由错误消息字符串组成的数组，这些字符串描述阻碍绑定执行的错误。

## 说明

方法：从来源组件获取数据，并将其分配给目标组件。如果绑定使用格式程序，则会在将数据分配给目标之前对其进行格式化。

此方法还会对数据进行验证，并会使目标和来源组件发出 valid 或 invalid 事件。数据即使无效也会被分配给目标，除非目标是只读的。

如果 *reverse* 参数设置为 true，并且绑定是双向的，则会反向执行绑定（从目标到来源）。

## 示例

**Button** 组件实例附带的以下代码将在按钮被单击时反向执行绑定（从目标组件到来源组件）。

```
on(click) {
 _root.myBinding.execute(true);
}
```

# CustomFormatter 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.CustomFormatter`

`CustomFormatter` 类定义两个方法（`format()` 和 `unformat()`），它们提供了将数据值从特定数据类型转换为字符串（反之亦然）的能力。默认情况下，这些方法不会执行任何操作；您必须在 `mx.data.binding.CustomFormatter` 的子类中实现它们。

若要创建自己的自定义格式程序，您需要首先创建实现 `format()` 和 `unformat()` 方法的 `CustomFormatter` 的子类。然后，您可以将该类分配给组件之间的绑定，方法是使用 ActionScript 创建一个新的 Binding 对象（请参见第 192 页的“[Binding 类（仅限 Flash Professional）](#)”），或者使用“组件”检查器中的“绑定”选项卡。有关使用“组件”检查器分配格式程序类的信息，请参见《使用 Flash》中的“架构格式程序”。

您也可以在“组件”检查器的“架构”选项卡上将格式程序类分配给组件属性。但是，在这种情况下，只有在需要字符串格式的数据时才会使用格式程序。相反，使用“绑定”面板分配或通过 ActionScript 创建的格式程序会在每次执行绑定时使用。

若要查看使用 ActionScript 编写和分配自定义格式程序的示例，请参见第 196 页的“[自定义格式程序范例](#)”。



若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。

有关 `mx.data.binding` 包中类的概述，请参见第 192 页的“[mx.data.binding 包中的类（仅限 Flash Professional）](#)”。

## 自定义格式程序范例

以下示例演示如何创建自定义格式程序类，然后使用 ActionScript 将其应用到两个组件之间的绑定。在此示例中，`NumericStepper` 组件的当前值（其 `value` 属性）被绑定到 `TextInput` 组件的当前值（其 `text` 属性）。在将 `NumericStepper` 组件的当前数字值（如 1、2 或 3）分配给 `TextInput` 组件之前，自定义格式程序类会将该值格式化为其对等的英文单词（如“one”、“two”或“three”）。



要创建和使用自定义格式程序：

1. 在 Flash 中，创建一个新的 ActionScript 文件。

2. 在文件中添加以下代码：

```
// NumberFormatter.as
class NumberFormatter extends mx.data.binding.CustomFormatter {
 // 格式化数字，返回字符串
 function format(rawValue) {
 var returnValue;
 var strArray = new Array('one', 'two', 'three');
 var numArray = new Array(1, 2, 3);
 returnValue = 0;
 for (var i = 0; i < strArray.length; i++) {
 if (rawValue == numArray[i]) {
 returnValue = strArray[i];
 break;
 }
 }
 return returnValue;
 } // 转换格式化后的值，返回原始值
 function unformat(formattedValue) {
 var returnValue;
 var strArray = new Array('one', 'two', 'three');
 var numArray = new Array(1, 2, 3);
 returnValue = "invalid";
 for (var i = 0; i < strArray.length; i++) {
 if (formattedValue == strArray[i]) {
 returnValue = numArray[i];
 break;
 }
 }
 return returnValue;
 }
}
```

3. 将 ActionScript 文件另存为 NumberFormatter.as。

4. 创建一个新的 Flash (FLA) 文件。

5. 在“组件”面板中将 TextInput 组件拖动到舞台上，并将其命名为 **textInput**。然后将一个 NumericStepper 组件拖动到舞台上，并将其命名为 **stepper**。

6. 打开时间轴并选择图层 1 上的第一帧。

7. 在“动作”面板中，将下面的代码添加到该面板中：

```
import mx.data.binding.*;
var x:NumberFormatter;
var customBinding = new Binding({component:stepper, property:"value",
 event:"change"}, {component:textInput, property:"text",
 event:"enter,change"}, {cls:mx.data.formatters.Custom,
 settings:{classname:"NumberFormatter"}});
```

第二行代码 (var x:NumberFormatter) 确保编译后的 SWF 文件中包含自定义格式程序类的字节代码。

8. 选择“窗口” > “公用库” > “类”以打开“类”库。

9. 选择“窗口” > “库”来打开文档的库。

10. 将 DataBindingClasses 从“类”库拖动到文档的库中。

这将使文档可以使用数据绑定运行时类。

11. 将 FLA 文件保存到包含 NumberFormatter.as 的同一文件夹中。

12. 测试文件（“控制” > “测试影片”）。

单击 NumericStepper 组件上的按钮，并观看 TextInput 组件的内容发生更新。

# CustomFormatter 类的方法摘要

下表列出了 CustomFormatter 类的方法。

| 方法                                         | 说明                      |
|--------------------------------------------|-------------------------|
| <a href="#">CustomFormatter.format()</a>   | 从原始数据类型转换为新的对象。         |
| <a href="#">CustomFormatter.unformat()</a> | 从字符串（或其它数据类型）转换为原始数据类型。 |

# CustomFormatter.format()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

此方法是自动调用的；您不必直接调用它。

## 参数

*rawData* 要格式化的数据。

## 返回

格式化后的值。

## 说明

方法：从原始数据类型转换为新的对象。

此方法不会默认执行。您必须在 `mx.data.binding.CustomFormatter` 的子类中定义此方法。

有关更多信息，请参见第 196 页的“自定义格式程序范例”。

# CustomFormatter.unformat()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

此方法是自动调用的；您不必直接调用它。

## 参数

*formattedData* 要转换回原始数据类型的已格式化数据。

## 返回

取消格式的值。

## 说明

方法：从字符串（或其它数据类型）转换为原始数据类型。此转换应与与 `CustomFormatter.format()` 恰好相反的转变。

此方法不会默认执行。您必须在 `mx.data.binding.CustomFormatter` 的子类中定义此方法。

有关更多信息，请参见第 196 页的“自定义格式程序范例”。

# CustomValidator 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.CustomValidator`

在需要对组件包含的数据字段执行自定义验证时，可以使用 `CustomValidator` 类。

若要创建自定义验证程序类，您需要首先创建 `mx.data.binding.CustomValidator` 的子类来实现名为 `validate()` 的方法。待验证的值会自动传递到此方法。有关如何实现此方法的更多信息，请参见 `CustomValidator.validate()`。

接着，您需要使用“组件”检查器的“架构”选项卡将自定义验证程序类分配给组件的字段。若要查看创建和使用自定义验证程序类的示例，请参见 `CustomValidator.validate()` 条目中的“示例”部分。

## 分配自定义验证程序：

1. 在“组件”检查器中，选择“架构”选项卡。
2. 选择要验证的字段，然后从“data type”弹出菜单中选择“Custom”。
3. 选择“validation options”字段（位于“架构”选项卡的底部），然后单击放大镜图标打开“自定义验证设置”对话框。
4. 在“ActionScript 类”文本框中输入您创建的自定义验证程序类的名称。

为了使指定的类包含在发布的 SWF 文件内，它必须位于类路径中。

提醒

若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。

有关 `mx.data.binding` 包中类的概述，请参见第 192 页的“`mx.data.binding` 包中的类（仅限 Flash Professional）”。

## CustomValidator 类的方法摘要

下表列出了 `CustomValidator` 类的方法。

| 方法                                             | 说明       |
|------------------------------------------------|----------|
| <code>CustomValidator.validate()</code>        | 对数据执行验证。 |
| <code>CustomValidator.validationError()</code> | 报告验证错误。  |

# CustomValidator.validate()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

此方法是自动调用的；您不必直接调用它。

## 参数

*value* 要验证的数据；可以是任何类型。

## 返回

无。

## 说明

方法：自动调用以验证 *value* 参数包含的数据。必须在 **CustomValidator** 的子类中实现此方法；默认实现不会执行任何操作。

可以使用任何 **ActionScript** 代码检查和验证数据。如果数据无效，此方法应调用带有相应消息的 `this.validationError()`。如果数据存在多个验证问题，则可以调用 `this.validationError()` 多次。

由于可能重复调用 `validate()`，因此请避免添加需要很长时间才能完成的代码。此方法的实现只应检查有效性，然后使用 `CustomValidator.validationError()` 报告任何错误。同样，这种实现不应作为验证测试的任何结果（如警示最终用户），而应该为 `valid` 和 `invalid` 事件创建事件侦听器，然后从这些事件侦听器中警示最终用户（请参见下面的示例）。

## 示例

以下过程演示如何创建和使用自定义验证程序类。**CustomValidator** 类 `OddNumbersOnly.as` 的 `validate()` 方法将不是奇数的任何值确定为无效。验证操作在 **NumericStepper** 组件值发生更改的任何时候进行，该组件已绑定到 **Label** 组件的 `text` 属性。

要创建和使用自定义验证程序类：

1. 在 Flash 中，创建一个新的 ActionScript (AS) 文件。

2. 将以下代码添加到该 AS 文件：

```
class OddNumbersOnly extends mx.data.binding.CustomValidator
{
 public function validate(value) {
 // 确保该值的类型为数字
 var n = Number(value);
 if (String(n) == "NaN") {
 this.validationError("'" + value + "' is not a number.");
 return;
 }
 // 确定数字为奇数
 if (n % 2 == 0) {
 this.validationError("'" + value + "' is not an odd number.");
 return;
 }
 // 数据无误，无需执行任何操作，只需返回
 }
}
```

3. 将该 AS 文件另存为 OddNumbersOnly.as。



AS 文件的名称必须与类的名称匹配。

4. 创建一个新的 Flash (FLA) 文件。

5. 打开“组件”面板。

6. 将一个 NumericStepper 组件从“组件”面板拖到舞台上，然后将其命名为 **stepper**。

7. 将 Label 组件拖到舞台上，然后将其命名为 **textLabel**。

8. 将一个 TextArea 组件拖到舞台上，然后将其命名为 **status**。

9. 选择 NumericStepper 组件，然后打开“组件”检查器。

10. 在“组件”检查器中选择“绑定”选项卡，然后单击“添加绑定” (+) 按钮。

11. 在“添加绑定”对话框中选择 Value 属性（唯一的属性），然后单击“确定”。

12. 在“组件”检查器中，双击“绑定”选项卡“绑定属性”窗格中的“bound to”，打开“绑定到”对话框。

13. 在“绑定到”对话框的“组件路径”窗格中选择 Label 组件，并在“架构位置”窗格中选择其 text 属性。单击“确定”。

14. 在舞台上选择 Label 组件，然后单击“组件”检查器面板中的“架构”选项卡。

15. 在“架构属性”窗格中，从“data type”弹出菜单中选择“Custom”。

16. 双击“架构属性”窗格中的“validation options”字段，打开“自定义验证设置”对话框。

17. 在“ActionScript 类”文本框中输入 **OddNumbersOnly**，即之前创建的 ActionScript 类的名称。单击“确定”。

18. 打开时间轴并选择图层 1 上的第一帧。

19. 打开“动作”面板。

20. 将以下代码添加到“动作”面板中：

```
function dataIsValid(evt) {
 if (evt.property == "text") {
 status.text = evt.messages;
 }
}

function dataIsInvalid(evt) {
 if (evt.property == "text") {
 status.text = "OK";
 }
}

textLabel.addEventListener("valid", dataIsValid);
textLabel.addEventListener("invalid", dataIsInvalid);
```

21. 将 FLA 文件另存为 **OddOnly.fla**，并保存到包含 **OddNumbersOnly.as** 的同一文件夹中。

22. 测试 SWF 文件（“控制” > “测试影片”）。

单击 **NumericStepper** 组件上的箭头更改它的值。注意在选择偶数和奇数时 **TextArea** 组件中出现的消息。

## CustomValidator.validationError()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

`this.validationError(errorMessage)`



只能从自定义验证程序类中调用此方法；关键字 `this` 指当前的 `CustomValidator` 对象。

### 参数

`errorMessage` 包含要报告的错误消息的字符串。

## 返回

无。

## 说明

方法：可以从 **CustomValidator** 的子类的 `validate()` 方法中调用，以便报告验证错误。如果不调用 `validationError()`，则会在 `validate()` 结束执行时生成 `valid` 事件。如果从 `validate()` 内调用 `validationError()` 一次或多次，则 `validate()` 返回后会生成 `invalid` 事件。

传递给 `validationError()` 的每条消息都可在传递给 `invalid` 事件处理函数的事件对象的消息 `messages` 属性中找到。

## 示例

请参见 `CustomValidator.validate()` 的“示例”部分。

# EndPoint 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.EndPoint`

**EndPoint** 类定义绑定的来源或目标。**EndPoint** 对象定义常数值、组件属性或组件属性的特定字段，您可以从中获取数据或者为其分配数据。它们还能够定义 **Binding** 对象侦听的事件或事件列表；在发生指定事件时，绑定即会执行。

当使用 **Binding** 类构造函数创建新的绑定时，您需要为其传递两个 **EndPoint** 对象：一个用于来源，一个用于目标。

```
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```

**EndPoint** 对象 `srcEndPoint` 和 `destEndPoint` 可按以下方式进行定义：


```
var srcEndPoint = new mx.data.binding.EndPoint();
var destEndPoint = new mx.data.binding.EndPoint();
srcEndPoint.component = source_txt;
srcEndPoint.property = "text";
srcEndPoint.event = "focusOut";
destEndPoint.component = dest_txt;
destEndPoint.property = "text";
```

如果用文字表述，上面代码的意思是“在来源文本字段失去焦点时，将其 `text` 属性值复制到目标文本字段的 `text` 属性”。



您也可以将一般 **ActionScript** 对象传递给 **Binding** 构造函数，而不是传递明确构造的 **EndPoint** 对象。唯一的要求是这些对象应定义所需的 **EndPoint** 属性，即 **component** 和 **property**。以下代码与上面显示的代码等效。

```
var srcEndPoint = {component:source_txt, property:"text"};
var destEndPoint = {component:dest_txt, property:"text"};
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```



若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。

有关 **mx.data.binding** 包中类的概述，请参见第 192 页的“**mx.data.binding** 包中的类（仅限 **Flash Professional**）”。

## EndPoint 类的属性摘要

下表列出了 **EndPoint** 类的属性。

| 方法                                 | 说明                                                |
|------------------------------------|---------------------------------------------------|
| <a href="#">EndPoint.component</a> | 指向组件实例的引用。                                        |
| <a href="#">EndPoint.constant</a>  | 一个常数值。                                            |
| <a href="#">EndPoint.event</a>     | 组件在数据更改时将发出的事件的名称或事件名称的数组。                        |
| <a href="#">EndPoint.location</a>  | 数据字段在组件实例的属性中的位置。                                 |
| <a href="#">EndPoint.property</a>  | <a href="#">EndPoint.component</a> 所指定的组件实例的属性名称。 |

## EndPoint 类的构造函数

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
new EndPoint()
```

### 返回

无。

### 说明

构造函数：创建新的 **EndPoint** 对象。

## 示例

此示例将创建名为 `source_obj` 的新 **EndPoint** 对象，并为其 `component` 和 `property` 属性赋值：

```
var source_obj = new mx.data.binding.EndPoint();
source_obj.component = myTextField;
source_obj.property = "text";
```

# EndPoint.component

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*endpointObj.component*

## 说明

属性；指向组件实例的引用。

## 示例

此示例将 **List** 组件的某个实例 (`listBox1`) 指定为 **EndPoint** 对象的组件参数。

```
var sourceEndPoint = new mx.data.binding.EndPoint();
sourceEndPoint.component = listBox1;
```

# EndPoint.constant

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*endpoint\_src.constant*

## 说明

属性：分配给 **EndPoint** 对象的常数值。此属性只能应用于作为组件间绑定的来源（而不是目标）的 **EndPoint** 对象。值可以是与绑定目标兼容的任何数据类型。如果指定了此属性，则指定 **EndPoint** 对象的所有其它 **EndPoint** 属性都将被忽略。

## 示例

在本示例中，字符串常数值 “hello” 被分配给 **EndPoint** 对象的 **constant** 属性。

```
var sourceEndPoint = new mx.data.binding.EndPoint();
sourceEndPoint.constant = "hello";
```

# EndPoint.event

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*endpointObj.event*

## 说明

属性：指定在分配给绑定属性的数据更改时由组件生成的事件名称或事件名称数组。当事件发生时，绑定即会执行。

指定的事件仅适用于用作绑定来源或双向绑定的目标的组件。有关创建双向绑定的更多信息，请参见第 192 页的 [“Binding 类（仅限 Flash Professional）”](#)。

## 示例

在此示例中，一个 **TextInput** 组件 (src\_txt) 的 **text** 属性被绑定到另一个 **TextInput** 组件 (dest\_txt) 的相同属性。当 src\_txt 组件发出 **focusOut** 或 **enter** 事件时，绑定即会执行。

```
var source = {component:src_txt, property:"text", event:["focusOut",
 "enter"]};
var dest = {component:myTextArea, property:"text"};
var newBind = new mx.data.binding.Binding(source, dest);
```

# EndPoint.location

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*endPointObj.location*

## 说明

属性；指定数据字段在组件实例的属性中的位置。可以以四种方式来指定位置：作为包含 XPath 表达式的字符串、作为包含 ActionScript 路径的字符串、作为字符串的数组或作为对象。

只有在数据是 XML 对象时，才能使用 XPath 表达式。（请参见下面的示例 1。）若要查看支持的 XPath 表达式的列表，请参见《使用 Flash》中的“创建索引绑定”。

对于 XML 和 ActionScript 对象，您还可以指定包含 ActionScript 路径的字符串。动作脚本路径包含由点分隔的字段名称（如 "a.b.c"）。

您也可以指定字符串数组作为位置。数组中的每个字符串均“下寻”到另一个嵌套级别。可以将此技术用于 XML 和 ActionScript 数据。（请参见下面的示例 2。）当用于 ActionScript 数据时，字符串数组相当于使用 ActionScript 路径；也就是说，数组 ["a","b","c"] 相当于 "a.b.c"。

如果将某个对象指定为位置，则该对象必须指定两个属性：path 和 indices。如上所述，path 属性是一个字符串数组，除了一个或多个指定字符串可能是特殊标记 "[n]"。对于 path 中出现的每个这种标记，在 indices 中必须有对应的索引项。在计算路径的值时，这些索引用于为数组建立索引。索引项可以是任意 EndPoint 对象。这种类型的位置只能应用于 ActionScript 数据，而不能应用于 XML。（请参见下面的示例 3。）

## 示例

示例 1：此示例使用 XPath 表达式指定名为 zip 的节点在 XML 对象中的位置：

```
var sourceEndPoint = new mx.databinding.EndPoint();
var sourceObj = new Object();
sourceObj.xml = new XML("<zip>94103</zip>");
sourceEndPoint.component = sourceObj;
sourceEndPoint.property = "xml";
sourceEndPoint.location = "/zip";
```

示例 2：此示例使用字符串数组“下寻”到嵌套的影片剪辑属性：

```
var sourceEndPoint = new mx.data.binding.EndPoint();
// 假设 movieClip1.ball.position 存在。
sourceEndPoint.component = movieClip1;
sourceEndPoint.property = "ball";
// 访问 movieClip1.ball.position.x。
sourceEndPoint.location = ["position","x"];
```

示例 3：此示例显示如何使用对象在复杂的数据结构中指定数据字段的位置：

```
var city = new Object();
city.theaters = [{theater: "t1", movies: [{name: "Good,Bad,Ugly"},
 {name:"Matrix Reloaded"}]}, {theater: "t2", movies: [{name: "Gladiator"},
 {name: "Catch me if you can"}]}}];
var srcEndPoint = new EndPoint();
srcEndPoint.component = city;
srcEndPoint.property = "theaters";
srcEndPoint.location = {path: ["[n]","movies","[n]","name"], indices:
 [{constant:0},{constant:0}]};
```

## EndPoint.property

### 可用性

Flash Player 6 (6.0.79.0)

### 版本

Flash MX Professional 2004。

### 用法

*endPointObj.property*

### 说明

属性；指定由 [EndPoint.component](#)（包含可绑定数据）所指定的组件实例的属性名称。



[EndPoint.component](#) 和 [EndPoint.property](#) 必须结合使用才能形成有效的 ActionScript 对象 / 属性组合。

### 示例


此示例将一个 **TextInput** 组件 (text\_1) 的 text 属性绑定到另一个 **TextInput** 组件 (text\_2) 的相同属性。

```
var sourceEndPoint = {component:text_1, property:"text"};
var destEndPoint = {component:text_2, property:"text"};
new Binding(sourceEndPoint, destEndPoint);
```

# ComponentMixins 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.ComponentMixins`

`ComponentMixins` 类定义自动添加到任何对象（充当绑定的来源或目标）或任何组件（充当 `ComponentMixins.initComponent()` 方法调用的目标）的属性和方法。这些属性和方法不会影响正常的组件功能；相反，它们还添加了对数据绑定十分有用的功能。

若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。

有关 `mx.data.binding` 包中类的概述，请参见第 192 页的“[mx.data.binding 包中的类（仅限 Flash Professional）](#)”。

## ComponentMixins 类的方法摘要

下表列出了 `ComponentMixins` 类的方法。

| 方法                                                 | 说明                                      |
|----------------------------------------------------|-----------------------------------------|
| <code>ComponentMixins.getField()</code>            | 返回用于在组件属性的指定位置获取和设置字段值的对象。              |
| <code>ComponentMixins.initComponent()</code>       | 将 <code>ComponentMixins</code> 方法添加到组件。 |
| <code>ComponentMixins.refreshDestinations()</code> | 执行所有将此对象作为来源端点的绑定。                      |
| <code>ComponentMixins.refreshFromSources()</code>  | 执行所有将此组件作为目标端点的绑定。                      |
| <code>ComponentMixins.validateProperty()</code>    | 检查以确定指定属性中的数据是否有效。                      |

## ComponentMixins.getField()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

`componentInstance.getField(propertyName, [location])`

## 说明

*propertyName* 包含所指定组件的属性名称的字符串。

*location* 一个可选参数，指示字段在组件属性内的位置。如果 *propertyName* 指定复杂的数据结构并且您对该结构的特定字段感兴趣，则此参数十分有用。*location* 属性可以采用以下三种形式之一：

- 包含 XPath 表达式的字符串。这种格式仅对 XML 数据结构有效。若要查看支持的 XPath 表达式的列表，请参见《使用 Flash》中的“创建索引绑定”。
- 一个字符串，包含由点分隔的字段名称（如 "a.b.c"）。此形式允许用于任何复杂数据（ActionScript 或 XML）。
- 字符串数组，其中每个字符串都是一个字段名称，如 ["a", "b", "c"]。此形式允许用于任何复杂数据（ActionScript 或 XML）。

## 返回

一个 `DataType` 对象。

## 说明

方法：返回一个 `DataType` 对象，您可以使用该对象的方法在组件属性的指定字段位置获取或设置数据值。有关更多信息，请参见第 216 页的“[DataType 类（仅限 Flash Professional）](#)”。

## 示例

此示例使用 `DataType.setAsString()` 方法设置位于组件属性中的字段的值。在这种情况下，属性 (`results`) 是一种复杂数据结构。

```
import mx.data.binding.*;
var field : DataType = myComponent.getField("results", "po.address.name1");
field.setAsString("Teri Randall");
```

## 另请参见

[DataType.setAsString\(\)](#)

# ComponentMixins.initComponent()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mx.data.binding.ComponentMixins.initComponent(componentInstance)
```

## 参数

*componentInstance* 指向组件实例的引用。

## 返回

无。

## 说明

方法（静态）：将所有 **ComponentMixins** 方法添加到 *componentInstance* 指定的组件中。系统将为数据绑定中涉及的所有组件自动调用此方法。若要使 **ComponentMixins** 方法可供数据绑定中未涉及的某个组件使用，您必须为该组件显式调用此方法。

## 示例

以下代码使 **ComponentMixins** 方法可供 **DataSet** 组件使用：

```
mx.data.binding.ComponentMixins.initComponent(_root.myDataSet);
```

# ComponentMixins.refreshDestinations()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
componentInstance.refreshDestinations()
```

## 参数

无。



## 返回

无。

## 说明

方法；执行 *componentInstance* 是来源 **EndPoint** 对象的所有绑定。此方法使您能够执行来源未发出“数据已更改”事件的绑定。

## 示例

以下示例执行 **DataSet** 组件实例（名为 *user\_data*）是来源 **EndPoint** 对象的所有绑定：

```
user_data.refreshDestinations();
```

# ComponentMixins.refreshFromSources()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
componentInstance.refreshFromSources()
```

## 参数

无。

## 返回

无。

## 说明

方法；执行 *componentInstance* 是目标 **EndPoint** 对象的所有绑定。此方法使您能够执行具有常数来源或未发出任何“数据已更改”事件的来源的绑定。

## 示例

以下示例执行 **ListBox** 组件实例（名为 *cityList*）是目标 **EndPoint** 对象的所有绑定：

```
cityList.refreshFromSources();
```

# ComponentMixins.validateProperty()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.validateProperty(propertyName)*

## 参数

*propertyName* 一个字符串，包含属于 *componentInstance* 的属性的名称。

## 返回

一个数组，或 null。

## 说明

方法：根据属性的架构设置确定 *propertyName* 中的数据是否有效。属性的架构设置是指“组件”检查器的“架构”选项卡上所指定的内容。

如果数据有效，此方法将返回 null；否则以字符串的形式返回错误消息数组。

验证仅适用于具有可用架构信息的字段。如果某个字段是包含其它字段的对象，则会以递归方式验证每个“子”字段，依此类推。每个单独的字段都会根据需要调度 valid 或 invalid 事件。对于 *propertyName* 包含的每个数据字段，此方法将调度 valid 或 invalid 事件，如下所示：

- 如果字段的值为 null，并且该值不是必需的，则此方法将返回 null。不会生成任何事件。
- 如果字段的值为 null，并且该值是必需的，则会返回错误并生成 invalid 事件。
- 如果字段的值不是 null 并且字段的架构不具有验证程序，则此方法将返回 null；不会生成任何事件。
- 如果该值不是 null 且字段的构架确实定义了验证程序，则数据由验证程序对象进行处理。如果数据有效，则会生成 valid 事件并返回 null；否则生成 invalid 事件并返回错误字符串数组。

## 示例

以下示例显示如何使用 validateProperty() 来确保用户所输入文本的长度有效。您可通过在“组件”检查器的“架构”选项卡中为字符串数据类型设置“validation options”（验证选项）来确定有效长度。如果用户在文本字段中输入了长度无效的字符串，则 validateProperty() 返回的错误消息将显示在“输出”面板中。

要对用户在 `TextInput` 组件中输入的文本进行验证：

1. 将一个 `TextInput` 组件从“组件”面板拖到舞台上，并命名为 `zipCode_txt`。
2. 选择 `TextInput` 组件，然后在“组件”检查器中单击“架构”选项卡。
3. 在“架构树”窗格（“架构”选项卡的顶部窗格）中，选择 `text` 属性。
4. 在“架构属性”窗格（“架构”选项卡的底部窗格）中，从“`data type`”弹出菜单中选择“`ZipCode`”。
5. 如果时间轴尚未打开，则将其打开。
6. 单击时间轴中“第 1 层”上的第一帧，然后打开“动作”面板（“窗口” > “动作”）。
7. 将以下代码添加到“动作”面板中：

```
// 将 ComponentMixin 方法添加到 TextInput 组件。
// 请注意，只有在组件尚未作为
// 来源或目标涉及在数据绑定中时，
// 才必须进行此步骤。
mx.data.binding.ComponentMixins.initComponent(zipCode_txt);
// 为组件定义事件侦听器函数：
validateResults = function (eventObj) {
 var errors:Array = eventObj.target.validateProperty("text");
 if (errors != null) {
 trace(errors);
 }
};
// 将侦听器函数注册到组件：
zipCode_txt.addEventListener("enter", validateResults);
```

8. 选择“窗口” > “公用库” > “类”以打开“类”库。
9. 选择“窗口” > “库”来打开文档的库。
10. 将 `DataBindingClasses` 从“类”库拖动到文档的库中。  
此步骤使数据绑定运行时类可供 `SWF` 文件在运行时使用。
11. 通过选择“控制” > “测试影片”对 `SWF` 文件进行测试。

在舞台上的 `TextInput` 组件中，输入一个无效的美国邮政编码，例如，全部由字母组成或包含少于五个数字的邮政编码。注意“输出”面板中显示的错误消息。

# DataType 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.DataType`

`DataType` 类提供对组件属性的数据字段的读写权限。若要获取 `DataType` 对象，您需要对组件调用 `ComponentMixins.getField()` 方法。然后，您可以调用 `DataType` 对象的方法来获取和设置字段的值。

如果您直接在组件实例上获取和设置字段的值，而不是使用 `DataType` 类方法，则数据以其“原始”形式提供。相反，在使用 `DataType` 方法获取或设置字段值时，系统将根据字段的架构设置处理这些值。

例如，以下代码将直接获取组件属性的值，并将其赋给变量。变量 `propVar` 包含作为 `propName` 属性的当前值的任何“原始”值。

```
var propVar = myComponent.propName;
```

下一示例通过使用 `DataType.getAsString()` 方法获取同一属性的值。在这种情况下，分配给 `stringVar` 的值就是根据架构设置处理后作为字符串返回的 `propName` 的值。

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");
var stringVar: String = dataTypeObj.getAsString();
```

有关如何指定字段的架构设置的更多信息，请参见《使用 Flash》中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

您也可以使用 `DataType` 类的方法来获取或设置不同数据类型的字段。如果可能，`DataType` 类会自动将原始数据转换为要求的类型。例如，在上面的代码示例中，检索出的数据被转换为字符串类型（即使原始数据是另一种类型）。

`ComponentMixins.getField()` 方法可供已包括在数据绑定中（作为来源、目标或索引）或已使用 `ComponentMixins.initComponent()` 初始化的组件使用。有关更多信息，请参见第 210 页的“`ComponentMixins` 类（仅限 Flash Professional）”。



若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。

有关 `mx.data.binding` 包中类的概述，请参见第 192 页的“`mx.data.binding` 包中的类（仅限 Flash Professional）”。

# DataType 类的方法摘要

下表列出了 `DataType` 类的方法。

| 方法                                       | 说明                   |
|------------------------------------------|----------------------|
| <code>DataType.getAnyTypedValue()</code> | 获取字段的当前值。            |
| <code>DataType.getAsBoolean()</code>     | 以布尔值的形式获取字段的当前值。     |
| <code>DataType.getAsNumber()</code>      | 以数字的形式获取字段的当前值。      |
| <code>DataType.getAsString()</code>      | 以字符串值的形式获取字段的当前值。    |
| <code>DataType.getTypedValue()</code>    | 以要求的数据类型的形式获取字段的当前值。 |
| <code>DataType.setAnyTypedValue()</code> | 在该字段中设置一个新值。         |
| <code>DataType.setAsBoolean()</code>     | 将字段设置为以布尔值形式提供的新值。   |
| <code>DataType.setAsNumber()</code>      | 将字段设置为以数字形式提供的新值。    |
| <code>DataType.setAsString()</code>      | 将字段设置为以字符串形式提供的新值。   |
| <code>DataType.setTypedValue()</code>    | 在该字段中设置一个新值。         |

# DataType 类的属性摘要

下表列出了 `DataType` 类的属性。

| 属性                              | 说明                                  |
|---------------------------------|-------------------------------------|
| <code>DataType.encoder</code>   | 提供对与此字段关联的编码器对象的引用。                 |
| <code>DataType.formatter</code> | 提供对与此字段关联的格式程序对象的引用。                |
| <code>DataType.kind</code>      | 提供对与此字段关联的 <code>Kind</code> 对象的引用。 |

# DataType.encoder

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`dataTypeObject.encoder`

## 说明

属性；提供对与此字段关联的编码器对象的引用（如果存在）。可以使用此属性访问由特定编码器（应用于“组件”检查器的“架构”选项卡中的字段）定义的任何属性和方法。

如果未将任何编码器应用于所讨论的字段，则此属性将返回 `undefined`。

有关随 **Flash** 提供的编码器的更多信息，请参见《使用 **Flash**》中的“架构编码器”。

## 示例

以下示例假设所访问的字段 (`isValid`) 使用布尔值编码器 (`mx.data.encoders.Bool`)。此编码器随 **Flash** 提供，并包含名为 `trueStrings` 的属性，该属性指定应将哪些字符串解释为值 `true`。以下代码将字段编码器的 `trueStrings` 属性设置为字符串“**Yes**”和“**Oui**”。

```
var myField:mx.data.binding.DataType = dataSet.getField("isValid");
myField.encoder.trueStrings = "Yes,Oui";
```

# DataType.formatter

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataTypeObject.formatter*

## 说明

属性；提供对与字段关联的格式程序对象的引用（如果存在）。可以使用此属性访问格式程序对象（应用于“组件”检查器的“架构”选项卡中的字段）的任何属性和方法。

如果未将任何格式程序应用于所讨论的字段，则此属性将返回 `undefined`。

有关随 **Flash** 提供的格式程序的更多信息，请参见《使用 **Flash**》中的“架构格式程序”。

## 示例

此示例假设所访问的字段正在使用随 **Flash Professional 8** 提供的数字格式程序 (`mx.data.formatters.NumberFormatter`)。此格式程序包含一个名为 `precision` 的属性，该属性指定小数点后显示的数字位数。以下代码将使用此格式程序的字段的 `precision` 属性设置为两位小数。

```
var myField:DataType = dataGrid.getField("currentBalance");
myField.formatter.precision = 2;
```

# DataType.getAnyTypedValue()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
dataTypeObject.getAnyTypedValue(suggestedTypes)
```

## 参数

*suggestedTypes* 由字符串组成的数组，这些字符串为字段指定首选数据类型（根据需要程度按降序排列）。

## 返回

字段的当前值，采用 *suggestedTypes* 数组中指定的某个数据类型的形式。

## 说明

方法：获取字段的当前值，并使用字段架构中的信息来处理值。如果字段能够提供值作为 *suggestedTypes* 数组中指定的第一个数据类型，则该方法将返回字段的值作为该数据类型。否则，方法将尝试提取字段的值作为 *suggestedTypes* 数组中指定的第二个数据类型，依此类推。

如果指定 `null` 作为 *suggestedTypes* 数组中的一项，则方法将以“组件”检查器的“架构”选项卡中指定的数据类型返回字段的值。指定 `null` 将始终会导致正在返回值的现象，因此，请仅在数组的结尾使用 `null`。

如果值无法以某个建议类型的形式返回，则会以“架构”选项卡中指定的类型返回。

## 示例

此示例将首先尝试以数值形式获取 **XMLConnector** 组件的 `results` 属性中某个字段 (`productInfo.available`) 的值；如果失败，则以字符串的形式获取。

```
import mx.data.binding.DataType;
import mx.data.binding.TypedValue;
var f: DataType = myXmlConnector.getField("results",
 "productInfo.available");
var b: TypedValue = f.getAnyTypedValue(["Number", "String"]);
```

## 另请参见

[ComponentMixins.getField\(\)](#)

# DataType.getAsBoolean()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
dataTypeObject.getAsBoolean()
```

## 参数

无。

## 返回

一个布尔值。

## 说明

方法；获取字段的当前值，并在必要时将该值转换为布尔值形式。

## 示例

在此示例中，将以布尔值的形式检索名为 `propName` 的字段（属于名为 `myComponent` 的组件），并将检索到的值赋给变量：

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");
var propValue:Boolean = dataTypeObj.getAsBoolean();
```

# DataType.getAsNumber()

## 可用性

Flash Player 6。

## 版本

Flash MX Professional 2004。

## 用法

```
dataTypeObject.getAsNumber()
```

## 参数

无。



## 返回

一个数字。

## 说明

方法：获取字段的当前值，并在必要时将该值转换为数字形式。

## 示例

在此示例中，将以数字的形式检索名为 `propName` 的字段（属于名为 `myComponent` 的组件），并将检索到的值赋给变量：

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");
var propValue:Number = dataTypeObj.getAsNumber();
```

## 另请参见

[DataType.getAnyTypedValue\(\)](#)

# DataType.getString()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataTypeObject*.getString()

## 参数

无。

## 返回

字符串。

## 说明

方法：获取字段的当前值，并在必要时将该值转换为字符串形式。

## 示例

在此示例中，将以字符串的形式检索名为 `propName` 的属性（属于名为 `myComponent` 的组件），并将检索到的值赋给变量：

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");
var propValue:String = dataTypeObj.getAsString();
```

## 另请参见

[DataType.getAnyTypedValue\(\)](#)

# DataType.getTypedValue()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
dataTypeObject.getTypedValue(requestedType)
```

## 参数

*requestedType* 包含数据类型的名称或 `null` 的字符串。

## 返回

一个 `TypedValue` 对象（请参见第 228 页的“[TypedValue 类（仅限 Flash Professional）](#)”）。

## 说明

方法；以指定形式返回字段的值（如果字段能以该形式提供其值）。如果字段无法以要求的形式提供值，则该方法返回 `null`。

如果将 `null` 指定为 *requestedType*，则该方法将以默认类型返回字段的值。

## 示例

以下示例返回数据类型被转换为布尔值的字段值。此值存储在 `bool` 变量中。

```
var bool:TypedValue = field.getTypedValue("Boolean");
```

# DataType.kind

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataTypeObject.kind*

## 参数

属性；提供与此字段关联的 **Kind** 对象的引用。可以使用此属性来访问 **Kind** 对象的属性和方法。

# DataType.setAnyTypedValue()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataTypeObject.setAnyTypedValue(newTypedValue)*

## 参数

*newTypedValue* 要在该字段中设置的 **TypedValue** 对象值。有关更多信息，请参见第 228 页的“**TypedValue** 类（仅限 Flash Professional）”。

## 返回

一个字符串数组，描述在尝试设置此新值时发生的任何错误。在以下任一条件下可能发生错误：

- 无法将提供的数据转换为此字段的数据类型（例如，无法将 "abc" 转换为数字）。
- 数据是可接受类型，但不满足该字段的验证条件。
- 该字段为只读。



错误消息的实际文本将因在该字段的架构中定义的数据类型、格式程序和编码器而异。

## 说明

方法：在字段中设置新值，并使用字段架构中的信息来处理字段。

此方法会通过首先调用 `DataType.setTypedValue()` 方法来设置值。如果失败，方法将检查以确定目标对象是否愿意接受字符串、布尔值或数值数据，如果答案肯定，则会尝试使用对应的 `ActionScript` 转换函数。

## 示例

以下示例创建新的 `TypedValue` 对象（一个布尔值），然后将该值分配给名为 `field` 的一个 `DataType` 对象。发生的任何错误都被分配给 `errors` 数组。

```
import mx.data.binding.*;
var t:TypedValue = new TypedValue (true, "Boolean");
var errors: Array = field.setAnyTypedValue (t);
```

## 另请参见

[DataType.setTypedValue\(\)](#)

# DataType.setAsBoolean()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
dataTypeObject.setAsBoolean(newBooleanValue)
```

## 参数

*newBooleanValue* 布尔值。

## 返回

无。

## 说明

方法：将字段设置为以布尔值形式提供的新值。该值被转换并存储为适合该字段的数据类型。

## 示例

以下示例将一个名为 `bool` 的变量设置为布尔值 `true`，然后将 `field` 所引用的值设置为 `true`。

```
var bool: Boolean = true;
field.setAsBoolean (bool);
```

# DataType.setAsNumber()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
dataTypeObject.setAsNumber(newNumberValue)
```

## 参数

*newNumberValue* 数字。

## 返回

无。

## 说明

方法：将字段设置为以数字形式提供的新值。该值被转换并存储为适合该字段的数据类型。

## 示例

以下示例将一个名为 `num` 的变量设置为数值 `32`，然后将 `field` 所引用的值设置为 `num`。

```
var num: Number = 32;
field.setAsNumber (num);
```

# DataType.setAsString()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataTypeObject.setAsString(newStringValue)*

## 参数

*newStringValue* 字符串。

## 返回

无。

## 说明

方法；将字段设置为以字符串形式提供的新值。该值被转换并存储为适合该字段的数据类型。

## 示例

以下示例将变量 `stringVal` 设置为字符串 "The new value"，然后将 `field` 的值设置为该字符串。

```
var stringVal: String = "The new value";
field.setAsString (stringVal);
```

# DataType.setTypedValue()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataTypeObject.setTypedValue(newTypedValue)*

## 参数

`newTypedValue` 要在该字段中设置的 `TypedValue` 对象值。

有关 `TypedValue` 对象的更多信息，请参见第 228 页的“[TypedValue 类（仅限 Flash Professional）](#)”。

## 返回

一个字符串数组，描述在尝试设置此新值时发生的任何错误。在以下任一条件下可能发生错误：

- 所提供数据的类型不可接受。
- 无法将提供的数据转换为此字段的数据类型（例如，无法将 "abc" 转换为数字）。
- 数据是可接受类型，但不满足该字段的验证条件。
- 该字段为只读。



错误消息的实际文本将因在该字段的架构中定义的数据类型、格式程序和编码器而异。

## 说明

方法：在字段中设置新值，并使用字段架构中的信息来处理字段。此方法的行为方式与 `DataType.setAnyTypedValue()` 类似，只是它不会同样尽力地将数据转换为可接受的数据类型。有关更多信息，请参见 `DataType.setAnyTypedValue()`。

## 示例

此示例创建新的 `TypedValue` 对象（一个布尔值），然后将该值分配给名为 `field` 的一个 `DataType` 对象。发生的任何错误都被分配给 `errors` 数组。

```
import mx.data.binding.*;
var bool:TypedValue = new TypedValue (true, "Boolean");
var errors: Array = field.setTypedValue (bool);
```

## 另请参见

[DataType.setTypedValue\(\)](#)

# TypedValue 类（仅限 Flash Professional）

ActionScript 类名称 `mx.data.binding.TypedValue`

`TypedValue` 对象包含数据值以及有关该值的数据类型的信息。`TypedValue` 对象以 `DataType` 类各种方法的参数的形式提供，并且是从这些方法中返回的。`TypedValue` 对象中的数据类型信息帮助 `DataType` 对象确定何时需要执行数据类型转换，以及如何进行。



若要使此类在运行时可用，您必须在 FLA 文件中包括数据绑定类。

有关 `mx.data.binding` 包中类的概述，请参见第 192 页的“[mx.data.binding 包中的类（仅限 Flash Professional）](#)”。

## TypedValue 类的属性摘要

下表列出了 `TypedValue` 类的属性。

| 属性                               | 说明                                     |
|----------------------------------|----------------------------------------|
| <code>TypedValue.type</code>     | 包含与 <code>TypedValue</code> 对象的值关联的架构。 |
| <code>TypedValue.typeName</code> | 指定 <code>TypedValue</code> 对象的值的数据类型。  |
| <code>TypedValue.value</code>    | 包含 <code>TypedValue</code> 对象的数据值。     |

## TypedValue 类的构造函数

### 可用性

Flash Player 6 (6.0.79.0)。

### 用法

```
new mx.data.binding.TypedValue(value, typeName, [type])
```

### 参数

*value* 任何类型的数据值。

*typeName* 包含值的数据类型名称的字符串。

*type* 可选的 `Schema` 对象，可更详细地描述数据的架构。只有在某些情形下（如将数据设置为 `DataSet` 组件的 `dataProvider` 属性时），此字段才是必需的。

### 说明

构造函数：创建新的 `TypedValue` 对象。



# TypedValue.type

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*typedValueObject.type*

## 说明

属性；包含与 TypedValue 对象的值关联的架构。

## 示例

此示例将在“输出”面板中显示 null：

```
var t: TypedValue = new TypedValue (true, "Boolean", null);
trace(t.type);
```

# TypedValue.typeName

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*typedValueObject.typeName*

## 说明

属性；包含 TypedValue 对象的值的数据类型名称。

## 示例

此示例将在“输出”面板中显示 Boolean：

```
var t: TypedValue = new TypedValue (true, "Boolean", null);
trace(t.typeName);
```

# TypedValue.value

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*typedValueObject.value*

## 说明

属性；包含 TypedValue 对象的数据值。

## 示例

此示例将在“输出”面板中显示 true：

```
var t: TypedValue = new TypedValue (true, "Boolean", null);
trace(t.value);
```

# DataGrid 组件（仅限 Flash Professional）

**DataGrid** 组件使您能够创建强大的数据驱动的显示和应用程序。可以使用 **DataGrid** 组件来实例化使用 **Macromedia Flash Remoting** 的记录集（从 **Macromedia ColdFusion**、**Java** 或 **.Net** 中的数据库查询中检索），然后将其显示在列中。您也可以使用数据集或数组中的数据来填充 **DataGrid** 组件。第 2 版 **DataGrid** 组件已进行改进，包括了水平滚动、更好的事件支持（包括对可编辑单元格的事件支持）、增强的排序功能以及性能优化。

您可以调整和自定义诸如网格中的字体、颜色以及列边框等特征。对于网格中的任何列，您都可以使用自定义影片剪辑作为单元格渲染器。（单元格渲染器显示单元格的内容。）可以使用滚动条来浏览网格中的数据；也可以禁用滚动条并使用 **DataGrid** 方法来创建页面视图样式的显示。有关自定义的更多信息，请参见第 280 页的“**DataGridColumn** 类（仅限 **Flash Professional**）”。

将 **DataGrid** 组件添加到应用程序时，可以使用“辅助功能”面板使其可由屏幕读取器访问。首先，您必须添加以下代码行，以便为 **DataGrid** 组件启用辅助功能：

```
mx.accessibility.DataGridAccImpl.enableAccessibility();
```

无论组件有多少实例，都只对组件启用一次辅助功能。有关更多信息，请参见《使用 **Flash**》中的第 19 章“创建辅助内容”。

# 与 DataGrid 组件进行交互（仅限 Flash Professional）

可以使用鼠标和键盘与 DataGrid 组件进行交互。

如果 `DataGrid.sortableColumns` 和 `DataGridColumn.sortOnHeaderRelease` 均为 `true`，则在某一列标题中单击会导致网格基于该列的单元格值进行排序。

如果 `DataGrid.resizableColumns` 为 `true`，则在列之间的区域中单击可允许您调整列的大小。

在某个可编辑单元格内单击会将焦点发送给该单元格；单击不可编辑的单元格不会影响焦点。如果某个单元格的 `DataGrid.editable` 和 `DataGridColumn.editable` 属性均为 `true`，则该单元格是可编辑的。

当 DataGrid 实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

| 键                              | 说明                                                                                      |
|--------------------------------|-----------------------------------------------------------------------------------------|
| 向下箭头                           | 如果正在编辑单元格，插入点将移到单元格文本的末尾。如果单元格不可编辑，则向下箭头处理选区的方式与 List 组件相同。                             |
| 向上箭头                           | 如果正在编辑单元格，插入点将移到单元格文本的开头。如果单元格不可编辑，则向上箭头处理选区的方式与 List 组件相同。                             |
| 向右箭头                           | 如果正在编辑单元格，插入点将向右移动一个字符。如果单元格不可编辑，向右箭头不会执行任何操作。                                          |
| 向左箭头                           | 如果正在编辑单元格，插入点将向左移动一个字符。如果单元格不可编辑，向左箭头不会执行任何操作。                                          |
| Return/Enter/<br>Shift+Enter 键 | 如果单元格可编辑，则会提交更改，并且插入点将移到同一列中单元格的下一行（向上或向下，视 shift 切换而定）。                                |
| Shift+Tab/Tab 键                | 将焦点移到前一项。当按下 Tab 键时，焦点将从网格中的最后一列回绕到下一行上的第一列。当按下 Shift+Tab 键时，回绕将反向进行。已选中获得焦点的单元格中的所有文本。 |

# 使用 DataGrid 组件（仅限 Flash Professional）

可以使用 **DataGrid** 组件作为许多种数据驱动应用程序的基础。您不但可以轻松地显示数据库查询（或其它数据）的格式化表格视图，而且可以使用单元格渲染器功能建立更为复杂和可编辑的用户界面片段。以下是 **DataGrid** 组件的实际用途：

- **Webmail** 客户端
- 搜索结果页
- 电子表格应用程序，如借贷计算器和纳税申请表应用程序。

## 了解 DataGrid 组件的设计

**DataGrid** 组件扩展了 [List 组件](#)。当使用 **DataGrid** 组件设计应用程序时，了解它下层的 **List** 类的设计机制会很有帮助。下面是 **Macromedia** 在开发 **List** 类时使用的一些基本假设和要求：

- 保持其短小、快速与简洁。  
不要毫无必要地使事情复杂化。这是首要的设计准则。下面列出的大多数要求都是基于此准则的。
- 列表具有统一的行高。  
每行的高度必须相同，该高度可以在创作期间或者在运行时设置。
- 列表必须能放大到容纳数千条记录。
- 列表不度量文本。

这给 **List** 和 **Tree** 组件带来了一个水平滚动的问题；有关更多信息，请参见第 704 页的“[了解 List 组件的设计](#)”。不过，**DataGrid** 组件支持将 "auto" 作为 `hScrollPolicy` 值，因为它度量列（每一项的列宽都相等）而不度量文本。

列表不度量文本这一事实解释了列表具有统一行高的原因。调整每行的大小来适合文本将需要进行大量的度量工作。例如，如果要在行高不统一的列表上精确地显示滚动条，就需要预先度量每个行。

- 可见行越多，列表性能越差。  
虽然列表能显示 5000 条记录，但它们不能一次呈现 5000 条记录。舞台上的可见行（通过 `rowCount` 属性指定）越多，列表呈现它们所需的工作也越多。如果可能，限制可见行的数量是最好的解决方案。

- 列表不是表。

**DataGrid** 组件用于为许多记录提供一个界面。它们的设计意图不是为了显示完整的信息，而是用于显示够用的信息，这样用户可以进一步查看更多信息。**Microsoft Outlook** 中的邮件视图就是一个很好的示例。您不会在网格中阅读整封电子邮件；否则邮件将难以阅读，而且也不便于用户操作。**Outlook** 只显示够用的信息，用户可以深入到邮件内部去查看详细信息。

## 了解 DataGrid 组件：数据模型和视图

从概念上来说，**DataGrid** 组件由数据模型和显示数据的视图组成。数据模型包含以下三个主要部分：

- **DataProvider**

用于填充数据网格的项目列表。同一帧中作为 **DataGrid** 组件的任何数组都会自动（从 **DataProvider API** 中）获得一些方法，这些方法允许您处理数据并将更改广播给多个视图。可以将实现 **DataProvider API** 的任何对象分配给 `DataGrid.dataProvider` 属性（其中包括记录集、数据集等）。以下代码将创建名为 `myDP` 的数据提供程序：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",
 price:"Cheap"});
```

- 项

用于在列的单元格中存储信息单元的 **ActionScript** 对象。数据网格实际上是一个可以显示多列数据的列表。可以将列表看作一个数组；列表的每个索引空间就是一个项。对于 **DataGrid** 组件，每个项均由多个字段组成。在以下代码中，大括号 (`{}`) 之间的内容就是一个项：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",
 price:"Cheap"});
```

- 字段

指明各项内列名称的标识符。它与列列表中的 `columnNames` 属性相对应。在 **List** 组件中，字段通常为 `label` 和 `data`，但在 **DataGrid** 组件中，字段可以是任何标识符。在以下代码中，字段为 `name`（名称）和 `price`（价格）：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",
 price:"Cheap"});
```

视图包含三个主要部分：

- 行

负责通过布置单元格来呈现网格项目的视图对象。每一行都水平布置在前一行的下方。

- 列

列是显示在网格中的字段，每个字段对应于每一列的 `columnName` 属性。

列是负责显示每一列的视图对象（`DataGridColumn` 类的实例），例如宽度、颜色、大小等。

可以使用三种方式将列添加到数据网格中：将 `DataProvider` 对象分配给 `DataGrid.dataProvider`（此操作会自动为第一项中的每个字段生成一列）；设置 `DataGrid.columnNames` 以指定将显示哪些字段；或者使用 `DataGridColumn` 类的构造函数来创建列，并调用 `DataGrid.addColumn()` 以将各列添加到网格中。

若要设置列的格式，请为整个数据网格设置样式属性，或者定义 `DataGridColumn` 对象，单独设置它们的样式格式，然后将它们添加到数据网格中。

- 单元格

负责呈现每一项的个别字段的视图对象。若要与数据网格通信，这些组件必须实现 `CellRenderer` API（请参见第 101 页的“[CellRenderer API](#)”）。对于基本数据网格，单元格是内置的 `ActionScript TextField` 对象。

## DataGrid 参数

您可以在“属性”检查器或“组件”检查器中，为每个 `DataGrid` 组件实例设置以下创作参数：

**editable** 是一个布尔值，它指示网格是 (true) 否 (false) 可编辑。默认值为 false。

**multipleSelection** 是一个布尔值，它指示是 (true) 否 (false) 可以选择多项。默认值为 false。

**rowHeight** 指示每行的高度（以像素为单位）。更改字体大小不会更改行高度。默认值为 20。

您可以编写 `ActionScript`，以便使用其属性、方法和事件来控制 `DataGrid` 组件的这些和其它选项。有关更多信息，请参见第 243 页的“[DataGrid 类（仅限 Flash Professional）](#)”。

## 创建具有 DataGrid 组件的应用程序

若要创建具有 DataGrid 组件的应用程序，您必须首先确定数据的来源。网格的数据可以来源于使用 Flash Remoting 从 Macromedia ColdFusion、Java 或 .Net 内的数据库查询中馈入的记录集，也可以来源于数据集或数组。若要将数据拉到网格中，您需要将 `DataGrid.dataProvider` 属性设置为记录集、数据集或数组。也可以使用 `DataGrid` 和 `DataGridColumn` 类的方法以本地方式创建数据。与 DataGrid 组件在同一帧中的任何 Array 对象均会复制 DataProvider API 的方法、属性和事件。

提醒

在使用 Data 组件将数据绑定到 DataGrid 组件时，该对象会逆向绑定各个列（类似于循环访问对象或数组）。因此，若要以不同方式对 DataGrid 组件中的数据进行排序，就必须显式定义各个列。

### 使用 Flash Remoting 将 DataGrid 组件添加到应用程序：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 在“组件”面板中，双击 DataGrid 组件以将其添加到舞台上。
3. 在“属性”检查器中，输入实例名称 **myDataGrid**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```
myDataGrid.dataProvider = recordSetInstance;
```

Flash Remoting 记录集 recordSetInstance 即会分配给 myDataGrid 的 dataProvider 属性。

5. 选择“控制”>“测试影片”。

### 使用本地数据提供程序将 DataGrid 组件添加到应用程序中：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 在“组件”面板中，双击 DataGrid 组件以将其添加到舞台上。
3. 在“属性”检查器中，输入实例名称 **myDataGrid**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",
 price:"Cheap"});
myDataGrid.dataProvider = myDP;
```

name（名称）和 price（价格）字段被用作列标题，它们的值将填充每一行中的单元格。

5. 选择“控制”>“测试影片”。



为应用程序中的 **DataGrid** 组件指定列和添加排序：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 在“组件”面板中，双击 **DataGrid** 组件以将其添加到舞台上。
3. 在“属性”检查器中，输入实例名称 **myDataGrid**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```
var myDataGrid:mx.controls.DataGrid;

// 创建列以启用数据的排序。
myDataGrid.addColumn("name");
myDataGrid.addColumn("score");

var myDP_array:Array = new Array({name:"Clark", score:3135},
 {name:"Bruce", score:403}, {name:"Peter", score:25})

myDataGrid.dataProvider = myDP_array;

// 为 DataGrid 创建侦听器对象。
var listener_obj:Object = new Object();
listener_obj.headerRelease = function(evt_obj:Object) {
 switch (evt_obj.target.columns[evt_obj.columnIndex].columnName) {
 case "name" :
 myDP_array.sortOn("name", Array.CASEINSENSITIVE);
 break;
 case "score" :
 myDP_array.sortOn("score", Array.NUMERIC);
 break;
 }
};

// 为 DataGrid 添加侦听器。
myDataGrid.addEventListener("headerRelease", listener_obj);
```

5. 选择“控制”>“测试影片”。

使用 **ActionScript** 创建 **DataGrid** 组件实例：

1. 将 **DataGrid** 组件从“组件”面板拖到当前文档的库中。

此操作将组件添加到库中，但不会在应用程序中显示。

2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
this.createClassObject(mx.controls.DataGrid, "my_dg", 10,
 {columnNames:["name", "score"]});
my_dg.setSize(140, 100);
my_dg.move(10, 40);
```

此脚本使用 `UIObject.createClassObject()` 方法创建 **DataGrid** 实例，然后设置网格大小并定位网格。

3. 创建一个数组，向数组中添加数据，并将该数组标识为数据网格的数据提供程序：

```
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;
```

4. 选择“控制” > “测试影片”。

## DataGrid 性能策略

由于显示的数据可以缩放大小，因此在使用 **DataGrid** 组件时，性能很快会成为一个主要的关注点。在速度相当快的计算机（使用到数据源的快速连接）上，显示了一百行的数据网格看上去可能相当不错。但一个月后，当数据增加到几千行时，用户可能会遇到迥然不同的情况。此外，如果到数据源的连接速度较慢，用户可能会遇到计算机速度较慢的情况。

以下是使用 **DataGrid** 组件时避免常见性能缺陷的一些建议。

- 构建并绑定一个数据结构，而不是直接添加列。

向 **DataGrid** 组件中添加列和数据有两种方法：一种是通过 `DataGrid.dataProvider` 属性绑定预先构建的数据结构（一个对象数组），一种是使用 **DataGrid** 类方法（如 `DataGrid.addColumn()` 和 `DataGrid.addItem()`）。应尽可能地使用第一种方法，即使使用 `DataGrid.dataProvider` 属性绑定到预先构建的数据结构，因为这使得 **DataGrid** 可以在尝试在屏幕上绘制它所需的所有列之前创建这些列。

您可能很想创建 `for` 循环来对所需的所有列调用 `DataGrid.addColumn()`。虽然这种方法看起来是一种简单易懂的方法，但请不要使用该方法。原因是每次调用 `DataGrid.addColumn()` 时，数据网格都会添加事件侦听器、排序并重绘其自身以呈现新列。使用 `DataGrid.addColumn()` 创建 20 个列会导致 **DataGrid** 对自身排序并不必要地重绘 20 次。在 **ActionScript** 中构建数据结构不需要呈现或说明事件。在将该数据结构分配给 **DataGrid** 组件的 `dataProvider` 属性时，只需一次传递即可完成所有绘制。

- 为详细数据提供进一步查看机制。

**DataGrid** 组件接口使用户可以快速地执行搜索，以便搜索到更多的详细信息。只提供执行最初搜索所需的数据，然后可在另一个搜索步骤中提供任何特定行或单元格的详细信息。这种处理办法不仅最大限度地减少了填充数据网格所需的初始数据，而且最大限度地减少了用户必须浏览才能找到所需内容的信息量。在数据网格中选择了感兴趣的行或项后，可以再次对数据源进行调用以获取相关的详细信息。这些详细信息还可以在某些其它用户界面机制中（如多行文本字段和图形的集合）更好地显示。

- 避免数据源和数据网格之间数据操作的循环。

如果可能并且能满足长期数据库的需要，则按照数据显示时的非常相似的格式和顺序来存储数据可以避免不必要的内存分配并节省用户计算机上的处理时间，而且能加快数据网格的响应时间。

- 避免使用可返回数据库中每一行的查询。

用户很少希望在每次访问数据时都查看每一条可用的记录。重要的是了解数据使用者要查找的内容并为他们提供缩小搜索范围的方法。如果他们通常只查看给定周内特定主题的最新记录，则默认情况下只显示相对较小的数据组，并能够扩展他们查看数据的范围。

如果数据量可能非常大，请考虑使用分页功能来限制其大小，方法是提供可能通常从查询返回的数据子集。例如，对于某个查询可能从数据库中返回的 10,000 行数据，不必一次全部查看，而可以调用前 20 行的子集，然后利用附加的导航按钮触发一个调用以用下面的 20 条记录来填充数据网格。

- 区分数据处理与 `CellRenderer` 处理。

通过 `CellRenderer` API，可以在数据网格中显示自定义单元格内容。有一种功能要求可能需要有条件地使用 `ComboBox` 组件或其它 UI 控件来填充数据网格。例如，根据在第二列中选择的内容，可以在第四列中重新填充或自动选择选项。尽可能将控件的此条件逻辑和重新填充操作与单元格的内容呈现过程区分开来，这一点是非常重要的。鼠标每次滑过单元格时，单元格即被选中，或者数据被更改，因此单元格中的内容或整个单元格有可能会重绘以保持最新状态。这意味着放入 `CellRenderer` 中的任何代码将不断重复地运行，因此应尽可能地减少在 `CellRenderer` 中执行的处理。如果您必须向 `CellRenderer` 中添加代码，则最好从 `CellRenderer` 中调用一个函数，以检测需要进行哪些更新并用最有效的方法进行更新。

- 在数据网格完成绘制后，使用 `UIObject.doLater()` 访问其属性。

数据网格实例需要绘制并加载完数据后，您才能访问数据网格的所有属性（如 `focusedCell` 和其它属性）。由于 `DataGrid` 没有“complete”事件，因此可以改用 `UIObject.doLater()` 来调用访问数据网格属性的函数。`UIObject.doLater()` 只在数据网格的属性可用后才执行该函数。若要查看示例，请参见 `DataGrid.focusedCell`。

# 自定义 DataGrid 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平和垂直方向上将 **DataGrid** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参见 `UIObject.setSize()`）。如果没有水平滚动条，列宽度将按比例进行调整。如果调整了列大小（并因此调整了单元格大小），则单元格中的文本可能会被裁剪。

## 对 DataGrid 组件使用样式

您可以设置样式属性以更改 **DataGrid** 组件的外观。**DataGrid** 组件从 **List** 组件继承样式。（请参见第 708 页的“对 **List** 组件使用样式”。）**DataGrid** 组件还支持以下样式：

| 样式                                   | 主题    | 说明                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>backgroundColor</code>         | 光晕和范例 | 可以为整个网格或每一列设置背景颜色。                                                                                                                                                                                                                                                                                                                                                                  |
| <code>backgroundDisabledColor</code> | 光晕和范例 | 当组件的 <code>enabled</code> 属性设置为 <code>"false"</code> 时的背景颜色。默认值为 <code>OxDDDDDD</code> （中度灰）。                                                                                                                                                                                                                                                                                       |
| <code>borderStyle</code>             | 光晕和范例 | <b>DataGrid</b> 组件使用 <code>RectBorder</code> 实例作为其边框，并对该类定义的样式做出响应。请参见第 985 页的“ <b>RectBorder</b> 类”。默认的边框样式值是 <code>"inset"</code> 。                                                                                                                                                                                                                                               |
| <code>headerColor</code>             | 光晕和范例 | 列标题的颜色。默认值为 <code>OxEAEAEA</code> （浅灰）                                                                                                                                                                                                                                                                                                                                              |
| <code>headerStyle</code>             | 光晕和范例 | 可以将列标题的 CSS 样式声明应用到网格或列以自定义标题样式。                                                                                                                                                                                                                                                                                                                                                    |
| <code>color</code>                   | 光晕和范例 | 文本颜色。“光晕”主题的默认值为 <code>Ox0B333C</code> ，“范例”主题的默认值为空白。                                                                                                                                                                                                                                                                                                                              |
| <code>disabledColor</code>           | 光晕和范例 | 组件禁用时的文本颜色。默认值为 <code>Ox848384</code> （深灰）。                                                                                                                                                                                                                                                                                                                                         |
| <code>embedFonts</code>              | 光晕和范例 | 一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。例如（使用 <b>DataGrid</b> 实例 <code>my_dg</code> ）：<br><code>my_dg.setStyle("fontFamily", "yourFont");</code><br><code>my_dg.embedFonts=true;</code><br>否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。 |
| <code>fontFamily</code>              | 光晕和范例 | 文本的字体名称。默认值为 <code>"_sans"</code> 。                                                                                                                                                                                                                                                                                                                                                 |
| <code>fontSize</code>                | 光晕和范例 | 字体的磅值。默认值为 10。                                                                                                                                                                                                                                                                                                                                                                      |

| 样式             | 主题    | 说明                                                                                                                                            |
|----------------|-------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| fontStyle      | 光晕和范例 | 字体样式: "normal" 或 "italic"。默认值为 "normal"。                                                                                                      |
| fontWeight     | 光晕和范例 | 字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。 |
| textAlign      | 光晕和范例 | 文本对齐方式: "left"、"right" 或 "center"。默认值为 "left"。                                                                                                |
| textDecoration | 光晕和范例 | 文本修饰: "none" 或 "underline"。默认值为 "none"。                                                                                                       |
| vGridLines     | 光晕和范例 | 一个布尔值, 指示是 (true) 否 (false) 显示垂直网格线。默认值为 true。                                                                                                |
| hGridLines     | 光晕和范例 | 一个布尔值, 指示是 (true) 否 (false) 显示水平网格线。默认值为 false。                                                                                               |
| vGridLineColor | 光晕和范例 | 垂直网格线的颜色。默认值为 0x666666 (中度灰)。                                                                                                                 |
| hGridLineColor | 光晕和范例 | 水平网格线的颜色。默认值为 0x666666 (中度灰)。                                                                                                                 |

## 为单独的列设置样式

可以为整个网格或为某一列设置颜色和文本样式。可以使用下面的语法为特定的列设置样式:

```
grid.getColumnAt(3).setStyle("backgroundColor", 0xFF00AA);
```

## 设置标题样式

可以通过 `headerStyle` 设置标题样式, 它本身就是一个样式属性。为此, 请创建 `CSSStyleDeclaration` 的一个实例, 并为该实例设置相应的标题属性, 然后将 `CSSStyleDeclaration` 指定到 `headerStyle` 属性, 如下示例所示。

```
import mx.styles.CSSStyleDeclaration;
var headerStyles = new CSSStyleDeclaration();
headerStyles.setStyle("fontStyle", "italic");
grid.setStyle("headerStyle", headerStyles);
```

## 为文档中的所有 DataGrid 组件设置样式

**DataGrid** 类继承自 **List** 类，而 **List** 类又继承自 **ScrollSelectList** 类。默认的分类样式属性在 **ScrollSelectList** 类上定义，而 **Menu** 组件和所有基于列表的组件均扩展了 **ScrollSelectList** 类。您可以直接对此类设置新的默认样式值，这些新设置将反映在所有受影响的组件中。

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

若要仅设置 **DataGrid** 组件的样式属性，您可以创建一个新的 **CSSStyleDeclaration** 实例，并将其存储在 `_global.styles.DataGrid` 中。

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.DataGrid == undefined) {
 _global.styles.DataGrid = new CSSStyleDeclaration();
}
_global.styles.DataGrid.setStyle("backgroundColor", 0xFF00AA);
```

在创建新的分类样式声明时，由 **ScrollSelectList** 声明提供的所有默认值都将丢失，包括支持鼠标事件所必需的 `backgroundColor`。若要创建分类样式声明并保留默认值，请使用 `for..in` 循环，将旧的设置复制到新的声明。

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.DataGrid;
for (var style in source) {
 target.setStyle(style, source.getStyle(style));
}
```

有关分类样式的更多信息，请参见《使用组件》中的“为组件类设置样式”。

## 对 DataGrid 组件使用外观

**DataGrid** 组件用于表示其可视状态的外观包括在构成数据网格所依据的子组件内（滚动条和 **RectBorder**）。有关这些子组件的外观的信息，请参见第 1283 页的“对 **UIScrollBar** 组件使用外观”和第 985 页的“**RectBorder** 类”。

# DataGrid 类（仅限 Flash Professional）

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > View > ScrollView > ScrollSelectList > [List 组件](#) > DataGrid

ActionScript 类名称 mx.controls.DataGrid

每个组件类都有一个 version 属性，而该属性是一个类属性。类属性只能用于该类本身。version 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.DataGrid.version);
```

提醒

代码 trace(myDataGridInstance.version); 返回 undefined。

## DataGrid 类的方法摘要

下表列出了 DataGrid 类的方法。

| 方法                                             | 说明                               |
|------------------------------------------------|----------------------------------|
| <a href="#">DataGrid.addColumn()</a>           | 将列添加到数据网格。                       |
| <a href="#">DataGrid.addColumnAt()</a>         | 将列添加到数据网格的指定位置。                  |
| <a href="#">DataGrid.addItem()</a>             | 将项目添加到数据网格。                      |
| <a href="#">DataGrid.addItemAt()</a>           | 将项目添加到数据网格的指定位置。                 |
| <a href="#">DataGrid.editField()</a>           | 替换位于指定位置的单元格数据。                  |
| <a href="#">DataGrid.getColumnAt()</a>         | 获取对位于指定位置的列的引用。                  |
| <a href="#">DataGrid.getColumnIndex()</a>      | 获取对位于指定索引的 DataGridColumn 对象的引用。 |
| <a href="#">DataGrid.removeAllColumns()</a>    | 从数据网格中删除所有列。                     |
| <a href="#">DataGrid.removeColumnAt()</a>      | 从数据网格的指定位置删除列。                   |
| <a href="#">DataGrid.replaceItemAt()</a>       | 使用另一项目替换位于指定位置的项目。               |
| <a href="#">DataGrid.spaceColumnsEqually()</a> | 平均间隔所有列。                         |

## 从 UIObject 类继承的方法

下表列出了 **DataGrid** 类从 **UIObject** 类继承的方法。调用这些方法时，请使用 *dataGridInstance.methodName* 的形式。

| 方法                                        | 说明                               |
|-------------------------------------------|----------------------------------|
| <code>UIObject.createClassObject()</code> | 创建指定类的对象。                        |
| <code>UIObject.createObject()</code>      | 创建对象的子对象。                        |
| <code>UIObject.destroyObject()</code>     | 破坏组件实例。                          |
| <code>UIObject.doLater()</code>           | 在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。 |
| <code>UIObject.getStyle()</code>          | 从样式声明或对象获取样式属性。                  |
| <code>UIObject.invalidate()</code>        | 标记对象使其在到达下一个帧间隔时进行重绘。            |
| <code>UIObject.move()</code>              | 将对象移动到要求的位置。                     |
| <code>UIObject.redraw()</code>            | 迫使对象有效以便能在当前帧中绘制。                |
| <code>UIObject.setSize()</code>           | 将对象调整为所要求的大小。                    |
| <code>UIObject.setSkin()</code>           | 设置对象的外观。                         |
| <code>UIObject.setStyle()</code>          | 设置样式声明或对象的样式属性。                  |

## 从 UIComponent 类继承的方法

下表列出了 **DataGrid** 类从 **UIComponent** 类继承的方法。调用这些方法时，请使用 *dataGridInstance.methodName* 的形式。

| 方法                                  | 说明             |
|-------------------------------------|----------------|
| <code>UIComponent.getFocus()</code> | 返回对具有焦点的对象的引用。 |
| <code>UIComponent.setFocus()</code> | 将焦点设置到组件实例中。   |



## 从 List 类继承的方法

下表列出了 **DataGrid** 类从 **List** 类继承的方法。调用这些方法时，请使用 *dataGridInstance.methodName* 的形式。

| 方法                                     | 说明                    |
|----------------------------------------|-----------------------|
| <a href="#">List.addItem()</a>         | 向列表的结尾添加项目。           |
| <a href="#">List.addItemAt()</a>       | 将项目添加到指定索引处的列表。       |
| <a href="#">List.getItemAt()</a>       | 返回指定索引处的项目。           |
| <a href="#">List.removeAll()</a>       | 删除列表中的所有项目。           |
| <a href="#">List.removeItemAt()</a>    | 删除指定索引处的项目。           |
| <a href="#">List.replaceItemAt()</a>   | 用其它项目替换指定索引处的项目。      |
| <a href="#">List.setPropertiesAt()</a> | 将指定的属性应用到指定的项目。       |
| <a href="#">List.sortItems()</a>       | 按照指定的比较函数对列表中的项目进行排序。 |
| <a href="#">List.sortItemsBy()</a>     | 按照指定的属性对列表中的项目进行排序。   |

## DataGrid 类的属性摘要

下表列出了 **DataGrid** 类的属性。

| 属性                                        | 说明                                                |
|-------------------------------------------|---------------------------------------------------|
| <a href="#">DataGrid.columnCount</a>      | 只读；显示的列的数目。                                       |
| <a href="#">DataGrid.columnNames</a>      | 显示为列的每个项目内的字段名数组。                                 |
| <a href="#">DataGrid.dataProvider</a>     | 数据网格的数据模型。                                        |
| <a href="#">DataGrid.editable</a>         | 一个布尔值，指示数据网格是 (true) 否 (false) 可编辑。               |
| <a href="#">DataGrid.focusedCell</a>      | 定义具有焦点的单元格。                                       |
| <a href="#">DataGrid.headerHeight</a>     | 列标题的高度（以像素为单位）。                                   |
| <a href="#">DataGrid.hScrollPolicy</a>    | 指示是显示 ("on")、不显示 ("off") 还是在必要时显示 ("auto") 水平滚动条。 |
| <a href="#">DataGrid.resizableColumns</a> | 一个布尔值，它指示列是 (true) 否 (false) 可以调整大小。              |
| <a href="#">DataGrid.selectable</a>       | 一个布尔值，它指示数据网格是 (true) 否 (false) 可选择。              |
| <a href="#">DataGrid.showHeaders</a>      | 一个布尔值，它指示列标题是 (true) 否 (false) 可见。                |
| <a href="#">DataGrid.sortableColumns</a>  | 一个布尔值，它指示列是 (true) 否 (false) 可排序。                 |

### 从 UIObject 类继承的属性

下表列出了 **DataGrid** 类从 **UIObject** 类继承的属性。从 **DataGrid** 对象访问这些属性时，请使用 `dataGridInstance.propertyName` 的形式。

| 属性                            | 说明                                     |
|-------------------------------|----------------------------------------|
| <code>UIObject.bottom</code>  | 对象的底边缘位置（相对于其父对象的底边缘）。只读。              |
| <code>UIObject.height</code>  | 对象的高度（以像素为单位）。只读。                      |
| <code>UIObject.left</code>    | 对象的左边缘（以像素为单位）。只读。                     |
| <code>UIObject.right</code>   | 对象的右边缘位置（相对于其父对象的右边缘）。只读。              |
| <code>UIObject.scaleX</code>  | 一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。         |
| <code>UIObject.scaleY</code>  | 一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。         |
| <code>UIObject.top</code>     | 对象上边缘的位置（相对于其父对象）。只读。                  |
| <code>UIObject.visible</code> | 一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。 |
| <code>UIObject.width</code>   | 对象的宽度（以像素为单位）。只读。                      |
| <code>UIObject.x</code>       | 对象的左边缘（以像素为单位）。只读。                     |
| <code>UIObject.y</code>       | 对象的上边缘（以像素为单位）。只读。                     |

### 从 UIComponent 类继承的属性

下表列出了 **DataGrid** 类从 **UIComponent** 类继承的属性。从 **DataGrid** 对象访问这些属性时，请使用 `dataGridInstance.propertyName` 的形式。

| 属性                                | 说明                     |
|-----------------------------------|------------------------|
| <code>UIComponent.enabled</code>  | 指明组件是否可以接收焦点和输入。       |
| <code>UIComponent.tabIndex</code> | 一个数字，指明文档中组件的 Tab 键顺序。 |

## 从 List 类继承的属性

下表列出了 **DataGrid** 类从 **List** 类继承的属性。从 **DataGrid** 对象访问这些属性时，请使用 *dataGridInstance.propertyName* 的形式。

| 属性                                     | 说明                                                                |
|----------------------------------------|-------------------------------------------------------------------|
| <a href="#">List.cellRenderer</a>      | 指定要使用的类或元件以显示列表的每一行。                                              |
| <a href="#">List.dataProvider</a>      | 列表项目的来源。                                                          |
| <a href="#">List.hPosition</a>         | 列表的水平位置。                                                          |
| <a href="#">List.hScrollPolicy</a>     | 指示是 ("on") 否 ("off") 显示水平滚动条。                                     |
| <a href="#">List.iconField</a>         | 各项目中用于指定图标字段。                                                     |
| <a href="#">List.iconFunction</a>      | 一个函数，它确定要使用的图标。                                                   |
| <a href="#">List.labelField</a>        | 指定各项目中用作标签文本的字段。                                                  |
| <a href="#">List.labelFunction</a>     | 一个函数，它确定各个项目的哪些字段要用作标签文本。                                         |
| <a href="#">List.length</a>            | 列表中的项目数。该属性为只读。                                                   |
| <a href="#">List.maxHPosition</a>      | 当将 <a href="#">List.hScrollPolicy</a> 设置为 "on" 时，指定列表可以向右滚动的像素数目。 |
| <a href="#">List.multipleSelection</a> | 指示列表中是 (true) 否 (false) 允许多选。                                     |
| <a href="#">List.rowCount</a>          | 列表中至少可以看到一部分的行数。                                                  |
| <a href="#">List.rowHeight</a>         | 列表中每行的像素高度。                                                       |
| <a href="#">List.selectable</a>        | 指示列表是 (true) 否 (false) 为可选择列表。                                    |
| <a href="#">List.selectedIndex</a>     | 单选列表中的选择索引。                                                       |
| <a href="#">List.selectedIndices</a>   | 多选列表中的已选择项目的数组。                                                   |
| <a href="#">List.selectedItem</a>      | 单选列表中的已选择项目。该属性为只读。                                               |
| <a href="#">List.selectedItems</a>     | 多选列表中的已选择的项目对象。该属性为只读。                                            |
| <a href="#">List.vPosition</a>         | 滚动列表，以便使最顶部可见的项目为指定的数。                                            |
| <a href="#">List.vScrollPolicy</a>     | 指示是显示 ("on")、不显示 ("off") 还是在需要时显示 ("auto") 垂直滚动条。                 |

# DataGrid 类的事件摘要

下表列出了 DataGrid 类的事件。

| 事件                                  | 说明                 |
|-------------------------------------|--------------------|
| <code>DataGrid.cellEdit</code>      | 在单元格值更改时广播。        |
| <code>DataGrid.cellFocusIn</code>   | 在单元格获得焦点时广播。       |
| <code>DataGrid.cellFocusOut</code>  | 在单元格失去焦点时广播。       |
| <code>DataGrid.cellPress</code>     | 在单元格被按下（单击）时广播。    |
| <code>DataGrid.change</code>        | 在选中项目时广播。          |
| <code>DataGrid.columnStretch</code> | 当用户在水平方向调整列的大小时广播。 |
| <code>DataGrid.headerRelease</code> | 在用户按下（或松开）标题时广播。   |

## 从 UIObject 类继承的事件

下表列出了 DataGrid 类从 UIObject 类继承的事件。

| 事件                           | 说明                 |
|------------------------------|--------------------|
| <code>UIObject.draw</code>   | 当对象将要绘制它的图形时进行广播。  |
| <code>UIObject.hide</code>   | 在对象的状态从可见变为不可见时广播。 |
| <code>UIObject.load</code>   | 创建子对象时广播。          |
| <code>UIObject.move</code>   | 移动了对象时广播。          |
| <code>UIObject.resize</code> | 在调整对象大小后广播。        |
| <code>UIObject.reveal</code> | 在对象的状态从不可见变为可见时广播。 |
| <code>UIObject.unload</code> | 卸载子对象时广播。          |

## 从 UIComponent 类继承的事件

下表列出了 DataGrid 类从 UIComponent 类继承的事件。

| 事件                                | 说明            |
|-----------------------------------|---------------|
| <code>UIComponent.focusIn</code>  | 当对象收到焦点时进行广播。 |
| <code>UIComponent.focusOut</code> | 当对象失去焦点时进行广播。 |
| <code>UIComponent.keyDown</code>  | 当按下按键时进行广播。   |
| <code>UIComponent.keyUp</code>    | 当松开按键时进行广播。   |

## 从 List 类继承的事件

下表列出了 DataGrid 类从 List 类继承的事件。

| 事件                                | 说明                   |
|-----------------------------------|----------------------|
| <a href="#">List.change</a>       | 只要用户交互造成选择更改就广播。     |
| <a href="#">List.itemRollOut</a>  | 鼠标指针在列表项上滑过然后又滑离时广播。 |
| <a href="#">List.itemRollOver</a> | 当鼠标指针滑过列表项时进行广播。     |
| <a href="#">List.scroll</a>       | 滚动列表时，进行广播。          |

# DataGrid.addColumn()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*myDataGrid.addColumn(dataGridColumn)*

*myDataGrid.addColumn(name)*

### 参数

*dataGridColumn* DataGridColumn 类的一个实例。

*name* 一个字符串，它指示要插入的新 DataGridColumn 对象的名称。

### 返回

对所添加的 DataGridColumn 对象的引用，或者返回指示新列名称的字符串。

### 说明

方法：将新列添加到数据网格的结尾。有关更多信息，请参见第 280 页的“DataGridColumn 类（仅限 Flash Professional）”。

## 示例

此示例显示为 **DataGrid** 组件创建列的三种不同方式。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中（请注意，首先会导入 **DataGridColumn** 类）：

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;
my_dg.setSize(320, 240);

// 向网格中添加列。
my_dg.addColumn("Red");

// 向网格中添加另一列。
my_dg.addColumn(new DataGridColumn("Green"));

// 向网格中添加再一列。
var blue_dgc:DataGridColumn = new DataGridColumn("Blue");
blue_dgc.width = 140;
blue_dgc.headerText = "Blue Column:";
my_dg.addColumn(blue_dgc);
```

# DataGrid.addColumnAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.addColumnAt(index, name)
myDataGrid.addColumnAt(index, dataGridColumn)
```

## 参数

*index* 索引位置，**DataGridColumn** 对象将添加在该处。第一个位置是 **0**。

*name* 一个字符串，它指示 **DataGridColumn** 对象的名称。

*dataGridColumn* **DataGridColumn** 类的一个实例。

## 返回

对所添加的 **DataGridColumn** 对象的引用，或者返回指示新列名称的字符串。

## 说明

方法：在指定位置添加新列。列将向右移动，并且其索引值会增加。有关更多信息，请参见第 280 页的“[DataGridColumn 类（仅限 Flash Professional）](#)”。

## 示例

此示例显示 `addColumnAt()` 的两种用法，并设置列宽。在舞台上具有一个名为 `my_dg` 的 `DataGrid` 实例的情况下，将以下代码粘贴到主时间轴的第一帧中（请注意，首先会导入 `DataGridColumn` 类）：

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;
my_dg.setSize(320, 240);

// 向网格中添加列。
my_dg.addColumnAt(0, "Orange");
var orange_dgc:DataGridColumn = my_dg.getColumnAt(0);
orange_dgc.width = 125;

var blue_dgc:DataGridColumn = new DataGridColumn("Blue");
blue_dgc.width = 75;
my_dg.addColumnAt(1, blue_dgc);
```

# DataGrid.addItem()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.addItem(item)
```

## 参数

*item* 要添加到网格的对象的实例。

## 返回

指向已添加实例的引用。

## 说明

方法：将项目添加到网格的结尾（最后一个项目索引之后）。



此方法与 `List.addItem()` 方法不同，后者传递的是对象而不是字符串。

## 示例

此示例创建一个标题为 “**name**”（姓名）的列，然后为 “**name**”（姓名）插入 `item_obj` 值。注意，因为只定义了 **name** 列，所以会忽略 “**age**” 值。如果未指定列（删除 `addColumn` 行），则 **DataGrid** 会自动创建相应列。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 向网格中添加列并添加数据。
my_dg.addColumn("name");

var item_obj:Object = {name:"Jim", age:30};
my_dg.addItem(item_obj);
```

# DataGrid.addItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.addItemAt(index, item)
```

## 参数

*index* 索引位置（在多个子节点之间），节点应添加在该处。第一个位置是 **0**。

*item* 用于显示节点的字符串。

## 返回

指向已添加对象实例的引用。

## 说明

方法：将项目添加到网格的指定位置。



## 示例

此示例创建标题为 “name” 的一个列，用数组中的数据填充列，然后在第一行中添加名称 “Chase”。注意，因为只定义了 name 列，所以会忽略 “age” 值。如果未指定列（删除 addColumn 行），则 DataGrid 会自动创建相应列。在舞台上具有一个名为 my\_dg 的 DataGrid 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
var my_dg:mx.controls.DataGrid;

// 向网格中添加列并添加数据。
my_dg.addColumn("name");

var myDP_array:Array = new Array({name:"John", age:33}, {name:"Jose",
 age:41});
my_dg.dataProvider = myDP_array;

var item_obj:Object = {name:"Chase", age:30};
my_dg.addItemAt(0, item_obj);
```

# DataGrid.cellEdit

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.cellEdit = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("cellEdit", listenerObject)
```

## 说明

事件：在单元格值发生更改时向所有注册的侦听器广播。

第 2 版 **Macromedia Component Architecture** 组件使用调度程序 / 侦听器事件模型。

**DataGrid** 组件在单元格的值更改时调度 `cellEdit` 事件，而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

`DataGrid.cellEdit` 事件的事件对象具有四个附加属性：

`columnIndex` 一个数字，它指示目标列的索引。

`itemIndex` 一个数字，它指示目标行的索引。

`oldValue` 单元格先前的值。

`type` 字符串 "cellEdit"。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在以下示例中，定义了名为 `myDataGridListener` 的处理函数，并将其作为第二个参数传递给 `myDataGrid.addEventListener()`。`cellEdit` 处理函数在 *eventObject* 参数中捕获事件对象。广播 `cellEdit` 事件时（更改“**score**”（分数）值并按 **Enter** 键后），会将 `trace` 语句发送到“输出”面板。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(320, 240);
my_dg.editable = true;

// 添加列并使第一列不可编辑。
my_dg.addColumn("name");
my_dg.getColumnAt(0).editable = false;
my_dg.addColumn("score");

var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

// 设置 DataGrid 的数据源。
my_dg.dataProvider = myDP_array;

// 创建侦听器对象。
var myListener_obj:Object = new Object();
myListener_obj.cellEdit = function(evt_obj:Object) {
 // 检索已更改的单元格的位置。
 var cell_obj:Object = "("+evt_obj.columnIndex+", "+evt_obj.itemIndex+")";
```

```
// 检索已更改的单元格值。
var value_obj:Object = evt_obj.target.selectedItem.score;
trace("The value of the cell at "+cell_obj+" has changed to "+value_obj);
};

// 添加侦听器对象。
my_dg.addEventListener("cellEdit", myListener_obj);
```

## DataGrid.cellFocusIn

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
listenerObject = new Object();
listenerObject.cellFocusIn = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("cellFocusIn", listenerObject)
```

### 说明

事件：在特定单元格获得焦点时向所有注册的侦听器广播。在广播了之前编辑的任何单元格的 `editCell` 和 `cellFocusOut` 事件后，将广播此事件。

第 2 版组件使用调度程序 / 侦听器事件模型。在 **DataGrid** 组件调度 `cellFocusIn` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

`DataGrid.cellFocusIn` 事件的事件对象具有三个附加属性：

`columnIndex` 一个数字，它指示目标列的索引。

`itemIndex` 一个数字，它指示目标行的索引。

`type` 字符串 `"cellFocusIn"`。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在以下示例中，定义了名为 dgListener 的处理函数，并将其作为第二个参数传递给了 my\_dg.addEventListener()。在广播 cellFocusIn 事件时，会将 trace 语句发送到“输出”面板。在舞台上具有一个名为 my\_dg 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

// 使 DataGrid 可编辑。
my_dg.editable = true;

// 创建侦听器对象。
var dgListener:Object = new Object();
dgListener.cellFocusIn = function(evt_obj:Object) {
 var cell_str:String = "(" + evt_obj.columnIndex + ", " +
 evt_obj.itemIndex + ")";
 trace("The cell at " + cell_str + " has gained focus");
};

// 添加侦听器。
my_dg.addEventListener("cellFocusIn", dgListener);
```

**提醒**

若要使以上代码可以工作，网格必须是可编辑的，并且只对可编辑的单元格广播此事件。因此，如果有两列，且只有一列是可编辑的（如“score”（分数）），则单击“name”（姓名）列中的某一行不会触发此事件。

# DataGrid.cellFocusOut

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.cellFocusOut = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("cellFocusOut", listenerObject)
```

## 说明

事件：在用户移离具有焦点的单元格时向所有注册的侦听器广播。可以使用事件对象属性来隔离留下的单元格。只有在广播了 `cellEdit` 事件之后，并在下一个单元格广播所有后续的 `cellFocusIn` 事件之前，此事件才会被广播。

第 2 版组件使用调度程序/侦听器事件模型。在 **DataGrid** 组件调度 `cellFocusOut` 事件时，该事件由附加到您创建的侦听器对象的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`DataGrid.cellFocusOut` 事件的事件对象具有三个附加属性：

`columnIndex` 一个数字，它指示目标列的索引。第一个位置是 0。

`itemIndex` 一个数字，它指示目标行的索引。第一个位置是 0。

`type` 字符串 "cellFocusOut"。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在以下示例中，定义了名为 `dgListener` 的处理函数，并将其作为第二个参数传递给了 `my_dg.addEventListener()`。在广播 `cellFocusOut` 事件时，会将 `trace` 语句发送到“输出”面板。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

// 使 DataGrid 可编辑。
my_dg.editable = true;

// 创建侦听器对象。
var dgListener:Object = new Object();
dgListener.cellFocusOut = function(evt_obj:Object) {
 var cell_str:String = "(" + evt_obj.columnIndex + ", " +
 evt_obj.itemIndex + ")";
 trace("The cell at " + cell_str + " has lost focus");
};

// 添加侦听器。
my_dg.addEventListener("cellFocusOut", dgListener);
```

**提醒**

若要使以上代码可以工作，网格必须是可编辑的，并且只对可编辑的单元格广播该事件。因此，如果有两列，且只有一列是可编辑的（如“score”（分数）），则单击“name”（姓名）列中的某一行不会触发此事件。

# DataGrid.cellPress

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.cellPress = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("cellPress", listenerObject)
```

## 说明

事件：当用户在单元格上按下鼠标按钮时向所有注册的侦听器广播。

第 2 版组件使用调度程序 / 侦听器事件模型。在 **DataGrid** 组件广播 `cellPress` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

`DataGrid.cellPress` 事件的事件对象具有三个附加属性：

`columnIndex` 一个数字，它指示被按下的列的索引。第一个位置是 0。

`itemIndex` 一个数字，它指示被按下的行的索引。第一个位置是 0。

`type` 字符串 "cellPress"。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在以下示例中，定义了名为 `dgListener` 的处理函数，并将其作为第二个参数传递给了 `grid.addEventListener()`。`cellPress` 处理函数在 `evt_obj` 参数中捕获事件对象。在广播 `cellPress` 事件时，会将 `trace` 语句发送到“输出”面板。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 设置范例数据。
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce", score:403},
 {name:"Peter", score:25}];

// 创建侦听器对象。
var dgListener:Object = new Object();
dgListener.cellPress = function(evt_obj:Object) {
```

```
var cell_str:String = ("+evt_obj.columnIndex+", "+evt_obj.itemIndex+");
trace("The cell at "+cell_str+" has been clicked");
};
```

```
// 添加侦听器。
my_dg.addEventListener("cellPress", dgListener);
```

## DataGrid.change

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("change", listenerObject)
```

### 说明

事件：在选中项目时向所有注册的侦听器广播。

第 2 版组件使用调度程序 / 侦听器事件模型。在 **DataGrid** 组件调度 `change` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`DataGrid.change` 事件的事件对象具有一个附加属性 `type`，其值为 `"change"`。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

### 示例

在以下示例中，定义了名为 `dgListener` 的处理函数，并将其作为第二个参数传递给了 `grid.addEventListener()`。`change` 处理函数在 `evt_obj` 参数中捕获事件对象。在广播 `change` 事件时，会将 `trace` 语句发送到“输出”面板。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 设置范例数据。
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce", score:403},
 {name:"Peter", score:25}];

// 创建侦听器对象。
```



```
var dgListener:Object = new Object();
dgListener.change = function(evt_obj:Object) {
 trace("The selection has changed to " + evt_obj.target.selectedIndex);
};

// 添加侦听器。
my_dg.addEventListener("change", dgListener);
```

## DataGrid.columnCount

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*myDataGrid.columnCount*

### 说明

属性（只读）：显示的列数。

### 示例

以下示例在“输出”面板中显示总列数。在舞台上具有一个名为 my\_dg 的 DataGrid 组件实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 向网格中添加列并添加数据。
my_dg.addColumn("a");
my_dg.addColumn("b");

my_dg.addItem({a:"one", b:"two"});

// 获取网格中的列数。
var colCount_num:Number = my_dg.columnCount;
trace("Number of columns: "+colCount_num);
```

# DataGrid.columnNames

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.columnNames*

## 说明

属性；显示为列的每个项目内的字段名数组。

## 示例

以下示例在单击列标题时，会在“输出”面板中显示列名称。在舞台上具有一个名为 my\_dg 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(200, 100);
my_dg.columnNames = ["Name", "Description", "Price"];

var dgListener:Object = new Object();
dgListener.headerRelease = function (evt_obj:Object) {
 trace("You clicked on the \" + my_dg.columnNames[evt_obj.columnIndex] + "\"
 column.");
}
my_dg.addEventListener("headerRelease", dgListener);
```

# DataGrid.columnStretch

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.columnStretch = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("columnStretch", listenerObject)
```

## 说明

事件：当用户在水平方向调整列的大小时向所有注册的侦听器广播。

第 2 版组件使用调度程序 / 侦听器事件模型。在 **DataGrid** 组件调度 `columnStretch` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`DataGrid.columnStretch` 事件的事件对象具有两个附加属性：

`columnIndex` 一个数字，它指示目标列的索引。第一个位置是 `0`。

`type` 字符串 `"columnStretch"`。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例在调整列标题大小时，会在“输出”面板中显示列索引号。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(240, 100);
```

```
// 设置范例数据。
```

```
var myDP_array:Array = new Array();
myDP_array.push({id:0, name:"Clark", score:3135});
myDP_array.push({id:1, name:"Bruce", score:403});
myDP_array.push({id:2, name:"Peter", score:25});
```

```
my_dg.dataProvider = myDP_array;
```

```
// 创建侦听器对象。
```

```
var dgListener:Object = new Object();
dgListener.columnStretch = function(evt_obj:Object) {
 trace("column " + evt_obj.columnIndex + " was resized");
};
```

```
// 添加侦听器。
```

```
my_dg.addEventListener("columnStretch", dgListener);
```

# DataGrid.dataProvider

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.dataProvider*

## 说明

属性；在 **DataGrid** 组件中查看的项目的数据模型。

数据网格将方法添加到 **Array** 类的原型，以使每个 **Array** 对象都符合 **DataProvider API**（请参见 **Classes/mx/controls/listclasses** 文件夹中的 **DataProvider.as**）。任何与数据网格位于同一帧或同一屏幕中的数组都会自动具有成为数据网格的数据模型所需的所有方法（**addItem()**、**getItemAt()** 等），并可用于向多个组件广播数据模型更改。

在 **DataGrid** 组件中，您需要指定在 **DataGrid.columnNames** 属性中显示的字段。如果在设置 **DataGrid.dataProvider** 属性之前没有为数据网格定义列集（通过设置 **DataGrid.columnNames** 属性或者调用 **DataGrid.addColumn()**），则一旦到达数据提供程序的第一项，数据网格将为该项中的每个字段生成列。

可以使用任何实现 **DataProvider API** 的对象作为数据网格（包括 **Flash Remoting** 记录集、数据集和数组）的数据提供程序。若要查看示例，请参见第 329 页的“**DataSet.dataProvider**”。

由于无论滚动位置如何，数据提供程序都保持一致，因此可以使用网格的数据提供程序与网格中的数据进行通信。

## 示例

以下示例创建要用作数据提供程序的数组，并将其直接分配给 **dataProvider** 属性：

```
my_dg.dataProvider = [{name:"Chris", price:"Priceless"}, {name:"Nigel",
 price:"cheap"}];
```

以下示例将创建带有 **DataProvider API** 的新 **Array** 对象。它使用 **for** 循环将 20 个项添加到网格中：

```
var myDP:Array = new Array();
for (var i=0; i<20; i++)
 myDP.addItem({id:i, name:"Dave", price:"Priceless"});
my_dg.dataProvider = myDP
```

# DataGrid.editable

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.editable*

## 说明

属性：确定用户是 (true) 否 (false) 能够编辑数据网格。若要使单独的列可编辑并且任何单元格都能获得焦点，此属性必须为 true。默认值为 false。

如果要使单独的列不可编辑，请使用 [DataGridColumn.editable](#) 属性。



如果 DataGrid 直接绑定到 WebServiceConnector 组件或 XMLConnector 组件，则 DataGrid 不可编辑也不可排序。如果要使网格可编辑或可排序，则必须将 DataGrid 组件绑定到 DataSet 组件，并将 DataSet 组件绑定到 WebServiceConnector 组件或 XMLConnector 组件。有关更多信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”。

## 示例

以下示例允许用户编辑网格中除第一列以外的所有列。在舞台上具有一个名为 my\_dg 的 DataGrid 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);
```

```
// 向网格中添加列并添加数据。
my_dg.addColumn("a");
my_dg.addColumn("b");
my_dg.addItem({a:"one", b:1});
my_dg.addItem({a:"two", b:2});
```

```
// 使 DataGrid 可编辑。
my_dg.editable = true;
// 使第一列成为只读列。
my_dg.getColumnAt(0).editable = false;
```

## 另请参见

[DataGridColumn.editable](#)

# DataGrid.editField()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.editField(index, colName, data)*

## 参数

*index* 目标单元格的索引。该数值是从零开始的。

*colName* 一个字符串，它指示包含目标单元格的列（字段）的名称。

*data* 要存储在目标单元格中的值。此参数可以是任何数据类型。

## 返回

单元格中的数据。

## 说明

方法：替换指定位置的单元格数据，并用新值刷新数据网格。所有包含该值的单元格都将触发其 `setValue()` 方法。

## 示例

单击按钮时，以下示例在网格中的第一列的第一行（索引值为 0）置入一个值。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例和一个名为 `my_btn` 的 **Button** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);
```

```
// 设置范例数据。
```

```
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce", score:403},
 {name:"Peter", score:25}];
```

```
// 创建侦听器对象。
```

```
var btnListener:Object = new Object();
btnListener.click = function() {
 // 用新值替换第一个字段。
 my_dg.editField(0, "name", "Arthur");
};
```

```
// 添加按钮侦听器。
```

```
my_btn.addEventListener("click", btnListener);
```

# DataGrid.focusedCell

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.focusedCell*

## 说明

属性：（仅适用于可编辑模式）定义具有焦点的单元格的对象实例。该对象必须具有字段 `columnIndex` 和 `itemIndex`，这两个字段均为整数，指示单元格的列和项的索引。原点为 (0,0)。默认值为 `undefined`。

## 示例

以下示例将具有焦点的单元格设置为第 2 列第 11 行（索引号为“10”，因为第一行的索引号为“0”）。由于在 **DataGrid** 绘制完之后才能访问单元格，因此使用 `UIObject.doLater()` 来延迟对 `focusedCell` 属性的使用：

```
// 创建一个具有 3 列和 50 行的数据提供程序。
var myDP:Array = new Array();
for (var i=0; i<50; i++)
 myDP.addItem({id:i, name:"Dave", price:"Priceless"});

// 将该数据提供程序分配给 DataGrid 实例，并将其设置为可编辑。
my_dg.dataProvider = myDP;
my_dg.editable = true;

// 在数据网格设置了其所有属性后，在当前时间轴中使用 UIObject.doLater() 调用该函数。
my_dg.doLater(this, "select");

function select() {
 my_dg.focusedCell = {columnIndex:1, itemIndex:10};
}
```

# DataGrid.getColumnAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index)
```

## 参数

*index* 要返回的 **DataGridColumn** 对象的索引。该数值是从零开始的。

## 返回

一个 **DataGridColumn** 对象。

## 说明

方法；获取对位于指定索引的 **DataGridColumn** 对象的引用。

## 示例

以下示例获取索引 0 处的 **DataGridColumn** 对象并更改文本。在舞台上具有一个名为 my\_dg 的 **DataGrid** 实例和一个名为 my\_btn 的 **Button** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);

// 设置范例数据。
my_dg.dataProvider = [{name:"Clark", score:3135}, {name:"Bruce", score:403},
 {name:"Peter", score:25}];

// 创建侦听器对象。
var btnListener:Object = new Object();
btnListener.click = function() {
 // 获取位置 0 处的列。
 var a_dgc = my_dg.getColumnAt(0);
 // 更改标题文本。
 a_dgc.headerText = "c";
};

// 添加按钮侦听器。
my_btn.addEventListener("click", btnListener);
```



# DataGrid.getColumnIndex()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid*.getColumnIndex(*columnName*)

## 参数

*columnName* 表示列名的字符串。

## 返回

指定列的索引的数字。

## 说明

方法；返回由 *columnName* 参数指定的列的索引。

## 示例

以下示例显示 “score” 列的索引号。在舞台上具有一个名为 my\_dg 的 DataGrid 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(150, 100);

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

var column_num:Number = my_dg.getColumnIndex("score");
trace("Column that has name of 'score': " + column_num);
```

# DataGrid.headerHeight

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.headerHeight*

## 说明

属性；数据网格的标题栏的高度（以像素为单位）。默认值为 20。

## 示例

以下示例将标题栏的高度设置为 40。在舞台上具有一个名为 my\_dg 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
// 设置网格属性。
my_dg.setSize(240, 100);
my_dg.spaceColumnsEqually();

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

my_dg.headerHeight = 40;
```

# DataGrid.headerRelease

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.headerRelease = function(eventObject){
 // 在此处插入您的代码。
}
myDataGridInstance.addEventListener("headerRelease", listenerObject)
```

## 说明

事件：在松开列标题时向所有注册的侦听器广播。可以将此事件与 `DataGridColumn.sortOnHeaderRelease` 属性一起使用，以便防止自动排序并允许您按所需方式进行排序。

第 2 版组件使用调度程序 / 侦听器事件模型。在 **DataGrid** 组件调度 `headerRelease` 事件时，该事件由附加到您创建的侦听器对象 (`listenerObject`) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`DataGrid.headerRelease` 事件的事件对象具有两个附加属性：

`columnIndex` 一个数字，它指示目标列的索引。

`type` 字符串 "headerRelease"。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

在以下示例中，定义了名为 `myListener` 的处理函数，并将其作为第二个参数传递给 `grid.addEventListener()`。`headerRelease` 处理函数在 `eventObject` 参数中捕获事件对象。在广播 `headerRelease` 事件时，`trace` 语句被发送到“输出”面板中。

```
var myListener = new Object();
myListener.headerRelease = function(event) {
 trace("column " + event.columnIndex + " header was pressed");
};
grid.addEventListener("headerRelease", myListener);
```

在以下示例中，使用列更改排序方向。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
var my_dg:mx.controls.DataGrid;
my_dg.setSize(150, 100);
my_dg.spaceColumnsEqually();
var myListener:Object = new Object();
myListener.headerRelease = function(evt:Object) {
 trace("column "+evt.columnIndex+" header was pressed");
 trace("\t current sort order is: "+evt.target.sortDirection);
 trace("");
};
my_dg.addEventListener("headerRelease", myListener);

my_dg.addColumn("a");
my_dg.addColumn("b");
my_dg.addItem({a:'one', b:1});
my_dg.addItem({a:'two', b:2});
```

通过访问 `sortDirection` 属性，您就能判断出排序顺序是升序还是降序。`sortDirection` 属性是一个字符串，因此它的值为 `ASC` 或 `DESC`。

## DataGrid.hScrollPolicy

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*myDataGrid.hScrollPolicy*

### 说明

属性：指定数据网格是否具有水平滚动条。此属性可以具有以下值：`"on"`、`"off"` 或 `"auto"`。默认值为 `"off"`。

如果将 `hScrollPolicy` 设置为 `"off"`，则会按比例缩放列以适应限定的宽度。



这与 `List` 组件不同，后者不能将 `hScrollPolicy` 设置为 `"auto"`。

## 示例

以下示例将水平滚动策略设置为自动，这意味着如果需要显示所有内容时，将会出现水平滚动条：

```
my_dg.setSize(150, 100);

// 向网格中添加列并添加数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});

my_dg.dataProvider = myDP_array;

my_dg.hScrollPolicy = "on";
```

# DataGrid.removeAllColumns()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.removeAllColumns()
```

## 参数

无。

## 返回

无。

## 说明

方法；从数据网格中删除所有 **DataGridColumn** 对象。调用此方法不会影响数据提供程序。

如果您正在设置的新数据提供程序具有与以前的数据提供程序不同的字段，且您想要清除显示的字段，则调用此方法。

## 示例

单击按钮时，以下示例将删除 **DataGrid** 中的所有 **DataGridColumn** 对象。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例和一个名为 `clear_button` 的 **Button** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);
my_dg.move(10, 40);

this.createClassObject(mx.controls.Button, "clear_button", 20,
 {label:"Clear"});
clear_button.move(10, 10);

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

var buttonListener:Object = new Object();
buttonListener.click = function (evt_obj:Object) {
 my_dg.removeAllColumns();
}
clear_button.addEventListener("click", buttonListener);
```

# DataGrid.removeColumnAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.removeColumnAt(index)
```

## 参数

*index* 要删除的列的索引。

## 返回

指向已删除 **DataGridColumn** 对象的引用。

## 说明

方法：删除位于指定索引处的 **DataGridColumn** 对象。

## 示例

单击按钮时，以下示例会删除第一个 **DataGridColumn** 对象。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例和一个名为 `name_button` 的 **Button** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);
my_dg.move(10, 40);

name_button.setSize(140, name_button.height);
name_button.move(10, 10);

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

// 创建侦听器对象。
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 my_dg.removeColumnAt(my_dg.getColumnIndex("name"));
 evt_obj.target.enabled = false;
};
// 添加按钮侦听器。
name_button.addEventListener("click", buttonListener);
```

# DataGrid.replaceItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.replaceItemAt(index, item)
```

## 参数

*index* 要替换的项的索引。

*item* 充当要用作替换项的项值的对象。

## 返回

先前的值。

## 说明

方法：替换位于指定索引处的项，并刷新网格的显示。

## 示例

以下示例用新条目替换行索引 2 处的项。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例和一个名为 `replace_button` 的 **Button** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);
my_dg.move(10, 40);

replace_button.move(10, 10);

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

// 创建侦听器对象。
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 // 替换前一个值
 var prevValue_obj:Object = my_dg.replaceItemAt(2, {name:"Frank",
 score:949});
 my_dg.selectedIndex = 2;
};
// 添加按钮侦听器。
replace_button.addEventListener("click", buttonListener);
```

# DataGrid.resizableColumns

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid.resizableColumns*

## 说明

属性：一个布尔值，它确定用户是 (true) 否 (false) 能够伸展网格的列。此属性必须为 true 才能让用户调整单独列的大小。默认值为 true。



## 示例

以下示例阻止用户调整列的大小。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

// 不允许调整列大小。
my_dg.resizableColumns = false;
```

# DataGrid.selectable

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid*.selectable

## 说明

属性；一个布尔值，它确定用户是 (`true`) 否 (`false`) 能够选择数据网格。默认值为 `true`。如果为 `false`，则在用户单击网格中的某一项并移走光标后，该项不会保持选中状态。

## 示例

以下示例阻止网格被选中。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.setSize(140, 100);

// 设置范例数据。
var myDP_array:Array = new Array();
myDP_array.push({name:"Clark", score:3135});
myDP_array.push({name:"Bruce", score:403});
myDP_array.push({name:"Peter", score:25});
my_dg.dataProvider = myDP_array;

my_dg.selectable = false;
```

# DataGrid.showHeaders

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.showHeaders
```

## 说明

属性；一个布尔值，它指示数据网格是 (true) 否 (false) 显示列标题。列标题将被加上阴影，以区别于网格中的其它行。如果 [DataGrid.sortableColumns](#) 设置为 true，则用户可以单击列标题对列的内容进行排序。showHeaders 的默认值为 true。

## 示例

以下示例将隐藏列标题：

```
my_dg.setSize(140, 100);

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 不显示标题。
my_dg.showHeaders = false;
```

## 另请参见

[DataGrid.sortableColumns](#)

# DataGrid.sortableColumns

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.sortableColumns
```

## 说明

属性：一个布尔值，它确定在用户单击列标题时是 (true) 否 (false) 能够对数据网格的列进行排序。如果要让单独的列可排序并要广播 `headerRelease` 事件，此属性必须为 `true`。默认值为 `true`。



如果 `DataGrid` 直接绑定到 `WebServiceConnector` 组件或 `XMLConnector` 组件，则 `DataGrid` 不可编辑也不可排序。如果要使网格可编辑或可排序，则必须将 `DataGrid` 组件绑定到 `DataSet` 组件，并将 `DataSet` 组件绑定到 `WebServiceConnector` 组件或 `XMLConnector` 组件。有关更多信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”。

## 示例

以下示例将禁用排序：

```
my_dg.setSize(140, 100);

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 不允许对列进行排序。
my_dg.sortableColumns = false;
```

## 另请参见

[DataGrid.headerRelease](#)

# DataGrid.spaceColumnsEqually()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.spaceColumnsEqually()
```

## 参数

无。

## 返回

无。

## 说明

方法：平均地重新间隔列。

## 示例

单击按钮时，以下示例重新间隔 `my_dg` 的各列。在舞台上具有一个名为 `my_dg` 的 **DataGrid** 实例和一个名为 `resize_button` 的 **Button** 实例的情况下，将以下代码粘贴到主时间轴的第一帧中：

```
my_dg.move(10, 40);
my_dg.setSize(200, 100);

resize_button.move(10, 10);
resize_button.setSize(200, resize_button.height);

my_dg.addColumn("guitar");
my_dg.addColumn("name");

// 设置范例数据。
my_dg.addItem({guitar:"Flying V", name:"maggot"});
my_dg.addItem({guitar:"SG", name:"dreschie"});
my_dg.addItem({guitar:"jagstang", name:"vitapup"});

// 创建侦听器对象。
var buttonListener:Object = new Object();
buttonListener.click = function() {
 my_dg.spaceColumnsEqually();
};
// 添加按钮侦听器。
resize_button.addEventListener("click", buttonListener);
```

# DataGridColumn 类（仅限 Flash Professional）

**ActionScript 类名称** mx.controls.gridclasses.DataGridColumn

可以创建并配置 **DataGridColumn** 对象以用作数据网格的列。**DataGrid** 类的许多方法都专用于管理 **DataGridColumn** 对象。**DataGridColumn** 对象存储在数据网格中一个从零开始的数组内；**0** 是最左边的列。在添加或创建了列之后，您可以调用 `DataGrid.getColumnAt(index)` 来访问它们。

可以通过三种方法在网格中添加或创建列。如果要配置列，请最好在将数据添加到数据网格之前使用第二种或第三种方法，以便不必创建列两次。

- 将一个数据提供程序或具有多个字段的项添加到未配置 `DataGridColumn` 对象的网格中。此方法将自动以 `for..in` 循环的反向顺序为每个字段生成列。例如，对于名为 `my_dg` 的 `DataGrid` 实例，使用以下代码：

```
my_dg.dataProvider = [{guitar:"Flying V", name:"maggot"}, {guitar:"SG",
 name:"dreschie"}, {guitar:"jagstang", name:"vitapup"}];
```

- 使用 `DataGrid.columnNames` 创建所需项字段的字段名，并按顺序为列出的每个字段生成 `DataGridColumn` 对象。此方法使您能够利用最少量的配置快速选择列并对列排序。此方法将删除所有以前的列信息。例如，对于名为 `my_dg` 的 `DataGrid` 实例，使用以下代码：

```
my_dg.columnNames = ["guitar","name"];
```

- 预先构建 `DataGridColumn` 对象并使用 `DataGrid.addColumn()` 将这些对象添加到数据网格中。由于该方法使您在列无论何时到达网格之前都能够使用适当的大小设置和格式设置添加列（这样减少了处理器需求），因此该方法十分有用并且最为灵活。有关更多信息，请参见第 282 页的“[DataGridColumn 类的构造函数](#)”。例如，对于名为 `my_dg` 的 `DataGrid` 实例，使用以下代码：

```
// 创建列对象。
var location_dgc:DataGridColumn = new DataGridColumn("Location");
location_dgc.width = 100;
// 将列添加到数据网格。
my_dg.addColumn(location_dgc);
```

## DataGridColumn 类的属性摘要

下表列出了 `DataGridColumn` 类的属性。

| 属性                                              | 说明                                           |
|-------------------------------------------------|----------------------------------------------|
| <code>DataGridColumn.cellRenderer</code>        | 要用于在此列中显示单元格的元件的链接标识符。                       |
| <code>DataGridColumn.columnName</code>          | 只读；与此列关联的字段名称。                               |
| <code>DataGridColumn.editable</code>            | 一个布尔值，它指示列是 (true) 否 (false) 可编辑。            |
| <code>DataGridColumn.headerRenderer</code>      | 要用于显示此列的标题的类的名称。                             |
| <code>DataGridColumn.headerText</code>          | 此列的标题文本。                                     |
| <code>DataGridColumn.labelFunction</code>       | 确定显示项目的哪个字段的函数。                              |
| <code>DataGridColumn.resizable</code>           | 一个布尔值，它指示列是 (true) 否 (false) 能够调整大小。         |
| <code>DataGridColumn.sortable</code>            | 一个布尔值，它指示列是 (true) 否 (false) 可排序。            |
| <code>DataGridColumn.sortOnHeaderRelease</code> | 一个布尔值，它指示在用户单击列标题时是 (true) 否 (false) 对列进行排序。 |
| <code>DataGridColumn.width</code>               | 列的宽度（以像素为单位）。                                |

# DataGridColumn 类的构造函数

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
new DataGridColumn(name)
```

## 参数

*name* 一个字符串，它指示 **DataGridColumn** 对象的名称。此参数是要显示的每个项目的字段。

## 返回

无。

## 说明

构造函数；创建一个 **DataGridColumn** 对象。使用此构造函数来创建要添加到 **DataGrid** 组件的列。在创建 **DataGridColumn** 对象后，您可以通过调用 `DataGrid.addColumn()` 将它们添加到数据网格中。

## 示例

以下示例将创建名为 `Location` 的 **DataGridColumn** 对象：

```
import mx.controls.gridclasses.DataGridColumn;
var column = new DataGridColumn("Location");
```

# DataGridColumn.cellRenderer

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index).cellRenderer
```

## 说明

属性：要用于在此列中显示单元格的元件的链接标识符。用于此属性的任何类都必须实现 `CellRenderer` API（请参见第 101 页的“[CellRenderer API](#)”）。默认值为 `undefined`。

## 示例

下面的示例使用链接标识符来设置一个新的单元格渲染器：

```
myGrid.getColumnAt(3).cellRenderer = "MyCellRenderer";
```

# DataGridColumn.columnName

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index).columnName
```

## 说明

属性（只读）：与此列关联的字段的名称。默认值是在 `DataGridColumn` 构造函数中调用的名称。

## 示例

以下示例将列名称显示为索引位置 1：

```
import mx.controls.gridclasses.DataGridColumn;
// 设置网格属性。
my_dg.setSize(150, 100);

// 向网格中添加列。
var name_dgc:DataGridColumn = my_dg.addColumn(new DataGridColumn("name"));
name_dgc.headerText = "Name: ";
var score_dgc:DataGridColumn = my_dg.addColumn(new DataGridColumn("score"));
score_dgc.headerText = "Score: ";

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 获取列名称。
var name_str:String = my_dg.getColumnAt(1).columnName;
trace(name_str);
```

另请参见

[DataGridColumn](#) 类的构造函数

# DataGridColumn.editable

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`myDataGrid.getColumnAt(index).editable`

## 说明

属性：确定用户是 (true) 否 (false) 能编辑列。若要使单独的列可编辑，[DataGrid.editable](#) 属性必须为 true（即使 [DataGridColumn.editable](#) 设置为 true）。默认值为 true。

注意

如果 DataGrid 直接绑定到 WebServiceConnector 组件或 XMLConnector 组件，则 DataGrid 不可编辑也不可排序。如果要使网格可编辑或可排序，则必须将 DataGrid 组件绑定到 DataSet 组件，并将 DataSet 组件绑定到 WebServiceConnector 组件或 XMLConnector 组件。有关更多信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”。

## 示例

以下示例将使网格中第一列中的项目不可编辑：

```
// 设置网格属性。
my_dg.setSize(150, 100);
my_dg.editable = true;

// 向网格中添加列。
my_dg.addColumn("name");
my_dg.addColumn("score");

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 不允许编辑第一列。
my_dg.getColumnAt(0).editable = false;
```

另请参见

[DataGrid.editable](#)



# DataGridColumn.headerRenderer

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index).headerRenderer
```

## 说明

属性：一个字符串，它指明要用于显示此列的标题的类名称。用于此属性的任何类都必须实现 `CellRenderer` API（请参见第 101 页的“[CellRenderer API](#)”）。默认值为 `undefined`。

## 示例

以下示例使用链接标识符来设置一个新的标题渲染器：

```
myGrid.getColumnAt(3).headerRenderer = "MyHeaderRenderer";
```

# DataGridColumn.headerText

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index).headerText
```

## 说明

属性：列标题中的文本。默认值为列名称。

此属性允许您显示除字段名称以外的内容作为标题。

## 示例

以下示例将列标题文本设置为 “Price (USD)”：

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;

var price_dgc:DataGridColumn = new DataGridColumn("price");
price_dgc.headerText = "Price (USD)";
price_dgc.width = 80;
my_dg.addColumn(price_dgc);

my_dg.addItem({price:"$14.99"});
```

# DataGridColumn.labelFunction

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDataGrid*.getColumnAt(*index*).labelFunction

## 说明

属性：指定用于确定显示每个项目的哪个字段（或字段组合）的函数。此函数会接收一个参数 *item*（表示要呈现的项），并且必须返回一个表示要显示的文本的字符串。此属性可用于创建在项目中没有相应字段的虚拟列。



指定的函数在未定义的范围运行。

## 示例

以下示例计算 “Subtotal”（小计）列的值：

```
import mx.controls.gridclasses.DataGridColumn;

var my_dg:mx.controls.DataGrid;
my_dg.setSize(300, 200);

// 设置列。
var guitar_dgc:DataGridColumn = new DataGridColumn("guitar");
var value_dgc:DataGridColumn = new DataGridColumn("value");
var tax_dgc:DataGridColumn = new DataGridColumn("tax");
var st_dgc:DataGridColumn = new DataGridColumn("Subtotal");
```

```
// 为“Subtotal”（小计）列定义 labelFunction。
st_dgc.labelFunction = function(item:Object):String {
 if ((item.value != undefined) && (item.tax != undefined)) {
 return "$"+(item.value+item.tax);
 }
};

// 向网格中添加列。
my_dg.addColumn(guitar_dgc);
my_dg.addColumn(value_dgc);
my_dg.addColumn(tax_dgc);
my_dg.addColumn(st_dgc);

// 设置数据模型。
my_dg.addItem({guitar:"Flying V", value:10, tax:1});
my_dg.addItem({guitar:"SG", value:20, tax:2});
my_dg.addItem({guitar:"jagstang", value:30, tax:3});
```

## DataGridColumn.resizable

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
myDataGrid.getColumnAt(index).resizable
```

### 说明

属性：一个布尔值，它指示用户是 (true) 否 (false) 能调整列的大小。若要使此属性生效，[DataGrid.resizableColumns](#) 属性必须设置为 true。默认值为 true。

### 示例

以下示例防止调整位于索引 0 处的列的大小：

```
// 设置网格属性。
my_dg.setSize(150, 100);
my_dg.addColumn("name");
my_dg.addColumn("score");

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 不允许调整第一列的大小。
my_dg.getColumnAt(0).resizable = false;
```

# DataGridColumn.sortable

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`myDataGrid.getColumnAt(index).sortable`

## 说明

属性：一个布尔值，它指示用户是 (true) 否 (false) 能对列进行排序。若要使此属性生效，`DataGrid.sortableColumns` 属性必须设置为 true。默认值为 true。

注意

如果 DataGrid 直接绑定到 WebServiceConnector 组件或 XMLConnector 组件，则 DataGrid 不可编辑也不可排序。如果要使网格可编辑或可排序，则必须将 DataGrid 组件绑定到 DataSet 组件，并将 DataSet 组件绑定到 WebServiceConnector 组件或 XMLConnector 组件。有关更多信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”。

## 示例

以下示例防止对位于索引 1 处的列进行排序：

```
// 设置网格属性。
my_dg.setSize(150, 100);

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 不允许对第二列进行排序。
my_dg.getColumnAt(1).sortable = false;
```

# DataGridColumn.sortOnHeaderRelease

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index).sortOnHeaderRelease
```

## 说明

属性：一个布尔值，它指示在用户单击列标题时是 (true) 否 (false) 自动对列进行排序。只有在 `DataGridColumn.sortable` 设置为 true 时，此属性才能设置为 true。如果 `DataGridColumn.sortOnHeaderRelease` 设置为 false，您可以捕获 `headerRelease` 事件并执行自己的排序。默认值为 true。

注意

如果 DataGrid 直接绑定到 `WebServiceConnector` 组件或 `XMLConnector` 组件，则 DataGrid 不可编辑也不可排序。如果要使网格可编辑或可排序，则必须将 DataGrid 组件绑定到 `DataSet` 组件，并将 `DataSet` 组件绑定到 `WebServiceConnector` 组件或 `XMLConnector` 组件。有关更多信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”。

## 示例

以下示例禁止对第二列进行排序：

```
// 设置网格属性。
my_dg.setSize(150, 100);

// 设置范例数据。
my_dg.addItem({name:"Clark", score:3135});
my_dg.addItem({name:"Bruce", score:403});
my_dg.addItem({name:"Peter", score:25});

// 不允许通过单击列标题对第二列进行排序。
my_dg.getColumnAt(1).sortOnHeaderRelease = false;
```

# DataGridColumn.width

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDataGrid.getColumnAt(index).width
```

## 说明

属性；指明列宽度的数值（以像素为单位）。默认值为 50。

## 示例

以下示例使第一列的宽度为 50 像素：

```
// 创建新的 DataProvider 组件。
var myDP_array:Array = new Array({name:"Chris", price:"Priceless"},
 {name:"Nigel", price:"Cheap"});
// 将 DataProvider 分配给 DataGrid。
my_dg.dataProvider = myDP_array;

// 更改 DataGrid 尺寸。
my_dg.setSize(140, 100);
my_dg.rowHeight = 30;
my_dg.getColumnAt(0).width = 50;
```

# DataHolder 组件（仅限 Flash Professional）

**DataHolder** 组件是数据的储备库，并且可用于在数据更改时生成事件。它的主要用途是容纳数据，并充当使用数据绑定的其它组件之间的连接器。

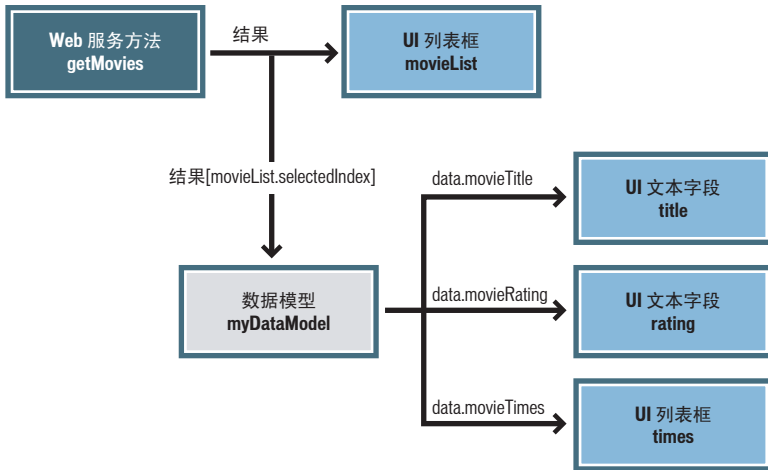
**DataHolder** 组件最初只有一个名为 `data` 的可绑定属性。您可以使用“组件”检查器中的“架构”选项卡来添加更多的属性。有关使用“架构”选项卡的更多信息，请参见《使用 Flash》中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

您可以将任何类型的数据分配给 **DataHolder** 属性，方法是在数据和另一属性之间创建绑定，或者使用自己的 **ActionScript** 代码。当该数据的值更改时，**DataHolder** 组件将发出一个名称与属性相同的事件，并执行与该属性关联的任何绑定。

大多数情况下，您不使用此组件来构建应用程序。只有在无法将外部数据直接绑定到另一组件并且您不想使用 **DataSet** 组件时，才需要该组件。在无法直接将组件（如连接器、用户界面组件或 **DataSet** 组件）绑定在一起时，**DataHolder** 组件十分有用。下面是您可以使用 **DataHolder** 组件的一些情况：

- 如果数据值是由 **ActionScript** 生成的，则可能需要将其绑定到某些其它组件。在这种情况下，您可以使用包含按需要绑定的属性的 **DataHolder** 组件。在将新值分配给这些属性时（例如，通过 **ActionScript**），这些值将被分发到数据绑定对象。

- 您可能从索引复杂的数据绑定中产生的数据值，如下图中所示。



在这种情况下，为方便起见，请将数据值绑定到某个 **DataHolder** 组件（在上图中称为 **DataModel**），然后使用该组件绑定到用户界面。



**DataHolder** 组件不应该对数据实施与 **DataSet** 组件相同的控制。它不管理或跟踪数据，也不具有更新数据的能力。它是一个在数据改变时用于存储数据和生成事件的存储库。

## 创建具有 **DataHolder** 组件的应用程序（仅限 Flash Professional）

在此范例中，您需要将数组属性添加到 **DataHolder** 组件的架构（一个数组）中，该架构的值由您编写的 **ActionScript** 代码确定。然后，您需要将该数组属性绑定到 **DataGrid** 组件的 **dataProvider** 属性，方法是使用“组件”检查器中的“绑定”选项卡。

**要在简单应用程序中使用 **DataHolder** 组件：**

1. 在 Flash 中，创建一个新文件。
2. 打开“组件”面板，将一个 **DataHolder** 组件拖动到舞台上，然后将其命名为 **dataHolder**。
3. 将一个 **DataGrid** 组件拖到舞台上，然后将其命名为 **namesGrid**。
4. 选择 **DataHolder** 组件并打开“组件”检查器。
5. 在“组件”检查器中，单击“架构”选项卡。
6. 单击位于“架构”选项卡顶部窗格中的“添加组件属性” (+) 按钮。



- 7. 在“架构”选项卡底部窗格的“field name”字段中键入 **namesArray**，然后从“data type”弹出菜单中选择“Array”。
- 8. 在“组件”检查器中单击“绑定”选项卡，然后在 **DataHolder** 组件的 `namesArray` 属性和 **DataGrid** 组件的 `dataProvider` 属性之间添加绑定。  
有关使用“绑定”选项卡创建绑定的更多信息，请参见《使用 Flash》中的“在“绑定”选项卡中处理绑定（仅限于 Flash Professional）”。
- 9. 在“时间轴”中选择“图层 1”上的第一帧，然后打开“动作”面板。
- 10. 在“动作”面板中输入下面的代码：

```
dataHolder.namesArray = [{name:"Tim"},{name:"Paul"},{name:"Jason"}];
```

这段代码使用多个对象填充 `namesArray` 数组。当此变量赋值执行时，将会执行您之前在 **DataHolder** 组件和 **DataGrid** 组件之间建立的绑定。
- 11. 通过选择“控制”>“测试影片”对文件进行测试。

# DataHolder 类

继承 MovieClip > DataHolder

ActionScript 类名称 mx.data.components.DataHolder

**DataHolder** 组件是数据的储备库，并且可用于在数据更改时生成事件。它的主要用途是容纳数据，并充当使用数据绑定的其它组件之间的连接器。

**DataHolder** 组件最初只有一个名为 `data` 的可绑定属性。您可以使用“组件”检查器中的“架构”选项卡来添加更多的属性。

## DataHolder 类的属性摘要

下表列出了 **DataHolder** 类的属性。

| 属性                              | 说明                            |
|---------------------------------|-------------------------------|
| <a href="#">DataHolder.data</a> | <b>DataHolder</b> 组件的默认可绑定属性。 |

# DataHolder.data

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dataHolder.data*

## 说明

属性：**DataHolder** 对象架构中的默认项目。此属性不是 **DataHolder** 组件的“永久”成员，而是组件每个实例的默认可绑定属性。通过使用“组件”检查器中的“架构”选项卡，您可以添加自己的可绑定属性，也可以删除默认的 `data` 属性。

有关使用“架构”选项卡的更多信息，请参见《使用 Flash》中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

## 示例

有关使用此组件的分步示例，请参见第 292 页的“创建具有 **DataHolder** 组件的应用程序（仅限 Flash Professional）”。

下面的代码演示关于如何使用变量数据填充 **DataHolder** 组件的一个简单示例。若要测试该应用程序，请在文本输入字段中输入一个值，然后单击 `addDate_btn` 实例，该实例就会将值添加到 **DataHolder** 组件。单击 `dumpDataHolder_btn` 实例以跟踪 **DataHolder** 组件的内容。

```
// 将两个按钮组件（addDate_btn 和 dumpDataHolder_btn）、一个 TextInput
（myDate_txt）和一个 DataHolder（myDataHolder）拖动到舞台上。将下面的
ActionScript 添加到第一帧：
```

```
var dhListener:Object = new Object();
dhListener.click = function() {
 trace("dumping DataHolder");
 trace(" " + myDataHolder.myDate);
 trace("");
};
var dateListener:Object = new Object();
dateListener.click = function() {
 myDataHolder.myDate = myDate_txt.text;
 trace("added value");
};
this.dumpDataHolder_btn.addEventListener("click", dhListener);
this.addDate_btn.addEventListener("click", dateListener);
```

**DataProvider API** 是一组方法和属性，数据源需要这些方法和属性才能使基于列表的类与其通信。数组、记录集和数据集均实现此 API。您可以通过实现本部分中描述的所有方法和属性来创建符合 **DataProvider** 的类。然后，基于列表的组件将能够使用该类作为数据提供程序。

## DataProvider 类

**ActionScript 类名称** mx.controls.listclasses.DataProvider

**DataProvider** 类的方法使您能够在显示数据的任何组件（也称为“视图”）中查询和修改数据。**DataProvider API** 还会在数据更改时广播 `change` 事件。多个视图可以使用同一数据提供程序，并且都可以接收 `change` 事件。

数据提供程序是项目的线性集合（类似于数组）。每个项目都是一个由多个数据字段组成的对象。通过使用 `DataProvider.getItemAt()`，您可以按索引访问这些项（就如同处理数组）。

数据提供程序最常用于数组。当 `Array` 对象与支持数据的组件处于同一帧或同一屏幕中时，支持数据的组件会将 **DataProvider API** 的所有方法应用到 `Array.prototype`。这样，您就可以将任何现有数组用作具有 `dataProvider` 属性的视图的数据。

由于 **DataProvider API** 的存在，提供数据视图的第 2 版 **Macromedia Component Architecture** 组件（`DataGrid`、`List`、`Tree` 等）还可以显示 **Flash Remoting RecordSet** 对象以及 `DataSet` 组件中的数据。**DataProvider API** 是支持数据的组件用于与其数据提供程序通信的语言。

在 **Macromedia Flash** 文档中，“**DataProvider**”是类的名称，`dataProvider` 是充当数据视图的每个组件的属性，而“数据提供程序”则是表示数据源的一般术语。

## DataProvider API 的方法摘要

下表列出了 DataProvider API 的方法。

| 方法                                         | 说明                                       |
|--------------------------------------------|------------------------------------------|
| <code>DataProvider.addItem()</code>        | 将项目添加到数据提供程序的结尾。                         |
| <code>DataProvider.addItemAt()</code>      | 将项目添加到数据提供程序的指定位置。                       |
| <code>DataProvider.editField()</code>      | 更改数据提供程序的某个字段。                           |
| <code>DataProvider.getEditingData()</code> | 从数据提供程序中获取用于编辑的数据。                       |
| <code>DataProvider.getItemAt()</code>      | 获取对位于指定位置的项目的引用。                         |
| <code>DataProvider.getItemID()</code>      | 返回项目的唯一 ID。                              |
| <code>DataProvider.removeAll()</code>      | 从数据提供程序中删除所有项目。                          |
| <code>DataProvider.removeItemAt()</code>   | 从数据提供程序的指定位置删除项目。                        |
| <code>DataProvider.replaceItemAt()</code>  | 用其它项目替换位于指定位置的项目。                        |
| <code>DataProvider.sortItems()</code>      | 根据比较函数或排序选项，对数据提供程序中的项目排序。               |
| <code>DataProvider.sortItemsBy()</code>    | 按字母或数值顺序、按指定的顺序、使用指定的字段名对数据提供程序中的项目进行排序。 |

## DataProvider API 的属性摘要

下表列出了 DataProvider API 的属性。

| 属性                               | 说明           |
|----------------------------------|--------------|
| <code>DataProvider.length</code> | 数据提供程序中的项目数。 |

## DataProvider API 的事件摘要

下表列出了 DataProvider API 的事件。

| 事件                                     | 说明            |
|----------------------------------------|---------------|
| <code>DataProvider.modelChanged</code> | 在数据提供程序更改时广播。 |

# DataProvider.addItem()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.addItem(item)
```

## 参数

*item* 包含数据的对象。它包含数据提供程序中的项目。

## 返回

无。

## 说明

方法；在数据提供程序的结尾添加新项目。此方法触发事件名称为 `addItem` 的 `modelChanged` 事件。

## 示例

下面的示例将某一项添加到数据提供程序 `myDP` 的结尾：

```
myDP.addItem({label : "this is an Item"});
```

# DataProvider.addItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.addItemAt(index, item)
```

## 参数

*index* 一个大于或等于 0 的数字。此数字指示项的插入位置；它是新项的索引。

*item* 包含项数据的对象。

## 返回

无。

## 说明

方法：将新项目添加到数据提供程序的指定索引处。大于数据提供程序长度的索引将被忽略。

此方法触发事件名称为 `addItem` 的 `modelChanged` 事件。

## 示例

下面的示例将某一项添加到数据提供程序 `myDP` 的第 4 个位置：

```
myDP.addItemAt(3, {label : "this is the fourth Item"});
```

# DataProvider.editField()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.editField(index, fieldName, newData)
```

## 参数

*index* 一个大于或等于 0 的数字；项的索引。

*fieldName* 一个字符串，它指示项中要修改的字段名称。

*newData* 要放入数据提供程序的新数据。

## 返回

无。

## 说明

方法：更改数据提供程序的某个字段。

此方法触发事件名称为 `updateField` 的 `modelChanged` 事件。

## 示例

以下代码修改第三项的 `label` 字段：

```
myDP.editField(2, "label", "mynewData");
```

# DataProvider.getEditingData()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.getEditingData(index, fieldName)
```

## 参数

*index* 一个大于或等于 0 且小于 `DataProvider.length` 的数字。此数字是要检索的项的索引。

*fieldName* 一个字符串，它指示所编辑字段的名称。

## 返回

要使用的可编辑格式数据。

## 说明

方法：从数据提供程序中检索要编辑的数据。这样，数据模型将能够提供不同格式的数据进行编辑和显示。

## 示例

以下代码将获取 `price` 字段的可编辑字符串：

```
trace(myDP.getEditingData(4, "price");
```

# DataProvider.getItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.getItemAt(index)
```

### 参数

*index* 一个大于或等于 0 且小于 `DataProvider.length` 的数字。此数字是要检索的项的索引。

### 返回

对检索出的项目的引用；如果索引超出范围，则为未定义。

### 说明

方法；检索对位于指定位置的项目的引用。

### 示例

以下代码将显示第 5 个项目的标签：

```
trace(myDP.getItemAt(4).label);
```

## DataProvider.getItemID()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
myDP.getItemID(index)
```

### 参数

*index* 一个大于或等于 0 的数字。

### 返回

一个数字，充当项目的唯一 ID。

### 说明

方法；返回项目的唯一 ID。此方法主要用于跟踪选择。此 ID 用于支持数据的组件中，对哪些项目的列表处于选定状态进行记录。

### 示例

此范例获取第 4 个项目的 ID：

```
var ID = myDP.getItemID(3);
```



# DataProvider.length

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myDP.length*

## 说明

属性（只读）：数据提供程序中的项数。

## 示例

此示例将 myArray 数据提供程序中的项数发送到“输出”面板：

```
trace(myArray.length);
```

# DataProvider.modelChanged

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.modelChanged = function(eventObject){
 // 此处插入您的代码。
}
myMenu.addEventListener("modelChanged", listenerObject)
```

## 说明

事件：在数据提供程序被修改时向所有其视图侦听器广播。侦听器通常是通过分配其 dataProvider 属性添加到模型的。

第 2 版组件使用调度程序 / 侦听器事件模型。当数据提供程序在某些方面发生更改时，它将会广播 modelChanged 事件，并且支持数据的组件将捕获该事件来更新显示，以便反映数据的更改。

Menu.modelChanged 事件的事件对象具有五个附加属性：

- eventName 该属性用于划分 modelChanged 事件的子类别。支持数据的组件使用此信息来避免完全刷新使用数据提供程序的组件实例（视图）。eventName 属性支持下面的值：
  - updateAll 整个视图都需要刷新（除滚动位置外）。
  - addItem 已添加一系列项。
  - removeItems 已删除一系列项。
  - updateItems 一系列项需要刷新。
  - sort 数据已排序。
  - updateField 项内的某个字段已更改，并且需要刷新。
  - updateColumn 数据提供程序中整个字段的定义需要刷新。
  - filterModel 已过滤模型，并且视图需要刷新（重置滚动位置）。
  - schemaLoaded 已声明数据提供程序的字段定义。
- firstItem 第一个受影响项的索引。
- lastItem 最后一个受影响项的索引。如果只有一项受影响，则该值与 firstItem 相等。
- removedIDs 已删除的项标识符组成的数组。
- fieldName 一个字符串，指示受影响字段的名称。

有关更多信息，请参见第 461 页的“EventDispatcher 类”。

## 示例

在下面的示例中，将定义一个名为 listener 的处理函数，并将其作为第二个参数传递给 addEventListener()。modelChanged 处理函数将在 evt 参数中捕获该事件对象。广播 modelChanged 事件时，trace 语句将被发送到“输出”面板。

```
listener = new Object();
listener.modelChanged = function(evt){
 trace(evt.eventName);
}
myList.addEventListener("modelChanged", listener);
```

# DataProvider.removeAll()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.removeAll()
```

### 参数

无。

### 返回

无。

### 说明

方法：删除数据提供程序中的所有项目。此方法触发事件名称为 `removeItems` 的 `modelChanged` 事件。

### 示例

此范例将删除数据提供程序中的所有项目：

```
myDP.removeAll();
```

## DataProvider.removeItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
myDP.removeItemAt(index)
```

### 参数

*index* 一个大于或等于 0 的数字。此数字是要删除的项的索引。

### 返回

无。

### 说明

方法：删除位于指定索引处的项目。已删除索引后面的索引将依次减 1。

此方法触发事件名称为 `removeItems` 的 `modelChanged` 事件。

### 示例

此范例将删除位于第 4 个位置的项目：

```
myDP.removeItemAt(3);
```

# DataProvider.replaceItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.replaceItemAt(index, item)
```

## 参数

*index* 一个大于或等于 0 的数字。此数字是要更改的项的索引。

*item* 充当新项的对象。

## 返回

无。

## 说明

方法：替换位于指定索引处的项目的内容。此方法触发事件名称为 `updateItems` 的 `modelChanged` 事件。

## 示例

此示例将使用具有 “new label” 标签的项目替换位于索引 3 处的项目：

```
myDP.replaceItemAt(3, {label : "new label"});
```

# DataProvider.sortItems()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.sortItems([compareFunc], [optionsFlag])
```

## 参数

*compareFunc* 对特定函数的引用，该函数比较两项以确定它们的排序顺序。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `sort (Array.sort method)`。此参数是可选的。

*optionsFlag* 允许您不必复制整个数组或者反复对其重新排序，即可对单个数组执行不同类型的多次排序。此参数是可选的。

以下是 *optionsFlag* 的可能值：

- `Array.DESENDING`，按从高到低的顺序排序。
- `Array.CASEINSENSITIVE`，按不区分大小写的方式排序。
- `Array.NUMERIC`，如果进行比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（如果指定该标志，字符串比较可以不区分大小写）。
- `Array.UNIQUESORT`，如果数组中有两个对象相同或者具有相同的排序字段，则返回错误代码 (0)，而不是排序后的数组。
- `Array.RETURNINDEXEDARRAY`，返回作为排序结果的整数索引数组。例如，下面的数组将返回代码的第二行，并且该数组保持不变：  

```
["a", "d", "c", "b"]
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 3 和 1：

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

## 返回

无。

## 说明

方法；根据指定的比较函数或一个或多个指定的排序选项，对数据提供程序中的项目排序。

此方法触发事件名称为 `sort` 的 `modelChanged` 事件。

## 示例

此示例根据大写标签进行排序。项 `a` 和 `b` 被传递到该函数，并包含 `label` 和 `data` 字段：

```
myList.sortItems(upperCaseFunc);
function upperCaseFunc(a,b){
 return a.label.toUpperCase() > b.label.toUpperCase();
}
```

# DataProvider.sortItemsBy()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myDP.sortItemsBy(fieldName, optionsFlag)
```

```
myDP.sortItemsBy(fieldName, order)
```

## 参数

*fieldName* 一个字符串，它指定要用于排序的字段名称。通常情况下，此值为 "label" 或 "data"。

*order* 一个字符串，它指定是以升序 ("ASC") 还是以降序 ("DESC") 对项进行排序。

*optionsFlag* 允许您不必复制整个数组或者反复对其重新排序，即可对单个数组执行不同类型的多次排序。此参数是可选的。

以下是 *optionsFlag* 的可能值：

- `Array.DECENDING` - 按从高到低的顺序排序。
- `Array.CASEINSENSITIVE` - 按不区分大小写的方式排序。
- `Array.NUMERIC` - 如果所比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（如果指定该标志，字符串比较可以不区分大小写）。
- `Array.UNIQUESORT` - 如果数组中有两个对象相同或者具有相同的排序字段，则此方法返回错误代码 (0)，而不是排序后的数组。
- `Array.RETURNINDEXEDARRAY` - 返回作为排序结果的整数索引数组。例如，下面的数组将返回代码的第二行，并且该数组保持不变：

```
["a", "d", "c", "b"]
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 3 和 1：

```
array.sort (Array.NUMERIC | Array.DECENDING)
```

## 返回

无。

## 说明

方法：按指定的顺序、使用指定的字段名对数据提供程序中的项目进行排序。如果 *fieldName* 项既包括文本字符串也包括整数，则首先列出整数项。*fieldName* 参数通常为 "label" 或 "data"，但高级编程人员可能会指定任何原始值。

此方法触发事件名称为 sort 的 modelChanged 事件。

本方法是对组件中数据排序的最快方法。它还保持组件的选择状态。sortItemsBy() 方法之所以运行速度很快，是因为它在排序时不运行任何 **ActionScript**。sortItems() 方法需要运行 **ActionScript** 的比较函数，因此速度较慢。

## 示例

下列代码使用列表项的标签按升序对列表中的项目进行排序：

```
myDP.sortItemsBy("label", "ASC");
```





# DataSet 组件（仅限 Flash Professional）

**DataSet** 组件使您能够将数据处理为可进行索引、排序、搜索、过滤和修改的对象的集合。

**DataSet** 组件功能包括 **DataSetIterator**（一组用于遍历和处理数据集合的方法）和 **DeltaPacket**（一组用于处理数据集合更新的接口和类）。大多数情况下，您不会直接使用这些类和接口；您将通过 **DataSet** 类提供的方法间接使用它们。

**DataSet** 组件管理的项目也称为传送对象。传送对象使用公共属性或存取器方法（用于读取和写入数据）显示驻留在服务器上的业务数据。**DataSet** 组件使开发人员能够处理复杂的客户端对象（反映其服务器端对应对象）或最简单的匿名对象集合（具有表示数据记录内的字段的公共属性）。有关传输对象的详细信息，请参见位于 <http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html> 的 Core J2EE Patterns Transfer Object（核心 J2EE 模式传输对象）。



DataSet 组件需要 Flash Player 7 或更高版本。

## 使用 DataSet 组件

通常情况下，您将 **DataSet** 组件与其它组件结合使用以处理和更新数据源：**Connector** 组件（用于连接到外部数据源）、用户界面组件（用于显示数据源中的数据）以及解析程序组件（用于将对数据集进行的更新转换为适当的格式，以便发送到外部数据源）。然后，您可以使用数据绑定将这些不同组件的属性绑定在一起。

**DataSet** 组件使用数据绑定类中的功能。如果您只是计划在 **ActionScript** 中使用 **DataSet** 组件，而不使用“组件”检查器中的“绑定”和“架构”选项卡来设置属性，则需要将数据绑定类导入到 **FLA** 文件，并在代码中设置必需的属性。请参见第 191 页的“使数据绑定类在运行时可用（仅限 **Flash Professional**）”。

有关如何使用 **DataSet** 组件管理 **Flash** 中的数据的一般信息，请参见《使用 **Flash**》中的“数据管理（仅限 **Flash Professional**）”。

## DataSet 参数

您可以为 **DataSet** 组件设置以下参数：

**itemClassName** 是一个字符串，它指示每次在 **DataSet** 组件内创建新项目时实例化的传输对象类的名称。

**DataSet** 组件使用传输对象来表示从外部数据源中检索的数据。如果将此参数留空，数据集将为您创建一个匿名传输对象。如果为此参数指定值，数据集将在每次添加新数据时对传输对象进行实例化。



必须在代码内的某个位置对这个类进行完全限定引用，以确保将它编译到应用程序中（如 `private var myItem:my.package.myItem;`）。

**logChanges** 是一个布尔值，默认为 `true`。如果将此参数设置为 `true`，则数据集会记录对其数据的所有更改，以及对相关传输对象调用的所有方法。

**readOnly** 是一个布尔值，默认为 `false`。如果将此参数设置为 `true`，则无法修改数据集。

您可以编写 **ActionScript** 代码来使用 **DataSet** 组件的属性、方法和事件，以控制这些选项和其它选项。有关更多信息，请参见第 313 页的“**DataSet 类 (仅限 Flash Professional)**”。

## DataSet 组件的通用工作流程

**DataSet** 组件的典型工作流程如下所示。

### 要使用 **DataSet** 组件：

1. 将 **DataSet** 组件的实例添加到应用程序，并为其指定一个实例名称。
2. 为 **DataSet** 组件选择“架构”选项卡，并创建组件属性以表示数据集的永久性字段。
3. 利用外部数据源中的数据加载 **DataSet** 组件。（有关更多信息，请参见《使用 Flash》中的“关于将数据加载到 **DataSet** 组件中”。）
4. 使用“组件”检查器的“绑定”选项卡将数据集字段绑定到应用程序内的用户界面组件。

在 **DataSet** 组件内选定或修改记录（传输对象）时，**UI** 控件将收到通知，并随之更新。另外，对于从 **UI** 控件内部进行的更改，**DataSet** 组件也会收到通知；这些更改会由数据集跟踪，并可以通过增量数据包的方式提取。

5. 在应用程序内调用 **DataSet** 组件的方法来管理数据。



除了上述步骤之外，您还可以将 **DataSet** 组件绑定到连接器和解析程序组件，以便提供用于对外部数据源中的数据进行访问、管理和更新的完整解决方法。

## 创建具有 DataSet 组件的应用程序

通常情况下，您可以将 DataSet 组件与其它用户界面组件一起使用，并通常与 Connector 组件（如 XMLConnector 或 WebServiceConnector）一起使用。数据集中的项目是借助于 Connector 组件或原始 ActionScript 数据填充的，并随后绑定到用户界面控件（如 List 或 DataGrid 组件）。

DataSet 组件使用数据绑定类中的功能。如果您只是计划在 ActionScript 中使用 DataSet 组件，而不使用“组件”检查器中的“绑定”和“架构”选项卡来设置属性，则需要将数据绑定类导入到 FLA 文件，并在代码中设置必需的属性。请参见第 191 页的“使数据绑定类在运行时可用（仅限 Flash Professional）”。

### 使用 DataSet 组件创建应用程序：

1. 在 Flash Professional 8 中，选择“文件”>“新建”。在“类型”列中选择“Flash 文档”，然后单击“确定”。
2. 如果“组件”面板尚未打开，请将其打开。
3. 将 DataSet 组件从“组件”面板拖到舞台上。在“属性”检查器中，为其指定实例名称 **user\_ds**。
4. 将 DataGrid 组件拖到舞台上，然后为其指定实例名称 **user\_dg**。
5. 将 DataGrid 组件的大小调整为约 300 像素宽，100 像素高。
6. 将一个 Button 组件拖到舞台上，然后为其指定实例名称 **next\_button**。
7. 在“时间轴”中选择“图层 1”上的第一帧，然后打开“动作”面板。
8. 将以下代码添加到“动作”面板中：

```
var recData_array:Array = [{id:0, firstName:"Mick", lastName:"Jones"},
 {id:1, firstName:"Joe", lastName:"Strummer"},
 {id:2, firstName:"Paul", lastName:"Simonon"}];
user_ds.items = recData_array;
```

这将使用一个对象数组填充 DataSet 对象的 items 属性，其中每个对象具有三个属性：id、firstName 和 lastName。

9. 将这三个属性和它们必需的数据类型添加到 DataSet 架构：
  - a. 在舞台上选择 DataSet 组件，打开“组件”检查器，并单击“架构”选项卡。
  - b. 单击“添加组件属性”，然后添加三个新属性，字段名分别为 id、firstName 和 lastName，数据类型分别为 Number、String 和 String。

或者，如果您更倾向于用代码来添加属性及其必需的数据类型，则可以将下面一行代码添加到“动作”面板，而无需执行上面的步骤 a 和 b：

```
// 添加必需的架构类型。
var i:mx.data.types.Str;
var j:mx.data.types.Num;
```

10. 要将 **DataSet** 组件的内容绑定到 **DataGrid** 组件的内容中，请打开“组件”检查器，然后单击“绑定”选项卡。
11. 在舞台上选择 **DataGrid** 组件 (`user_dg`)，然后单击“组件”检查器中的“添加绑定” (+) 按钮。
12. 在“添加绑定”对话框中，选择 **dataProvider: Array**，然后单击“确定”。
13. 在“组件”检查器中双击 **bound to** 字段。
14. 在出现的“绑定到”对话框中，从“组件路径”列中选择 **DataSet <user\_ds>**，然后从“架构位置”列中选择 **dataProvider: Array**。
15. 要将 **DataSet** 组件的选定索引绑定到 **DataGrid** 组件的选定索引，请在舞台上选择 **DataGrid** 组件，并在“组件”检查器中再次单击“添加绑定” (+) 按钮。
16. 在出现的对话框中，选择 **selectedIndex: Number**。单击“确定”。
17. 在“组件”检查器中双击 **bound to** 字段，打开“绑定到”对话框。
18. 在“组件路径”字段中，从“组件路径”列中选择 **DataSet <user\_ds>**，然后从“架构位置”列中选择 **selectedIndex: Number**。
19. 在“动作”面板中输入下面的代码：

```
next_button.addEventListener("click", nextBtnClick);
function nextBtnClick(evt_obj:Object):Void {
 user_ds.next();
}
```

此代码使用 `DataSet.next()` 方法浏览到 **DataSet** 对象项目集中的下一个项目。由于之前已将 **DataGrid** 对象的 `selectedIndex` 属性绑定到 **DataSet** 对象的同一属性，因此，如果更改 **DataSet** 对象中的当前项，**DataGrid** 对象中的当前（选定）项也会随之更改。

20. 保存文件，然后选择“控制” > “测试影片”来测试 SWF 文件。

**DataGrid** 对象即会被指定项目填充。注意单击按钮将如何更改 **DataGrid** 对象中的选定项目。

# DataSet 类（仅限 Flash Professional）

继承 MovieClip > DataSet

ActionScript 类名称 mx.data.components.DataSet

DataSet 组件使您能够将数据处理为可进行索引、排序、搜索、过滤和修改的对象的集合。

DataSet 组件功能包括 DataSetIterator（一组用于遍历和处理数据集合的方法）和 DeltaPacket（一组用于处理数据集合更新的接口和类）。大多数情况下，您不会直接使用这些类和接口；您将通过 DataSet 类提供的方法间接使用它们。

## DataSet 类的方法摘要

下表列出了 DataSet 类的方法。

| 方法                                       | 说明                                               |
|------------------------------------------|--------------------------------------------------|
| <a href="#">DataSet.addItem()</a>        | 将指定项目添加到集合。                                      |
| <a href="#">DataSet.addItemAt()</a>      | 将项目添加到数据集的指定位置。                                  |
| <a href="#">DataSet.addSort()</a>        | 创建集合中项目的新的已排序视图。                                 |
| <a href="#">DataSet.applyUpdates()</a>   | 指出 deltaPacket 属性具有您可以使用数据绑定或 ActionScript 访问的值。 |
| <a href="#">DataSet.changesPending()</a> | 指出集合是否具有尚未以增量数据包形式发送的待定更改。                       |
| <a href="#">DataSet.clear()</a>          | 从集合的当前视图中清除所有项目。                                 |
| <a href="#">DataSet.createItem()</a>     | 返回最新初始化的集合项目。                                    |
| <a href="#">DataSet.disableEvents()</a>  | 停止向侦听器发送 DataSet 事件。                             |
| <a href="#">DataSet.enableEvents()</a>   | 继续向侦听器发送 DataSet 事件。                             |
| <a href="#">DataSet.find()</a>           | 在集合的当前视图中定位项目。                                   |
| <a href="#">DataSet.findFirst()</a>      | 在集合的当前视图中定位出现的第一个项目。                             |
| <a href="#">DataSet.findLast()</a>       | 在集合的当前视图中定位出现的最后一个项目。                            |
| <a href="#">DataSet.first()</a>          | 移至集合当前视图中的第一个项目。                                 |
| <a href="#">DataSet.getItemId()</a>      | 返回指定项目的唯一 ID。                                    |
| <a href="#">DataSet.getIterator()</a>    | 返回当前重复值的克隆。                                      |
| <a href="#">DataSet.getLength()</a>      | 返回数据集中的项目数。                                      |
| <a href="#">DataSet.hasNext()</a>        | 指明当前的重复值是否位于集合视图的结尾。                             |
| <a href="#">DataSet.hasPrevious()</a>    | 指明当前的重复值是否位于集合视图的开头。                             |
| <a href="#">DataSet.hasSort()</a>        | 指明是否存在指定的排序。                                     |

| 方法                                       | 说明                                            |
|------------------------------------------|-----------------------------------------------|
| <code>DataSet.isEmpty()</code>           | 指明集合是否为空。                                     |
| <code>DataSet.last()</code>              | 移至集合当前视图中的最后一个项目。                             |
| <code>DataSet.loadFromSharedObj()</code> | 加载从共享对象中恢复此 <code>DataSet</code> 集合所需的所有相关数据。 |
| <code>DataSet.locateById()</code>        | 将当前重复值移到具有指定 ID 的项目。                          |
| <code>DataSet.next()</code>              | 移至集合当前视图中的下一个项目。                              |
| <code>DataSet.previous()</code>          | 移至集合当前视图中的上一个项目。                              |
| <code>DataSet.removeAll()</code>         | 从集合中删除所有项目。                                   |
| <code>DataSet.removeItem()</code>        | 从集合中删除指定项目。                                   |
| <code>DataSet.removeItemAt()</code>      | 删除指定位置的数据集项目。                                 |
| <code>DataSet.removeRange()</code>       | 删除当前重复值的范围设置。                                 |
| <code>DataSet.removeSort()</code>        | 从 <code>DataSet</code> 对象中删除指定排序。             |
| <code>DataSet.saveToSharedObj()</code>   | 将 <code>DataSet</code> 对象中的数据保存到共享对象。         |
| <code>DataSet.setIterator()</code>       | 设置 <code>DataSet</code> 对象的当前重复值。             |
| <code>DataSet.setRange()</code>          | 设置当前重复值的范围设置。                                 |
| <code>DataSet.skip()</code>              | 在集合的当前视图中按指定项目数向前或向后移动。                       |
| <code>DataSet.useSort()</code>           | 使指定排序处于活动状态。                                  |

## DataSet 类的属性摘要

下表列出了 `DataSet` 类的属性。

| 属性                                 | 说明                       |
|------------------------------------|--------------------------|
| <code>DataSet.currentItem</code>   | 返回集合中的当前项目。              |
| <code>DataSet.dataProvider</code>  | 返回数据提供程序。                |
| <code>DataSet.deltaPacket</code>   | 返回对集合所做的更改，或分配要对集合进行的更改。 |
| <code>DataSet.filtered</code>      | 指明是否已筛选项目。               |
| <code>DataSet.filterFunc</code>    | 用于在集合中筛选项目的用户定义函数。       |
| <code>DataSet.items</code>         | 集合中的项目。                  |
| <code>DataSet.itemClassName</code> | 指定项目时要创建的对象名称。           |
| <code>DataSet.length</code>        | 指定集合当前视图中的项目数。           |
| <code>DataSet.logChanges</code>    | 指明是否记录对集合或其项目所做的更改。      |

| 属性                                 | 说明                   |
|------------------------------------|----------------------|
| <code>DataSet.properties</code>    | 包含此集合内任何传输对象的属性（字段）。 |
| <code>DataSet.readOnly</code>      | 指明是否能修改集合。           |
| <code>DataSet.schema</code>        | 以 XML 格式指定集合的架构。     |
| <code>DataSet.selectedIndex</code> | 包含集合中当前项目的索引。        |

## DataSet 类的事件摘要

下表列出了 `DataSet` 类的事件。

| 事件                                      | 说明                                                 |
|-----------------------------------------|----------------------------------------------------|
| <code>DataSet.addItem</code>            | 在将项目添加到集合之前广播。                                     |
| <code>DataSet.afterLoaded</code>        | 在分配了 <code>items</code> 属性之后广播。                    |
| <code>DataSet.calcFields</code>         | 在应更新计算所得字段时广播。                                     |
| <code>DataSet.deltaPacketChanged</code> | 在 <code>DataSet</code> 对象的增量数据包已更改并且可以使用时广播。       |
| <code>DataSet.iteratorScrolled</code>   | 在重复值的位置更改时广播。                                      |
| <code>DataSet.modelChanged</code>       | 在集合中项目的某些方面已被修改时广播。                                |
| <code>DataSet.newItem</code>            | 在 <code>DataSet</code> 对象构建了新传输对象时（但在将其添加到集合之前）广播。 |
| <code>DataSet.removeItem</code>         | 在删除了项目时广播。                                         |
| <code>DataSet.resolveDelta</code>       | 在将增量数据包分配给包含消息的 <code>DataSet</code> 对象时广播。        |

## DataSet.addItem

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.addItem = function (eventObj:Object) {
 // ...
}
```

```
};
dataSetInstance.addEventListener("addItem", listenerObject);
```

用法 2:

```
on (addItem) {
 // ...
}
```

## 说明

事件：就在将新记录插入此集合之前生成。

如果将事件对象的 `result` 属性设置为 `false`，则添加操作将被取消；如果将其设置为 `true`，则允许添加操作。

事件对象 (*eventObj*) 包含以下属性：

`target` 生成该事件的 **DataSet** 对象。

`type` 字符串 "addItem"。

`item` 指向集合中要添加的项目的引用。

`result` 一个布尔值，它指定是否应添加指定项目。默认情况下，此值为 `true`。

## 示例

如果名为 `userHasAdminPrivs()` 的用户定义函数返回 `false`，下面的 `addItem` 事件处理函数将取消新项目的添加操作；否则允许添加项目。

```
function userHasAdminPrivs():Boolean {
 return false; // 将该值更改为 "true" 以允许插入项。
}
```

```
my_ds.addEventListener("addItem", addItemListener);
my_ds.addItem({name:"Bobo", occupation:"clown"});
```

```
function addItemListener(evt_obj:Object):Void {
 if (userHasAdminPrivs()) {
 // 允许添加项目。
 evt_obj.result = true;
 trace("Item added");
 } else {
 // 不允许添加项目；用户没有管理员权限。
 evt_obj.result = false;
 trace("Error, insufficient permissions");
 }
}
```

## 另请参见

[DataSet.removeItem](#)



# DataSet.addItem()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.addItem([obj])
```

## 参数

*obj* 要添加到此集合的对象。此参数是可选的。

## 返回

布尔值：如果该项目已添加到集合，则为 `true`；反之，则为 `false`。

## 说明

方法：将指定记录（传输对象）添加到集合以便管理。新添加的项目将成为数据集的当前项目。如果未指定 *obj* 参数，则会通过 `DataSet.createItem()` 自动创建新的对象。

新项目在集合中的位置取决于是否为当前重复值指定了排序。如果未使用排序，则会将该项目添加到集合的结尾。如果使用了排序，则会依据项目在当前排序中的位置将其添加到集合中。

有关初始化和构建传输对象的更多信息，请参见 `DataSet.createItem()`。

## 示例

以下示例使用 `DataSet.addItem()` 创建新的项目，并将其添加到数据集：

```
my_ds.addEventListener("addItem", addItemListener);
my_ds.addItem({name:"Bobo", occupation:"clown"});

function addItemListener(evt_obj:Object):Void {
 trace("adding item");
}
```

以下示例演示如何通过处理函数内部将 addItem 事件的结果设置为 true 或 false 来接受或拒绝向 DataSet 中插入项。将一个 DataSet 组件拖到舞台上，然后为其指定实例名称 my\_ds。将一个 DataGrid 组件拖到舞台上，然后为其指定实例名称 my\_dg。将一个 CheckBox 组件拖到舞台上，然后为其指定实例名称 my\_ch。将一个 Button 组件拖到舞台上，然后为其指定实例名称 submit\_button。使用“组件”检查器的“架构”选项卡将 name 和 occupation 两个属性添加到 DataSet 组件中。使用“组件”检查器的“绑定”面板创建 my\_ds.dataProvider 属性和 my\_dg.dataProvider 属性之间的绑定。将以下 ActionScript 代码添加到主时间轴的第 1 帧中：

```
my_ds.addEventListener("addItem", addItemListener);
submit_button.addEventListener("click", submitListener);
function userHasAdminPrivs():Boolean {
 return my_ch.selected;
}
function addItemListener(evt_obj:Object):Void {
 if (userHasAdminPrivs()) {
 // 允许添加项目。
 evt_obj.result = true;
 trace("Item added");
 } else {
 // 不允许添加项；用户没有管理员权限。
 evt_obj.result = false;
 trace("Error, insufficient permissions");
 }
}
function submitListener(evt_obj:Object):Void {
 my_ds.addItem({name:"bobo", occupation:"clown"});
}
```

另请参见

[DataSet.createItem\(\)](#)

## DataSet.addItemAt()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSetInstance.addItemAt(index, item)
```

### 参数

*index* 一个等于或大于 0 的数字。此数字指示插入项目的位置；它是新项目的索引。

*item* 包含项目数据的对象。

### 返回

一个布尔值，它指示是否添加了项目：true 指示添加了项目，false 指示该项目在数据集中已存在。

### 说明

方法：将新项目添加到数据集的指定索引处。大于数据提供程序长度的索引将被忽略。

此方法触发事件类型为 addItem 的 modelChanged 事件。

### 示例

以下示例使用 addItemAt() 方法在 DataSet 的第一个位置添加项目：

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});
my_ds.addItemAt(0, {name:"Bobo", years:1});
```

## DataSet.addSort()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*dataSetInstance.addSort(name, fieldList, sortOptions)*

### 参数

*name* 一个字符串，它指定排序的名称。

*fieldList* 一个字符串数组，它指定要进行排序的字段名。

*sortOptions* 以下一个或多个整数（常数）值，它们指示为此排序使用哪些选项。使用换位“或”运算符 (|) 分隔多个值。指定一个或多个下面的值：

- DataSetIterator.Ascending 按升序对项目进行排序。这是默认的排序选项（如果未指定任何选项）。

- `DataSetIterator.Descending` 依据指定的项目属性按降序对项目进行排序。
- `DataSetIterator.Unique` 防止排序（如果任何字段包含相同的值）。
- `DataSetIterator.CaseInsensitive` 在排序操作期间比较两个字符串时忽略大小写。默认情况下，如果正在进行排序的属性是字符串，则排序是区分大小写的。

如果将 `DataSetIterator.Unique` 指定为排序选项且正在排序的数据不唯一，添加了指定的排序名称或者 `fieldList` 数组中指定的属性在此数据集中不存在，则会引发 `DataSetError` 例外。

## 返回

无。

## 说明

方法；依据 `fieldList` 参数指定的属性为当前重复值创建新的升序或降序排序。**Flash** 在创建了新的排序后会自动将新的排序分配给当前重复值，并将其存储在排序集合中以供稍后检索。

## 示例

以下代码将创建名为 "nameSort" 的新排序，该排序将对 `DataSet` 对象的 "name" 字段执行不区分大小写的降序排序。

```
import mx.data.components.datasetclasses.DataSetIterator;

my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addSort("nameSort", ["name"], DataSetIterator.Descending |
 DataSetIterator.Unique | DataSetIterator.CaseInsensitive);
```

在以下示例中，通过向舞台上的 `TextInput` 组件实例中输入第一个和最后一个名称并单击“提交”按钮，您可以向 `DataSet` 组件中动态添加数据。向 `DataSet` 组件添加项后，可以通过单击舞台上的“清除”按钮清除数据集。将一个 `DataGrid` 组件拖到舞台上，然后为其指定实例名称 `my_dg`。将两个 `Button` 组件拖到舞台上，然后分别为其指定实例名称 `submit_button` 和 `clear_button`。将一个 `DataSet` 组件拖到舞台上，然后为其指定实例名称 `my_ds`。将两个 `TextInput` 组件拖到舞台上，然后分别为其指定实例名称 `firstName_ti` 和 `lastName_ti`。将一个 `Alert` 组件拖到当前文档的库中。使用“组件”检查器的“架构”选项卡将 `firstName` 和 `lastName` 两个组件属性添加到 `DataSet` 组件的架构中。接着，使用“组件”检查器中的“绑定”选项卡添加从 `DataSet` 组件的 `dataProvider` 属性到 `DataGrid` 组件的 `dataProvider` 属性的数据绑定。最后，将以下代码粘贴到主时间轴的第一帧中：

```
import mx.controls.Alert;

my_ds.addSort("lastFirst", ["lastName", "firstName"]);
```

```

my_dg.enabled = false;
clear_button.enabled = false;
submit_button.label = "Submit";
clear_button.label = "Clear";

my_ds.addEventListener("addItem", addItemListener);
my_ds.addEventListener("modelChanged", modelChangedListener);
submit_button.addEventListener("click", submitListener);
clear_button.addEventListener("click", clearListener);

function modelChangedListener(evt_obj:Object):Void {
 my_dg.enabled = (evt_obj.target.length > 0);
 clear_button.enabled = my_dg.enabled;
}
function submitListener(evt_obj:Object):Void {
 my_ds.addItem({firstName:firstName_ti.text, lastName:lastName_ti.text});
}
function addItemListener(evt_obj:Object):Void {
 if ((evt_obj.item.firstName.length == 0) || (evt_obj.item.lastName.length
 == 0)) {
 Alert.show("Error, first name or last name cannot be blank.", "Error",
 Alert.OK, _level0);
 evt_obj.result = false;
 } else {
 firstName_ti.text = "";
 lastName_ti.text = "";
 }
}
function clearListener(evt_obj:Object):Void {
 Alert.show("Are you sure you want to clear the data?", "Warning", Alert.OK
 | Alert.CANCEL, _level0, clearConfirmListener);
}
function clearConfirmListener(evt_obj:Object):Void {
 switch (evt_obj.detail) {
 case Alert.OK:
 my_ds.clear();
 break;
 case Alert.CANCEL:
 break;
 }
}
}

```

选择“控制”>“测试影片”在创作环境中对文档进行测试。在两个 **TextInput** 实例中都输入一些文本，然后单击“提交”按钮。**DataGrid** 实例中将添加一个新项。单击“清除”按钮将显示一个 **Alert** 组件实例，其中的确认消息会询问您是否要清除 **DataGrid** 的内容。单击“确定”将会清除 **DataSet** 组件的 `dataProvider` 属性（然后会因为绑定关系而清除 **DataGrid** 组件的 `dataProvider` 属性）。单击“取消”将取消 **Alert** 组件实例。

**另请参见**

[DataSet.removeSort\(\)](#)

# DataSet.afterLoaded

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.afterLoaded = function (eventObj:Object):Void {
 // ...
};
dataSetInstance.addEventListener("afterLoaded", listenerObject);
```

用法 2:

```
on (afterLoaded) {
 // ...
}
```

## 说明

事件：在分配了 `DataSet.items` 属性后立即广播。

事件对象 (*eventObj*) 包含以下属性：

`target` 生成该事件的 **DataSet** 对象。

`type` 字符串 "afterLoaded"。

## 示例

在以下示例中，一旦分配了数据集 `contact_ds` 中的项，名为 `contactForm` 的表单（未显示）将变得可见。

```
contact_ds.addEventListener("afterLoaded", loadListener);
var loadListener:Object = new Object();
loadListener.afterLoaded = function (evt_obj:Object) {
 if (evt_obj.target == "contact_ds") {
 contactForm.visible = true;
 }
};
```

以下示例将使用 **DataSet** 组件的 `afterLoaded` 事件来填充舞台上某个 **List** 组件的 `dataProvider` 属性。将 **List** 组件和 **DataSet** 组件拖到舞台上，并分别为它们指定实例名称 `my_list` 和 `my_ds`。将以下 **ActionScript** 代码添加到主时间轴的第一帧中：

```
my_list.labelField = "name";

var itemsListener:Object = new Object();
itemsListener.afterLoaded = function (evt_obj:Object):Void {
 trace("After loaded");
 my_list.dataProvider = evt_obj.target.items;
}
my_ds.addEventListener("afterLoaded", itemsListener);

var item_array:Array = [{name:"Douglas"}, {name:"Vinnie"},
 {name:"Katherine"}, {name:"David"}];
my_ds.items = item_array;
```

## DataSet.applyUpdates()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
dataSetInstance.applyUpdates()
```

### 返回

无。

### 说明

方法：指出 `DataSet.deltaPacket` 属性具有您可以使用数据绑定或直接通过 **ActionScript** 访问的值。在调用此方法之前，`DataSet.deltaPacket` 属性为 `null`。如果已通过 `DataSet.disableEvents()` 方法禁用了事件，则此方法无效。

调用此方法还会为当前 `DataSet.deltaPacket` 属性创建事务 ID，并发出 `deltaPacketChanged` 事件。有关更多信息，请参见 `DataSet.deltaPacket`。

## 示例

以下代码将对 `my_ds DataSet` 调用 `applyUpdates()` 方法。

```
my_ds.applyUpdates();
```

以下示例向舞台上的 `my_ds DataSet` 实例中添加四个项目并在 `deltaPacket` 属性的顶层显示每个项目：

```
my_ds.addItem({name:"Thomas", age:35, gender:"M"});
my_ds.addItem({name:"Orville", age:33, gender:"M"});
my_ds.addItem({name:"Jonathan", age:48, gender:"M"});
my_ds.addItem({name:"Carol", age:31, gender:"F"});

my_ds.applyUpdates();
var i:String;
for (i in my_ds.deltaPacket) {
 trace(i + ":\t" + my_ds.deltaPacket[i]);
}
```

## 另请参见

[DataSet.deltaPacket](#)

# DataSet.calcFields

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.calcFields = function (eventObj:Object):Void {
 // ...
};
dataSetInstance.addEventListener("calcFields", listenerObject);
```

用法 2：

```
on (calcFields) {
 // ...
}
```



## 说明

事件：在需要确定集合中当前项目的计算所得字段的值时生成。计算所得字段是指 **Kind** 属性在“组件”检查器中的“架构”选项卡上设置为“**Calculated**”的字段。您创建的 `calcFields` 事件侦听器应执行必需的计算，并设置计算所得字段的值。

在非计算所得字段（即 **Kind** 属性在“架构”选项卡上设置为“**Data**”的字段）的值更新时，也会调用此事件。

有关 **Kind** 属性的更多信息，请参见《使用 Flash》中的“架构种类”。



不要更改此事件中任何非计算所得字段的值，因为这将导致“无限循环”。只能设置 `calcFields` 事件内计算所得字段的值。

# DataSet.changesPending()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.changesPending()

## 返回

一个布尔值。

## 说明

方法：如果集合或集合内的任何项目具有尚未以增量数据包形式发送的待定更改，则返回 `true`；否则返回 `false`。

## 示例

如果 **DataSet** 集合或该集合内的任何项目具有尚未提交到增量数据包的修改，以下代码将启用“保存更改”按钮（未显示）。

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addEventListener("modelChanged", modelChangedListener);
function modelChangedListener(evt_obj:Object):Void {
 if (evt_obj.target.changesPending()) {
```

```
 trace("changes pending");
 submitChanges_button.enabled = true;
 }
}
submitChanges_button.enabled = false;
my_ds.addItem({name:"Hal", years:4});
```

## DataSet.clear()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*dataSetInstance*.clear()

### 返回

无。

### 说明

方法：删除集合当前视图中的项目。哪些项目被视为“可查看”取决于任何当前的过滤器和当前重复值的范围设置。因此，调用此方法可能不会清除集合中的所有项目。要清除集合中的所有项目而不考虑当前重复值的视图，请使用 [DataSet.removeAll\(\)](#)。

如果在调用此方法时 [DataSet.logChanges](#) 设置为 true，则会为集合内的所有项目将“remove”条目添加到 [DataSet.deltaPacket](#) 中。

### 示例

以下示例将从 **DataSet** 集合的当前视图中删除所有项。由于 `logChanges` 属性设置为 true，因此将记录这些项目的删除。

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addSort("nameSort", ["name"]);
my_ds.filtered = true;
my_ds.filterFunc = function(item:Object):Boolean {
 return (item.years >= 3);
};
my_ds.logChanges = true;
```

```
my_ds.clear(); // 删除从 DataSet 中过滤的项。
my_ds.removeSort("nameSort");
```

### 另请参见

[DataSet.deltaPacket](#)、[DataSet.logChanges](#)

## DataSet.createItem()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
dataSetInstance.createItem([itemData])
```

### 参数

*itemData* 与项目关联的数据。此参数是可选的。

### 返回

新构建的项目。

### 说明

方法；创建不与集合关联的项目。可以使用 [DataSet.itemClassName](#) 属性指定所创建的对象。如果未指定 [DataSet.itemClassName](#) 值并且省略了 *itemData* 参数，则会构建匿名对象。此匿名对象的属性将根据 [DataSet.schema](#) 当前指定的架构被设置为默认值。

在调用此方法时，[DataSet.newItem](#) 事件的所有侦听器都将收到通知，并且能够在此方法返回项之前对项进行处理。可选项数据用于初始化通过 [DataSet.itemClassName](#) 属性指定的类，或者在 [DataSet.itemClassName](#) 为空时用作项。

如果无法加载通过 [DataSet.itemClassName](#) 属性指定的类，则会引发 `DataSetError` 异常。

### 示例

```
my_ds.addEventListener("newItem", newItemListener);
function newItemListener(evt_obj:Object):Void {
 trace("new item was added: {name:'" + evt_obj.item.name + "', years:" +
 evt_obj.item.years + "}");
}

my_ds.addItem(my_ds.createItem({name:"Wilson", years:3}));
my_ds.addItem({name:"Tom", years:2});
```

```
my_ds.filtered = true;
my_ds.filterFunc = function(item:Object):Boolean {
 return (item.years % 2 == 0);
};
```

### 另请参见

[DataSet.itemClassName](#)、[DataSet.newItem](#)、[DataSet.schema](#)

## DataSet.currentItem

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*dataSetInstance*.currentItem

### 说明

属性（只读）：返回 **DataSet** 集合中的当前项目，或者，如果集合为空或集合当前重复值的视图为空，则会返回 `null`。

此属性提供对集合内的项目的直接访问。不会跟踪通过直接访问此对象所进行的更改（在 [DataSet.deltaPacket](#) 属性中），也不会对此对象的任何属性应用任何架构设置。

### 示例

以下示例显示 `name` 属性（在名为 `customers_ds` 的数据集的当前项目中定义）的值。

```
customers_ds.addItem({name:"Milton", years:3});
customers_ds.addItem({name:"Mark", years:3});
customers_ds.addItem({name:"Sarah", years:1});
customers_ds.addItem({name:"Michael", years:2});
customers_ds.addItem({name:"Frank", years:2});

trace(customers_ds.currentItem.name); // Frank
```

# DataSet.dataProvider

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.dataProvider

## 说明

属性；此数据集的数据提供程序。此属性为用户界面控件（如 **List** 和 **DataGrid** 组件）提供数据。

有关 **DataProvider** API 的更多信息，请参见第 295 页的“**DataProvider API**”。

## 示例

以下代码将 **DataSet** 对象的 **dataProvider** 属性分配给 **DataGrid** 组件的对应属性。

```
my_dg.dataProvider = my_ds.dataProvider;
```

# DataSet.deltaPacket

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.deltaPacket

## 说明

属性；返回增量数据包，它包含对 *dataSet* 集合及其项目所做的所有更改操作。在对 *dataSet* 调用 **DataSet.applyUpdates()** 之前，此属性为 **null**。

在调用 **DataSet.applyUpdates()** 时，会将一个事务 ID 分配给增量数据包。此事务 ID 用于在从服务器开始并返回到客户端的更新往返行程中标识增量数据包。增量数据包（具有匹配事务 ID）为 **deltaPacket** 属性分配的任何后续值都将被视为对之前所发送更改的服务器响应。具有匹配 ID 的增量数据包用于更新集合，并报告包内指定的错误。

将会向 `DataSet.resolveDelta` 事件的侦听器报告错误或服务器消息。请注意，在将具有匹配 ID 的增量数据包分配给 `DataSet.deltaPacket` 时，将忽略 `DataSet.logChanges` 设置。没有匹配事务 ID 的增量数据包将更新集合，就像直接使用 **DataSet API** 一样。这样可能会创建额外的增量条目，具体取决于 `dataSet` 和增量数据包当前的 `DataSet.logChanges` 设置。

如果为增量数据包分配了匹配的事务 ID，并且无法在原始增量数据包中找到新分配的增量数据包中的某个项目，则会引发 `DataSetError` 例外。

#### 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.logChanges](#)、[DataSet.resolveDelta](#)

## DataSet.deltaPacketChanged

#### 可用性

Flash Player 7。

#### 版本

Flash MX Professional 2004。

#### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.deltaPacketChanged = function (eventObj:Object):Void {
 // ...
};
dataSetInstance.addEventListener("deltaPacketChanged", listenerObject);
```

用法 2:

```
on (deltaPacketChanged) {
 // ...
}
```

#### 说明

事件：在指定 **DataSet** 对象的 `deltaPacket` 属性已更改并且可供使用时广播。

#### 另请参见

[DataSet.deltaPacket](#)

# DataSet.disableEvents()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.disableEvents()*

## 返回

无。

## 说明

方法：为 **DataSet** 对象禁用事件。如果禁用了事件，则在对集合中的项目进行更改或 **DataSet** 对象滚动到集合中的另一项目时，将不会更新用户界面控件（如 **DataGrid** 组件）。

若要重新启用事件，必须调用 [DataSet.enableEvents\(\)](#)。`disableEvents()` 方法可被调用多次，因此 `enableEvents()` 必须被调用相同的次数才能重新启用事件发送。

## 示例

在以下示例中，在对集合中的项进行更改前事件已被禁用，因此 **DataSet** 对象将不会尝试刷新控件而影响性能：

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
 trace("model changed, DataSet now has " + evt_obj.target.items.length + "
 items");
}
// 禁用数据集的事件。
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while(my_ds.hasPrevious()) {
 my_ds.price *= 0.5; // 所有项均为五折！
 my_ds.previous();
}

trace("After:");
```

```
traceItems();

// 告诉数据集现在应该更新控件。
my_ds.enableEvents();

function traceItems():Void {
 for (var i:Number = 0; i < my_ds.items.length; i++) {
 trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
 }
 trace("");
}
```

另请参见

[DataSet.enableEvents\(\)](#)

## DataSet.enableEvents()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

*dataSetInstance*.enableEvents()

返回

无。

说明

方法；在通过调用 [DataSet.disableEvents\(\)](#) 禁用了事件之后为 **DataSet** 对象重新启用事件。若要为 **DataSet** 对象重新启用事件，enableEvents() 方法的调用次数必须等于或大于 disableEvents() 的调用次数。

示例

在以下示例中，在对集合中的项进行更改前事件已被禁用，因此 **DataSet** 对象将不会尝试刷新控件而影响性能。

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
 trace("model changed, DataSet now has " + evt_obj.target.items.length + "
 items");
}
// 禁用数据集的事件。
```



```

my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while(my_ds.hasPrevious()) {
 my_ds.price *= 0.5; // 所有项均为五折!
 my_ds.previous();
}

trace("After:");
traceItems();

// 告诉数据集现在应该更新控件。
my_ds.enableEvents();

function traceItems():Void {
 for (var i:Number = 0; i < my_ds.items.length; i++) {
 trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
 }
 trace("");
}

```

**另请参见**

[DataSet.disableEvents\(\)](#)

## DataSet.filtered

**可用性**

Flash Player 7。

**版本**

Flash MX Professional 2004。

**用法**

*dataSetInstance.filtered*

**说明**

属性；一个布尔值，它指明是否过滤当前重复值中的数据。默认值为 `false`。当此属性为 `true` 时，则为集中的每个项调用由 [DataSet.filterFunc](#) 指定的过滤函数。

## 示例

在以下示例中，对名为 `employee_ds` 的 **DataSet** 对象启用过滤。假设 **DataSet** 集合中的每条记录都包含一个名为 `empType` 的字段。如果当前项目中的 `empType` 字段设置为 "management"，以下过滤函数将返回 `true`；否则返回 `false`。

```
employee_ds.filtered = true;
employee_ds.filterFunc = function (item:Object) {
 // 过滤出职务为经理的那些员工。
 return(item.empType != "management");
};
```

以下示例将使用 **XMLConnector** 组件动态加载的内容填充到 **DataGrid** 组件中。当用户单击舞台上的某个 **CheckBox** 实例时，将会在 **DataGrid** 组件中自动过滤和更新 **DataSet** 组件的内容。

将以下组件拖到舞台上，并为它们指定以下实例名称：

- **CheckBox** (`editorsChoice_ch`)
- **DataGrid** (`reviews_dg`)
- **DataSet** (`reviews_ds`)
- **XMLConnector** (`reviews_xmlconn`)

下载以下 XML 文档的副本并将其保存到您的本地硬盘上：<http://www.helpexamples.com/flash/xml/reviews.xml>。

使用 **XMLConnector** 组件将动态加载此 XML 文档，但您将使用本地副本将 XML 架构导入到 **DataSet** 组件中。选择舞台上的 **XMLConnector** 实例，并选择“组件”检查器中的“架构”选项卡。选择 `results` 属性并单击“从示例 XML 文件导入架构”按钮。选择您已下载到本地硬盘的 XML 文档并单击“打开”。XML 文档的架构将被导入到 **DataSet** 组件中。在 `reviews_xmlconn`

**XMLConnector** 实例在舞台上仍然保持选中状态的情况下，添加从

`reviews_xmlconn.results.reviews.review` 数组到 `reviews_ds` **DataSet** 实例的 `dataProvider` 属性的绑定。在“绑定”选项卡中将数据绑定的方向设置为“out”。选择舞台上的 `reviews_ds` **DataSet** 实例，并为 `dataProvider` 属性添加另一个绑定。在保持选中 `reviews_ds` 实例的情况下，选择“组件”检查器的“绑定”选项卡。单击“添加绑定”按钮，然后添加从 **DataSet** 组件的 `dataProvider` 属性到 `reviews_dg` **DataGrid** 实例的 `dataProvider` 属性的绑定。

将以下代码添加到主时间轴中的第 1 帧：

```
editorsChoice_ch.label = "Editor's Choice";

reviews_xmlconn.direction = "receive";
reviews_xmlconn.multipleSimultaneousAllowed = false;
reviews_xmlconn.URL = "http://www.helpexamples.com/flash/xml/reviews.xml";
reviews_xmlconn.trigger();

reviews_dg.setSize(320, 240);
```

```

reviews_dg.addColumn("name");
reviews_dg.addColumn("rating");
reviews_dg.addColumn("reviewDate");
reviews_dg.getColumnAt(0).width = 100;
reviews_dg.getColumnAt(1).width = 100;
reviews_dg.getColumnAt(2).width = 100;
reviews_dg.setStyle("alternatingRowColors", [0xFFFFFF, 0xF6F6F6]);

editorsChoice_ch.addEventListener("click", editorsChoiceListener);
function editorsChoiceListener(evt_obj:Object):Void {
 reviews_ds.filtered = evt_obj.target.selected;
 reviews_ds.filterFunc = function(item:Object):Boolean {
 return (item.editorsChoice == 1);
 };
}

```

选择“控制”>“测试影片”。默认情况下，**DataGrid** 组件将显示对外部 XML 文档的每次评论。单击 **Editor Choice** 复选框组件将导致对 **DataSet** 组件进行过滤，这样将只会显示标记为 **Editor Choice** 的特定评论。

另请参见

[DataSet.filterFunc](#)

## DataSet.filterFunc

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```

dataSetInstance.filterFunc = function (item:Object):Boolean {
 // 返回 true|false;
};

```

说明

属性；指定一个函数，该函数确定在集合的当前视图中包括哪些项目。[DataSet.filtered](#) 设置为 `true` 时，会为集合中的每条记录（传输对象）调用分配给此属性的函数。对于传送到函数的每个项目，如果项目应包括在当前视图中，则它应返回 `true`，或者，如果项目不应包括在当前视图中，则应返回 `false`。

当更改数据集的过滤函数时，必须将 `filtered` 属性设置为 `false`，然后再设置为 `true`，以便生成正确的 `modelChanged` 事件。更改 `filterFunc` 属性不会生成事件。

而且，如果加载数据时（modelChanged 或 updateAll）已经存在过滤器，那么在 filtered 设置为 false 然后又设置回 true 之前，不会应用该过滤器。

## 示例

在以下示例中，对名为 employee\_ds 的 **DataSet** 对象启用过滤。如果每个项目中的 empType 字段设置为 "management"，则指定的过滤函数返回 true；否则返回 false。

```
employee_ds.filtered = true;
employee_ds.filterFunc = function (item:Object):Boolean {
 // 过滤出职务为经理的那些员工。
 return(item.empType != "management");
};
```

在以下示例中，您将根据 **ComboBox** 组件中的所选项过滤 **DataSet** 组件的内容。**ComboBox** 组件允许您选择所有人、仅限男性或仅限女性。

将 **DataSet**、**DataGrid** 和 **ComboBox** 组件拖到舞台上，并分别为它们指定实例名称 my\_ds、data\_dg 和 filter\_cb。选择舞台上的 my\_ds **DataSet** 实例并添加两个新属性：name 和 gender。使用“组件”检查器的“绑定”面板在 **DataSet** 的 dataProvider 属性和 **DataGrid** 组件的 dataProvider 之间创建数据绑定。将以下 **ActionScript** 代码添加到主时间轴的第一帧中：

```
my_ds.dataProvider = new Array({name:"Charles", gender:"M"}, {name:"Buddy",
 gender:"M"}, {name:"Walter", gender:"M"}, {name:"Ellen", gender:"F"},
 {name:"Jamie", gender:"F"}, {name:"Sarah", gender:"F"}, {name:"Adam",
 gender:"M"});
my_ds.addSort("nameSort", ["name"]);
my_ds.addSort("genderSort", ["gender", "name"]);
my_ds.useSort("genderSort");
```

```
filter_cb.dataProvider = [{label:"All", value:""} , {label:"Male only",
 value:"M"}, {label:"Female only", value:"F"}];
filter_cb.addEventListener("change", filterListener);
function filterListener(evt_obj:Object):Void {
 var selItem:Object = evt_obj.target.selectedItem;
 if (selItem.value.length == 0) {
 my_ds.filtered = false;
 } else {
 my_ds.filtered = true;
 my_ds.filterFunc = function(item:Object):Boolean {
 return (item.gender == selItem.value);
 }
 }
}
```

以下示例将使用 **XMLConnector** 组件动态加载的内容填充到 **DataGrid** 组件中。当用户单击舞台上的 **CheckBox** 实例时，将会在 **DataGrid** 组件中自动过滤和更新 **DataSet** 组件的内容。

将以下组件拖到舞台上，并为它们指定以下实例名称：

- **CheckBox** (editorsChoice\_ch)
- **DataGrid** (reviews\_dg)
- **DataSet** (reviews\_ds)
- **XMLConnector** (reviews\_xmlconn)

下载以下 XML 文档的副本并将其保存到您的本地硬盘上: [www.helpexamples.com/flash/xml/reviews.xml](http://www.helpexamples.com/flash/xml/reviews.xml)。此 XML 文档使用 **XMLConnector** 组件动态加载。您将使用本地副本将 XML 架构导入到 **DataSet** 组件中。选择舞台上的 **XMLConnector** 实例并选择“组件”检查器中的“架构”选项卡。选择 results 属性并单击“从示例 XML 文件导入架构”。选择您下载到本地硬盘的 XML 文档并单击“打开”。XML 文档的架构将被导入到 **DataSet** 组件中。在 reviews\_xmlconn **XMLConnector** 实例在舞台上保持选中状态的情况下，添加从 reviews\_xmlconn.results.reviews.review 数组到 reviews\_ds **DataSet** 实例的 dataProvider 属性的绑定。在“绑定”选项卡中将数据绑定的方向设置为“out”。选择舞台上的 reviews\_ds **DataSet** 实例并为 dataProvider 属性添加另一个绑定。在保持选中 reviews\_ds 实例的情况下，选择“组件”检查器的“绑定”选项卡。单击“添加绑定”按钮，添加从 **DataSet** 组件的 dataProvider 属性到 reviews\_dg **DataGrid** 实例的 dataProvider 属性的绑定。

将以下代码添加到主时间轴的第 1 帧中：

```
editorsChoice_ch.label = "Editor's Choice";

reviews_xmlconn.direction = "receive";
reviews_xmlconn.multipleSimultaneousAllowed = false;
reviews_xmlconn.URL = "http://www.helpexamples.com/flash/xml/reviews.xml";
reviews_xmlconn.trigger();

reviews_dg.setSize(320, 240);
reviews_dg.addColumn("name");
reviews_dg.addColumn("rating");
reviews_dg.addColumn("reviewDate");
reviews_dg.getColumnAt(0).width = 100;
reviews_dg.getColumnAt(1).width = 100;
reviews_dg.getColumnAt(2).width = 100;
reviews_dg.setStyle("alternatingRowColors", [0xFFFFFF, 0xF6F6F6]);

editorsChoice_ch.addEventListener("click", editorsChoiceListener);
function editorsChoiceListener(evt_obj:Object):Void {
 reviews_ds.filtered = evt_obj.target.selected;
 reviews_ds.filterFunc = function(item:Object):Boolean {
 return (item.editorsChoice == 1);
 };
}
```

选择“控制”>“测试影片”。默认情况下，**DataGrid** 组件将显示对外部 XML 文档的每次评论。单击 **Editor Choice** 复选框组件将导致对 **DataSet** 组件进行过滤，这样将只会显示标记为 **Editor Choice** 的特定评论。

另请参见

[DataSet.filtered](#)

## DataSet.find()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*dataSetInstance.find(searchValues)*

### 参数

*searchValues* 一个数组，它包含要在当前排序中查找的一个或多个字段值。

### 返回

如果找到这些值，则返回 `true`；否则返回 `false`。

### 说明

方法；使用 *searchValues* 指定的字段值搜索集合的当前视图以查找项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。如果找到，则找到的项目将成为 **DataSet** 对象中的当前项目。

*searchValues* 指定的值的顺序必须与当前排序指定的字段列表的顺序相同（请参见下面的示例）。

如果当前排序不唯一，则找到的记录（传输对象）不确定。如果要在非唯一排序中查找出现的第一个或最后一个传输对象，请使用 [DataSet.findFirst\(\)](#) 或 [DataSet.findLast\(\)](#)。

所指定数据的转换基于基础字段的类型。例如，如果指定 `["05-02-02"]` 作为搜索值，则会使用基础日期字段，通过日期的 `DataType.setAsString()` 方法来转换值。如果指定 `[new Date().getTime()]`，则使用日期的 `DataType.setAsNumber()` 方法。

## 示例

以下示例将在当前集合中查找 name 和 id 字段相应包含值 "Bobby" 和 105 的项。如果找到，则会使用 `DataSet.getItemId()` 来获取该项在集合中的唯一标识符，并使用 `DataSet.locateById()` 将当前重复值放在该项上。

```
var studentID:String = null;
student_ds.addSort("id", ["name","id"]);
// 查找由 "Bobby" 和 105 标识的传输对象。
// 请注意，搜索字段的顺序匹配
// 在 addSort() 中指定的顺序。
if (student_ds.find(["Bobby", 105])) {
 studentID = student_ds.getItemId();
}
// 现在使用 locateById() 将当前重复值定位到
// 集合中 ID 与 studentID 匹配的项上。
if (studentID != null) {
 student_ds.locateById(studentID);
}
```

## 另请参见

`DataSet.applyUpdates()`、`DataSet.getItemId()`、`DataSet.locateById()`

# DataSet.findFirst()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.findFirst(searchValues)
```

## 参数

*searchValues* 一个数组，它包含要在当前排序中查找的一个或多个字段值。

## 返回

如果找到这些项，则返回 true；否则，返回 false。

## 说明

方法：使用 `searchValues` 指定的字段值搜索集合的当前视图以查找第一个项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

`searchValues` 指定的值的顺序必须与当前排序指定的字段列表的顺序相同（请参见下面的示例）。

所指定数据的转换基于基础字段的类型。例如，如果指定的搜索值为 `["05-02-02"]`，则会使用基础日期字段通过日期的 `setAsString()` 方法来转换值。如果指定的值是 `[new Date().getTime()]`，则使用日期的 `setAsNumber()` 方法。

## 示例

以下示例将使用 `DataSet.find()` 在当前集合中搜索 `name` 和 `id` 字段分别包含值 "Bobby" 和 105 的项。如果找到，则会使用 `DataSet.getItemId()` 来获取该项的唯一标识符，并使用 `DataSet.locateById()` 将当前重复值放在该项上。

若要测试此示例，请将一个 **DataSet** 组件拖到舞台上，并为其指定实例名称 `student_ds`。使用“组件”检查器的“架构”选项卡，将 `name`（数据类型：字符串）和 `id`（数据类型：数字）这两个属性添加到 **DataSet** 中。如果库中没有 `DataBindingClasses` 编译剪辑的副本，请从“类”库（“窗口” > “公用库” > “类”）中拖动该编译剪辑的副本。将以下 **ActionScript** 代码添加到主时间轴的第 1 帧中：

```
student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

trace("Before find() > " + student_ds.currentItem.name); // Billy

var studentID:String;
student_ds.addSort("id", ["name","id"]);
if (student_ds.find(["Bobby", 105])) {
 studentID = student_ds.getItemId();
 student_ds.locateById(studentID);
 trace("After find() > " + student_ds.currentItem.name); // Bobby
} else {
 trace("We lost Billy!");
}
```

## 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.getItemId\(\)](#)、[DataSet.locateById\(\)](#)



# DataSet.findLast()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.findLast(searchValues)
```

## 参数

*searchValues* 一个数组，它包含要在当前排序中查找的一个或多个字段值。

## 返回

如果找到这些项，则返回 `true`；否则，返回 `false`。

## 说明

方法；使用 *searchValues* 指定的字段值搜索集合的当前视图以查找最后一个项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

*searchValues* 指定的值的顺序必须与当前排序指定的字段列表的顺序相同（请参见下面的示例）。

指定数据的转换基于基础字段的类型。例如，如果指定的搜索值为 `["05-02-02"]`，则会使用基础日期字段通过日期的 `setAsString()` 方法来转换值。如果指定的值是 `[new Date().getTime()]`，则使用日期的 `setAsNumber()` 方法。

## 示例

以下示例将在当前集合中搜索 `name` 和 `age` 字段包含 "Bobby" 和 "13" 的最后一项。如果找到，则会使用 `DataSet.getItemId()` 来获取项在集合中的唯一标识符，并使用 `DataSet.locateById()` 将当前重复值放在该项上。

```
var studentID:String = null;
student_ds.addSort("nameAndAge", ["name", "age"]);
// 查找具有指定值的最后一个传输对象。
// 请注意，搜索字段的顺序匹配
// 在 addSort() 中指定的顺序。
if (student_ds.findLast(["Bobby", "13"])) {
 studentID = student_ds.getItemId();
}
```

```
// 现在使用 locateById() 将当前重复值定位到
// 集合中 ID 与 studentID 匹配的项上。
if (studentID != null) {
 student_ds.locateById(studentID);
}
```

### 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.getItemId\(\)](#)、[DataSet.locateById\(\)](#)

## DataSet.first()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
dataSetInstance.first()
```

### 返回

无。

### 说明

方法：使集合当前视图中的第一个项目成为当前项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

### 示例

下面的代码将数据集 `inventory_ds` 放在其集合中的第一项上，然后使用 `DataSet.currentItem` 属性显示该项包含的 `price` 属性的值。

```
inventory_ds.first();
trace("The price of the first item is:" + inventory_ds.currentItem.price);
```

以下示例将遍历当前集合视图中的所有项（从开头位置开始），并对每一项的 `price` 属性执行计算。

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.first();
while (my_ds.hasNext()) {
 my_ds.currentItem.price *= 0.5; // 所有项均为五折！
 my_ds.next();
}
```

```
for (var i in my_ds.items) {
 trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

另请参见

[DataSet.last\(\)](#)

## DataSet.getItemId()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSetInstance.getItemId([index])
```

参数

*index* 一个数值，它指定要获取其 ID 的当前视图中的项目。此参数是可选的。

返回

字符串。

说明

方法；返回当前项目在集合中的标识符，或返回由 *index* 指定的项目的标识符。此标识符只有在该集合中才是唯一的，它由 [DataSet.addItem\(\)](#) 自动分配。

示例

以下代码将获取当前项在集合中的唯一 ID，然后在“输出”面板中显示该 ID。

```
var itemNo:String = my_ds.getItemId();
trace("Employee id(" + itemNo + ")");
```

以下示例将使用 [DataSet.find\(\)](#) 在当前集合中搜索 *name* 和 *id* 字段分别包含值 "Bobby" 和 105 的项。如果找到，则会使用 [DataSet.getItemId\(\)](#) 来获取该项的唯一标识符，并使用 [DataSet.locateById\(\)](#) 将当前重复值放在该项上。

要测试此示例，请将 **DataSet** 组件拖到舞台上并为其指定实例名称 `student_ds`。使用“组件”检查器的“架构”选项卡，将 `name`（数据类型：字符串）和 `id`（数据类型：数字）两个属性添加到 **DataSet** 中。如果库中没有 `DataBindingClasses` 编译剪辑的副本，请从“类”库（“窗口” > “公用库” > “类”）中拖动该编译剪辑的副本。将以下 **ActionScript** 代码添加到主时间轴的第 1 帧中：

```
student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

trace("Before find() > " + student_ds.currentItem.name); // Billy

var studentID:String;
student_ds.addSort("id", ["name","id"]);
if (student_ds.find(["Bobby", 105])) {
 studentID = student_ds.getItemId();
 student_ds.locateById(studentID);
 trace("After find() > " + student_ds.currentItem.name); // Bobby
} else {
 trace("We lost Billy!");
}
```

另请参见

[DataSet.addItem\(\)](#)

## DataSet.getIterator()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSetInstance.getIterator()
```

返回

一个 `ValueListIterator` 对象。

说明

方法：返回此集合的一个新重复值；此重复值是正在使用的当前重复值的克隆，其中包括它在集合内的当前位置。此方法主要供想要访问同一集合同时出现的多个视图的高级用户使用。

## 示例

以下示例使用 `DataSet.find()` 在当前集合中搜索 `name` 字段包含 "Bobby" 值的项。即使 `myIterator` 重复值指向 **Bobby** 的记录，但 `student_ds` 的主重复值仍然指向最后一个记录 **Billy**。

要测试此示例，请将 **DataSet** 组件拖到舞台上并为其指定实例名称 `student_ds`。使用“组件”检查器的“架构”选项卡，将 `name`（数据类型：字符串）和 `id`（数据类型：数字）两个属性添加到 **DataSet** 组件中。如果库中没有 `DataBindingClasses` 编译剪辑的副本，请从“类”库（“窗口” > “公用库” > “类”）中拖动该编译剪辑的副本。将以下 **ActionScript** 代码添加到主时间轴的第 1 帧中：

```
import mx.data.to.ValueListIterator;

student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

var myIterator:ValueListIterator = student_ds.getIterator();
myIterator.sortOn(["name"]);
myIterator.find({name:"Bobby"}).id = "999";

trace(student_ds.currentItem.name + " [" + student_ds.currentItem.id + "]");
// Billy [107]

student_ds.addSort("id", ["name", "id"]);
if (student_ds.find({name:"Bobby", id:999})) {
 student_ds.locateById(student_ds.getItemId());
 trace(student_ds.currentItem.name + " [" + student_ds.currentItem.id +
 "]");
 // Bobby [999]
} else {
 trace("We lost Billy!");
}
```

# DataSet.getLength()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.getLength()
```

## 返回

数据集中的项目数。

## 说明

方法：返回数据集中的项目数。

## 示例

下面的示例调用 `getLength()`：

```
//...
var my_ds:mx.data.components.DataSet;
my_ds = _parent.thisShelf.compactDiscs_ds;
trace ("Data set size is: " + my_ds.getLength());
//...
```

# DataSet.hasNext()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.hasNext()
```

## 返回

一个布尔值。

## 说明

方法：如果当前重复值位于其集合视图的结尾处，则返回 `false`；否则返回 `true`。

## 示例

以下示例将遍历当前集合视图中的所有项（从开头位置开始），并对每一项的 `price` 属性执行计算。

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.first();
while (my_ds.hasNext()) {
 my_ds.currentItem.price *= 0.5; // 所有项均为五折！
 my_ds.next();
}
```

```
for (var i in my_ds.items) {
 trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

#### 另请参见

[DataSet.currentItem](#)、[DataSet.first\(\)](#)、[DataSet.next\(\)](#)

## DataSet.hasPrevious()

#### 可用性

Flash Player 7。

#### 版本

Flash MX Professional 2004。

#### 用法

*dataSetInstance*.hasPrevious()

#### 返回

一个布尔值。

#### 说明

方法：如果当前重复值位于其集合视图的开头处，则返回 false；否则返回 true。

#### 示例

以下示例将遍历当前集合视图中的所有项（从最后一项开始），并对每一项的 price 属性执行计算：

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.last();
while (my_ds.hasPrevious()) {
 my_ds.currentItem.price *= 0.5; // 所有项均为五折！
 my_ds.previous();
}

for (var i in my_ds.items) {
 trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

#### 另请参见

[DataSet.currentItem](#)、[DataSet.skip\(\)](#)、[DataSet.previous\(\)](#)

# DataSet.hasSort()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.hasSort(sortName)*

## 参数

*sortName* 一个字符串，它包含使用 `DataSet.addSort()` 创建的排序的名称。

## 返回

一个布尔值。

## 说明

方法；如果 *sortName* 指定的排序存在，则返回 `true`；否则返回 `false`。

## 示例

以下代码将测试名为“**nameSort**”的排序是否存在。如果该排序已存在，则会通过 `DataSet.useSort()` 方法使其成为当前排序。如果具有该名称的排序不存在，则会通过 `DataSet.addSort()` 创建一个排序。若要测试此示例，请将一个 **DataSet** 组件和 **List** 组件拖到舞台上，并分别为它们指定实例名称 `my_ds` 和 `my_list`。使用“组件”检查器的“绑定”选项卡在 **DataSet** 组件的 `dataProvider` 属性和 **List** 组件的 `dataProvider` 属性之间添加一个绑定。使用“组件”检查器的“架构”选项卡在 `my_ds` **DataSet** 的架构中创建两个属性：`name`（数据类型：字符串）和 `years`（数据类型：数字）。将以下代码添加到主时间轴的第一帧中：

```
import mx.data.components.datasetclasses.DataSetIterator;
my_list.labelField = "name";

my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

if (my_ds.hasSort("nameSort")) {
 my_ds.useSort("nameSort");
} else {
 my_ds.addSort("nameSort", ["name"], DataSetIterator.Descending);
}
```



另请参见

[DataSet.addSort\(\)](#)、[DataSet.applyUpdates\(\)](#)、[DataSet.useSort\(\)](#)

## DataSet.isEmpty()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

*dataSetInstance*.isEmpty()

返回

一个布尔值。

说明

方法；如果指定的 **DataSet** 对象不包含任何项目（也就是说，如果 *dataSet.length* == 0），则返回 true。

示例

以下代码将在“删除记录”按钮（未显示）所应用到的 **DataSet** 对象为空时禁用该按钮：

```
if (my_ds.isEmpty()) {
 delete_button.enabled = false;
}
```

另请参见

[DataSet.length](#)

## DataSet.items

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

*dataSetInstance*.items

## 说明

属性：由 *my\_ds* 管理的项目的数组。

## 示例

以下示例会将一个对象数组分配给 **DataSet** 对象的 *items* 属性：

```
var recData:Array = [{id:0, firstName:"Mick", lastName:"Jones"},
 {id:1, firstName:"Joe", lastName:"Strummer"},
 {id:2, firstName:"Paul", lastName:"Simonon"}];
my_ds.items = recData;
```

# DataSet.itemClassName

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.itemClassName*

## 说明

属性：一个字符串，指明在将项目添加到集合中时应创建的类的名称。您指定的类必须实现 **TransferObject** 接口，如下所示。

```
interface mx.data.to.TransferObject {
 function clone():Object;
 function getPropertyData():Object;
 function setPropertyData(propData:Object):Void;
}
```

您也可以在“属性”检查器中设置此属性。

要使指定的类在运行时可用，您必须同时在 **SWF** 文件的代码内的某处对该类进行完全限定引用，如以下代码片断中所示：

```
var myItem:my.package.myItem;
```

如果在 **DataSet.items** 数组已加载之后试图修改此属性的值，则会引发 **DataSetError** 异常。

有关更多信息，请参见第 1135 页的“**TransferObject** 接口”。

# DataSet.iteratorScrolled

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.iteratorScrolled = function (eventObj:Object) {
 // ...
};
dataSetInstance.addEventListener("iteratorScrolled", listenerObject);
```

用法 2:

```
on (iteratorScrolled) {
 // ...
}
```

## 说明

事件：在当前重复值已滚动到集合中的新项目之后立即生成。

事件对象 (*eventObj*) 包含以下属性：

**target** 生成该事件的 **DataSet** 对象。

**type** 字符串 "iteratorScrolled"。

**scrolled** 一个数值，它指定重复值滚动了多少项目；正值指示重复值在集合中向前移动；负值指示它在集合中向后移动。

## 示例

在以下示例中，应用程序的状态栏（未显示）将在当前重复值的位置更改时更新：

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

my_ds.addEventListener("iteratorScrolled", iteratorScrolledListener);

my_ds.first(); // 触发 iteratorScrolled 事件。

function iteratorScrolledListener(evt_obj:Object):Void {
 trace(" 已滚动显示重复值。");
}
```

# DataSet.last()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.last()

## 返回

无。

## 说明

方法：使集合当前视图中的最后一个项目成为当前项目。

## 示例

以下代码（附加到 **Button** 组件）将转到 **DataSet** 集合中的最后一项：

```
function goLast(evt_obj:obj):Void {
 inventory_ds.last();
}
goLast_button.addEventListener("click", goLast);
```

以下示例将遍历当前集合视图中的所有项（从最后一项开始），并对每一项的 price 属性执行计算：

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.last();
while (my_ds.hasPrevious()) {
 my_ds.currentItem.price *= 0.5; // 所有项均为五折！
 my_ds.previous();
}

for (var i in my_ds.items) {
 trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

## 另请参见

[DataSet.first\(\)](#)

# DataSet.length

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.length

## 说明

属性（只读）：指定集合当前视图中的项目数。这个可查看的项目数以当前过滤器和范围的设置为基础。

## 示例

在以下示例中，在对集合中的项进行更改前事件已被禁用，因此 **DataSet** 对象将不会尝试刷新控件而影响性能：

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
 trace("model changed, DataSet now has " + evt_obj.target.length + "
 items");
}
// 禁用数据集的事件。
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while(my_ds.hasPrevious()) {
 my_ds.price *= 0.5; // 所有项均为五折！
 my_ds.previous();
}

trace("After:");
traceItems();

// 告诉数据集现在应该更新控件。
my_ds.enableEvents();

function traceItems(label:String):Void {
```

```
for (var i:Number = 0; i < my_ds.length; i++) {
 trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
}
trace("");
}
```

## DataSet.loadFromSharedObj()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*dataSetInstance*.loadFromSharedObj(*objName*, [*localPath*])

### 参数

*objName* 一个字符串，它指定要检索的共享对象的名称。名称中可以包含正斜杠（如“work/addresses”）。在指定的名称中不允许包含空格和以下字符：

~ % & \ ; : " ' , < > ? #

*localPath* 一个可选字符串参数，它指定创建了共享对象的 SWF 文件的完整或部分路径。此字符串用于确定对象在用户计算机上的存储位置。默认值是 SWF 文件的完整路径。

### 返回

无。

### 说明

方法；加载从共享对象中恢复此 **DataSet** 集合所需的所有相关数据。若要将 **DataSet** 集合保存到共享对象，请使用 [DataSet.saveToSharedObj\(\)](#)。

**DataSet.loadFromSharedObject()** 方法将覆盖此 **DataSet** 集合内可能存在的任何数据或待定更改。请注意，**DataSet** 集合的实例名称用于在指定的共享对象内标识数据。

如果指定的共享对象未找到或从中检索数据时有问题，此方法将引发 **DataSetError** 例外。

## 示例

以下示例将尝试加载与名为 `my_ds` 的数据集关联的共享对象 `webapp/customerInfo`。将在 `try...catch` 代码块内调用该方法。

```
import mx.data.components.datasetclasses.DataSetError;
try {
 my_ds.loadFromSharedObj("webapp/customerInfo");
} catch(e:DataSetError) {
 trace("Unable to load shared object.");
}
```

## 另请参见

[DataSet.saveToSharedObj\(\)](#)

# DataSet.locateById()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.locateById(*id*)

## 参数

*id* 待查找集合中项目的字符串标识符。

## 返回

一个布尔值。

## 说明

方法；将当前重复值放在 **ID** 与 *id* 匹配的集合项目上。如果指定的 **ID** 可与集合中的某个项目匹配，此方法将返回 `true`；否则返回 `false`。

## 示例

以下示例将使用 `DataSet.find()` 在当前集合中搜索 `name` 和 `id` 字段分别包含值 "Bobby" 和 105 的项。如果找到，则会使用 `DataSet.getItemId()` 来获取该项的唯一标识符，并使用 `DataSet.locateById()` 将当前重复值放在该项上。

若要测试此示例，请将一个 **DataSet** 组件拖到舞台上，并为其指定实例名称 `student_ds`。使用“组件”检查器的“架构”选项卡将 `name`（字符串）和 `id`（数字）两个属性添加到 **DataSet** 中。如果库中没有 `DataBindingClasses` 编译剪辑的副本，请从“类”库（“窗口” > “公用库” > “类”）中拖动该编译剪辑的副本。将以下 **ActionScript** 代码添加到主时间轴的第 1 帧中：

```
student_ds.addItem({name:"Barry", id:103});
student_ds.addItem({name:"Bobby", id:105});
student_ds.addItem({name:"Billy", id:107});

trace("Before find() > " + student_ds.currentItem.name); // Billy

var studentID:String;
student_ds.addSort("id", ["name","id"]);
if (student_ds.find(["Bobby", 105])) {
 studentID = student_ds.getItemId();
 student_ds.locateById(studentID);
 trace("After find() > " + student_ds.currentItem.name); // Bobby
} else {
 trace("We lost Billy!");
}
```

## 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.find\(\)](#)、[DataSet.getItemId\(\)](#)

# DataSet.logChanges

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

`dataSetInstance.logChanges`



## 说明

属性：一个布尔值，它指定是 (true) 否 (false) 应在 `DataSet.deltaPacket` 中记录对数据集或其项所做的更改。

如果此属性设置为 true，在集合级别和项目级别执行的操作将被记录。集合级别的更改包括在集合中添加项目或从中删除项目的操作。项目级别的更改包括对项目进行的属性更改，以及通过 `DataSet` 组件对项目进行的方法调用。

## 示例

以下示例将为名为 `userData` 的 `DataSet` 对象禁用记录操作。

```
user_ds.logChanges = false;
```

## 另请参见

[DataSet.deltaPacket](#)

# DataSet.modelChanged

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 说明

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.modelChanged = function (eventObj:Object):Void {
 // ...
};
dataSetInstance.addEventListener("modelChanged", listenerObject);
```

用法 2：

```
on (modelChanged) {
 // ...
}
```

说明

事件：在集合的某些方面已更改（例如，删除项目或将项目添加到集合、项目属性的值发生更改或对集合进行了过滤或排序）时广播。

事件对象 (*eventObj*) 包含以下属性：

*target* 生成该事件的 **DataSet** 对象。

*type* 字符串 "iteratorScrolled"。

*firstItem* 受更改影响的集合中第一个项目的索引（号）。

*lastItem* 受更改影响的集合中最后一个项目的索引（号），如果只有一个项目受影响，则它与 *firstItem* 相等。

*fieldName* 一个字符串，它包含受影响的字段的名称。除非对 **DataSet** 对象的属性进行了更改，否则此属性为 *undefined*。

*eventName* 一个字符串，它描述发生的更改。它可以是以下值之一：

| 字符串值           | 说明                      |
|----------------|-------------------------|
| "addItem"      | 已添加一系列项目。               |
| "filterModel"  | 已过滤模型，并且视图需要刷新（重置滚动位置）。 |
| "removeItems"  | 已删除一系列项目。               |
| "schemaLoaded" | 已声明数据提供程序的字段定义。         |
| "sort"         | 数据已排序。                  |
| "updateAll"    | 整个视图都需要刷新（除滚动位置外）。      |
| "updateColumn" | 数据提供程序内整个字段的定义都需要刷新。    |
| "updateField"  | 项目内的字段已更改，并且需要刷新。       |
| "updateItems"  | 一系列项目需要刷新。              |

示例

在以下示例中，无论何时在数据集内添加或删除项，都会调度 **modelChanged** 事件：

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
 trace("[event =" + evt_obj.eventName + "] the data set now has " +
 evt_obj.target.items.length + " items.");
}
my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});
my_ds.removeItemAt(0);
```

在以下示例中，如果已从集合中删除项并且目标 **DataSet** 对象中没有其它项，则会禁用“删除项”按钮：

```
my_ds.addEventListener("modelChanged", onModelChanged);
function onModelChanged(evt_obj:Object):Void {
 trace("model changed, DataSet now has " + evt_obj.target.items.length + "
 items");
}
// 禁用数据集的事件。
my_ds.disableEvents();

my_ds.addItem({name:"Apples", price:14});
my_ds.addItem({name:"Bananas", price:8});

trace("Before:");
traceItems();

my_ds.last();
while (my_ds.hasPrevious()) {
 my_ds.price *= 0.5; // 所有项均为五折！
 my_ds.previous();
}

trace("After:");
traceItems();

// 告诉数据集现在应该更新控件。
my_ds.enableEvents();

function traceItems(label:String):Void {
 for (var i:Number = 0; i < my_ds.items.length; i++) {
 trace("\t" + my_ds.items[i].name + " - $" + my_ds.items[i].price);
 }
 trace("");
}
```

**另请参见**

[DataSet.isEmpty\(\)](#)

# DataSet.newItem

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.newItem = function (eventObj:Object) {
 // ...
};
dataSetInstance.addEventListener("newItem", listenerObject);
```

用法 2:

```
on (newItem) {
 // ...
}
```

## 说明

事件：在通过 `DataSet.createItem()` 构建新的传输对象时广播。此事件的侦听器可在将项目添加到集合之前对其进行修改。

事件对象 (*eventObj*) 包含以下属性：

*target* 生成该事件的 **DataSet** 对象。

*type* 字符串 "iteratorScrolled"。

*item* 指向所创建项目的引用。

## 示例

以下示例在将新创建的项添加到集合之前对其进行修改：

```
function newItemEvent(evt_obj:Object):Void {
 var employee:Object = evt_obj.item;
 employee.name = "newGuy";
 // 属性数据恰好是 XML。
 employee.zip =
 employee.getPropertyData().firstChild.childNodes[1].attributes.zip;
}
employees_ds.addEventListener("newItem", newItemEvent);
```

# DataSet.next()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.next()

## 返回

无。

## 说明

方法：使集合当前视图中的下一个项目成为当前项目。在当前视图中包含哪些项取决于任何当前的过滤器和范围设置。

## 示例

以下示例将遍历当前集合视图中的所有项（从开头位置开始），并对每一项的 price 属性执行计算：

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});

my_ds.first();
while (my_ds.hasNext()) {
 my_ds.currentItem.price *= 0.5; // 所有项均为五折！
 my_ds.next();
}

for (var i in my_ds.items) {
 trace(my_ds.items[i].name + ": " + my_ds.items[i].price);
}
```

## 另请参见

[DataSet.first\(\)](#)、[DataSet.hasNext\(\)](#)

# DataSet.previous()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.previous()

## 返回

无。

## 说明

方法：使集合当前视图中的上一个项目成为当前项目。在当前视图中包含哪些项取决于任何当前的过滤器和范围设置。

## 示例

以下示例将在数据集内的每一项间循环并跟踪每项的价格：

```
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});
my_ds.last();
while (my_ds.hasPrevious()) {
 trace(my_ds.currentItem.price);
 my_ds.previous();
}
```

以下示例将在当前集合视图中的所有项间循环（从最后一项开始），并对每一项中的字段执行计算：

```
my_ds.last();
while (my_ds.hasPrevious()) {
 my_ds.price *= 0.5; // 所有项均为五折！
 my_ds.previous();
}
```

## 另请参见

[DataSet.first\(\)](#)、[DataSet.hasNext\(\)](#)

# DataSet.properties

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.properties*

## 说明

属性（只读）；返回一个对象，该对象包含此集合内任意传送对象所有公开的属性（字段）。

## 示例

以下示例将显示名为 my\_ds 的 **DataSet** 对象中所有属性的名称：

```
var i:String;
for (i in my_ds.properties) {
 trace("field '" + i + "' has value " + my_ds.properties[i]);
}
```

# DataSet.readOnly

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.readOnly*

## 说明

属性；一个布尔值，它指定此集合是可以修改 (*false*) 还是只读 (*true*)。将此属性设置为 *true* 可防止对集合进行更新。默认值为 *false*。

您也可以在“属性”检查器中设置该属性。

## 示例

以下示例使名为 `my_ds` 的 **DataSet** 对象变为只读，然后尝试更改属于集合中当前项目的属性的值。这将会引发 `DataSetError` 异常。

```
import mx.data.components.datasetclasses.DataSetError;
my_ds.readOnly = true;
try {
 // 这将会引发一个异常。
 my_ds.addItem({name:'Joe'});
} catch (e:DataSetError) {
 // DataSet “my_ds” 不存在按 “name” 指定的排序。
 trace("DataSetError >> " + e.message);
}
```

## 另请参见

[DataSet.currentItem](#)

# DataSet.removeAll()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.removeAll()

## 返回

无。

## 说明

方法；删除 **DataSet** 集合中的所有项目。

## 示例

以下示例将删除 **DataSet** 集合 `contact_ds` 中的所有项：

```
contact_ds.removeAll();
```



# DataSet.removeItem

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.removeItem = function (eventObj:Object):Void {
 // ...
};
dataSetInstance.addEventListener("removeItem", listenerObject);
```

用法 2:

```
on (removeItem) {
 // ...
}
```

## 说明

事件：就在从此集合中删除新项目之前生成。

如果将事件对象的 `result` 属性设置为 `false`，则删除操作将被取消；如果将其设置为 `true`，则允许删除操作。

事件对象 (*eventObj*) 包含以下属性：

`target` 生成该事件的 **DataSet** 对象。

`type` 字符串 "removeItem"。

`item` 指向集合中要删除的项目的引用。

`result` 一个布尔值，它指定是否应删除项目。默认情况下，此值为 `true`。

## 示例

在以下示例中，如果名为 `userHasAdminPrivs()` 的用户定义函数返回 `false`，则 `on(removeItem)` 事件处理函数将取消新项的删除操作；否则，允许执行删除操作：

```
on (removeItem) {
 if (globalObj.userHasAdminPrivs()) {
 // 允许删除项。
 eventObj.result = true;
 } else {
 // 不允许删除项；用户没有管理员权限。
 eventObj.result = false;
 }
}
```

如果名为 `userHasAdminPrivs()` 的用户定义函数返回 `false`，则下面的 `removeItem` 事件处理函数将取消对现有项的删除操作；否则，允许删除项：

```
function userHasAdminPrivs():Boolean {
 return false; // 将该值更改为“true”以允许插入项
}

function removeItemListener(evt_obj:Object):Void {
 if (userHasAdminPrivs()) {
 // Allow the item removal.
 evt_obj.result = true;
 trace("Item removed");
 } else {
 // 不允许删除项；用户没有管理员权限。
 evt_obj.result = false;
 trace("Error, insufficient permissions");
 }
}

my_ds.addEventListener("removeItem", removeItemListener);
my_ds.addItem({name:"item a", price:16});
my_ds.addItem({name:"item b", price:9});
my_ds.removeItemAt(0);
```

## 另请参见

[DataSet.addItem](#)

# DataSet.removeItem()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.removeItem([item])
```

## 参数

*item* 应删除的项目。此参数是可选的。

## 返回

一个布尔值。如果成功删除了项目，则返回 true；否则返回 false。

## 说明

方法：从集合中删除指定项目；或者，如果忽略了 *item* 参数，则删除当前项目。如果 [DataSet.logChanges](#) 为 true，则此操作将被记录到 [DataSet.deltaPacket](#)。

## 示例

以下代码删除当前重复值位置上的项目。若要测试此示例，请将一个 **DataSet** 组件添加到舞台上，并为其指定实例名称 my\_ds。将以下代码添加到主时间轴的第 1 帧中：

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

trace(my_ds.getLength()); // 5
trace(my_ds.currentItem.name); // Frank
my_ds.removeItem();
trace(my_ds.getLength()); // 4
trace(my_ds.currentItem.name); // Michael
```

## 另请参见

[DataSet.deltaPacket](#)、[DataSet.logChanges](#)

# DataSet.removeItemAt()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.removeItemAt(index)*

## 参数

*index* 一个大于或等于 0 的数字。此数字是要删除的项目的索引。

## 返回

一个指示是否删除了项目的布尔值。

## 说明

方法：删除位于指定索引处的项目。已删除索引后面的索引将依次减 1。

此方法触发事件名称为 `removeItems` 的 `modelChanged` 事件。

此外，它还触发 [DataSet.removeItem](#) 事件，该事件包含 `result` 和 `item` 属性。`result` 属性用于确定是否能删除项目（由事件的 `item` 属性引用）。默认情况下，`result` 属性设置为 `true`。如果没有为 `removeItem` 事件指定任何事件侦听器，则默认情况下将会删除项。

通过侦听 `removeItem` 事件并将该事件的 `result` 属性设置为 `false`，事件侦听器可以阻止项被删除，如以下示例所示：

```
function removeItem(evt_obj:Object):Void {
 // 不允许任何人删除 customerId == 0 的项。
 evt_obj.result = (evt_obj.item.customerId != 0);
}
```

## 示例

以下示例将删除数据集内位于第一个位置的项:

```
my_ds.addItem({name:"Milton", years:3});
my_ds.addItem({name:"Mark", years:3});
my_ds.addItem({name:"Sarah", years:1});
my_ds.addItem({name:"Michael", years:2});
my_ds.addItem({name:"Frank", years:2});

trace(my_ds.getLength()); // 5
trace(my_ds.currentItem.name); // Frank
my_ds.removeItemAt(0);
trace(my_ds.getLength()); // 4
trace(my_ds.currentItem.name); // Frank
```

# DataSet.removeRange()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.removeRange()*

## 返回

无。

## 说明

方法: 删除由 [DataSet.setRange\(\)](#) 为当前重复值指定的当前端点设置。

## 示例

```
my_ds.addSort("name_id", ["name", "id"]);
my_ds.setRange(["Bobby", 105],["Cathy", 110]);
while (my_ds.hasNext()) {
 my_ds.gradeLevel = "5"; // 更改此范围中的所有级别。
 my_ds.next();
}
my_ds.removeRange();
my_ds.removeSort("name_id");
```

## 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.hasNext\(\)](#)、[DataSet.next\(\)](#)、  
[DataSet.removeSort\(\)](#)、[DataSet.setRange\(\)](#)

# DataSet.removeSort()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.removeSort(sortName)
```

## 参数

*sortName* 一个字符串，它指定要删除的排序的名称。

## 返回

无。

## 说明

方法：从此 **DataSet** 对象中删除指定排序（如果排序存在）。如果指定的排序不存在，此方法将引发 **DataSetError** 例外。

## 示例

以下示例将在 **DataSet** 组件中创建一系列项并修改每一项的 **gradeLevel** 属性。若要测试此示例，请将一个 **DataSet** 组件拖到舞台上，并为其指定实例名称 **my\_ds**。在选中 **DataSet** 组件的情况下，使用“组件”检查器的“架构”选项卡在 **DataSet** 组件的架构中创建三个新属性。将新属性命名为 **name**、**id** 和 **gradeLevel**，并分别将其数据类型指定为“String”、“Number”和“Number”。从“类”公用库（“窗口” > “公用库” > “类”）添加 **DataBindingClasses** 编译剪辑的副本，并将以下 **ActionScript** 代码添加到主时间轴的第一帧中：

```
my_ds.addItem({name:"Billy", id:104, gradeLevel:4});
my_ds.addItem({name:"Bobby", id:105, gradeLevel:4});
my_ds.addItem({name:"Carrie", id:106, gradeLevel:4});
my_ds.addItem({name:"Cathy", id:110, gradeLevel:4});
my_ds.addItem({name:"Mally", id:112, gradeLevel:3});

my_ds.addSort("name_id", ["name", "id"]);
my_ds.setRange(["Bobby", 105], ["Cathy", 110]);
while (my_ds.hasNext()) {
 my_ds.gradeLevel = "5"; // 更改此范围中的所有级别。
 my_ds.next();
}
my_ds.removeRange();
my_ds.removeSort("name_id");
```

```
for (var i=0; i<my_ds.length; i++) {
 trace(my_ds.items[i].name + " > " + my_ds.items[i].gradeLevel);
}
```

#### 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.hasNext\(\)](#)、[DataSet.next\(\)](#)、  
[DataSet.removeRange\(\)](#)、[DataSet.setRange\(\)](#)

## DataSet.resolveDelta

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.resolveDelta = function (eventObj:Object):Void {
 // ...
};
dataSetInstance.addEventListener("resolveDelta", listenerObject);
```

用法 2:

```
on (resolveDelta) {
 // ...
}
```

### 说明

事件：在将增量数据包分配给 [DataSet.deltaPacket](#) 时广播，它的事务 ID 与之前从 [DataSet](#) 对象中检索到的增量数据包的事务 ID 匹配，并且具有与该增量数据包所包含的任何增量或 [DeltaItem](#) 对象关联的消息。

此事件使您能够在尝试应用先前提交的更改时协调从服务器返回的任何错误。通常，您可以使用此事件来显示包含冲突值的“协调对话框”，从而允许用户对数据进行适当的修改以便能够重新发送。

事件对象 (*eventObj*) 包含以下属性:

*target* 生成该事件的 **DataSet** 对象。

*type* 字符串 "resolveDelta"。

*data* 增量对象及关联 **DeltaItem** 对象的数组, 它具有长度不为零的消息。

## 示例

以下示例显示一个名为 *reconcileForm* 的表单 (未显示), 并对该表单对象调用一个方法 (*setReconcileData()*), 该方法使用户能够协调服务器返回的任何冲突值:

```
import mx.data.components.datasetclasses.*;
my_ds.addEventListener("resolveDelta", onResolveDelta);
function onResolveDelta(eventObj:Object) {
 reconcileForm.visible = true;
 reconcileForm.setReconcileData(eventObj.data);
}
// 在 reconcileForm 代码中
function setReconcileData(data:Array):Void {
 var di:DeltaItem;
 var ops:Array = ["property", "method"];
 var cl:Array;
 // change list
 var msg:String;
 for (var i = 0; i<data.length; i++) {
 cl = data[i].getChangeList();
 for (var j = 0; j<cl.length; j++) {
 di = cl[j];
 msg = di.message;
 if (msg.length>0) {
 trace(" 发生下列问题: '"+msg+"' 。此问题是在对 '"+di.name+"' 的当前服务器
值 ["+di.curValue+"], 进行 '"+ops[di.kind]+"' 修改时发生的, 所发送的值为
["+di.newValue+"] 。请修改!");
 }
 }
 }
}
```



# DataSet.saveToSharedObj()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.saveToSharedObj(objName, [localPath])
```

## 参数

*objName* 一个字符串，它指定要创建的共享对象的名称。名称中可以包含正斜杠（如“work/addresses”）。在指定的名称中不允许包含空格和以下字符：

~ % & \ ; : " ' , < > ? #

*localPath* 一个可选字符串参数，它指定创建了共享对象的 SWF 文件的完整路径或部分路径。此字符串用于确定对象在用户计算机上的存储位置。默认值是 SWF 文件的完整路径。

## 返回

无。

## 说明

方法：保存将此 **DataSet** 集合恢复到共享对象所需的所有相关数据。它使用户能够在与来源数据断开连接的情况下工作（如果来源数据是网络资源）。此方法将覆盖此 **DataSet** 集合的指定共享对象内可能存在的任何数据。若要从共享对象恢复 **DataSet** 集合，请使用 [DataSet.loadFromSharedObj\(\)](#)。请注意，**DataSet** 集合的实例名称用于在指定的共享对象内标识数据。

如果无法创建共享对象或在将数据刷新到其中时有问题，此方法将引发 `DataSetError` 例外。

## 示例

以下示例将在 `try..catch` 块内调用 `saveToSharedObj()`，如果在将数据保存到共享对象时出现问题，此示例将显示一个错误。

```
import mx.data.components.datasetclasses.DataSetError;
try {
 my_ds.saveToSharedObj("webapp/customerInfo");
} catch(e:DataSetError) {
 trace("Unable to create shared object");
}
```

## 另请参见

[DataSet.loadFromSharedObj\(\)](#)

# DataSet.schema

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.schema*

## 说明

属性：为此 **DataSet** 对象提供以 XML 方式表示的架构。分配给此属性的 XML 必须具有以下格式：

```
<?xml version="1.0"?>
<properties>
 <property name="propertyName">
 <type name="dataType" />
 <encoder name="dataType">
 <options>
 <dataFormat>format options</dataFormat>
 </options>
 </encoder>
 <kind name="dataKind">
 </kind>
 </property>
 <property> ... </property>
 ...
</properties>
```

如果指定的 XML 不符合上面的格式，则会引发 `DataSetError` 异常。

## 示例

以下示例将数据集 `my_ds` 的架构设置为新的 XML 对象，此对象包含设置相应格式的 XML：

```
my_ds.schema = new XML("<properties><property name='billable'> ..etc.. </properties>");
```

# DataSet.selectedIndex

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.selectedIndex

## 说明

属性：指定集合内的选定索引。可以将此属性绑定到 **DataGrid** 或 **List** 组件中的选定项目，反之亦然。若要查看演示此操作的完整示例，请参见第 311 页的“创建具有 **DataSet** 组件的应用程序”。

## 示例

以下示例将 **DataSet** 对象 (user\_ds) 的选定索引设置为 **DataGrid** 组件 (user\_dg) 中的选定索引。

```
user_ds.selectedIndex = user_dg.selectedIndex;
```

# DataSet.setIterator()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.setIterator(*iterator*)

## 参数

*iterator* 调用 **DataSet.getIterator()** 时返回的重复值对象。

## 返回

无。

## 说明

方法：将指定的重复值分配给此 **DataSet** 对象，并使其成为当前重复值。指定的重复值必须来自之前对它所分配到的 **DataSet** 对象上的 [DataSet.getIterator\(\)](#) 的调用；否则，将会引发 **DataSetError** 异常。

## 示例

```
import mx.data.to.ValueListIterator;
myIterator:ValueListIterator = my_ds.getIterator();
myIterator.sortOn(["name"]);
my_ds.setIterator(myIterator);
```

## 另请参见

[DataSet.getIterator\(\)](#)

# DataSet.setRange()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance.setRange(startValues, endValues)*

## 参数

*startValues* 范围内第一个传输对象的属性的关键值数组。

*endValues* 范围内最后一个传输对象的属性的关键值数组。

## 返回

无。

## 说明

方法：设置当前重复值的端点。这些端点定义一个范围，重复值将在其中进行运算。只有在通过 [DataSet.addSort\(\)](#) 为当前重复值设置了有效的排序时，此方法才有效。

如果需要分组的值，则为当前重复值设置一个范围比使用过滤函数更有效（请参见 [DataSet.filterFunc\(\)](#)）。

## 示例

以下示例将选择一定范围的学生，并将这些学生的姓名显示到“输出”面板：

```
my_ds.addItem({name:"Billy", id:104, gradeLevel:4});
my_ds.addItem({name:"Bobby", id:105, gradeLevel:4});
my_ds.addItem({name:"Carrie", id:106, gradeLevel:4});
my_ds.addItem({name:"Cathy", id:110, gradeLevel:4});
my_ds.addItem({name:"Mally", id:112, gradeLevel:3});
my_ds.addSort("name_id",["name", "id"]);
my_ds.setRange(["Bobby", 105],["Cathy", 110]);
while (my_ds.hasNext()) {
 trace(my_ds.name); // Bobby..Cathy
 my_ds.next();
}
```

## 另请参见

[DataSet.addSort\(\)](#)、[DataSet.hasNext\(\)](#)、[DataSet.next\(\)](#)、[DataSet.removeRange\(\)](#)、[DataSet.removeSort\(\)](#)

# DataSet.skip()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*dataSetInstance*.skip(*offset*)

## 参数

*offset* 一个整数，它指定重复值位置要移动的记录数。

## 返回

无。

## 说明

方法；将当前重复值的位置在集合内向前或向后移动由 *offset* 指定的距离。如果 *offset* 值为正，则重复值的位置向前移动；如果值为负，则向后移动。

如果指定的 **offset** 超出了集合的开头（或结尾），重复值将被定位在集合的开头（或结尾）。

## 示例

以下示例将当前重复值定位在集合中的第一项，然后移动到倒数第二项，并对属于该项的字段执行计算：

```
my_ds.addItem({name:"Billy", id:104, gradeLevel:4});
my_ds.addItem({name:"Carrie", id:106, gradeLevel:4});
my_ds.addItem({name:"Mally", id:112, gradeLevel:3});
my_ds.addItem({name:"Cathy", id:110, gradeLevel:4});
my_ds.addItem({name:"Bobby", id:105, gradeLevel:4});
my_ds.first();
var itemsToSkip:Number = 3;
trace(my_ds.currentItem.name); // Billy
my_ds.skip(itemsToSkip);
trace(my_ds.currentItem.name); // Mally
```

# DataSet.useSort()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
dataSetInstance.useSort(sortName, order)
```

## 参数

*sortName* 一个字符串，它包含要使用的排序的名称。

*order* 一个整数值，它指示排序的排序顺序；该值必须为 `DataSetIterator.Ascending` 或 `DataSetIterator.Descending`。

## 返回

无。

## 说明

方法：将当前重复值的排序切换到 *sortName* 指定的排序（如果存在）。如果指定的排序不存在，则会引发 `DataSetError` 例外。

若要创建排序，请使用 `DataSet.addSort()`。

## 示例

以下示例使用 `DataSet.hasSort()` 来确定名为 "customer" 的排序是否存在。如果存在, 代码将调用 `DataSet.useSort()` 使 "customer" 成为当前排序。否则, 代码将使用 `DataSet.addSort()` 创建一个具有该名称的排序。

```
if (my_ds.hasSort("customer")) {
 my_ds.useSort("customer");
} else {
 my_ds.addSort("customer", ["customer"], DataSetIterator.Descending);
}
```

## 另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.hasSort\(\)](#)





# DateChooser 组件（仅限 Flash Professional）

# 14

**DateChooser** 组件是一个允许用户选择日期的日历。它包含一些按钮，这些按钮允许用户在月份之间来回滚动并单击某个日期将其选中。可以设置指示月份和日名称、星期的第一天和任何禁用日期以及加亮显示当前日期的参数。

每个 **DateChooser** 实例的实时预览都反映创作过程中“属性”检查器或“组件”检查器指示的值。

## 使用 DateChooser 组件 （仅限 Flash Professional）

**DateChooser** 可用于任何您想让用户选择日期的场合。例如，您可以将 **DateChooser** 组件用于酒店预订系统中，其中某些日期是可选择的，其它日期则是禁用的。您也可以在用户选择日期时显示当前事件（如表演或会议）的应用程序中使用 **DateChooser** 组件。

### DateChooser 参数

在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中，可以为每个 **DateChooser** 组件实例设置以下创作参数：

**dayNames** 设置一星期中各天的名称。该值是一个数组，其默认值为 ["S", "M", "T", "W", "T", "F", "S"]。

**disabledDays** 指示一星期中禁用的各天。该参数是一个数组，并且最多具有七个值。默认值为 []（空数组）。

**firstDayOfWeek** 指示一星期中的哪一天（其值为 0-6, 0 是 **dayNames** 数组的第一个元素）显示在日期选择器的第一列中。此属性更改“日”列的显示顺序。

**monthNames** 设置在日历的标题行中显示的月份名称。该值是一个数组，其默认值为 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

**showToday** 指示是否要加亮显示今天的日期。默认值为 true。

在“组件”检查器（“窗口” > “组件检查器”）中，可以为每个 **DateChooser** 组件实例设置以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 **true**。

**visible** 是一个布尔值，它指示对象是 (**true**) 否 (**false**) 可见。默认值为 **true**。

提醒

**minHeight** 和 **minWidth** 属性由内部的大小调整例程使用。这两个属性在 **UIObject** 中定义，并可以根据需要被不同的组件覆盖。在为应用程序创建自定义布局管理器时，可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不具有明显效果。

您可以编写 **ActionScript**，使用其属性、方法和事件来控制 **DateChooser** 组件的这些和其它选项。有关更多信息，请参见第386页的“**DateChooser** 类（仅限 **Flash Professional**）”。

## 创建具有 **DateChooser** 组件的应用程序

以下过程解释了如何在创作时将 **DateChooser** 组件添加到应用程序。在此示例中，日期选择器允许用户从航空公司预订系统中选择一个日期。10 月 15 日之前的所有日期必须被禁用。同时，12 月中的某个范围必须被禁用以创建一个假日关闭期，并且星期一也必须被禁用。

### 创建具有 **DateChooser** 组件的应用程序：

1. 在“组件”面板中双击 **DateChooser** 组件，将其添加到舞台。
2. 在“属性”检查器中，输入实例名称 **flightCalendar**。
3. 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以设置可选日期的范围：

```
flightCalendar.selectableRange = {rangeStart:new Date(2003, 9, 15),
rangeEnd:new Date(2003, 11, 31)}
```

这段代码将一个值分配给 **ActionScript** 对象中的 **selectableRange** 属性，该属性包含两个 **Date** 对象，而这两个对象分别具有变量名称 **rangeStart** 和 **rangeEnd**。这样，此代码就定义了用户可以在其中选择日期的范围的上限和下限。

4. 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以设置假日禁用日期的范围：

```
flightCalendar.disabledRanges = [{rangeStart: new Date(2003, 11, 15),
rangeEnd: new Date(2003, 11, 26)}];
```

5. 在“动作”面板中，在时间轴的第一帧上输入以下代码，以禁用星期一：

```
flightCalendar.disabledDays=[1];
```

6. 选择“控制” > “测试影片”。

使用 **ActionScript** 创建 **DateChooser** 组件实例：

- 1. 将 **DateChooser** 组件从“组件”面板拖到当前文档的库中。
- 2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：  

```
this.createClassObject(mx.controls.DateChooser, "my_dc", 1);
```

此脚本使用 [第 1254 页](#) 的“**UIObject.createClassObject()**”方法创建 **DateChooser** 实例，然后调整网格大小并定位网格。
- 3. 选择“控制” > “测试影片”。

## 自定义 **DateChooser** 组件 (仅限 **Flash Professional**)

在创作过程中和运行时，可以在水平和垂直方向上改变 **DateChooser** 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，使用 `setSize()` 方法（请参见 [UIObject.setSize\(\)](#)）。

### 对 **DateChooser** 组件使用样式

您可以设置样式属性来更改 **DateChooser** 实例的外观。如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

**DateChooser** 组件支持下列样式：

样式	主题	说明
themeColor	光晕	用于变换图象和所选日期的发亮的颜色。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
backgroundColor	光晕和范例	背景色。默认值为 <code>OxEFEBEF</code> （浅灰）。
borderColor	光晕和范例	边框颜色。默认值为 <code>Ox919999</code> 。
		<b>DateChooser</b> 组件使用纯色单像素线作为其边框。此边框不能通过样式或外观进行修改。
headerColor	光晕和范例	组件标题的背景色。默认颜色为白色。
rollOverColor	光晕和范例	滑过的日期的背景颜色。“光晕”主题的默认值为 <code>OxE3FFD6</code> （亮绿），“范例”主题的默认值为 <code>OxAAAAAA</code> （浅灰）。
selectionColor	光晕和范例	选定日期的背景颜色。“光晕”主题的默认值为 <code>OxCDFFC1</code> （浅绿），“范例”主题的默认值为 <code>OxEEEEEE</code> （极浅灰）。

样式	主题	说明
todayColor	光晕和范例	今天的日期的背景颜色。默认值为 0x666666（深灰）。
color	光晕和范例	文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。
disabledColor	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。
embedFonts	光晕和范例	一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。
fontFamily	光晕和范例	文本的字体名称。默认值为 "_sans"。
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式："normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细："none" 或 "bold"。默认值为 "none"。在调用 setStyle() 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 getStyle() 的调用将返回 "none"。
textDecoration	光晕和范例	文本修饰："none" 或 "underline"。默认值为 "none"。

**DateChooser** 组件使用四类文本显示月份名称、星期几、今天的日期以及常规日期。**DateChooser** 组件本身设置的文本样式属性对常规日期文本进行控制，并为其它文本提供默认值。要设置特定类别文本的文本样式，请使用下面的类级别样式声明。

声明名称	说明
HeaderDateText	月份名称。
WeekDayStyle	星期几。
TodayStyle	今天的日期。

下面的示例演示如何将月份名称和星期几设置为深红色。

```
_global.styles.HeaderDateText.setStyle("color", 0x660000);
_global.styles.WeekDayStyle.setStyle("color", 0x660000);
```

# 对 DateChooser 组件使用外观

DateChooser 组件使用外观来表示下个月和上个月按钮，以及今天指示符。若要在创作时设置 DateChooser 组件的外观，请在其中一个主题的 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/DateChooser Assets/States 文件夹中的外观元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。

在此组件中，只有月份滚动按钮才能被动态地设置外观。DateChooser 组件使用以下外观属性：

属性	说明
backMonthButtonUpSymbolName	上个月按钮弹起状态。默认值为 backMonthUp。
backMonthButtonDownSymbolName	上个月按钮按下状态。默认值为 backMonthDown。
backMonthButtonDisabledSymbolName	上个月按钮禁用状态。默认值为 backMonthDisabled。
fwdMonthButtonUpSymbolName	下个月按钮弹起状态。默认值为 fwdMonthUp。
fwdMonthButtonDownSymbolName	下个月按钮按下状态。默认值为 fwdMonthDown。
fwdMonthButtonDisabledSymbolName	下个月按钮禁用状态。默认值为 fwdMonthDisabled。

按钮元件完全照原样使用，而不应用颜色或调整大小。大小由元件在创作时确定。

## 创建 DateChooser 外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 HaloTheme.flc 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在操作系统上的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板上，展开 Flash UI Components 2/Themes/MMDefault 文件夹，并将 DateChooser Assets 文件夹拖动到您的文档的库中。
4. 在文档的库中展开 DateChooser Assets/States 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如，打开 backMonthDown 元件。
6. 按需要自定义元件。  
例如，将箭头的色调更改为红色。
7. 对所有要自定义的元件重复步骤 5-6。  
例如，更改前进箭头按下元件的色调以匹配后退箭头。
8. 单击“返回”按钮返回主时间轴。
9. 将 DateChooser 组件拖动到舞台上。

10. 选择 “控制” > “测试影片”。

提醒

DateChooser Assets/States 文件夹还包含一个 Day Skins 文件夹，其中有一个外观元素 cal\_todayIndicator。此元素可在创作期间修改，以便对今天指示符进行自定义。但是，它不能在特定的 DateChooser 实例上动态更改为使用其它元件。此外，cal\_todayIndicator 元件必须是单纯色图形，因为 DateChooser 组件会将 todayColor 颜色整个地应用到图形。图形可以有裁剪，但请记住，今天的日期的默认文本颜色是白色，DateChooser 的默认背景是白色，因此如果裁减去今天指示符的外观元素的中间部分，将会使得今天的日期无法被识别，除非同时也更改背景色或今天文本颜色。

## DateChooser 类（仅限 Flash Professional）

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > DateChooser

ActionScript 类名称 mx.controls.DateChooser

DateChooser 类的属性允许访问选定的日期以及显示的月份和年份。您也可以设置日和月的名称，指明禁用的日期和可选的日期，设置一星期中的第一天，以及指明当前日期是否应加亮显示。

使用 ActionScript 设置 DateChooser 类的属性会覆盖在 “属性” 检查器或 “组件” 检查器中设置的同名参数。

每个组件类都有一个 version 属性，该属性是一个类属性。类属性只能用于该类本身。version 属性会返回一个字符串，该字符串指示组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.controls.DateChooser.version);
```

提醒

代码 trace(myDC.version); 返回 undefined。

# DateChooser 类的方法摘要

没有 DateChooser 类专用的方法。

## 从 UIObject 类继承的方法

下表列出了 DateChooser 类从 UIObject 类继承的方法。从 DateChooser 对象调用这些方法时，请使用 *dateChooserInstance.methodName* 的形式。

方法	说明
<a href="#">UIObject.createClassObject()</a>	创建指定类的对象。
<a href="#">UIObject.createObject()</a>	创建对象的子对象。
<a href="#">UIObject.destroyObject()</a>	破坏组件实例。
<a href="#">UIObject.doLater()</a>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<a href="#">UIObject.getStyle()</a>	从样式声明或对象获取样式属性。
<a href="#">UIObject.invalidate()</a>	标记对象使其在到达下一个帧间隔时进行重绘。
<a href="#">UIObject.move()</a>	将对象移动到要求的位置。
<a href="#">UIObject.redraw()</a>	迫使对象有效以便能在当前帧中绘制。
<a href="#">UIObject.setSize()</a>	将对象调整为所要求的大小。
<a href="#">UIObject.setSkin()</a>	设置对象的外观。
<a href="#">UIObject.setStyle()</a>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 DateChooser 类从 UIComponent 类继承的方法。从 DateChooser 对象调用这些方法时，请使用 *dateChooserInstance.methodName* 的形式。

方法	说明
<a href="#">UIComponent.getFocus()</a>	返回对具有焦点的对象的引用。
<a href="#">UIComponent.setFocus()</a>	将焦点设置到组件实例中。

# DateChooser 类的属性摘要

下表列出了供 DateChooser 类专用的属性。

属性	说明
<code>DateChooser.dayNames</code>	一个数组，指明一星期中各天的名称。
<code>DateChooser.disabledDays</code>	一个数组，指示为日期选择器中所有适用日期禁用的一星期中的各天。
<code>DateChooser.disabledRanges</code>	禁用日期的范围或单个禁用的日期。
<code>DateChooser.displayedMonth</code>	一个数字，指示 monthNames 数组中要在日期选择器中显示的元素。
<code>DateChooser.displayedYear</code>	一个数字，指明要显示的年份。
<code>DateChooser.firstDayOfWeek</code>	一个数字，指示 dayNames 数组中要在日期选择器的第一列中显示的元素。
<code>DateChooser.monthNames</code>	一个字符串数组，指明月份名称。
<code>DateChooser.selectableRange</code>	单个可选日期或可选日期的范围。
<code>DateChooser.selectedDate</code>	一个 Date 对象，指明可选日期。
<code>DateChooser.showToday</code>	一个布尔值，指明是否加亮显示当前日期。

## 从 UIObject 类继承的属性

下表列出了 DateChooser 类从 UIObject 类继承的属性。从 DateChooser 对象访问这些属性时，请使用 `dateChooserInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。



属性	说明
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 `DateChooser` 类从 `UIComponent` 类继承的属性。从 `DateChooser` 对象访问这些属性时，请使用 `dateChooserInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指示组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指示文档中组件的 Tab 键顺序。

## DateChooser 类的事件摘要

下表列出了 `DateChooser` 类专用的事件。

事件	说明
<code>DateChooser.change</code>	在选择一个日期时广播。
<code>DateChooser.scroll</code>	在单击月份按钮时广播。

## 从 UIObject 类继承的事件

下表列出了 `DateChooser` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 DateChooser 类从 UIComponent 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## DateChooser.change

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 // ...
};
dateChooserInstance.addEventListener("change", listenerObject);
```

用法 2:

```
on (change) {
 // ...
}
```

### 说明

事件：当选择日期时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*dataChooserInstance*) 调度一个事件 (在本示例中为 `change`)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作“处理函数”) 处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 `DateChooser` 实例。在附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码（附加到日期选择器 `my_dc`）将“`_level0.my_dc`”发送到“输出”面板：

```
on (change) {
 trace(this);
}
```

## 示例

此示例是在时间轴的某一帧上编写的，它会在名为 `my_dc` 的 `DateChooser` 实例发生更改时向“输出”面板发送一条消息。第一行代码创建一个名为 `form` 的侦听器对象。第二行代码为该侦听器对象的 `change` 事件定义一个函数。该函数内部是一条 `trace()` 语句，它使用自动传递到该函数的事件对象（在本例中是 `eventObj`）来生成消息。事件对象的 `target` 属性是生成该事件的组件（在本例中是 `my_dc`）。

```
// 创建侦听器对象。
var dcListener:Object = new Object();
dcListener.change = function(evt_obj:Object) {
 var thisDate:Date = evt_obj.target.selectedDate;
 trace("date selected: " + thisDate);
};
```

```
// 向日期选择器中添加侦听器对象。
my_dc.addEventListener("change", dcListener);
```

# DateChooser.dayNames

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance*.dayNames

## 说明

属性；包含一星期中各天的名称的数组。星期日是第一天（索引位置为 0），其它各天的名称按顺序跟在后面。默认值为 ["S", "M", "T", "W", "T", "F", "S"]。

## 示例

以下示例更改星期几的值：

```
my_dc.dayNames = new Array("Su", "Mo", "Tu", "We", "Th", "Fr", "Sa");
```

# DateChooser.disabledDays

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance*.disabledDays

## 说明

属性；指明一星期中被禁用的各天的数组。一个月中属于指定日期范围内的所有日期都被禁用。此数组的元素可以具有介于 0（星期日）和 6（星期六）之间的值。默认值为 []（空数组）。

## 示例

以下示例将禁用星期日和星期六，以使用户只能选择每星期工作日：

```
my_dc.disabledDays = [0, 6];
```

# DateChooser.disabledRanges

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance.disabledRanges*

## 说明

属性；禁用某一天或一个范围中的各天。此属性是对象数组。该数组中的每个对象都必须是指定要禁用的某一天的 **Date** 对象，或者是包含 `rangeStart` 和 `/` 或 `rangeEnd` 属性（这两个属性的值都必须是 **Date** 对象）的对象。`rangeStart` 和 `rangeEnd` 属性描述日期范围的界限。如果这两个属性中的任何一个属性被省略，则范围在该方向上无限制。

`disabledRanges` 的默认值为 `undefined`。

在为 `disabledRanges` 属性定义日期时应指定一个完整日期。例如，应指定 `new Date(2003,6,24)`，而不是 `new Date()`。如果未指定完整日期，则 **Date** 对象会返回当前的日期和时间。如果未指定时间，则该时间返回为 00:00:00。

## 示例

以下示例定义一个数组，该数组具有 `rangeStart` 和 `rangeEnd` **Date** 对象，这两个对象将禁用 5 月 7 日和 6 月 7 日之间的日期：

```
my_dc.disabledRanges = [{rangeStart: new Date(2003, 4, 7), rangeEnd: new
 Date(2003, 5, 7)}];
```

以下示例禁用 11 月 7 日之后的所有日期：

```
my_dc.disabledRanges = [{rangeStart: new Date(2003, 10, 7)}];
```

以下示例禁用 10 月 7 日之前的所有日期：

```
my_dc.disabledRanges = [{rangeEnd: new Date(2002, 9, 7)}];
```

以下示例仅禁用 12 月 7 日：

```
my_dc.disabledRanges = [new Date(2003, 11, 7)];
```

以下示例禁用 4 月 7 日和 4 月 21 日：

```
my_dc.disabledRanges = [new Date(2003, 3, 7), new Date(2003, 3, 21)];
```

# DateChooser.displayedMonth

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance.displayedMonth*

## 说明

属性；指明显示哪一月份的数字。该数字指示 `monthNames` 数组中的一个元素，**0** 是第一个月份。默认值是当前日期的月份。

## 示例

以下示例将显示的月份设置为 **December**（12 月）：

```
my_dc.displayedMonth = 11;
```

## 另请参见

[DateChooser.displayedYear](#)

# DateChooser.displayedYear

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance.displayedYear*

## 说明

属性；指示显示哪一年份的四位数数字。默认值为当前年份。

## 示例

以下示例将显示的年份设置为 2010:

```
my_dc.displayedYear = 2010;
```

## 另请参见

[DateChooser.displayedMonth](#)

# DateChooser.firstDayOfWeek

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
dateChooserInstance.firstDayOfWeek
```

## 说明

属性：一个数值，指示一星期中的哪一天（0-6，0 是 `dayNames` 数组的第一个元素）显示在 **DateChooser** 组件的第一列中。更改该属性将更改日列的顺序，但对 `dayNames` 属性的顺序没有影响。默认值为 0（星期日）。

## 示例

以下示例将一星期的第一天设置为 **Monday**（星期一）：

```
// 将日历中一星期的第一天设置为 Monday（星期一）。
my_dc.firstDayOfWeek = 1;
```

```
// 禁用 0（星期一）。即使 Monday（星期一）现在是 DateChooser 中的第一天，但 Sunday（星期日）的数组索引仍然为 0。
my_dc.disabledDays = [0];
```

## 另请参见

[DateChooser.dayNames](#)

# DateChooser.monthNames

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance*.monthNames

## 说明

属性：一个字符串数组，在 **DateChooser** 组件的顶部指示月份名称。默认值为 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

## 示例

以下示例设置 my\_dc 实例的月份名称：

```
my_dc.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
 "Sept", "Oct", "Nov", "Dec"];
```

# DateChooser.scroll

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
 //...
}
dateChooserInstance.addEventListener("scroll", listenerObject)
```

用法 2：

```
on (scroll) {
 //...
}
```



## 说明

事件：在单击月份按钮时广播到所有注册的侦听器。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*myDC*) 调度一个事件（在本例中为 *scroll*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。**scroll** 事件的事件对象具有另一个属性 *detail*，该属性可以具有以下值之一：*nextMonth*、*previousMonth*、*nextYear*、*previousYear*。

最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **DateChooser** 实例。在附加到组件的 `on()` 处理函数内部使用的关键字 *this* 是指该组件实例。例如，以下代码（附加到日期选择器 *myDC*）将“\_level0.myDC”发送到“输出”面板：

```
on (scroll) {
 trace(this);
}
```

## 示例

此示例是在时间轴的某一帧上编写的，当在名为 *my\_dc* 的 **DateChooser** 实例上单击月份按钮时，它将向“输出”面板发送一条消息。第一行代码创建一个名为 *form* 的侦听器对象。第二行代码为该侦听器对象的 *scroll* 事件定义一个函数。该函数内部是一条 `trace()` 语句，它使用自动传递到该函数的事件对象（在本例中是 *evt\_obj*）来生成消息。

```
// 创建侦听器对象。
var dcListener:Object = new Object();
dcListener.scroll = function(evt_obj:Object) {
 trace(evt_obj.detail);
};
```

```
// 向日期选择器中添加侦听器对象。
my_dc.addEventListener("scroll", dcListener);
```

# DateChooser.selectableRange

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance.selectableRange*

## 说明

属性；设置可选的某一天或一定范围的可选日期。用户不能滚动到可选择范围以外。此属性的值是由名为 `rangeStart` 和 `rangeEnd` 的两个 **Date** 对象组成的对象。`rangeStart` 和 `rangeEnd` 属性指定可选日期范围的界限。如果只定义 `rangeStart`，则启用 `rangeStart` 后的所有日期。如果只定义 `rangeEnd`，则启用 `rangeEnd` 前的所有日期。默认值为 `undefined`。

如果您想要只启用某一天，则可以使用单个 **Date** 对象作为 `selectableRange` 的值。

定义日期时，请指定完整日期，如 `new Date(2003,6,24)` 而不是 `new Date()`。如果未指定完整日期，则 **Date** 对象会返回当前的日期和时间。如果未指定时间，则该时间返回为 `00:00:00`。

`DateChooser.selectedDate` 的值在超出可选范围时将被设置为 `undefined`。

如果当前月份超出可选范围，`DateChooser.displayedMonth` 和 `DateChooser.displayedYear` 的值将设置为可选范围中最接近的最后一个月份。例如，如果当前显示的月份是 8 月，并且可选范围是从 2003 年 6 月到 2003 年 7 月，则显示的月份将更改为 2003 年 7 月。

## 示例

以下示例将可选范围定义为 5 月 7 日和 6 月 7 日之间的日期（包括这两个日期）：

```
my_dc.selectableRange = {rangeStart: new Date(2001, 4, 7), rangeEnd: new Date(2003, 5, 7)};
```

以下示例将可选范围定义为 5 月 7 日之后的日期（包括 5 月 7 日）：

```
my_dc.selectableRange = {rangeStart: new Date(2005, 4, 7)};
```

以下示例将可选范围定义为 6 月 7 日之前的日期（包括 6 月 7 日）：

```
my_dc.selectableRange = {rangeEnd: new Date(2005, 5, 7)};
```

以下示例将可选日期定义为仅限 6 月 7 日：

```
my_dc.selectableRange = new Date(2005, 5, 7);
```

# DateChooser.selectedDate

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance.selectedDate*

## 说明

属性：一个 **Date** 对象，如果该值在 `selectableRange` 属性值的范围内，则指示所选日期。默认值为 `undefined`。

不能在禁用的范围内、可选择的范围外或禁用的日期上设置 `selectedDate` 属性。如果此属性设置为以上这些日期之一，则值为 `undefined`。

## 示例

以下示例将所选日期设置为 6 月 7 日：

```
my_dc.selectedDate = new Date(2005, 5, 7);
```

# DateChooser.showToday

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateChooserInstance.showToday*

## 说明

属性：一个布尔值，它确定是否加亮显示当前日期。默认值为 `true`。

## 示例

以下示例关闭对今天日期的加亮显示：

```
my_dc.showToday = false;
```



## DateField 组件（仅限 Flash Professional）

**DateField** 组件是一个不可选择的文本字段，它显示右边带有日历图标的日期。如果未选定日期，则该文本字段为空白，并且当前日期的月份显示在日期选择器中。当用户在日期字段边框内的任意位置单击时，将会弹出一个日期选择器，并显示选定日期所在月份内的日期。当日期选择器打开时，用户可以使用月份滚动按钮在月份和年份之间来回滚动，并选择一个日期。如果选定某个日期，则会关闭日期选择器，并将所选日期输入到日期字段中。



如果用户选择两次相同的日期，则会清除日期字段。若要防止用户意外取消选定需要的日期，请参见第 423 页的 `DateField.selectedDate` 示例。

**DateField** 组件的实时预览不会反映创作时“属性”检查器或“组件”检查器指示的值，因为该组件是一个弹出组件，在创作时是不可见的。

## 使用 DateField 组件（仅限 Flash Professional）

**DateField** 组件可用于您想让用户选择日期的任何场合。例如，您可以将 **DateField** 组件用于酒店预订系统中，其中某些日期是可选择的，其它日期则是禁用的。您也可以在用户选择日期时显示当前事件（如表演或会议）的应用程序中使用 **DateField** 组件。

### DateField 参数

您可以在“属性”检查器或“组件”检查器中，为每个 **DateField** 组件实例设置以下创作参数：

**dayNames** 设置一星期中各天的名称。该值是一个数组，其默认值为 ["S", "M", "T", "W", "T", "F", "S"]。

**disabledDays** 指示一星期中禁用的各天。该参数是一个数组，并且最多具有七个值。默认值为 []（空数组）。

**firstDayOfWeek** 指示一星期中的哪一天（其值为 0-6, 0 是 dayNames 数组的第一个元素）显示在日期选择器的第一列中。此属性更改“日”列的显示顺序。

默认值为 0，即代表星期日的“S”。

**monthNames** 设置在日历的标题行中显示的月份名称。该值是一个数组，其默认值为 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

**showToday** 指示是否要加亮显示今天的日期。默认值为 true。

您可以编写 **ActionScript**，以便使用其属性、方法和事件来控制 **DateField** 组件的这些和其它选项。有关更多信息，请参见第 406 页的“**DateField 类（仅限 Flash Professional）**”。

## 创建具有 DateField 组件的应用程序

以下过程解释了如何在创作时将 **DateField** 组件添加到应用程序。在此示例中，**DateField** 组件使用户能够从航空公司预订系统中选择一个日期。当前日期之前的所有日期必须被禁用。12 月中一个 15 天的范围也必须被禁用，以便创建一个假日关闭期。同时，由于某些航班在星期一不可用，因此必须为这些航班禁用所有星期一。

### 创建具有 DateField 组件的应用程序：

1. 在“组件”面板中双击 **DateField** 组件，将其添加到舞台。
2. 在“属性”检查器中，输入实例名称 **flightCalendar**。
3. 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以设置可选日期的范围：

```
flightCalendar.selectableRange = {rangeStart:new Date(2001, 9, 1),
rangeEnd:new Date(2003, 11, 1)};
```

这段代码将一个值分配给 **ActionScript** 对象中的 **selectableRange** 属性，该属性包含两个 **Date** 对象，而这两个对象分别具有变量名称 **rangeStart** 和 **rangeEnd**。这样，此代码就定义了用户可以在其内选择日期的范围的上限和下限。

4. 在“动作”面板中，在时间轴的第一帧上输入以下代码以设置禁用日期的范围，一个范围处于 12 月，另一个代表当前日期之前的所有日期：

```
flightCalendar.disabledRanges = [{rangeStart: new Date(2003, 11, 15),
rangeEnd: new Date(2003, 11, 31)}, {rangeEnd: new Date(2003, 6, 16)}];
```

5. 在“动作”面板中，在时间轴的第一帧上输入以下代码，以禁用星期一：

```
flightCalendar.disabledDays=[1];
```

6. “控制” > “测试影片”。

**使用 ActionScript 创建 DateField 组件实例：**

- 1. 将 DateField 组件从“组件”面板拖到当前文档的库中。  
此操作将组件添加到库中，但不会在应用程序中显示。
- 2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：  
`this.createClassObject(mx.controls.DateField, "my_df", 1);`  
此脚本使用 `UIObject.createClassObject()` 方法创建 DateField 实例。
- 3. 选择“控制” > “测试影片”。

# 自定义 DateField 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平方向上改变 DateField 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，则使用 `setSize()` 方法（请参见 `UIObject.setSize()`）。设置宽度不会改变 DateField 组件内日期选择器的尺寸。但是，您可以使用 `pullDown` 属性来访问 DateChooser 组件并设置其尺寸。

## 对 DateField 组件使用样式

您可以设置样式属性来更改日期字段实例的外观。如果样式属性的名称以“Color”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

DateField 组件支持下列样式：

样式	主题	说明
themeColor	光晕	用于变换图象和所选日期的发亮的颜色。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"
backgroundColor	光晕和范例	背景色。默认值为 0xEFEBEF（浅灰）。
borderColor	光晕和范例	边框颜色。默认值为 0x919999。  DateField 组件的下拉列表使用纯色单像素线作为其边框。此边框不能通过样式或外观进行修改。
headerColor	光晕和范例	下拉标题的背景颜色。默认颜色为白色。

样式	主题	说明
rolloverColor	光晕和范例	滑过的日期的背景颜色。“光晕”主题的默认值为 <code>OxE3FFD6</code> （亮绿），“范例”主题的默认值为 <code>OxAAAAAA</code> （浅灰）。
selectionColor	光晕和范例	选定日期的背景颜色。“光晕”主题的默认值为 <code>OxCDFFC1</code> （浅绿），“范例”主题的默认值为 <code>OxEEEEEE</code> （极浅灰）。
todayColor	光晕和范例	今天的日期的背景颜色。默认值为 <code>Ox666666</code> （深灰）。
color	光晕和范例	文本颜色。“光晕”主题的默认值为 <code>Ox0B333C</code> ，“范例”主题的默认值为空白。
disabledColor	光晕和范例	组件禁用时的文本颜色。默认值为 <code>Ox848384</code> （深灰）。
embedFonts	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
fontFamily	光晕和范例	文本的字体名称。默认值为 <code>"_sans"</code> 。
fontSize	光晕和范例	字体的磅值。默认值为 <code>10</code> 。
fontStyle	光晕和范例	字体样式： <code>"normal"</code> 或 <code>"italic"</code> 。默认值为 <code>"normal"</code> 。
fontWeight	光晕和范例	字体粗细： <code>"none"</code> 或 <code>"bold"</code> 。默认值为 <code>"none"</code> 。在调用 <code>setStyle()</code> 期间，所有组件还可以接受值 <code>"normal"</code> 来代替 <code>"none"</code> ，但随后对 <code>getStyle()</code> 的调用将返回 <code>"none"</code> 。
textDecoration	光晕和范例	文本修饰： <code>"none"</code> 或 <code>"underline"</code> 。默认值为 <code>"none"</code> 。

**DateField** 组件使用四类文本分别显示月份名称、星期几、今天的日期以及常规日期。对 **DateField** 组件本身设置的文本样式属性控制常规日期文本和在折叠状态下显示的文本，并为其它文本提供默认值。要设置特定类别文本的文本样式，请使用下面的类级别样式声明。

声明名称	说明
<code>HeaderDateText</code>	月份名称。
<code>WeekDayStyle</code>	星期几。
<code>TodayStyle</code>	今天的日期。

下面的示例演示如何将月份名称和星期几设置为深红色。

```
_global.styles.HeaderDateText.setStyle("color", 0x660000);
_global.styles.WeekDayStyle.setStyle("color", 0x660000);
```



# 对 DateField 组件使用外观

DateField 组件使用外观表示弹出图标的可视状态、文本输入周围的边框的 RectBorder 实例、弹出图标的 DateChooser 实例。若要在创作时设置弹出图标的外观，请在其中一个主题 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/DateField Assets/States 文件夹中的外观元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。有关设置 RectBorder 和 DateChooser 实例外观的信息，请参见第 985 页的“RectBorder 类”和第 385 页的“对 DateChooser 组件使用外观”。

除了以上提及的子组件所使用的外观以外，DateField 组件还使用下面的外观属性来动态地设置弹出图标的外观：

属性	说明
openDateUp	弹出图标的弹起状态。
openDateDown	弹出图标的按下状态。
openDateOver	弹出图标的滑过状态。
openDateDisabled	弹出图标的禁用状态。

## 要创建 DateField 外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 HaloTheme.fla 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在操作系统上的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板上，展开 Flash UI Components 2/Themes/MMDefault 文件夹，并将 DateField Assets 文件夹拖动到您的文档的库中。
4. 在文档库中展开 DateField Assets 文件夹。
5. 确保为“在第一帧导出”选择了 DateFieldAssets 元件。
6. 在文档库中展开 DateField Assets/States 文件夹。
7. 打开要自定义的元件以进行编辑。  
例如，打开 openIconUp 元件。
8. 按需要自定义元件。  
例如，在日历图像上画一个向下的箭头。
9. 对所有要自定义的元件重复步骤 7-8。  
例如，在所有元件上画一个向下的箭头。
10. 单击“返回”按钮返回主时间轴。
11. 将 DateField 组件拖动到舞台上。
12. 选择“控制”>“测试影片”。

# DateField 类（仅限 Flash Professional）

继承 MovieClip > UIObject 类 > UIComponent 类 > ComboBase > DateField

ActionScript 类名称 mx.controls.DateField

**DateField** 类的属性使您能够访问选定的日期以及显示的月份和年份。您也可以设置日和月的名称，指明禁用的日期和可选的日期，设置一星期中的第一天，以及指明当前日期是否应加亮显示。

使用 **ActionScript** 设置 **DateField** 类的属性会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

每个组件类都有一个 `version` 属性，该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.controls.DateField.version);
```

剪辑

代码 `trace(myDateFieldInstance.version);` 返回 `undefined`。

## DateField 类的方法摘要

下表列出了 **DateField** 类的方法。

方法	说明
<code>DateField.close()</code>	关闭弹出 <code>DateChooser</code> 子组件。
<code>DateField.open()</code>	打开弹出 <code>DateChooser</code> 子组件。

## 从 UIObject 类继承的方法

下表列出了 **DateField** 类从 **UIObject** 类继承的方法。从 **DateField** 对象调用这些方法时，请使用 `dateFieldInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。

方法	说明
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 `DateField` 类从 `UIComponent` 类继承的方法。从 `DateField` 对象调用这些方法时，请使用 `dateFieldInstance.methodName` 的形式。

方法	说明
<code>UIComponent.setFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## DateField 类的属性摘要

下表列出了 `DateField` 类的属性。

属性	说明
<code>DateField.dateFormatter</code>	对要在文本字段中显示的日期进行格式化的函数。
<code>DateField.dayNames</code>	一个数组，指明一星期中各天的名称。
<code>DateField.disabledDays</code>	指示一星期中被禁用的各天的数组。
<code>DateField.disabledRanges</code>	禁用日期的范围或单个禁用的日期。
<code>DateField.displayedMonth</code>	一个数字，它指示要显示 <code>monthNames</code> 数组中的哪个元素。
<code>DateField.displayedYear</code>	一个数字，指明要显示的年份。
<code>DateField.firstDayOfWeek</code>	一个数字，它指示 <code>dayNames</code> 数组中要在 <code>DateField</code> 组件的第一列中显示的元素。
<code>DateField.monthNames</code>	一个字符串数组，指明月份名称。
<code>DateField.pullDown</code>	指向 <code>DateChooser</code> 子组件的引用。该属性为只读。
<code>DateField.selectableRange</code>	单个可选日期或可选日期的范围。
<code>DateField.selectedDate</code>	一个 <code>Date</code> 对象，指明可选日期。
<code>DateField.showToday</code>	一个布尔值，指明是否加亮显示当前日期。

## 从 UIObject 类继承的属性

下表列出了 **DateField** 类从 **UIObject** 类继承的属性。从 **DateField** 对象访问这些属性时，请使用 `dateFieldInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 **DateField** 类从 **UIComponent** 类继承的属性。从 **DateField** 对象访问这些属性时，请使用 `dateFieldInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## DateField 类的事件摘要

下表列出了 **DateField** 类的事件。

事件	说明
<code>DateField.change</code>	在选择一个日期时广播。
<code>DateField.close</code>	在 <code>DateChooser</code> 子组件关闭时广播。
<code>DateField.open</code>	在 <code>DateChooser</code> 子组件打开时广播。
<code>DateField.scroll</code>	在单击月份按钮时广播。

## 从 UIObject 类继承的事件

下表列出了 `DateField` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 `DateField` 类从 `UIComponent` 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

# DateField.change

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 // ...
};
dateFieldInstance.addEventListener("change", listenerObject);
```

用法 2:

```
on (change) {
 // ...
}
```

## 说明

事件：当选择日期时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*dateFieldInstance*) 调度一个事件（在本示例中为 *change*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 *on()* 处理函数，并且必须直接附加到一个 **DateField** 实例。附加到组件的 *on()* 处理函数内部使用的关键字 *this* 是指该组件实例。例如，以下代码附加到日期字段 *my\_df* 上，它将 “\_level0.my\_df” 发送到“输出”面板：

```
on (change) {
 trace(this);
}
```

## 示例

下面的示例是在时间轴上的某一帧上编写的，它会在一个名为 *my\_df* 的日期字段发生更改时向“输出”面板发送一条消息。第一行代码创建一个名为 *dfListener* 的侦听器对象。第二行代码为侦听器对象的 *change* 事件定义一个函数。该函数内部是一个 *trace()* 语句，它使用自动传递到该函数的事件对象（在本例中是 *evt\_obj*）来生成消息。事件对象的 *target* 属性是生成该事件的组件，在本例中是 *my\_df*。从事件对象的 *target* 属性中可以访问 [DateField.selectedDate](#) 属性。最后一行从 *my\_df* 调用 [EventDispatcher.addEventListener\(\)](#)，并将 *change* 事件和 *dfListener* 侦听器对象作为参数传递给该方法。

```
// 创建侦听器对象。
var dfListener:Object = new Object();
dfListener.change = function(evt_obj:Object){
 var thisDate:Date = evt_obj.target.selectedDate;
 trace("date selected: " + thisDate);
}

// 向 DateField 中添加侦听器对象。
my_df.addEventListener("change", dfListener);
```

# DateField.close()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance.close()*

## 返回

无。

## 说明

方法；关闭弹出菜单。

## 示例

单击 my\_btn 按钮时，以下代码会关闭 my\_df 日期字段实例的日期选择器弹出菜单：

```
// 创建侦听器对象。
var btnListener:Object = new Object();
btnListener.click = function() {
 my_df.close();
};

// 添加按钮侦听器。
my_btn.addEventListener("click", btnListener);
```

# DateField.close

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.close = function(eventObject:Object) {
 // ...
```

```
};
dateFieldInstance.addEventListener("close", listenerObject);
```

用法 2:

```
on (close) {
 // ...
}
```

## 说明

事件：当用户在图标外单击或选择一个日期后 **DateChooser** 子组件关闭时向所有注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*dateFieldInstance*) 调度一个事件（在本示例中为 *close*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 *on()* 处理函数，并且必须直接附加到一个 **DateField** 实例。附加到组件的 *on()* 处理函数内部使用的关键字 *this* 是指该组件实例。例如，以下代码附加到日期字段 *my\_df* 上，它将 “\_level0.my\_df” 发送到“输出”面板：

```
on (close) {
 trace(this);
}
```

## 示例

下面的示例是在时间轴上的某一帧上编写的，它会在 *my\_df* 内的日期选择器关闭时向“输出”面板发送一条消息。第一行代码创建一个名为 *dfListener* 的侦听器对象。第二行代码为侦听器对象的 *close* 事件定义一个函数。该函数内部是一条 *trace()* 语句，它使用自动传递到该函数的事件对象（在本例中是 *evt\_obj*）来生成消息。事件对象的 *target* 属性是生成该事件的组件，在本例中是 *my\_df*。从事件对象的 *target* 属性中可以访问 *selectedDate* 属性。最后一行从 *my\_df* 调用 [EventDispatcher.addEventListener\(\)](#)，并将 *close* 事件和 *dfListener* 侦听器对象作为参数传递给该方法。

```
// 创建侦听器对象。
var dfListener:Object = new Object();
dfListener.close = function(evt_obj:Object){
 trace("PullDown Closed" + evt_obj.target.selectedDate);
}
// 向 DateField 中添加侦听器对象。
my_df.addEventListener("close", dfListener);
```



# DateField.dateFormatter

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.dateFormatter

## 说明

属性：对要在文本字段中显示的日期进行格式化的函数。该函数必须接收 **Date** 对象作为参数，并以要显示的格式返回字符串。

## 示例

以下范例将设置函数以返回要显示的日期的格式：

```
my_df.dateFormatter = function(d:Date){
 return d.getFullYear()+" / "+(d.getMonth()+1)+" / "+d.getDate();
};
```

# DateField.dayNames

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.dayNames

## 说明

属性：包含一星期中各天的名称的数组。星期日是第一天（索引位置为 0），其它各天的名称按顺序跟在后面。默认值为 ["S", "M", "T", "W", "T", "F", "S"]。

### 示例

以下示例将一星期的第 5 天（星期四）的值从 “T” 更改为 “R”：

```
my_df.dayNames[4] = "R";
```

以下示例将相应地更改所有天的值：

```
my_df.dayNames = new Array("Su", "Mo", "Tu", "We", "Th", "Fr", "Sa");
```

## DateField.disabledDays

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
dateFieldInstance.disabledDays
```

### 说明

属性：指明一星期中被禁用的各天的数组。一个月中属于指定日期范围内的所有日期都被禁用。此数组的元素可以具有介于 0（星期日）和 6（星期六）之间的值。默认值为 []（空数组）。

### 示例

以下示例将禁用星期日和星期六，以使用户只能选择每星期工作日：

```
my_df.disabledDays = [0, 6];
```

# DateField.disabledRanges

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance.disabledRanges*

## 说明

属性；禁用某一天或一个范围中的各天。此属性是对象数组。该数组中的每个对象都必须是指定要禁用的某一天的 **Date** 对象，或者是包含 `rangeStart` 和 `/` 或 `rangeEnd` 属性（这两个属性的值都必须是 **Date** 对象）的对象。`rangeStart` 和 `rangeEnd` 属性描述日期范围的界限。如果这两个属性中的任何一个属性被省略，则范围在该方向上无限制。

`disabledRanges` 的默认值为 `undefined`。

当您为 `disabledRanges` 属性定义日期时，请指定完整的日期，如 `new Date(2003,6,24)` 而不是 `new Date()`。如果没有指定完整日期，则该时间返回为 `00:00:00`。

## 示例

以下示例定义一个数组，该数组具有 `rangeStart` 和 `rangeEnd` **Date** 对象，这两个对象将禁用 5 月 7 日和 6 月 7 日之间的日期：

```
my_df.disabledRanges = [{rangeStart: new Date(2003, 4, 7), rangeEnd: new
 Date(2003, 5, 7)}];
```

以下示例禁用 11 月 7 日之后的所有日期：

```
my_df.disabledRanges = [{rangeStart: new Date(2003, 10, 7)}];
```

以下示例禁用 10 月 7 日之前的所有日期：

```
my_df.disabledRanges = [{rangeEnd: new Date(2002, 9, 7)}];
```

以下示例仅禁用 12 月 7 日：

```
my_df.disabledRanges = [new Date(2003, 11, 7)];
```

以下示例禁用 12 月 7 日和 12 月 20 日：

```
my_df.disabledRanges = [new Date(2003, 11, 7), new Date(2003, 11, 20)];
```

# DateField.displayedMonth

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.displayedMonth

## 说明

属性：指明显示哪一月份的数字。该数字指示 `monthNames` 数组中的元素，**0** 是第一个月份。默认值是当前日期的月份。

## 示例

以下示例将显示的月份设置为 **December**（12 月）：

```
my_df.displayedMonth = 11;
```

## 另请参见

[DateField.displayedYear](#)

# DateField.displayedYear

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.displayedYear

## 说明

属性：指明显示哪一年份的数字。默认值为当前年份。

## 示例

以下示例将显示的年份设置为 **2010**：

```
my_df.displayedYear = 2010;
```

## 另请参见

[DateField.displayedMonth](#)

# DateField.firstDayOfWeek

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.firstDayOfWeek

## 说明

属性：一个数字，指示一星期中的哪一天（0-6，0 是 dayNames 数组的第一个元素）显示在 **DateField** 组件的第一列中。更改该属性将更改日列的顺序，但对 dayNames 属性的顺序没有影响。默认值为 0（星期日）。

## 示例

以下示例将一星期的第一天设置为 **Monday**（星期一）：

```
my_df.firstDayOfWeek = 1;
```

## 另请参见

[DateField.dayNames](#)

# DateField.monthNames

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.monthNames

## 说明

属性：一个字符串数组，在 **DateField** 组件的顶部指明月份名称。默认值为 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

## 示例

以下示例设置 my\_df 实例的月份名称：

```
my_df.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
 "Sept", "Oct", "Nov", "Dec"];
```

# DateField.open()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance.open()*

## 返回

无。

## 说明

方法；打开弹出 **DateChooser** 子组件。

## 示例

单击 my\_btn 按钮时，以下代码会打开 my\_df 日期字段实例的日期选择器弹出菜单：

```
// 创建侦听器对象。
var btnListener:Object = new Object();
btnListener.click = function() {
 my_df.open();
};

// 添加按钮侦听器。
my_btn.addEventListener("click", btnListener);
```

# DateField.open

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.open = function(eventObject:Object) {
 // ...
```

```
};
dateFieldInstance.addEventListener("open", listenerObject);
```

用法 2:

```
on (open) {
 // ...
}
```

## 说明

事件：当用户单击图标后 **DateChooser** 子组件打开时，向所有注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*dateFieldInstance*) 调度一个事件（在本示例中为 *open*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 *on()* 处理函数，并且必须直接附加到一个 **DateField** 实例。附加到组件的 *on()* 处理函数内部使用的关键字 *this* 是指该组件实例。例如，以下代码附加到日期字段 *my\_df* 上，它将 “\_level0.my\_df” 发送到 “输出” 面板：

```
on (open) {
 trace(this);
}
```

## 示例

下面的示例是在时间轴上的某一帧上编写的，它会在一个名为 *my\_df* 的日期字段打开时向 “输出” 面板发送一条消息。此消息以当前月份的索引号结束。第一行代码创建一个名为 *dfListener* 的侦听器对象。第二行代码为侦听器对象的 *open* 事件定义一个函数。该函数内部是一条 *trace()* 语句，它使用自动传递到该函数的事件对象（在本例中是 *evt\_obj*）来生成消息。事件对象的 *target* 属性是生成该事件的组件，在本例中是 *my\_df*。从事件对象的 *target* 属性中可以访问 [DateField.selectedDate](#) 属性。最后一行从 *my\_df* 调用 [EventDispatcher.addEventListener\(\)](#)，并将 *open* 事件和 *dfListener* 侦听器对象作为参数传递给该方法。

```
// 创建侦听器对象。
var dfListener:Object = new Object();
dfListener.open = function(evt_obj:Object){
 trace("PullDown Opened" + evt_obj.target.displayedMonth);
}
// 向 DateField 中添加侦听器对象。
my_df.addEventListener("open", dfListener);
```

# DateField.pullDown

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.pullDown

## 说明

属性（只读）：对 **DateField** 组件包含的 **DateChooser** 组件的引用。当用户单击 **DateField** 组件时，**DateChooser** 子组件将被实例化。但是，如果在用户单击该组件之前引用了 `pullDown` 属性，则 **DateChooser** 将被实例化并随后被隐藏。

## 示例

以下示例将 **DateChooser** 子组件的可见性设置为 `false`，然后将 **DateChooser** 子组件的大小设置为 300 像素高和 300 像素宽：

```
my_df.pullDown._visible = false;
my_df.pullDown.setSize(300, 300);
```

# DateField.scroll

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
 // ...
};
dateFieldInstance.addEventListener("scroll", listenerObject);
```



用法 2:

```
on (scroll) {
 // ...
}
```

## 说明

事件：在单击月份按钮时广播到所有注册的侦听器。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*dateFieldInstance*) 调度一个事件（在本示例中为 `scroll`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`scroll` 事件的事件对象具有另一个属性 `detail`，该属性可以具有以下值之一：`nextMonth`、`previousMonth`、`nextYear`、`previousYear`。

最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 `DateField` 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到日期字段 `my_df` 上，它将 “`_level0.my_df`” 发送到 “输出” 面板：

```
on (scroll) {
 trace(this);
}
```

## 示例

下面的示例是在时间轴的某一帧上编写的，当用户在名为 `my_df` 的 `DateField` 实例上单击月份按钮时，它将向 “输出” 面板发送一条消息。第一行代码创建一个名为 `dfListener` 的侦听器对象。第二行代码为侦听器对象的 `scroll` 事件定义一个函数。该函数内部是一条 `trace()` 语句，它使用自动传递到该函数的事件对象（在本例中是 `evt_obj`）来生成消息。事件对象的 `target` 属性是生成该事件的组件，在本例中是 `my_df`。最后一行从 `my_df` 调用 `EventDispatcher.addEventListener()`，并将 `scroll` 事件和 `dfListener` 侦听器对象作为参数传递给该方法。

```
// 创建侦听器对象。
var dfListener:Object = new Object();
dfListener.scroll = function(evt_obj:Object) {
 trace(evt_obj.detail);
};
```

```
// 向 DateField 中添加侦听器对象。
my_df.addEventListener("scroll", dfListener);
```

# DateField.selectableRange

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance*.selectableRange

## 说明

属性：设置可选的某一天或一定范围的可选日期。此属性的值是由名为 `rangeStart` 和 `rangeEnd` 的两个 **Date** 对象组成的对象。`rangeStart` 和 `rangeEnd` 属性指定可选日期范围的界限。如果只定义 `rangeStart`，则启用 `rangeStart` 后的所有日期。如果只定义 `rangeEnd`，则启用 `rangeEnd` 前的所有日期。默认值为 `undefined`。

如果您想要只启用某一天，则可以使用一个 **Date** 对象作为 `selectableRange` 的值。

定义日期时，请指定完整日期，如 `new Date(2003,6,24)` 而不是 `new Date()`。如果没有指定完整日期，则该时间返回为 `00:00:00`。

`DateField.selectedDate` 的值在超出可选范围时将被设置为 `undefined`。

如果当前月份超出可选范围，`DateField.displayedMonth` 和 `DateField.displayedYear` 的值将设置为可选范围中最接近的最后一个月份。例如，如果当前显示的月份是 8 月，并且可选范围是从 2003 年 6 月到 2003 年 7 月，则显示的月份将更改为 2003 年 7 月。

## 示例

以下示例将可选范围定义为 5 月 7 日和 6 月 7 日之间的日期（包括这两个日期）：

```
my_df.selectableRange = {rangeStart: new Date(2001, 4, 7), rangeEnd: new Date(2003, 5, 7)};
```

以下示例将可选范围定义为 5 月 7 日之后的日期（包括 5 月 7 日）：

```
my_df.selectableRange = {rangeStart: new Date(2003, 4, 7)};
```

以下示例将可选范围定义为 6 月 7 日之前的日期（包括 6 月 7 日）：

```
my_df.selectableRange = {rangeEnd: new Date(2003, 5, 7)};
```

以下示例将可选日期定义为仅限 6 月 7 日：

```
my_df.selectableRange = new Date(2003, 5, 7);
```

# DateField.selectedDate

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance.selectedDate*

## 说明

属性：一个 **Date** 对象，如果该值在 `selectableRange` 属性值的范围内，则指示所选日期。  
默认值为 `undefined`。

## 示例

以下示例将所选日期设置为 6 月 7 日：

```
my_df.selectedDate = new Date(2003, 5, 7);
```

以下示例在舞台上使用名为 `my_df` 的 **DateField** 实例来显示如何禁用已选定的日期（另外，用户可以通过再次单击来清除日期字段条目）：

```
function dfListener(evt_obj:Object):Void {
 my_df.disabledRanges = [my_df.selectedDate];
}
my_df.addEventListener("change", dfListener);
```

# DateField.showToday

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*dateFieldInstance.showToday*

## 说明

属性：一个布尔值，它确定是否加亮显示当前日期。默认值为 `true`。

## 示例

以下示例关闭对今天日期的加亮显示：

```
my_df.showToday = false;
```



继承 Object > Delegate

ActionScript 类名称 mx.utils.Delegate

**Delegate** 类允许您在特定范围内运行函数。提供了这个类之后，您就可以将同一个事件发送给两个不同的函数（请参见《使用组件》中的“将事件委托给函数”），并可以在包含类的范围内调用函数。

当将一个函数作为参数传递给 `EventDispatcher.addEventListener()` 时，将会在广播器组件实例的范围内调用该函数，而不是在声明该函数的对象的范围内调用（请参见《使用组件》中的“委托函数的范围”）。您可以通过调用 `Delegate.create()` 在声明对象的范围内调用该函数。

## Delegate 类的方法摘要

下表列出了 **Delegate** 类的方法。

方法	说明
<code>Delegate.create()</code>	一个静态方法，它允许您在特定范围内运行函数。

# Delegate.create()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`Delegate.create(scopeObject, function)`

## 参数

*scopeObject* 对对象的引用。这是运行函数的范围。

*function* 对函数的引用。

## 说明

**Method (static)**：允许您将事件委托给特定的范围和函数。应使用以下语法：

```
import mx.utils.Delegate;
compInstance.addEventListener("eventName", Delegate.create(scopeObject,
 function));
```

*scopeObject* 参数指定调用指定函数的范围。

## 示例

若要查看 `Delegate.create()` 的示例，请参见《使用组件》中的“委托事件”。

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# DeltaItem 类（仅限 Flash Professional）

**ActionScript 类名称** mx.data.components.datasetclasses.DeltaItem

**DeltaItem** 类提供了关于对传输对象所执行的各个操作的信息。它指示是直接对传输对象的属性进行更改，还是通过方法调用进行更改。它还提供传输对象属性的原始状态。例如，如果增量数据包的源是数据集，则 **DeltaItem** 对象包含关于编辑的任何字段的信息。

除上述内容以外，**DeltaItem** 对象还能包含服务器响应信息，例如当前值和消息。

当访问增量数据包中的更改时使用 **DeltaItem** 类。若要访问这些更改，请使用 `DeltaPacket.getIterator()`，它将返回增量的重复值。每个增量包含零个或多个 **DeltaItem** 实例，您可以通过 `Delta.getItemByName()` 或 `Delta.getChangeList()` 访问这些实例。

## DeltaItem 类的属性摘要

下表列出了 **DeltaItem** 类的属性。

属性	说明
<code>DeltaItem.argList</code>	如果通过方法调用进行更改，则这是传递到方法的值的数组。该属性为只读。
<code>DeltaItem.curValue</code>	如果是对属性进行更改，则这是该属性的当前服务器值。该属性为只读。
<code>DeltaItem.delta</code>	<b>DeltaItem</b> 对象的关联增量。该属性为只读。
<code>DeltaItem.kind</code>	更改的类型。
<code>DeltaItem.message</code>	与 <b>DeltaItem</b> 对象关联的服务器消息。
<code>DeltaItem.name</code>	更改过的属性或方法的名称。该属性为只读。
<code>DeltaItem.newValue</code>	如果是对属性进行了更改，则这是该属性的新值。该属性为只读。
<code>DeltaItem.oldValue</code>	如果是对属性进行了更改，则这是该属性的旧值。该属性为只读。

# DeltaItem.argList

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem*.argList

## 说明

属性（只读）；传递到更改方法的值的数组。此属性仅在更改的类型是 `DeltaItem.Method` 时适用。

# DeltaItem.curValue

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem*.curValue

## 说明

属性（只读）；一个对象，它包含传输对象的服务器副本上的当前属性值。此属性仅在更改的类型是 `DeltaItem.Property` 时适用，而且仅在已从服务器返回并正应用到用户分辨率数据集的增量中才相关。



# DeltaItem.delta

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem.delta*

## 说明

属性（只读）：与 **DeltaItem** 对象关联的增量。创建 **DeltaItem** 对象时，它与某个增量关联，并将其自身添加到该增量的更改列表中。此属性提供对此项目所属的增量的引用。

# DeltaItem.kind

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem.kind*

## 说明

属性；一个数字，它指示更改的类型。使用以下常数评估此属性：

- **DeltaItem.Property** 对传输对象的属性进行了更改。
- **DeltaItem.Method** 通过方法调用对传输对象进行了更改。

# DeltaItem.message

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem.message*

## 说明

属性；一个字符串，它包含与此 **DeltaItem** 对象关联的服务器消息。这可以是有关在增量数据包中尝试的属性或方法调用更改的任何消息。此消息通常只在已从服务器返回并正应用到分辨率 **DataSet** 的增量中才相关。

# DeltaItem.name

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem.name*

## 说明

属性（只读）；一个字符串，它包含已更改属性的名称（如果更改的类型是 **DeltaItem.Property**）或进行该更改的方法的名称（如果更改的类型是 **DeltaItem.Method**）。

# DeltaItem.newValue

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem*.newValue

## 说明

属性（只读）；一个对象，它包含属性的新值。此属性仅在更改的类型是 DeltaItem.Property 时适用。

# DeltaItem.oldValue

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*deltaitem*.oldValue

## 说明

属性（只读）；一个对象，它包含属性的旧值。此属性仅在更改的类型是 DeltaItem.Property 时适用。



# Delta 接口（仅限 Flash Professional）

**ActionScript 接口名称** mx.data.components.datasetclasses.Delta

**Delta** 接口提供对传输对象、集合以及传输对象级别更改的访问。通过此接口，您可以访问传输对象中的新值和以前的值。例如，如果从某个数据集获取了增量数据包，则每个增量将表示一个添加、编辑或删除的行。

**Delta** 接口还提供对关联服务器端进程返回消息的访问。有关客户端 - 服务器交互的更多信息，请参见第 971 页的“[RDBMSResolver 组件（仅限 Flash Professional）](#)”。

在将更改发送到服务器之前使用 **Delta** 接口检查增量数据包，并检查从服务器端处理返回的消息。

## Delta 接口的方法摘要

下表列出了 **Delta** 接口的方法。

方法	说明
<a href="#">Delta.addDeltaItem()</a>	添加指定的 <b>DeltaItem</b> 实例。
<a href="#">Delta.getChangeList()</a>	返回对当前项目所做更改的数组。
<a href="#">Delta.getDeltaPacket()</a>	返回包含增量的增量数据包。
<a href="#">Delta.getId()</a>	返回当前项目在 <b>DeltaPacket</b> 集合内的唯一 ID。
<a href="#">Delta.getItemByName()</a>	返回指定的 <b>DeltaItem</b> 对象。
<a href="#">Delta.getMessage()</a>	返回与当前项目关联的消息。
<a href="#">Delta.getOperation()</a>	返回对原始集合内的当前项目所执行的操作。
<a href="#">Delta.getSource()</a>	返回被更改的传输对象。

# Delta.addDeltaItem()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
delta.addDeltaItem(deltaitem)
```

## 参数

*deltaitem* 要添加到此增量的 **DeltaItem** 实例。

## 返回

无。

## 说明

方法；添加指定的 **DeltaItem** 实例。如果指定的 **DeltaItem** 实例已存在，则会被此方法替换。

## 示例

以下示例调用 addDeltaItem() 方法：

```
//...
var d:Delta = new DeltaImpl("ID1345678", curItem, DeltaPacketConsts.Added,
 "", false);
d.addDeltaItem(new DeltaItem(DeltaItem.Property, "ID", {oldValue:15,
 newValue:16}));
//...
```

# Delta.getChangeList()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
delta.getChangeList()
```

## 参数

无。

## 返回

关联的 **DeltaItem** 实例的数组。

## 说明

方法：返回关联的 **DeltaItem** 实例的数组。数组中的每个 **DeltaItem** 实例描述对项目所做的更改。

## 示例

以下示例调用 `getChangeList()` 方法：

```
//...
case mx.data.components.datasetclasses.DeltaPacketConsts.Modified: {
 // dpDelta 是 Delta 类型的变量。
 var changes:Array = dpDelta.getChangeList();
 for(var i:Number = 0; i<changes.length; i++) {
 // getChangeMessage 是用户定义的方法。
 changeMsg = _parent.getChangeMessage(changes[i]);
 trace(changeMsg);
 }
//...
```

# Delta.getDeltaPacket()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
delta.getDeltaPacket()
```

## 参数

无。

## 返回

包含此增量的增量数据包。

## 说明

方法：返回包含此增量的增量数据包。使用此方法可编写用于通常可在增量级别上处理增量数据包的代码。

## 示例

以下示例使用 `getDeltaPacket()` 方法访问增量数据包的数据源：

```
while(dpCursor.hasNext()) {
 dpDelta = Delta(dpCursor.next());
 trace("DeltaPacket source is: " + dpDelta.getDeltaPacket().getSource());
}
```

# Delta.getId()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
delta.getId()
```

## 参数

无。



## 返回

一个对象；它返回此项目在 **DeltaPacket** 集合内的唯一 ID。

## 说明

方法；返回此项目在 **DeltaPacket** 集合内的唯一标识符。在源组件中为增量数据包使用此 ID，以接收更新以及对从中生成该增量数据包的项目进行更改。例如，假设 **DataSet** 组件将更新发送到服务器，且服务器返回新的关键字段值，则此方法允许 **DataSet** 组件检查生成的增量数据包，查找原始的传输对象，并对其进行适当的更新。

## 示例

以下示例调用 `getId()` 方法：

```
while(dpCursor.hasNext()) {
 dpDelta = Delta(dpCursor.next());
 trace("id ["+dpDelta.getId()+"]");
}
```

# Delta.getItemByName()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

`delta.getItemByName(name)`

## 参数

*name* 一个字符串，它为关联的 **DeltaItem** 对象指定属性或方法的名称。

## 返回

*name* 指定的 **DeltaItem** 对象。如果未找到匹配 *name* 的 **DeltaItem** 对象，则此方法返回 `null`。

## 说明

方法；返回由 *name* 指定的 **DeltaItem** 对象。如果需要按名称对传输对象进行方法调用或属性更改，此方法提供了最有效的访问方法。

## 示例

以下示例调用 `getItemByName()` 方法：

```
private function buildFieldTag(deltaObj:Delta, field:Object,
 isKey:Boolean):String {
 var chgItem:DeltaItem = deltaObj.getItemByName(field.name);
 var result:String= "<field name=\"" + field.name + "\" type=\"" +
 field.type.name + "\"";
 var oldValue:String;
 var newValue:String;
 if (deltaObj.getOperation() != DeltaPacketConsts.Added) {
 oldValue = (chgItem != null ? (chgItem.oldValue != null ?
 encodeFieldValue(field.name, chgItem.oldValue) : __nullValue) :
 encodeFieldValue(field.name, deltaObj.getSource()[field.name]));
 newValue = (chgItem.newValue != null ? encodeFieldValue(field.name,
 chgItem.newValue) : __nullValue);
 result+= " oldValue=\"" + oldValue + "\"";
 result+= chgItem != null ? " newValue=\"" + newValue + "\" : ";
 result+= " key=\"" + isKey.toString() + "\" />";
 }
 else {
 result+= " newValue=\"" +encodeFieldValue(field.name,
 deltaObj.getSource()[field.name]) + "\"";
 result+= " key=\"" + isKey.toString() + "\" />";
 }
 return result;
}
```

# Delta.getMessage()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*delta*.getMessage()

## 参数

无。

## 返回

一个字符串；返回与 *delta* 关联的消息。

### 说明

方法：返回此增量的关联消息。通常只有在为了响应尝试的更新而从服务器返回增量数据包的情况下，才会填充此消息。有关更多信息，请参见第 971 页的“[RDBMSResolver 组件](#)（仅限 [Flash Professional](#)）”。

### 示例

以下示例调用 `getMessage()` 方法：

```
//...
var dpi:Iterator = dp.getIterator();
var d:Delta;
while(dpi.hasNext()) {
 d= dpi.next();
 trace(d.getMessage());
}
//...
```

## Delta.getOperation()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

`delta.getOperation()`

### 参数

无。

### 返回

一个数字：返回对原始集合内的某项目执行的操作。

### 说明

方法：返回对原始集合内的某项目执行的操作。它的有效值包括 `DeltaPacketConsts.Added`、`DeltaPacketConsts.Removed` 和 `DeltaPacketConsts.Modified`。

您必须导入 `mx.data.components.datasetclasses.DeltaPacketConsts` 或完全限定每个常数。

## 示例

以下示例调用 `getOperation()` 方法：

```
while(dpCursor.hasNext()) {
 dpDelta = Delta(dpCursor.next());
 op=dpDelta.getOperation();
 trace("DeltaPacket source is: " + dpDelta.getDeltaPacket().getSource());
 switch(op) {
 case mx.data.components.datasetclasses.DeltaPacketConsts.Added:
 trace("***In case DeltaPacketConsts.Added ***");
 case mx.data.components.datasetclasses.DeltaPacketConsts.Modified: {
 trace("***In case DeltaPacketConsts.Modified ***");
 }
 }
}
```

# Delta.getSource()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
delta.getSource()
```

## 参数

无。

## 返回

被更改的传输对象。

## 说明

方法；返回被更改的传输对象。

## 示例

以下示例调用 `getSource()` 方法:

```
while(dpCursor.hasNext()) {
 dpDelta = Delta(dpCursor.next());
 op=dpDelta.getOperation();
 switch(op) {
 case mx.data.components.datasetclasses.DeltaPacketConsts.Modified: {
 // 原始值为
 trace("Unmodified source is: ");
 var src = dpDelta.getDeltaPacket().getSource();
 for(var i in src){
 if(typeof(src[i]) != "function"){
 trace(i+"="+src[i]);
 }
 }
 }
 }
}
```



# DeltaPacket 接口（仅限 Flash Professional）

ActionScript 接口名称 `mx.data.components.datasetclasses.DeltaPacket`

DeltaPacket 接口由 DataSet 组件的 `deltaPacket` 属性提供，DataSet 组件是 Flash MX Professional 2004 中的数据管理功能的一部分。（有关更多信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”）。增量数据包通常由解析程序组件内部使用。DeltaPacket 接口和相关的 Delta 接口以及 DeltaItem 类允许您管理对数据的更改。这些组件在运行时没有可视外观。

增量数据包是一组优化的说明，这些说明描述对数据集中的数据已进行的所有更改。当调用 `DataSet.applyUpdates()` 方法时，DataSet 组件将会填充 `DataSet.deltaPacket` 属性。通常，此属性（通过数据绑定）连接到诸如 `RDBMSResolver` 之类的解析程序组件。解析程序将增量数据包转换为采用相应格式的更新包。



除非您要编写自己的自定义解析程序，否则您无需了解或编写用于访问增量数据包的方法或属性的代码。

增量数据包包含一个或多个增量（请参见第 433 页的“Delta 接口（仅限 Flash Professional）”），而每个增量包含零个或多个增量项（请参见第 427 页的“DeltaItem 类（仅限 Flash Professional）”）。

## DeltaPacket 接口的方法摘要

下表列出了 DeltaPacket 接口的方法。

方法	说明
<code>DeltaPacket.getConfigInfo()</code>	返回特定于 DeltaPacket 接口的实现的配置信息。
<code>DeltaPacket.getIterator()</code>	返回某增量数据包的重复值，该值循环访问该增量数据包的增量列表。
<code>DeltaPacket.getSource()</code>	返回增量数据包的源。这是已公开此增量数据包的组件。
<code>DeltaPacket.getTimestamp()</code>	返回实例化增量数据包的日期和时间。

方法	说明
<code>DeltaPacket.getTransactionId()</code>	返回此增量数据包的事务 ID。
<code>DeltaPacket.logChanges()</code>	指示增量数据包的使用者是否应记录它指定的更改。

## DeltaPacket.getConfigInfo()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
deltaPacket.getConfigInfo(info)
```

### 属性

*info* 对象：包含特定于实现的信息。

### 返回

一个对象，它包含特定的 `DeltaPacket` 实现所需的信息。

### 说明

方法；返回特定于 `DeltaPacket` 接口的实现的配置信息。此方法允许 `DeltaPacket` 接口的实现访问自定义的信息。

### 示例

以下示例调用 `getConfigInfo()` 方法：

```
// ...
new DeltaPacketImpl(source, getTransactionId(), null, logChanges(),
 getConfigInfo());
// ...
```



# DeltaPacket.getIterator()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
deltaPacket.getIterator()
```

## 参数

无。

## 返回

**DeltaPacket** 集合重复值的接口，该值循环访问该增量数据包的增量列表。

## 说明

方法：返回 **DeltaPacket** 集合的重复值。它提供了一种用于遍历增量数据包中的更改的机制。有关此接口的完整说明，请参见第 691 页的“[Iterator 接口（仅限 Flash Professional）](#)”。

## 示例

以下示例使用 `getIterator()` 方法来访问增量数据包中增量的重复值，并使用 `while` 语句来遍历增量：

```
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
trace("*** Test deltapacket. Trans ID is: " + deltapkt.getTransactionId() + "
 ***");
var OPS:Array = new Array("added", "removed", "modified");
var dpCursor:Iterator = deltapkt.getIterator();
var dpDelta:Delta;
var op:Number;
var changeMsg:String;
while(dpCursor.hasNext()) {
 dpDelta = Delta(dpCursor.next());
 op=dpDelta.getOperation();
}
```

# DeltaPacket.getSource()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
deltaPacket.getSource()
```

## 参数

无。

## 返回

一个对象；它是 **DeltaPacket** 集合的源。此对象通常继承 **MovieClip** 对象，但并不要求一定如此。例如，如果源是数据集，则此对象可能是 `_level0.myDataSet`。

## 说明

方法；返回 **DeltaPacket** 集合的源。

## 示例

以下示例调用 `getSource()` 方法：

```
// ...
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
var dpSourceText:String = "Source: " + deltapkt.getSource();
trace(dpSourceText);
// ...
```

# DeltaPacket.getTimestamp()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
deltaPacket.getTimestamp()
```

### 参数

无。

### 返回

一个 **Date** 对象，它包含创建增量数据包的日期和时间。

### 说明

方法；返回创建增量数据包的日期和时间。

### 示例

以下示例调用 `getTimestamp()` 方法：

```
// ...
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
var dpTime:String = " Time: " + deltapkt.getTimestamp();
trace(dpTime);
// ...
```

## DeltaPacket.getTransactionId()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
deltaPacket.getTransactionId()
```

### 参数

无。

### 返回

一个字符串；它是增量数据包的单个事务组合的唯一事务 ID。

### 说明

方法；返回增量数据包的事务 ID。此唯一标识符用于对增量数据包的发送 / 接收事务分组。数据集使用它来确定增量数据包是否属于通过调用 `DataSet.applyUpdates()` 而产生的同一事务。

## 示例

以下示例调用 `getTransactionId()` 方法：

```
// ...
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
trace("*** Trans ID is: " + deltapkt.getTransactionId() + " ***");
// ...
```

# DeltaPacket.logChanges()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
deltaPacket.logChanges()
```

## 参数

无。

## 返回

布尔值；当增量数据包的使用者应当记录在其中发现的更改时，为 `true`。

## 说明

方法；当此增量数据包的使用者应当记录它指定的更改时，返回 `true`。此值主要用于通过以下方式对数据集之间的更改进行通信：一种是通过共享对象进行通信，另一种是从服务器数据集到本地数据集进行通信。在这两种情况下，数据集均不应记录指定的更改。

## 示例

以下示例调用 `logChanges()` 方法：

```
var deltapkt:DeltaPacket = _parent.myDataSet.deltaPacket;
if(deltapkt.logChanges()) {
 trace("*** We need to log changes. ***");
}
else {
 trace("*** We do not need to log changes");
}
```

# DepthManager 类

ActionScript 类名称 `mx.managers.DepthManager`

**DepthManager** 类允许您管理对任何组件或影片剪辑的相对深度（包括 `_root`）的分配。它还允许您为系统级服务（如指针和工具提示）管理在 `_root` 上的最深的特殊剪辑中保留的深度。

通常，深度管理器使用其自身的“混合”算法自动管理组件。除非您是高级 Flash 开发人员，否则无需使用其 API。



若要使用影片剪辑实例的 **DepthManager** 类，库中或舞台上必须具有组件，并在 ActionScript 的开始部分使用“`import mx.managers.DepthManager`”。

以下方法组成了按相对深度排序的 API：

- `DepthManager.createChildAtDepth()`
- `DepthManager.createClassChildAtDepth()`
- `DepthManager.setDepthAbove()`
- `DepthManager.setDepthBelow()`
- `DepthManager.setDepthTo()`

以下方法组成了保留深度空间 API：

- `DepthManager.createClassObjectAtDepth()`
- `DepthManager.createObjectAtDepth()`

## DepthManager 类的方法摘要

下表列出了 **DepthManager** 类的方法。

方法	说明
<code>DepthManager.createChildAtDepth()</code>	在指定深度处创建指定元件的子级。
<code>DepthManager.createClassChildAtDepth()</code>	在该指定深度处创建指定类的对象。
<code>DepthManager.createClassObjectAtDepth()</code>	在特殊最深剪辑中的指定深度处创建指定类的实例。

方法	说明
<code>DepthManager.createObjectAtDepth()</code>	在最深剪辑中的指定深度处创建一个对象。
<code>DepthManager.setDepthAbove()</code>	将深度设置到指定实例之上。
<code>DepthManager.setDepthBelow()</code>	将深度设置到指定实例之下。
<code>DepthManager.setDepthTo()</code>	将深度设置为最深剪辑中的指定实例。

## DepthManager 类的属性摘要

下表列出了 **DepthManager** 类的属性。显示的常数值是 **DepthManager** 算法用于排列深度的默认值。如果您跟踪以下属性，就会在“输出”面板中看到这些常数值。

不过，在实现了 **DepthManager** 方法（如 `DepthManager.setDepthTo()`）后，使用以下一种属性，然后跟踪组件或影片剪辑深度，就会看到 **DepthManager** 按 20 的增量设置深度。当 **Flash** 需要在中间插入其它内容的情况下，算法会根据其它脚本、组件等来增加深度。

属性	说明
<code>DepthManager.kBottom</code>	具有常数值 202 的静态属性。
<code>DepthManager.kCursor</code>	具有常数值 101 的静态属性。这是光标深度。
<code>DepthManager.kNotopmost</code>	具有常数值 204 的静态属性。
<code>DepthManager.kTooltip</code>	具有常数值 102 的静态属性。这是工具提示深度。
<code>DepthManager.kTop</code>	具有常数值 201 的静态属性。
<code>DepthManager.kTopmost</code>	具有常数值 203 的静态属性。

## DepthManager.createChildAtDepth()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
movieClipInstance.createChildAtDepth(linkageName, depthFlag[, initObj])
```

### 参数

*linkageName* 一个链接标识符。此参数是一个字符串。

*depthFlag* 它是下列值之一：[DepthManager.kTop](#)、[DepthManager.kBottom](#)、[DepthManager.kTopmost](#)、[DepthManager.kNotopmost](#)。所有深度标志都是 **DepthManger** 类的静态属性。您必须引用 **DepthManager** 包（如 `mx.managers.DepthManager.kTopmost`），或者使用 `import` 语句导入 **DepthManager** 包。

*initObj* 一个初始化对象。此参数是可选的。

### 返回

指向所创建的对象引用。返回类型是 **MovieClip**。

### 说明

方法：在 *depthFlag* 所指定的深度创建一个由 *linkageName* 指定的元件的子实例。

### 示例

以下示例创建 **MinuteSymbol** 影片剪辑的实例 `minuteHand`，并将其放在 `clock` 前面：

```
import mx.managers.DepthManager;
minuteHand = clock.createClassChildAtDepth("MinuteSymbol", DepthManager.kTop);
```

## DepthManager.createClassChildAtDepth()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
movieClipInstance.createClassChildAtDepth(className, depthFlag[, initObj])
```

### 参数

*className* 类名称。此参数为 **Function** 类型。

*depthFlag* 它是下列值之一：[DepthManager.kTop](#)、[DepthManager.kBottom](#)、[DepthManager.kTopmost](#)、[DepthManager.kNotopmost](#)。所有深度标志都是 **DepthManger** 类的静态属性。您必须引用 **DepthManager** 包（如 `mx.managers.DepthManager.kTopmost`），或者使用 `import` 语句导入 **DepthManager** 包。

*initObj* 一个初始化对象。此参数是可选的。

### 返回

指向所创建的子级的引用。返回类型是 **UIObject**。

## 说明

方法：在 *depthFlag* 所指定的深度创建一个由 *className* 指定的类的子类。

## 示例

下面的代码在所有 **NoTopmost** 对象之上绘制一个焦点矩形：

```
import mx.managers.DepthManager
this.ring = createClassChildAtDepth(mx.skins.RectBorder, DepthManager.kTop);
```

以下代码创建一个 **Button** 类的实例，并将其 *label* 属性值作为 *initObj* 参数传递给该实例：

```
import mx.managers.DepthManager
button1 = createClassChildAtDepth(mx.controls.Button, DepthManager.kTop,
 {label: "Top Button"});
```

# DepthManager.createClassObjectAtDepth()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.createClassObjectAtDepth(className, depthSpace[, initObj])`

## 参数

*className* 类名称。此参数为 **Function** 类型。

*depthSpace* 它是下列值之一：[DepthManager.kCursor](#)、[DepthManager.kTooltip](#)。所有深度标志都是 **DepthManger** 类的静态属性。您必须引用 **DepthManager** 包（如 `mx.managers.DepthManager.kCursor`），或者使用 `import` 语句导入 **DepthManager** 包。

*initObj* 一个初始化对象。此参数是可选的。

## 返回

指向所创建的对象引用。返回类型是 **UIObject**。

## 说明

方法：在 *depthSpace* 所指定的深度创建一个由 *className* 指定的类的对象。该方法用于访问特殊最高深度剪辑中保留的深度空间。



## 示例

下面的示例创建一个 **Button** 类的对象：

```
import mx.managers.DepthManager
myCursorButton = DepthManager.createClassObjectAtDepth(mx.controls.Button,
 DepthManager.kCursor, {label: "Cursor"});
```

# DepthManager.createObjectAtDepth()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.createObjectAtDepth(linkageName, depthSpace[, initObj])`

## 参数

*linkageName* 一个链接标识符。此参数为 **String** 类型。

*depthSpace* 它是下列值之一：`DepthManager.kCursor`、`DepthManager.kTooltip`。所有深度标志都是 **DepthManger** 类的静态属性。您必须引用 **DepthManager** 包（如 `mx.managers.DepthManager.kCursor`），或者使用 `import` 语句导入 **DepthManager** 包。

*initObj* 一个可选的初始化对象。

## 返回

指向所创建的对象引用。返回类型是 **MovieClip**。

## 说明

方法：在指定深度处创建一个对象。该方法用于访问特殊最高深度剪辑中保留的深度空间。

## 示例

下面的示例创建一个 **TooltipSymbol** 元件的实例，并将其放在为工具提示保留的深度：

```
import mx.managers.DepthManager
myCursorTooltip = DepthManager.createObjectAtDepth("TooltipSymbol",
 DepthManager.kTooltip);
```

# DepthManager.kBottom

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.kBottom`

## 说明

属性（静态）：具有常数值 202 的属性。在调用 [DepthManager.createClassChildAtDepth\(\)](#) 和 [DepthManager.createChildAtDepth\(\)](#) 时，该属性作为参数进行传递，可将内容放置在其它内容之后。

# DepthManager.kCursor

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.kCursor`

## 说明

属性（静态）：具有常数值 101 的属性。在调用 [DepthManager.createClassObjectAtDepth\(\)](#) 和 [DepthManager.createObjectAtDepth\(\)](#) 时，该参数作为参数进行传递，可请求定位在光标深度处。

# DepthManager.kNotopmost

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.kNotopmost`

## 说明

属性（静态）：具有常数值 204 的属性。在调用 [DepthManager.createClassChildAtDepth\(\)](#) 和 [DepthManager.createChildAtDepth\(\)](#) 时，该属性作为参数进行传递，可请求从最顶层删除。

# DepthManager.kTooltip

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.kTooltip`

## 说明

属性（静态）：具有常数值 102 的属性。在调用 [DepthManager.createClassObjectAtDepth\(\)](#) 和 [DepthManager.createObjectAtDepth\(\)](#) 时，该属性作为参数进行传递，可在工具提示深度处放置对象。

# DepthManager.kTop

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.kTop`

## 说明

属性（静态）：具有常数值 201 的属性。在调用 `DepthManager.createClassChildAtDepth()` 和 `DepthManager.createChildAtDepth()` 时，该属性作为参数进行传递，可请求定位在 `DepthManager.kTopmost` 内容之下和其它内容之上。

# DepthManager.kTopmost

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`DepthManager.kTopmost`

## 说明

属性（静态）：具有常数值 203 的属性。在调用 `DepthManager.createClassChildAtDepth()` 和 `DepthManager.createChildAtDepth()` 时使用此属性，可请求定位在其它内容（包括 `DepthManager.kTop` 对象）之上。

# DepthManager.setDepthAbove()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*movieClipInstance.setDepthAbove(instance)*

## 参数

*instance* 一个实例名称。此参数为 **MovieClip** 类型。

## 返回

无。

## 说明

方法：将影片剪辑或组件实例的深度设置到 *instance* 参数所指定实例的深度之上，必要时还可以移动其它对象。

# DepthManager.setDepthBelow()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

*movieClipInstance.setDepthBelow(instance)*

## 参数

*instance* 一个实例名称。此参数为 **MovieClip** 类型。

## 返回

无。

## 说明

方法：将影片剪辑或组件实例的深度设置到指定的实例的深度之下，如若需要还可移动其它对象。

## 示例

以下代码将 `textInput` 实例的深度设置到 `button` 的深度之下：

```
textInput.setDepthBelow(button);
```

# DepthManager.setDepthTo()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
movieClipInstance.setDepthTo(depthFlag)
```

## 参数

*depthFlag* 它是下列值之一：`DepthManager.kTop`、`DepthManager.kBottom`、`DepthManager.kTopmost`、`DepthManager.kNotopmost`。所有深度标志都是 `DepthManger` 类的静态属性。您必须引用 `DepthManager` 包（如 `mx.managers.DepthManager.kTopmost`），或者使用 `import` 语句导入 `DepthManager` 包。

## 返回

无。

## 说明

方法：将 *movieClipInstance* 的深度设置为 *depthFlag* 指定的值。此方法将实例移动到其它深度，以便留出空间供其它对象使用。`DepthManager` 使用“混合”算法按 20 的增量设置深度。在 `Flash` 需要在中间插入其它内容的情况下，算法会根据其它脚本、组件等来增量深度。

## 示例

以下示例使用两个组件（或影片剪辑），当单击每个组件（或影片剪辑）时，示例将以 20 的增量交替增加它们的深度。首先将一个 **Button** 组件添加到舞台上，并为其指定实例名称 a\_btn，然后将另一个 **Button** 组件添加到舞台上，并为其指定实例名称 b\_btn。确保按钮按以下方式重叠：



```
import mx.managers.DepthManager;

a_btn.onRelease = function() {
 b_btn.setDepthTo(DepthManager.kTop);
 var b_depth:Number = b_btn.getDepth();
 trace(b_depth);
}

b_btn.onRelease = function() {
 a_btn.setDepthTo(DepthManager.kTop);
 var a_depth:Number = a_btn.getDepth();
 trace(a_depth);
}
```

测试 SWF 文件。当您单击最上面的按钮时，另一个按钮将会更改深度并移动到前面，并且“输出”面板会显示该按钮的深度。值为 20，然后为 40、60，您单击一次增加 20。

提醒

如果您将 **DepthManager** 与影片剪辑实例而不是与组件实例一起使用，则需要向库中添加一个 UI 组件（如果库中还没有这样的组件），以便 **DepthManager** 可以正常运行。**DepthManager** 需要舞台上或库中具有组件才能起作用。

有关深度和堆叠顺序的更多信息，请参见《学习 Flash 中的 ActionScript 2.0》中的“确定下一个最大的可用深度”。





# EventDispatcher 类

事件允许应用程序了解用户何时与组件进行了交互操作，以及组件的外观或生命周期何时发生了重要更改，例如其创建、删除或调整大小。

**EventDispatcher** 类的方法允许您添加和删除事件侦听器，以便代码可以相应地响应事件。例如，可以使用 `EventDispatcher.addEventListener()` 方法向组件实例注册侦听器。触发组件事件时，侦听器会被调用。

如果想要编写一个自定义对象，该对象发出与用户界面无关的事件，则 **EventDispatcher** 用作 **UIComponent** 的混合件时，要比 **UIEventDispatcher** 小，运行速度也要快。

## 事件对象

事件对象作为参数传递到侦听器。事件对象是一种 **ActionScript** 对象，这种对象具有的属性中包含有关所发生的事件的信息。您可以在侦听器回调函数内使用事件对象来访问所广播的事件的名称，或者访问广播该事件的组件的实例名称。例如，以下代码使用 `evtObj` 事件对象的 `target` 属性来访问 `myButton` 实例的 `label` 属性，并将值发送到“输出”面板：

```
listener = new Object();
listener.click = function(evtObj){
 trace("The " + evtObj.target.label + " button was clicked");
}
myButton.addEventListener("click", listener);
```

有些事件对象属性在 W3C 规范 ([www.w3.org/TR/DOM-Level-3-Events/events.html](http://www.w3.org/TR/DOM-Level-3-Events/events.html)) 中定义，但在 **Macromedia Component Architecture** 的第 2 版中并未实现。下表列出了第 2 版的每个事件对象所具有的属性。一些事件还定义有其它属性，如果是这样的话，这些属性将在该事件的条目中列出。

属性	说明
<code>type</code>	指示事件名称的字符串。
<code>target</code>	对广播事件的组件实例的引用。

# EventDispatcher 类 (API)

ActionScript 类名称 `mx.events.EventDispatcher`

## EventDispatcher 类的方法摘要

下表列出了 `EventDispatcher` 类的方法。

方法	说明
<code>EventDispatcher.addEventListener()</code>	向组件实例注册侦听器。
<code>EventDispatcher.dispatchEvent()</code>	以编程方式发送事件。
<code>EventDispatcher.removeEventListener()</code>	从组件实例删除事件侦听器。

## EventDispatcher.addEventListener()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004 和 Flash MX Professional 2004。

### 用法

`componentInstance.addEventListener(event, listener)`

### 参数

*event* 表示事件名称的字符串。

*listener* 对侦听器对象或函数的引用。

### 返回

无。

## 说明

方法：在播放事件的组件实例上注册侦听器对象。事件发生时，将通知侦听器对象或函数。您可以从任何组件实例调用此方法。例如，以下代码将侦听器注册到组件实例 `myButton`：

```
myButton.addEventListener("click", myListener);
```

在调用 `addEventListener()` 将侦听器注册到组件实例之前，您必须先将该侦听器定义为对象或函数。如果侦听器是对象，则必须定义一个回调函数，当事件发生时，将会调用该回调函数。通常，回调函数与注册侦听器所使用的事件同名。如果侦听器是函数，则事件发生时，将调用该函数。有关更多信息，请参见《使用组件》中的“使用侦听器处理事件”。

您可以将多个侦听器注册到一个组件实例，但必须为每个侦听器单独调用 `addEventListener()`。而且，也可以将一个侦听器注册到多个组件实例，但必须为每个实例单独调用 `addEventListener()`。例如，以下代码定义了一个侦听器对象，并将其分配给两个 **Button** 组件实例，这两个实例的 `label` 属性分别是 `button1` 和 `button2`：

```
lo = new Object();
lo.click = function(evt){
 trace(evt.target.label + " clicked");
}
button1.addEventListener("click", lo);
button2.addEventListener("click", lo);
```

不保证执行顺序。无法要求在调用一个侦听器之前先调用另一个侦听器。

事件对象作为参数传递给侦听器。事件对象具有包含有关所发生事件的信息的属性。您可以在侦听器回调函数内使用事件对象来访问有关所发生的事件类型的信息，以及哪个实例广播该事件的信息。在上面的示例中，事件对象是 `evt`（您可以将任何标识符用作事件对象名称）。它在 `if` 语句内使用，用于确定单击了哪个按钮实例。有关更多信息，请参见《使用组件》中的“关于事件对象”。

## 示例

以下示例定义了一个侦听器对象 `myListener`，并为 `click` 事件定义了回调函数。然后，它调用 `addEventListener()` 将 `myListener` 侦听器对象注册到组件实例 `myButton`。

```
myListener = new Object();
myListener.click = function(evt){
 trace(evt.type + " triggered");
}
myButton.addEventListener("click", myListener);
```

若要测试这段代码，请将实例名称为 `myButton` 的 **Button** 组件放在舞台上，并将这段代码置于第一帧中。

# EventDispatcher.dispatchEvent()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

`dispatchEvent(eventObject)`

## 参数

*eventObject* 对事件对象的引用。事件对象必须具有 `type` 属性，该属性是一个指示事件名称的字符串。通常，事件对象还具有 `target` 属性，该属性是广播该事件的实例的名称。您可以为事件对象定义其它属性，这些属性将有助于用户在发送该事件时捕获有关该事件的信息。

## 返回

无。

## 说明

方法：将事件发送到任何已向类的实例注册的侦听器。通常从组件的类文件内部调用此方法。例如，`SimpleButton.as` 类文件调度 `click` 事件。

## 示例

以下示例将调度一个 `click` 事件：

```
dispatchEvent({type:"click"});
```

# EventDispatcher.removeEventListener()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

```
componentInstance.removeEventListener(event, listener)
```

## 参数

*event* 表示事件名称的字符串。

*listener* 对侦听器对象或函数的引用。

## 返回

无。

## 说明

方法：从播放事件的组件实例注销侦听器对象。



# FLVPlayback 组件（仅限 Flash Professional）

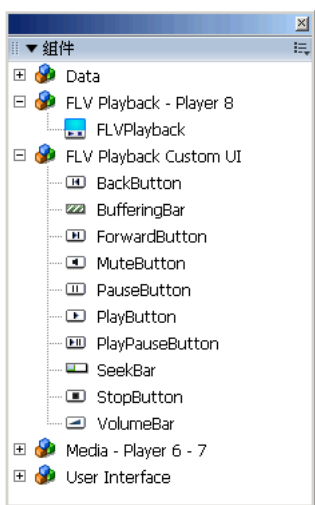
# 22

通过 FLVPlayback 组件，您可以轻松地将视频播放器包括在 Flash 应用程序中，以便播放通过 HTTP 渐进式下载的 Flash 视频 (FLV) 文件，或者从 Flash Communication Server (FCS) 或 Flash Video Streaming Service (FVSS) 播放 FLV 文件流。

易于使用的 FLVPlayback 组件具有以下特性和优点：

- 可拖到舞台并顺利地快速实现
- 提供预先设计的外观集合，这些外观可用于自定义其播放控件的外观
- 允许高级用户创建他们自己的外观
- 提供可用于将视频与文本、图形和动画同步的提示点
- 提供对自定义内容的实时预览
- 保持合理大小的 SWF 文件

FLVPlayback 组件包括 FLV 回放自定义用户界面组件。FLVPlayback 组件是显示区域（或视频播放器）的组合，从中可以查看 FLV 文件以及允许您对该文件进行操作的控件。FLV 回放自定义用户界面组件提供控制按钮和机制，可用于播放、停止、暂停 FLV 文件以及对该文件进行其它控制。这些控件包括 BackButton、BufferingBar、ForwardButton、MuteButton、PauseButton、PlayButton、PlayPauseButton、SeekBar、StopButton 和 VolumeBar。FLVPlayback 组件和 FLV 回放自定义用户界面控件显示在“组件”面板中，如下图所示：



用于将回放控件添加到 FLVPlayback 组件的过程称作外观设置。FLVPlayback 组件具有初始默认外观 ClearOverPlaySeekMute.swf，它为播放、搜索和静音功能提供一目了然的控件。若要更改此外观，您具有以下选择：

- 从预设计外观集合中进行选择
- 从 FLV 回放自定义用户界面组件中选择单独的控件并对它们进行自定义
- 创建自定义外观并将该外观添加到预设计外观集合

在选择了不同外观后，所选的外观将成为新的默认外观。

有关选择或创建 FLVPlayback 组件的外观的更多信息，请参见第 484 页的“自定义 FLVPlayback 组件”。

FLVPlayback 组件还包括一个 ActionScript 应用程序编程接口 (API)。该 API 包括 FLVPlayback 类、VideoError 类和 VideoPlayer 类。有关这些类的更多信息，请参见第 497 页的“FLVPlayback 类”、第 649 页的“VideoPlayer 类”和第 642 页的“VideoError 类”。



# 使用 FLVPlayback 组件

FLVPlayback 组件的使用过程基本上由两个步骤组成：第一步是将该组件放置在舞台上，第二步是指定一个供它播放的 FLV 文件。但除此之外，您还可以设置不同的参数，以控制其行为并描述 FLV 文件。

## 创建具有 FLVPlayback 组件的应用程序

您可以通过以下方式将 FLVPlayback 组件包括在您的应用程序中：

- 将 FLVPlayback 组件从“组件”面板拖到舞台上，并且为 contentPath 参数指定一个值。
- 使用视频导入向导在舞台上创建组件，并通过选择外观来自定义该组件。
- 使用 MovieClip attachMovie() 方法动态地在舞台上创建一个 FLVPlayback 实例，并且假定该组件位于库中。

从“组件”面板中拖出 FLVPlayback 组件：

1. 在“组件”面板中，单击加号 (+) 按钮打开“FLV Playback - Player 8”条目。
2. 将 FLVPlayback 组件拖到舞台上。
3. 在选中舞台上的 FLVPlayback 组件的情况下，在“组件”检查器的“参数”选项卡上找到 contentPath 参数的 Value 单元格，并输入指定以下内容之一的字符串：
  - 指向 FLV 文件的本地路径
  - 指向 FLV 文件的 URL
  - 指向 XML 文件的 URL，该文件说明如何播放 FLV 文件有关如何创建 XML 文件以描述一个或多个 FLV 文件的信息，请参见第 653 页的“使用 SMIL 文件”。
4. 在选中舞台上的 FLVPlayback 组件的情况下，在“组件”检查器的“参数”选项卡上单击 skin 参数的 Value 单元格。
5. 单击放大镜图标以打开“选择外观”对话框。
6. 选择以下选项之一：
  - 从“外观”下拉列表中，选择一种预制外观以将一组回放控件附加到组件中。
  - 如果已创建了一种自定义外观，可从下拉菜单中选择“自定义外观 URL”，然后在“URL”文本框中输入包含此外观的 SWF 文件的 URL。
  - 选择“无”，然后将单个 FLV 回放自定义用户界面组件拖到舞台上，以添加回放控件。



在前两种情况下，会在弹出菜单上部的查看窗格中显示外观的预览效果。

7. 单击“确定”以关闭“选择外观”对话框。
8. 从“控制”菜单中选择“测试影片”以执行 SWF 文件并启动视频。

### 使用视频导入向导：

1. 选择 “文件” > “导入” > “导入视频”。
2. 通过选择以下选项之一，指示视频文件的位置：
  - 在我的本地计算机上
  - 已经部署到 Web、FCS 或 FVSS 服务器
3. 根据您的选择，输入指定视频文件位置的路径或 URL；然后单击 “下一步”。
4. 如果您选择了文件路径，则会在旁边看到 “部署” 对话框，从中可以选择列出的选项之一，以指定希望部署视频的方式：
  - 从标准 Web 服务器渐进式下载
  - 从 Flash Video Streaming Service 流式加载
  - 从 Flash Communication Server 流式加载
  - 在 SWF 中嵌入视频并按时间轴播放



不要选择 “嵌入视频” 选项。FLVPlayback 组件仅播放外部视频流。此选项不将 FLVPlayback 组件放置于舞台上。

5. 单击 “下一步”。
6. 选择以下选项之一：
  - 从 “外观” 下拉列表中，选择一种预制外观以将一组回放控件附加到组件中。
  - 如果已经为该组件创建了一种自定义外观，则可从下拉菜单中选择 “自定义外观 URL”，然后在 “URL” 文本框中输入包含此外观的 SWF 文件的 URL。
  - 选择 “无”，然后将单个 FLV 回放自定义用户界面组件拖到舞台上，以添加回放控件。



在前两种情况下，会在弹出菜单上部的查看窗格中显示外观的预览效果。

7. 单击 “确定” 以关闭 “选择外观” 对话框。
8. 阅读 “完成视频导入” 对话框中的内容，注意下一步的操作，然后单击 “完成”。
9. 如果您尚未保存 FLA 文件，将出现 “另存为” 对话框。
10. 从 “控制” 菜单中选择 “测试影片”，以执行 SWF 文件并启动视频。

## 使用 ActionScript 动态创建实例：

1. 将 FLVPlayback 组件从“组件”面板拖到库（“窗口” > “库”）中。
2. 将以下代码添加到时间轴第 1 帧的“动作”面板上。将 `install_drive` 更改为安装了 Flash 8 的驱动器，并修改路径以反映用于安装的 Skins 文件夹的位置。

```
import mx.video.*;
this.attachMovie("FLVPlayback", "my_FLVPlybk", 10, {width:320,
 height:240, x:100, y:100});
my_FLVPlybk.skin = "file:///install_drive|/Program Files/Macromedia/Flash
 8/en/Configuration/Skins/ClearOverPlaySeekMute.swf"
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

`attachMovie()` 方法属于 **MovieClip** 类。您可以使用它来创建 **FLVPlayback** 组件的实例，因为 **FLVPlayback** 类扩展 **MovieClip** 类。



如果未设置 `contentPath` 和 `skin` 属性，生成的影片剪辑将显示为空。

3. 从“控制”菜单中选择“测试影片”，以执行 SWF 文件并启动 FLV 文件。

## FLVPlayback 组件参数

对于每个 **FLVPlayback** 组件实例，都可以在“组件”检查器或“属性”检查器中设置以下参数：

**autoPlay** 确定 FLV 文件的播放方式的布尔值。如果为 `true`，则该组件将在加载 FLV 文件后立即播放。如果为 `false`，则该组件加载第一帧然后暂停。对于默认视频播放器 (0)，默认值是 `true`，对于其它项则为 `false`。有关如何在单个 **FLVPlayback** 实例中使用多个视频播放器的更多信息，请参见第 481 页的“播放多个 FLV 文件”。

**autoRewind** 一个布尔值，用于确定 FLV 文件在它完成播放时是否自动后退。如果为 `true`，则播放头达到末端或用户单击“停止”按钮时，**FLVPlayback** 组件会自动使 FLV 文件后退到开始处。如果为 `false`，则组件在播放 FLV 文件的最后一帧后会停止，并且不自动后退。默认值为 `true`。

**autoSize** 一个布尔值，如果为 `true`，则在运行时调整组件大小以使用源 FLV 文件尺寸。这些尺寸是在 FLV 文件中进行编码的，并且不同于 **FLVPlayback** 组件的默认尺寸。默认值为 `false`。有关更多信息，请参见第 514 页的 **FLVPlayback.autoSize**。

**bufferTime** 在开始回放前，在内存中缓冲 FLV 文件的秒数。此参数影响 FLV 文件流，这些文件在内存中缓冲，但不下载。对于通过 HTTP 渐进式下载的 FLV 文件，增加该值只会带来很小的好处，尽管它可以改善在旧式、速度较慢的计算机上查看高质量视频的查看效果。默认值是 0.1。有关更多信息，请参见第 525 页的 **FLVPlayback.BUFFERTIME**。



设置此参数并不确保某些 FLV 文件内容将在回放开始前下载。

**contentPath** 一个字符串，指定 FLV 文件的 URL，或者指定描述如何播放一个或多个 FLV 文件的 XML 文件。您可以指定本地计算机上的路径、HTTP 路径或实时消息传输协议 (RTMP) 路径。双击此参数的 Value 单元格以打开“内容路径”对话框。默认值为一个空字符串。

如果未指定 contentPath 参数的值，则 Flash 执行 FLVPlayback 实例时什么也不会发生。有关更多信息，请参见第 473 页的“指定 contentPath 参数”。

**cuePoints** 描述 FLV 文件的提示点的字符串。提示点允许您同步包含 Flash 动画、图形或文本的 FLV 文件中的特定点。默认值为一个空字符串。有关更多信息，请参见第 474 页的“使用提示点”。

**isLive** 一个布尔值，如果为 true，则指定 FLV 文件正从 Flash Communication Server 实时加载流。实时流的一个示例就是在发生新闻事件的同时显示这些事件的视频。默认值为 false。有关更多信息，请参见第 553 页的 FLVPlayback.isLive。

**maintainAspectRatio** 一个布尔值，如果为 true，则调整 FLVPlayback 组件中视频播放器的大小，以保持源 FLV 文件的高宽比；FLV 文件根据舞台上 FLVPlayback 组件的尺寸进行缩放。autoSize 参数优先于此参数。默认值为 true。有关更多信息，请参见第 557 页的 FLVPlayback.maintainAspectRatio。

**skin** 一个参数，用于打开“选择外观”对话框，从该对话框中可以选择组件的外观。默认值最初是预先设计的外观，但它在以后将成为上次选择的外观。如果您选择 None，则 FLVPlayback 实例并不具有用于操作 FLV 文件的控制元素。如果 autoPlay 参数设置为 true，则会自动播放 FLV 文件。有关更多信息，请参见第 484 页的“自定义 FLVPlayback 组件”。

**skinAutoHide** 一个布尔值，如果为 true，则当鼠标不在 FLV 文件或外观区域（如果外观是不在 FLV 文件查看区域上的外部外观）上时隐藏外观。默认值为 false。有关更多信息，请参见第 617 页的 FLVPlayback.skin。

**totalTime** 源 FLV 文件中的总秒数，精确到毫秒。默认值为 0。

如果您使用 FCS 或 FVSS，则该组件始终从服务器获取总时间。

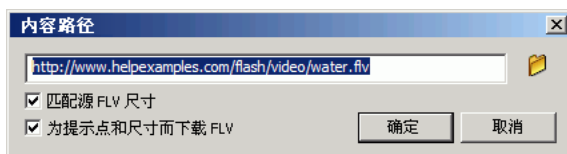
如果使用通过 HTTP 的渐进式下载，则该组件会在此值设置为大于零时使用此数值。否则，它将尝试获取 FLV 文件元数据的时间。有关更多信息，请参见第 629 页的 FLVPlayback.totalTime。

**volume** 一个从 0 到 100 的数字，用于表示相对于最大音量 (100) 的百分比。有关更多信息，请参见第 634 页的 FLVPlayback.volume。

上述每个参数都在 FLVPlayback 类中具有等效属性。设置属性后将覆盖“组件”检查器或“属性”检查器中的参数设置。

## 指定 contentPath 参数

contentPath 参数允许您指定 FLV 文件的名称和位置，这两项将指示 Flash 如何播放该文件。通过在“组件”检查器中双击 contentPath 参数的 Value 单元格，可以打开“内容路径”对话框。对话框类似于下图：



该对话框提供两个复选框，可用于确定 FLVPlayback 实例的尺寸并指定是否从 FLV 文件获取尺寸和提示点信息。有关更多信息，请参见第 474 页的“FLV 文件选项”。

## 内容路径

输入说明 FLV 文件播放方式的 FLV 文件或 XML 文件的 URL 或本地路径。如果您不知道 FLV 文件的确切位置，则通过单击文件夹图标打开“浏览器”对话框，可以帮助您找到正确的位置。浏览查找 FLV 文件时，如果它位于目标 SWF 文件所在位置或该位置以下，则 Flash 会自动使该路径成为相对于该位置的路径，以便通过 Web 服务器使用。否则，该路径是 Windows 或 Macintosh 绝对路径。若要输入本地 XML 文件的名称，您必须键入路径和名称。

如果指定 HTTP URL，则 FLV 文件将作为一个渐进式下载播放。如果您指定作为 RTMP URL 的某个 URL，则该 FLV 文件从 FCS 或 FVSS 流式加载。指向 XML 文件的 URL 也可以是来自 FCS 或 FVSS 的 FLV 文件流。

注意

在“内容路径”对话框中单击“确定”后，该组件将更新 cuePoints 参数的值，因为在内容路径更改的情况下它可能不再适用。因此，您可以丢失所有禁用的提示点，但不可以丢失 ActionScript 提示点。（如果新的 FLV 文件包含相同的提示点，则将不会丢失禁用的提示点，在您只更改路径时可能会发生这一情况。）因此，您可能要通过 ActionScript 而不是通过“提示点”对话框来禁用非 ActionScript 提示点。

您也可以为多个带宽指定说明如何播放多个 FLV 文件流的 XML 文件的位置。XML 文件使用同步多媒体集成语言 (SMIL) 来描述 FLV 文件。有关 XML SMIL 文件的说明，请参见第 653 页的“使用 SMIL 文件”。

您还可以通过使用 ActionScript FLVPlayback.contentPath 属性以及 FLVPlayback.play() 和 FLVPlayback.load() 方法，指定 FLV 文件的名称和位置。这三个替代方法优先于“组件”检查器中的 contentPath 参数。有关更多信息，请参见第 533 页的 FLVPlayback.contentPath、第 570 页的 FLVPlayback.play() 和第 555 页的 FLVPlayback.load()。

## FLV 文件选项

“内容路径”对话框也具有两个选项。第一个选项是“匹配源 FLV 尺寸”，它指定舞台上的 FLVPlayback 实例是否应匹配源 FLV 文件的尺寸。源 FLV 文件包含用于播放的首选高、宽尺寸。如果您选择第一个选项，将调整 FLVPlayback 实例的尺寸以匹配这些首选尺寸。但是，只有在同时选中了第二个选项的情况下，此选项才可用。

第二个选项是“为提示点和尺寸下载 FLV”，只有在内容路径是 HTTP 或 RTMP URL 的情况下才启用该选项，这意味着该 FLV 文件不是本地的。任何不是以 .flv 结尾的路径都被认为是网络路径，因为它必须是 XML 文件并可以指向位于任何位置的 FLV 文件。此选项指定以下内容：为了获取 FLV 文件尺寸以及在该文件中嵌入的任何提示点定义，是下载还是流式加载该 FLV 文件的某一部分。Flash 使用尺寸来调整 FLVPlayback 实例的大小，并且它将提示点定义加载到“组件”检查器的 cuePoints 参数中。如果未选择此选项，则禁用第一个选项。

## 使用提示点

提示点是一个点，在播放 FLV 文件时，视频播放器在该点处调度一个 cuePoint 事件。您可以在想要与网页上的其它元素交互时向 FLV 文件添加提示点。例如，您可能想要显示文本或图形，或者要与 Flash 动画同步，或者要影响 FLV 文件的播放，具体为暂停 FLV 文件、搜索到不同时间点或切换到不同 FLV 文件。提示点允许您接收 ActionScript 代码中的控制并将 FLV 文件中的这些点与网页上的其它动作同步。

有三种类型的提示点：导航、事件和 ActionScript。导航提示点和事件提示点也称作嵌入式提示点，因为它们嵌入在 FLV 文件流和 FLV 文件的元数据包中。

导航提示点允许您搜索到 FLV 文件中的特定帧，因为它在尽可能接近您指定的时间在 FLV 文件内创建关键帧。关键帧是在 FLV 文件流的图像帧之间出现的数据段。在您搜索到导航提示点时，组件将搜索到该关键帧并启动 cuePoint 事件。

事件提示点使您能够将 FLV 文件内的时间点与网页上的外部事件同步。cuePoint 事件在指定的时间精确发生。您可以使用视频导入向导或 Flash 视频编码器，在 FLV 文件中嵌入导航提示点和事件提示点。有关视频导入向导和 Flash 视频编码器的更多信息，请参见《使用 Flash》中的第 11 章“使用视频”。

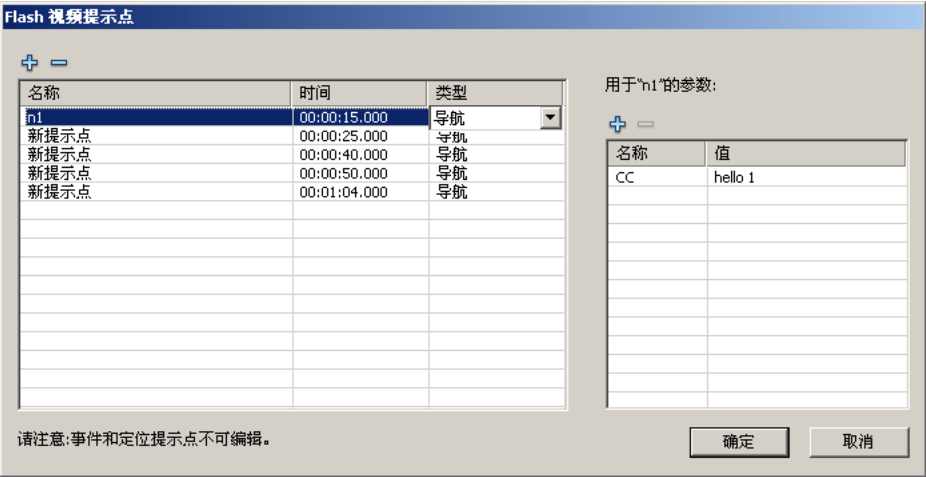
ActionScript 提示点是一种外部提示点，您可以通过组件的“Flash 视频提示点”对话框或通过 FLVPlayback.addASCuePoint() 方法添加。该组件在 FLV 文件之外存储和跟踪 ActionScript 提示点，因此，这些提示点在精确性上要低于嵌入式提示点。ActionScript 提示点精确度为十分之一秒。您可以通过降低 playheadUpdateInterval 属性值来提高 ActionScript 提示点的精确度，因为组件在播放头更新时会为 ActionScript 提示点生成 cuePoint 事件。有关更多信息，请参见第 576 页的“FLVPlayback.playheadUpdateInterval”。

在 **ActionScript** 中和 **FLV** 文件的元数据内，提示点表示为具有以下属性的对象：`name`、`time`、`type` 和 `parameters`。`name` 属性是一个字符串，其中包含为提示点分配的名称。`time` 属性是一个数字，表示提示点的发生时间，用小时、分钟、秒和毫秒 (**HH:MM:SS.mmm**) 表示。`type` 属性是一个字符串，根据您创建的提示点的类型，此字符串可以是 `"navigation"`、`"event"` 或 `"actionscript"`。`parameters` 属性是由指定的名称 - 值对组成的数组。

在发生 `cuePoint` 事件时，可以通过 `info` 属性在事件对象中提供提示点对象。有关更多信息，请参见第 477 页的“侦听 `cuePoint` 事件”。

## 使用“Flash 视频提示点”对话框

通过在“组件”检查器中双击 `cuePoints` 参数的 **Value** 单元格，可以打开“Flash 视频提示点”对话框。对话框类似于下图：



该对话框显示嵌入式提示点和 **ActionScript** 提示点。可以使用此对话框添加和删除 **ActionScript** 提示点以及提示点参数。还可以启用或禁用嵌入式提示点。但是，不能添加、更改或删除嵌入式提示点。

### 添加 **ActionScript** 提示点：

1. 双击“组件”检查器中 `cuePoints` 参数的 **Value** 单元格以打开“Flash 提示点”对话框。
2. 单击在提示点列表之上、位于左上角的加号 (+)，以添加默认的 **ActionScript** 提示点条目。
3. 单击“名称”列中的“新提示点”文本，并编辑该文本以命名提示点。
4. 单击“时间”值 `00:00:00:000` 以编辑它，并且为要发生的提示点分配一个时间。您可以按小时、分钟、秒和毫秒 (**HH:MM:SS.mmm**) 指定该时间。

如果存在多个提示点，该对话框会根据时间顺序，将新提示点移到列表中相应的位置。

- 5. 若要为所选提示点添加参数，请单击“参数”部分之上的加号 (+)，然后在“名称”列和“值”列中输入值。对每个参数均重复此步骤。
- 6. 若要添加多个 **ActionScript** 提示点，请为每个提示点重复第 2 步到第 5 步。
- 7. 单击“确定”以保存更改。

**删除 **ActionScript** 提示点：**

- 1. 双击“组件”检查器中 `cuePoints` 参数的 **Value** 单元格以打开“Flash 提示点”对话框。
- 2. 选择您要删除的提示点。
- 3. 单击提示点列表之上、位于左上角的减号 (-)，以删除它。
- 4. 为要删除的每个提示点重复第 2 步和第 3 步。
- 5. 单击“确定”以保存更改。

**启用或禁用嵌入式 FLV 文件提示点：**

- 1. 双击“组件”检查器中 `cuePoints` 参数的 **Value** 单元格以打开“Flash 提示点”对话框。
- 2. 选择您要启用或禁用的提示点。
- 3. 单击“类型”列中的值以触发弹出菜单，或者单击下箭头。
- 4. 单击提示点类型的名称（如“事件”或“导航”）即可启用它。单击“禁用”可以禁用它。
- 5. 单击“确定”以保存更改。

## 在 **ActionScript** 中使用提示点

您可以使用 **ActionScript** 添加 **ActionScript** 提示点、侦听 `cuePoint` 事件、查找任何类型或特定类型的提示点、搜索到导航提示点、启用或禁用提示点、检查是否启用了某个提示点以及删除提示点。

本节中的示例使用名为 `cuepoints.flv` 的 FLV 文件，它包含以下三个提示点：

名称	时间	类型
point1	00:00:00.418	导航
point2	00:00:07.748	导航
point3	00:00:16.020	导航



## 添加 ActionScript 提示点

您可以使用 `addASCuePoint()` 方法将 **ActionScript** 提示点添加到 **FLV** 文件。以下示例在 **FLV** 文件可供播放时向该 **FLV** 文件添加两个 **ActionScript** 提示点。该示例使用提示点对象添加第一个提示点，在其属性中指定该提示点的时间、名称和类型。第二个调用使用该方法的时间 `time` 和 `name` 参数指定时间和名称。

```
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv"
var cuePt:Object = new Object(); // 创建提示点对象
cuePt.time = 2.02;
cuePt.name = "ASpt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
// 使用 time 和 name 参数添加第二个 AS 提示点
my_FLVPlybk.addASCuePoint(5, "ASpt2");
```

有关更多信息，请参见第 508 页的 `FLVPlayback.addASCuePoint()`。

## 侦听 cuePoint 事件

`cuePoint` 事件允许您在发生 `cuePoint` 事件时接收 **ActionScript** 代码中的控制。在以下示例中，当提示点出现时，`cuePoint` 侦听器会调用一个事件处理函数，该事件处理函数显示 `playheadTime` 属性的值以及提示点的名称和类型。

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 trace("Elapsed time in seconds: " + my_FLVPlybk.playheadTime);
 trace("Cue point name is: " + eventObject.info.name);
 trace("Cue point type is: " + eventObject.info.type);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

有关 `cuePoint` 事件的更多信息，请参见第 534 页的 `FLVPlayback.cuePoint`。

## 查找提示点

通过使用 **ActionScript**，您可以找到任何类型的提示点、找到与某个时间最接近的提示点或者使用具体名称找到下一个提示点。

以下示例中的 `ready` 事件处理函数调用 `findCuePoint()` 方法以找到提示点 `ASpt1`，然后调用 `findNearestCuePoint()` 方法以找到最接近提示点 `ASpt1` 的时间的导航提示点：

```
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv"
var rtn_obj:Object = new Object(); // 创建提示点对象
my_FLVPlybk.addASCuePoint(2.02, "ASpt1"); // 添加 AS 提示点
var listenerObject:Object = new Object();
```

```

listenerObject.ready = function(eventObject:Object):Void {
 rtn_obj = my_FLVPlybk.findCuePoint("ASpt1");
 traceit(rtn_obj);
 rtn_obj = my_FLVPlybk.findNearestCuePoint(rtn_obj.time,
 FLVPlayback.NAVIGATION);
 traceit(rtn_obj);
}
my_FLVPlybk.addEventListener("ready", listenerObject);
function traceit(cuePoint:Object):Void {
 trace("Cue point name is: " + cuePoint.name);
 trace("Cue point time is: " + cuePoint.time);
 trace("Cue point type is: " + cuePoint.type);
}

```

在以下示例中，ready 事件处理函数查找提示点 ASpt，并且调用 findNextCuePointWithName() 方法以找到具有相同名称的下一个提示点：

```

import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv"
var rtn_obj:Object = new Object(); // 创建提示点对象
my_FLVPlybk.addASCuePoint(2.02, "ASpt"); // 添加 AS 提示点
my_FLVPlybk.addASCuePoint(3.4, "ASpt"); // 添加第二个 AS 提示点
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 rtn_obj = my_FLVPlybk.findCuePoint("ASpt");
 traceit(rtn_obj);
 rtn_obj = my_FLVPlybk.findNextCuePointWithName(rtn_obj);
 traceit(rtn_obj);
}
my_FLVPlybk.addEventListener("ready", listenerObject);
function traceit(cuePoint:Object):Void {
 trace("Cue point name is: " + cuePoint.name);
 trace("Cue point time is: " + cuePoint.time);
 trace("Cue point type is: " + cuePoint.type);
}

```

有关查找提示点的更多信息，请参见第 539 页的 [FLVPlayback.findCuePoint\(\)](#)、第 542 页的 [FLVPlayback.Findnearestcuepoint\(\)](#) 和第 544 页的 [FLVPlayback.findNextCuePointWithName\(\)](#)。

## 搜索导航提示点

您可以搜索到导航提示点、搜索到指定时间的下一个导航提示点以及搜索到指定时间的前一个导航提示点。以下示例播放 FLV 文件 **cuepoints.flv**，并在发生 **ready** 事件时搜索到位于 **7.748** 的提示点。在发生 **cuePoint** 事件时，该示例调用 `seekToPrevNavCuePoint()` 方法以搜索到第一个提示点。在 **cuePoint** 事件发生时，该示例调用 `seekToNextNavCuePoint()` 方法，以通过向 `eventObject.info.time`（这是当前提示点的时间）添加 **10** 秒来搜索最后一个提示点。

```
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
 my_FLVPlayback.seekToNavCuePoint("point2");
}
my_FLVPlayback.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
 trace(eventObject.info.time);
 if(eventObject.info.time == 7.748)
 my_FLVPlayback.seekToPrevNavCuePoint(eventObject.info.time - .005);
 else
 my_FLVPlayback.seekToNextNavCuePoint(eventObject.info.time + 10);
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
my_FLVPlayback.contentPath = "http://helpexamples.com/flash/video/
 cuepoints.flv";
```

有关更多信息，请参见第 607 页的 `FLVPlayback.seekToNavCuePoint()`、第 609 页的 `FLVPlayback.seekToNextNavCuePoint()` 和第 610 页的 `FLVPlayback.seekToPrevNavCuePoint()`。

## 启用和禁用嵌入式 FLV 文件提示点

您可以通过 `setFLVCuePointEnabled()` 方法启用和禁用嵌入式 FLV 文件提示点。禁用的提示点不触发 **cuePoint** 事件，并且不与 `seekToCuePoint()`、`seekToNextNavCuePoint()` 和 `seekToPrevNavCuePoint()` 方法一起使用。不过，您可以使用 `findCuePoint()`、`findNearestCuePoint()` 和 `findNextCuePointWithName()` 方法找到禁用的提示点。

您可以使用 `isFLVCuePointEnabled()` 方法来测试是否启用了嵌入式 **FLV** 文件提示点。以下示例在视频准备播放时禁用嵌入式提示点 `point2` 和 `point3`。但在发生第一个 `cuePoint` 事件时，事件处理函数进行测试以确定提示点 `point3` 是否被禁用，如果确定该提示点被禁用，则启用它。

```
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVPlybk.setFLVCuePointEnabled(false, "point2");
 my_FLVPlybk.setFLVCuePointEnabled(false, "point3");
}
my_FLVPlybk.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 trace("Cue point time is: " + eventObject.info.time);
 trace("Cue point name is: " + eventObject.info.name);
 trace("Cue point type is: " + eventObject.info.type);
 if (my_FLVPlybk.isFLVCuePointEnabled("point2") == false) {
 my_FLVPlybk.setFLVCuePointEnabled(true, "point2");
 }
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

有关更多信息，请参见第 551 页的 [FLVPlayback.isFLVCuePointEnabled\(\)](#) 和第 613 页的 [FLVPlayback.setFLVCuePointEnabled\(\)](#)。

## 删除 **ActionScript** 提示点

您可以使用 `removeASCuePoint()` 方法删除 **ActionScript** 提示点。以下示例在提示点 `ASpt1` 发生时删除提示点 `ASpt2`：

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 trace("Cue point name is: " + eventObject.info.name);
 if (eventObject.info.name == "ASpt1") {
 my_FLVPlybk.removeASCuePoint("ASpt2");
 trace("Removed cue point ASpt2");
 }
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

有关更多信息，请参见第 586 页的 [FLVPlayback.removeASCuePoint\(\)](#)。

# 播放多个 FLV 文件

您只需通过在前一个 FLV 文件完成播放时在 `contentPath` 属性中加载新的 URL，就可以在 **FLVPlayback** 实例中连续播放多个 FLV 文件。例如，以下 **ActionScript** 代码侦听 `complete` 事件，该事件在 FLV 文件完成播放后发生。在发生此事件时，代码在 `contentPath` 属性中设置新 FLV 文件的名称和位置，并调用 `play()` 方法以播放新视频。

```
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv";
var listenerObject:Object = new Object();
// 侦听 complete 事件; 播放新的 FLV
listenerObject.complete = function(eventObject:Object):Void {
 if (my_FLVPlayback.contentPath == "http://www.helpexamples.com/flash/video/
 clouds.flv") {
 my_FLVPlayback.play("http://www.helpexamples.com/flash/video/water.flv");
 }
};
my_FLVPlayback.addEventListener("complete", listenerObject);
```

## 使用多个视频播放器

您还可以在 **FLVPlayback** 组件的单个实例内打开多个视频播放器，以播放多个视频并在这些视频播放时在它们之间切换。

您在将 **FLVPlayback** 组件拖到舞台上时创建初始视频播放器。该组件自动为初始视频播放器分配数字 **0**，并使其成为默认播放器。若要创建其它视频播放器，只需将 `activeVideoPlayerIndex` 属性设置为新的数字。通过设置 `activeVideoPlayerIndex` 属性，还可以使指定的视频播放器成为活动视频播放器，**FLVPlayback** 类的属性和方法将会影响该播放器。但是，设置 `activeVideoPlayerIndex` 属性并不会使视频播放器可见。若要使视频播放器可见，请将 `visibleVideoPlayerIndex` 属性设置为该视频播放器的编号。有关这些属性如何与 **FLVPlayback** 类的方法和属性交互的更多信息，请参见 [第 505 页的 `FLVPlayback.activeVideoPlayerIndex`](#) 和 [第 632 页的 `FLVPlayback.visibleVideoPlayerIndex`](#)。

以下 **ActionScript** 代码加载 `contentPath` 属性,以便在默认视频播放器中播放 **FLV** 文件并为其添加提示点。在发生 `ready` 事件时,事件处理函数通过将 `activeVideoPlayerIndex` 属性设置为数字 **1**,打开另一个视频播放器。它为第二个视频播放器指定 **FLV** 文件和提示点,然后再次使默认播放器 (**0**) 成为活动的视频播放器。

```
/**
 * 要求:
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
// 向默认播放器添加提示点
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv";
my_FLVPlayback.addASCuePoint(3, "1st_switch");
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 // 添加另一个视频播放器并为其创建提示点
 my_FLVPlayback.activeVideoPlayerIndex = 1;
 my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
 my_FLVPlayback.addASCuePoint(3, "2nd_switch");
 my_FLVPlayback.activeVideoPlayerIndex = 0;
};
my_FLVPlayback.addEventListener("ready", listenerObject);
```

若要在播放一个 **FLV** 文件时切换到另一个 **FLV** 文件,必须获取控制以在您的 **ActionScript** 代码中进行该切换。提示点允许您使用 `cuePoint` 事件在 **FLV** 文件中介入特定点。以下代码为 `cuePoint` 事件创建一个侦听器,并且调用用于暂停活动视频播放器 (**0**) 的处理函数,切换到第二个播放器 (**1**),然后播放其 **FLV** 文件:

```
// 创建一个侦听器对象
var listenerObject:Object = new Object();
// 为 cuePoint 事件添加处理函数
listenerObject.cuePoint = function(eventObject:Object):Void {
 // 显示导致该事件的视频播放器的编号
 trace("Hit cuePoint event for player: " + eventObject.vp);
 // 测试视频播放器并相应地切换 FLV
 if (eventObject.vp == 0) {
 my_FLVPlayback.pause(); // 暂停第一个 FLV
 my_FLVPlayback.activeVideoPlayerIndex = 1; // 使第二个播放器成为活动播放器
 my_FLVPlayback.visibleVideoPlayerIndex = 1; // 使第二个播放器可见
 my_FLVPlayback.play(); // 开始播放新的播放器 /FLV
 } else if (eventObject.vp == 1) {
 my_FLVPlayback.pause(); // 暂停第二个 FLV
 my_FLVPlayback.activeVideoPlayerIndex = 0; // 使第一个播放器成为活动播放器
 my_FLVPlayback.visibleVideoPlayerIndex = 0; // 使第一个播放器可见
 my_FLVPlayback.play(); // 开始播放第一个播放器
 }
}
```

```
// 为 cuePoint 事件添加侦听器
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
 trace("Hit complete event for player: " + eventObject.vp);
 if (eventObject.vp == 0) {
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.visibleVideoPlayerIndex = 1;
 my_FLVPlybk.play();
 } else {
 my_FLVPlybk.closeVideoPlayer(1);
 }
}
my_FLVPlybk.addEventListener("complete", listenerObject);
```

在您创建新的视频播放器时，**FLVPlayback** 实例将其属性设置为默认视频播放器的值，但 `contentPath`、`totalTime` 和 `isLive` 属性除外，**FLVPlayback** 实例始终将它们设置为以下默认值：默认值分别为空字符串、**0** 和 `false`。该实例将 `autoPlay` 属性（对于默认视频播放器，默认为 `true`）设置为 `false`。`cuePoints` 属性没有任何影响，并且不影响以后加载到默认视频播放器中。

控制音量、位置、尺寸、可见性和用户界面控件的方法和属性始终是全局的，并且设置 `activeVideoPlayerIndex` 属性后并不影响它们的行为。有关这些方法和属性的更多信息以及设置 `activeVideoPlayerIndex` 属性的影响，请参见[第 505 页的 `FLVPlayback.activeVideoPlayerIndex`](#)。其余的属性和方法针对 `activeVideoPlayerIndex` 属性值所标识的视频播放器。

但是，控制尺寸的属性和方法与 `visibleVideoPlayerIndex` 属性相互影响。有关更多信息，请参见[第 632 页的“`FLVPlayback.visibleVideoPlayerIndex`”](#)。

## 从 FCS 流式加载 FLV 文件

如果您使用 FCS 将 FLV 文件流式加载到 **FLVPlayback** 组件，则必须将 `main.asc` 文件添加到 Flash Communication Server FLV 应用程序。您可以在 `Flash 8/Samples` 和 `Tutorials/Samples/Components/FLVPlayback/main.asc` 下的 Flash 8 应用程序文件夹下找到该 `main.asc` 文件。

设置 FCS 以便流式加载 FLV 文件：

1. 在您的 FCS 应用程序文件夹中创建一个文件夹，将其命名为 **my\_application** 之类的名称。
2. 将 main.asc 文件复制到 my\_application 文件夹中。
3. 在 my\_application 文件夹中创建名为 **streams** 的文件夹。
4. 在 streams 文件夹中创建名为 **\_definst\_** 的文件夹。
5. 将您的 FLV 文件放置于 **\_definst\_** 文件夹中。

若要访问 Flash Communication Server 上的 FLV 文件，请使用 `rtmp://my_servername/my_application/stream.flv` 之类的 URL。

有关管理 Flash Communication Server 的更多信息，包括如何设置实时流，请参见位于 [www.macromedia.com/support/documentation/en/flashcom/](http://www.macromedia.com/support/documentation/en/flashcom/) 的 FCS 文档。在使用 FCS 播放实时流时，需要将 FLVPlayback 属性 `isLive` 设置为 `true`。有关更多信息，请参见第 553 页的 `FLVPlayback.isLive`。

## 自定义 FLVPlayback 组件

本部分说明如何自定义 FLVPlayback 组件。有关自定义组件的全面概述（包括使用样式、外观、主题的术语和基本概念），请参见《使用组件》中的“自定义组件”。但是，用于自定义其它组件的大多数方法并不使用 FLVPlayback 组件。若要自定义 FLVPlayback 组件，请只使用在此节中描述的技术。

以下选项可供您自定义 FLVPlayback 组件：选择预先设计的外观，分别设置 FLV 回放自定义用户界面组件的外观，或者创建新外观。您还可以使用 FLVPlayback 属性修改外观的行为。

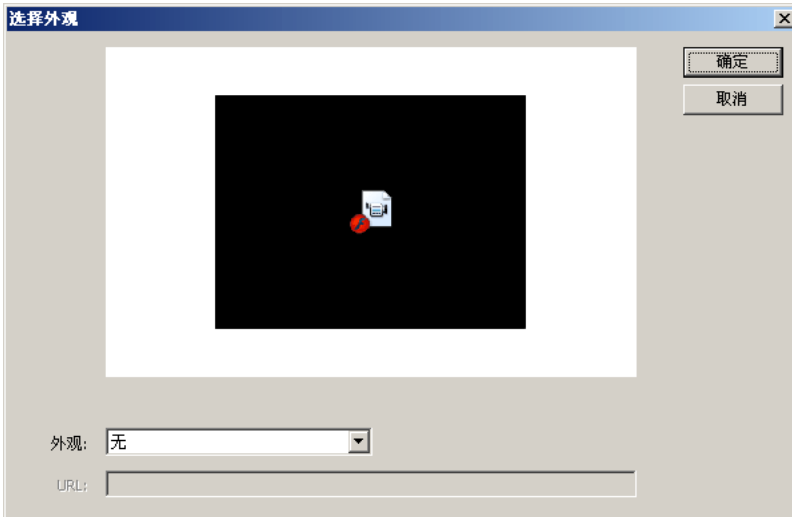


为使外观可使用您的 FLVPlayback 组件，您必须将外观 SWF 文件与应用程序 SWF 文件一起上载到 Web 服务器。



## 选择预先设计的外观

通过单击“组件”检查器中 `skin` 参数的 `value` 单元格，可以选择 `FLVPlayback` 组件的外观。然后，单击放大图标以打开以下“选择外观”对话框，从中可以选择一种外观或者提供用于指定外观 `SWF` 文件位置的 `URL`。



在“外观”弹出菜单中列出的外观位于 `Flash 8 Configuration/Skins` 文件夹或用户的本地 `Configuration/Skins` 文件夹中。通过创建新外观并将 `SWF` 文件放置在 `Skins` 文件夹中，可以使新外观在此对话框中可用。该名称在弹出菜单中出现，并且具有 `.swf` 扩展名。有关创建外观集的更多信息，请参见第 491 页的“创建新外观”。

如果您想要使用 `FLV` 回放自定义用户界面设置 `FLVPlayback` 组件的外观，请从弹出菜单中选择“无”。

## 单独设置 `FLV` 回放自定义用户界面组件的外观

`FLV` 回放自定义用户界面组件允许您在 `FLA` 文件内自定义 `FLVPlayback` 控件的外观，并且允许您在预览网页时看到结果。但是，根据设计，这些组件无法缩放。您应该编辑影片剪辑及其内容，使其具有特定大小。因此，一般而言，通常最好在舞台上具有所需大小的 `FLVPlayback` 组件，并且将 `autoSize` 和 `maintainAspectRatio` 属性设置为 `false`。

若要开始，只需将所需 `FLV` 回放自定义用户界面组件从“组件”面板中拖出，将它们放置于舞台上的适当位置，并且在“属性”检查器中为每个组件指定一个实例名称。当组件位于舞台上后，即可像对待任何其它元件一样编辑它们。

在打开这些组件后，可以看到各个组件之间在设置上都存在微小差异。

## 按钮组件

按钮组件具有类似的结构。按钮包括 `BackButton`、`ForwardButton`、`MuteButton`、`PauseButton`、`PlayButton`、`PlayPauseButton` 和 `StopButton`。大多数按钮都在第 1 帧上具有实例名称为 `placeholder_mc` 的单个影片剪辑。这通常是用于按钮的正常状态的实例，但不是必须要具有它。在第 2 帧的舞台上，每个显示状态都有四个影片剪辑：正常、指针经过、按下和禁用。（在运行时，组件永远不会实际转到第 2 帧；这些影片剪辑放置于此处是为了使编辑更方便，并且强制它们加载到 SWF 文件中，而不必在“元件属性”对话框内选中“在第一帧导出”复选框。不过，您仍然必须选择“为 **ActionScript** 导出”选项。）

为了设置按钮的外观，只需编辑上述各影片剪辑即可。您可以更改其大小及外观。

某些 **ActionScript** 通常在第 1 帧中出现。您应该不需要更改此脚本。它只停止第 1 帧上的播放头，并且指定哪些影片剪辑将用于哪些状态。

### PlayPauseButton 和 MuteButton 按钮

`PlayPauseButton` 和 `MuteButton` 按钮在设置上与其它按钮不同；它们只具有带两个图层且没有脚本的一个帧。在该帧上，有两个按钮，彼此叠放。在第一个示例中，这两个按钮是 `Play` 按钮和 `Pause` 按钮；在第二个示例中，这两个按钮是 `Mute-on` 按钮和 `Mute-off` 按钮。若要设置 `PlayPauseButton` 或 `MuteButton` 按钮的外观，请按照第 485 页的“单独设置 FLV 回放自定义用户界面组件的外观”中所述分别设置这两个内部按钮的外观；不需要进行额外的操作。

### BackButton 和 ForwardButton 按钮

`BackButton` 和 `ForwardButton` 按钮在设置上也与其它按钮不同。在第 2 帧上，它们具有额外的影片剪辑，您可以将它们作为帧使用，并环绕这两个按钮或其中一个按钮。这些影片剪辑不是必需的，并且没有特殊功能；只是出于方便目的才提供它们。若要使用它们，只需将它们从“库”面板拖到舞台上，然后将它们放置于所需位置。如果您不想使用它们，或者不使用它们，或者从“库”面板中删除它们。

提供的大多数按钮都基于影片剪辑的公共集，以便您可以一次更改所有按钮的外观。您可以使用此功能，或者可以替换这些公共剪辑，并使每个按钮在外观上不同。

## BufferingBar 组件

该缓冲栏组件十分简单：它由一个动画组成，该动画在组件进入缓冲状态时出现，并且它不需要任何特殊的 **ActionScript** 来配置它。默认情况下，它是一个从左向右移动的有斑纹的条，在该条上有一个矩形遮罩，使其呈现“理发店招牌”效果，但对于此配置没有什么特殊设置。

尽管外观 SWF 文件中的缓冲栏使用 9 切片缩放（因为它们需要在运行时缩放），但 `BufferingBar` FLV 自定义用户界面组件没有并且无法使用 9 切片缩放，因为它嵌套了影片剪辑。如果您想使 `BufferingBar` 更宽或更高，则可能需要更改其内容，而不是对其进行缩放。

## SeekBar 和 VolumeBar 组件

**SeekBar** 和 **VolumeBar** 组件十分相似，尽管它们在功能上不同。它们都具有手柄，都使用相同的手柄跟踪机制，并且都支持在其中嵌套剪辑以跟踪进度和完成程度。

在许多地方，**FLVPlayback** 组件中的 **ActionScript** 代码都假定 **SeekBar** 或 **VolumeBar** 组件的注册点位于内容的左上角，因此，维持这一惯例十分重要。否则，手柄以及进度和完成程度影片剪辑可能会有问题。

尽管外观 **SWF** 文件中的搜索栏使用 9 切片缩放（因为它们需要在运行时缩放），但 **SeekBar** **FLV** 自定义用户界面组件没有并且无法使用 9 切片缩放，因为它嵌套了影片剪辑。如果您想使 **SeekBar** 更宽或更高，则可能需要更改其内容，而不是缩放其大小。

### 手柄

手柄影片剪辑的一个实例位于第 2 帧。与 **BackButton** 和 **ForwardButton** 组件一样，该组件永远不会实际转到第 2 帧；这些影片剪辑放置于此是为了使编辑更方便，并且以这种方式强制它们加载到 **SWF** 文件中，而不必在“元件属性”对话框中选中“在第一帧导出”复选框。不过，您仍然必须选择“为 **ActionScript** 导出”选项。

您可注意到，手柄影片剪辑在背景中具有一个 **Alpha** 设置为 0 的矩形。此矩形增加了该手柄的点击区域的大小，这样，不必更改手柄外观就可以更容易地控制手柄，这与按钮的点击状态类似。因为该手柄是在运行时动态创建的，所以它必须是一个影片剪辑，而不是一个按钮。对于任何其它原因，**Alpha** 设置为 0 的这一矩形都不是必需的；通常，您可以用所需任何图像替代手柄内的内容。不过，如果将注册点保持在手柄影片剪辑的中部水平居中，则它的使用效果最好。

以下 **ActionScript** 代码位于 **SeekBar** 组件的第 1 帧上，用于管理手柄：

```
stop();
handleLinkageID = "SeekBarHandle";
handleLeftMargin = 2;
handleRightMargin = 2;
handleY = 11;
```

由于第 2 帧的内容，对 **stop()** 函数的调用是必需的。

第二行指定哪个元件要用作手柄；并且，如果您只编辑第 2 帧上的手柄影片剪辑实例，则应该无需对此进行更改。在运行时，**FLVPlayback** 组件将舞台上指定的影片剪辑的实例作为 **Bar** 组件实例的同级创建，这意味着它们具有相同的父影片剪辑。因此，如果您的栏位于根级别，则手柄也必须位于根级别。

变量 **handleLeftMargin** 确定手柄的原始位置 (0%)，而变量 **handleRightMargin** 确定在结束时它所处的位置 (100%)。这两个数值提供与栏控件左端和右端的偏移量，正数值标记栏内部的限制，负数值标记栏外部的限制。这两个偏移量根据其注册点指定手柄可以转到的位置。如果您将注册点放置于手柄的中间，则该手柄的左侧和右侧将越过边界。为了正常使用，搜索栏影片剪辑必须将其注册点放置于其内容的左上角。

变量 `handleY` 确定该手柄相对于栏实例的 `y` 位置。这基于每个影片剪辑的注册点。范例手柄的注册点位于三角形的尖端，以便相对于可见部分放置它，而不考虑不可见的点击状态矩形。此外，为了正常使用，栏影片剪辑必须将其注册点保持在其内容的左上角。

因此，以这些限制为例，如果在 (100, 100) 位置设置了某个栏控件，并且其宽度为 100 个像素，则手柄在水平方向上的范围可从 102 到 198，并且在垂直方向上保持 111。如果您将 `handleLeftMargin` 和 `handleRightMargin` 更改为 -2，并将 `handleY` 更改为 -11，则手柄在水平方向上的范围可从 98 到 202，并且在垂直方向上保持 89。

## 进度和完成程度影片剪辑

**SeekBar** 组件具有进度影片剪辑，**VolumeBar** 具有完成程度影片剪辑。但在实践中，任何 **SeekBar** 或 **VolumeBar** 都可以不具有或者同时具有这两个影片剪辑，或者具有其中一个影片剪辑。它们在结构上相同，并在行为上类似，但跟踪不同的值。进度影片剪辑随着 FLV 文件下载而填充（这仅适用于 HTTP 下载，因为如果从 FCS 进行流式加载，它始终是满的），而完成程度影片剪辑随着手柄从左向右移动而填充。

**FLVPlayback** 组件通过查找特定的实例名称找到这些影片剪辑实例，因此，您的进度影片剪辑实例必须使栏影片剪辑作为其父级，并且具有实例名称 `progress_mc`。完成程度影片剪辑实例必须具有实例名称 `fullness_mc`。

您在设置进度和完成程度影片剪辑时，在其内部嵌套 `fill_mc` 影片剪辑实例与否均可。

**VolumeBar** `fullness_mc` 影片剪辑使用 `fill_mc` 影片剪辑显示该方法，**SeekBar** `progress_mc` 影片剪辑则不使用 `fill_mc` 影片剪辑而显示该方法。

如果您想要进行的填充不通过扭曲外观无法缩放，则在内部嵌套了 `fill_mc` 影片剪辑的方法十分有用。

在 **VolumeBar** `fullness_mc` 影片剪辑中，嵌套的 `fill_mc` 影片剪辑实例被遮罩。您或者可以在创建影片剪辑时遮罩它，或者可以在运行时动态地创建遮罩。如果您用影片剪辑遮罩它，则将该实例命名为 `mask_mc` 并对它进行设置，以便 `fill_mc` 显示为就像百分比为 100% 时一样。如果您不遮罩 `fill_mc`，则动态创建的遮罩将是矩形，并且在 100% 时与 `fill_mc` 的大小相同。

`fill_mc` 影片剪辑通过两种方法之一与遮罩一起显示，具体方法取决于 `fill_mc.slideReveal` 是 `true` 还是 `false`。

如果 `fill_mc.slideReveal` 是 `true`，则 `fill_mc` 从左向右移动，以通过遮罩显示它。在 0% 处，它自始至终向左，因此不会有任何部分通过遮罩显示。随着百分比的增加，它向右移动，直到达到 100%，此时它它位于舞台上的创建位置之后。

如果 `fill_mc.slideReveal` 是 `false` 或未定义（默认行为），则遮罩将从左向右调整大小，以显示 `fill_mc` 的更多部分。在它达到 0% 时，遮罩就在水平方向上缩放到 05，并且随着百分比的增加，`_xscale` 也将增加，直到达到 100%，这时它将显示所有 `fill_mc`。`_xscale = 100` 并不是必需的，因为在创建 `mask_mc` 时可能已对它进行了缩放。

没有 `fill_mc` 的方法比具有 `fill_mc` 的方法更简单，但它水平扭曲填充。如果您不进行该扭曲，则必须使用 `fill_mc`。`SeekBar progress_mc` 阐释了这一方法。

进度或完成程度影片剪辑根据百分比进行水平缩放。在 0% 处，实例的 `_xscale` 被设置为 0，这使其不可见。随着该百分比增加，将对 `_xscale` 进行调整，直到达到 100%，此时该影片剪辑的大小将与创建它时在舞台上的大小相同。同样，`_xscale = 100` 并不是必需的，因为在创建该影片剪辑实例时可能已对它进行了缩放。

## 连接您的 FLV 回放自定义用户界面组件

您必须编写 `ActionScript` 代码以便将您的 `FLV` 回放自定义用户界面组件连接到 `FLVPlayback` 组件的实例。首先，必须向 `FLVPlayback` 实例分配一个名称，然后使用 `ActionScript` 将您的 `FLV` 回放自定义用户界面组件实例分配给相应的 `FLVPlayback` 属性。在以下示例中，`FLVPlayback` 实例是 `my_FLVPlybk`，`FLVPlayback` 属性名称后跟有句点 (`.`)，并且 `FLV` 回放自定义用户界面控件实例位于等号 (`=`) 的右侧：

```
// FLVPlayback 实例 = my_FLVPlybk
my_FLVPlybk.playButton = playbtn; // 将 playButton 属性设置为 playbtn，等等
my_FLVPlybk.pauseButton = pausebtn;
my_FLVPlybk.playPauseButton = playpausebtn;
my_FLVPlybk.stopButton = stopbtn;
my_FLVPlybk.muteButton = mutebtn;
my_FLVPlybk.backButton = backbtn;
my_FLVPlybk.forwardButton = forbtn;
my_FLVPlybk.volumeBar = volbar;
my_FLVPlybk.seekBar = seekbar;
my_FLVPlybk.bufferingBar = bufbar;
```

### 示例

以下几个步骤创建自定义的 `StopButton`、`PlayPauseButton`、`MuteButton` 和 `SeekBar` 控件：

1. 将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 `my_FLVPlybk`。
2. 通过“组件”检查器将 `contentPath` 参数设置为 `http://www.helpexamples.com/flash/video/cuepoints.flv`。
3. 将 `Skin` 参数设置为“无”。
4. 将 `StopButton`、`PlayPauseButton` 和 `MuteButton` 拖到舞台上，然后将它们放置于 `FLVPlayback` 实例上，并垂直堆叠在左侧。在“属性”检查器中为每个按钮提供一个实例名称（如 `my_stopbtn`、`my_plypausbtn` 和 `my_mutebtn`）。
5. 在“库”面板中，打开 `FLVPlayback Skins` 文件夹，然后打开该文件夹下面的 `SquareButton` 文件夹。
6. 选择 `SquareBgDown` 影片剪辑，并且双击它以在舞台上打开它。
7. 右击 (Windows) 或者按住 `Ctrl` 键单击 (Macintosh)，然后从菜单中选择“全部”，并删除该元件。

8. 选择椭圆工具，在相同位置上绘制一个椭圆，然后将填充设置为蓝色 # (0033FF)。
9. 在“属性”检查器中，将宽度 (W:) 设置为 40，将高度 (H:) 设置为 20。将 x 坐标 (X:) 设置为 0.0，将 y 坐标 (Y:) 设置为 0.0。
10. 对 SquareBgNormal 重复第 6 步到第 8 步，但将填充更改为黄色 (#FFFF00)。
11. 对 SquareBgOver 重复第 6 步到第 8 步，但将填充更改为绿色 (#006600)。
12. 编辑按钮内不同元件图标 (PauseIcon、PlayIcon、MuteOnIcon、MuteOffIcon 和 StopIcon) 的影片剪辑。您可以在 FLV Playback Skins/Label Button/Assets 下的“库”面板中找到这些影片剪辑，其中 Label 是按钮的名称，如“播放”、“暂停”等。为每个按钮执行以下步骤：
  - a. 选择“全选”选项。
  - b. 将颜色更改为红色 (#FF0000)。
  - c. 缩放 300%。
  - d. 将内容的 X: 位置更改为 7.0，以便更改图标在每个按钮状态中的水平位置。



通过用此方法更改该位置，您可以避免打开每个按钮状态和移动图标影片剪辑实例。

13. 单击时间轴上的蓝色“返回”箭头以返回到第 1 场景的第 1 帧。
14. 将一个 SeekBar 组件拖到该舞台上，并且将其放置在 FLVPlayback 实例的右下角。
15. 在“库”面板中，双击 SeekBar 以在舞台上打开它。
16. 将其缩放到 400%。
17. 选择轮廓，并且将颜色设置为红色 (#FF0000)。
18. 双击 FLVPlayback Skins/Seek Bar 文件夹中的 SeekBarProgress，并且将颜色设置为黄色 (#FFFF00)。
19. 双击 FLVPlayback Skins/Seek Bar 文件夹中的 SeekBarHandle，并且将颜色设置为红色 (#FF0000)。
20. 单击时间轴上的蓝色“返回”箭头以返回到第 1 场景的第 1 帧。
21. 选择舞台上的 SeekBar 实例，并为其指定实例名称 **my\_seekbar**。
22. 在时间轴第 1 帧的“动作”面板上，为 video 类添加一条 import 语句，并且将按钮和搜索栏名称分配给相应的 FLVPlayback 属性，如下示例所示：

```
import mx.video.*;
my_FLVPlayback.stopButton = my_stopbtn;
my_FLVPlayback.playPausebtn = my_playpausebtn;
my_FLVPlayback.muteButton = my_mutebtn;
my_FLVPlayback.seekBar = my_seekbar;
```
23. 同时按下 Ctrl+Enter 以测试该影片。

## 创建新外观

创建外观 SWF 文件的最佳方式是复制与 Flash 8 一起提供的外观文件之一，并且将它用作起始点。您可以在 < 语言代码 >/Configuration/SkinFLA 的 Flash 8 应用程序文件夹中找到这些外观的 FLA 文件。若要使您的已完成的外观 SWF 文件可用作“选择外观”对话框中的选项，请将它放置于 Flash 8 应用程序文件夹或用户的本地 Configuration/Skins 文件夹的 < 语言代码 >/Configuration/Skins 文件夹中。

您将发现，为更改按钮的外观或按钮的烙印（背景）的外观而进行的修改相当容易，不必更改尺寸。安装的所有外观都具有基于不同颜色烙印的相同按钮，因此，您只需更改烙印的颜色，即可进行生动的更改。您可以通过只移动占位符剪辑进行更改，例如在布局影片剪辑中重新安排控件。您可以看到这些更改与它们将在已完成的 SWF 文件中的显示完全一样。

在查看已安装的 Flash 8 外观 FLA 文件时，舞台上的某些项似乎不是必需的，但它们中的很多项被放置于引导图层中。若要快速看到在 SWF 文件中实际显示的内容，请按下 **Ctrl-Enter** 以测试该影片。这还将显示 9 切片缩放是如何影响某些控件的，因为在您进行创作时 9 切片缩放不起作用。

以下几节介绍一些对 SeekBar、BufferingBar 和 VolumeBar 影片剪辑的更复杂的自定义和更改。

## 使用 layout\_mc

在您打开 Flash 8 外观 FLA 文件时，将在舞台的左上角发现名为 layout\_mc 的影片剪辑。此剪辑必须被命名为 layout\_mc。您在同一帧上发现的 layout\_mc 剪辑和 ActionScript 代码将定义在运行时放置控件的方式。

尽管 layout\_mc 看起来很像外观在运行时的样子，但此剪辑的内容在运行时不可见。它只用于计算控件的放置位置。舞台上的其它控件将在运行时使用。

layout\_mc 内是用于名为 video\_mc 的 FLVPlayback 组件的占位符。所有其它控件都相对于 video\_mc 放置。如果您从某个 Flash 8 FLA 文件开始并且更改控件的大小，则很可能通过移动这些占位符剪辑修复布局。

每个占位符剪辑都具有特定的实例名称。占位符剪辑的名称是：playpause\_mc、play\_mc、pause\_mc、stop\_mc、back\_mc、bufferingBar\_mc、seekBar\_mc、volumeMute\_mc 和 volumeBar\_mc。

哪个剪辑用于控件并不重要。通常，对按钮使用正常状态剪辑。对于其它控件，使用该控件的剪辑，但这只是出于方便目的。所有重要事项就是 x（水平）和 y（垂直）位置以及占位符的高度和宽度。

您还可以在标准控件旁具有所需任何数目的背景和前景剪辑。但是，您必须使用以下命名约定: **bg1\_mc**、**bg2\_mc** 等用于背景剪辑; **fg1\_mc**、**fg2\_mc** 等用于前景剪辑。不能省略编号。例如，如果您具有 **bg1\_mc** 和 **bg3\_mc**，但没有 **bg2\_mc**，则不使用 **bg3\_mc**。此方案在设计上将背景剪辑放置于控件后，将 **bg1\_mc** 放置于底部并将 **bg2\_mc** 放置于其上，并将前景剪辑放置于控件上，首先是 **fg1\_mc**，然后在 **fg1\_mc** 上是 **fg2\_mc**，依此类推。但是，剪辑的分层关系实际由舞台上相应控件的顺序确定，以便确保它正确。

**bg1\_mc** 剪辑比较特殊。如果您将 `FLVPlayback.skinAutoHide` 属性设置为 `true`，则外观显示鼠标何时位于 **bg1\_mc** 剪辑上。这对于显示在视频播放器的边界外的外观十分重要。有关 `skinAutoHide` 属性的信息，请参见第 496 页的“修改外观行为”。

在 Flash 8 FLA 文件中，**bg1\_mc** 用于烙印，并且某些文件使用 **bg2\_mc** 用于环绕 Forward 和 Back 按钮的边界。

### ActionScript

以下 **ActionScript** 代码通常适用于所有控件。某些控件具有定义附加行为的特定 **ActionScript**，在针对该控件的那一节中将对对此加以说明。

初始的 **ActionScript** 代码定义外观的最小宽度和高度。“选择外观”对话框显示这些值，并且这些值在运行时使用，以防止外观缩放到低于其最小大小。如果您不想指定最小大小，则将其保留为未定义或者小于或等于零。

```
// 建议使用此外观的视频的最小宽度和高度
// 如果没有最小值，则保留为未定义或 <= 0
layout_mc.minWidth = 270;
layout_mc.minHeight = 60;
```

每个占位符都可以具有适用于它的以下属性:

属性	说明
<code>mc:MovieClip</code>	用于此控件的舞台上的实例。如果未设置， <code>layout_mc.foo_mc.mc</code> 将默认为 <code>foo_mc</code> 。
<code>anchorLeft:Boolean</code>	控件相对于 <code>FLVPlayback</code> 实例左侧的位置。默认值为 <code>true</code> （除非 <code>anchorRight</code> 被显式设置为 <code>true</code> ），然后它默认为 <code>false</code> 。
<code>anchorRight:Boolean</code>	控件相对于 <code>FLVPlayback</code> 实例的右侧的位置。默认值为 <code>false</code> 。
<code>anchorBottom:Boolean</code>	控件相对于 <code>FLVPlayback</code> 实例底部的位置。默认值为 <code>true</code> （除非 <code>anchorTop</code> 被显式设置为 <code>true</code> ），然后它默认为 <code>false</code> 。
<code>anchorTop:Boolean</code>	控件相对于 <code>FLVPlayback</code> 实例顶部的位置。默认值为 <code>false</code> 。



如果 `anchorLeft` 和 `anchorRight` 属性均为 `true`，则该控件将在运行时水平缩放。如果 `anchorTop` 和 `anchorBottom` 属性均为 `true`，则该控件将在运行时垂直缩放。

若要看到这些属性的效果，请查看它们在 **Flash 8** 外观中的使用方式。**BufferingBar** 和 **SeekBar** 控件是可以缩放的唯一控件，它们彼此层叠放置，并且都将 `anchorLeft` 和 `anchorRight` 属性设置为 `true`。**BufferingBar** 和 **SeekBar** 左侧的所有控件都将 `anchorLeft` 设置为 `true`，而其右侧的所有控件都将 `anchorRight` 设置为 `true`。所有控件都将 `anchorBottom` 设置为 `true`。

您可以尝试编辑 `layout_mc` 影片剪辑，以生成控件位于其顶部（而不是底部）的外观。您只需要将控件移到顶部（相对于 `video_mc`），并且对于所有控件将 `anchorTop` 设置为等于 `true`。

## 按钮状态

所有按钮状态都放置于舞台上，但这些影片剪辑实例在舞台上的放置位置并不重要。但重要的是，它们以特定方式嵌套在影片剪辑内，并且每个剪辑实例都具有正确的实例名称。

剪辑实例的结构及其实例名称显示在以下示例中：

```
playpause_mc
 play_mc
 up_mc, over_mc, down_mc, disabled_mc
 pause_mc
 up_mc, over_mc, down_mc, disabled_mc
stop_mc
 up_mc, over_mc, down_mc, disabled_mc
back_mc
 up_mc, over_mc, down_mc, disabled_mc
forward_mc
 up_mc, over_mc, down_mc, disabled_mc
volumeMute_mc
 on_mc
 up_mc, over_mc, down_mc, disabled_mc
 off_mc
 up_mc, over_mc, down_mc, disabled_mc
```

请注意，**Flash 8 FLA** 文件在舞台上具有附加的“**Forward**”（快进）和“**Back**”（后退）按钮。这两个按钮位于引导图层上，并在那里显示 **ForwardBackBorder**、**ForwardBorder** 和 **BackBorder** 影片剪辑的用法。有关更多信息，请参见第 496 页的“**背景剪辑和前景剪辑**”。

您可以根据需要编辑各种状态。请记住，所有状态都由其注册点放置于相同位置；因此，如果某些状态大于其它状态，则可能无法将艺术作品放置于 (0, 0)，因为它位于大多数 **Flash 8** 按钮外观中。您可能会发现，在某些情况下，将注册点保持在艺术作品的中心更容易。

如果您不想使用所有这些状态，则可以省略某些状态，但 `up_mc` 不应省略。`up_mc` 剪辑用于省略的状态。

如果您想要具有单独的“播放”和“暂停”按钮，而不是组合的“播放 - 暂停”按钮，只需将 `play_mc` 和 `pause_mc` 剪辑放置于舞台上，并且不用 `playpause_mc` 剪辑环绕它们。除了在第 491 页的“使用 `layout_mc`”中介绍的代码外，无需使用其它 `ActionScript` 代码设置按钮。

## 缓冲栏

缓冲栏具有两个影片剪辑：`bufferingBar_mc` 和 `bufferingBarFill_mc`。舞台上各剪辑与其它剪辑的相对位置很重要，因为这种相对位置将一直保留。缓冲栏使用两个单独的剪辑，因为组件缩放 `bufferingBar_mc`，但不缩放 `bufferingBarFill_mc`。

`bufferingBar_mc` 剪辑对自身应用 9 切片缩放，因此在缩放时边框将不会扭曲。`bufferingBarFill_mc` 剪辑非常宽，因此它始终足够宽，完全无需缩放。在运行时将自动对它进行遮罩，以便只显示 `bufferingBar_mc` 高于伸展的部分。默认情况下，遮罩的准确尺寸将根据 `bufferingBar_mc` 和 `bufferingBarFill_mc` 的 `x`（水平）位置之间的差异，而在 `bufferingBar_mc` 内的左边距和右边距上保持相等。您可以使用 `ActionScript` 代码自定义该位置。

如果您的缓冲栏不需要缩放或者不使用 9 切片缩放，则可以像 `FLV` 回放自定义用户界面 `BufferingBar` 组件一样设置它。有关更多信息，请参见第 486 页的“`BufferingBar` 组件”。

缓冲栏具有以下附加属性：

属性	说明
<code>fill_mc:MovieClip</code>	指定缓冲栏填充的实例名称。默认值为 <code>bufferingBarFill_mc</code> 。

## 搜索栏和音量栏

搜索栏也具有两个影片剪辑：`seekBar_mc` 和 `seekBarProgress_mc`。舞台上各剪辑与其它剪辑的相对位置很重要，因为这种相对位置将一直保留。尽管这两个剪辑都缩放，但 `seekBarProgress_mc` 无法嵌套在 `seekBar_mc` 内，因为 `seekBar_mc` 使用 9 切片缩放，而 9 切片缩放无法很好地使用嵌套的影片剪辑。

`seekBar_mc` 剪辑对自身应用 9 切片缩放，因此在缩放时边框将不会扭曲。`seekBarProgress_mc` 剪辑也会缩放，但它却扭曲。它并不使用 9 切片缩放，因为它是一个填充，而填充在被扭曲时显示很正常。

`seekBarProgress_mc` 剪辑在没有 `fill_mc` 的情况下工作，与 `progress_mc` 剪辑在 FLV 回放自定义用户界面组件中的工作方式十分相似。换言之，它不被遮罩并且水平缩放。在 100% 时，`bufferingBarProgress_mc` 的确切尺寸由 `bufferingBar_mc` 剪辑内的左边距和右边距定义。默认情况下，这些尺寸是相等的并且基于 `seekBar_mc` 和 `seekBarProgress_mc` 的 `x`（水平）位置之间的差异。您可以在搜索栏影片剪辑中使用 `ActionScript` 自定义尺寸，如下示例所示：

```
progressLeftMargin = 2;
progressRightMargin = 2;
progressY = 11;
fullnessLeftMargin = 2;
fullnessRightMargin = 2;
fullnessY = 11;
```

与 FLV 回放自定义用户界面 `SeekBar` 组件一样，可以为搜索栏创建完成程度影片剪辑。如果您的搜索栏不需要缩放，或者它缩放但不具有 9 切片缩放，则您可以使用用于 FLV 回放自定义用户界面组件的任何方法来设置 `progress_mc` 或 `fullness_mc`。有关更多信息，请参见第 488 页的“进度和完成程度影片剪辑”。

因为 Flash 8 外观的音量条不缩放，所以采用与 `VolumeBar` FLV 回放自定义用户界面组件相同的方法建立其结构。有关更多信息，请参见第 487 页的“`SeekBar` 和 `VolumeBar` 组件”。例外情况就是手柄的实现方式不同。有关这一点的更多信息，请参见以下部分。

## 手柄

`SeekBar` 和 `VolumeBar` 手柄都放置于舞台上该条的旁边。默认情况下，手柄的左边距、右边距和 `y` 轴值按其相对于条影片剪辑的位置设置。左边距按手柄的 `x`（水平）位置和条的 `x`（水平）位置之间的差值设置，而右边距等于左边距。您可以在 `SeekBar` 或 `VolumeBar` 影片剪辑中通过 `ActionScript` 自定义这些值。以下示例所采用的 `ActionScript` 代码就是用于 FLV 回放自定义用户界面组件的代码：

```
handleLeftMargin = 2;
handleRightMargin = 2;
handleY = 11;
```

除了这些属性之外，手柄就是简单的影片剪辑，其设置方式与在 FLV 回放自定义用户界面组件中的设置方式相同。两者都具有 `alpha` 属性设置为 0 的矩形背景。提供它们只是为了增加点击区域，它们并不是必需的。

# ActionScript

搜索栏和音量条支持以下附加属性:

属性	说明
handle_mc:MovieClip	用于手柄的影片剪辑。默认值为 seekBarHandle_mc 或 volumeBarHandle_mc。
progress_mc:MovieClip	用于进度的影片剪辑。默认值为 seekBarProgress_mc 或 volumeBarProgress_mc。
fullness_mc:MovieClip	用于完成程度的影片剪辑。默认值为 seekBarFullness 或 volumeBarFullness。

## 背景剪辑和前景剪辑

影片剪辑 **chrome\_mc** 和 **forwardBackBorder\_mc** 是作为背景剪辑实现的。

在舞台和占位符 “Forward”（快进）和 “Back”（后退）按钮上的 **ForwardBackBorder**、**ForwardBorder** 和 **BackBorder** 影片剪辑中，唯一不在引导图层上的影片剪辑是 **ForwardBackBorder**。它只位于实际使用 “Forward”（快进）和 “Back”（后退）按钮的外观中。

不需要附加的 **ActionScript** 来设置背景剪辑和前景剪辑。

## 修改外观行为

**bufferingBarHidesAndDisablesOthers** 属性和 **skinAutoHide** 属性允许您自定义 **FLVPlayback** 外观的行为。

通过将 **bufferingBarHidesAndDisablesOthers** 属性设置为 **true**，可使 **FLVPlayback** 组件隐藏 **SeekBar** 及其手柄，并且在组件进入缓冲状态时禁用 “播放” 和 “暂停” 按钮。在 **FLV** 文件通过慢速连接（**bufferTime** 属性设置得较高，例如设置为 **10**）从 **FCS** 流式加载时，上述功能特别有用。在此情况下，不耐烦的用户可能会尝试通过单击 “播放” 和 “暂停” 按钮启动搜索，这可能会导致文件播放时间延迟得更长。您可以通过将 **bufferingBarHidesAndDisablesOthers** 设置为 **true**，并在组件处于缓冲状态时禁用 **SeekBar** 组件以及 “暂停” 和 “播放” 按钮，以防止上述行为发生。

**skinAutoHide** 属性只影响预先设计的外观 **SWF** 文件，并不影响通过 **FLV** 回放自定义用户界面组件创建的控件。如果设置为 **true**，**FLVPlayback** 组件将在鼠标不在查看区域上时隐藏外观。此属性的默认值是 **true**。

# FLVPlayback 类

继承 MovieClip > FLVPlayback 类

ActionScript 类名称 mx.video.FLVPlayback

FLVPlayback 可扩展 MovieClip 类并可包装 VideoPlayer 对象。有关 VideoPlayer 类的信息，请参见第 649 页的“VideoPlayer 类”。

与其它组件不同，FLVPlayback 组件不扩展 UIObject 或 UIComponent，因此，不支持这些类的方法和属性。例如，您必须调用 MovieClip 方法 attachMovie() 而不是 UIObject 方法 createClassObject()，来创建 ActionScript 中组件的一个实例。

FLVPlayback 类的方法和属性使您能够使用 FLVPlayback 组件在 Flash 应用程序中播放和控制 FLV 文件。

通过 ActionScript 设置 FLVPlayback 类的属性会覆盖“属性”检查器或“组件”检查器中用于初始设置属性的等效参数。

每个组件类都有一个 version 属性，而该属性是一个类属性。类属性只能用于该类本身。version 属性会返回一个字符串，该字符串指示组件的版本。以下代码在“输出”面板中显示版本：

```
trace(mx.video.FLVPlayback.version);
```

## FLVPlayback 类的方法摘要

下表列出了 FLVPlayback 类的方法：

方法	说明
<code>FLVPlayback.addASCuePoint()</code>	添加一个 ActionScript 提示点。
<code>FLVPlayback.addEventListener()</code>	为指定的事件创建侦听器。
<code>FLVPlayback.bringVideoPlayerToFront()</code>	将一个视频播放器置于堆叠的视频播放器的前面。
<code>FLVPlayback.closeVideoPlayer()</code>	为具有指定索引的视频播放器关闭 NetStream 并删除该视频播放器。
<code>FLVPlayback.findCuePoint()</code>	查找具有指定时间、指定名称或指定时间和名称的提示点的指定类型。
<code>FLVPlayback.Findnearestcuepoint()</code>	查找位于或接近给定时间或具有给定名称的提示点的指定类型。
<code>FLVPlayback.findNextCuePointWithName()</code>	查找其名称与 findCuePoint() 或 findNearestCuePoint() 方法返回的提示点的名称相同的下一个提示点。
<code>FLVPlayback.getVideoPlayer()</code>	获取由 index 参数指定的视频播放器。

方法	说明
<code>FLVPlayback.isFLVCuePointEnabled()</code>	如果 <code>ActionScript</code> 禁用 <code>FLV</code> 文件嵌入式提示点, 则返回 <code>false</code> 。
<code>FLVPlayback.load()</code>	开始加载 <code>autoPlay</code> 属性设置为 <code>false</code> 的 <code>FLV</code> 文件。
<code>FLVPlayback.pause()</code>	暂停播放视频流。
<code>FLVPlayback.play()</code>	开始播放视频流, 同时允许加载和播放新的 <code>FLV</code> 文件。
<code>FLVPlayback.removeASCuePoint()</code>	删除一个 <code>ActionScript</code> 提示点。
<code>FLVPlayback.removeEventListener()</code>	删除事件侦听器。
<code>FLVPlayback.seek()</code>	在文件中搜索到给定时间, 用秒表示, 小数精确到毫秒。
<code>FLVPlayback.seekPercent()</code>	在文件中定位到某个百分比。
<code>FLVPlayback.seekSeconds()</code>	与 <code>FLVPlayback.seek()</code> 相同。
<code>FLVPlayback.seekToNavCuePoint()</code>	搜索到指定时间或指定时间之后具有给定名称的导航提示点。
<code>FLVPlayback.seekToNextNavCuePoint()</code>	基于指定时间搜索到下一个导航提示点。
<code>FLVPlayback.seekToPrevNavCuePoint()</code>	基于指定时间搜索到前一个导航提示点。
<code>FLVPlayback.setFLVCuePointEnabled()</code>	启用或禁用一个或多个 <code>FLV</code> 文件提示点。
<code>FLVPlayback.setScale()</code>	同时设置 <code>scaleX</code> 和 <code>scaleY</code> 。
<code>FLVPlayback.setSize()</code>	同时设置 <code>width</code> 和 <code>height</code> 。
<code>FLVPlayback.stop()</code>	停止播放视频流。

# FLVPlayback 类的属性摘要

FLVPlayback 类同时具有类属性和实例属性。

## FLVPlayback 类属性

以下属性仅对 FLVPlayback 类存在。它们是只读常数，适用于应用程序中 FLVPlayback 组件的所有实例。

属性	值	说明
FLVPlayback.ACTIONSCRIPT	"actionscript"	可以用作 findCuePoint() 和 findNearestCuePoint() 方法的类型参数。
FLVPlayback.ALL	"all"	可以用作 findCuePoint() 和 findNearestCuePoint() 方法的类型参数。
FLVPlayback.BUFFERING	"buffering"	一个用于测试 state 属性的值。
FLVPlayback.CONNECTION_ERROR	"connectionError"	一个用于测试 state 属性的值。
FLVPlayback.DISCONNECTED	"disconnected"	一个用于测试 state 属性的值。
FLVPlayback.EVENT	"event"	可以用作 findCuePoint() 和 findNearestCuePoint() 方法的类型参数。
FLVPlayback.FLV	"flv"	可以用作 findCuePoint() 和 findNearestCuePoint() 方法的类型参数。
FLVPlayback.LOADING	"loading"	一个用于测试 state 属性的值。
FLVPlayback.NAVIGATION	"navigation"	可以用作 findCuePoint() 和 findNearestCuePoint() 方法的类型参数。
FLVPlayback.PAUSED	"paused"	一个用于测试 state 属性的值。
FLVPlayback.PLAYING	"playing"	一个用于测试 state 属性的值。
FLVPlayback.REWINDING	"rewinding"	一个用于测试 state 属性的值。
FLVPlayback.SEEKING	"seeking"	一个用于测试 state 属性的值。
FLVPlayback.STOPPED	"stopped"	一个用于测试 state 属性的值。
FLVPlayback.version	表示组件版本号的数值	确定您正使用的 FLVPlayback 组件的版本。

## 实例属性

下表列出了 `FLVPlayback` 类的实例属性。这组属性适用于应用程序中 `FLVPlayback` 组件的每个实例。例如，如果您将 `FLVPlayback` 组件的两个实例拖到舞台上，则每个实例都具有上述一组属性。

属性	说明
<code>FLVPlayback.activeVideoPlayerIndex</code>	一个数字，指定哪个 FLV 文件流受其它方法、属性和事件的影响。使用此属性可以管理多个 FLV 文件流；默认值为 0。
<code>FLVPlayback.autoPlay</code>	一个布尔值，如果为 <code>true</code> ，则指定组件在加载 FLV 文件后立即播放它。默认值为 <code>true</code> 。
<code>FLVPlayback.autoRewind</code>	一个布尔值，如果为 <code>true</code> ，则在停止播放时，使 FLV 文件后退到第 1 帧。
<code>FLVPlayback.autoSize</code>	一个布尔值，如果为 <code>true</code> ，则视频大小将自动调整为源尺寸。
<code>FLVPlayback.backButton</code>	作为 <code>BackButton</code> 控件的 <code>MovieClip</code> 对象。
<code>FLVPlayback.bitrate</code>	一个数字，它按每秒位数指定用户的带宽。在许多情况下用于确定要播放哪个 FLV 文件。
<code>FLVPlayback.buffering</code>	一个布尔值，如果视频处于缓冲状态，则为 <code>true</code> 。只读。
<code>FLVPlayback.bufferingBar</code>	作为 <code>bufferingBar</code> 控件的 <code>MovieClip</code> 对象。
<code>FLVPlayback.bufferingBarHidesAndDisablesOthers</code>	在组件进入缓冲状态时影响控件的行为。
<code>FLVPlayback.BUFFERTIME</code>	一个数字，指定开始回放视频流前要在内存中缓冲的秒数。
<code>FLVPlayback.bytesLoaded</code>	一个数字，指示 HTTP 下载的下载程度，以字节数表示。只读。
<code>FLVPlayback.bytesTotal</code>	一个数字，指定 HTTP 下载的总下载字节数。只读。
<code>FLVPlayback.contentPath</code>	一个字符串，指定要加载的 FLV 文件的 URL。
<code>FLVPlayback.cuePoints</code>	一个数组，说明 <code>ActionScript</code> 提示点和已禁用的嵌入式 FLV 文件提示点。该数组应永远不直接用于 <code>ActionScript</code> ，而只应通过“提示点”对话框设置。



属性	说明
<code>FLVPlayback.forwardButton</code>	作为 ForwardButton 控件的 MovieClip 对象。
<code>FLVPlayback.height</code>	一个数字，指定视频的高度，以像素为单位。
<code>FLVPlayback.idleTimeout</code>	Flash 结束由于播放暂停或停止而导致 FCS 连接空闲之前的时间，以毫秒为单位。
<code>FLVPlayback.isLive</code>	一个布尔值，如果是实时视频流，则为 true。
<code>FLVPlayback.isRTMP</code>	一个布尔值，如果 FLV 是从 FCS 或 FVSS 流式加载的，则为 true。只读。
<code>FLVPlayback.maintainAspectRatio</code>	一个布尔值，如果为 true，则保持视频高宽比。
<code>FLVPlayback.metadata</code>	一个对象，它是通过调用 onMetaData() 回调函数（如果有）而接收到的元数据信息包。只读。
<code>FLVPlayback.metadataLoaded</code>	一个布尔值，如果已经遇到并处理了元数据包或者如果肯定不会处理元数据包，则为 true。只读。
<code>FLVPlayback.muteButton</code>	作为 MuteButton 控件的 MovieClip 对象。
<code>FLVPlayback.ncMgr</code>	一个 INCManger 对象，它提供对实现 INCManger 的类的实例的访问。只读。
<code>FLVPlayback.pauseButton</code>	作为 PauseButton 控件的 MovieClip 对象。
<code>FLVPlayback.paused</code>	一个布尔值，如果 FLV 文件处于暂停状态，则为 true。只读。
<code>FLVPlayback.playButton</code>	作为 PlayButton 控件的 MovieClip 对象。
<code>FLVPlayback.playheadPercentage</code>	一个数字，以占 FLV 文件总持续时间的百分比的形式指定播放头时间。
<code>FLVPlayback.playheadTime</code>	一个数字，表示当前播放头的时间或位置（以秒为单位计算），可以是小数。
<code>FLVPlayback.playheadUpdateInterval</code>	一个数字，表示每个 playheadUpdate 事件之间的时间量，以毫秒为单位。
<code>FLVPlayback.playing</code>	一个布尔值，如果 FLV 文件正在播放，则为 true。只读。

属性	说明
<code>FLVPlayback.playPauseButton</code>	作为 <code>PlayPauseButton</code> 控件的 <code>MovieClip</code> 对象。
<code>FLVPlayback.preferredHeight</code>	一个数字，指定源 FLV 文件的高度。
<code>FLVPlayback.preferredWidth</code>	一个数字，指定源 FLV 文件的宽度。
<code>FLVPlayback.progressInterval</code>	一个数字，表示每个 <code>progress</code> 事件之间的时间量，以毫秒为单位。
<code>FLVPlayback.scaleX</code>	一个数字，指定水平缩放。
<code>FLVPlayback.scaleY</code>	一个数字，指定垂直缩放。
<code>FLVPlayback.scrubbing</code>	一个布尔值，如果用户正在拖动 <code>seekBar</code> 手柄，则为 <code>true</code> 。只读。
<code>FLVPlayback.seekBar</code>	作为 <code>SeekBar</code> 控件的 <code>MovieClip</code> 对象。
<code>FLVPlayback.seekBarInterval</code>	一个数字，指定进行拖拽时检查 <code>seekBar</code> 手柄的频率，以毫秒为单位。默认值为 250。
<code>FLVPlayback.seekBarScrubTolerance</code>	一个数字（百分比），指定更新之前用户可以移动 <code>scrubBar</code> 手柄的距离。该值以百分比指定，范围从 1 至 100。
<code>FLVPlayback.seekToPrevOffset</code>	一个数字（秒数）， <code>seekToPrevNavCuePoint()</code> 方法将它的时间与上一个提示点进行比较时所使用的秒数。
<code>FLVPlayback.skin</code>	一个字符串，指定外观 SWF 文件的名称。
<code>FLVPlayback.skinAutoHide</code>	一个布尔值，如果为 <code>true</code> ，则鼠标未滑过视频上时隐藏组件外观。默认值为 <code>false</code> 。
<code>FLVPlayback.state</code>	一个字符串，指定组件的状态。用 <code>load()</code> 、 <code>play()</code> 、 <code>stop()</code> 、 <code>pause()</code> 和 <code>seek()</code> 方法来设置。只读。
<code>FLVPlayback.stateResponsive</code>	一个布尔值，如果处于响应状态，则为 <code>true</code> 。只读。
<code>FLVPlayback.stopButton</code>	作为 <code>StopButton</code> 控件的 <code>MovieClip</code> 对象。
<code>FLVPlayback.stopped</code>	一个布尔值，如果处于停止状态，则为 <code>true</code> 。只读。
<code>FLVPlayback.totalTime</code>	一个数字，表示视频的总播放时间，以秒为单位。

属性	说明
<code>FLVPlayback.transform</code>	一个对象，提供对 <code>Sound.setTransform()</code> 和 <code>Sound.getTransform()</code> 方法的直接访问，以提供更多声音控制。
<code>FLVPlayback.visible</code>	一个布尔值。如果为 <code>true</code> ，则 <code>FLVPlayback</code> 组件可见。
<code>FLVPlayback.visibleVideoPlayerIndex</code>	一个数字，可用于管理多个 FLV 文件流。设置哪个视频播放器实例是可见和可听到的。默认值为 <code>0</code> 。
<code>FLVPlayback.volume</code>	一个数字，介于 <code>0</code> 到 <code>100</code> 的范围内，指示音量控制设置。
<code>FLVPlayback.volumeBar</code>	作为 <code>VolumeBar</code> 控件的 <code>MovieClip</code> 对象。
<code>FLVPlayback.volumeBarInterval</code>	一个数字，指定进行拖拽时检查 <code>volumeBar</code> 手柄的频率，以秒为单位。默认值为 <code>250</code> 。
<code>FLVPlayback.volumeBarScrubTolerance</code>	一个数字（百分比），指定更新之前用户可以移动音量栏手柄的距离。
<code>FLVPlayback.width</code>	一个数字，指定组件实例的宽度，以像素为单位。
<code>FLVPlayback.x</code>	一个数字，指定视频播放器的水平位置，以像素为单位。
<code>FLVPlayback.y</code>	一个数字，指定视频播放器的垂直位置，以像素为单位。

## FLVPlayback 类的事件摘要

下表列出了 `FLVPlayback` 类的事件：

事件	说明
<code>FLVPlayback.buffering</code>	在进入缓冲状态时调度。
<code>FLVPlayback.close</code>	通过超时或通过调用 <code>close()</code> 方法关闭 <code>NetConnection</code> 时调度。
<code>FLVPlayback.complete</code>	播放完成（到达 FLV 文件的末端）时调度。
<code>FLVPlayback.cuePoint</code>	到达提示点时调度。
<code>FLVPlayback.fastForward</code>	通过调用 <code>seek()</code> 方法向前移动播放头的位置时调度。
<code>FLVPlayback.metadata</code>	第一次到达 FLV 文件元数据时调度。

事件	说明
<code>FLVPlayback.paused</code>	在进入暂停状态时调度。
<code>FLVPlayback.playheadUpdate</code>	默认情况下，在播放 FLV 文件时每隔 0.25 秒调度一次。您可以使用 <code>playheadUpdateInterval</code> 属性指定频率。
<code>FLVPlayback.playing</code>	在进入播放状态时调度。
<code>FLVPlayback.progress</code>	每隔 0.25 秒调度一次，从调用 <code>load()</code> 方法时开始，到所有字节加载结束或者出现网络错误时结束。您可以使用 <code>progressInterval</code> 属性指定频率。
<code>FLVPlayback.ready</code>	加载 FLV 文件并可以显示它时调度。
<code>FLVPlayback.resize</code>	调整视频大小时调度。
<code>FLVPlayback.rewind</code>	通过调用 <code>seek()</code> 向后移动播放头位置时或者完成自动后退操作时调度。
<code>FLVPlayback.scrubFinish</code>	在用户使用 <code>SeekBar</code> 停止拖拽时间轴时调度。
<code>FLVPlayback.scrubStart</code>	在用户使用 <code>SeekBar</code> 开始拖拽时间轴时调度。
<code>FLVPlayback.seek</code>	通过调用 <code>seek()</code> 或者通过使用相应控件更改了播放头位置时调度。
<code>FLVPlayback.skinError</code>	在加载外观 SWF 文件发生错误时调度。
<code>FLVPlayback.skinLoaded</code>	在加载外观 SWF 文件时调度。
<code>FLVPlayback.stateChange</code>	回放状态发生更改时调度。
<code>FLVPlayback.stopped</code>	在进入停止状态时调度。
<code>FLVPlayback.Volumeupdate</code>	通过 <code>volume</code> 属性更改音量时调度。

## FLVPlayback.ACTIONSCRIPT

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
mx.video.FLVPlayback.ACTIONSCRIPT
```

### 说明

一个只读的 **FLVPlayback** 类属性，包含字符串常数 "actionscript"，并用作具有 `findCuePoint()` 和 `findNearestCuePoint()` 方法的 `type` 属性。

## 示例

以下示例使用 ACTIONSCRIPT 常数设置 findCuePoint() 方法的 type 属性。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求:
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
 var cuePt:Object = new Object(); // 创建提示点对象
 cuePt.time = 2.444;
 cuePt.name = "ASCuePt1";
 my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
}
my_FLVPlybk.addEventListener("ready", listenerObject)
listenerObject.playing = function(eventObject:Object) {
 var rtn_obj:Object = new Object();
 if(rtn_obj = my_FLVPlybk.findCuePoint(2.444,
 FLVPlayback.ACTIONSCRIPT)){
 trace("Found cue point " + rtn_obj.name);
 }
}
my_FLVPlybk.addEventListener("playing", listenerObject)
```

## 另请参见

[FLVPlayback.findCuePoint\(\)](#)

# FLVPlayback.activeVideoPlayerIndex

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk*.activeVideoPlayerIndex

说明

属性：一个数字，指定其它 API 影响哪个视频播放器实例。此属性用于管理多个 FLV 文件流。默认值为 0。

该属性并不使视频播放器可见；使用 visibleVideoPlayerIndex 属性可使视频播放器可见。首次将 activeVideoPlayerIndex 设置为数值时，会创建一个新视频播放器。创建新视频播放器时，它的属性设置为默认视频播放器的值 (activeVideoPlayerIndex == 0)，contentPath、totalTime 和 isLive 除外，它们始终设置为默认值（分别为空字符串、0 和 false），而 autoPlay 始终设置为 false（仅对于默认视频播放器，即 0，该默认值为 true）。cuePoints 属性没有任何影响，因为它对于随后加载到默认视频播放器中没有任何影响。

控制音量、位置、尺寸、可见性和用户界面控件的 API 始终是全局的，它们的行为不受 activeVideoPlayerIndex 设置的影响。具体而言，设置 activeVideoPlayerIndex 属性并不影响以下属性和方法。

不受 activeVideoPlayerIndex 影响的属性和方法

backButton	playPauseButton	skin	width
bufferingBar	scaleX	stopButton	x
bufferingBarHidesAndDisablesOthers		transform	y
forwardButton	scaleY	visible	setSize()
height	seekBar	volume	setScale()
muteButton	seekBarInterval	volumeBar	
pauseButton	seekBarScrubTolerance	volumeBarInterval	
playButton	seekToPrevOffset	volumeBarScrubTolerance	

提醒

visibleVideoPlayerIndex 属性（不是 activeVideoPlayerIndex 属性）确定外观所控制的视频播放器。

但控制尺寸的 API 却会与 visibleVideoPlayerIndex 属性交互。有关更多信息，请参见第 632 页的“FLVPlayback.visibleVideoPlayerIndex”。

其余 API 根据 activeVideoPlayerIndex 的设置将特定视频播放器作为目标。

侦听事件时，您可以获取所有视频播放器的所有事件。若要区分哪个事件用于哪个视频播放器，请使用事件的 vp 属性，它是一个数字，对应于 activeVideoPlayerIndex 和 visibleVideoPlayerIndex 中设置的数值。除了 resize 和 volume 以外（它们不特定于视频播放器，而是用于 FLVPlayback 实例的全局事件），所有事件都具有此属性。

例如，若要在背景中加载另一个 FLV 文件，可将 `activeVideoPlayerIndex` 设置为 1，并调用 `load()` 方法。若要准备显示此 FLV 文件而隐藏第一个 FLV 文件时，可将 `visibleVideoPlayerIndex` 设置为 1。

## 示例

以下示例创建两个视频播放器，以在单个 FLV 文件实例中连续播放两个 FLV 文件。它设置 `activeVideoPlayerIndex` 属性，以在视频播放器及其相应的 FLV 文件之间切换。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
// 为默认播放器指定 FLV 的名称和位置
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 // 添加另一个视频播放器，并指定其 FLV 的名称和位置
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
 // 重置为默认的视频播放器，它将自动播放其 FLV
 my_FLVPlybk.activeVideoPlayerIndex = 0;
};
my_FLVPlybk.addEventListener("ready", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
 // 如果 complete 事件是针对第二个 FLV，则使默认播放器活动并可见
 if (eventObject.vp == 1) {
 my_FLVPlybk.activeVideoPlayerIndex = 0;
 my_FLVPlybk.visibleVideoPlayerIndex = 0;
 }
 else { // 使第二个播放器活动并可见，并且播放 FLV
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.visibleVideoPlayerIndex = 1;
 my_FLVPlybk.play();
 }
};
// 添加 complete 事件的侦听器
my_FLVPlybk.addEventListener("complete", listenerObject);
```

## 另请参见

[FLVPlayback.bringVideoPlayerToFront\(\)](#)、[FLVPlayback.getVideoPlayer\(\)](#)、[VideoPlayer](#) 类、[FLVPlayback.visibleVideoPlayerIndex](#)

# FLVPlayback.addASCuePoint()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVplybk.addASCuePoint(cuePoint:Object)
my_FLVplybk.addASCuePoint(time:Number, name:String[, parameters:Object])
```

## 参数

*cuePoint* 一个对象，具有 *name:String* 和 *time:Number* 属性（以秒为单位），对提示点进行说明。它可能还具有 *parameters:Object* 属性，用以容纳名称 / 值对。可以将 *type:String* 设置为 "actionscript"。如果 *type* 丢失或者设置为其它值，则自动设置此值。如果对象不符合这些惯例，则该方法将引发 **VideoError** 错误。

*time* 一个数字，表示要添加的新提示点的时间。如果使用 *time* 参数，则其后必须使用 *name* 参数。

*name* 一个字符串，如果您提交 *time* 参数而不是 *CuePoint* 对象，则指定提示点的名称。

*parameters* 用于提示点的可选参数。

## 返回

已使用以下附加属性添加的提示点对象的副本：

- *array* 由搜索过的提示点组成的数组。只能将此数组视为只读，因为在该数组中添加、删除或编辑对象时会导致提示点故障。
- *index* 返回的提示点在数组中的索引。

## 说明

方法：添加 **ActionScript** 提示点，并与使用“提示点”对话框添加 **ActionScript** 提示点具有几乎相同的效果，只是前者在应用程序执行时出现，而不是在应用程序开发期间出现。

设置 *contentPath* 属性会清除提示点信息。若要为下一个要加载的 FLV 文件设置提示点信息，请首先设置 *contentPath* 属性。

可以添加多个具有相同名称和时间的 **ActionScript** 提示点。在使用 *removeASCuePoint()* 方法删除 **ActionScript** 提示点时，具有相同名称和时间的所有提示点都将被删除。



## 示例

以下示例向某个 FLV 文件中添加两个 **ActionScript** 提示点。该示例使用 *CuePoint* 参数添加第一个提示点，并使用 *time* 和 *name* 参数添加第二个提示点。播放过程中出现每个提示点时，**cuePoint** 事件的侦听器会在文本区域中显示 *playheadTime* 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。将一个 **TextArea** 组件拖到舞台中的 **FLVPlayback** 实例下，然后为其指定实例名称 **my\_ta**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 - 舞台上的 TextArea 组件具有实例名称 my_ta
*/
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_ta.visible = false;
// 创建提示点对象
var cuePt:Object = new Object(); // 创建提示点对象
cuePt.time = 2.444;
cuePt.name = "ASCuePt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
// 使用 time 和 name 参数添加第二个 AS 提示点
my_FLVPlybk.addASCuePoint(5, "ASCuePt2");
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 my_ta.text = "Elapsed time in seconds: " + my_FLVPlybk.playheadTime;
 my_ta.visible = true;
};
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

## 另请参见

[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.removeASCuePoint\(\)](#)

# FLVPlayback.addEventListener()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlybk.addEventListener(event:String, listener:Object):Void
my_FLVPlybk.addEventListener(event:String, listener:Function):Void
```

## 参数

*event* 一个字符串，指定您为其注册侦听器的事件的名称。如果该侦听器是一个对象，则该名称也是要调用的侦听器对象函数的名称。

*listener* 您要为事件注册的侦听器对象或函数的名称。

## 返回

无。

## 说明

方法：为指定的事件注册侦听器对象或函数。如果该侦听器是一个对象，则该对象必须具有为它定义的函数，并且函数的名称与事件名称相同。如果该侦听器是一个函数，则它是将被调用以处理该事件的函数的名称。

## 示例

以下示例侦听 `complete` 事件，并在它发生时在文本区域中显示一条消息。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。将一个 `TextArea` 组件拖到舞台中的 `FLVPlayback` 实例下，然后为其指定实例名称 **my\_ta**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
用法 1: listener object
/**
 要求:
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 - 舞台上的 TextArea 组件具有实例名称 my_ta
*/
import mx.video.*;
my_ta.visible = false;
my_FLVPlybk.contentPath = "http://www.helpeexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object(); // 创建侦听器对象
listenerObject.complete = function(eventObject:Object):Void {
 my_ta.text = "That's All Folks!";
}
```

```

 my_ta.visible = true;
 };
 my_FLVPlayback.addEventListener("complete", listenerObject);

用法 2: listener function
/**
 要求:
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 - 舞台上的 TextArea 组件具有实例名称 my_ta
*/
import mx.video.*;
my_ta.visible = false;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
function the_end(eventObject:Object):Void {
 my_ta.text = "That's All Folks!";
 my_ta.visible = true;
};
my_FLVPlayback.addEventListener("complete", the_end);

```

### 另请参见

[FLVPlayback 类的事件摘要](#)、[FLVPlayback.removeEventListener\(\)](#)

## FLVPlayback.ALL

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.FLVPlayback.ALL`

### 说明

一个只读的 `FLVPlayback` 类属性，可以包含字符串常数 `"all"`。可以将此属性用作 `findCuePoint()` 和 `findNearestCuePoint()` 方法的 `type` 参数。

### 示例

以下示例在所有提示点中查找名为 `point2` 并且时间为 `7.748` 的提示点。该示例显示已找到的提示点的 `type` 和 `time` 属性。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **`my_FLVPlayback`**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
```

要求:

```
- 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
*/
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
// 创建提示点对象
var listenerObject = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 var cuePt:Object = new Object(); // 创建提示点对象
 cuePt.name = "point2";
 cuePt.time = 7.748;
 if(cuePt = my_FLVPlayback.findCuePoint(cuePt, FLVPlayback.ALL)) // 查找提示点
 trace("found a " + cuePt.type + " cue point at " + cuePt.time);
 else
 trace("cue point not found");
}
my_FLVPlayback.addEventListener("ready", listenerObject);
```

另请参见

[FLVPlayback.findCuePoint\(\)](#)

## FLVPlayback.autoPlay

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

*my\_FLVPlayback*.autoPlay

说明

属性：一个布尔值，如果设置为 `true`，则在设置了 `contentPath` 属性后会立即播放 **FLV** 文件。如果设置为 `false`，则组件会等待播放命令。即使 `autoPlay` 设置为 `false`，组件也会立即加载内容。默认值为 `true`。

如果在加载新 **FLV** 文件期间将该属性设置为 `true`，则除非设置了 `contentPath`，否则该属性不起作用。

## 示例

以下示例禁止 FLV 文件的播放，而将播放头设置为播放时间开始后的 30% 并在该点开始播放。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoPlay = false;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVPlybk.seekPercent(30);
 my_FLVPlybk.play();
};
my_FLVPlybk.addEventListener("ready", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)

# FLVPlayback.autoRewind

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlybk.autoRewind`

## 说明

属性：一个布尔值，如果为 `true`，则播放停止时（由于播放器到达流的末端或调用了 `stop()` 方法），会使 FLV 文件后退到第 1 帧。此属性对于实时流无意义。默认值为 `true`。

## 示例

以下示例将 `autoRewind` 属性设置为 `false`，以防止 **FLV** 文件完成播放时自动后退。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 库中有 FLVPlayback 组件
 */
my_FLVPlybk.autoRewind = false;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## FLVPlayback.autoSize

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`my_FLVPlybk.autoSize`

### 说明

属性：一个布尔值，如果为 `true`，则会使视频自动调整为源 **FLV** 文件尺寸大小。如果加载 **FLV** 文件后将此属性从 `false` 设置为 `true`，则会立即开始自动调整大小。默认值为 `false`。

## 示例

以下示例首先在 **FLV** 文件准备播放时显示该 **FLV** 文件的源尺寸（`preferredWidth` 和 `preferredHeight`）。然后，它调用 `setSize()` 以更改 **FLVPlayback** 实例的尺寸，并且触发 `resize` 事件。接下来，它将 `autoSize` 属性设置为 `true`，并且触发另一个 `resize` 事件，该事件将大小恢复为源 **FLV** 文件的尺寸。`resize` 事件处理函数在发生每个事件后都在“输出”面板中显示 **FLVPlayback** 实例的尺寸。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 库中有 FLVPlayback 组件
 */
import mx.video.*;
my_FLVPlybk.maintainAspectRatio = false;
var listenerObject:Object = new Object();
```

```
listenerObject.ready = function(eventObject:Object) {
 trace("FLV width is: " + my_FLVPlayback.preferredWidth + " FLV height is: " +
 my_FLVPlayback.preferredHeight);
 my_FLVPlayback.setSize(400, 400);
 my_FLVPlayback.autoSize = true;
};
my_FLVPlayback.addEventListener("ready", listenerObject);
listenerObject.resize = function(eventObject:Object) {
 trace("my_FLVPlayback width is: " + my_FLVPlayback.width + "; my_FLVPlayback.height
 is: " + my_FLVPlayback.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.maintainAspectRatio](#)、[FLVPlayback.preferredHeight](#)、  
[FLVPlayback.preferredWidth](#)、[FLVPlayback.resize](#)

## FLVPlayback.backButton

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlayback.backButton*

### 说明

属性：作为 **BackButton** 回放控件的 **MovieClip** 对象。有关将 **FLV** 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 **FLV** 回放自定义用户界面组件的外观”。

单击 **BackButton** 控件将导致 `rewind` 事件。

## 示例

以下示例使用 backButton、forwardButton、playButton、pauseButton 和 stopButton 属性，来将各个 FLV 回放自定义用户界面控件附加到 FLVPlayback 组件。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**，然后在“组件”检查器中将 skin 参数设置为“无”。接下来，添加以下各个 FLV 回放自定义用户界面组件，并为其指定括号中显示的实例名称：BackButton (**my\_bkbtn**)、ForwardButton (**my\_fwdbtn**)、PlayButton (**my\_plybtn**)、PauseButton (**my\_pausbtn**) 和 StopButton (**my\_stopbtn**)。然后将以下几行代码添加到“动作”面板中：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 - 库中有 FLV 自定义用户界面的 BackButton、ForwardButton、PlayButton、PauseButton
 和
 StopButton 组件
*/
import mx.video.*;
my_FLVPlybk.backButton = my_bkbtn;
my_FLVPlybk.forwardButton = my_fwdbtn;
my_FLVPlybk.playButton = my_plybtn;
my_FLVPlybk.pauseButton = my_pausbtn;
my_FLVPlybk.stopButton = my_stopbtn;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.forwardButton](#)、[FLVPlayback.rewind](#)

# FLVPlayback.bitrate

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk*.bitrate



## 说明

属性：一个数字，指定用于传输 FLV 文件的每秒比特数。

进行渐进式下载时，您可以使用 SMIL 格式，但必须设置比特率，因为不会进行自动检测。

在从 FCS 流式加载视频时，可以提供一个 SMIL 文件，该文件说明如何根据带宽在多个流之间切换。FCS 自动检测带宽，在本示例中，如果设置了带宽，则忽略 bitrate。

有关使用 SMIL 文件的更多信息，请参见第 653 页的“使用 SMIL 文件”。

## 示例

以下示例检查两个单选按钮，以确定从指定的 SMIL 文件选择某个 FLV 文件时要使用的比特率。SMIL 文件中的相关 video 标签显示在以下代码中：

```
<switch>
 <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1">
 <video src="myvideo_isdn.flv" dur="3:00.1">
</switch>
```

对于低速连接，这段代码设置 bitrate 属性以强制选择相应的 FLV 文件。对于更高速度的连接，它利用自动带宽检测功能从 FCS 流式加载，并且不设置比特率。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。拖动 RadioButton 组件，再将一个 Label 组件拖到“库”面板中。然后将以下代码添加到时间轴第 1 帧的“动作”面板上。在用于加载 contentPath 属性的语句中，用您的 SMIL 文件的名称和位置替代斜体文本。

```
/**
 要求：
 - 库中有 FLVPlayback 组件
 - 库中有 RadioButton 组件
 - 库中有 Label 组件
*/
import mx.video.*;
import mx.controls.*;

this.createClassObject(Label, "my_prompt", 10);
my_prompt.text = "Please indicate your connection speed: ";
this.createClassObject(RadioButton, "dialup", 20, {label:"Dialup modem (56 KB)", groupName:"radioGroup"});
this.createClassObject(RadioButton, "isdn", 30, {label:"ISDN (128 KB)", groupName:"radioGroup"});
my_prompt.autoSize = "left";
dialup.setSize(200, 30);
isdn.setSize(200, 30);
// 在舞台上定位 RadioButton。
my_prompt.move(my_FLVPlayback.x, my_FLVPlayback.y + my_FLVPlayback.height + 40);
dialup.move(my_FLVPlayback.x, my_prompt.y + my_prompt.height + 5);
isdn.move(my_FLVPlayback.x, dialup.y + 15);
```

```
// 创建侦听器对象
var rbListener:Object = new Object();
rbListener.click = function(eventObject:Object){
 if(dialup.selected) { // 对于调制解调器
 my_FLVPlayback.bitrate = 56000;
 } // 对于 ISDN (或更高的带宽), 允许自动检测
}
// 添加侦听器
radioGroup.addEventListener("click", rbListener);
my_FLVPlayback.contentPath = "http://www.someserver.com/video/sample.smil";
```

## FLVPlayback.bringVideoPlayerToFront()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.bringVideoPlayerToFront(index:Number)
```

### 参数

*index* 一个数字, 它是视频播放器前移的索引

### 说明

方法: 将一个视频播放器置于堆叠的视频播放器的前面。用于视频播放器之间的自定义传输。堆叠顺序与用于 `activeVideoPlayerIndex` 属性的堆叠顺序相同: 0 是位于底部, 1 是在 0 之上, 2 是在 1 之上, 依此类推。

### 示例

以下示例使用两个视频播放器来播放两个 FLV 文件。在第一个 FLV 文件 (`cuepoints.flv`) 中的三个提示点分别出现时, 该示例调用 `bringVideoPlayerToFront()` 方法, 以便将其它 FLV 文件置于前面。因为对于视频播放器编号 1, 该示例将 `_alpha` 属性设置为 75, 所以, 在该播放器中播放的 FLV 文件 (`plane_cuepoints`) 是透明的, 并且在该 FLV 文件位于前面后同时使这两个 FLV 文件可见。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
*/
my_FLVPlayback.load("http://www.helpexamples.com/flash/video/cuepoints.flv");
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
 if (eventObject.target.contentPath == "http://www.helpexamples.com/flash/video/cuepoints.flv") {
 // 以下代码将在第一个 FLV 就绪后引发
 my_FLVPlayback.activeVideoPlayerIndex = 1;
 my_FLVPlayback.load("http://www.helpexamples.com/flash/video/plane_cuepoints.flv");
 } else {
 // 以下代码将在第二个 FLV 就绪后引发
 eventObject.target.activeVideoPlayerIndex = 0;
 eventObject.target.play();
 eventObject.target.activeVideoPlayerIndex = 1;
 eventObject.target.play();
 var layerOnTop:MovieClip = eventObject.target.getVideoPlayer(1);
 layerOnTop._alpha = 75;
 layerOnTop._visible = true;
 }
}
my_FLVPlayback.addEventListener("ready", listenerObject);

var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {

 // 根据线索名称，使其中一个位于另一个的前面
 if (eventObject.info.name == "point1") {
 trace(eventObject.info.name + " : 0 to front");
 eventObject.target.bringVideoPlayerToFront(1);
 } else if (eventObject.info.name == "point2") {
 trace(eventObject.info.name + " : 1 to front");
 eventObject.target.bringVideoPlayerToFront(0);
 } else if (eventObject.info.name == "point3") {
 trace(eventObject.info.name + " : 0 to front");
 eventObject.target.bringVideoPlayerToFront(1);
 }
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
```

### 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.getVideoPlayer\(\)](#)、[VideoPlayer](#) 类、[FLVPlayback.visibleVideoPlayerIndex](#)

# FLVPlayback.buffering

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.buffering = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("buffering", listenerObject);
```

## 说明

事件：FLVPlayback 实例进入缓冲状态时调度。FLVPlayback 实例在调用 play() 方法或者单击 Play 控件之后，而在进入正在播放状态之前，通常会进入这种状态。事件对象具有 state、playheadTime 和 vp 属性，是此事件适用的视频播放器的索引号。请参见[第 505 页的 FLVPlayback.activeVideoPlayerIndex](#) 和[第 632 页的 FLVPlayback.visibleVideoPlayerIndex](#)。

FLVPlayback 实例还会在缓冲开始时调度 stateChange 事件。

## 示例

以下示例创建 buffering 事件的侦听器。在 buffering 事件发生时，事件处理函数调用 trace() 方法以显示 state 和 vp 属性的值。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.buffering = function(eventObject:Object) {
 trace("The state property has a value of " + eventObject.target.state);
 trace("The video player number is: " + eventObject.vp);
};
my_FLVplybk.addEventListener("buffering", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

另请参见

[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.state](#)、[FLVPlayback.removeEventListener\(\)](#)

## FLVPlayback.BUFFERING

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

`mx.video.FLVPlayback.BUFFERING`

说明

属性：一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "buffering"。您可以将此属性与 `state` 属性进行比较，以确定该组件是否处于缓冲状态。

示例

以下示例在该组件进入缓冲状态时显示 `FLVPlayback.BUFFERING` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(eventObject.state == FLVPlayback.BUFFERING)
 trace(FLVPlayback.BUFFERING);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.buffering

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.buffering*

## 说明

属性；一个布尔值，如果视频处于缓冲状态，则为 true。只读。

## 示例

以下示例为 buffering 事件创建一个侦听器，并在该事件发生时在“输出”面板中显示一条消息。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(my_FLVPlayback.buffering){
 trace("The video is buffering");
 }
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.buffering](#)、[FLVPlayback.BUFFERING](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.bufferingBar

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlayback.bufferingBar`

## 说明

属性；作为缓冲栏控件的 **MovieClip** 对象。此控件显示 FLV 文件何时处于加载状态或缓冲状态。有关将 **FLV** 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 **FLV** 回放自定义用户界面组件的外观”。

## 示例

以下示例通过设置以下属性，将各个 **FLV** 回放自定义用户界面控件附加到 **FLVPlayback** 组件：playPauseButton、stopButton、backButton、forwardButton 和 bufferingBar。只有在开始播放之前缓冲 **FLV** 文件时才显示缓冲栏。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**，然后在“组件”检查器中将 **skin** 参数设置为“无”。接下来，添加以下各个 **FLV** 回放自定义用户界面组件，并为其指定括号中显示的实例名称：**PlayPauseButton** (**my\_bkbtn**)、**StopButton** (**my\_stopbtn**)、**BackButton** (**my\_bkbtn**)、**ForwardButton** (**my\_fwdbtn**) 和 **BufferingBar** (**my\_buffrbar**)。然后将以下几行代码添加到时间轴第 1 帧的“动作”面板中：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 - 库中有 FLV 回放自定义用户界面 PlayPauseButton、StopButton、BackButton、
 ForwardButton 和 BufferingBar 组件
*/
import mx.video.*;
my_FLVPlayback.playPauseButton = my_plypausbtn;
my_FLVPlayback.stopButton = my_stopbtn;
my_FLVPlayback.backButton = my_bkbtn;
my_FLVPlayback.forwardButton = my_fwdbtn;
my_FLVPlayback.bufferingBar = my_buffrbar;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.bufferingBarHidesAndDisablesOthers](#)

# FLVPlayback.bufferingBarHidesAndDisablesOthers

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.bufferingBarHidesAndDisablesOthers*

## 说明

属性；如果设置为 true，则在 FLV 文件处于缓冲状态时隐藏 **SeekBar** 控件，并禁用 **Play**、**Pause**、**PlayPause**、**BackButton** 和 **ForwardButton** 控件。此属性可用于在 FLV 文件正通过慢速连接下载或流式加载时，防止用户使用这些控件尝试加快 FLV 文件的播放速度。

## 示例

以下示例假定正在从 FCS 或 FVSS 播放 FLV 流文件。它设置

**bufferingBarHidesAndDisablesOthers** 属性，以在 FLV 文件正在缓冲时禁用 **Play**、**Pause**、**PlayPause**、**BackButton** 和 **ForwardButton** 控件并隐藏 **SeekBar** 控件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上。在加载 **contentPath** 属性的语句中，将斜体文本替换为 FLV 文件在 FCS 上的名称和位置。

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.bufferTime = 15;
my_FLVPlayback.bufferingBarHidesAndDisablesOthers = true;
my_FLVPlayback.contentPath = "rtmp://host_name/somefolder/vid_name.flv";
```

## 另请参见

[FLVPlayback.bufferingBar](#)



# FLVPlayback.BUFFERTIME

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.bufferTime*

## 说明

属性；一个数字，指定开始播放视频流前要在内存中缓冲的秒数。对于通过 RTMP 流式处理的 FLV 文件（它们并不下载并且只在内存中缓冲），从默认值 0.1 增加此设置可能十分重要。对于通过 HTTP 渐进式下载的 FLV 文件，增加该值只会带来很小的好处，尽管它可以改善在旧式、速度较慢的计算机上查看高质量视频的查看效果。



此属性并不指定在开始回放前要下载的 FLV 文件量。

## 示例

以下示例将缓冲时间设置为 8 秒，以用于从 FCS 流式加载的 FLV 文件。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到“动作”面板中时间轴的第 1 帧上。在加载 `contentPath` 属性的语句中，将斜体文本替换为 FLV 文件在 FCS 上的名称和位置。

/\*\*

  要求：

  - 舞台上的 FLVPlayback 组件具有实例名称 my\_FLVPlayback

\*/

```
import mx.video.*;
```

```
my_FLVPlayback.bufferTime = 8;
```

```
my_FLVPlayback.contentPath = "rtmp://host_name/somefolder/vid_name.flv";
```

# FLVPlayback.bytesLoaded

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.bytesLoaded

## 说明

属性；一个数字，指示 HTTP 下载的下载范围，以字节数表示。在没有流时、流来自 FCS 时或者没有可用信息时，返回 -1。返回值只用于 HTTP 下载。只读。

## 示例

以下示例显示 bytesLoaded 属性的初始值以及 ready 事件发生时该属性的值。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 trace("State is " + eventObject.state + "; ready to play");
 // 显示此时已加载的字节数
 trace("Bytes loaded: " + my_FLVPlayback.bytesLoaded);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
trace("Bytes loaded: " + my_FLVPlayback.bytesLoaded); // 如果加载没有开始，则为 -1
```

# FLVPlayback.bytesTotal

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.bytesTotal

## 说明

属性；一个数字，指定 HTTP 下载的总下载字节数。在没有流时、流来自 FCS 时或者没有可用信息时，返回 -1。返回值只用于 HTTP 下载。只读。

## 示例

以下示例使用 bytesTotal 属性显示要为 HTTP 下载加载的字节数。在 metadataReceived 事件发生时，事件处理函数在文本区域 **my\_ta** 中显示该值。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。将一个 TextArea 组件拖到舞台中的 FLVPlayback 实例下，然后为其指定实例名称 **my\_ta**。然后将以下代码添加到“动作”面板中时间轴的第 1 帧内：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 - 舞台上的 TextArea 组件具有实例名称 my_ta
*/
import mx.video.*;
my_ta.visible = false;
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
 my_ta.text = "Loading: " + my_FLVPlayback.bytesTotal + " bytes.";
 my_ta.visible = true;
};
my_FLVPlayback.addEventListener("metadataReceived", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

# FLVPlayback.close

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.close = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("close", listenerObject);
```

## 说明

事件：**FLVPlayback** 实例在以下情况下调度此事件：在它通过超时或者通过调用 `closeVideoPlayer()` 方法关闭 **NetConnection** 时；在您调用 `load()` 方法或 `play()` 方法或者设置 `contentPath` 并导致 RTMP 连接因此关闭时。只有在从 **FCS** 或 **FVSS** 流式加载时，**FLVPlayback** 实例才会调度此事件。事件对象具有 `state` 和 `playheadTime` 属性。

事件具有 `vp` 属性，它是事件适用的视频播放器的索引号。

## 示例

以下示例假定正在从 **FCS** 或 **FVSS** 播放 **FLV** 流文件。在 **FLV** 文件完成后，`complete` 事件的侦听器将 `contentPath` 属性设置为新 **FLV** 文件的位置，这将对用于第一个 **FLV** 文件的 **RTMP** 连接触发 `close` 事件。`close` 事件的侦听器显示发生事件的视频播放器的索引号。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上。在加载 `contentPath` 属性的语句中，将斜体文本替换为 **FLV** 文件在 **FCS** 上的名称和位置。

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
*/
import mx.video.*;
var listenerObject:Object = new Object();
// 侦听 RTMP 连接的 close 事件；显示视频播放器的索引
listenerObject.close = function(eventObject:Object) {
 trace("Closed connection for video player: " + eventObject.vp);
};
my_FLVplybk.addEventListener("close", listenerObject);
```

```
// 侦听 complete 事件; 播放新的 FLV
listenerObject.complete = function(eventObject:Object) {
 if (my_FLVPlayback.contentPath != "http://www.helpexamples.com/flash/video/
 water.flv") {
 my_FLVPlayback.play("http://www.helpexamples.com/flash/video/water.flv");
 }
};
my_FLVPlayback.addEventListener("complete", listenerObject);
my_FLVPlayback.contentPath = "rtmp://my_servername/my_application/stream.flv";
```

### 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.closeVideoPlayer\(\)](#)、[FLVPlayback.contentPath](#)、[FLVPlayback.load\(\)](#)、[FLVPlayback.play\(\)](#)、[FLVPlayback.visibleVideoPlayerIndex](#)

## FLVPlayback.closeVideoPlayer()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.closeVideoPlayer(index:Number)
```

### 参数

*index* 一个数字，表示要关闭的视频播放器的索引。

### 返回

已关闭的 **VideoPlayer** 对象。

### 说明

方法；关闭 **NetStream** 并删除由 *index* 参数指定的视频播放器。如果已关闭的视频播放器是活动或可见的视频播放器，则 **FLVPlayback** 实例将活动和 / 或可见的视频播放器设置为默认播放器（索引为 0）。您无法关闭默认播放器；如果试图这样做，将导致组件引发错误。

## 示例

以下示例将创建两个视频播放器，用来在一个 **FLVPlayback** 实例中连续播放两个 **FLV** 文件。在第二个 **FLV** 文件完成后，**complete** 事件的事件处理函数调用 **closeVideoPlayer()** 方法以关闭第二个播放器。如果您单击“播放”按钮以再次播放 **FLV** 文件，会看到第二个播放器的视频播放器消失，这导致该组件引发一个错误 (**VideoError**) 并显示一条消息，该消息指出 **FLVPlayback** 实例找不到 **FLV** 文件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
// 为默认播放器指定 FLV 的名称和位置
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 // 添加另一个视频播放器，并指定其 FLV 的名称和位置
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
 // 重置为默认的视频播放器，它将自动播放其 FLV
 my_FLVPlybk.activeVideoPlayerIndex = 0;
};
my_FLVPlybk.addEventListener("ready", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
 // 如果 complete 事件是针对第二个 FLV，则使默认播放器活动并可见
 if (eventObject.vp == 1) {
 my_FLVPlybk.activeVideoPlayerIndex = 0;
 my_FLVPlybk.visibleVideoPlayerIndex = 0;
 my_FLVPlybk.closeVideoPlayer(1); // 关闭第二个视频播放器
 } else { // 使第二个播放器活动并可见，并且播放 FLV
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.visibleVideoPlayerIndex = 1;
 my_FLVPlybk.play();
 }
};
// 添加 complete 事件的侦听器
my_FLVPlybk.addEventListener("complete", listenerObject);
```

## 另请参见

[FLVPlayback.close](#)、[FLVPlayback.activeVideoPlayerIndex](#)、  
[FLVPlayback.visibleVideoPlayerIndex](#)

# FLVPlayback.complete

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("complete", listenerObject);
```

## 说明

事件：播放完成（由于播放器到达了 FLV 文件的末端）时调度。如果您调用 stop() 或 pause() 方法或者单击相应控件，则组件不调度该事件。该事件对象具有 state 和 playheadTime 属性。

如果应用程序使用渐进式下载、不显式设置 totalTime 属性并且下载未在元数据中指定持续时间的 FLV 文件，则视频播放器会在调度此事件前将 totalTime 设置为近似的总时间值。

视频播放器还会调度 stateChange 和 stopped 事件。

## 示例

以下示例使用 playheadTime 属性在“输出”面板中显示 complete 事件发生时 FLV 文件的播放运行时间。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
*/
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object):Void {
 trace("Elapsed play time at completion is: " + my_FLVplybk.playheadTime);
};
my_FLVplybk.addEventListener("complete", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)、[FLVPlayback.stopped](#)、[FLVPlayback.totalTime](#)

## FLVPlayback.CONNECTION\_ERROR

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

`mx.video.FLVPlayback.CONNECTION_ERROR`

说明

一个只读的 `FLVPlayback` 类属性，它包含字符串常数 `"connectionError"`。您可以将此属性与 `state` 属性进行比较，以确定是否已发生连接错误状态。

示例

以下示例通过在 `contentPath` 属性中指定无效的 FLV 文件名 (`nosuch.flv`)，强制发生连接错误。该示例使用 `CONNECTION_ERROR` 属性检测 `stateChange` 事件的侦听器中的错误。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
no_such.flv";
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(my_FLVPlybk.state == FLVPlayback.CONNECTION_ERROR)
 trace("State: " + FLVPlayback.CONNECTION_ERROR);
}
my_FLVPlybk.addEventListener("stateChange", listenerObject);
```

另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)



# FLVPlayback.contentPath

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.contentPath*

## 说明

属性：一个字符串，指定要进行流式处理的 FLV 文件的 URL 以及如何对其进行流式处理。URL 可以是指向 FLV 文件的 HTTP URL、指向流的 RTMP URL，也可以是指向 XML 文件的 HTTP URL。

如果您通过“组件”检查器或“属性”检查器设置此属性，则 FLV 文件在发生下一个 `MovieClip.onEnterFrame` 事件时开始加载和播放。在延迟期间可以设置影响加载的 `isLive`、`autoPlay` 和 `cuePoints` 等属性。它还允许放置在第一帧上的 `ActionScript` 在 `FLVPlayback` 组件开始播放前对其产生影响。

如果您通过 `ActionScript` 设置此属性，则 `FLVPlayback` 实例将关闭当前的 FLV 文件并立即开始加载新的 FLV 文件。`autoPlay`、`totalTime` 和 `isLive` 属性影响新 FLV 文件的加载方式，因此，如果您要设置这些属性，则必须在设置 `contentPath` 属性前设置它们。

通过将 `autoPlay` 属性设置为 `false`，可以防止自动播放新的 FLV 文件。

## 示例

以下示例将在 `ActionScript` 中设置 `contentPath` 属性，以指定要播放的 FLV 文件的位置。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。不要在“组件”检查器中将某个值赋给 `contentPath` 参数。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(event:Object):Void {
 my_FLVPlayback.setSize(my_FLVPlayback.preferredWidth,
 my_FLVPlayback.preferredHeight);
}
my_FLVPlayback.addEventListener("metadataReceived", listenerObject);
```

另请参见

[FLVPlayback.autoPlay](#)、[FLVPlayback.isLive](#)、[FLVPlayback.play\(\)](#)、[FLVPlayback.totalTime](#)

## FLVPlayback.cuePoint

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("cuePoint", listenerObject);
```

说明

事件：到达提示点时调度。事件对象具有一个 `info` 属性，该属性包含 `NetStream.onCuePoint` 回调为 FLV 文件提示点接收的 `info` 对象。对于 **ActionScript** 提示点，它包含已传递到 **ActionScript** 提示点方法或属性中的对象。

此事件具有 `vp` 属性，它是适用于事件的视频播放器的索引号。

示例

以下示例向某个 FLV 文件中添加两个 **ActionScript** 提示点。该示例使用 `cuePoint` 参数添加第一个提示点，并使用 `time` 和 `name` 参数添加第二个提示点。出现每个提示点时，`cuePoint` 事件的侦听器会在文本区域中显示 `playheadTime` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。将一个 **TextArea** 组件拖到舞台中的 **FLVPlayback** 实例下，然后为其指定实例名称 **my\_ta**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 * - 舞台上的 TextArea 组件具有实例名称 my_ta
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_ta.visible = false;
// 创建提示点对象
```

```

var cuePt:Object = new Object(); // 创建提示点对象
cuePt.time = 1.444;
cuePt.name = "elapsed_time";
cuePt.type = "actionsript";
my_FLVPlayback.addASCuePoint(cuePt); // 添加 AS 提示点
// 使用 time 和 name 参数添加第二个 AS 提示点
my_FLVPlayback.addASCuePoint(5.3, "elapsed_time2");
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 my_ta.text = "Cue at: " + eventObject.info.time + " occurred";
 my_ta.visible = true;
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);

```

### 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.visibleVideoPlayerIndex](#)

## FLVPlayback.cuePoints

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlayback.cuePoints*

### 说明

属性：一个数组，说明 **ActionScript** 提示点和已禁用的嵌入式 **FLV** 文件提示点。此属性专门为“组件”检查器使用而创建，通过任何其它方式进行设置将不起作用。其值只影响加载的第一个 **FLV** 文件，并且只有通过“组件”检查器或“属性”检查器中设置 `contentPath` 属性来进行加载时才会起作用。



在 **ActionScript** 中无法访问此属性。若要在 **ActionScript** 中访问提示点信息，请使用 `metadata` 属性。

若要使用 **ActionScript** 添加、删除、启用或禁用提示点，请使用 `addASCuePoint()`、`removeASCuePoint()` 或 `setFLVCuePointEnabled()`。

## 另请参见

[FLVPlayback.contentPath](#)、[FLVPlayback.addASCuePoint\(\)](#)、[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.Findnearestcuepoint\(\)](#)、[FLVPlayback.findNextCuePointWithName\(\)](#)、[FLVPlayback.isFLVCuePointEnabled\(\)](#)、[FLVPlayback.metadata](#)、[FLVPlayback.metadataReceived](#)、[FLVPlayback.removeASCuePoint\(\)](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.setFLVCuePointEnabled\(\)](#)

# FLVPlayback.DISCONNECTED

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
mx.video.FLVPlayback.DISCONNECTED
```

## 说明

一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "disconnected"。您可以将此属性与 `state` 属性进行比较，以确定是否存在断开连接状态。

**FLVPlayback** 实例在您设置 `contentPath` 属性前处于断开连接状态。

## 示例

以下示例只在“输出”面板中显示一条消息，该消息确认 **FLVPlayback** 实例在设置 `contentPath` 属性前处于断开连接状态。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;

if(my_FLVPlybk.state == FLVPlayback.DISCONNECTED)
 trace("FLVPlayback instance is currently disconnected");
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.contentPath](#)、[FLVPlayback.state](#)

# FLVPlayback.EVENT

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.FLVPlayback.EVENT`

## 说明

一个只读的 `FLVPlayback` 类属性，它包含字符串常数 `"event"`。可以将此属性用作 `findCuePoint()` 和 `findNearestCuePoint()` 方法的 `type` 参数。

## 示例

以下示例使用 `FLVPlayback.EVENT` 属性来指定它想要查找名为 `myCue`、类型为 `event` 的提示点。它显示返回的提示点的 `name`、`time` 和 `type` 属性，并且 `cuePoint` 侦听器在提示点出现时在“输出”面板中显示 **“Hit it!”**。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **`my_FLVPlayback`**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 plane_cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 if(rtn_cuePt = my_FLVPlayback.findCuePoint("myCue", FLVPlayback.EVENT)){
 trace("Cue point name is: " + rtn_cuePt.name);
 trace("Cue point time is: " + rtn_cuePt.time);
 trace("Cue point type is: " + rtn_cuePt.type);
 }
}
my_FLVPlayback.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 if(eventObject.info.name == "myCue")
 trace("Hit it!");
}
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
```

## 另请参见

[FLVPlayback.findCuePoint\(\)](#)

# FLVPlayback.fastForward

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.fastForward = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("fastForward", listenerObject);
```

事件：在播放头的位置由于手动或通过 **ActionScript** 进行搜索或者通过单击 **ForwardButton** 控件而前移时调度。事件对象具有 state、playheadTime 和 vp 属性。playheadTime 属性反映目标时间，而 vp 属性是事件适用的视频播放器的索引号。

FLVPlayback 实例也调度 seek 和 playheadUpdate 事件。

## 示例

以下示例在 fastForward 事件发生时捕获该事件的出现，并且在“输出”面板中显示播放头已运行时间。在 ready 事件发生时，对 seekPercent() 方法的调用将触发 fastForward 事件。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
*/
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVplybk.seekPercent(35);
};
my_FLVplybk.addEventListener("ready", listenerObject);

listenerObject.fastForward = function(eventObject:Object):Void {
 trace("fastforward event; playhead time is: " + eventObject.playheadTime);
};
my_FLVplybk.addEventListener("fastForward", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.forwardButton](#)、[FLVPlayback.seek](#)、[FLVPlayback.seek\(\)](#)、[FLVPlayback.seekBar](#)、[FLVPlayback.seekPercent\(\)](#)、[FLVPlayback.seekSeconds\(\)](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevOffset](#)、[FLVPlayback.state](#)、[FLVPlayback.playheadTime](#)

# FLVPlayback.findCuePoint()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVplybk.findCuePoint(time:Number[, type:String]):Object
my_FLVplybk.findCuePoint(name:String[, type:String]):Object
my_FLVplybk.findCuePoint(cuePoint:Object[, type:String]):Object
```

## 参数

*time* 一个数字，表示要搜索的提示点的时间，以秒为单位。该方法仅使用前三个小数位，而对提供的所有多余小数位进行舍入。该方法返回与此时间匹配的提示点。如果多个 **ActionScript** 提示点具有相同时间，则该方法只返回按字母顺序排在最前的名称。如果未找到匹配项，它将返回 `null`。

*name* 一个字符串，表示要搜索的提示点的名称。该方法返回与此名称匹配的第一个提示点，或者在没有找到匹配项时返回 `null`。

*cuePoint* 一个对象，表示包含要搜索的 *time* 和 *name* 属性的提示点对象。如果 *name* 属性没有任何值或者为 `null`，则搜索的行为就像该参数是表示要搜索的时间的数字一样。如果 *time* 属性没有任何值、为 `null` 或者小于零，则搜索的行为就像该参数是包含要搜索的名称的字符串一样。如果提供了 *time* 和 *name* 属性并且存在与这两个值匹配的提示点，则该方法返回此对象。否则，该方法返回 `null`。

*type* 可选。一个字符串，指定要搜索的提示点的类型。该参数的可能值包括：  
"actionscript"、"all"、"event"、"flv" 或 "navigation"。可以使用以下类属性指定这些值：[FLVPlayback.ACTIONSCRIPT](#)、[FLVPlayback.ALL](#)、[FLVPlayback.EVENT](#)、[FLVPlayback.FLV](#) 和 [FLVPlayback.NAVIGATION](#)。如果未指定此参数，则默认值是 "all"，这意味着该方法将搜索所有提示点类型。

## 返回

一个对象，它是找到的提示点对象的副本，具有以下附加属性：

`array` 由搜索过的提示点组成的数组。只能将此数组视为只读，因为在该数组中添加、删除或编辑对象时会导致提示点故障。

`index` 返回的提示点在数组中的索引。

如果未找到匹配项，则返回 `null`。

## 说明

方法；查找属于 `type` 参数所指定类型并且具有您通过这些参数指定的时间、名称或二者组合的提示点。

如果您没有提供提示点的时间值或名称值，或者时间为 `null`、未定义或小于零并且名称是 `null` 或未定义，则该方法将引发 **VideoError** 错误 1002。有关更多信息，请参见第 642 页的“**VideoError** 类”。

该方法在搜索中包含了禁用的提示点。请使用 `isFLVCuePointEnabled()` 方法来确定提示点是否已禁用。

## 示例

以下示例向某个 FLV 文件中添加两个 **ActionScript** 提示点，然后调用 `findCuePoint()` 方法三次。第一个调用查找时间为 7.748 的导航提示点。第二个调用只使用名称值查找提示点 "ASCue1"。第三个调用使用一个提示点对象，该对象指定时间 10 以及名称 "ASCue2"。该示例在第三个调用后，显示返回的提示点对象的内容，包括已搜索到的提示点（在此示例中是 **ActionScript** 提示点）组成的数组。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
// 创建提示点对象
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var cuePt:Object = new Object();
var rtn_cuePt:Object = new Object(); // 创建对象以便返回
cuePt.time = 4.444;
cuePt.name = "ASCue1";
my_FLVPlayback.addASCuePoint(cuePt); // 添加 AS 提示点
// 使用 time 和 name 参数添加第二个 AS 提示点
my_FLVPlayback.addASCuePoint(10, "ASCue2");
// 使用 time 属性查找导航提示点
rtn_cuePt = my_FLVPlayback.findCuePoint(7.748, FLVPlayback.NAVIGATION);
```



```

// 只使用名称查找 ActionScript 提示点
rtn_cuePt = my_FLVPlaybk.findCuePoint("AScue1");
// 使用提示点对象查找 ActionScript 提示点
cuePt.time = 10;
cuePt.name = "AScue2";
rtn_cuePt = my_FLVPlaybk.findCuePoint(cuePt, FLVPlayback.ACTIONSCRIPT);
// 查看返回的提示点对象包含的内容
for (i in rtn_cuePt) {
 // 如果返回的是数组对象，则展开它并显示内容
 if (typeof rtn_cuePt[i] == "object") {
 tracer(rtn_cuePt[i]);
 } else {
 trace(i + " " + rtn_cuePt[i]);
 }
}
// 显示提示点组成的数组
function tracer(cuepts:Array) {
 for (i in cuepts) {
 if (typeof cuepts[i] == "object") { // 如果对象嵌套
 tracer(cuepts[i]); // 显示对象
 } else {
 // 显示名称：值对
 trace(i + " " + cuepts[i]);
 }
 }
}

```

## 另请参见

[FLVPlayback.addASCuePoint\(\)](#)、[FLVPlayback.cuePoints](#)、  
[FLVPlayback.Findnearestcuepoint\(\)](#)、[FLVPlayback.findNextCuePointWithName\(\)](#)、  
[FLVPlayback.isFLVCuePointEnabled\(\)](#)、[FLVPlayback.removeASCuePoint\(\)](#)、  
[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#)、  
[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.setFLVCuePointEnabled\(\)](#)

# FLVPlayback.Findnearestcuepoint()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVplybk.findNearestCuePoint(time:Number[, type:String]):Object
my_FLVplybk.findNearestCuePoint(name:String[, type:String]):Object
my_FLVplybk.findNearestCuePoint(cuePoint:Object[, type:String]):Object
```

## 参数

*time* 一个数字，表示要搜索的提示点的时间，以秒为单位。该方法仅使用前三个小数位，而对提供的所有多余小数位进行舍入。该方法返回与此时间匹配的提示点，或者返回属于所指定类型的最接近的较早提示点。如果多个提示点具有相同的时间（只有在 **ActionScript** 提示点上才会出现此情况），则该方法只返回按字母顺序排在最前的名称。如果未找到匹配项，它将返回 `null`。

*name* 一个字符串，表示要搜索的提示点的名称。该方法返回与此名称匹配的提示点，或者在没有找到匹配项时返回 `null`。

*cuePoint* 一个对象，表示包含要搜索的 `time` 和 `name` 属性的提示点对象。如果您提供了某一时间并且 `name` 属性没有任何值或者为 `null`，则搜索的行为就像该参数是表示要搜索的时间的数字一样。如果您提供了一个名称并且 `time` 属性没有任何值、为 `null` 或者小于零，则搜索的行为就像该参数是包含要搜索的名称的字符串一样。如果提供了 `time` 和 `name` 属性并且存在与这两个值匹配的提示点，则该方法返回此对象。如果它没有找到与时间和名称都匹配的匹配项，则返回与名称匹配并且具有较早时间的第一个提示点。如果没有具有该名称的较早提示点，则返回与该名称匹配的提示点。否则，该方法返回 `null`。

*type* 可选。一个字符串，指定要搜索的提示点的类型。该参数的可能值包括：`"actionscript"`、`"all"`、`"event"`、`"flv"` 或 `"navigation"`。可以使用以下类属性指定这些值：`FLVPlayback.ACTIONSCRIPT`、`FLVPlayback.ALL`、`FLVPlayback.EVENT`、`FLVPlayback.FLV` 和 `FLVPlayback.NAVIGATION`。如果未指定此参数，则默认值是 `"all"`，这意味着该方法将搜索所有提示点类型。

## 返回

一个对象，它是找到的提示点对象的副本，具有以下附加属性：

**array** 由搜索到的提示点组成的数组。只能将此数组视为只读，因为在该数组中添加、删除或编辑对象时会导致提示点故障。

**index** 返回的提示点在数组中的索引。

如果未找到匹配项，则返回 `null`。

## 说明

方法：查找匹配或早于您指定的时间的提示点；或者，如果您同时指定了时间和名称并且没有较早的匹配该名称的提示点，则查找匹配该名称的提示点。否则，该方法返回 `null`。

该方法在搜索中包含了禁用的提示点。请使用 `isFLVCuePointEnabled()` 方法来确定提示点是否已禁用。

如果时间为 `null`、未定义或小于 0，并且名称为 `null` 或未定义，则该方法会引发一个 **VideoError** 错误 (1002)。

## 示例

以下示例为 FLV 文件创建一个时间在 4.07 秒的 **ActionScript** 提示点。当此提示点出现时，`cuePoint` 事件处理函数调用 `findNearestCuePoint()` 方法，以找到最接近稍后的 5 秒的任何类型的提示点。“输出”面板显示返回的提示点的名称、时间和类型。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var cuePt:Object = new Object(); // 创建提示点对象
cuePt.time = 4.07;
cuePt.name = "ASpt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
function cuePoint(eventObject:Object):Void {
 if (eventObject.info.name == "ASpt1") {
 var rtn_obj:Object = new Object();
 rtn_obj = my_FLVPlybk.findNearestCuePoint(eventObject.info.time + 5);
 trace("Cue point name is: " + rtn_obj.name);
 trace("Cue point time is: " + rtn_obj.time);
 trace("Cue point type is: " + rtn_obj.type);
 }
}
my_FLVPlybk.addEventListener("cuePoint", cuePoint);
```

## 另请参见

[FLVPlayback.addASCuePoint\(\)](#)、[FLVPlayback.cuePoints](#)、[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.findNextCuePointWithName\(\)](#)、[FLVPlayback.isFLVCuePointEnabled\(\)](#)、[FLVPlayback.removeASCuePoint\(\)](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.setFLVCuePointEnabled\(\)](#)

# FLVPlayback.findNextCuePointWithName()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.findNextCuePointWithName(my_cuePoint:Object)
```

## 参数

*my\_cuePoint* 一个提示点对象，它由 [findCuePoint\(\)](#) 方法、[findNearestCuePoint\(\)](#) 方法或者以前对此方法的调用返回。

## 返回

一个对象，它是找到的提示点对象的副本，具有以下附加属性：

**array** 由搜索到的提示点组成的数组。只能将此数组视为只读，因为在该数组中添加、删除或编辑对象时会导致提示点故障。

**index** 返回的提示点在数组中的索引。

如果未找到匹配项，则返回 `null`。

## 说明

方法：在 *my\_cuePoint.array* 中查找与 *my\_cuePoint.name* 同名的下一个提示点。

*my\_cuePoint* 对象必须是由 [findCuePoint\(\)](#) 方法、[findNearestCuePoint\(\)](#) 方法或者以前对此方法的调用返回的提示点对象。此方法使用 `array` 属性（上面这些方法将该属性添加到提示点对象）。

该方法在搜索中包含了禁用的提示点。请使用 [isFLVCuePointEnabled\(\)](#) 方法来确定提示点是否已禁用。

如果在数组中没有具有匹配名称的其它提示点，则返回 `null`。

## 示例

以下示例创建三个名称为 "transition" 的 **ActionScript** 提示点。在 ready 事件发生时, 事件处理函数调用 findCuePoint() 方法以查找具有此名称的第一个提示点。如果它找到一个匹配项, 则调用 findNextName() 函数, 而该函数调用 findNextCuePointWithName() 方法, 并且传递返回的提示点对象 (rtn\_obj) 以找到具有相同名称的任何其它提示点。

将一个 **FLVPlayback** 组件拖到舞台上, 并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的 “动作” 面板上:

```
/**
 * 要求:
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var cuePt:Object = new Object(); // 创建提示点对象
cuePt.time = 6.27;
cuePt.name = "transition";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
cuePt.time = 7.06;
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
cuePt.time = 11.13;
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
var listenerObject:Object = new Object();
listenerObject.ready = function():Void {
 var rtn_obj:Object = new Object();
 // 只使用名称字符串查找提示点
 if (rtn_obj = my_FLVPlybk.findCuePoint("transition")) {
 trace("Cue point name is: " + rtn_obj.name);
 trace("Cue point time is: " + rtn_obj.time);
 trace("Cue point type is: " + rtn_obj.type);
 findNextName(rtn_obj);
 }
}
my_FLVPlybk.addEventListener("ready", listenerObject);
// 查找具有相同名称的其它提示点
function findNextName(cuePt:Object):Void {
 while(cuePt = my_FLVPlybk.findNextCuePointWithName(cuePt)) {
 trace("Cue point name is: " + cuePt.name);
 trace("Cue point time is: " + cuePt.time);
 trace("Cue point type is: " + cuePt.type);
 }
}
```

另请参见

`FLVPlayback.addASCuePoint()`、`FLVPlayback.cuePoints`、  
`FLVPlayback.findCuePoint()`、`FLVPlayback.Findnearestcuepoint()`、  
`FLVPlayback.isFLVCuePointEnabled()`、`FLVPlayback.removeASCuePoint()`、  
`FLVPlayback.seekToNavCuePoint()`、`FLVPlayback.seekToNextNavCuePoint()`、  
`FLVPlayback.seekToPrevNavCuePoint()`、`FLVPlayback.setFLVCuePointEnabled()`

## FLVPlayback.FLV

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

`mx.video.FLVPlayback.FLV`

说明

一个只读的 `FLVPlayback` 类属性，它包含字符串常数 `"flv"`。可以将此属性用作 `findCuePoint()` 和 `findNearestCuePoint()` 方法的 `type` 参数。

示例

以下示例在 `FLV` 文件提示点中查找名称为 `point2`、时间为 `7.748` 的提示点，并且显示找到的时间和类型。 `FLV` 文件提示点是导航提示点和事件提示点。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **`my_FLVPlybk`**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
// 创建提示点对象
var listenerObject = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 var cuePt:Object = new Object(); // 创建提示点对象
 cuePt.name = "point3";
 cuePt.time = 16.02;
 if(cuePt = my_FLVPlybk.findCuePoint(cuePt, FLVPlayback.FLV)) // 查找提示点
 trace("found a " + cuePt.type + " cue point at " + cuePt.time);
}
```

```

 else
 trace("cue point not found");
 }
 my_FLVPlayback.addEventListener("ready", listenerObject);

```

### 另请参见

[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.Findnearestcuepoint\(\)](#)

## FLVPlayback.forwardButton

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlayback*.forwardButton

### 说明

属性：作为 **Forward** 按钮控件的 **MovieClip** 对象。有关将 FLV 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 FLV 回放自定义用户界面组件的外观”。

### 示例

以下示例使用 backButton、forwardButton、playButton、pauseButton 和 stopButton 属性，来将各个 FLV 回放自定义用户界面控件附加到 FLVPlayback 组件。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**，然后在“组件”检查器中将 skin 参数设置为“无”。接下来，添加以下各个 FLV 自定义用户界面组件，并为其指定括号中显示的实例名称 **BackButton (my\_bkbtn)**、**ForwardButton (my\_fwdbtn)**、**PlayButton (my\_plybtn)**、**PauseButton (my\_pausbtn)** 和 **StopButton (my\_stopbtn)**。然后将以下几行代码添加到“动作”面板中：

```

/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 * - 库中有 FLV 自定义用户界面的 BackButton、ForwardButton、PlayButton、PauseButton
 和 StopButton 组件
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbtn;
my_FLVPlayback.forwardButton = my_fwdbtn;
my_FLVPlayback.playButton = my_plybtn;

```

```
my_FLVPlaybk.pauseButton = my_pausbbtn;
my_FLVPlaybk.stopButton = my_stopbbtn;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.fastForward](#)、[FLVPlayback.seek](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

## FLVPlayback.getVideoPlayer()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVplaybk.getVideoPlayer(index:Number)
```

### 返回

一个 `VideoPlayer` 对象。

### 说明

方法；获取由 `index` 指定的视频播放器。如果可能，最好使用 `FLVPlayback` 方法和属性访问 `VideoPlayer` 方法和属性。每个 `VideoPlayer._name` 属性都是它的索引。

### 示例

以下示例使用两个视频播放器来播放两个 FLV 文件。在第二个 FLV 文件触发 `ready` 事件时，该示例调用 `getVideoPlayer()` 方法获取视频播放器编号 1，并将其 `_alpha` 属性设置为 50。这导致该播放器中的 FLV 文件 (`plane_cuepoints`) 是透明的，并使两个 FLV 文件同时可见。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlaybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlaybk
*/
my_FLVPlaybk.load("http://www.helpexamples.com/flash/video/cuepoints.flv");
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
```



```

 if (eventObject.target.contentPath == "http://www.helpexamples.com/
flash/video/cuepoints.flv") {
 // 以下代码将在第一个 FLV 就绪后引发
 my_FLVPlayback.activeVideoPlayerIndex = 1;
 my_FLVPlayback.load("http://www.helpexamples.com/flash/video/
plane_cuepoints.flv");
 } else {
 // 以下代码将在第二个 FLV 就绪后引发
 eventObject.target.activeVideoPlayerIndex = 0;
 eventObject.target.play();
 eventObject.target.activeVideoPlayerIndex = 1;
 eventObject.target.play();
 var layerOnTop:MovieClip = eventObject.target.getVideoPlayer(1);
 layerOnTop._alpha = 50;
 layerOnTop._visible = true;
 }
 }
}
my_FLVPlayback.addEventListener("ready", listenerObject);

```

### 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.bringVideoPlayerToFront\(\)](#)、[FLVPlayback.visibleVideoPlayerIndex](#)

## FLVPlayback.height

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlayback.height*

### 说明

属性；一个数字，指定源 **FLVPlayback** 实例的高度。此属性只影响 **FLVPlayback** 实例的高度，并且不包括可加载的外观 **SWF** 文件的高度。使用 **FLVPlayback.height** 属性，而不使用 **MovieClip.\_height** 属性，因为如果加载外观 **SWF** 文件，**\_height** 属性可能给出不同的值。

## 示例

以下示例设置 `width` 和 `height` 属性，以更改视频播放器的大小。它首先将 `maintainAspectRatio` 属性设置为 `false`，以防止视频播放器在尺寸发生变化时自动调整大小。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.maintainAspectRatio = false;
my_FLVPlybk.width = 300;
my_FLVPlybk.height = 350;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.preferredHeight](#)、[FLVPlayback.preferredWidth](#)、[FLVPlayback.maintainAspectRatio](#)、[FLVPlayback.resize](#)、[FLVPlayback.setSize\(\)](#)、[FLVPlayback.width](#)

# FLVPlayback.idleTimeout

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk.idleTimeout*

## 说明

属性；Flash 结束由于播放暂停或停止而导致 FCS 连接空闲之前的时间，以毫秒为单位。此属性对通过 HTTP 的 FLV 文件下载没有影响。

如果设置了此属性，则视频流空闲时，会用新值重新开始超时期间。

默认值为 300,000，即 5 分钟。

## 示例

以下示例假定正在从 FCS 或 FVSS 播放 FLV 流文件。该示例将 `idleTimeout` 属性设置为 10 毫秒这个较低的值，这将触发超时，并因此对 RTMP 连接触发 `close` 事件。`close` 事件的侦听器显示发生事件的视频播放器的索引号。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。在“组件”检查器中，为 `contentPath` 参数赋予一个值，该值指定来自 FCS 或 FVSS 的流式 FLV 文件的位置。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.idleTimeout = 10;
var listenerObject:Object = new Object();
// 侦听 RTMP 连接的 close 事件；显示视频播放器的索引
listenerObject.close = function(eventObject:Object) {
 trace("Closed connection for video player: " + eventObject.vp);
};
my_FLVPlybk.addEventListener("close", listenerObject);
```

## 另请参见

[FLVPlayback.close](#)

# FLVPlayback.isFLVCuePointEnabled()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVplybk.isFLVCuePointEnabled(time:Number)
my_FLVplybk.isFLVCuePointEnabled(name:String)
my_FLVplybk.isFLVCuePointEnabled(cuePoint:Object)
```

## 参数

*time* 一个数字，表示要搜索的提示点的时间，以秒为单位。

*name* 一个字符串，表示要搜索的提示点的名称。

*cuePoint* 一个提示点对象，该对象具有提示点的 `time` 和 `name` 属性。该方法不检查传入提示点对象上的任何其它属性。如果未定义 `time` 或 `name`，则该方法只使用已定义的属性。

## 返回

一个布尔值，如果找到了提示点并禁用了提示点，则为 `false`；如果提示点未禁用或不存在，则为 `true`。如果给定的时间未定义、为 `null`、小于 **0** 或者只提供了提示点名称，则只有在具有此名称的所有提示点都被禁用的情况下，该方法才返回 `false`。

## 说明

方法：如果禁用 **FLV** 文件嵌入式提示点，则返回 `false`。您可以通过以下两个方法之一禁用提示点：通过 “**Flash** 视频提示点” 对话框设置 `cuePoints` 属性，或者通过调用 `setFLVCuePointEnabled()` 方法。

只有在 `metadataLoaded` 属性为 `true`、`metadata` 属性不为 `null` 或者在 `metadataReceived` 事件发生后，此函数的返回值才有意义。`metadataLoaded` 为 `false` 时，此函数始终返回 `true`。

## 示例

以下示例在发生 `ready` 事件时禁用 `point2` 提示点。在发生第一个 `cuePoint` 事件时，事件处理函数调用 `isFLVCuePointEnabled()` 方法以确定提示点是否被禁用，如果确定该提示点被禁用，则事件处理函数启用它。**FLV** 文件包含以下嵌入的提示点：**point1**, 00:00:00:418 ; **point2**, 00:00:07.748 ; **point3**, 00:00:16:020。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的 “动作” 面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
function ready(eventObject:Object) {
 my_FLVPlybk.setFLVCuePointEnabled(false, "point2");
}
my_FLVPlybk.addEventListener("ready", ready);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
 trace("Elapsed time in seconds: " + my_FLVPlybk.playheadTime);
 trace("Cue point name is: " + eventObject.info.name);
 trace("Cue point type is: " + eventObject.info.type);
 if (my_FLVPlybk.isFLVCuePointEnabled("point2") == false) {
 my_FLVPlybk.setFLVCuePointEnabled(true, "point2");
 }
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

另请参见

[FLVPlayback.cuePoint](#)、[FLVPlayback.findCuePoint\(\)](#)、  
[FLVPlayback.findNearestCuePoint\(\)](#)、[FLVPlayback.findNextCuePointWithName\(\)](#)、  
[FLVPlayback.setFLVCuePointEnabled\(\)](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、  
[FLVPlayback.seekToNextNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)

## FLVPlayback.isLive

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

*my\_FLVPlayback.isLive*

说明

属性：一个布尔值，如果是实时视频流，则为 `true`。只有在从 **FCS** 或 **FVSS** 进行流式加载时，此属性才有效。对于 **HTTP** 下载，会忽略此属性的值。

如果在加载新 **FLV** 文件期间设置该属性，则直到为新的 **FLV** 文件设置了 `contentPath` 参数，该属性才起作用。

示例

以下示例假定正在从 **FCS** 播放实时流。在发生 `playing` 事件时，该示例显示 `isLive` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上。在加载 `contentPath` 属性的语句中，将斜体文本替换为 **FLV** 文件在 **FCS** 上的名称和位置。

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.playing = function(event:Object:Object) {
 trace("The isLive property is " + my_FLVPlayback.isLive);
};
my_FLVPlayback.addEventListener("playing", listenerObject);
my_FLVPlayback.contentPath = "rtmp://my_servername/my_application/stream.flv";
```

另请参见

[FLVPlayback.contentPath](#)、[FLVPlayback.load\(\)](#)

# FLVPlayback.isRTMP

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.isRTMP*

## 说明

属性；一个布尔值，如果 FLV 文件是使用 RTMP 从 FCS 或 FVSS 流式加载的，则为 true。对于任何其它 FLV 文件源，它的值都是 false。只读。

## 示例

以下示例假定正在从 FCS 或 FVSS 播放 FLV 流文件。在 playing 事件发生时，该示例显示 isRTMP 属性的值，以指示该 FLV 文件是否来自 RTMP URL。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上。在加载 contentPath 属性的语句中，将斜体文本替换为 FLV 文件在 FCS 上的名称和位置。

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.bufferTime = 7;
var listenerObject:Object = new Object();
// 侦听 RTMP 连接上的播放事件；显示 isRTMP 的结果
listenerObject.playing = function(eventObject:Object) {
 trace("Value of isRTMP property is: " + my_FLVPlayback.isRTMP);
};
my_FLVPlayback.addEventListener("playing", listenerObject);
my_FLVPlayback.contentPath = "rtmp://my_servername/my_application/stream.flv";
```

## 另请参见

[FLVPlayback.contentPath](#)、[FLVPlayback.load\(\)](#)、[FLVPlayback.play\(\)](#)

# FLVPlayback.load()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVplybk.load(contentPath:String[, totalTime:Number, isLive:Boolean])
```

## 参数

*contentPath* 一个字符串，它指定要进行流式处理的 FLV 文件的 URL 以及如何对其进行流式处理。该 URL 可以是本地路径、指向 FLV 文件的 HTTP URL、指向 FLV 文件流的 RTMP URL 或指向 XML 文件的 HTTP URL。

*totalTime* 一个数字，表示视频的总播放时间。可选。

*isLive* 一个布尔值，如果是实时视频流，则为 true。只有从 FVSS 或 FCS 流式加载时，此值才起作用。对于 HTTP 下载，会忽略此属性的值。可选。

## 返回

无。

## 说明

方法；开始加载 FLV 文件，并提供快捷方式，用于将 autoPlay 属性设置为 false 以及设置 contentPath、totalTime 和 isLive 属性（如果指定）。如果未定义 totalTime 和 isLive 属性，则不会设置这两个属性。如果 contentPath 属性未定义、为 null 或为空字符串，则此方法不起作用。

## 示例

以下示例调用 load() 方法来加载用 contentPath 参数指定的 FLV 文件。它在加载 FLV 文件前后显示 autoPlay 属性的值，并且调用 play() 方法以开始播放 FLV 文件。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
trace("Before load, autoPlay is: " + my_FLVplybk.autoPlay);
my_FLVplybk.load("http://www.helpexamples.com/flash/video/water.flv");
```

```
trace("After load, autoPlay is: " + my_FLVPlayback.autoPlay);
my_FLVPlayback.play();
```

### 另请参见

[FLVPlayback.contentPath](#)、[FLVPlayback.isLive](#)、[FLVPlayback.totalTime](#)

## FLVPlayback.LOADING

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.FLVPlayback.LOADING`

### 说明

一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "loading"。您可以将此属性与 `state` 属性进行比较，以确定该组件是否处于加载状态。

### 示例

以下示例在 FLV 文件处于加载状态时显示 `FLVPlayback.LOADING` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
 if(my_FLVPlayback.state == FLVPlayback.LOADING)
 trace("State is " + FLVPlayback.LOADING);
}
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)



# FLVPlayback.maintainAspectRatio

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.maintainAspectRatio

## 说明

属性；一个布尔值，如果为 true，则保持视频高宽比。如果加载 FLV 文件后，此属性从 false 更改为 true，并且 autoSize 属性为 false，则视频会立即开始自动调整大小。默认值为 true。

## 示例

以下示例调用 setSize() 方法以更改 FLVPlayback 实例的大小，并且触发 resize 事件。maintainAspectRatio 属性（默认值为 true）强制另一个 resize 事件保持高宽比。resize 事件处理函数在“输出”面板中为发生的两次大小调整事件显示调整大小后的 FLVPlayback 实例的宽度和高度。如果将 maintainAspectRatio 设置为 false，则 setSize() 方法所指定的尺寸将生效。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
// maintainAspectRatio 的默认值为 true，它导致在大小发生更改时调整大小。
// 从以下行中删除注释分隔符，以禁用大小调整。

// my_FLVPlayback.maintainAspectRatio = false;
var listenerObject:Object = new Object();
listenerObject.resize = function(event:Object:Object) {
 trace("resize event; Width is: " + eventObject.target.width + " Height is: "
 + eventObject.target.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_FLVPlayback.setSize(300, 300);
```

另请参见

[FLVPlayback.autoSize](#)、[FLVPlayback.height](#)、[FLVPlayback.preferredHeight](#)、[FLVPlayback.preferredWidth](#)、[FLVPlayback.resize](#)、[FLVPlayback.setSize\(\)](#)、[FLVPlayback.width](#)

# FLVPlayback.metadata

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.metadata*

## 说明

属性：一个对象，它是通过调用 `NetStream.onMetaData()` 回调函数（如果可用）而接收到的元数据信息包。只读。

如果 FLV 文件是用 **Flash 8** 编码器编码的，则 `metadata` 属性包含以下信息。早期的 FLV 文件只包含 `height`、`width` 和 `duration` 值。

参数	说明
<code>canSeekToEnd</code>	一个布尔值，如果 FLV 文件是用最后一帧（它允许定位到渐进式下载影片剪辑的末尾）上的关键帧编码的，则该值为 <code>true</code> 。如果 FLV 文件不是用最后一帧上的关键帧编码的，则该值为 <code>false</code> 。
<code>cuePoints</code>	嵌入在 FLV 文件中的提示点对象组成的数组，每个提示点对应一个对象。如果 FLV 文件不包含任何提示点，则值是未定义的。每个对象都具有以下属性： <ul style="list-style-type: none"><li><code>type</code> 一个字符串，它将提示点的类型指定为 "navigation" 或 "event"。</li><li><code>name</code> 一个字符串，表示提示点的名称。</li><li><code>time</code> 一个数字，表示以秒为单位的提示点的时间，精确到三位小数（毫秒）。</li><li><code>parameters</code> 一个可选对象，具有用户在创建提示点时指定的名称 - 值对。</li></ul>
<code>audiocodecid</code>	一个数字，指示已使用的音频编解码器（编码 / 解码技术）。
<code>audiodelay</code>	一个数字，指示原始 FLV 文件的 FLV 文件 "time 0" 中存在哪些时间。为了正确同步音频，视频内容需要有少量的延迟。
<code>audiodatarate</code>	一个数字，表示每秒音频的千字节数。

参数	说明
videocodecid	一个数字，表示用于对视频进行编码的编解码器版本。
framerate	一个数字，表示 FLV 文件的帧频。
videodatarate	一个数字，表示 FLV 文件的视频数据速率。
height	一个数字，表示 FLV 文件的高度。
width	一个数字，表示 FLV 文件的宽度。
duration	一个数字，以秒为单位指定 FLV 文件的持续时间。

## 示例

以下示例在“输出”面板中显示来自 FLV 文件 `cuepoints.flv` 的 metadata 抽样值。它在 metadataReceived 事件发生时显示该数据。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
 trace("canSeekToEnd is " + my_FLVPlayback.metadata.canSeekToEnd);
 trace("Number of cue points is " +
 my_FLVPlayback.metadata.cuePoints.length);
 trace("Frame rate is " + my_FLVPlayback.metadata.framerate);
 trace("Height is " + my_FLVPlayback.metadata.height);
 trace("Width is " + my_FLVPlayback.metadata.width);
 trace("Duration is " + my_FLVPlayback.metadata.duration + " seconds");
};
my_FLVPlayback.addEventListener("metadataReceived", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.metadataLoaded](#)、[FLVPlayback.metadataReceived](#)

# FLVPlayback.metadataLoaded

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk.metadataLoaded*

## 说明

属性：一个布尔值，如果遇到并处理了元数据包，或者如果 FLV 文件在没有元数据包的情况下被编码，则该值为 true。换言之，如果接收了元数据，或者永远不打算获取任何元数据，则该值为 true。因此，您知道是否具有元数据；并且，如果不具有元数据，则知道不必为其等待。如果您只想知道是否具有元数据，则可以用以下代码检查该值：

```
FLVPlayback.metadata != null
```

使用此属性可以通过用于查找以及启用或禁用提示点的方法，检查是否可以检索有用信息。  
只读。

## 示例

以下示例创建 progress 事件的侦听器。在该事件发生时，该示例检查 metadataLoaded 属性是否为 true；如果为 true，则在“输出”面板中显示元数据值 height、width 和 duration。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.progress = function(event:Object):Void {
 if(my_FLVPlybk.metadataLoaded){
 trace("Height is " + my_FLVPlybk.metadata.height);
 trace("Width is " + my_FLVPlybk.metadata.width);
 trace("Duration is " + my_FLVPlybk.metadata.duration + " seconds");
 }
};
my_FLVPlybk.addEventListener("progress", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.metadata](#)、[FLVPlayback.metadataReceived](#)

# FLVPlayback.metadataReceived

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("metadataReceived", listenerObject);
```

## 说明

事件：第一次到达 FLV 文件元数据时调度。事件对象具有一个 info 属性，而该属性包含 NetStream.onMetaData 回调所接收的 info 对象。

该事件还具有 vp 属性，它是此事件适用的视频播放器的索引号。有关更多信息，请参见 [第 505 页的 FLVPlayback.activeVideoPlayerIndex](#) 和 [第 632 页的 FLVPlayback.visibleVideoPlayerIndex](#)。

## 示例

以下示例创建 metadataReceived 事件的侦听器。在该事件发生时，事件处理函数将 metadata 属性中描述的每个提示点的名称、时间和类型发送到“输出”面板。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.metadataReceived = function(eventObject:Object):Void {
 var i:Number = 0;
 trace("This FLV contains the following cue points:");
 while(i < my_FLVplybk.metadata.cuePoints.length) {
 trace("\nName: " + my_FLVplybk.metadata.cuePoints[i].name);
 trace(" Time: " + my_FLVplybk.metadata.cuePoints[i].time);
 trace(" Type is " + my_FLVplybk.metadata.cuePoints[i].type);
 ++i;
 }
};
```

```
my_FLVPlybk.addEventListener("metadataReceived", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

### 另请参见

[FLVPlayback.metadata](#)、[FLVPlayback.metadataLoaded](#)

## FLVPlayback.muteButton

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlybk.muteButton
```

### 说明

属性：作为 **Mute** 按钮控件的 **MovieClip** 对象。有关将 **FLV** 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 **FLV** 回放自定义用户界面组件的外观”。

单击 **muteButton** 控件时将调度 **volumeUpdate** 事件。

### 示例

以下示例使用 **backButton**、**forwardButton**、**playPauseButton**、**stopButton** 和 **muteButton** 属性将各个 **FLV** 自定义用户界面控件附加到 **FLVPlayback** 组件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**，然后在“组件”检查器中将 **skin** 参数设置为“无”。接下来，添加以下各个 **FLV** 自定义用户界面组件，并为其指定括号中显示的实例名称：**BackButton** (**my\_bkbtn**)、**ForwardButton** (**my\_fwdbtn**)、**PlayPauseButton** (**my\_plypausbtn**)、**StopButton** (**my\_stopbtn**) 和 **MuteButton** (**my\_mutebtn**)。然后将以下几行代码添加到时间轴第 1 帧的“动作”面板中：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 - 库中有 FLV 自定义用户界面 BackButton、ForwardButton、PlayPauseButton、
 StopButton 和 MuteButton 组件
*/
import mx.video.*;
my_FLVPlybk.backButton = my_bkbtn;
my_FLVPlybk.forwardButton = my_fwdbtn;
```

```
my_FLVPlaybk.playPauseButton = my_plypausbbtn;
my_FLVPlaybk.stopButton = my_stopbbtn;
my_FLVPlaybk.muteButton = my_mutebbtn;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.skin](#)、[FLVPlayback.volume](#)、[FLVPlayback.volumeBar](#)、[FLVPlayback.Volumeupdate](#)

## FLVPlayback.NAVIGATION

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.FLVPlayback.NAVIGATION`

### 说明

一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "navigation"。可以将此属性用作 `findCuePoint()` 和 `findNearestCuePoint()` 方法的 `type` 参数。

### 示例

以下示例使用 `FLVPlayback.NAVIGATION` 属性指定要查找的提示点的类型。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlaybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlaybk
 */
import mx.video.*;
// 使用 time 属性查找导航提示点
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 var rtn_cuePt:Object = new Object();
 rtn_cuePt = my_FLVPlaybk.findCuePoint(7.748, FLVPlayback.NAVIGATION);
 trace("Found cue point at " + rtn_cuePt.time + " of type " +
 rtn_cuePt.type);
}
```

```
my_FLVPlybk.addEventListener("ready", listenerObject)
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

### 另请参见

[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.Findnearestcuepoint\(\)](#)

## FLVPlayback.ncMgr

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlybk.ncMgr*

### 说明

属性；一个 **INCManger** 对象，它提供对实现 **INCManger** 的类的实例的访问，而该类是 **NCManager** 类的接口。

您可以使用此属性来实现需要自定义初始化的自定义 **INCManger**。只读。

### 示例

以下示例在发生 **ready** 事件时显示 **NetConnection** **DEFAULT\_TIMEOUT** 属性值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
// 指定 FLV 的名称和位置
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 var NC:Object = new Object();
 NC = my_FLVPlybk.ncMgr;
 trace("Net connection timeout is " + NC.DEFAULT_TIMEOUT + "
 milliseconds");
};
my_FLVPlybk.addEventListener("ready", listenerObject);
```



另请参见

[VideoPlayer 类](#)

## FLVPlayback.pause()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVplybk.pause()
```

### 参数

无。

### 返回

无。

### 说明

方法：暂停视频流的播放。

### 示例

以下示例将为 playheadUpdate 事件创建一个侦听器。在该事件发生时，事件处理函数查看播放头时间是否介于 5 秒和 5.05 秒之间。如果介于这个范围内，则事件处理函数调用 pause() 方法来暂停 FLV 文件的播放。paused 事件处理函数提示您按下“播放”按钮以继续。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。将一个 TextArea 组件拖到舞台中的 FLVPlayback 实例下，然后为其指定实例名称 **my\_ta**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
my_ta.visible = false;
my_FLVplybk.playheadUpdateInterval = 5;
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
```

```

 if ((eventObject.playheadTime >= 5) && (eventObject.playheadTime < 5.05))
 {
 my_FLVPlybk.pause();
 }
 };
my_FLVPlybk.addEventListener("playheadUpdate", listenerObject);
listenerObject.paused = function(eventObject:Object):Void {
 my_ta.text = "Paused; push Play to continue";
 my_ta.visible = true;
};
my_FLVPlybk.addEventListener("paused", listenerObject);

```

### 另请参见

[FLVPlayback.paused](#)、[FLVPlayback.play\(\)](#)、[FLVPlayback.rewind](#)

## FLVPlayback.pauseButton

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlybk.pauseButton*

### 说明

属性；作为 **PauseButton** 控件的 **MovieClip**。有关将 **FLV** 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 **FLV** 回放自定义用户界面组件的外观”。

### 示例

以下示例使用 **backButton**、**forwardButton**、**playButton**、**pauseButton** 和 **stopButton** 属性，来将各个 **FLV** 自定义用户界面控件附加到 **FLVPlayback** 组件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**，然后在“组件”检查器中将 **skin** 参数设置为“无”。接下来，添加以下各个 **FLV** 自定义用户界面组件，并为其指定括号中显示的实例名称 **BackButton** (**my\_bkbtn**)、**ForwardButton** (**my\_fwdbtn**)、**PlayButton** (**my\_plybtn**)、**PauseButton** (**my\_pausbtn**) 和 **StopButton** (**my\_stopbtn**)。然后将以下几行代码添加到时间轴第 1 帧的“动作”面板中：

```
/**
```

要求：

- 舞台上的 **FLVPlayback** 组件具有实例名称 **my\_FLVPlybk**
- 库中有 **FLV** 自定义用户界面的 **BackButton**、**ForwardButton**、**PlayButton**、**PauseButton** 和

```

 StopButton 组件
*/
import mx.video.*;
my_FLVPlaybk.backButton = my_bkbtn;
my_FLVPlaybk.forwardButton = my_fwdbtn;
my_FLVPlaybk.playButton = my_plybtn;
my_FLVPlaybk.pauseButton = my_pausbtn;
my_FLVPlaybk.stopButton = my_stopbtn;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";

```

### 另请参见

[FLVPlayback.playButton](#)、[FLVPlayback.playPauseButton](#)、[FLVPlayback.skin](#)

## FLVPlayback.PAUSED

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
mx.video.FLVPlayback.PAUSED
```

### 说明

一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "paused"。您可以将此属性与 `state` 属性进行比较，以查看该组件是否处于暂停状态。

### 示例

以下示例使用 **FLVPlayback.PAUSED** 属性在用户单击“暂停”按钮时显示 FLV 文件的状态。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlaybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```

/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlaybk
*/
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(eventObject.state == FLVPlayback.PAUSED)
 trace("FLV is " + FLVPlayback.PAUSED);
}

```

```
my_FLVPlayback.addEventListener("stateChange", listenerObject)
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

## FLVPlayback.paused

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.paused = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVPlayback.addEventListener("paused", listenerObject);
```

### 说明

事件：播放器进入暂停状态时调度。调用 `pause()` 方法或者单击相应控件时会发生此事件；如果 `autoPlay` 为 `false`，则加载 **FLV** 文件时，有时也会发生此事件（可能进入 `stopped` 状态）。事件对象具有 `state`、`playheadTime` 和 `vp` 属性，它是此事件适用的视频播放器的索引号。有关 `vp` 属性的更多信息，请参见[第 505 页的 `FLVPlayback.activeVideoPlayerIndex`](#) 和[第 632 页的 `FLVPlayback.visibleVideoPlayerIndex`](#)。

还会调度 `stateChange` 事件。

### 示例

以下示例将为 `playheadUpdate` 事件创建一个侦听器。该事件发生时，事件处理函数检查 `playheadTime` 属性是否介于 5 秒和 5.05 秒之间。如果介于这个范围内，则事件处理函数调用 `pause()` 方法来暂停 **FLV** 文件的播放。这会触发 `paused` 事件，对于此事件，`paused` 事件处理函数会显示消息 “The FLV is paused!”（FLV 已暂停！）

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.playheadUpdateInterval = 5;
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
 if ((eventObject.playheadTime >= 5) && (eventObject.playheadTime < 5.05))
 {
 my_FLVPlybk.pause();
 }
}
my_FLVPlybk.addEventListener("playheadUpdate", listenerObject);
listenerObject.paused = function(eventObject:Object) {
 trace("FLV is " + my_FLVPlybk.state + "!");
};
my_FLVPlybk.addEventListener("paused", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

### 另请参见

[FLVPlayback.pause\(\)](#)、[FLVPlayback.paused](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

## FLVPlayback.paused

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlybk.paused*

### 说明

属性；一个布尔值，如果 FLV 文件处于暂停状态，则为 `true`。只读。

## 示例

以下示例创建 `stateChange` 事件的侦听器。在该事件发生时，它检查 `paused` 属性以确定组件是否处于暂停状态。如果处于暂停状态，则它在“输出”面板中对该结果显示一条消息。您必须在 **FLV** 文件正在播放时单击“暂停”按钮，以使暂停状态出现。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object) {
 if(my_FLVPlybk.paused)
 trace("FLV is in " + FLVPlayback.PAUSED + " state");
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.paused](#)、[FLVPlayback.PAUSED](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.play()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlybk.play ([contentPath:String, totalTime:Number, isLive:Boolean])
```

## 参数

*contentPath* 一个字符串，它指定要进行流式处理的 FLV 文件的 URL 以及如何对其进行流式处理。该 URL 可以是本地路径、指向 FLV 文件的 HTTP URL、指向 FLV 文件流的 RTMP URL 或指向 XML 文件的 HTTP URL。它是可选的，但必须通过“组件”检查器或 **ActionScript** 设置 *contentPath* 属性，否则此方法将不起作用。

*totalTime* 一个数字，表示视频的总播放时间。可选。

*isLive* 一个布尔值，如果是实时视频流，则为 true。只有在从 FCS 或 FVSS 进行流式加载时，此值才有效。对于 HTTP 下载，会忽略此属性的值。可选。

## 返回

无。

## 说明

方法；播放视频流。不带参数时，此方法只将 FLV 文件从暂停或停止状态转换为播放状态。

如果使用参数，则此方法起到快捷键的作用，可将 *autoPlay* 属性设置为 true，并设置 *isLive*、*totalTime* 和 *contentPath* 属性。如果未定义 *totalTime* 和 *isLive* 属性，则不会设置这两个属性。

## 示例

以下示例禁止自动播放 FLV 文件，调用 *seekSeconds()* 方法以将播放头设置为视频中的 20 秒处，并调用 *play()* 方法从该点开始播放 FLV 文件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoPlay = false;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVPlybk.seekSeconds(4);
 my_FLVPlybk.play();
};
my_FLVPlybk.addEventListener("ready", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.autoPlay](#)、[FLVPlayback.contentPath](#)、[FLVPlayback.load\(\)](#)、[FLVPlayback.pause\(\)](#)、[FLVPlayback.stop\(\)](#)

# FLVPlayback.playButton

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlybk.playButton`

## 说明

属性；作为“播放”按钮的 `MovieClip` 对象。有关将 FLV 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 FLV 回放自定义用户界面组件的外观”。

## 示例

以下示例使用 `backButton`、`forwardButton`、`playButton`、`pauseButton` 和 `stopButton` 属性，来将各个 FLV 自定义用户界面控件附加到 `FLVPlayback` 组件。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。接下来，添加以下各个 FLV 自定义用户界面组件，并为其指定括号中显示的实例名称：`BackButton` (**my\_bkbtn**)、`ForwardButton` (**my\_fwdbtn**)、`PlayButton` (**my\_plybtn**)、`PauseButton` (**my\_pausbtn**) 和 `StopButton` (**my\_stopbtn**)。然后将以下几行代码添加到“动作”面板中：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 - 库中有 FLV 自定义用户界面的 BackButton、ForwardButton、PlayButton、PauseButton
 和
 StopButton 组件
*/
import mx.video.*;
my_FLVPlybk.backButton = my_bkbtn;
my_FLVPlybk.forwardButton = my_fwdbtn;
my_FLVPlybk.playButton = my_plybtn;
my_FLVPlybk.pauseButton = my_pausbtn;
my_FLVPlybk.stopButton = my_stopbtn;
```

## 另请参见

[FLVPlayback.playing](#)、[FLVPlayback.skin](#)



# FLVPlayback.playheadPercentage

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlybk.playheadPercentage`

## 说明

属性：一个数字，它将当前的 `playheadTime` 指定为 `totalTime` 属性的百分比。如果访问此属性，则它包含已运行的播放时间的百分比。如果设置此属性，则它会导致搜索操作定向到表示 FLV 文件播放时间百分比的点。

该属性的值相对于 `totalTime` 属性的值。

如果您指定无效的百分比，或者 `totalTime` 属性未定义、为 `null` 或者小于或等于 0，则该组件将引发 **VideoError**。

## 示例

以下示例显示在 `point2` 提示点出现时已播放的 FLV 文件的百分比。在 `point3` 提示点处，它将 `playheadPercentage` 设置为 10，这导致搜索操作定位于距 FLV 文件开始处 10% 的点，并且创建回放循环。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 if(eventObject.info.name == "point2")
 trace("point2 occurred at " + my_FLVPlybk.playheadPercentage + "
 percent of FLV");
 if(eventObject.info.name == "point3")
 my_FLVPlybk.playheadPercentage = 10;
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

## 另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.seekPercent\(\)](#)、[FLVPlayback.totalTime](#)

# FLVPlayback.playheadTime

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlayback.playheadTime`

## 说明

属性；一个数字，表示当前播放头的时间或位置（以秒为单位计算），可以是小数。设置此属性会触发搜索，并具有搜索的所有限制。

播放头时间发生变化（包括播放 FLV 文件时每隔 0.25 秒更改一次）时，该组件会调度 playheadUpdate 事件。

出于几种原因，在调用一种搜索方法或者设置 playheadTime 以引发搜索后，playheadTime 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 FLV 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 playheadTime 属性，playheadTime 是不会立即更新的。若要在搜索完成后获取时间，请侦听 seek 事件，该事件在 playheadTime 属性更新后才会引发。

## 示例

以下示例会在播放 FLV 文件的过程中捕获发生的 stateChange 事件，并在“输出”面板中显示播放头已运行的时间。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 trace(my_FLVPlayback.state + ": playhead time is: " +
 my_FLVPlayback.playheadTime);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

另请参见

[FLVPlayback.playheadUpdate](#)、[FLVPlayback.playheadUpdateInterval](#)、[FLVPlayback.seek\(\)](#)、[FLVPlayback.stateChange](#)

## FLVPlayback.playheadUpdate

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

```
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("playheadUpdate", listenerObject);
```

说明

事件：在 **FLV** 文件正按 `playheadUpdateInterval` 属性所指定的频率播放时调度。默认值为 **0.25** 秒。视频播放器处于暂停或停止状态时，该组件不调度此事件（除非发生搜索）。事件对象具有 `state`、`playheadTime` 和 `vp` 属性。

示例

以下示例会在播放 **FLV** 文件的过程中捕获发生的 `playheadUpdate` 事件，并在“输出”面板中显示播放头已运行的时间。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
 trace(my_FLVplybk.state + ": playhead time is: " +
 eventObject.playheadTime);
};
my_FLVplybk.addEventListener("playheadUpdate", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.playheadUpdateInterval](#)

# FLVPlayback.playheadUpdateInterval

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.playheadUpdateInterval
```

## 说明

属性；一个数字，表示每个 playheadUpdate 事件之间的时间，以毫秒为单位。播放 FLV 文件时设置此属性会重新启动计时器。默认值为 250。

因为 **ActionScript** 提示点在播放头更新时开始，所以，降低 playheadUpdateInterval 属性的值可增加 **ActionScript** 提示点的精确度。

因为播放头更新间隔是通过调用 setInterval() 全局函数设置的，所以，更新操作无法比 SWF 文件帧频更频繁地触发，这与此方法设置的任何间隔一样。因此，举例来说，对于每秒 12 帧的默认帧频而言，您可以创建的最低有效间隔大约为 83 毫秒，或者 1 秒（1000 毫秒）的 12 分之 1。

## 示例

以下示例将 playheadUpdateInterval 属性设置为 3000，并创建一个侦听器，用于在播放 FLV 文件时捕获发生的 playheadUpdate 事件。该事件发生时，事件处理函数会在“输出”面板中显示播放头已运行的时间。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.playheadUpdateInterval = 3000;
var listenerObject:Object = new Object();
listenerObject.playheadUpdate = function(eventObject:Object):Void {
 trace("playhead time is: " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("playheadUpdate", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.playheadUpdate](#)

# FLVPlayback.PLAYING

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.FLVPlayback.PLAYING`

## 说明

一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "playing"。您可以将此属性与 `state` 属性进行比较，以确定该组件是否处于播放状态。

## 示例

以下示例使用 `FLVPlayback.PLAYING` 属性确定在 `stateChange` 事件发生时状态是否等于 "playing"。它还将该常数作为“输出”面板中显示的消息的一部分而包括在内。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(eventObject.state == FLVPlayback.PLAYING)
 trace(my_FLVPlybk.contentPath + " is now " + FLVPlayback.PLAYING);
}
my_FLVPlybk.addEventListener("stateChange", listenerObject);
```

## 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.playing

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var :Object = new Object();
listenerObject.playing = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("playing", listenerObject);
```

## 说明

事件：进入播放状态时调度。在调用 `play()` 方法或者单击相应控件后可能并不会立即发生此事件，通常先进入缓冲状态，然后再进入播放状态。事件对象具有 `state`、`playheadTime` 和 `vp` 属性，它是此事件适用的视频播放器的索引号。有关 `vp` 属性的更多信息，请参见第 505 页的 [FLVPlayback.activeVideoPlayerIndex](#) 和第 632 页的 [FLVPlayback.visibleVideoPlayerIndex](#)。

**FLVPlayback** 实例还会调度 `stateChange` 事件。

## 示例

以下示例在 `playing` 事件发生时在文本区域中显示 `contentPath` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPLYbk**。将一个 **TextArea** 组件拖到舞台中的 **FLVPlayback** 实例下，然后为其指定实例名称 **my\_ta**。然后将以下代码添加到“动作”面板中时间轴的第 1 帧内：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPLYbk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.playing = function(eventObject:Object):Void {
 my_ta.text = "Now playing: " + my_FLVPLYbk.contentPath;
}
my_FLVPLYbk.addEventListener("playing", listenerObject);
my_FLVPLYbk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.play\(\)](#)、[FLVPlayback.playing](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.playing

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.playing

## 说明

属性；一个布尔值，如果 FLV 文件处于播放状态，则为 true。只读。

## 示例

以下示例会侦听播放 FLV 文件过程中发生的 stateChange 事件。事件发生时，此示例会在“输出”面板中显示 playing 属性的值。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
trace(my_FLVPlayback.state + ": playing property is " + my_FLVPlayback.playing);
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 trace(my_FLVPlayback.state + ": playing property is " + my_FLVPlayback.playing);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.playing](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.playPauseButton

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.playPauseButton

## 说明

属性；作为 PlayPauseButton 的 MovieClip 对象。有关将 FLV 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 FLV 回放自定义用户界面组件的外观”。

## 示例

以下示例使用 playPauseButton、stopButton、backButton 和 forwardButton 属性将各个 FLV 自定义用户界面控件附加到 FLVPlayback 组件。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。接下来，添加以下各个 FLV 自定义用户界面组件，并为其指定括号中显示的实例名称：BackButton (**my\_bkbtn**)、ForwardButton (**my\_fwdbtn**)、PlayPauseButton (**my\_plypausebtn**) 和 StopButton (**my\_stopbtn**)。然后将以下几行代码添加到“动作”面板中：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 - 库中有 FLV 自定义用户界面 PlayPauseButton、StopButton、BackButton 和
 ForwardButton 组件
*/
import mx.video.*;
my_FLVPlayback.playPauseButton = my_plypausebtn;
my_FLVPlayback.stopButton = my_stopbtn;
my_FLVPlayback.backButton = my_bkbtn;
my_FLVPlayback.forwardButton = my_fwdbtn;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.playButton](#)、[FLVPlayback.playPauseButton](#)、[FLVPlayback.paused](#)、[FLVPlayback.skin](#)



# FLVPlayback.preferredHeight

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.preferredHeight

## 说明

属性；一个数字，指定源 FLV 文件的高度。调用 `play()` 或 `load()` 方法后，此信息不会立即生效。它在 `ready` 事件开始时生效。如果 `autoSize` 属性或 `maintainAspectRatio` 属性的值为 `true`，则开始 `resize` 事件时，最好读取该值。只读。

## 示例

以下示例将在发生 `ready` 事件时设置 **FLVPlayback** 实例的大小。当发生 `cuePoint` 事件时，该示例将其大小重置为 `preferredHeight` 和 `preferredWidth` 属性所指定的大小。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
 trace("width is: " + my_FLVPlayback.width);
 trace("height is: " + my_FLVPlayback.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVPlayback.setSize(250, 350);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
listenerObject.cuePoint = function(eventObject:Object):Void {
 my_FLVPlayback.setSize(my_FLVPlayback.preferredWidth,
 my_FLVPlayback.preferredHeight);
};
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_FLVPlayback.addASCuePoint(1.5, "AScp1");
```

另请参见

[FLVPlayback.autoSize](#)、[FLVPlayback.height](#)、[FLVPlayback.maintainAspectRatio](#)、[FLVPlayback.preferredWidth](#)、[FLVPlayback.ready](#)、[FLVPlayback.setSize\(\)](#)、[FLVPlayback.setScale\(\)](#)、[FLVPlayback.width](#)

## FLVPlayback.preferredWidth

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlybk.preferredWidth*

### 说明

属性；指定源 **FLV** 文件的宽度。在调用 `play()` 或 `load()` 方法时此信息不会立即生效；它在 `ready` 事件启动时生效。如果 `autoSize` 或 `maintainAspectRatio` 属性的值为 `true`，则 `resize` 事件启动时，最好读取该值。只读。

### 示例

以下示例将在发生 `ready` 事件时设置 **FLVPlayback** 实例的大小。当发生 `cuePoint` 事件时，该示例将其大小重置为 `preferredHeight` 和 `preferredWidth` 属性所指定的大小。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
 trace("width is: " + my_FLVPlybk.width);
 trace("height is: " + my_FLVPlybk.height);
};
my_FLVPlybk.addEventListener("resize", listenerObject);
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVPlybk.setSize(250, 350);
};
my_FLVPlybk.addEventListener("ready", listenerObject);
listenerObject.cuePoint = function(eventObject:Object):Void {
```

```
my_FLVPlybk.setSize(my_FLVPlybk.preferredWidth,
my_FLVPlybk.preferredHeight);
};
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
my_FLVPlybk.addASCuePoint(1.5, "AScp1");
```

### 另请参见

[FLVPlayback.autoSize](#)、[FLVPlayback.height](#)、[FLVPlayback.maintainAspectRatio](#)、[FLVPlayback.preferredHeight](#)、[FLVPlayback.ready](#)、[FLVPlayback.setSize\(\)](#)、[FLVPlayback.setScale\(\)](#)、[FLVPlayback.width](#)

## FLVPlayback.progress

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("progress", listenerObject);
```

### 说明

事件：按 `progressInterval` 属性所指定的频率调度，从加载开始时开始，到所有字节加载结束或者出现网络错误时结束。默认值为每 0.25 秒。

仅针对渐进式 HTTP 下载进行调度。用已加载的字节数表示进度。该事件对象具有 `bytesLoaded` 和 `bytesTotal` 属性，它们与具有相同名称的 `FLVPlayback` 属性相同。

该事件还具有 `vp` 属性，它是此事件适用的视频播放器的索引号。有关 `vp` 属性的更多信息，请参见 `FLVPlayback.activeVideoPlayerIndex` 和

`FLVPlayback.visibleVideoPlayerIndex`。

## 示例

以下示例将 `progressInterval` 属性设置为 **001** 毫秒（因为 **FLV** 文件较短），然后显示针对发生的每个 `progress` 事件而加载的字节数。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.progressInterval = 001;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
 trace(eventObject.bytesLoaded);
}
my_FLVPlayback.addEventListener("progress", listenerObject);
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.bytesLoaded](#)、[FLVPlayback.bytesTotal](#)、[FLVPlayback.progressInterval](#)、[FLVPlayback.visibleVideoPlayerIndex](#)

# FLVPlayback.progressInterval

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.progressInterval

## 说明

属性：一个数字，表示每个 `progress` 事件之间的时间，以毫秒为单位。如果您在播放视频流时设置此属性，则会重新启动计时器。默认值为 **250**。

## 示例

以下示例将 `progressInterval` 属性设置为 001 毫秒（因为 FLV 文件较小），然后显示针对发生的每个 `progress` 事件而加载的字节数。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.progressInterval = 001;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object):Void {
 trace(eventObject.bytesLoaded);
}
my_FLVPlayback.addEventListener("progress", listenerObject);
```

## 另请参见

[FLVPlayback.progress](#)

# FLVPlayback.ready

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVPlayback.addEventListener("ready", listenerObject);
```

## 说明

事件：加载 FLV 文件并可以显示时调度。在您使用 `play()` 或 `load()` 方法加载新的 FLV 文件后首次进入响应状态时，该事件开始。它只为加载的每个 FLV 文件开始一次。

事件对象具有 `state`、`playheadTime` 和 `vp` 属性。`vp` 属性是此事件适用的视频播放器的索引号。有关 `vp` 属性的更多信息，请参见第 505 页的 `FLVPlayback.activeVideoPlayerIndex` 和第 632 页的 `FLVPlayback.visibleVideoPlayerIndex`。

## 示例

```
/**
 * 要求:
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 * - 舞台上的 TextArea 组件具有实例名称 my_ta
 */
import mx.video.*;
my_ta.visible = false;
my_FLVPlayback.autoPlay = false;
my_ta.setSize(260, 30);
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 my_ta.text = "The FLV is ready. Push Play to start playing";
 my_ta.visible = true;
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.state](#)、[FLVPlayback.visibleVideoPlayerIndex](#)

# FLVPlayback.removeASCuePoint()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.removeASCuePoint(CuePoint:Object):Object
my_FLVPlayback.removeASCuePoint(time:Number):Object
my_FLVPlayback.removeASCuePoint(name:String):Object
```

## 参数

*CuePoint* 一个提示点对象，其中包含要删除的提示点的 *time* 和 *name* 属性。该方法不检查传入提示点对象上的任何其它属性。如果 *time* 或 *name* 为 null 或者未定义，则该方法只使用可用的属性。如果仅指定 *name*，则该方法删除具有此名称的第一个提示点。

*time* 一个数字，包含要删除的提示点的时间。该方法删除具有此时间的第一个提示点。

*name* 一个字符串，包含要删除的提示点的名称。该方法删除具有此名称的第一个提示点。

## 返回

已删除的提示点对象。如果没有匹配的提示点，则该方法返回 `null`。

## 说明

方法：从当前已加载的 **FLV** 文件中删除 **ActionScript** 提示点。只使用 *CuePoint* 参数中的 `name` 和 `time` 属性来查找要删除的提示点。

如果有多个 **ActionScript** 提示点与搜索条件匹配，则只删除其中一个。要删除所有提示点，可在循环中用相同的参数反复调用此函数，直到返回 `null` 为止。

当设置了 `contentPath` 属性时，会清除提示点信息，因此，若要为下一个要加载的 **FLV** 文件设置提示点信息，应先设置 `contentPath` 属性。

## 示例

以下示例会向 **FLV** 文件中添加 **ActionScript** 提示点，然后调用 `removeASCuePoint()` 方法来删除此提示点。它会在“输出”面板中显示已删除的提示点的名称。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
// 创建提示点对象
var cuePt:Object = new Object(); // 创建提示点对象
var rtn_cuePt:Object = new Object(); // 创建对象以便返回值
cuePt.time = 4.444;
cuePt.name = "ripples";
my_FLVPlybk.addASCuePoint(cuePt); // 添加 AS 提示点
if ((rtn_cuePt = my_FLVPlybk.removeASCuePoint(cuePt)) != null) {
 trace("Removed cue point: " + rtn_cuePt.name);
}
```

## 另请参见

[FLVPlayback.addASCuePoint\(\)](#)、[FLVPlayback.findCuePoint\(\)](#)、  
[FLVPlayback.Findnearestcuepoint\(\)](#)、[FLVPlayback.findNextCuePointWithName\(\)](#)

# FLVPlayback.removeEventListener()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.removeEventListener(event:String, listener:Object):Void
my_FLVPlayback.removeEventListener(event:String, listener:Function):Void
```

## 参数

*event* 一个字符串，指定要删除其侦听器的事件的名称。

*listener* 对要删除的侦听器对象或函数的引用。

## 返回

无。

## 说明

方法：从组件实例删除事件侦听器。

## 示例

以下示例在出现第一个提示点时删除 `cuePoint` 事件的侦听器。这会导致只检测三个提示点中的第一个提示点。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
用法 1: listener object
/**
 要求:
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
*/
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var listenerObject:Object = new Object(); // 创建侦听器对象
listenerObject.cuePoint = function(eventObject:Object):Void {
 trace("Hit cue point at " + eventObject.info.time);
 my_FLVPlayback.removeEventListener("cuePoint", listenerObject);
};
my_FLVPlayback.addEventListener("cuePoint", listenerObject);
```



用法 2: listener function

```
/**
 * 要求:
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_ta.visible = false;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
function cuePoint(eventObject:Object):Void {
 trace("Hit cue point at " + eventObject.info.time);
 my_FLVPlayback.removeEventListener("cuePoint", cuePoint);
};
my_FLVPlayback.addEventListener("cuePoint", cuePoint);
```

另请参见

[FLVPlayback.addEventListener\(\)](#)

## FLVPlayback.resize

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

```
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVPlayback.addEventListener("resize", listenerObject);
```

说明

事件：调整视频大小时调度。在设置 `visibleVideoPlayerIndex` 属性并且切换到具有不同尺寸的视频播放器时，会发生此事件。该事件对象具有 `auto`、`x`、`y`、`width`、`height` 和 `vp` 属性，后者是此事件适用的视频播放器的索引号。有关 `vp` 属性的更多信息，请参见第 505 页的 [FLVPlayback.activeVideoPlayerIndex](#) 和第 632 页的 [FLVPlayback.visibleVideoPlayerIndex](#)。

在自动调整大小时 `auto` 属性为 `true`，因为 `autoSize` 或 `maintainAspectRatio` 属性是 `true`。在此情况下，可能为可见的视频播放器之外的另一个视频播放器调度事件。即使在尝试自动调整组件大小后尺寸并不实际更改，也会发生该事件。

在 `auto` 属性为 `false` 时，该事件始终适用于可见的视频播放器。`vp` 属性仍出现，但将始终等于 `visibleVideoPlayerIndex` 属性。

如果您要切换到其尺寸与当前可见播放器不同的某个视频播放器，则在设置 `visibleVideoPlayerIndex` 属性时，该组件调度该事件（`auto` 设置为 `false`）。

## 示例

以下示例播放两个 FLV 文件。该示例将一个 **ActionScript** 提示点添加到第一个 FLV 文件；并且在 `cuePoint` 事件发生时，切换到第二个更小的视频播放器，以播放第二个 FLV 文件。当该示例为视频播放器设置 `visibleVideoPlayerIndex` 属性时，会触发 `resize` 事件，该事件将显示当前视频播放器的大小和位置。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
// 禁用 autoSize 和 maintainAspectRatio
my_FLVPlybk.autoSize = false;
my_FLVPlybk.maintainAspectRatio = false;
// 播放此 FLV
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv";
// 添加一个提示点
my_FLVPlybk.addASCuePoint(3, "switch_here");
var listenerObject:Object = new Object();// 创建侦听器
listenerObject.cuePoint = function(eventObject:Object):Void {
 // 添加另一个视频播放器
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 // 播放此 FLV
 my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
 // 更改此视频播放器的大小
 my_FLVPlybk.setSize(240, 180);
 my_FLVPlybk.visibleVideoPlayerIndex = 1; // 使它可见
 my_FLVPlybk.play(); // 播放 FLV
};
// 添加 cuePoint 事件的侦听器
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
listenerObject.resize = function(eventObject:Object):Void {
 // 显示位置和尺寸
 trace("Video player is #" + my_FLVPlybk.activeVideoPlayerIndex);
 trace("X coordinate is: " + eventObject.x);
 trace("Y coordinate is: " + eventObject.y);
 trace("Width is: " + eventObject.width);
 trace("Height is: " + eventObject.height);
};
```

```
};
// 添加 resize 事件的侦听器
my_FLVPlayback.addEventListener("resize", listenerObject);
```

### 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.autoSize](#)、  
[FLVPlayback.height](#)、[FLVPlayback.maintainAspectRatio](#)、  
[FLVPlayback.preferredHeight](#)、[FLVPlayback.preferredWidth](#)、  
[FLVPlayback.setSize\(\)](#)、[FLVPlayback.state](#)、[FLVPlayback.width](#)、[FLVPlayback.x](#)、  
[FLVPlayback.y](#)

## FLVPlayback.rewind

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.rewind = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVPlayback.addEventListener("rewind", listenerObject);
```

### 说明

事件：通过调用 `seek()` 向后移动播放头的位置时或者当自动后退完成时调度。

事件对象具有 `auto`、`state` 和 `playheadTime` 属性。如果该事件导致向后搜索，则 `auto` 属性为 `false`。如果它导致自动后退，则 `auto` 属性为 `true`。

`playheadTime` 属性是目标时间。

在自动后退发生时使用 "rewinding" 状态调度 `stateChange` 事件。`stateChange` 事件在完成后退前不启动。在通过搜索实现后退时会发生 `seek` 事件。在后退发生时 **FLVPlayback** 实例还调度 `playheadUpdate` 事件。

`rewind` 事件具有 `vp` 属性，它是此事件适用的视频播放器的索引号。有关 `vp` 属性的更多信息，请参见 [FLVPlayback.activeVideoPlayerIndex](#) 和 [FLVPlayback.visibleVideoPlayerIndex](#) 属性。

## 示例

以下示例将 `autoRewind` 属性设置为 `true`，并且侦听 `rewind` 事件。在该事件发生时，事件处理函数会在“输出”面板中显示 `vp`、`state` 和 `playheadTime` 属性的值。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoRewind = true;
var listenerObject:Object = new Object();
listenerObject.rewind = function(eventObject:Object) {
 trace("Video player is #" + eventObject.vp);
 trace("State is: " + eventObject.state);
 trace("Playhead time is: " + eventObject.playheadTime);
};
my_FLVPlybk.addEventListener("rewind", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.playheadTime](#)、[FLVPlayback.playheadUpdateFLVPlayback.seek\(\)](#)、[FLVPlayback.seekPercent\(\)](#)、[FLVPlayback.seekSeconds\(\)](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.REWINDING

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.FLVPlayback.REWINDING`

## 说明

一个只读的 `FLVPlayback` 类属性，它包含字符串常数 `"rewinding"`。您可以将此属性与 `state` 属性进行比较，以确定该组件是否处于后退状态。

## 示例

以下示例为 `stateChange` 事件创建一个侦听器，并且使用 `FLVPlayback.REWINDING` 属性确定该组件是否处于后退状态。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(eventObject.state == FLVPlayback.REWINDING)
 trace("The current state is " + FLVPlayback.REWINDING);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.scaleX

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk.scaleX*

## 说明

属性；表示水平缩放的一个数字。标准缩放为 100。

## 示例

以下示例将 FLVPlayback 实例的 `scaleX`（水平）和 `scaleY`（垂直）属性设置为 150%。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.scaleX = 150;
my_FLVPlayback.scaleY = 150;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.setScale\(\)](#)、[FLVPlayback.scaleY](#)

# FLVPlayback.scaleY

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.scaleY*

## 说明

属性；表示垂直缩放的一个数字。标准缩放为 100。

## 示例

以下示例将 FLVPlayback 实例的水平 (`scaleX`) 和垂直 (`scaleY`) 缩放设置为 150%。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.scaleX = 150;
```

```
my_FLVPlaybk.scaleY = 150;
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.scaleX](#)、[FLVPlayback.setScale\(\)](#)

## FLVPlayback.scrubbing

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlaybk.scrubbing*

### 说明

属性；一个布尔值，如果使用 **SeekBar** 拖拽用户，则为 true，否则为 false。只读。

拖拽表示抓住搜索栏的手柄并沿一个方向拖动它，使其定位 **FLV** 文件中的某个特定场景。

### 示例

以下示例在发生 seek 事件时显示 scrubbing 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlaybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：



必须抓住搜索栏的手柄，并对它进行拖放操作才能引发此事件。

/\*\*

  要求：

  - 舞台上的 FLVPlayback 组件具有实例名称 my\_FLVPlaybk

\*/

```
import mx.video.*;
```

```
my_FLVPlaybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

```
var listenerObject:Object = new Object();
```

```
listenerObject.seek = function(eventObject:Object):Void {
 if(my_FLVPlaybk.scrubbing)
```

```
 trace("User is scrubbing at: " + eventObject.playheadTime);
```

```
};
```

```
my_FLVPlaybk.addEventListener("seek", listenerObject);
```

另请参见

[FLVPlayback.seek](#)、[FLVPlayback.seekBar](#)、[FLVPlayback.scrubFinish](#)、[FLVPlayback.scrubStart](#)

## FLVPlayback.scrubFinish

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

```
var listenerObject:Object = new Object();
listenerObject.scrubFinish = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("scrubFinish", listenerObject);
```

说明

事件：在用户停止使用 **SeekBar** 拖拽 FLV 文件时调度。拖拽表示抓住搜索栏的手柄并沿一个方向拖动它，使其定位 FLV 文件中的某个特定场景。在用户释放 **SeekBar** 的手柄时拖拽停止。

该事件对象具有 `state` 和 `playheadTime` 属性。该状态将是 "seeking"，直到拖拽停止为止。该组件还使用等于新状态的 `state` 属性调度 `stateChange` 事件，而新属性应是 "playing"、"paused"、"stopped" 或 "buffering"。

示例

以下示例侦听 `scrubFinish` 事件并显示拖拽停止的时间。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：



必须抓住搜索栏的手柄，并对它进行拖放操作才能引发此事件。

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
```



```
listenerObject.scrubFinish = function(eventObject:Object):Void {
 trace("Scrubbing stopped at " + eventObject.playheadTime);
 trace("Current state is " + eventObject.state);
};
my_FLVPlybk.addEventListener("scrubFinish", listenerObject);
```

### 另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.seek](#)、[FLVPlayback.seekBar](#)、[FLVPlayback.scrubStart](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

## FLVPlayback.scrubStart

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.scrubStart = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVPlybk.addEventListener("scrubStart", listenerObject);
```

### 说明

事件：在用户使用 **SeekBar** 开始拖拽 **FLV** 文件时调度。拖拽表示抓住 **SeekBar** 的手柄并沿一个方向拖动它，使其定位 **FLV** 文件中的某个特定场景。拖拽在用户单击 **SeekBar** 手柄时开始，并在用户释放手柄时结束。

该事件对象具有 `state` 和 `playheadTime` 属性。

该组件还调度 `state` 属性等于“seeking”的 `stateChange` 事件。状态在用户停止拖拽前保持“seeking”。

### 示例

以下示例侦听 `scrubStart` 事件并显示拖拽开始的时间。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：



必须抓住 **SeekBar** 的手柄，并对它进行拖动操作才能引发此事件。

```

/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.scrubStart = function(eventObject:Object):Void {
 trace("Scrubbing began at " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("scrubStart", listenerObject);

```

### 另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.scrubbing](#)、[FLVPlayback.scrubFinish](#)、[FLVPlayback.seekBar](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

## FLVPlayback.seek

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```

var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVPlayback.addEventListener("seek", listenerObject);

```

### 说明

事件：通过调用 `seek()`、设置 `playheadTime` 属性或者使用 **SeekBar** 控件更改了播放头位置时调度。`playheadTime` 属性是目标时间。该事件对象具有 `state`、`playheadTime` 和 `vp` 属性，最后一个属性是该事件所应用的视频播放器的索引编号。

在向后进行搜索时，**FLVPlayback** 实例会调度 `rewind` 事件，而在向前进行搜索时，会调度 `fastForward` 事件。它还调度 `playheadUpdate` 事件。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 **FLV** 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

## 示例

以下示例在发生 `ready` 事件时搜索到 **FLV** 文件中的 2 秒处。`seek()` 函数触发一个 `seek` 事件，在此点上侦听器显示 `playheadTime` 以及 **FLVPlayback** 实例的名称。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object) {
 trace("A seek event occurred at " + eventObject.playheadTime);
};
my_FLVPlayback.addEventListener("seek", listenerObject);
listenerObject.ready = function(eventObject:Object) {
 my_FLVPlayback.seek(2);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.fastForward](#)、  
[FLVPlayback.playheadTime](#)、[FLVPlayback.playheadUpdate](#)、[FLVPlayback.rewind](#)、  
[FLVPlayback.seek\(\)](#)、[FLVPlayback.seekPercent\(\)](#)、[FLVPlayback.seekSeconds\(\)](#)、  
[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#)、  
[FLVPlayback.seekToPrevNavCuePoint\(\)](#)

# FLVPlayback.seek()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.seek(time:Number)
```

## 参数

*time* 一个数字，指定放置播放头的时间，以秒为单位。

## 返回

无。

## 说明

方法；在文件中搜索到给定时间，以秒为单位指定，精确到三位小数（毫秒）。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 **FLV** 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

## 示例

以下示例禁止自动播放 **FLV** 文件，调用 `seek()` 方法以将播放头设置为视频中的 3 秒处，并从该点开始播放 **FLV** 文件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.autoPlay = false;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_FLVPlayback.seek(3);
my_FLVPlayback.play();
```

另请参见

[FLVPlayback.playheadTime](#)、[FLVPlayback.seek](#)、[FLVPlayback.seekPercent\(\)](#)、[FLVPlayback.seekSeconds\(\)](#)

## FLVPlayback.seekBar

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

*my\_FLVPlayback.seekBar*

说明

属性：作为播放时的搜索栏控件的 **MovieClip** 对象。有关将 FLV 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 FLV 回放自定义用户界面组件的外观”。

示例

以下示例使用 `backButton`、`forwardButton`、`playButton`、`pauseButton`、`stopButton` 和 `seekBar` 属性将各个 FLV 自定义用户界面控件附加到 **FLVPlayback** 组件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**，然后在“组件”检查器中将 `skin` 参数设置为“无”。接下来，添加以下各个 FLV 自定义用户界面组件，并为其指定括号中显示的实例名称：**BackButton** (**my\_bkbtn**)、**ForwardButton** (**my\_fwdbtn**)、**PlayPauseButton** (**my\_plypausbtn**)、**StopButton** (**my\_stopbtn**) 和 **SeekBar** (**my\_seekBar**)。然后将以下几行代码添加到时间轴第 1 帧的“动作”面板中：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 * - FLV Custom UI BackButton, ForwardButton, PlayPauseButton, StopButton and
 * SeekBar
 * components in the Library
 */
import mx.video.*;
my_FLVPlayback.backButton = my_bkbtn;
my_FLVPlayback.forwardButton = my_fwdbtn;
my_FLVPlayback.playPauseButton = my_plypausbtn;
my_FLVPlayback.stopButton = my_stopbtn;
my_FLVPlayback.seekBar = my_seekBar;
```

```
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

### 另请参见

[FLVPlayback.scrubbing](#)、[FLVPlayback.scrubFinish](#)、[FLVPlayback.scrubStart](#)、[FLVPlayback.seek](#)

## FLVPlayback.seekBarInterval

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.seekBarInterval
```

### 说明

属性；一个数字，指定进行拖拽时检查搜索栏手柄的频率，以毫秒为单位。默认值为 250。

因为此间隔是通过调用全局 `setInterval()` 函数设置的，所以，更新操作无法比 SWF 文件帧频更频繁地启动。因此，举例来说，对于每秒 12 帧的默认帧频而言，您可以创建的最低有效间隔大约为 83 毫秒，或者 1 秒（1000 毫秒）的 12 分之 1。

### 示例

以下示例将 `seekBarInterval` 设置降低到 50 毫秒，并且如果用户正在拖拽，则它显示 `playheadTime` 属性的值。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
 if(my_FLVPlayback.scrubbing) {
 my_FLVPlayback.seekBarInterval = 50;
 trace("User is scrubbing at: " + eventObject.playheadTime);
 }
}
```

```
};
my_FLVPlayback.addEventListener("seek", listenerObject);
```

### 另请参见

[FLVPlayback.seekBar](#)、[FLVPlayback.seekBarScrubTolerance](#)

## FLVPlayback.seekBarScrubTolerance

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlayback.seekBarScrubTolerance*

### 说明

属性；一个数字，指定发生更新之前用户可以移动 **SeekBar** 手柄的距离。该值以百分比指定，范围从 1 至 100。默认值为 5。

### 示例

以下示例检查在发生 seek 事件时用户是否正在进行拖拽；如果正进行拖拽，则将 `seekBarScrubTolerance` 属性的值降低为 0，以增加 **SeekBar** 位置的更新次数和 seek 事件的频率。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：



必须抓住搜索栏的手柄，并对它进行拖放操作才能引发此事件。

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.seek = function(eventObject:Object):Void {
 if(my_FLVPlayback.scrubbing) {
 my_FLVPlayback.seekBarScrubTolerance = 0;
 trace("User is scrubbing at: " + eventObject.playheadTime);
 }
}
```

```
};
my_FLVPlybk.addEventListener("seek", listenerObject);
```

### 另请参见

[FLVPlayback.scrubbing](#)、[FLVPlayback.scrubFinish](#)、[FLVPlayback.scrubStart](#)、[FLVPlayback.seekBar](#)、[FLVPlayback.seekBarInterval](#)

## FLVPlayback.SEEKING

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.FLVPlayback.SEEKING`

### 说明

一个只读的 `FLVPlayback` 类属性，它包含字符串常数 `"seeking"`。您可以将此属性与 `state` 属性进行比较，以确定该组件是否处于搜索状态。

### 示例

以下示例使用 `FLVPlayback.SEEKING` 属性确定在 `stateChange` 事件发生时状态是否为 `"seeking"`；如果是，则对该结果显示一条消息。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(eventObject.state == FLVPlayback.SEEKING)
 trace("The current state is " + FLVPlayback.SEEKING);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
```

### 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)



# FLVPlayback.seekPercent()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.seekPercent(percent:Number)
```

## 参数

*percent* 一个数字，指定放置播放头的 FLV 文件长度的百分比。

## 返回

无。

## 说明

方法；搜索到文件的某个百分比并在那里放置播放头。该百分比是介于 0 和 100 之间的数值。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 FLV 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

## 示例

以下示例禁止自动播放 FLV 文件。在 FLV 文件就绪时，将播放头设置为播放时间开始后的 30% 处，并在该点开始播放。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.autoPlay = false;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
 my_FLVPlayback.seekPercent(30);
}
```

```
 my_FLVPlayback.play();
}
my_FLVPlayback.addEventListener("ready", listenerObject);
```

### 另请参见

[FLVPlayback.seek](#)、[FLVPlayback.seek\(\)](#)、[FLVPlayback.seekSeconds\(\)](#)

## FLVPlayback.seekSeconds()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.seekSeconds(time:Number)
```

### 参数

*time* 一个数字，它指定总播放时间中播放头所在位置的时间，以秒为单位。

### 返回

无。

### 说明

方法；在文件中搜索到给定时间，以秒为单位指定，精确到三位小数（毫秒）。此方法执行与 `seek()` 方法相同的操作；提供此方法是为了与 `seekPercent()` 方法对称。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 FLV 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

### 示例

以下示例禁止自动播放 FLV 文件，调用 `seekSeconds()` 方法以将播放头设置为视频中的 5 秒处，并从该点开始播放 FLV 文件。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```

/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.autoPlay = false;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
 my_FLVPlayback.seekSeconds(4);
 my_FLVPlayback.play();
}
my_FLVPlayback.addEventListener("ready", listenerObject);

```

### 另请参见

[FLVPlayback.seek](#)、[FLVPlayback.seek\(\)](#)、[FLVPlayback.seekPercent\(\)](#)

## FLVPlayback.seekToNavCuePoint()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```

my_FLVPlayback.seekToNavCuePoint(time:Number):Void
my_FLVPlayback.seekToNavCuePoint(name:String):Void
my_FLVPlayback.seekToNavCuePoint(cuePoint:Object):Void

```

### 参数

*time* 一个数字，表示要搜索的导航提示点的时间。该方法仅使用前三个小数位，而对所有多余小数位进行舍入。

*name* 一个字符串，包含要搜索的提示点的名称。

*cuePoint* 一个提示点对象，在其中设置 *time* 和 *name* 属性以指定要搜索的提示点。

### 返回

无。

## 说明

方法；搜索到匹配指定的时间或更晚的导航提示点。如果时间未定义、为 `null` 或小于 `0`，则该方法在时间 `0` 处开始进行搜索。

如果您只指定一个时间，则该方法搜索到匹配该时间或更晚的提示点。

如果您指定一个名称，则该方法搜索到与该名称匹配的第一个启用的提示点（有关启用 / 禁用提示点的更多信息，请参见第 613 页的“[FLVPlayback.setFLVCuePointEnabled\(\)](#)”）。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 `FLV` 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

## 示例

以下示例在发生 `ready` 事件时搜索到名为 `point2` 的提示点。`cuePoint` 事件处理函数显示发生的每个提示点的 `name`、`time` 和 `type` 值。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 `my_FLVPlybk`。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 my_FLVPlybk.seekToNavCuePoint("point2");
}
my_FLVPlybk.addEventListener("ready", listenerObject);
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object):Void {
 trace("Cue point name is: " + eventObject.info.name);
 trace("Cue point time is: " + eventObject.info.time);
 trace("Cue point type is: " + eventObject.info.type);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.cuePoint](#)、[FLVPlayback.seek](#)、[FLVPlayback.seek\(\)](#)、[FLVPlayback.seekToNextNavCuePoint\(\)](#)

# FLVPlayback.seekToNextNavCuePoint()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.seekToNextNavCuePoint([time:Number])
```

## 参数

*time* 一个数字，表示查找下一个导航提示点的开始时间，以秒为单位。默认值是当前的 `playheadTime` 属性。可选。

## 返回

无。

## 说明

方法：根据当前的 `playheadTime` 属性值搜索到下一个导航提示点。该方法跳过已禁用的导航提示点，并且在没有其它提示点的情况下转到 **FLV** 文件的末尾。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 **FLV** 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

## 示例

以下示例在出现名为 `point2` 的提示点时搜索到下一个导航提示点。这会导致跳过 **FLV** 文件中名为 `point2` 和 `point3` 的两个提示点之间的那一部分。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(event:Object:Object) {
 if(eventObject.info.name == "point2")
```

```
 my_FLVPlayback.seekToNextNavCuePoint(eventObject.info.time);
 }
 my_FLVPlayback.addEventListener("cuePoint", listenerObject);
 my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

### 另请参见

[FLVPlayback.cuePoint](#)、[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.playheadTime](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.isFLVCuePointEnabled\(\)](#)、[FLVPlayback.setFLVCuePointEnabled\(\)](#)

## FLVPlayback.seekToPrevNavCuePoint()

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.seekToPrevCueNavPoint([time:Number])
```

### 参数

*time* 一个数字，表示查找上一个导航提示点的开始时间，以秒为单位。默认值为 `playheadTime` 属性的当前值。可选。

### 返回

无。

### 说明

方法；根据当前的 `playheadTime` 属性值搜索到上一个导航提示点。如果没有上一个提示点，则会转到开始处。此方法会跳过已禁用的导航提示点。

出于几种原因，在调用一种搜索方法或者设置 `playheadTime` 以引发搜索后，`playheadTime` 属性中不太可能立即包含预期值。首先，对于渐进式下载，由于只能搜索关键帧，因此搜索将转到指定时间后第一个关键帧的时间。（对于流式下载，搜索总是转到指定的确切时间，即使源 FLV 文件在此处没有关键帧也是如此。）其次，由于搜索是异步执行的，因此，如果调用搜索方法或者设置 `playheadTime` 属性，`playheadTime` 是不会立即更新的。若要在搜索完成后获得这一时间，需要侦听 `seek` 事件，该事件只有在更新 `playheadTime` 属性后才启动。

## 示例

以下示例在 **point2** 提示点出现时搜索到上一个导航提示点，并且创建一个循环来播放 FLV 文件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
 if(eventObject.info.name == "point2")
 my_FLVPlybk.seekToPrevNavCuePoint(eventObject.info.time);
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject);
```

## 另请参见

[FLVPlayback.cuePoint](#)、[FLVPlayback.findCuePoint\(\)](#)、[FLVPlayback.playheadTime](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)、[FLVPlayback.isFLVCuePointEnabled\(\)](#)、[FLVPlayback.setFLVCuePointEnabled\(\)](#)

# FLVPlayback.seekToPrevOffset

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk.seekToPrevOffset*

## 说明

属性：一个数字（以秒为单位），[seekToPrevNavCuePoint\(\)](#) 方法将它的时间与上一个提示点的时间进行比较时使用该数字。如果您刚好处于某个提示点之前，则该方法使用该值可以确保您能够跳过该提示点到达上一个提示点，而不必转到刚发生的同一个提示点。默认值为 1 秒。

## 示例

以下示例最初将 `seekToPrevOffset` 属性设置为 **10**，这导致首先调用 `seekToPrevNavCuePoint()` 方法以找到提示点 **point1**。但是，当 **point3** 处发生初始 `cuePoint` 事件时，该示例将 `seekToPrevOffset` 属性降低到 **1** 秒，这将导致随后对 `seekToPrevNavCuePoint()` 方法的调用到达提示点 **point2**。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 **1** 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object) {
 my_FLVPlybk.seekToPrevOffset = 10;
 my_FLVPlybk.seekToNavCuePoint("point3");
}
my_FLVPlybk.addEventListener("ready", listenerObject)
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
 trace("hit cue point at " + eventObject.info.time);
 if(eventObject.info.name == "point3"){
 my_FLVPlybk.seekToPrevNavCuePoint(eventObject.info.time);
 my_FLVPlybk.seekToPrevOffset = 1;
 }
}
my_FLVPlybk.addEventListener("cuePoint", listenerObject)
```

## 另请参见

[FLVPlayback.seekToPrevNavCuePoint\(\)](#)



# FLVPlayback.setFLVCuePointEnabled()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVplybk.setFLVCuePointEnabled(enabled:Boolean, time:Number)
my_FLVplybk.setFLVCuePointEnabled(enabled:Boolean, name:String)
my_FLVplybk.setFLVCuePointEnabled(enabled:Boolean, cuePoint:Object)
```

## 参数

*enabled* 一个布尔值，指定启用 (true) 还是禁用 (false) 某个 FLV 文件提示点。

*time* 一个数字，表示要设置的提示点的时间，以秒为单位。

*name* 要设置的提示点的名称。

*cuePoint* 一个提示点对象，该对象具有匹配要设置的提示点的 *name* 和 *time* 属性。该方法不检查传入提示点对象上的任何其它属性。如果未定义 *time* 或 *name*，则该方法尝试只使用可用值来匹配提示点。

## 返回

一个数字。如果 *metadataLoaded* 为 true，则该方法返回其启用状态已更改的提示点的数量。如果 *metadataLoaded* 为 false，则该方法返回 -1，因为该组件尚无法确定要设置的提示点（如果有）。但是，在元数据到达后，该组件会相应地设置指定的提示点。

## 说明

方法；启用或禁用一个或多个 FLV 文件提示点。之所以禁用这些提示点，是因为要将它们作为事件进行调度，或者因为要使用 `seekToPrevNavCuePoint()`、`seekToNextNavCuePoint()` 和 `seekToNavCuePoint()` 方法导航到这些提示点。

在将 *contentPath* 属性设置为不同的 FLV 文件时，会删除提示点信息，因此，在为下一个要加载的 FLV 文件设置提示点信息前，应先设置 *contentPath* 属性。

在加载元数据后，对 `isFLVCuePointEnabled()` 方法的调用才反映此函数所导致的更改。

## 示例

以下示例在发生 `ready` 事件时禁用 `point2` 和 `point3` 提示点。`cuePoint` 事件处理函数在“输出”面板中显示出现的每个提示点的名称和时间。FLV 文件包含以下嵌入的提示点：时间为 `00:00:00:418` 的 `point1`，时间为 `00:00:07.748` 的 `point2`，时间为 `00:00:16:020` 的 `point3`。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 `my_FLVPlybk`。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
function ready(eventObject:Object) {
 my_FLVPlybk.setFLVCuePointEnabled(false, "point2");
 my_FLVPlybk.setFLVCuePointEnabled(false, 16.02);
}
my_FLVPlybk.addEventListener("ready", ready);
function cuePoint(eventObject:Object) {
 trace("Cue point name is: " + eventObject.info.name);
 trace("Cue point time is: " + eventObject.info.time);
}
my_FLVPlybk.addEventListener("cuePoint", cuePoint);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

## 另请参见

[FLVPlayback.cuePoint](#)、[FLVPlayback.findCuePoint\(\)](#)、  
[FLVPlayback.Findnearestcuepoint\(\)](#)、[FLVPlayback.findNextCuePointWithName\(\)](#)、  
[FLVPlayback.isFLVCuePointEnabled\(\)](#)、[FLVPlayback.seekToNavCuePoint\(\)](#)、  
[FLVPlayback.seekToNextNavCuePoint\(\)](#)、[FLVPlayback.seekToPrevNavCuePoint\(\)](#)

# FLVPlayback.setScale()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlybk.setScale(xs:Number, ys:Number)
```

## 参数

*xs* 表示水平缩放的一个数字。

*ys* 表示垂直缩放的一个数字。

## 返回

无。

## 说明

方法；同时设置 `scaleX` 和 `scaleY` 属性。因为单独设置任一属性，都可能导致自动调整大小，所以，同时设置 `scaleX` 和 `scaleY` 属性可能比单独设置它们效率更高。

如果 `autoSize` 为 `true`，则此方法无效，因为播放器可以设置其自身的尺寸。如果 `maintainAspectRatio` 属性为 `true`，并且 `autoSize` 为 `false`，则更改 `scaleX` 或 `scaleY` 会导致自动调整大小。

## 示例

以下示例调用 `setScale()` 方法以缩放 `FLVPlayback` 实例的水平 (x) 和垂直 (y) 尺寸。该示例将 `maintainAspectRatio` 属性设置为 `false`，以防止自动调整大小并允许根据指定显示缩放。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;

my_FLVPlybk.maintainAspectRatio = false;
my_FLVPlybk.setScale(200, 175);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.scaleX](#)、[FLVPlayback.scaleY](#)、[FLVPlayback.setSize\(\)](#)

# FLVPlayback.setSize()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.setSize(w:Number, h:Number)
```

## 参数

*w* 一个数字，指定视频播放器的宽度。

*h* 一个数字，指定视频播放器的高度。

## 返回

无。

## 说明

方法；同时设置宽度和高度。因为单独设置任一属性，都可能导致自动调整大小，所以，同时设置 `width` 和 `height` 属性可能比单独设置它们效率更高。

如果 `autoSize` 为 `true`，则此方法无效，因为播放器可以设置其自身的尺寸。如果 `maintainAspectRatio` 为 `true`，并且 `autoSize` 属性为 `false`，则更改宽度或高度会导致自动调整大小。

## 示例

以下示例调用 `setSize()` 方法，以便将 `FLVPlayback` 实例的大小设置为宽 150 像素、高 150 像素。`resize` 事件处理函数显示实际的宽度和高度，因为 `maintainAspectRatio` 属性默认为 `true`，所以自动调整大小将保持高宽比。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
// maintainAspectRatio 默认为 true，因此尺寸将反映为
my_FLVPlayback.setSize(150, 150);
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object):Void {
 trace("Player's width is: " + my_FLVPlayback.width)
 trace("Player's height is: " + my_FLVPlayback.height)
```

```
};
my_FLVPlayback.addEventListener("resize", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

#### 另请参见

[FLVPlayback.height](#)、[FLVPlayback.width](#)、[FLVPlayback.setScale\(\)](#)

## FLVPlayback.skin

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.skin
```

### 说明

属性：一个字符串，指定外观 SWF 文件的 URL。此字符串可以包含文件名、相对路径（如 `Skins/my_Skin.swf`）或绝对 URL（如 `http://www.myskins.org/MySkin.swf`）。

### 示例

以下示例将外观 `ArcticExternal.swf` 应用到 `FLVPlayback` 组件的实例中。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。将 `ArcticExternalAll.swf` 文件从 `Flash Configuration/Skins` 文件夹复制到您的工作文件夹中。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.skin = "ArcticExternalAll.swf";
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

#### 另请参见

[FLVPlayback.bufferingBarHidesAndDisablesOthers](#)、[FLVPlayback.skinAutoHide](#)、[FLVPlayback.skinError](#)、[FLVPlayback.skinLoaded](#)

# FLVPlayback.skinAutoHide

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVplybk*.skinAutoHide

## 说明

属性；一个布尔值，如果为 `true`，则鼠标不在视频上时，隐藏组件外观。此属性只影响通过设置 `skin` 属性加载的外观，而不影响从 `FLV` 回放自定义用户界面组件创建的外观。默认值为 `false`。

## 示例

以下示例将 `skinAutoHide` 属性设置为 `true`，因此，除非鼠标位于视频上，否则包含回放控件的组件外观并不出现。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。在“组件检查器”中选择一个外观，然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
my_FLVplybk.skinAutoHide = true;
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.bufferingBarHidesAndDisablesOthers](#)、[FLVPlayback.skin](#)

# FLVPlayback.skinError

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.skinError = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("skinError", listenerObject);
```

## 说明

事件：在加载外观 SWF 文件发生错误时调度。该事件具有包含错误消息的 message 属性。

## 示例

以下示例尝试用虚构的外观文件的名称加载 skin 属性，并且在 skinError 事件发生时显示事件 message 属性的内容。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴的第 1 帧中：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;

var listenerObject:Object = new Object();
listenerObject.skinError = function(eventObject:Object):Void {
 trace(eventObject.message);
}
my_FLVplybk.addEventListener("skinError", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
my_FLVplybk.skin = "NoSuchSkin.swf";
```

## 另请参见

[FLVPlayback.skin](#)、[FLVPlayback.skinLoaded](#)

# FLVPlayback.skinLoaded

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.skinLoaded = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("skinLoaded", listenerObject);
```

## 说明

事件：在加载外观 SWF 文件时调度。直到 ready 和 skinLoaded（或 skinError）事件都启动，该组件才开始播放 FLV 文件。

## 示例

以下示例在 skinLoaded 事件启动时显示组件的外观的名称。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴的第 1 帧中：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.skinLoaded = function(eventObject:Object):Void {
 trace("Skin: " + eventObject.target.skin + " has loaded");
};
my_FLVplybk.addEventListener("skinLoaded", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.skin](#)、[FLVPlayback.skinError](#)



# FLVPlayback.state

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.state*

## 说明

属性；一个字符串，指定组件的状态。用 `load()`、`play()`、`stop()`、`pause()` 和 `seek()` 方法来设置此属性。只读。

`state` 属性的可能值包括："buffering"、"connectionError"、"disconnected"、"loading"、"paused"、"playing"、"rewinding"、"seeking" 和 "stopped"。您可以使用 `FLVPlayback` 类属性测试这些状态。有关更多信息，请参见[第 499 页的“FLVPlayback 类属性”](#)。

## 示例

在以下示例中，在 `FLV` 文件播放过程中每次发生 `stateChange` 事件时，都在“输出”面板内显示 `state` 属性。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 trace(my_FLVPlayback.state);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.buffering](#)、[FLVPlayback.paused](#)、[FLVPlayback.playing](#)、[FLVPlayback.stateChange](#)、[FLVPlayback.stopped](#)

# FLVPlayback.stateChange

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("stateChange", listenerObject);
```

## 说明

事件：回放状态更改时调度。该事件对象具有 state 和 playheadTime 属性。

此事件可用于跟踪回放何时进入或离开不可响应状态（例如在连接、调整大小或后退过程中），在这些时间中，当播放器进入可响应状态时，play()、pause()、stop() 和 seek() 方法会将要执行的请求排队。

此事件具有 vp 属性，它是适用于此事件的视频播放器的索引号。有关 vp 属性的更多信息，请参见第 505 页的 [FLVPlayback.activeVideoPlayerIndex](#) 和第 632 页的 [FLVPlayback.visibleVideoPlayerIndex](#)。

## 示例

在以下示例中，在 FLV 文件播放过程中每次发生 stateChange 事件时，都在“输出”面板内显示 state 属性。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPLYBK**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPLYBK
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 trace(my_FLVPLYBK.state);
};
my_FLVPLYBK.addEventListener("stateChange", listenerObject);
my_FLVPLYBK.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

另请参见

[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.state](#)

## FLVPlayback.stateResponsive

可用性

Flash Player 8。

版本

Flash Professional 8。

用法

*my\_FLVPlayback.stateResponsive*

说明

属性；一个布尔值，如果状态为可响应，则为 `true`。如果状态为不可响应，则对 `play()`、`load()`、`stop()`、`pause()` 和 `seek()` 方法的调用将被排队，以供以后在状态更改为可响应状态时执行。因为这些调用被排队并在以后执行，所以，通常不必跟踪 `stateResponsive` 属性的值。可响应状态是：`disconnected`、`stopped`、`playing`、`paused` 和 `buffering`。只读。

示例

以下示例在 FLV 文件播放时随着状态的变化显示 `state` 和 `stateResponsive` 属性的值。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(event:Object):Void {
 trace(my_FLVPlayback.state + "; responsive: " + my_FLVPlayback.stateResponsive);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.stop()

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlybk*.stop()

## 参数

无。

## 返回

无。

## 说明

方法；停止播放视频。如果 `autoRewind` 属性为 `true`，则 **FLV** 文件会后退到开始处。

## 示例

以下示例侦听 `playheadUpdate` 事件，当已运行的 `playheadTime` 大于或等于 5 秒时，侦听器会调用 `stop()` 方法停止播放 **FLV** 文件。另一个侦听器侦听 `stopped` 事件，并会显示 `playheadTime` 和 `state` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.autoRewind = false;
var listenerObject:Object = new Object();
listenerObject.stopped = function(eventObject:Object):Void {
 trace("playhead time is: " + eventObject.playheadTime);
 trace("The video player state is: " + eventObject.state);
};
my_FLVPlybk.addEventListener("stopped", listenerObject);
listenerObject.playheadUpdate = function(eventObject:Object):Void {
 if (eventObject.playheadTime >= 5) {
 my_FLVPlybk.stop();
 }
};
```

```
my_FLVPlybk.addEventListener("playheadUpdate", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.pause\(\)](#)、[FLVPlayback.play\(\)](#)

## FLVPlayback.stopButton

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

*my\_FLVPlybk*.stopButton

### 说明

属性；作为 **Stop** 按钮控件的 **MovieClip** 对象。有关将 **FLV** 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 **FLV** 回放自定义用户界面组件的外观”。

### 示例

以下示例使用 **backButton**、**forwardButton**、**playButton**、**pauseButton** 和 **stopButton** 属性，来将各个 **FLV** 回放自定义用户界面控件附加到 **FLVPlayback** 组件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后在“组件”检查器中将 **skin** 参数设置为“无”。接下来，添加以下各个 **FLV** 回放自定义用户界面组件，并为其指定括号中显示的实例名称：**BackButton** (**my\_bkbtn**)、**ForwardButton** (**my\_fwdbtn**)、**PlayButton** (**my\_plybtn**)、**PauseButton** (**my\_pausbtn**) 和 **StopButton** (**my\_stopbtn**)。然后将以下几行代码添加到“动作”面板中：

```
/**
 * Requires:
 * - FLVPlayback component on the Stage with an instance name of my_FLVPlybk
 * - FLV Custom UI BackButton, ForwardButton, PlayButton, PauseButton, and
 * StopButton components in the Library
 */
import mx.video.*;
my_FLVPlybk.backButton = my_bkbtn;
my_FLVPlybk.forwardButton = my_fwdbtn;
my_FLVPlybk.playButton = my_plybtn;
my_FLVPlybk.pauseButton = my_pausbtn;
```

```
my_FLVPlybk.stopButton = my_stopbttt;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.pauseButton](#)、[FLVPlayback.playButton](#)、  
[FLVPlayback.playPauseButton](#)、[FLVPlayback.skin](#)、[FLVPlayback.stopped](#)

## FLVPlayback.STOPPED

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.FLVPlayback.STOPPED`

### 说明

一个只读的 **FLVPlayback** 类属性，它包含字符串常数 "stopped"。您可以将此属性与 `state` 属性进行比较，以确定该组件是否处于停止状态。

### 示例

以下示例在该组件进入停止状态时显示 `FLVPlayback.STOPPED` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 要求:
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
*/
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 if(eventObject.state == FLVPlayback.STOPPED)
 trace("State is " + FLVPlayback.STOPPED);
};
my_FLVPlybk.addEventListener("stateChange", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

### 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)

# FLVPlayback.stopped

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.stopped = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("stopped", listenerObject);
```

## 说明

事件：在进入停止状态时调度。调用 stop() 方法或单击 stopButton 控件时发生此事件。在某些情况下，在加载 FLV 文件时，如果 autoPlay 属性为 false（也可能进入 paused 状态），也可能发生此事件。当播放头停在 FLV 文件末端时，FLVPlayback 实例也会调度此事件。该事件对象具有 state、playheadTime 和 vp 属性，最后一个属性是该事件所应用的视频播放器的索引编号。有关 vp 属性的更多信息，请参见[第 505 页的](#)

[FLVPlayback.activeVideoPlayerIndex](#) 和[第 632 页的](#)

[FLVPlayback.visibleVideoPlayerIndex](#)。

FLVPlayback 实例还会调度 stateChange 事件。

## 示例

以下示例侦听播放 FLV 文件过程中发生的 stopped 事件。该事件发生时，该示例会在“输出”面板中显示播放头已运行的时间。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stopped = function(eventObject:Object):Void {
 trace(my_FLVplybk.state + ": playhead time is: " +
 eventObject.playheadTime);
};
my_FLVplybk.addEventListener("stopped", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
```

## 另请参见

[FLVPlayback.addListener\(\)](#)、[FLVPlayback.playheadTime](#)、[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)、[FLVPlayback.stop\(\)](#)

# FLVPlayback.stopped

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.stopped*

## 说明

属性：一个布尔值，如果 **FLVPlayback** 实例的状态为 `stopped`，该值则为 `true`。只读。

## 示例

以下示例会侦听播放 **FLV** 文件过程中发生的 `stateChange` 事件。当该事件发生时，该示例在“输出”面板中显示 `stopped` 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object):Void {
 trace(my_FLVPlayback.state + ": stopped property is: " +
 my_FLVPlayback.stopped);
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.state](#)、[FLVPlayback.stateChange](#)、[FLVPlayback.stop\(\)](#)、[FLVPlayback.stopped](#)



# FLVPlayback.totalTime

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlayback.totalTime`

## 说明

属性；表示视频总播放时间的数字，以秒为单位。在从 FCS 流式加载并使用默认 NCManager 时，服务器端 API 会自动确定此值，并且该值会覆盖通过此属性设置或从元数据中收集的任何值。如果您在 SMIL 文件中设置此值，上述说明同样成立。设置 `contentPath` 属性后，可以在到达停止或播放状态时读取此属性。此属性对于来自 FCS 的实时流无意义。

对于 HTTP 下载，如果 FLV 文件含有嵌入的元数据，则会自动确定该值；否则，应显式设置该值或者设置为 0。如果您显式设置该值，则会忽略流中的元数据值。

设置此属性时，该值会影响通过设置 `contentPath` 加载的下一个 FLV 文件。但它对已加载的 FLV 文件没有任何影响。而且，在加载 FLV 文件之后，此属性才会返回传递到其中的新值。

如果从未设置此属性（显式或自动），则回放仍会有效，但可能导致搜索控件出问题。

## 示例

以下示例在加载完成后发生 `ready` 事件时，以秒为单位显示 FLV 文件的总时间。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 trace("Total play time for this video is: " + my_FLVPlayback.totalTime);
};
my_FLVPlayback.addEventListener("ready", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.contentPath](#)、[FLVPlayback.playheadTime](#)、[FLVPlayback.playing](#)、[FLVPlayback.stopped](#)

# FLVPlayback.transform

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.transform*

## 说明

属性；一个对象，提供对 `Sound.setTransform()` 和 `Sound.getTransform()` 方法的直接访问，以提供声音控制。您必须将此属性设置为一个对象，以便初始化该对象并使更改生效。读取此属性可以为您提供可以更改的当前设置的副本。默认值为 `undefined`。

## 示例

以下示例设置 `transform` 属性，以便只从左扬声器播放 FLV 文件的声音。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
/* 只从左扬声器播放所有音频 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.stateChange = function(eventObject:Object) {
 if (eventObject.target.state == "loading") { // 如果为 loading
 myTransform = new Object();
 myTransform.ll = 100;
 myTransform.lr = 100;
 myTransform.rr = 0;
 myTransform.rl = 0;
 my_FLVPlayback.transform = myTransform;
 }
};
my_FLVPlayback.addEventListener("stateChange", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.volume](#)

# FLVPlayback.version

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
mx.video.FLVPlayback.version
```

## 说明

一个只读的 **FLVPlayback** 类属性，它包含组件的版本号。

## 示例

以下代码在“输出”面板中显示组件的版本号。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
import mx.video.*;
trace(FLVPlayback.version);
```

# FLVPlayback.visible

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
my_FLVPlayback.visible
```

## 说明

属性；一个布尔值，如果为 **true**，则会使 **FLVPlayback** 组件可见。如果为 **false**，则会使组件不可见。默认值为 **true**。

## 示例

以下示例将 **visible** 属性设置为 **false**，以便在 FLV 文件播放完成后，使 **FLVPlayback** 实例不可见。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```

/**
 * 要求:
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object):Void {
 my_FLVPlayback.visible = false;
};
my_FLVPlayback.addEventListener("complete", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";

```

### 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.closeVideoPlayer\(\)](#)、[FLVPlayback.visibleVideoPlayerIndex](#)

## FLVPlayback.visibleVideoPlayerIndex

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
my_FLVPlayback.visibleVideoPlayerIndex
```

### 说明

属性；一个数字，可用于管理多个 **FLV** 文件流。设置在其余视频播放器被隐藏和静音的情况下，哪个视频播放器实例可见、可听到并且是由外观或播放控件控制的。默认值是 **0**。它并不使视频播放器成为可供大多数 **API** 使用的目标；请改用 `activeVideoPlayerIndex` 属性。

用于控制影响此属性的尺寸的方法和属性。设置视频播放器尺寸的方法和属性（`setScale()`、`setSize()`、`width`、`height`、`scaleX`、`scaleY`）可用于所有视频播放器。但是，根据是否在这些视频播放器上设置了 `autoSize` 或 `maintainAspectRatio`，播放器可能会有不同的尺寸。使用 `width`、`height`、`scaleX` 和 `scaleY` 属性读取尺寸时，只能为您提供可见视频播放器的尺寸。其它视频播放器的尺寸可能相同，也可能不同。

若要在各个视频播放器不可见时获取它们的尺寸，可侦听 `resize` 事件并存储大小值。

从总体上来说，此属性不会暗示组件是否可见，它只能指示组件不可见时哪个视频播放器可见。若要设置整个组件的可见性，请使用 `visible` 属性。

## 示例

以下示例将创建两个视频播放器，用来在一个 `FLVPlayback` 实例中连续播放两个 FLV 文件。它设置 `visibleVideoPlayerIndex` 属性，以使视频播放器和 FLV 文件可见。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
// 为默认播放器指定 FLV 的名称和位置
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 clouds.flv"
var listenerObject:Object = new Object();
listenerObject.ready = function(eventObject:Object):Void {
 // 添加另一个视频播放器，并指定其 FLV 的名称和位置
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
 // 重置为默认的视频播放器，它将自动播放其 FLV
 my_FLVPlybk.activeVideoPlayerIndex = 0;
};
my_FLVPlybk.addEventListener("ready", listenerObject);
listenerObject.complete = function(eventObject:Object):Void {
 // 如果 complete 事件是针对第二个 FLV，则使默认播放器活动并可见
 if (eventObject.vp == 1) {
 my_FLVPlybk.activeVideoPlayerIndex = 0;
 my_FLVPlybk.visibleVideoPlayerIndex = 0;
 } else { // 使第二个播放器活动并可见，并且播放 FLV
 my_FLVPlybk.activeVideoPlayerIndex = 1;
 my_FLVPlybk.visibleVideoPlayerIndex = 1;
 my_FLVPlybk.play();
 }
};
// 添加 complete 事件的侦听器
my_FLVPlybk.addEventListener("complete", listenerObject);
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)、[FLVPlayback.setScale\(\)](#)、[FLVPlayback.setSize\(\)](#)、[FLVPlayback.height](#)、[FLVPlayback.width](#)、[FLVPlayback.scaleX](#)、[FLVPlayback.scaleY](#)、[FLVPlayback.visible](#)

# FLVPlayback.volume

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.volume

## 说明

属性；一个数字，范围介于 0 到 100 之间，表示音量控件设置。默认值是 100。

## 示例

以下示例将初始音量设置为相对较低的设置 10。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
// 您可以将此值从 0 更改为 100
my_FLVPlayback.volume = 10;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

另请参见

[FLVPlayback.transform](#)、[FLVPlayback.volumeBar](#)、[FLVPlayback.volumeBarInterval](#)、[FLVPlayback.Volumeupdate](#)

# FLVPlayback.volumeBar

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlybk.volumeBarInterval`

## 说明

属性；作为搜索栏控件的 **MovieClip** 对象。有关将 FLV 回放自定义用户界面组件用于回放控件的更多信息，请参见第 485 页的“单独设置 FLV 回放自定义用户界面组件的外观”。

## 示例

以下示例使用 `backButton`、`forwardButton`、`playButton`、`pauseButton`、`stopButton` 和 `volumeBar` 属性，将各个 FLV 自定义用户界面控件附加到 **FLVPlayback** 组件。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**，然后在“组件”检查器中将 `skin` 参数设置为“无”。接下来，添加以下各个 FLV 自定义用户界面组件，并为其指定括号中显示的实例名称：**BackButton** (**my\_bkbtn**)、**ForwardButton** (**my\_fwdbtn**)、**PlayButton** (**my\_plybtn**)、**PauseButton** (**my\_pausbtn**)、**StopButton** (**my\_stopbtn**) 和 **VolumeBar** (**my\_vBar**)。然后将以下几行代码添加到“动作”面板中：

```
/**
 要求：
 - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 - 库中有 FLV 自定义用户界面 BackButton、ForwardButton、PlayButton、PauseButton、
 StopButton 和 VolumeBar 组件
*/
import mx.video.*;
my_FLVPlybk.backButton = my_bkbtn;
my_FLVPlybk.forwardButton = my_fwdbtn;
my_FLVPlybk.playButton = my_plybtn;
my_FLVPlybk.pauseButton = my_pausbtn;
my_FLVPlybk.stopButton = my_stopbtn;
my_FLVPlybk.volumeBar = my_vBar;
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.volume](#)、[FLVPlayback.volumeBarInterval](#)、  
[FLVPlayback.volumeBarScrubTolerance](#)、[FLVPlayback.Volumeupdate](#)

# FLVPlayback.volumeBarInterval

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`my_FLVPlybk.volumeBarInterval`

## 说明

属性；一个数字，指定拖拽时检查音量栏手柄位置的频率，以毫秒为单位。默认值为 250。

## 示例

以下示例将 `volumeBarInterval` 属性设置为 1 秒（1000 毫秒），并且创建一个 `volumeUpdate` 事件，而该事件在用户拖动音量栏上的手柄时显示播放头时间和音量。由于 `volumeBarInterval` 设置，`volumeUpdate` 事件大约按 1 秒钟的间隔发生。

将一个 `FLVPlayback` 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlybk
 */
import mx.video.*;
my_FLVPlybk.volumeBarInterval = 1000;
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(event:Object:Object) {
 trace("Playhead time is: " + my_FLVPlybk.playheadTime);
 trace("Volume is: " + my_FLVPlybk.volume);
};
my_FLVPlybk.addEventListener("volumeUpdate", listenerObject);
my_FLVPlybk.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.volume](#)、[FLVPlayback.volumeBar](#)、

[FLVPlayback.volumeBarScrubTolerance](#)、[FLVPlayback.Volumeupdate](#)



# FLVPlayback.volumeBarScrubTolerance

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback*.volumeBarScrubTolerance

## 说明

属性；一个数字，指定更新之前用户可以移动音量栏手柄的距离。该值以百分比表示。默认值是 5。

## 示例

以下示例将 volumeBarScrubTolerance 属性设置为 20，并且创建一个 volumeUpdate 事件，而该事件在用户拖动音量栏上的手柄时显示音量设置。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.volumeBarScrubTolerance = 20;
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(event:Object:Object) {
 trace("Playhead time is: " + my_FLVPlayback.playheadTime);
 trace("Volume is: " + my_FLVPlayback.volume);
};
my_FLVPlayback.addEventListener("volumeUpdate", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 cuepoints.flv";
```

## 另请参见

[FLVPlayback.volumeBar](#)、[FLVPlayback.volumeBarInterval](#)

# FLVPlayback.Volumeupdate

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(eventObject:Object):Void {
 // 在此插入事件处理代码
};
my_FLVplybk.addEventListener("volumeUpdate", listenerObject);
```

## 说明

事件：在音量由于用户移动 **volumeBar** 控件的手柄或者由于在 **ActionScript** 中设置 **volume** 属性而发生变化时调度。事件对象具有 **volume** 属性。

## 示例

以下示例为用户对音量进行的任何调整而在“输出”面板中显示 **volume** 属性的值。

将一个 **FLVPlayback** 组件拖到舞台上，并为其指定实例名称 **my\_FLVplybk**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVplybk
 */
import mx.video.*;
var listenerObject:Object = new Object();
listenerObject.volumeUpdate = function(eventObject:Object):Void {
 trace("Volume setting is: " + eventObject.volume);
};
my_FLVplybk.addEventListener("volumeUpdate", listenerObject);
my_FLVplybk.contentPath = "http://www.helpexamples.com/flash/video/
cuepoints.flv";
```

## 另请参见

[FLVPlayback.addEventListener\(\)](#)、[FLVPlayback.volume](#)、[FLVPlayback.volumeBar](#)

# FLVPlayback.width

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.width*

## 说明

属性；一个数字，指定舞台上 FLVPlayback 实例的宽度。此属性只影响 FLVPlayback 实例的宽度，并且不包括可被加载的外观 SWF 文件的宽度。使用 FLVPlayback.width 属性，而不使用 MovieClip.\_width 属性，因为如果加载外观 SWF 文件，\_width 属性可能给出不同值。

## 示例

以下示例设置 width 和 height 属性，这会导致 resize 事件（因为 maintainAspectRatio 属性的默认值为 true）。该事件发生时，事件处理函数会在“输出”面板中显示调整大小后的 FLVPlayback 实例的宽度和高度。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
// maintainAspectRatio (默认值为 true) 影响实际尺寸
my_FLVPlayback.width = 400;
my_FLVPlayback.height = 350;
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object) {
 trace("Width is: " + eventObject.target.width + " Height is: " +
 eventObject.target.height);
};
my_FLVPlayback.addEventListener("resize", listenerObject);
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
water.flv";
```

## 另请参见

[FLVPlayback.height](#)、[FLVPlayback.setSize\(\)](#)、[FLVPlayback.preferredHeight](#)、[FLVPlayback.preferredWidth](#)

# FLVPlayback.x

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.x*

## 说明

属性；一个数字，指定视频播放器的水平坐标（位置）。此属性只影响 FLVPlayback 实例的水平位置，并且不包括可更改该位置的外观 SWF 文件（在应用时）的位置。使用 FLVPlayback.x 属性，而不使用 MovieClip.\_x 属性，因为如果加载外观 SWF 文件，\_x 属性可能给出不同值。

## 示例

以下示例将 FLVPlayback 实例放置在距左侧 50 像素和距顶部 25 像素的位置。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_FLVPlayback.x = 50;
my_FLVPlayback.y = 25;
```

## 另请参见

[FLVPlayback.y](#)

# FLVPlayback.y

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

*my\_FLVPlayback.y*

## 说明

属性；一个数字，指定视频播放器的垂直坐标（位置）。此属性只影响 FLVPlayback 实例的垂直位置，而且不包括可更改该位置的外观 SWF 文件（在应用时）的位置。使用 FLVPlayback.y 属性，而不使用 MovieClip.\_y 属性，因为如果加载外观 SWF 文件，\_y 属性可能给出不同值。

## 示例

以下示例将 FLVPlayback 实例放置在距左侧 25 像素和距顶部 50 像素的位置。

将一个 FLVPlayback 组件拖到舞台上，并为其指定实例名称 **my\_FLVPlayback**。然后将以下代码添加到时间轴第 1 帧的“动作”面板上：

```
/**
 * 要求：
 * - 舞台上的 FLVPlayback 组件具有实例名称 my_FLVPlayback
 */
import mx.video.*;
my_FLVPlayback.contentPath = "http://www.helpexamples.com/flash/video/
 water.flv";
my_FLVPlayback.x = 25;
my_FLVPlayback.y = 50;
```

## 另请参见

[FLVPlayback.x](#)

# VideoError 类

继承    Error > VideoError

ActionScript 类名称    mx.video.VideoError

VideoError 类的属性允许您对使用 FLVPlayback 组件时出现的错误条件进行诊断。

mx.video.VideoError 类扩展 Error 类。

## VideoError 类的属性摘要

下表列出了 VideoError 类的属性：

属性	说明
<code>VideoError.code</code>	数字错误代码。
<code>VideoError.DELETE_DEFAULT_PLAYER</code>	一个数字，指示尝试删除默认的视频播放器。
<code>VideoError.DELETE_DEFAULT_PLAYER</code>	一个数字，指示非法提示点。
<code>VideoError.INVALID_CONTENT_PATH</code>	一个数字，指示无效的 <code>contentPath</code> 值。
<code>VideoError.INVALID_SEEK</code>	一个数字，指示无效的搜索。
<code>VideoError.INVALID_XML</code>	一个数字，指示在 XML 文件中遇到了无效的 XML。
<code>VideoError.NO_BITRATE_MATCH</code>	一个数字，指示无法找到匹配任一比特率的默认 FLV 文件。
<code>VideoError.NO_CONNECTION</code>	一个数字，指示该方法无法连接到服务器或者无法找到服务器上的 FLV 文件。
<code>VideoError.NO_CUE_POINT_MATCH</code>	一个数字，指示未找到匹配的提示点。

## VideoError.code

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.VideoError.code`

### 说明

标识错误条件的数字代码。

## 示例

以下示例在“输出”面板中显示错误条件：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {
 trace ("Error code is: " + err.code)
 ...
}
```

# VideoError.DELETE\_DEFAULT\_PLAYER

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.VideoError.DELETE_DEFAULT_PLAYER`

## 说明

值 **1007**，在您调用 `FLVPlayback.closeVideoPlayer()` 方法以尝试关闭默认的视频播放器（编号 **0**）时发生。您不能删除默认的视频播放器。

## 示例

以下代码检查 `DELETE_DEFAULT_PLAYER` 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {

if (err.code == DELETE_DEFAULT_PLAYER) {
 ...
}
}
```

## 另请参见

[FLVPlayback.activeVideoPlayerIndex](#)

# VideoError.ILLEGAL\_CUE\_POINT

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.VideoError.ILLEGAL_CUE_POINT`

## 说明

值 1002，指示找到无效提示点。

## 示例

以下代码检查 `ILLEGAL_CUE_POINT` 错误代码：

```
try {
 ...
} catch (err:VideoError) {
 if (err.code == ILLEGAL_CUE_POINT) {
 ...
 }
}
```

## 另请参见

[FLVPlayback.cuePoint](#)

# VideoError.INVALID\_CONTENT\_PATH

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.VideoError.INVALID_CONTENT_PATH`

## 说明

值 1004，指示已找到无效的 `contentPath` 值。



### 示例

以下代码检查 `INVALID_CONTENT_PATH` 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {
 if (err.code == INVALID_CONTENT_PATH) {
 ...
 }
}
```

### 另请参见

[FLVPlayback.contentPath](#)

## VideoError.INVALID\_SEEK

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

`mx.video.VideoError.INVALID_SEEK`

### 说明

值 1003，指示尝试进行了无效的搜索。

### 示例

以下代码检查 `INVALID_SEEK` 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {
 if (err.code == INVALID_SEEK) {
 ...
 }
}
```

### 另请参见

[FLVPlayback.seek\(\)](#)

# VideoError.INVALID\_XML

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.VideoError.INVALID_XML`

## 说明

值 1005，指示遇到了无效的 XML。在下载和分析 SMIL 文件时可能发生无效的 XML 错误。VideoError.message 属性包含确切描述问题的文本。有关更多信息，请参见[第 653 页](#)的“使用 SMIL 文件”。

## 示例

以下代码检查 INVALID\_XML 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {
 if (err.code == INVALID_XML) {
 ...
 }
}
```

## 另请参见

[FLVPlayback.contentPath](#)

# VideoError.NO\_BITRATE\_MATCH

## 可用性

Flash Player 8。

## 版本

Flash Professional 8。

## 用法

`mx.video.VideoError.NO_BITRATE_MATCH`

### 说明

值 1006，指示没有列出匹配任何比特率的默认 FLV 文件。只在下载和分析 SMIL 文件时发生。有关更多信息，请参见第 653 页的“使用 SMIL 文件”。

### 示例

以下代码检查 NO\_BITRATE\_MATCH 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {
 if (err.code == NO_BITRATE_MATCH) {
 ...
 }
}
```

### 另请参见

[FLVPlayback.bitrate](#)

## VideoError.NO\_CONNECTION

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
mx.video.VideoError.NO_CONNECTION
```

### 说明

值 1000，指示该方法无法连接到服务器或者无法找到服务器上的 FLV 文件。

### 示例

以下代码检查 NO\_CONNECTION 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {
 if (err.code == NO_CONNECTION) {
```

```
 ...
}
}
```

## VideoError.NO\_CUE\_POINT\_MATCH

### 可用性

Flash Player 8。

### 版本

Flash Professional 8。

### 用法

```
mx.video.VideoError.NO_CUE_POINT_MATCH
```

### 说明

值 1001，指示未找到匹配的提示点。

### 示例

以下代码检查 NO\_CUE\_POINT\_MATCH 错误代码：

```
import mx.video.*;

try {
 ...
} catch (err:VideoError) {

 if (err.code == NO_CUE_POINT_MATCH) {
 ...
 }
}
```

### 另请参见

[FLVPlayback.findCuePoint\(\)](#)

# VideoPlayer 类

继承 MovieClip > VideoPlayer 类

ActionScript 类名称 mx.video.VideoPlayer

VideoPlayer 可扩展 MovieClip 类并可包装 Video 对象。

FLVPlayback 类包装 VideoPlayer 类，Macromedia 强烈鼓励您在几乎所有类中都使用 FLVPlayback 类。使用 FLVPlayback 类可以访问 VideoPlayer 类中的所有功能。

之所以在此处包括 VideoPlayer 类，是为了允许您使用较小的 SWF 文件创建视频播放器。VideoPlayer 类不允许您包括外观或回放控件，但它具有较小的 API。例如，尽管将发生 cuePoint 事件，但您无法找到或搜索到提示点。

此外，举例而言，FLVPlayback 类自动与 NCManager 类连接，以访问 FCS 上的流式 FLV 文件。您在设置 contentPath 属性和将某个 URL 传递到 play() 和 load() 方法时，将与 NCManager 类交互。但是，如果您使用 VideoPlayer 类本身，则必须在 ActionScript 代码中包括以下语句，以确保包括 NCManager 类：

```
_forceNCManager:mx.video.NCManager;
```

NCManager 类还具有接口类 INCManager，它允许您通过自定义类替换 NCManager 类，以管理网络通信。如果您进行了上述替换，则还需要包括以下语句，以用提供的类名称替换 NCManager：

```
mx.video.VideoPlayer.DEFAULT_INCMANAGER = "mx.video.NCManager";
```

如果您在使用默认的 NCManager 类，则不需要添加此语句。



您还可以设置 DEFAULT\_INCMANAGER，以便用 FLVPlayback 组件替换默认的 mx.video.NCManager。

为了为多个带宽处理多个流，NCManager 支持 SMIL 的一个子集。有关更多信息，请参见第 653 页的“使用 SMIL 文件”。

本节概要介绍 VideoPlayer 类。您可以在 [www.macromedia.com/go/videoplayer](http://www.macromedia.com/go/videoplayer) 中找到 VideoPlayer 类的方法、属性和事件的详细文档。

# VideoPlayer 类的方法摘要

下表列出了 VideoPlayer 类的方法：

方法	说明
VideoPlayer.addEventListener()	为指定的事件创建侦听器。
VideoPlayer.close()	关闭视频流和 FCS 连接。
VideoPlayer.load()	加载 FLV 文件，但不开始播放。在调整大小（如果需要）后，暂停 FLV 文件。
VideoPlayer.pause()	暂停播放视频流。
VideoPlayer.play()	开始播放视频流。
VideoPlayer.removeEventListener()	删除事件侦听器。
VideoPlayer.seek()	在文件中搜索到指定的时间，用秒表示，小数精确到毫秒。
VideoPlayer.setScale()	同时设置 scaleX 和 scaleY。
VideoPlayer.setSize()	同时设置 width 和 height。
VideoPlayer.stop()	停止播放视频流。

# VideoPlayer 类的属性摘要

VideoPlayer 类具有类和实例属性。

## 类属性

以下属性仅对于 VideoPlayer 类发生。它们是只读常数，适用于 VideoPlayer 类的所有实例。

属性	值	说明
VideoPlayer.BUFFERING	"buffering"	state 属性的可能值。指示在调用 play() 或 load() 后立即进入的状态。
VideoPlayer.CONNECTION_ERROR	"connectionError"	state 属性的可能值。指示发生了连接错误。
VideoPlayer.DEFAULT_INCMANAGER	"mx.video.NCManager"	INCManger 接口的默认 (mx.video.NCManager) 或自定义实现的名称。
VideoPlayer.DISCONNECTED	"disconnected"	state 属性的可能值。指示已断开 FLV 文件流的连接。

属性	值	说明
VideoPlayer.LOADING	"loading"	state 属性的可能值。指示正在加载 FLV 文件。
VideoPlayer.PAUSED	"paused"	state 属性的可能值。指示 FLV 文件已暂停。
VideoPlayer.PLAYING	"playing"	state 属性的可能值。指示正在播放 FLV 文件。
VideoPlayer.RESIZING	"resizing"	state 属性的可能值。指示正在调整 FLV 文件的大小。
VideoPlayer.REWINDING	"rewinding"	state 属性的可能值。指示 FLV 文件正在后退。
VideoPlayer.SEEKING	"seeking"	state 属性的可能值。指示 FLV 文件正在搜索。
VideoPlayer.STOPPED	"stopped"	state 属性的可能值。指示 FLV 文件已停止。
VideoPlayer.version	X.X.X.XX	作为组件的版本号的数字。

## 实例属性

下表列出了 **VideoPlayer** 类的实例属性。这组属性适用于 **VideoPlayer** 类的每个实例。

属性	说明
VideoPlayer.autoRewind	一个布尔值，如果为 true，则在停止播放时，使 FLV 文件后退到第一帧。
VideoPlayer.autoSize	一个布尔值，如果为 true，则视频大小将自动调整为源尺寸。
VideoPlayer.bufferTime	一个数字，指定开始播放视频流前要在内存中缓冲的秒数。
VideoPlayer.bytesLoaded	一个数字，指示 HTTP 下载的下程度，以字节数表示。只读。
VideoPlayer.bytesTotal	一个数字，指定 HTTP 下载的总下载字节数。只读。
VideoPlayer.connected	一个布尔值，它指示 FLV 文件流是否已连接。只读。
VideoPlayer.height	一个数字，指定视频的高度，以像素为单位。
VideoPlayer.idleTimeout	由于播放暂停或停止而导致 FCS 连接空闲之前的时间量，以毫秒为单位。
VideoPlayer.isLive	一个布尔值，如果是实时视频流，则为 true。不适用于 HTTP 下载。

属性	说明
<code>VideoPlayer.isRTMP</code>	一个布尔值，如果正在从 FCS 流式加载 FLV 文件，则为 <code>true</code> 。只读。
<code>VideoPlayer.maintainAspectRatio</code>	一个布尔值，如果为 <code>true</code> ，则保持视频高宽比。
<code>VideoPlayer.metadata</code>	一个对象，它是通过调用 <code>onMetaData()</code> 回调函数（如果有）而接收到的元数据信息包。只读。
<code>VideoPlayer.ncMgr</code>	一个 <code>INCManager</code> 对象，它提供对实现 <code>INCManager</code> 的类的实例的访问。
<code>VideoPlayer.playheadTime</code>	一个数字，表示当前播放头的时间或位置（以秒为单位计算），可以是小数。
<code>VideoPlayer.playheadUpdateInterval</code>	一个数字，表示每个 <code>playheadUpdate</code> 事件之间的时间量，以毫秒为单位。
<code>VideoPlayer.progressInterval</code>	一个数字，表示每个 <code>progress</code> 事件之间的时间量，以毫秒为单位。
<code>VideoPlayer.scaleX</code>	一个数字，指定水平缩放。
<code>VideoPlayer.scaleY</code>	一个数字，指定垂直缩放。
<code>VideoPlayer.state</code>	一个字符串，指定组件的状态。用 <code>load()</code> 、 <code>play()</code> 、 <code>stop()</code> 、 <code>pause()</code> 和 <code>seek()</code> 方法来设置。只读。
<code>VideoPlayer.stateResponsive</code>	一个布尔值，如果处于响应状态（即可以在当前状态中启用控件），则为 <code>true</code> 。只读。
<code>VideoPlayer.totalTime</code>	一个数字，表示视频的总播放时间。
<code>VideoPlayer.transform</code>	一个对象，提供对 <code>Sound.setTransform()</code> 和 <code>Sound.getTransform()</code> 方法的直接访问，以提供更多声音控制。
<code>VideoPlayer.url</code>	一个字符串，指定已加载（或正在加载）的流的 URL。
<code>VideoPlayer.videoHeight</code>	一个数字，指定 FLV 文件的高度。
<code>VideoPlayer.videoWidth</code>	一个数字，指定 FLV 文件的宽度。
<code>VideoPlayer.visible</code>	一个布尔值，如果为 <code>true</code> ，则 FLV 文件可见。
<code>VideoPlayer.volume</code>	一个数字，介于 0 到 100 的范围内，指示音量控制设置。
<code>VideoPlayer.width</code>	一个数字（百分比），指定更新之前用户可以移动音量栏手柄的距离。
<code>VideoPlayer.x</code>	一个数字，指定视频播放器的水平尺寸，以像素为单位。
<code>VideoPlayer.y</code>	一个数字，指定视频播放器的垂直尺寸，以像素为单位。



# VideoPlayer 类的事件摘要

下表列出了 VideoPlayer 类的事件：

事件	说明
VideoPlayer.close	通过超时或通过调用 close() 方法关闭视频流时调度。
VideoPlayer.complete	播放完成（到达 FLV 文件的末端）时调度。
VideoPlayer.cuePoint	到达提示点时调度。
VideoPlayer.metadataReceived	第一次到达 FLV 文件元数据时调度。
VideoPlayer.playheadUpdate	在播放 FLV 文件时每隔 0.25 秒调度一次。
VideoPlayer.progress	每隔 0.25 秒调度一次，从调用 load() 方法时开始，到所有字节加载结束或者出现网络错误时结束。
VideoPlayer.ready	加载 FLV 文件并可以显示它时调度。
VideoPlayer.resize	调整视频大小时调度。
VideoPlayer.rewind	通过调用 seek() 向后移动播放头位置时或者完成自动后退操作时调度。
VideoPlayer.stateChange	回放状态发生更改时调度。

## 使用 SMIL 文件

为了为多个带宽处理多个流，VideoPlayer 类可以使用支持 SMIL 的一个子集的辅助类 (NCManager)。SMIL 用于标识视频流的位置、FLV 文件的布局（宽和高）以及对应于不同带宽的源 FLV 文件。它还可以用于指定 FLV 文件的比特率和持续时间。

以下示例显示一个 SMIL 文件，该文件使用 RTMP 从 FCS 流式加载多个带宽 FLV 文件：

```
<smil>
 <head>
 <meta base="rtmp://myserver/mypgm/" >
 <layout>
 <root-layout width="240" height="180" >
 </layout>
 </head>
 <body>
 <switch>
 <video src="myvideo_mdm.flv" system-bitrate="56000"
dur="3:00.1">
 <video src="myvideo_isdn.flv" system-bitrate="128000"
dur="3:00.1">
 <ref src="myvideo_cable.flv" dur="3:00.1"/>
 </switch>
 </body>
```

</smil>

<head> 标签可以包含 <meta> 和 <layout> 标签。<meta> 标签仅支持 base 属性，该属性用于指定流式视频（来自 FCS 的 RTMP）的 URL。

<layout> 标签仅支持 root-layout 元素，该元素用于设置 height 和 width 属性，因此可确定用来呈现 FLV 文件的窗口的大小。这两个属性仅接受像素值，而不接受百分比。

在 SMIL 文件的正文内，您或者可以包括指向 FLV 源文件的单个链接；或者，如果您正在从 FCS 流式加载多个带宽的多个文件（如前面的示例中所示），则可以使用 <switch> 标签列出这些源文件。

<switch> 标签内的 video 和 ref 标签意义是相同的；也就是说，它们都可以使用 src 属性指定 FLV 文件。进一步讲，每种标签都可以使用 region、system-bitrate 和 dur 属性指定 FLV 文件的区域、所需的最小带宽和持续时间。

在 <body> 标签内，只允许出现 <video>、<src> 或 <switch> 标签中的一个。

以下示例为不使用带宽检测的单个 FLV 文件显示渐进式下载：

```
<smil>
 <head>
 <layout>
 <root-layout width="240" height="180" />
 </layout>
 </head>
 <body>
 <video src="myvideo.flv" />
 </body>
</smil>
```

## <smil>

### 可用性

Flash Professional 8。

### 用法

<smil>

...

*child tags*

...

</smil>

### 属性

无。

### 子标签

<head>, <body>

### 父标签

无。

### 说明

顶级标签，用于标识 SMIL 文件。

### 示例

以下示例显示指定了三个 FLV 文件的一个 SMIL 文件：

```
<smil>
 <head>
 <meta base="rtmp://myserver/mypgm/" >
 <layout>
 <root-layout width="240" height="180" >
 </layout>
 </head>
 <body>
 <switch>
 <video src="myvideo_mdm.flv" system-bitrate="56000"
dur="3:00.1">
 <video src="myvideo_isdn.flv" system-bitrate="128000"
dur="3:00.1">
 <ref src="myvideo_cable.flv" dur="3:00.1"/>
 </switch>
 </body>
</smil>
```

## <head>

### 可用性

Flash Professional 8。

### 用法

<head>

...

*child tags*

...

</head>

### 属性

无。

### 子标签

<meta>, <layout>

### 父标签

<smil>

### 说明

支持 <meta> 和 <layout> 标签，并指定源 FLV 文件的位置和默认布局（高度和宽度）。

### 示例

以下示例将根布局设置为 240 x 180 像素：

```
<head>
 <meta base="rtmp://myserver/mypgm/" >
 <layout>
 <root-layout width="240" height="180" >
 </layout>
</head>
```

## <meta>

### 可用性

Flash Professional 8。

### 用法

<meta/>

### 属性

base

### 子标签

<layout>

### 父标签

无。

### 说明

包含 base 属性，该属性指定源 FLV 文件的位置 (RTMP URL)。

### 示例

以下示例为 myserver 上的基础位置显示 meta 标签：

```
<meta base="rtmp://myserver/mypgm/" >
```

## <layout>

### 可用性

Flash Professional 8。

### 用法

```
<layout>
...
child tags
...
</layout>
```

### 属性

无。

### 子标签

<root-layout>

### 父标签

<meta>

### 说明

指定 FLV 文件的宽度和高度。

### 示例

以下示例指定了一个 240 x 180 像素的布局：

```
<layout>
 <root-layout width="240" height="180" >
</layout>
```

## <root-layout>

### 可用性

Flash Professional 8。

### 用法

```
<root-layout...attributes.../>
```

### 属性

Width、height

### 子标签

None。

## 父标签

<layout>

## 说明

指定 FLV 文件的宽度和高度。

## 示例

以下示例指定了一个 240 x 180 像素的布局：

```
<root-layout width="240" height="180" >
```

## <body>

## 可用性

Flash Professional 8。

## 用法

```
<body>
...
child tags
...
</body>
```

## 属性

无。

## 子标签

<video>, <ref>, <switch>

## 父标签

<smil>

## 说明

包含 <video>、<ref> 和 <switch> 标签，用于分别指定源 FLV 文件的名称和持续时间以及最小带宽。在使用 <switch> 标签时只支持 system-bitrate 属性。在 <body> 标签内，只支持出现 <switch>、<video> 或 <ref> 标签中的一个实例。

## 示例

以下示例指定了三个 FLV 文件，其中两个是使用 video 标签指定的，另一个则是使用 ref 标签指定的：

```
<body>
 <switch>
 <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1">
 <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1">
 <ref src="myvideo_cable.flv" dur="3:00.1"/>
 </switch>
</body>
```

## <video>

### 可用性

Flash Professional 8。

### 用法

```
<video...attributes.../>
```

### 属性

src, system-bitrate, dur

### 子标签

None

### 父标签

<body>

### 说明

与 <ref> 标签的用途相当。支持 src 和 dur 属性，这两个属性指定源 FLV 文件的名称及其持续时间。dur 属性支持完整时间格式 (00:03:00:01) 和不完整时间格式 (03:00:01)。

## 示例

以下示例将设置视频的来源和持续时间：

```
<video src="myvideo_mdm.flv" dur="3:00.1">
```

## <ref>

### 可用性

Flash Professional 8。

### 用法

```
<ref...attributes.../>
```

### 属性

src, system-bitrate, dur

### 子标签

None

### 父标签

<body>

### 说明

与 <video> 标签的用途相当。支持 src 和 dur 属性, 这两个属性指定源 FLV 文件的名称及其持续时间。dur 属性支持完整时间格式 (00:03:00:01) 和不完整时间格式 (03:00:01)。

### 示例

以下示例将设置视频的来源和持续时间:

```
<ref src="myvideo_cable.flv" dur="3:00.1"/>
```

## <switch>

### 可用性

Flash Professional 8。

### 用法

```
<switch>
...
child tags
...
</switch/>
```

### 属性

无

### 子标签

<video>, <ref>



## 父标签

<body>

## 说明

与 <video> 或 <ref> 子标签一起使用，用于为多个带宽视频流列出 **FLV** 文件。<switch> 标签支持 system-bitrate 属性，该属性指定最小带宽以及 src 和 dur 属性。

## 示例

以下示例指定了三个 **FLV** 文件，其中两个是使用 video 标签指定的，另一个则是使用 ref 标签指定的：

```
<switch>
 <video src="myvideo_mdm.flv" system-bitrate="56000" dur="3:00.1">
 <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1">
 <ref src="myvideo_cable.flv" dur="3:00.1"/>
</switch>
```



您可以使用焦点管理器类来指定一个顺序，当用户按 **Tab** 键在应用程序中导航时，组件将按此顺序接收焦点。您还可以使用焦点管理器在文档中设置一个按钮，当用户按 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时，该按钮会接收键盘输入。例如，当用户填完一张表单后，他们应该能够使用 **Tab** 键在字段之间切换，然后按 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 来提交表单。

所有组件都实现了对焦点管理器的支持；您无需编写代码来调用 **FocusManager** 类。

提醒

对焦点管理器的支持将覆盖 `on(keyPress)` 全局处理函数的使用。由于所有组件都实现了焦点管理器，因此对于包含组件和使用 `on(keyPress)` 全局处理函数的应用程序，需要为每个控件（包括组件 *和* 影片剪辑）显式设置一个 `tabIndex`（请参见第 664 页的“使用焦点管理器”）。或者，您可以为特定的键添加事件侦听器，这样更为适宜，并且焦点管理器将不会覆盖相应的事件处理函数。有关为键创建事件侦听器的更多信息，请参见《学习 Flash 中的 ActionScript 2.0》中的“捕获按键”。

焦点管理器也会与系统管理器进行交互，当激活或取消激活弹出窗口时，系统管理器会激活或取消激活 **FocusManager** 实例。每个模式窗口都有一个 **FocusManager** 实例，所以，该窗口中的组件就成为了它们自己的 **Tab** 集，这样就可以防止用户按 **Tab** 键切换到其它窗口中的组件。

焦点管理器可识别单选按钮组（这些按钮具有已定义的 `RadioButton.groupName` 属性），并将焦点设置到该组中 `selected` 属性设置为 `true` 的实例。当按 **Tab** 键时，焦点管理器会查看下一个对象是否与当前对象具有相同的组名。如果是这样，那么它会自动将焦点移动到下一个具有不同组名的对象。其它支持 `groupName` 属性的组件组也可以使用这一功能。

焦点管理器处理由于鼠标单击造成的焦点更改。如果用户单击一个组件，则该组件就被赋予焦点。

提醒

使用焦点管理器（“控制”>“测试影片”）测试脚本时，请在测试模式中选择“控制”>“禁用快捷键”；否则，焦点管理器不起作用。此外，默认情况下，创作环境中还使用 **Tab** 键和快捷键。因此，如果使用测试模式，**Tab** 键导航、**Enter** 键和其它键的组合可能会意外执行或者失败。应当在创作环境外部的播放器中测试这些功能。

# 使用焦点管理器

焦点管理器不会自动给组件分配焦点。如果想要某个组件在加载应用程序时具有焦点，必须编写一个对组件调用 `FocusManager.setFocus()` 的脚本。



如果在加载应用程序时调用 `FocusManager.setFocus()` 为组件设置焦点，则焦点环不会出现在该组件的周围。组件具有焦点，但不存在指示器。

若要在应用程序中创建焦点导航，请在应接收焦点的任何对象（包括按钮）上设置 `tabIndex` 属性。当用户按下 **Tab** 键时，焦点管理器就会查找一个已启用对象，而且此对象具有比 `tabIndex` 当前值更高的 `tabIndex` 属性。焦点管理器达到 `tabIndex` 属性的最高值后，它就会返回到零。因此，在以下示例中，首先由 `comment` 对象（很可能是 `TextArea` 组件）接收焦点，然后由 `okButton` 对象接收焦点：

```
comment.tabIndex = 1;
okButton.tabIndex = 2;
```

您还可以使用“辅助功能”面板来分配 **Tab** 键索引值。

如果舞台上的任何内容都不具有 **Tab** 键索引值，则焦点管理器使用 `depth`（堆叠顺序，或 `z` 顺序）。深度最初按组件拖到舞台上的顺序设置；不过，您也可以使用“修改”>“排列”>“移至顶层”或“移至底层”命令来确定最终的深度。

若要创建一个当用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时接收焦点的按钮，可将 `FocusManager.defaultPushButton` 属性设置为所需按钮的实例名称，如下所示：

```
focusManager.defaultPushButton = okButton;
```



焦点管理器与对象放在舞台上的时间（对象的深度顺序）有关，而与它们在舞台上的相对位置无关。这与 Flash Player 处理 **Tab** 键排序的方式不同。

# 使用焦点管理器允许 Tab 键切换

您可以使用焦点管理器创建一个方案，该方案允许用户通过按 **Tab** 键，在 **Flash** 应用程序的对象之间循环切换。（**Tab** 键方案中的对象称为“**Tab 目标**”。）焦点管理器检查对象父级的 `tabEnabled` 和 `tabChildren` 属性，以便找到对象。

影片剪辑可以是 **Tab** 目标的容器或 **Tab** 目标本身，也可以两者都不是：

影片剪辑类型	tabEnabled	tabChildren
Tab 目标的容器	false	true
Tab 目标	true	false
两者均不是	false	false

提醒

这与默认的 **Flash Player** 行为不同，在默认行为中，容器的 `tabChildren` 属性可以是 `undefined`。

请考虑以下情况。主时间轴的舞台上有两个文本字段 (`txt1` 和 `txt2`) 和一个影片剪辑 (`mc`)，该影片剪辑包含一个 **DataGrid** 组件 (`grid1`) 和另一个文本字段 (`txt3`)。您将使用下面的代码，允许用户按 **Tab** 键，以如下顺序在对象之间循环切换：`txt1`、`txt2`、`grid1`、`txt3`。

提醒

默认情况下，会启用 **FocusManager** 和 **TextField** 实例。

```
// 让焦点管理器知道 mc 具有子项；
// 这会覆盖 mc.focusEnabled=true；
mc.tabChildren=true；
mc.tabEnabled=false；
// 设置 Tab 键切换顺序。
txt1.tabIndex = 1；
txt2.tabIndex = 2；
mc.grid1.tabIndex = 3；
mc.txt3.tabIndex = 4；

// 将初始焦点设置为 txt1。
txt1.text = "focus"；
focusManager.setFocus(txt1)；
```

如果影片剪辑没有 `onPress` 或 `onRelease` 方法，或者没有 `tabEnabled` 属性，则除非您将 `focusEnabled` 设置为 `true`，否则焦点管理器看不到它。除非禁用，否则输入文本字段始终在 **Tab** 方案中。

如果 **Flash** 应用程序正在 **Web** 浏览器中播放，则在用户单击应用程序中某处之前，该应用程序都不具有焦点。另外，一旦用户单击 **Flash** 应用程序，那么按 **Tab** 键可以导致焦点跳到 **Flash** 应用程序之外。若要将 **Tab** 键的切换范围限制在 **Flash Player 7 ActiveX** 控件的 **Flash** 应用程序内，请将以下参数添加到 **HTML** `<object>` 标记中：

```
<param name="SeamlessTabbing" value="false"/>
```

## 创建具有焦点管理器的应用程序

以下步骤会在 Flash 应用程序中创建一个焦点方案。

### 创建焦点方案：

1. 将 **TextInput** 组件从“组件”面板拖到舞台中。
2. 在“属性”检查器中，为它分配实例名称 **comment**。
3. 将 **Button** 组件从“组件”面板拖到舞台中。
4. 在“属性”检查器中，为它分配实例名称 **okButton**，并将标签参数设置为 **OK**。
5. 在“动作”面板的第 1 帧中，输入下列代码：

```
comment.tabIndex = 1;
okButton.tabIndex = 2;
focusManager.setFocus(comment);
function click(evt){
 trace(evt.type);
}
okButton.addEventListener("click", this);
```

6. 选择“控制” > “测试影片”。
7. 选择“控制” > “禁用快捷键”。

此代码设置 **Tab** 键的切换顺序。尽管注释字段没有初始焦点环，但它具有初始焦点，因此您无需单击注释字段，就能在其中开始键入。此外，您必须选择“禁用快捷键”菜单选项才能使焦点在测试模式下正常工作。

## 自定义焦点管理器

通过更改 **themeColor** 样式的值，可以更改光晕主题中焦点环的颜色，如此示例所示：

```
_global.style.setStyle("themeColor", "haloBlue");
```

焦点管理器使用 **FocusRect** 外观来绘制焦点。可以替换或修改此外观，子类也可以覆盖 **UIComponent.drawFocus** 以绘制自定义的焦点指示符。

# FocusManager 类 (API)

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > FocusManager

ActionScript 类名称 mx.managers.FocusManager

您可以使用焦点管理器来指定一个顺序，当用户按 **Tab** 键在应用程序中定位时，组件将按此顺序接收焦点。您还可以使用 **FocusManager** 类在文档中设置一个按钮，当用户按 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时，该按钮会接收键盘输入。

提示

在继承自 **UIComponent** 的类文件中，不建议引用 `_root.focusManager`。每个 **UIComponent** 实例都继承一个 `getFocusManager()` 方法，该方法返回对 **FocusManager** 实例的引用，此实例负责控制组件的焦点方案。

## FocusManager 类的方法摘要

下表列出了 **FocusManager** 类的方法。

方法	说明
<code>FocusManager.getFocus()</code>	返回对具有焦点的对象的引用。
<code>FocusManager.sendDefaultPushButtonEvent()</code>	给注册到默认普通按钮的侦听器对象发送一个 <code>click</code> 事件。
<code>FocusManager.setFocus()</code>	给指定对象设置焦点。

## 从 UIObject 类继承的方法

下表列出了 **FocusManager** 类从 **UIObject** 类继承的方法。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。

方法	说明
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 `FocusManager` 类从 `UIComponent` 类继承的方法。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## FocusManager 类的属性摘要

下表列出了 `FocusManager` 类的属性。

属性	说明
<code>FocusManager.defaultPushButton</code>	一个对象，该对象在用户按 <code>Return</code> 键或 <code>Enter</code> 键时接收 <code>click</code> 事件。
<code>FocusManager.defaultPushButtonEnabled</code>	指示是启用 ( <code>true</code> ) 还是禁用 ( <code>false</code> ) 默认普通按钮的键盘处理功能。默认值为 <code>true</code> 。
<code>FocusManager.enabled</code>	指示是启用 ( <code>true</code> ) 还是禁用 ( <code>false</code> ) <code>Tab</code> 键的处理功能。默认值为 <code>true</code> 。
<code>FocusManager.nextTabIndex</code>	<code>tabIndex</code> 属性的下一个值。

## 从 UIObject 类继承的属性

下表列出了 `FocusManager` 类从 `UIObject` 类继承的属性。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 <code>x</code> 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 <code>y</code> 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。



属性	说明
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 `UIComponent` 类继承的属性

下表列出了 `FocusManager` 类从 `UIComponent` 类继承的属性。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## FocusManager 类的事件摘要

没有 `FocusManager` 类专用的事件。

## 从 `UIObject` 类继承的事件

下表列出了 `FocusManager` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 FocusManager 类从 UIComponent 类继承的事件。

事件	说明
<a href="#">UIComponent.focusIn</a>	当对象收到焦点时进行广播。
<a href="#">UIComponent.focusOut</a>	当对象失去焦点时进行广播。
<a href="#">UIComponent.keyDown</a>	当按下按键时进行广播。
<a href="#">UIComponent.keyUp</a>	当松开按键时进行广播。

# FocusManager.defaultPushButton

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004 和 Flash MX Professional 2004。

### 用法

`focusManager.defaultPushButton`

### 说明

属性：为应用程序指定默认普通按钮。当用户按 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时，默认普通按钮的侦听器会接收到一个 `click` 事件。默认值为 `undefined`，而且该属性的数据类型是对象。

焦点管理器使用 **SimpleButton** 类的强调样式声明以可视方式指示当前的默认普通按钮。

`defaultPushButton` 属性的值始终是具有焦点的按钮。设置 `defaultPushButton` 属性无法将初始焦点指定给默认的普通按钮。如果应用程序中有多个按钮，则在按下 **Enter** 键或 **Return** 键时，当前具有焦点的按钮将接收 `click` 事件。如果在按下 **Enter** 键或 **Return** 键时某个其它组件具有焦点，则将 `defaultPushButton` 属性重置为其原始值。

### 示例

以下代码将默认普通按钮设置为 `OKButton` 实例：

```
focusManager.defaultPushButton = OKButton;
```

### 另请参见

[FocusManager.defaultPushButtonEnabled](#)、  
[FocusManager.sendDefaultPushButtonEvent\(\)](#)

# FocusManager.defaultPushButtonEnabled

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`focusManager.defaultPushButtonEnabled`

## 说明

属性：一个布尔值，它确定是 (true) 否 (false) 启用默认普通按钮的键盘处理功能。如果将 `defaultPushButtonEnabled` 设置为 `false`，则组件可以接收 **Return** 键或 **Enter** 键，并在内部对其进行处理。必须通过监视组件的 `onKillFocus()` 方法（请参见《ActionScript 2.0 语言参考》中的 **onKillFocus (MovieClip.onKillFocus handler)**）或 `focusOut` 事件重新启用默认普通按钮处理功能。默认值为 `true`。

此属性供高级组件开发人员使用。

## 示例

下列代码会禁用默认普通按钮处理功能：

```
focusManager.defaultPushButtonEnabled = false;
```

# FocusManager.enabled

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`focusManager.enabled`

### 说明

属性：一个布尔值，它确定是 (true) 否 (false) 为特定的一组焦点对象启用 **Tab** 键处理功能。（例如，其它弹出窗口可能具有它自己的焦点管理器。）如果将 `enabled` 设置为 `false`，则组件可以接收 **Tab** 处理按键，并在内部对它们进行处理。必须通过监视组件的 `onKillFocus()` 方法（请参见《ActionScript 2.0 语言参考》中的 `onKillFocus` (MovieClip.onKillFocus handler)）或 `focusOut` 事件重新启用焦点管理器处理功能。默认值为 `true`。

### 示例

下面的代码禁用 **Tab** 键处理功能：

```
focusManager.enabled = false;
```

## FocusManager.setFocus()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004 和 Flash MX Professional 2004。

### 用法

```
focusManager.setFocus()
```

### 参数

无。

### 返回

对具有焦点的对象的引用。

### 说明

方法：返回对当前具有焦点的对象的引用。

### 示例

如果当前具有焦点的对象是 `myInputText`，则以下代码会将焦点设置到 `myOKButton`：

```
if (focusManager.setFocus() == myInputText)
{
 focusManager.setFocus(myOKButton);
}
```

### 另请参见

[FocusManager.setFocus\(\)](#)

# FocusManager.nextTabIndex

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`FocusManager.nextTabIndex`

## 说明

属性；下一个可用的 **Tab** 键索引号。该属性用于动态地设置对象的 `tabIndex` 属性。

## 示例

以下代码为 `mycheckbox` 实例赋予下一个最高的 `tabIndex` 值：

```
mycheckbox.tabIndex = focusManager.nextTabIndex;
```

## 另请参见

[UIComponent.tabIndex](#)

# FocusManager.sendDefaultPushButtonEvent()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

`focusManager.sendDefaultPushButtonEvent()`

## 参数

无。

## 返回

无。

## 说明

方法：向注册到默认普通按钮的侦听器对象发送一个 `click` 事件。使用该方法可以用编程方式发送 `click` 事件。

## 示例

以下代码在用户选择 **CheckBox** 实例 `chb`（复选框将标记为“Automatic Login”）时触发默认普通按钮的 `click` 事件，并填写用户名和密码字段：

```
name_txt.tabIndex = 1;
password_txt.tabIndex = 2;
chb.tabIndex = 3;
submit_ib.tabIndex = 4;

focusManager.defaultPushButton = submit_ib;

chbObj = new Object();
chbObj.click = function(o){
 if (chb.selected == true){
 name_txt.text = "Jody";
 password_txt.text = "foobar";
 focusManager.sendDefaultPushButtonEvent();
 } else {
 name_txt.text = "";
 password_txt.text = "";
 }
}
chb.addEventListener("click", chbObj);

submitObj = new Object();
submitObj.click = function(o){
 if (password_txt.text != "foobar"){
 trace("error on submit");
 } else {
 trace("Yeah! sendDefaultPushButtonEvent worked!");
 }
}
submit_ib.addEventListener("click", submitObj);
```

## 另请参见

[FocusManager.defaultPushButton](#)

# FocusManager.setFocus()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

```
focusManager.setFocus(object)
```

## 参数

*object* 对要接收焦点的对象的引用。

## 返回

无。

## 说明

方法：将焦点设置为指定的对象。如果您想设置焦点的对象不在主时间轴上，请使用以下代码：

```
_root.focusManager.setFocus(object);
```

## 示例

以下代码将焦点设置到 myOKButton：

```
focusManager.setFocus(myOKButton);
```

## 另请参见

[FocusManager.setFocus\(\)](#)



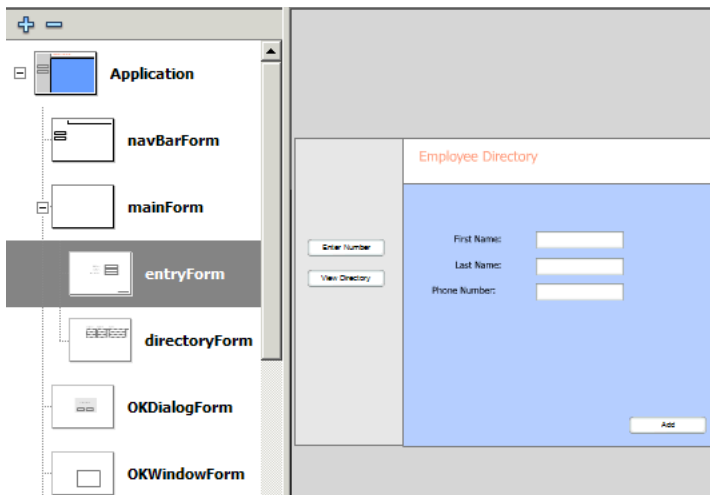


# Form 类（仅限 Flash Professional）

Form 类提供您在 Flash 的“屏幕轮廓”窗格中所创建表单的运行时行为。有关使用屏幕的概述，请参见《使用 Flash》中的“使用屏幕（仅限 Flash Professional）”。

## 使用 Form 类（仅限 Flash Professional）

表单用作图形对象（例如，应用程序中的用户界面元素）和应用程序状态的容器。可以使用“屏幕大纲”窗格使您正在创建的应用程序的不同状态可视化，其中每个表单都是不同的应用程序状态。例如，以下图示显示了使用表单设计的应用程序范例的“屏幕大纲”窗格。



表单应用程序范例的“屏幕大纲”视图

此图示显示名为“Employee Directory”的应用程序范例的轮廓，其中包含一些表单。名为“entryForm”的表单（在上面的图示中处于选中状态）包含一些用户界面对象，包括输入文本字段、标签和普通按钮。通过切换此表单的可见性（使用 `Form.visible` 属性），同时也切换其它表单的可见性，开发人员可以轻松地为用户显示此表单。

通过使用“行为”面板，还可以将行为和控件附加到表单上。有关向屏幕添加过渡和控件的更多信息，请参见《使用 Flash》中的“使用行为为屏幕创建控件和过渡（仅限于 Flash Professional）”。

因为 Form 类扩展了 Loader 类，所以可以轻松地将外部内容（SWF 或 JPEG 文件）加载到表单中。例如，表单的内容可以是单独的 SWF 文件，其本身可能包含表单。通过此方法，可以使表单应用程序模块化，从而能更轻松地维护应用程序，同时还降低了最初的下载时间。有关更多信息，请参见第 992 页的“将外部内容加载到屏幕（仅限 Flash Professional）”。

## Form 的参数

您可以在“属性”检查器或“组件”检查器中，为每个 Form 实例设置以下创作参数：

**autoload** 指示 **contentPath** 参数所指定的内容是应该自动加载 (true)，还是应该等到调用 `Loader.load()` 方法时再加载 (false)。默认值为 true。

**contentPath** 指定表单的内容。该参数可以是一个影片剪辑的链接标识符，也可以是要加载到幻灯片中的 SWF 或 JPEG 文件的绝对或相对 URL。默认情况下，加载的内容会进行剪辑以适合幻灯片的大小。

**visible** 指定表单在第一次加载时是 (true) 否 (false) 可见。

## Form 类（仅限 Flash Professional）

继承 MovieClip > UIObject 类 > UIComponent 类 > View > Loader 组件 > Screen 类（仅限 Flash Professional）> Form

ActionScript 类名称 mx.screens.Form

Form 类提供您在 Flash 的“屏幕轮廓”窗格中所创建表单的运行时行为。

## Form 类的方法摘要

下表列出了 Form 类的方法。

方法	说明
<code>Form.getChildForm()</code>	返回指定索引处的子表单。

## 从 UIObject 类继承的方法

下表列出了 **Form** 类从 **UIObject** 类继承的方法。当从 **Form** 对象调用这些方法时，请使用语法 *formInstance.methodName*。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **Form** 类从 **UIComponent** 类继承的方法。当从 **Form** 对象调用这些方法时，请使用语法 *formInstance.methodName*。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## 从 Loader 类继承的方法

下表列出了 **Form** 类从 **Loader** 类继承的方法。当从 **Form** 对象调用这些方法时，请使用语法 *formInstance.methodName*。

方法	说明
<code>Loader.load()</code>	加载由 <code>contentPath</code> 属性指定的内容。

## 从 Screen 类继承的方法

下表列出了 **Form** 类从 **Screen** 类继承的方法。当从 **Form** 对象调用这些方法时，请使用语法 *formInstance.methodName*。

方法	说明
<code>Screen.getChildScreen()</code>	返回此屏幕位于特定索引处的子屏幕。

## Form 类的属性摘要

下表列出了 **Form** 类专用的属性。

属性	说明
<code>Form.currentFocusedForm</code>	只读；返回包含全局当前焦点的表单。
<code>Form.indexInParentForm</code>	只读；返回此表单在其父表单的子表单列表中的索引（从零开始）。
<code>Form.numChildForms</code>	只读；返回此表单包含的子表单的数量。
<code>Form.parentIsForm</code>	只读；指定此表单的父对象是否也是表单。
<code>Form.parentForm</code>	只读；对表单的父表单的引用。
<code>Form.rootForm</code>	只读；返回包含此表单的表单树（或子树）的根。
<code>Form.visible</code>	指定表单在其父表单、幻灯片、影片剪辑或 SWF 文件可见时是否可见。

## 从 UIObject 类继承的属性

下表列出了 **Form** 类从 **UIObject** 类继承的属性。当从 **Form** 对象访问这些属性时，请使用语法 *formInstance.propertyName*。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。

属性	说明
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 **Form** 类从 **UIComponent** 类继承的属性。当从 **Form** 对象访问这些属性时，请使用语法 `formInstance.propertyName`。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## 从 Loader 类继承的属性

下表列出了 **Form** 类从 **Loader** 类继承的属性。当从 **Form** 对象访问这些属性时，请使用语法 `formInstance.propertyName`。

属性	说明
<code>Loader.autoLoad</code>	一个布尔值，它指示内容是自动加载 ( <code>true</code> ) 还是必须调用 <code>Loader.load()</code> 才能加载 ( <code>false</code> )。
<code>Loader.bytesLoaded</code>	只读属性，指明已经加载的字节数。
<code>Loader.bytesTotal</code>	指明内容中的总字节数的只读属性。
<code>Loader.content</code>	对加载器内容的引用。该属性为只读。
<code>Loader.contentPath</code>	一个字符串，它指明要加载的内容的 URL。
<code>Loader.percentLoaded</code>	一个数字，它指明已加载内容的百分比。该属性为只读。
<code>Loader.scaleContent</code>	一个布尔值，它指示是内容进行缩放以适合加载器 ( <code>true</code> )，还是加载器进行缩放以适合内容 ( <code>false</code> )。

## 从 Screen 类继承的属性

下表列出了 **Form** 类从 **Screen** 类继承的属性。当从 **Form** 对象访问这些属性时，请使用语法 *formInstance.propertyName*。

属性	说明
<a href="#">Screen.currentFocusedScreen</a>	只读；返回包含全局当前焦点的屏幕。
<a href="#">Screen.indexInParent</a>	只读；返回该屏幕在其父屏幕的子屏幕列表中的索引（从零开始）。
<a href="#">Screen.numChildScreens</a>	只读；返回屏幕包含的子屏幕的数量。
<a href="#">Screen.parentIsScreen</a>	只读；返回一个布尔值（true 或 false），该值指示屏幕的父对象本身是否是屏幕。
<a href="#">Screen.rootScreen</a>	只读；返回包含此屏幕的树或子树的根屏幕。

## Form 类的事件摘要

没有 **Form** 类专用的事件。

## 从 UIObject 类继承的事件

下表列出了 **Form** 类从 **UIObject** 类继承的事件。

事件	说明
<a href="#">UIObject.draw</a>	当对象将要绘制它的图形时进行广播。
<a href="#">UIObject.hide</a>	在对象的状态从可见变为不可见时广播。
<a href="#">UIObject.load</a>	创建子对象时广播。
<a href="#">UIObject.move</a>	移动了对象时广播。
<a href="#">UIObject.resize</a>	在调整对象大小后广播。
<a href="#">UIObject.reveal</a>	在对象的状态从不可见变为可见时广播。
<a href="#">UIObject.unload</a>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 Form 类从 UIComponent 类继承的事件。

事件	说明
<a href="#">UIComponent.focusIn</a>	当对象收到焦点时进行广播。
<a href="#">UIComponent.focusOut</a>	当对象失去焦点时进行广播。
<a href="#">UIComponent.keyDown</a>	当按下按键时进行广播。
<a href="#">UIComponent.keyUp</a>	当松开按键时进行广播。

## 从 Loader 类继承的事件

下表列出了 Form 类从 Loader 类继承的事件。

事件	说明
<a href="#">Loader.complete</a>	当内容加载完成时触发。
<a href="#">Loader.progress</a>	在内容加载过程中触发。

## 从 Screen 类继承的事件

下表列出了 Form 类从 Screen 类继承的事件。

事件	说明
<a href="#">Screen.allTransitionsInDone</a>	当应用到屏幕的所有“输入”过渡结束时进行广播。
<a href="#">Screen.allTransitionsOutDone</a>	当应用到屏幕的所有“输出”过渡结束时进行广播。
<a href="#">Screen.mouseDown</a>	当在直接属于屏幕的对象（形状或影片剪辑）上按下鼠标按钮时进行广播。
<a href="#">Screen.mouseDownSomewhere</a>	当在舞台上的某处（不一定要在属于此屏幕的对象上）按下鼠标按钮时进行广播。
<a href="#">Screen.mouseMove</a>	当鼠标在屏幕上移动时进行广播。
<a href="#">Screen.mouseOut</a>	当鼠标从屏幕内移出时进行广播。
<a href="#">Screen.mouseOver</a>	当鼠标从屏幕外移入时进行广播。
<a href="#">Screen.mouseUp</a>	当在直接属于屏幕的对象（形状或影片剪辑）上松开鼠标按钮时进行广播。
<a href="#">Screen.mouseUpSomewhere</a>	当在舞台上的某处（不一定要在属于此屏幕的对象上）松开鼠标按钮时进行广播。

# Form.currentFocusedForm

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mx.screens.Form.currentFocusedForm
```

## 说明

属性（只读）：返回包含全局当前焦点的表单对象。实际的焦点可能在表单自身上，或者在影片剪辑、文本对象或者该表单内的组件上。如果没有当前焦点，则可以为 `null`。

## 示例

以下代码附加到一个按钮（未显示），并显示具有当前焦点的表单的名称。

```
trace("The form with the current focus is: " +
 mx.screens.Form.currentFocusedForm);
```

# Form.getChildForm()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myForm.getChildForm(childIndex)
```

## 参数

*childIndex* 一个数字，它指示要返回的子表单的索引（从零开始）。

## 返回

一个表单对象。

## 说明

方法：返回 *myForm* 的子表单（其索引为 *childIndex*）。



## 示例

以下示例将在“输出”面板中显示属于名为 `application` 的根 `Form` 对象的所有子 `Form` 对象的名称。

```
for (var i:Number = 0; i < _root.application.numChildForms; i++) {
 var childForm:mx.screens.Form = _root.application.getChildForm(i);
 trace(childForm._name);
}
```

## 另请参见

[Form.numChildForms](#)

# Form.indexInParentForm

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myForm*.indexInParentForm

## 说明

属性（只读）：包含 *myForm* 在其父表单的子表单列表中从零开始的索引。如果 *myForm* 的父对象是屏幕而不是表单（例如，是幻灯片），则 `indexInParentForm` 始终为 `0`。

## 示例

```
var myIndex:Number = myForm.indexInParent;
if (myForm == myForm._parent.getChildForm(myIndex)) {
 trace("I'm where I should be");
}
```

## 另请参见

[Form.getChildForm\(\)](#)

# Form.numChildForms

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myForm*.numChildForms

## 说明

属性（只读）：从 **mx.screens.Form** 类直接派生的 *myForm* 所包含的子表单的数量。此属性不包括 *myForm* 所包含的任何幻灯片；只包括表单。



当使用扩展 **mx.screens.Form** 的自定义 ActionScript 2.0 类时，该表单不计入 numChildForms 属性。

## 示例

以下代码遍历 *myForm* 中包含的所有子表单，并在“输出”面板中显示它们的名称。

```
var howManyKids:Number = myForm.numChildForms;
for(i=0; i<howManyKids; i++) {
 var childForm = myForm.getChildForm(i);
 trace(childForm._name);
}
```

## 另请参见

[Form.getChildForm\(\)](#)

# Form.parentIsForm

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myForm*.parentIsForm

## 说明

属性（只读）：返回一个布尔值，指示所指定表单的父对象是 (true) 否 (false) 也是表单。如果此属性为 false，则 *myForm* 位于其表单层次结构的根位置。

## 示例

```
if (myForm.parentIsForm) {
 trace("I have "+myForm._parent.numChildScreens+" sibling screens");
} else {
 trace("I am the root form and have no siblings");
}
```

# Form.parentForm

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myForm*.parentForm

## 说明

属性（只读）：对表单的父表单的引用。

## 示例

以下示例代码位于名为 *myForm* 的屏幕上，该屏幕是您从“新建文档”对话框中选择“Flash 表单应用程序”时创建的默认 *form1* 屏幕的子屏幕。

```
onClipEvent(keyDown){
 var parentForm:mx.screens.Form = this.parentForm;
 trace(parentForm);
}
// 输出: _level0.application.form1
```

# Form.rootForm

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myForm.rootForm*

## 说明

属性（只读）：返回表单层次结构顶层中包含 *myForm* 的表单。如果 *myForm* 包含于非表单的对象（如幻灯片）中，则此属性会返回 *myForm*。

## 示例

在以下示例中，将一个对 *myForm* 的根表单的引用放入名为 *root* 的变量中。如果分配给 *root* 的值引用 *myForm*，则 *myForm* 位于其表单树的顶层。

```
var root:mx.screens.Form = myForm.rootForm;
if(root == myForm) {
 trace("myForm is the top form in its tree");
}
```

# Form.visible

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myForm.visible*

## 说明

属性：确定 *myForm* 在其父表单、幻灯片、影片剪辑或 SWF 文件可见时是否可见。也可以使用 Flash 创作环境中的“属性”检查器设置此属性。

当此属性设置为 `true` 时, *myForm* 将接收到 `reveal` 事件; 当设置为 `false` 时, *myForm* 将接收到 `hide` 事件。您可以将在表单接收到上述一种事件时执行的过渡附加到表单上。有关向屏幕添加过渡的更多信息, 请参见《使用 Flash》中的“使用行为为屏幕创建控件和过渡 (仅限于 Flash Professional)”。

### 示例

以下代码 (位于时间轴帧上) 将包含该按钮的表单的 `visible` 属性设置为 `false`。

```
btn0k.addEventListener("click", btn0kClick);
function btn0kClick(eventObj:Object):Void {
 eventObj.target._parent.visible = false;
}
```



# Iterator 接口（仅限 Flash Professional）

ActionScript 类名称 `mx.utils.Iterator`

Iterator 接口使您能够遍历集合中包含的对象。

## Iterator 接口的方法摘要

下表列出了 Iterator 接口的方法。

方法	说明
<code>Iterator.hasNext()</code>	指示重复值是否具有其它项目。
<code>Iterator.next()</code>	返回重复值的下一个项目。

## Iterator.hasNext()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004.

### 用法

```
iterator.hasNext()
```

### 返回

一个布尔值，指示重复值中是 (true) 否 (false) 有其它实例。

### 说明

方法；指示重复值中是否存在其它实例。此方法通常在遍历重复值时用于 while 语句。

## 示例

以下示例使用 `hasNext()` 方法来控制对集合中项的重复值的遍历：

```
on (click) {
 var myColl:mx.utils.Collection;
 myColl = _parent.thisShelf.MyCompactDisks;
 var itr:mx.utils.Iterator = myColl.getIterator();
 while (itr.hasNext()) {
 var cd:CompactDisk = CompactDisk(itr.next());
 var title:String = cd.Title;
 var artist:String = cd.Artist;
 trace("Title: "+title+" Artist: "+artist);
 }
}
```

# Iterator.next()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004.

## 用法

```
iterator.next()
```

## 返回

充当重复值中下一项的对象。

## 说明

方法：返回重复值中下一项的实例。必须将此实例转换为正确的类型。

## 示例

以下示例使用 `next()` 方法来访问集合中的下一项：

```
on (click) {
 var myColl:mx.utils.Collection;
 myColl = _parent.thisShelf.MyCompactDisks;
 var itr:mx.utils.Iterator = myColl.getIterator();
 while (itr.hasNext()) {
 var cd:CompactDisk = CompactDisk(itr.next());
 var title:String = cd.Title;
 var artist:String = cd.Artist;
 trace("Title: "+title+" Artist: "+artist);
 }
}
```



一个 **Label** 组件就是一行文本。您可以指定一个标签采用 **HTML** 格式。您也可以控制标签的对齐和大小。**Label** 组件没有边框、不能具有焦点，并且不广播任何事件。

每个 **Label** 实例的实时预览反映了创作时在“属性”检查器中或在“组件”检查器中对参数所做的更改。标签没有边框，因此，查看它的实时预览的唯一方法就是设置其 **text** 参数。实时预览不支持 **autoSize** 参数。

将 **Label** 组件添加到应用程序时，可以使用“辅助功能”面板使其可以由屏幕阅读器进行访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.LabelAccImpl.enableAccessibility();
```

无论一个组件有多少实例，都只对它启用一次辅助功能。有关更多信息，请参见《使用 **Flash**》中的第 19 章“创建辅助内容”。

## 使用 Label 组件

**Label** 组件用于为表单中的另一个组件创建文本标签，例如，**TextInput** 字段左侧的“姓名：”标签接受用户的姓名。如果您要使用基于 **Macromedia Component Architecture** 第 2 版的组件构建一个应用程序，那么，使用 **Label** 组件来替代普通文本字段就是一个好方法，因为您可以使用样式来维持一致的外观。

如果要旋转 **Label** 组件，必须嵌入字体。请参见第 695 页的“对 **Label** 组件使用样式”。

## Label 参数

在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中，可以为每个 **Label** 组件实例设置以下创作参数：

**autoSize** 指示如何调整标签的大小并对齐标签以适合文本。默认值为 `none`。参数可以是以下四个值之一：

- `none`，指定不调整标签大小或对齐标签来适合文本。
- `left`，指定调整标签的右边和底边的大小以适合文本。不会调整左边和上边的大小。
- `center`，指定调整标签左边和右边的大小以适合文本。标签的水平中心锚定在它原始的水平中心位置。
- `right`，指定调整标签左边和底边的大小以适合文本。不会调整上边和右边的大小。



Label 组件的 `autoSize` 属性与内置 `ActionScript TextField` 对象的 `autoSize` 属性不同。

**html** 指示标签是 (`true`) 否 (`false`) 采用 **HTML** 格式。如果此参数设置为 `true`，则不能使用样式来设置标签的格式，但您可以使用 `font` 标记将文本格式设置为 **HTML**。默认值为 `false`。

**text** 指示标签的文本；默认值是 `Label`。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **Label** 组件实例设置以下附加参数：

**visible** 是一个布尔值，它指示对象是 (`true`) 否 (`false`) 可见。默认值为 `true`。



`minHeight` 和 `minWidth` 属性由内部的大小调整例程使用。它们在 `UIObject` 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以使用 **Label** 实例的方法、属性和事件为其编写 **ActionScript** 来设置其它选项。有关更多信息，请参见第 696 页的“**Label** 类”。

## 创建具有 Label 组件的应用程序

以下过程解释了如何在创作时将 **Label** 组件添加到应用程序。在本例中，标签位于购物车应用程序中某个带有日期的组合框的旁边。

### 创建具有 Label 组件的应用程序：

1. 将 **Label** 组件从“组件”面板拖至舞台。
2. 在“组件”检查器中，为标签参数输入 **Expiration Date**（过期日期）。

### 使用 ActionScript 创建 Label 组件实例：

1. 将 **Label** 组件从“组件”面板拖到当前文档的库中。

此操作将组件添加到库中，但不会在应用程序中显示。

2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
this.createClassObject(mx.controls.Label, "my_label", 1);
my_label.text = "Hello World";
```

此脚本使用 `UIObject.createClassObject()` 方法创建 **Label** 实例。

3. 选择“控制” > “测试影片”。

## 自定义 Label 组件

在创作过程中和运行时，可以在水平和垂直方向上改变 **Label** 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。您也可以设置 `autoSize` 创作参数；设置此参数不会改变实时预览中的边框，但是会调整标签的大小。有关更多信息，请参见第 694 页的“**Label 参数**”。在运行时，则使用 `setSize()` 方法（请参见 `UIObject.setSize()` 或 `Label.autoSize`）。

## 对 Label 组件使用样式

您可以设置样式属性来更改标签实例的外观。**Label** 组件实例中的所有文本必须采用相同的样式。例如，对同一标签内的单词设置 `color` 样式时，不能将一个单词设置为“blue”，而将另一个单词设置为“red”。

如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关样式的更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

Label 组件支持下列样式：

样式	主题	说明
color	光晕和范例	文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。
disabledColor	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。
embedFonts	光晕和范例	一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。
fontFamily	光晕和范例	文本的字体名称。默认值为 “_sans”。
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式：“normal”或“italic”。默认值为 “normal”。
fontWeight	光晕和范例	字体粗细：“none”或“bold”。默认值为 “none”。在调用 setStyle() 期间，所有组件还可以接受值 “normal”来代替 “none”，但随后对 getStyle() 的调用将返回 “none”。
textAlign	光晕和范例	文本对齐方式：“left”、“right”或“center”。默认值为 “left”。
textDecoration	光晕和范例	文本修饰：“none”或“underline”。默认值为 “none”。

## 对 Label 组件使用外观

Label 组件没有任何可改变外观的可视元素。

# Label 类

继承 MovieClip > [UIObject 类](#) > Label

ActionScript 类名称 mx.controls.Label

Label 类的属性允许您在运行时为标签指定文本，指明文本是否可采用 HTML 格式，以及标签是否自动调整大小以适应文本。


使用 ActionScript 设置 Label 类的属性会覆盖在“属性”检查器中或“组件”检查器中设置的同名参数。

当您访问标签属性的值时，请确保在您尝试访问所需属性之前组件已完成加载。请考虑以下示例：

```
var listenerObject:Object = new Object();
listenerObject.load = function(){
 trace(label.width);
};
label.addEventListener("load", listenerObject);
```

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.controls.Label.version);
```



代码 `trace(myLabelInstance.version)`；返回 `undefined`。

## Label 类的方法摘要

没有 `Label` 类专用的方法。

### 从 `UIObject` 类继承的方法

下表列出了 `Label` 类从 `UIObject` 类继承的方法。从 `Label` 对象调用这些方法时，请使用 `labelInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

# Label 类的属性摘要

下表列出了 Label 类的属性。

属性	说明
<code>Label.autoSize</code>	一个字符串，指示如何调整标签大小和对齐标签以适合其 <code>text</code> 属性的值。有四种可能的值: "none"、"left"、"center" 和 "right"。默认值为 "none"。
<code>Label.html</code>	一个布尔值，它指示标签是 (true) 否 (false) 可以采用 HTML 格式。
<code>Label.text</code>	标签上的文本。

## 从 UIObject 类继承的属性

下表列出了 Label 类从 UIObject 类继承的属性。当您访问这些属性时，请使用 `labelInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 <code>x</code> 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 <code>y</code> 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

# Label 类的事件摘要

没有 Label 类专用的事件。

## 从 UIObject 类继承的事件

下表列出了 Label 类从 UIObject 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

# Label.autoSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`labelInstance.autoSize`

## 说明

属性：一个字符串，它指示如何调整标签大小和对齐标签以适合其 `text` 属性的值。有四种可能的值："none"、"left"、"center" 和 "right"。默认值为 "none"。

- `none` 不调整标签大小或对齐标签来适合文本。
- `left` 调整标签右边和底边的大小以适合文本。不会调整左边和上边的大小。
- `center` 调整标签左边和右边的大小以适合文本。标签的水平中心锚定在它原始的水平中心位置。

- right 调整标签左边和底边的大小以适合文本。不会调整上边和右边的大小。



Label 组件的 `autoSize` 属性与内置 `ActionScript TextField` 对象的 `autoSize` 属性不同。

### 示例

在以下示例中，标签实例 `my_label` 将调整标签左边和底边的大小以适合所有文本：

```
my_label.text = "A really long label with Label.autoSize set";
my_label.autoSize = "right";
```

## Label.html

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*labelInstance.html*

### 说明

属性；一个布尔值，它指示标签是 (`true`) 否 (`false`) 可以采用 **HTML** 格式。默认值为 `false`。html 属性设置为 `true` 的 **Label** 组件不能使用样式来设定格式。

若要从 **HTML** 格式的文本获得纯文本，请将 **HTML** 属性设置为 `false`，然后访问 `text` 属性。这将去掉 **HTML** 格式，所以在检索纯文本之前，可能需要将标签文本复制到屏幕以外的 **Label** 或 **TextArea** 组件中。

### 示例

以下示例将 `html` 属性设置为 `true`，这样便可以使用 **HTML** 设置标签的格式。然后，将 `text` 属性设置为一个包含 **HTML** 格式的字符串。

```
my_label.html = true;
my_label.text = "The Royal Nonesuch";
my_label.autoSize = "right";
```

单词 “Royal” 以粗体显示。



# Label.text

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*labelInstance*.text

## 说明

属性；标签的文本。默认值为 "Label"。

## 示例

以下代码设置了 **Label** 实例 my\_label 的 text 属性，并将该值发送到“输出”面板：

```
my_label.text = "The Royal Nonesuch";
trace(my_label.text);
```



**List** 组件是一个可滚动的单选或多选列表框。列表还可显示图形及其它组件。您在单击标签或数据参数字段时，会出现“值”对话框，您可以使用该对话框来添加显示在列表中的项目。您也可以使用 `List.addItem()` 和 `List.addItemAt()` 方法来将项添加到列表。

**List** 组件使用基于零的索引，其中索引为 0 的项目就是显示在顶端的项目。当使用 **List** 类的方法和属性添加、删除或替换列表项时，您可能需要指定该列表项的索引。

在单击列表或按 **Tab** 键切换到列表时，列表获得焦点，然后您可使用以下键控制它：

键	说明
字母数字键	跳转到标签中以 <code>Key.getAscii()</code> 作为首字符的下一项。
Ctrl	允许选择和取消选择多个不相邻的项目的切换键。
向下箭头	选区会向下移动一项。
Home	选区会移动到列表顶端。
Page Down	选区会向下移动一页。
Page Up	选区会向上移动一页。
Shift	允许进行连续选择。
向上箭头	选区会向上移动一项。

提醒

Page Up 键和 Page Down 键使用的页的大小比可以显示的项数少一项。例如，在一个十行的下拉列表中向下翻页，将会依次显示第 0-9 项、第 9-18 项、第 18-27 项等等，每页都会有一个重叠项。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

每个 **List** 实例在舞台上的实时预览反映在创作过程中对属性检查器或组件检查器中的参数所做的更改。

当您将在 **List** 组件添加到应用程序后，就可以使用“辅助功能”面板，使其可由屏幕读取器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.ListAccImpl.enableAccessibility();
```

无论一个组件有多少实例，都只对它启用一次辅助功能。有关更多信息，请参见《使用 **Flash**》中的第 19 章“创建辅助内容”。

## 使用 List 组件

您可以建立一个列表，以使用户可以选择一项或多项。例如，用户访问电子商务 **Web** 站点时需要选择想要购买的项目。一共有 30 个项目，用户在列表中上下滚动，并通过单击选择一项。

您也可以设计一个列表，该列表使用自定义影片剪辑作为行，这样就可以向用户显示更多信息。例如，在电子邮件应用程序中，每个信箱可能就是一个 **List** 组件，而每行可能会有指明优先级和状态的图标。

## 了解 List 组件的设计

当使用 **List** 组件（或扩展 **List** 类的任何组件）设计应用程序时，了解列表的设计方式是很有帮助的。下面是 **Macromedia** 在开发 **List** 类时使用的一些基本假设和要求：

- 保持其短小、快速与简洁。

不要毫无必要地使事情复杂化。这是首要的设计准则。下面列出的大多数要求都是基于此准则的。

- 列表具有统一的行高。

每行的高度必须相同，该高度可以在创作期间或者在运行时设置。

- 列表必须能放大到容纳数千条记录。

- 列表不度量文本。

此限制最有可能带来分歧。因为列表必须能放大到容纳数千条记录，而每条记录都可能包含一个超长字符串，列表无法增大以适应其最长的文本字符串，也不能在“自动”模式下添加水平滚动条。同时，度量数千条字符串的工作量是非常巨大的。折衷的办法是使用 `maxHPosition` 属性，该属性在 `vScrollPolicy` 设置为 "on" 时为列表提供用于滚动的额外的缓冲空间。

如果您确信可能要处理长字符串，请将 `hScrollPolicy` 设置为 "on"，然后向 **List** 或 **Tree** 组件添加一个 200 像素的 `maxHPosition` 值。这样大致可确保用户能够通过滚动查看所有内容。不过，**DataGrid** 组件确实支持以 "auto" 作为 `hScrollPolicy` 值，因为它度量的是列（每一项的列宽都相同）而不是文本。

列表不度量文本这个事实也解释了列表具有统一行高的原因。调整每行的大小来适合文本将需要进行大量的度量工作。例如，如果要在行高不统一的列表上精确地显示滚动条，就需要预先度量每个行。

- 可见行越多，列表性能越差。

虽然列表能显示 5000 条记录，但它们不能一次呈现 5000 条记录。舞台上的可见行（通过 `rowCount` 属性指定）越多，列表呈现它们所需的工作也越多。如果可能，限制可见行的数量是最好的解决方案。

- 列表不是表。

例如，**DataGrid** 组件（扩展了 **List** 类）旨在为大量记录提供一个界面。它们的设计意图不是为了显示完整的信息，而是用于显示够用的信息，这样用户可以进一步查看更多信息。**Microsoft Outlook** 中的邮件视图就是一个很好的示例。您不会在网格中看到整个电子邮件；否则邮件将难以阅读，客户端的性能也让人难以忍受。**Outlook** 只显示够用的信息，用户可以深入到邮件内部去查看详细信息。

## List 参数

可以在“属性”检查器或“组件”检查器中为每个 **List** 组件实例设置以下创作参数：

**data**，由填充列表数据的值组成的数组。默认值为 []（空数组）。没有相应的运行时属性。

**labels**，由填充列表的标签值的文本值组成的数组。默认值为 []（空数组）。没有相应的运行时属性。

**multipleSelection**，一个布尔值，它指示是 (true) 否 (false) 可以选择多个值。默认值为 false。

**rowHeight**，指示每行的高度，以像素为单位。默认值是 20。设置字体不会更改行的高度。

您可以使用 **List** 实例的方法、属性和事件为其编写 **ActionScript** 来设置其它选项。有关更多信息，请参见第 711 页的“**List 类**”。

## 创建具有 List 组件的应用程序

以下过程解释了如何在创作时将 **List** 组件添加到应用程序。在本例中，列表是一个有三个项目的范例。

### 将单个 List 组件添加到应用程序：

1. 将一个 **List** 组件从“组件”面板拖到舞台上。
2. 选择“任意变形”工具，并调整组件的大小以适合应用程序。
3. 在属性检查器中，执行以下操作：
  - 输入实例名称 **my\_list**。
  - 为标签参数输入 **Item1**、**Item2** 和 **Item3**。
  - 为数据参数输入 **item1.html**、**item2.html** 和 **item3.html**。
4. 选择“控制”>“测试影片”，以查看包含其项的列表。
5. 返回到创作环境，插入新的图层并将其命名为 **actions**。
6. 在 **actions** 图层的第 1 帧中添加以下 **ActionScript** 代码。

```
my_list.change = function(evt:Object) {
 getURL(evt.target.selectedItem.data, "_blank");
};
my_list.addEventListener("change", my_list);
```

### 使用数据提供程序填充 List 实例：

1. 将一个 **List** 组件从“组件”面板拖到舞台上。
2. 选择“任意变形”工具，并调整组件的大小以适合应用程序。
3. 在属性检查器中，输入实例名称 **my\_list**。
4. 在时间轴中选择第一帧，在“动作”面板中，输入以下代码：

```
my_list.dataProvider = myDP;
```

如果已经定义了名为 **myDP** 的数据提供程序，则列表中将填入数据。（有关数据提供程序的更多信息，请参见 [List.dataProvider](#)。）

5. 选择“控制”>“测试影片”，以查看包含其项的列表。

### 使用 List 组件控制影片剪辑实例

1. 将一个 List 组件从“组件”面板拖到舞台上。
2. 选择“任意变形”工具，并调整组件的大小以适合应用程序。
3. 在属性检查器中，输入实例名称 **my\_list**。
4. 在舞台上创建一个影片剪辑，并为其指定实例名称 **my\_mc**。
5. 在元件编辑模式下打开该影片剪辑，然后添加一些动画。
6. 插入一个新的图层，并将其命名为 **actions**。
7. 在 **actions** 图层的第 1 帧中添加以下 **ActionScript** 代码。

```
my_list.addItem({label:"play", data:"play"});
my_list.addItem({label:"stop", data:"stop"});
var listHandler:Object = new Object();
listHandler.change = function(evt:Object) {
 switch (evt.target.selectedItem.data) {
 case "play" :
 my_mc.play();
 break;
 case "stop" :
 my_mc.stop();
 break;
 default :
 trace("unhandled event: "+evt.target.selectedItem.data);
 break;
 }
};
my_list.addEventListener("change", listHandler);
```

8. 选择“控制” > “测试影片”，以使用列表来停止和播放 **my\_mc** 影片剪辑实例。

### 使用 ActionScript 创建 List 组件实例：

1. 将 List 组件从“组件”面板拖到库中。

此操作将组件添加到库中，但不会在应用程序中显示。

2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
this.createClassObject(mx.controls.List, "my_list", 1);
my_list.addItem({label:"One", data:dt1});
my_list.addItem({label:"Two", data:dt2});
```

此脚本使用 `UIObject.createClassObject()` 方法创建 List 实例。

3. 选择“控制” > “测试影片”。

# 自定义 List 组件

在创作过程中和运行时，可以在水平方向和垂直方向上将 **List** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，则使用 `List.setSize()` 方法（请参见 `UIObject.setSize()`）。

当调整列表的大小后，列表的行会在水平方向收缩，剪下其中的任何文本。在垂直方向，列表根据需要增加或删除行。滚动条自动对它们定位。有关滚动条的更多信息，请参见第 1011 页的“**ScrollPane 组件**”。

## 对 List 组件使用样式

您可以设置样式属性以更改 **List** 实例的外观。

**List** 组件使用下列样式：

样式	主题	说明
themeColor	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
alternatingRowColors	光晕和范例	指定交替模式中行的颜色。它的值可以是两个或多个颜色（例如 0xFF00FF、0xCC6699 和 0x996699）组成的数组。与单值颜色样式不同，alternatingRowColors 不接受颜色名称；颜色的值必须为数字颜色代码。默认情况下，没有设置此样式，而是对所有行使用 backgroundColor 来代替此样式。
backgroundColor	光晕和范例	列表的背景颜色。默认的颜色为白色，在类样式声明中定义。如果指定了 alternatingRowColors，则此样式被忽略。
backgroundDisabledColor	光晕和范例	当组件的 enabled 属性设置为 "false" 时的背景颜色。默认值为 0xDDDDDD（中度灰）。
borderStyle	光晕和范例	List 组件使用 RectBorder 实例作为其边框，并响应在该类中定义的样式。请参见第 985 页的“ <b>RectBorder 类</b> ”。默认边框样式为 "inset"。
color	光晕和范例	文本颜色。
disabledColor	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。



样式	主题	说明
embedFonts	光晕和范例	一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。
fontFamily	光晕和范例	文本的字体名称。默认值为 "_sans"。
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式："normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细："none" 或 "bold"。默认值为 "none"。在调用 setStyle() 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 getStyle() 的调用将返回 "none"。
textAlign	光晕和范例	文本对齐方式："left"、"right" 或 "center"。默认值为 "left"。
textDecoration	光晕和范例	文本修饰："none" 或 "underline"。默认值为 "none"。
textIndent	光晕和范例	表示文本缩进的数字。默认值为 0。
defaultIcon	光晕和范例	在每行上显示的默认图标的名称。默认值为 undefined，表示不显示图标。
repeatDelay	光晕和范例	用户在滚动条中第一次按下某个按钮与开始重复此动作之间间隔的毫秒数。默认值为 500（半秒）。
repeatInterval	光晕和范例	用户在滚动条的某个按钮上使鼠标按键保持按下状态时两次自动单击之间所间隔的毫秒数。默认值为 35。
rollOverColor	光晕和范例	滑过的行的背景颜色。“光晕”主题的默认值为 0xE3FFD6（亮绿），“范例”主题的默认值为 0xA9AAAA（浅灰）。当通过调用 setStyle() 更改 themeColor 时，框架会将 rollOverColor 设置为一个与所选 themeColor 相关的值。
selectionColor	光晕和范例	所选行的背景颜色。“光晕”主题的默认值为 0xCDFFC1（浅绿），“范例”主题的默认值为 0xEEEEEE（极浅灰）。当通过调用 setStyle() 更改 themeColor 时，框架会将 selectionColor 设置为与所选 themeColor 相关的值。
selectionDuration	光晕和范例	在正常状态和所选状态之间过渡所耗费的时间，以毫秒为单位。默认值为 200。

样式	主题	说明
<code>selectionDisabledColor</code>	光晕和范例	所选行的背景颜色。默认值为 <code>0xDDDDDD</code> （中度灰）。由于此属性的默认值与 <code>backgroundDisabledColor</code> 的默认值相同，因此除非更改了其中的某个样式属性，否则当组件被禁用时该选区会不可见。
<code>selectionEasing</code>	光晕和范例	对用于控制选择状态间过渡的扩大等式的引用。这只适用于从正常状态到所选择状态的过渡。默认等式使用正弦输入 / 输出公式。有关更多信息，请参见《使用组件》中的“自定义组件动画”。
<code>textRollOverColor</code>	光晕和范例	指针在文本上滑过时文本的颜色。默认值为 <code>0x2B333C</code> （深灰）。设置 <code>rollOverColor</code> 时此样式非常重要，因为这两个设置必须相辅相成，才能使文本在指针滑过期间易于查看。
<code>textSelectedColor</code>	光晕和范例	所选行的文本颜色。默认值为 <code>0x005F33</code> （深灰）。设置 <code>selectionColor</code> 时此样式非常重要，因为这两个设置必须相互补充，才能使文本在被选择期间易于查看。
<code>useRollOver</code>	光晕和范例	确定滑过一行时是否激活加亮显示该行。默认值为 <code>true</code> 。

## 为文档中的所有 List 组件设置样式

`List` 类继承自 `ScrollSelectList` 类。默认的分类样式属性在 `ScrollSelectList` 类上定义，而 `Menu` 组件和所有基于列表的组件均扩展了 `ScrollSelectList` 类。可以直接为此类设置新的默认样式值，新的设置将反映在所有受影响的组件中。

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

若要仅设置 `List` 组件和基于 `List` 的组件的样式属性，可创建一个新的 `CSSStyleDeclaration` 实例，并将其存储在 `_global.styles.List` 中。

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.List == undefined) {
 _global.styles.List = new CSSStyleDeclaration();
}
_global.styles.List.setStyle("backgroundColor", 0xFF00AA);
```

在创建新的类级别样式声明时，由 `ScrollSelectList` 声明提供的所有默认值都将丢失。这包括支持鼠标事件所需的 `backgroundColor`。若要创建类级别样式声明并保留默认值，请使用 `for..in` 循环将旧的设置复制到新声明中。

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.List;
for (var style in source) {
 target.setStyle(style, source.getStyle(style));
}
```

要向 **List** 组件而不是扩展 **List** 的组件（**DataGrid** 和 **Tree**）提供样式，必须为这些子类提供类级别样式声明。

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.DataGrid == undefined) {
 _global.styles.DataGrid = new CSSStyleDeclaration();
}
_global.styles.DataGrid.setStyle("backgroundColor", 0xFFFFFF);
if (_global.styles.Tree == undefined) {
 _global.styles.Tree = new CSSStyleDeclaration();
}
_global.styles.Tree.setStyle("backgroundColor", 0xFFFFFF);
```

有关类级别样式的更多信息，请参见《使用组件》中的“为组件类设置样式”。

## 在 List 组件中使用外观

**List** 组件使用 **RectBorder** 的实例作为其边框，使用滚动条来滚动图像。有关设置这些可视元素的外观的更多信息，请参见第 985 页的“**RectBorder** 类”和第 1283 页的“对 **UIScrollBar** 组件使用外观”。

# List 类

继承 **MovieClip** > **UIObject** 类 > **UIComponent** 类 > **View** > **ScrollView** > **ScrollSelectList** > **List**

**ActionScript** 类名称 **mx.controls.List**

**List** 组件由下列三个部分组成：项、行和一个数据提供程序。

项是用于将信息单位存储在列表中的 **ActionScript** 对象。可以将列表看作一个数组；数组中的每个索引空间就是一个项。项是一个对象，它通常有一个显示出来的 `label` 属性和一个用于存储数据的 `data` 属性。

行是用于显示项的组件。默认情况下，列表会提供行（使用 `SelectableRow` 类），或者您也可以提供行，它们通常作为 `SelectableRow` 类的子类。`SelectableRow` 类实现 `CellRenderer` API，该接口是属性和方法的集合，通过这些属性和方法，列表可以操作各行并将数据和状态信息（例如，大小、已被选中等等）发送到行以供显示。

数据提供程序是列表中的项列表的数据模型。同一帧中作为列表的任何数组都会自动得到一些方法，这些方法允许您操作数据并将更改广播给多个视图。您可以创建一个 `Array` 实例或者从服务器上获取一个实例，然后将它用作多个列表、组合框、数据网格等的数据模型。`List` 组件包含代理其数据提供程序的方法（如 `addItem()` 和 `removeItem()`）。如果没有为列表提供外部数据提供程序，则这些方法会自动创建一个数据提供程序实例，该实例会通过 `List.dataProvider` 被公开。

若要将 `List` 组件添加到应用程序的 `Tab` 键顺序，请设置其 `tabIndex` 属性（请参见 `UIComponent.tabIndex`）。`List` 组件使用焦点管理器覆盖默认的 `Flash Player` 焦点矩形，并绘制一个带圆角的自定义焦点矩形。有关更多信息，请参见《使用组件》中的“创建自定义焦点导航”。

每个组件类都有一个 `version` 属性，该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.List.version);
```

提醒

代码 `trace(myListInstance.version);` 返回 `undefined`。

## List 类的方法摘要

下表列出了 `List` 类的方法。

方法	说明
<code>List.addItem()</code>	向列表的结尾添加项目。
<code>List.addItemAt()</code>	将项目添加到指定索引处的列表。
<code>List.getItemAt()</code>	返回指定索引处的项目。
<code>List.removeAll()</code>	删除列表中的所有项目。
<code>List.removeItemAt()</code>	删除指定索引处的项目。
<code>List.replaceItemAt()</code>	用其它项目替换指定索引处的项目。
<code>List.setPropertiesAt()</code>	将指定的属性应用到指定的项目。
<code>List.sortItems()</code>	按照指定的比较函数对列表中的项目进行排序。
<code>List.sortItemsBy()</code>	按照指定的属性对列表中的项目进行排序。

## 从 UIObject 类继承的方法

下表列出了 **List** 类从 **UIObject** 类继承的方法。调用这些方法时，请使用 *listInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **List** 类从 **UIComponent** 类继承的方法。调用这些方法时，请使用 *listInstance.methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# List 类的属性摘要

下表列出了 `List` 类的属性。

属性	说明
<code>List.cellRenderer</code>	指定要使用的类或元件以显示列表的每一行。
<code>List.dataProvider</code>	列表项的来源。
<code>List.hPosition</code>	列表的水平位置。
<code>List.hScrollPolicy</code>	指示是 ("on") 否 ("off") 显示水平滚动条。
<code>List.iconField</code>	各项目中用于指定图标的数据。
<code>List.iconFunction</code>	一个函数，它确定要使用的图标。
<code>List.labelField</code>	指定各项目中用作标签文本的数据。
<code>List.labelFunction</code>	一个函数，它确定各个项目的哪些数据要用作标签文本。
<code>List.length</code>	列表中的项目数。该属性为只读。
<code>List.maxHPosition</code>	当将 <code>List.hScrollPolicy</code> 设置为 "on" 时，指定列表可以向右滚动的像素数目。
<code>List.multipleSelection</code>	指定列表中是 (true) 否 (false) 允许多选。
<code>List.rowCount</code>	列表中至少可以看到一部分的行数。
<code>List.rowHeight</code>	列表中每行的像素高度。
<code>List.selectable</code>	指定列表是 (true) 否 (false) 为可选择列表。
<code>List.selectedIndex</code>	单选列表中的选择索引。
<code>List.selectedIndices</code>	多选列表中的已选择项目的数组。
<code>List.selectedItem</code>	单选列表中的已选择项目。该属性为只读。
<code>List.selectedItems</code>	多选列表中的已选择的项目对象。该属性为只读。
<code>List.vPosition</code>	列表最顶部的可见项目。
<code>List.vScrollPolicy</code>	指示是显示 ("on")、不显示 ("off") 还是在需要时显示 ("auto") 垂直滚动条。

## 从 UIObject 类继承的属性

下表列出了 **List** 类从 **UIObject** 类继承的属性。访问这些属性时，请使用 *listInstance.propertyName* 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 **List** 类从 **UIComponent** 类继承的属性。访问这些属性时，请使用 *listInstance.propertyName* 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

# List 类的事件摘要

下表列出了 List 类的事件。

事件	说明
List.change	只要用户交互造成选择更改就广播。
List.itemRollOut	指针在列表项上滑过然后又滑离时广播。
List.itemRollOver	当指针滑过列表项时进行广播。
List.scroll	滚动列表时，进行广播。

## 从 UIObject 类继承的事件

下表列出了 List 类从 UIObject 类继承的事件。

事件	说明
UIObject.draw	当对象将要绘制它的图形时进行广播。
UIObject.hide	在对象的状态从可见变为不可见时广播。
UIObject.load	创建子对象时广播。
UIObject.move	移动了对象时广播。
UIObject.resize	在调整对象大小后广播。
UIObject.reveal	在对象的状态从不可见变为可见时广播。
UIObject.unload	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 List 类从 UIComponent 类继承的事件。

事件	说明
UIComponent.focusIn	当对象收到焦点时进行广播。
UIComponent.focusOut	当对象失去焦点时进行广播。
UIComponent.keyDown	当按下按键时进行广播。
UIComponent.keyUp	当松开按键时进行广播。



# List.addItem()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listInstance.addItem(label[, data])
```

```
listInstance.addItem(itemObject)
```

## 参数

*label* 一个字符串，它指示新项的标签。

*data* 项的数据。此参数是可选参数，它可以是任何数据类型。

*itemObject* 一个项对象，通常具有 *label* 属性和 *data* 属性。

## 返回

在其位置添加了项的索引。

## 说明

方法：在列表的结尾添加新项。

在第一个用法示例中，始终使用指定的 *label* 属性和 *data* 属性（如果已指定）来创建项对象。

第二个用法示例添加了指定的项对象。

调用此方法会修改 **List** 组件的数据提供程序。如果与其它组件共享数据提供程序，则那些组件也会更新。

## 示例

以下两行代码都会将项添加到 *my\_list* 实例。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;
```

```
my_list.addItem("this is an Item");
```

```
my_list.addItem({label:"Gordon", age:"very old", data:123});
```

# List.addItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listInstance.addItemAt(index, label[, data])
```

```
listInstance.addItemAt(index, itemObject)
```

## 参数

*index* 一个大于或等于 0 的数字，指示项的位置。

*label* 一个字符串，它指示新项的标签。

*data* 项的数据。此参数是可选参数，它可以是任何数据类型。

*itemObject* 一个项对象，通常具有 *label* 属性和 *data* 属性。

## 返回

在其位置添加了项的索引。

## 说明

方法：将新项添加到 *index* 参数指定的位置。

在第一个用法示例中，始终使用指定的 *label* 属性和 *data* 属性（如果已指定）来创建项对象。

第二个用法示例添加了指定的项对象。

调用此方法会修改 **List** 组件的数据提供程序。如果与其它组件共享数据提供程序，则那些组件也会更新。

## 示例

以下示例将一个项目添加到第一个索引位置，这是列表中的第二个项目。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.addItem("this is an Item");
my_list.addItem({label:"Gordon", age:"very old", data:123});
my_list.addItemAt(1, {label:"Red", data:0xFF0000});
```

# List.cellRenderer

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listInstance.cellRenderer
```

## 说明

属性；指定要用于列表各行的单元格渲染器。此属性必须是类对象引用，或者是元件链接标识符。用于此属性的任何类都必须实现 [CellRenderer API](#)。

## 示例

下面的示例使用链接标识符来设置一个新的单元格渲染器：

```
my_list.cellRenderer = "ComboBoxCell";
```

# List.change

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 // 此处是您的代码。
};
listInstance.addEventListener("change", listenerObject);
```

用法 2：

```
on (change) {
 // 此处是您的代码。
}
```

## 说明

事件：当所选的列表的索引因用户交互操作而改变时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本示例中为 *change*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法：当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见[第 462 页的“EventDispatcher 类 \(API\)”](#)。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **List** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **List** 实例 `my_list` 上，它将 “\_level0.my\_list” 发送到 “输出” 面板：

```
on (change) {
 trace(this);
}
```

## 示例

以下示例将向 **List** 组件中添加三项。如果更改所选列表值，则会导致新选定项的值显示在 “输出” 面板中。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});

// 创建侦听器对象。
var listListener:Object = new Object();
listListener.change = function(evt_obj:Object) {
 trace("Value changed to: " + evt_obj.target.value);
}

// 添加侦听器。
my_list.addEventListener("change", listListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# List.dataProvider

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.dataProvider*

## 说明

属性：在列表中查看的项目的数据模型。此属性的值可以是一个数组或任何实现 **DataProvider** API 的对象。默认值为 []。有关更多信息，请参见第295页的“**DataProvider API**”。

和其它支持数据的组件一样，**List** 组件会将方法添加到 **Array** 对象的原型，以使它们符合 **DataProvider** API。因此，任何同时作为列表存在的数组都可以自动获得作为列表的数据模型所需的所有方法（`addItem()`、`getItemAt()` 等），并可用于向多个组件广播模型更改。

如果数组包含对象，则会访问 `List.labelField` 或 `List.labelFunction` 属性，以确定要显示项的哪些部分。默认值为 "label"，因此如果存在 label 字段，则会选择显示该字段；如果不存在该字段，则显示用逗号分隔的所有字段的列表。



如果该数组在每个索引处都包含字符串，而不包含对象，该列表就无法对项目进行排序和保持选定状态。进行任何排序都会导致丢失所做的选择。

任何实现 **DataProvider** API 的实例都可以作为 **List** 组件的数据提供程序。这包括 **Flash Remoting** 记录集、**Firefly** 数据集等等。

## 示例

以下示例使用一个字符串数组填充列表：

```
my_list.dataProvider = ["Ground Shipping", "2nd Day Air", "Next Day Air"];
```

本示例创建一个数据提供程序数组，并将其分配给 `dataProvider` 属性，如下所示：

```
var myDP_array:Array = new Array();
my_list.dataProvider = myDP_array;

var accounts_array:Array = new Array();
accounts_array.push({name:"checkings", accountID:12345});
accounts_array.push({name:"savings", accountID:67890});
```

```
for (var i:Number = 0; i < accounts_array.length; i++) {
 // 对数据提供程序所做的这些更改将广播到列表。
 myDP_array.addItem({label:accounts_array[i].name,
 data:accounts_array[i].accountID});
}
```

## List.getItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance.getItemAt(index)*

### 参数

*index* 一个大于或等于 0 且小于 `List.length` 的数字。它指定要检索的项目的索引。

### 返回

具有索引的项对象；如果索引超出范围，则为 `undefined`。

### 说明

方法；检索位于指定索引处的项。此方法将从数组、`DataProvider` 或使用 `CellRenderer.setValue()` 创建的数据对象中获取数据项。

### 示例

以下代码显示索引位置 2 处的项的标签。若要试用此代码，请将一个 `List` 组件拖到舞台上，并为其指定实例名称 **my\_list**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});

trace(my_list.getItemAt(2).label);
```

# List.hPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.hPosition*

## 说明

属性：水平滚动列表到指定的像素数。只有在 `hScrollPolicy` 的值是 "on"，而且列表的 `maxHPosition` 大于 0 的情况下，才能设置 `hPosition`。

## 示例

每当水平滚动列表实例时，以下代码都会显示 `hPosition` 的当前值。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.setSize(150, 100);
my_list.hScrollPolicy = "on";
my_list.maxHPosition = 50;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});

var listListener:Object = new Object();
listListener.scroll = function (evt_obj:Object) {
 trace("my_list.hPosition = " + my_list.hPosition);
}
my_list.addEventListener("scroll", listListener);
```

# List.hScrollPolicy

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.hScrollPolicy*

## 说明

属性；确定是否显示水平滚动条的字符串；该值可以是 "on" 或 "off"。默认值为 "off"。水平滚动条不会度量文本，您必须设置最大水平滚动位置（请参见 [List.maxHPosition](#)）。



List.hScrollPolicy 不支持值 "auto"。

## 示例

以下代码最多可以使列表水平滚动 200 个像素。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.setSize(150, 100);
my_list.hScrollPolicy = "on";
my_list.maxHPosition = 200;

my_list.addItem({data:'flash', label:'Flash'});
my_list.addItem({data:'dreamweaver', label:'Dreanweaver'});
my_list.addItem({data:'coldfusion', label:'ColdFusion'});
```

## 另请参见

[List.hPosition](#)、[List.maxHPosition](#)



# List.iconField

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance*.iconField

## 说明

属性；指定了要用作图标标识符的字段名称。如果字段的值为 `undefined`，则使用 `defaultIcon` 样式所指定的默认图标。如果 `defaultIcon` 样式为 `undefined`，则不使用任何图标。

## 示例

以下示例将 `iconField` 属性设置为每一项的 `icon` 属性。若要试用此代码，请将一个 `List` 组件拖到舞台上，并为其指定实例名称 **my\_list**，然后创建实例名称分别为 `flash`、`dreamweaver` 和 `cf` 的三个元件。将以下代码添加到时间轴的第一帧中：

```
/**
 要求：
 - 舞台上有 List 组件（实例名称: my_list）
 - 库中有链接 ID 为 “flash” 的影片剪辑 / 图形元件
 - 库中有链接 ID 为 “dreamweaver” 的影片剪辑 / 图形元件
 - 库中有链接 ID 为 “cf” 的影片剪辑 / 图形元件
*/

var my_list:mx.controls.List;

my_list.setSize(200, 100);

my_list.addItem({data:"flash", label:"Flash", icon:"flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver",
 icon:"dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion", icon:"cf"});

my_list.iconField = "icon";
```

## 另请参见

[List.iconFunction](#)

# List.iconFunction

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ListInstance*.iconFunction

## 说明

属性：指定一个函数，用于确定每行将使用哪个图标来显示其项目。此函数会接收参数 *item*（该参数指示所呈现的项），并且必须返回一个表示图标元件标识符的字符串。

## 示例

下面范例添加了一些图标，这些图标指明文件是图像还是文本文档。如果 `data.fileExtension` 字段包含值 "jpg" 或 "gif"，则使用 "pictureIcon" 等图标。

```
my_list.iconFunction = function(item:Object):String {
 var type:String = item.data.fileExtension;
 if (type == "jpg" || type == "gif") {
 return "pictureIcon";
 } else if (type == "doc" || type == "txt") {
 return "docIcon";
 }
}
```

以下示例将 `iconField` 属性设置为每一项的 `icon` 属性。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**，然后创建实例名称分别为 `flash`、`dreamweaver` 和 `cf` 的三个元件。将以下代码添加到时间轴的第一帧中：

```
/**
 要求：
 - 舞台上 有 List 组件（实例名称：my_list）
 - 库中有链接 ID 为 “flashIcon” 的影片剪辑 / 图形元件
*/

var my_list:mx.controls.List;

my_list.setSize(200, 100);

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.iconFunction = function(item:Object):String {
```

```
 if (item.data == "flash") {
 // 将图标放在包含数据 “flash” 的列表项的旁边。
 return "flashIcon";
 }
 };
 my_list.iconField = "icon";
```

## List.itemRollOut

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.itemRollOut = function(eventObject:Object) {
 // 此处是您的代码。
};
listInstance.addEventListener("itemRollOut", listenerObject);
```

用法 2:

```
on (itemRollOut) {
 // 此处是您的代码。
}
```

### 事件对象

除了事件对象的标准属性之外，itemRollOut 事件还具有 index 属性，该属性指定已经滚动出去的项的数量。

## 说明

事件：当指针滑过然后又滑出列表项时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本示例中为 *itemRollOut*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 462 页的“[EventDispatcher 类 \(API\)](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **List** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **List** 实例 `my_list` 上，它将“\_level0.my\_list”发送到“输出”面板：

```
on (itemRollOut) {
 trace(this);
}
```

## 示例

以下示例向“输出”面板发送一条消息，指示已在哪个项索引编号上滑过：

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

// 创建侦听器对象。
var listListener:Object = new Object();
listListener.itemRollOut = function(evt_obj:Object) {
 trace("Item #" + evt_obj.index + " has been rolled out.");
};

// 添加侦听器。
my_list.addEventListener("itemRollOut", listListener);
```

## 另请参见

[List.itemRollOver](#)

# List.itemRollOver

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.itemRollOver = function(eventObject:Object) {
 // 此处是您的代码。
};
listInstance.addEventListener("itemRollOver", listenerObject);
```

用法 2:

```
on (itemRollOver) {
 // 此处是您的代码。
}
```

## 事件对象

除了事件对象的标准属性之外，itemRollOver 事件还具有 index 属性，该属性指定已滑过的项的数量。

## 说明

事件：当滑过列表项时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本示例中为 itemRollOver），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 462 页的“[EventDispatcher 类 \(API\)](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **List** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **List** 实例 `my_list` 上，它将 “\_level0.my\_list” 发送到 “输出” 面板：

```
on (itemRollOver) {
 trace(this);
}
```

## 示例

以下示例向 “输出” 面板发送一条消息，指示已在哪个项索引编号上滑过：

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

// 创建侦听器对象。
var listListener:Object = new Object();
listListener.itemRollOver = function (evt_obj:Object) {
 trace("Item #" + evt_obj.index + " has been rolled over.");
};

// 添加侦听器。
my_list.addEventListener("itemRollOver", listListener);
```

## 另请参见

[List.itemRollOut](#)

# List.labelField

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance*.labelField

## 说明

属性：指定每个项目中的一个字段用作显示文本。此属性会获得该字段的值，并将其用作标签。默认值为 "label"。

## 示例

以下示例将 `labelField` 属性设置为每一项的 "name" 字段。“Nina”将显示为第二行代码中添加的项目的标签：

```
var my_list:mx.controls.List;

my_list.labelField = "name";
my_list.addItem({name: "Nina", age: 25});
```

## 另请参见

[List.labelFunction](#)

# List.labelFunction

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ListInstance*.labelFunction

## 说明

属性：指定用于确定显示每个项目的哪个字段（或字段组合）的函数。此函数会接收一个参数 *item*（该参数指示所呈现的项），并且必须返回一个表示要显示的文本的字符串。

## 示例

以下示例让标签显示项的一些设置了格式的细节：

```
var my_list:mx.controls.List;

my_list.setSize(300, 100);

// 定义列表数据的显示方式。
my_list.labelFunction = function(item_obj:Object):String {
 var label_str:String = item_obj.label + " - Code is: " + item_obj.data;
 return label_str;
}

// 向列表中添加数据。
my_list.addItem({data:"f", label:"Flash"});
my_list.addItem({data:"d", label:"Dreamweaver"});
my_list.addItem({data:"c", label:"ColdFusion"});
```

## 另请参见

[List.labelField](#)

# List.length

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.length*

## 说明

属性（只读）：列表中的项数。

## 示例

以下示例显示列表的数据提供程序中当前的项数：

```
var my_list:mx.controls.List;

// 向列表中添加数据。
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var listLength_num:Number = my_list.length;
trace("Length of List: " + listLength_num);
```

# List.maxHPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.maxHPosition*

## 说明

属性；指定当 `List.hScrollPolicy` 设置为 "on" 时，列表可以滚动的像素数。列表无法精确地测量所包含的文本的宽度。必须设置 `maxHPosition` 以指示列表需要的滚动量。如果不设置此属性，则列表将不水平滚动。



## 示例

以下示例创建了一个带 200 像素的可以水平滚动的列表：

```
var my_list:mx.controls.List;

my_list.setSize(150, 100);
my_list.hScrollPolicy = "on";
my_list.maxHPosition = 200;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
```

## 另请参见

[List.hScrollPolicy](#)

# List.multipleSelection

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.multipleSelection*

## 说明

属性；指示是可以多选 (true) 还是只可以单选 (false)。默认值为 false。

## 示例

以下示例测试确定是否可以选择多个项目，如果可以，则在标签组件中显示说明。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。然后，将一个 **Label** 组件拖到舞台上，并为其指定实例名称 **my\_label**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.multipleSelection = true;
```

```
if (my_list.multipleSelection) {
 my_label.text = "Hold down Control or Shift to select multiple items";
 my_label.autoSize = "left";
}
```

## List.removeAll()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
listInstance.removeAll()
```

### 返回

无。

### 说明

方法：删除列表中的所有项。

调用此方法会修改 **List** 组件的数据提供程序。如果与其它组件共享数据提供程序，则那些组件也会更新。

### 示例

当按下按钮时，以下代码将会清除 **List** 组件中的所有项。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。然后，将一个 **Button** 组件拖到舞台上，并为其指定实例名称 **remove\_button**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;
var remove_button:mx.controls.Button;

remove_button.label = "Remove";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 my_list.removeAll();
 evt_obj.target.enabled = false;
}
remove_button.addEventListener("click", buttonListener);
```

# List.removeItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listInstance.removeItemAt(index)
```

## 参数

*index* 一个数字，指示项的位置。此值必须大于 0 且小于 `List.length`。

## 返回

一个对象；已删除的项（如果项不存在，则为 `undefined`）。

## 说明

方法；删除指定索引位置处的项目。指定索引后面的列表索引将依次减 1。

调用此方法会修改 **List** 组件的数据提供程序。如果与其它组件共享数据提供程序，则那些组件也会更新。

## 示例

当按下按钮时，以下代码将会清除 **List** 组件中的选定项。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。然后，将一个 **Button** 组件拖到舞台上，并为其指定实例名称 **remove\_button**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;
var remove_button:mx.controls.Button;

remove_button.label = "Remove";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 if (my_list.selectedIndex != undefined) {
 my_list.removeItemAt(my_list.selectedIndex);
 }
}
remove_button.addEventListener("click", buttonListener);
```

# List.replaceItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listInstance.replaceItemAt(index, label[, data])
```

```
listInstance.replaceItemAt(index, itemObject)
```

## 参数

*index* 一个大于 0 且小于 `List.length` 的数字，指示项的插入位置（新项的索引）。

*label* 一个字符串，它指示新项的标签。

*data* 项的数据。此参数是可选参数，它可以是任何类型。

*itemObject* 用作项的对象，通常包含 `label` 和 `data` 属性。

## 返回

无。

## 说明

方法：替换位于指定索引处的项目的内容。

调用此方法会修改 **List** 组件的数据提供程序。如果与其它组件共享数据提供程序，则那些组件也会更新。

## 示例

以下示例将替换当前所选位置的项。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。然后，将一个 **Button** 组件拖到舞台上，并为其指定实例名称 **replace\_button**。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;
var replace_button:mx.controls.Button;

replace_button.label = "Replace";

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
```

```
 if (my_list.selectedIndex != undefined) {
 my_list.replaceItemAt(my_list.selectedIndex, {data:"flex",
 label:"Flex"});
 }
 }
 replace_button.addEventListener("click", buttonListener);
```

## List.rowCount

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance*.rowCount

### 说明

属性：列表中至少可以看到一部分的行数。如果您已经使用像素缩放了一个列表，但需要计算它的行数，则该属性很有用。相反，设置行数可以保证显示准确的行数，而不会在底部出现部分行。

代码 `my_list.rowCount = num` 相当于代码 `my_list.setSize(my_list.width, h)`（其中 `h` 为显示 `num` 个项所需的高度）。

默认值基于创作时设置的或由 `list.setSize()` 方法（请参见 [UIObject.setSize\(\)](#)）设置的列表高度。

### 示例

以下示例会发现列表中的可以看到的项数量：

```
var rowCount = my_list.rowCount;
```

以下示例会使列表显示四项：

```
my_list.rowCount = 4;
```

以下示例将删除列表底部的部分行（如果存在部分行）：

```
my_list.rowCount = my_list.rowCount;
```

以下示例将列表设置为它可以完整显示的最小行数：

```
my_list.rowCount = 1;
trace("my_list has " + my_list.rowCount + " rows");
```

以下示例将使用 `setSize()` 方法调整列表的大小，然后设置 8 项的行数：

```
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.setSize(200, 30);
my_list.rowCount = 8;
trace("my_list has " + my_list.rowCount + " rows.");
```

## List.rowHeight

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance*.rowHeight

### 说明

属性：列表中每行的像素高度。字体设置不会增加行高以适合文字，因此设置 `rowHeight` 属性是确保项完全显示的最佳方法。默认值为 20。

### 示例

以下范例将每行设置为 30 像素：

```
my_list.rowHeight = 30;
```

以下示例将每行的行高设置为 30 个像素，然后调整列表的大小以匹配它所包含的项总数：

```
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.rowHeight = 30;
my_list.rowCount = my_list.length;
```

# List.scroll

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
 // 此处是您的代码。
};
listInstance.addEventListener("scroll", listenerObject);
```

用法 2:

```
on (scroll) {
 // 此处是您的代码。
}
```

## 事件对象

除了标准的事件对象属性之外，scroll 事件还有另一个属性 direction。它是一个字符串，有两个可能的值："horizontal" 和 "vertical"。对于 **ComboBox** 滚动事件，该值始终是 "vertical"。

## 说明

事件：当列表滚动时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本示例中为 scroll），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 462 页的“[EventDispatcher 类 \(API\)](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **List** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **List** 实例 `my_list` 上，它将 “\_level0.my\_list” 发送到 “输出” 面板：

```
on (scroll) {
 trace(this);
}
```

### 示例

每次滚动列表项时，以下示例都会发送列表的方向和位置：

```
var my_list:mx.controls.List;

my_list.rowCount = 2;
for (var i:Number = 0; i < 10; i++) {
 my_list.addItem({data:i, label:"Item #" + i});
}

var listListener:Object = new Object();
listListener.scroll = function(evt_obj:Object) {
 trace("list scrolled (direction:" + evt_obj.direction + ", position:" +
 evt_obj.position + ")");
};
my_list.addEventListener("scroll", listListener);
```

## List.selectable

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance*.selectable

### 说明

属性；一个布尔值，它指示列表是 (true) 否 (false) 为可选择列表。默认值为 true。



## 示例

以下示例通过将 `selectable` 属性设置为 `false` 来阻止用户选择列表中的项：

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.selectable = false;
```

# List.selectedIndex

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance*.selectedIndex

## 说明

属性：单选列表的已选择索引。如果没有选中任何内容，则该值为 `undefined`；如果选择了多项，则该值等于选中的最后一项。如果为 `selectedIndex` 分配一个值，则会清除当前所做任何选择，并选中指定项。

使用 `selectedIndex` 属性更改选定的内容时将不会发送 `change` 事件。若要发送 `change` 事件，请使用以下代码：

```
my_list.dispatchEvent({type:"change", target:my_list});
```

## 示例

以下示例在默认情况下将选择列表中的第一项，并在每当用户选择新项时显示当前所选项的索引：

```
var my_list:mx.controls.List;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

// 默认情况下选择第一项。
my_list.selectedIndex = 0;

var listListener:Object = new Object();
```

```
listListener.change = function(evt_obj:Object) {
 trace("selectedIndex = " + evt_obj.target.selectedIndex);
}
my_list.addEventListener("change", listListener);
```

### 另请参见

[List.selectedIndex](#)、[List.selectedItem](#)、[List.selectedItems](#)

## List.selectedIndex

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance*.selectedIndex

### 说明

属性：所选项目的索引数组。指定此属性将替换当前选择。将 selectedIndex 设置为一个长度为零的数组（或 undefined）将取消当前选择。如果没有选择任何项，则该值为 undefined。

selectedIndex 属性反映项的选择顺序。如果您单击第二项，再单击第三项，然后单击第一项，则 selectedIndex 将返回 [1,2,0]。

### 示例

以下示例将检索所选的索引：

```
var selIndices:Array = my_list.selectedIndex;
```

以下示例选择了四项：

```
var my_array = new Array (1, 4, 5, 7);
my_list.selectedIndex = my_array;
```

以下示例在默认情况下选择两个列表项，并在“输出”面板中显示其标签属性：

```
my_list.setSelection = true;
```

```
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
```

```
my_list.selectedIndex = [0, 2];
```

```

var numSelected:Number = my_list.selectedIndices.length;
for (var i:Number = 0; i < numSelected; i++) {
 trace("selectedIndices[" + i + "] = "+
 my_list.getItemAt(my_list.selectedIndices[i]).label);
}

```

### 另请参见

[List.selectedIndex](#)、[List.selectedItem](#)、[List.selectedItems](#)

## List.selectedItem

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance*.selectedItem

### 说明

属性（只读）：单选列表中的项对象。（在有多个选中项的多选列表中，selectedItem 将返回最近选中的项。）如果未选择任何内容，则该值为 undefined。

### 示例

以下示例将显示选中的标签：

```
trace(my_list.selectedItem.label);
```

只要用户从列表中选择新项，以下示例就显示所选项的内容：

```
my_list.setSelection = true;
```

```

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

```

```

// 创建侦听器对象。
var listListener:Object = new Object();
listListener.change = function(evt_obj:Object) {
 // 显示对象的每个属性。
 var tempStr:String = "[object";
 for (var i:String in evt_obj.target.selectedItem) {
 tempStr += " " + i + ":'" + evt_obj.target.selectedItem[i]+"'";
 }
 tempStr += "]";
}

```

```
 trace(tempStr);
 };
 // 添加侦听器。
 my_list.addEventListener("change", listListener);
```

### 另请参见

[List.selectedIndex](#)、[List.selectedIndices](#)、[List.selectedItems](#)

## List.selectedItems

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*listInstance*.selectedItems

### 说明

属性（只读）：所选项对象组成的数组。在多选列表中，selectedItems 允许您以项对象的形式访问选中的一组项。

### 示例

以下示例会检索一个所选项目对象的数组：

```
var myObjArray:Array = my_list.selectedItems;
```

以下示例在舞台上显示两个 **List** 实例。用户从第一个列表中选择一项时，会将所选项复制到第二个列表。若要试用此代码，您必须将 **List** 组件的副本添加到当前文档的库中。将以下代码添加到时间轴的第一帧中：

```
this.createClassObject(mx.controls.List, "my_list", 10,
 {multipleSelection:true});
my_list.setSize(200, 100);

this.createClassObject(mx.controls.List, "selectedItems_list", 20,
 {selectable:false});
selectedItems_list.setSize(200, 100);
selectedItems_list.move(0, 110);

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

var listListener:Object = new Object();
```

```
listListener.change = function(evt_obj:Object) {
 trace("You have selected " + my_list.selectedItems.length + " items.");
 selectedItems_list.dataProvider = my_list.selectedItems;
}
my_list.addEventListener("change", listListener);
```

另请参见

[List.selectedIndex](#)、[List.selectedItem](#)、[List.selectedIndices](#)

## List.setPropertiesAt()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

```
listInstance.setPropertiesAt(index, styleObj)
```

参数

*index* 一个大于 0 或小于 [List.length](#) 的数字，它指示要更改的项的索引。

*styleObj* 一个对象，它枚举要设置的属性和值。

返回

无。

说明

方法；将指定的属性应用到指定的项目。支持的属性有 `icon` 和 `backgroundColor`。

示例

以下示例将第三个项目的背景颜色更改为红色，并为其指定一个图标。若要试用此代码，请将一个 **List** 组件拖到舞台上，并为其指定实例名称 **my\_list**。然后，将 **MovieClip/** 图形元件添加到链接标识符为“**file**”的库中。将以下代码添加到时间轴的第一帧中：

```
var my_list:mx.controls.List;

my_list.setSize(200, 100);

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.setPropertiesAt(2, {backgroundColor:0xFF0000, icon: "file"});
```

# List.sortItems()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.sortItems(compareFunc)*

## 参数

*compareFunc* 对函数的引用。此函数用于比较两个项目，以确定它们的排序顺序。

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `sort (Array.sort method)`。

## 返回

无。

## 说明

方法：通过使用在 *compareFunc* 参数中指定的函数对列表中的项进行排序。

## 示例

以下示例基于大写标签对项目进行排序。请注意，传递给函数的参数 `a` 和 `b` 是具有 `label` 和 `data` 属性的项。

```
var my_list:mx.controls.List;

my_list.setSize(200, 100);
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.sortItems(upperCaseFunc);
function upperCaseFunc(a:Object, b:Object):Boolean {
 return (a.label.toUpperCase() > b.label.toUpperCase());
}
```

## 另请参见

[List.sortItemsBy\(\)](#)

# List.sortItemsBy()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listInstance.sortItemsBy(fieldName, optionsFlag)
```

```
listInstance.sortItemsBy(fieldName, order)
```

## 参数

*fieldName* 一个字符串，它指定要用于排序的字段的名称。通常情况下，此值为 "label" 或 "data"。

*order* 一个字符串，它指定是以升序 ("ASC") 还是以降序 ("DESC") 对项进行排序。

*optionsFlag* 允许您不必复制整个数组或者反复对其重新排序，即可对单个数组执行不同类型的多次排序。

以下是 *optionsFlag* 的可能值：

- `Array.DECENDING`，按从高到低的顺序排序。
- `Array.CASEINSENSITIVE`，忽略大小写进行排序。
- `Array.NUMERIC`，如果进行比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（如果指定该标记，字符串比较可以不区分大小写）。
- `Array.UNIQUESORT`，如果数组中有两个对象相同或者具有相同的排序字段，则返回错误代码 (0)，而不是排序后的数组。
- `Array.RETURNINDEXEDARRAY`，返回作为排序结果的整数索引数组。例如，下面的数组将返回代码的第二行，并且该数组保持不变：

```
["a", "d", "c", "b"]
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 3 和 1：

```
my_array.sort (Array.NUMERIC | Array.DECENDING)
```

## 返回

无。

## 说明

方法：按指定的顺序并使用指定的字段名称对列表中的项目进行排序。如果 *fieldName* 项既包括文本字符串也包括整数，则首先列出整数项。*fieldName* 参数通常为 "label" 或 "data"，但您可以指定任何原始数据值。

本方法是对组件中数据排序的最快方法。它还保持组件的选择状态。`sortItemsBy()` 方法之所以运行速度很快，是因为它在排序时不运行任何 **ActionScript**。`sortItems()` 方法需要运行 **ActionScript** 的比较函数，因此速度较慢。

## 示例

以下代码使用列表项的标签以升序对列表中的项进行排序：

```
var my_list:mx.controls.List;

my_list.setSize(200, 100);
my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});

my_list.sortItemsBy("label", "ASC");
```

## 另请参见

[List.sortItems\(\)](#)

# List.vPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.vPosition*

## 说明

属性：设置列表最顶部的可见项目。如果将此属性设置为一个不存在的索引号，则列表将滚动到最接近的索引。默认值为 0。



## 示例

只要滚动列表内容，以下示例就显示列表中 `vPosition` 的当前值：

```
my_list.setSize(200, 60);
my_list.rowCount = 4;
my_list.vPosition = 2;

my_list.addItem({data:"flash", label:"Flash"});
my_list.addItem({data:"flex", label:"Flex"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"fireworks", label:"Fireworks"});
my_list.addItem({data:"contribute", label:"Contribute"});
my_list.addItem({data:"breeze", label:"Breeze"});

var listListener:Object = new Object();
listListener.scroll = function(evt_obj:Object) {
 trace("my_list.vPosition = " + my_list.vPosition);
}
my_list.addEventListener("scroll", listListener);
```

# List.vScrollPolicy

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*listInstance.vScrollPolicy*

## 说明

属性；一个字符串，它确定列表是否支持垂直滚动。该属性的值可以是 "on"、"off" 或 "auto"。值 "auto" 指定在需要时显示滚动条。

## 示例

以下示例禁用列表的垂直滚动条：

```
var my_list:mx.controls.List;

my_list.setSize(200, 60);
my_list.rowCount = 4;
my_list.vScrollPolicy = "off";

my_list.addItem({data:"flash", label:"Flash"});
```

```
my_list.addItem({data:"flex", label:"Flex"});
my_list.addItem({data:"coldfusion", label:"ColdFusion"});
my_list.addItem({data:"dreamweaver", label:"Dreamweaver"});
my_list.addItem({data:"fireworks", label:"Fireworks"});
my_list.addItem({data:"contribute", label:"Contribute"});
my_list.addItem({data:"breeze", label:"Breeze"});

var listListener:Object = new Object();
listListener.scroll = function(evt_obj:Object) {
 trace("my_list.vPosition = " + my_list.vPosition);
}
my_list.addEventListener("scroll", listListener);
```

您仍然可以通过使用 `List.vPosition` 或者使用鼠标或键盘来创建滚动功能。

### 另请参见

[List.vPosition](#)

## Loader 组件

**Loader** 组件是一个容器，可以显示 **SWF** 或 **JPEG** 文件（但不能显示“渐进式”**JPEG** 文件）。您可以缩放加载器的内容，或者调整加载器自身的大小来匹配内容的大小。默认情况下，会调整内容的大小以适应加载器。您在运行时也可以加载内容，并监控加载进度（不过内容加载一次后会被缓存，所以进度会快速跳进到 100%）。

**Loader** 组件不能接收焦点。但是，**Loader** 组件中加载的内容可以接受焦点，并且可以有自己的焦点交互操作。有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

每个 **Loader** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。

可以使用“辅助功能”面板使 **Loader** 组件的内容可由屏幕读取器访问。有关更多信息，请参见《使用 Flash》中的第 19 章“创建辅助内容”。

## 使用 Loader 组件

每当需要从一个远程位置检索内容并将其拖到 **Flash** 应用程序中时，您都可以使用加载器。例如，您可以使用加载器将公司徽标（**JPEG** 文件）添加到表单。您也可以使用加载器来继承并利用已经完成的 **Flash** 作品。例如，如果您已经创建了一个 **Flash** 应用程序，但想扩展该应用程序，可以使用加载器将旧的应用程序拖到新应用程序中，或者将旧应用程序作为某个选项卡界面的一部分。再例如，您可以在显示相片的应用程序中使用加载器组件。使用 `Loader.load()`、`Loader.percentLoaded` 和 `Loader.complete` 可以在加载过程中控制图像加载的计时，并为用户显示进度栏。

如果将某些第 2 版 **Macromedia Component Architecture** 组件加载到 **SWF** 文件或 **Loader** 组件中，这些组件可能无法正常工作。这些组件包括：**Alert**、**ComboBox**、**DateField**、**Menu**、**MenuBar** 和 **Window**。

调用 `loadMovie()` 或者加载到 **Loader** 组件时，请使用 `_lockroot` 属性。如果使用了 **Loader** 组件，请添加以下代码：

```
myLoaderComponent.content._lockroot = true;
```

如果通过调用 `loadMovie()` 来使用影片剪辑，请添加以下代码：

```
myMovieClip._lockroot = true;
```

如果不在加载器影片剪辑中将 `_lockroot` 设置为 `true`，则加载器仅可以访问它自己的库，而不能访问已加载的影片剪辑中的库。

Flash Player 7 支持 `_lockroot` 属性。有关该属性的信息，请参见《ActionScript 2.0 语言参考》中的 `_lockroot (MovieClip._lockroot property)`。

`Loader`、`ScrollPane` 和 `Window` 之类的组件具有可以确定内容完成加载的时间的事件。因此，如果您要对 `Loader`、`ScrollPane` 或 `Window` 的内容设置属性，请在“complete”事件处理函数中添加属性语句，如下示例所示：

```
loadtest = new Object();
loadtest.complete = function(eventObject){
 content_mc._rotation= 45;
}
my_loader.addEventListener("complete", loadtest)
```

有关更多信息，请参见第 761 页的“[Loader.complete](#)”。

## Loader 参数

在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中，可以为每个 `Loader` 组件实例设置以下创作参数：

**autoload** 指示内容是应该自动加载 (`true`)，还是应该等到调用 `Loader.load()` 方法时再进行加载 (`false`)。默认值为 `true`。

**contentPath** 是一个绝对或相对的 URL，它指示要加载到加载器的文件。相对路径必须是相对于加载内容的 SWF 文件的路径。该 URL 必须与 Flash 内容当前驻留的 URL 在同一子域中。为了在 Flash Player 中或者在测试模式下使用 SWF 文件，必须将所有 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说明。默认值在开始加载之前为 `undefined`。

如果您在其它域中具有策略文件，则 `Loader` 可以从这些域中加载内容。请参见《学习 Flash 中的 ActionScript 2.0》中的“允许跨域数据加载”。

**scaleContent** 指示是内容进行缩放以适合加载器 (`true`)，还是加载器进行缩放以适合内容 (`false`)。默认值为 `true`。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **Loader** 组件实例设置以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 `true`。

**visible** 是一个布尔值，它指示对象是 (`true`) 否 (`false`) 可见。默认值为 `true`。

提醒

`minHeight` 和 `minWidth` 属性由内部的大小调整例程使用。它们在 `UIObject` 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以使用 **Loader** 实例的方法、属性和事件来编写 `ActionScript`，以为其设置其它选项。有关更多信息，请参见第 755 页的“[Loader 类](#)”。

## 创建具有 Loader 组件的应用程序

以下过程解释了如何在创作时将 **Loader** 组件添加到应用程序。在本例中，加载器将从一个虚拟的 URL 中加载一个徽标 JPEG。

### 创建具有 Loader 组件的应用程序：

1. 将 **Loader** 组件从“组件”面板上拖到舞台上。
2. 在“属性”检查器中，输入实例名称 **flower**。
3. 在舞台上和“组件”检查器中选择加载器，然后为 `contentPath` 参数输入 `http://www.flash-mx.com/images/image1.jpg`。

### 使用 ActionScript 创建 Loader 组件实例：

1. 将 **Loader** 组件从“组件”面板拖到库中。
2. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
this.createClassObject(mx.controls.Loader, "my_loader", 1);
my_loader.contentPath = "http://www.flash-mx.com/images/image1.jpg";
```

此脚本使用第 1254 页的“`UIObject.createClassObject()`”方法来创建 **Loader** 实例。
3. 选择“控制” > “测试影片”。

# 自定义 Loader 组件

在创作过程中和运行时，可以在水平方向和垂直方向上将 **Loader** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，则使用 `setSize()` 方法（请参见 `UIObject.setSize()`）。

**Loader** 组件的尺寸调整行为由 `scaleContent` 属性控制。当 `scaleContent` 为 `true` 时，内容会进行缩放以适合加载器的边界（并在调用 `UIObject.setSize()` 时重新进行缩放）。当 `scaleContent` 为 `false` 时，组件的大小固定为内容的大小，而 `UIObject.setSize()` 将无效。

## 使用具有 Loader 组件的样式

**Loader** 组件使用以下样式。

样式	主题	说明
<code>borderStyle</code>	光晕和范例	<b>Loader</b> 组件使用 <code>RectBorder</code> 实例作为其边框，并对该类定义的样式做出响应。请参见第 985 页的“ <code>RectBorder</code> 类”。
		默认边框样式为 <code>"none"</code> 。

例如：

```
my_ldr.setStyle("backgroundColor", 0xEEEEEE);
```

有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

## 使用具有 Loader 组件的外观

**Loader** 组件使用 `RectBorder` 的实例作为其边框（请参见第 985 页的“`RectBorder` 类”）。

# Loader 类

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > View > Loader


ActionScript 类名称 mx.controls.Loader

Loader 类的属性允许您设置要加载的内容并在运行时监视它的加载进度。

使用 **ActionScript** 设置 **Loader** 类的属性将会覆盖在 “属性” 检查器或 “组件” 检查器中设置的同名参数。

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.Loader.version);
```

 代码 `trace(myLoaderInstance.version);` 返回 `undefined`。

## Loader 类的方法摘要

下表列出了 **Loader** 类的方法。

方法	说明
<a href="#">Loader.load()</a>	加载由 <code>contentPath</code> 属性指定的内容。

## 从 UIObject 类继承的方法

下表列出了 **Loader** 类从 **UIObject** 类继承的方法。从 **Loader** 对象调用这些方法时，请使用 `LoaderInstance.methodName` 的形式。

方法	说明
<a href="#">UIObject.createClassObject()</a>	创建指定类的对象。
<a href="#">UIObject.createObject()</a>	创建对象的子对象。
<a href="#">UIObject.destroyObject()</a>	破坏组件实例。
<a href="#">UIObject.doLater()</a>	在 “属性” 检查器和 “组件” 检查器中设置了参数之后，调用一个函数。
<a href="#">UIObject.getStyle()</a>	从样式声明或对象获取样式属性。
<a href="#">UIObject.invalidate()</a>	标记对象使其在到达下一个帧间隔时进行重绘。
<a href="#">UIObject.move()</a>	将对象移动到要求的位置。
<a href="#">UIObject.redraw()</a>	迫使对象有效以便能在当前帧中绘制。

方法	说明
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **Loader** 类从 **UIComponent** 类继承的方法。从 **Loader** 对象调用这些方法时，请使用 `LoaderInstance.methodName` 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## Loader 类的属性摘要

下表列出了 **Loader** 类的属性。

属性	说明
<code>Loader.autoLoad</code>	一个布尔值，它指示内容是自动加载 ( <code>true</code> ) 还是必须调用 <code>Loader.load()</code> 才能加载 ( <code>false</code> )。
<code>Loader.bytesLoaded</code>	只读属性，指明已经加载的字节数。
<code>Loader.bytesTotal</code>	指明内容中的总字节数的只读属性。
<code>Loader.content</code>	对加载器内容的引用。该属性为只读。
<code>Loader.contentPath</code>	一个字符串，它指明要加载的内容的 URL。
<code>Loader.percentLoaded</code>	一个数字，它指明已加载内容的百分比。该属性为只读。
<code>Loader.scaleContent</code>	一个布尔值，它指示是内容会进行缩放以适合加载器 ( <code>true</code> )，还是加载器会进行缩放以适合内容 ( <code>false</code> )。



## 从 UIObject 类继承的属性

下表列出了 **Loader** 类从 **UIObject** 类继承的属性。从 **Loader** 对象访问这些属性时，请使用 *LoaderInstance.propertyName* 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 **Loader** 类从 **UIComponent** 类继承的属性。从 **Loader** 对象访问这些属性时，请使用 *LoaderInstance.propertyName* 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

# Loader 类的事件摘要

下表列出了 Loader 类的事件。

事件	说明
<code>Loader.complete</code>	当内容加载完成时触发。
<code>Loader.progress</code>	在内容加载过程中触发。

## 从 UIObject 类继承的事件

下表列出了 Loader 类从 UIObject 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 Loader 类从 UIComponent 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

# Loader.autoLoad

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance*.autoLoad

## 说明

属性；一个布尔值，它指示是自动加载内容 (true)，还是等到调用 `Loader.load()` 时才加载内容 (false)。默认值为 true。

## 示例

以下代码将加载器组件设置为等待调用 `Loader.load()`：

```
loader.autoLoad = false;
```

# Loader.bytesLoaded

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance*.bytesLoaded

## 说明

属性（只读）；已经加载的内容的字节数目。在开始加载内容之前，默认值为 0。

## 示例

以下代码通过当前文档的库中的 **Loader** 组件和 **ProgressBar** 组件创建进度条和加载器实例。然后，代码创建一个具有 progress 事件处理函数的侦听器对象，该事件处理函数会显示加载的进度。该侦听器将被注册到 my\_ldr 实例。

使用 createClassObject() 创建实例时，必须使用 move() 方法将其放置在舞台上。请参见 [UIObject.move\(\)](#)。

```
import mx.controls.Loader;
import mx.controls.ProgressBar;

System.security.allowDomain("http://www.flash-mx.com");

this.createClassObject(Loader, "my_ldr", 10);
this.createClassObject(ProgressBar, "my_pb", 20, {source:"my_ldr"});

my_ldr.move(1, 50);
my_pb.move(1, 1);

var loaderListener:Object = new Object();
loaderListener.progress = function(evt_obj:Object) {
 // evt_obj.target 为生成 progress 事件的组件，
 // 即加载器。
 my_pb.setProgress(my_ldr.bytesLoaded, my_ldr.bytesTotal);
 // 显示进度。
};
my_ldr.addEventListener("progress", loaderListener);
my_ldr.contentPath = "http://www.flash-mx.com/images/image2.jpg";
```

# Loader.bytesTotal

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance*.bytesTotal

## 说明

属性（只读）：内容的大小，以字节为单位。在内容开始加载前，默认值为 0。

## 示例

以下代码创建一个进度栏和一个 **Loader** 组件。然后，代码创建一个具有 `progress` 事件处理函数的加载侦听器对象，该事件处理函数会显示加载的进度。侦听器将被注册到 `my_ldr` 实例，如下所示：

```
import mx.controls.Loader;
import mx.controls.ProgressBar;
this.createClassObject(ProgressBar, "my_pb", 998);
this.createClassObject(Loader, "my_ldr", 999);
my_pb.move(1, 1);
my_ldr.move(1, 50);
my_pb.source = "my_ldr";
var loadListener:Object = new Object();
loadListener.progress = function(eventObj){
 // eventObj.target 为生成进度事件的组件，
 // 即加载器。
 my_pb.setProgress(my_ldr.bytesLoaded, my_ldr.bytesTotal); // 显示进度。
}
my_ldr.addEventListener("progress", loadListener);
my_ldr.contentPath = "http://www.flash-mx.com/images/image2.jpg";
```

## 另请参见

[Loader.bytesLoaded](#)

# Loader.complete

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObj:Object){
 // ...
};
loaderInstance.addEventListener("complete", listenerObject);
```

用法 2：

```
on (complete) {
 // ...
}
```

## 说明

事件：当内容加载完成时向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*loaderInstance*) 调度一个事件（在本示例中为 *complete*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **Loader** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **Loader** 实例 `myLoaderComponent` 上，并将“\_level0.myLoaderComponent”发送到“输出”面板：

```
on (complete) {
 trace(this);
}
```

## 示例

以下示例创建一个 **Loader** 组件 `my_ldr`，然后为 *complete* 事件定义一个侦听器对象。该示例从网页加载一个图像；当加载完成后，侦听器将在“输出”面板中显示一条消息。

将一个 **Loader** 组件拖到库中，然后向时间轴的第一帧添加以下代码。

```
/**
 要求：
 - 库中有 Loader 组件。
*/

System.security.allowDomain("http://www.flash-mx.com");

// 创建加载器实例。
this.createClassObject(mx.controls.Loader, "my_ldr", 10);

// 创建侦听器对象。
var loaderListener:Object = new Object();
loaderListener.complete = function(evt_obj:Object){
 trace("loading complete");
}

// 添加侦听器。
my_ldr.addEventListener("complete", loaderListener);
my_ldr.load("http://www.flash-mx.com/images/image2.jpg");
```

# Loader.content

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance.content*

## 说明

属性（只读）：对包含所加载文件的内容的影片剪辑实例的引用。该值在开始加载之前为 undefined。请为 Loader.complete 事件的事件处理函数中的内容设置属性。

## 示例

**Loader** 组件有一个 “complete” 事件，因此，在尝试访问加载器内容的属性之前，即可确定内容是否已完全加载。

以下示例将在 **complete** 事件的事件处理函数中使用 Loader.content 属性。将 **Loader** 组件从“组件”面板拖到当前文档的库中，使该组件显示在库中。然后向主时间轴的第一帧中添加以下 **ActionScript** 代码：

```
this.createClassObject(mx.controls.Loader, "my_ldr", 10);
my_ldr.contentPath = "http://www.flash-mx.com/images/image1.jpg";
// 给内容分配一个变量。
var content_mc:MovieClip = my_ldr.content;

var loadtest:Object = new Object();
loadtest.complete = function(){
 // 设置内容的属性。
 content_mc._alpha = 50;
 content_mc._rotation = 45;
 trace(content_mc._width);
}
my_ldr.addEventListener("complete", loadtest);
```

# Loader.contentPath

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance*.contentPath

## 说明

属性：一个字符串，指明要加载到 **loader** 中的文件的绝对 URL 或相对 URL。相对路径必须相对于加载内容的 SWF 文件。该 URL 必须与正加载的 SWF 文件位于同一子域中。

如果您要使用 **Flash Player**，或者在 **Flash** 中使用测试模式，则必须将所有 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信息。

## 示例

以下示例指示加载器实例显示 **logo.swf** 文件的内容：

```
flower.contentPath = "http://www.flash-mx.com/images/image1.jpg"
```

以下示例在单击 **Button** 实例 **my\_btn** 时从 **Loader** 中卸载内容：

```
flower.contentPath = "http://www.flash-mx.com/images/image1.jpg"
function clicked(){
 flower.contentPath = "";
}
my_btn.addEventListener("click", clicked);
```

# Loader.load()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance*.load([*path*])



## 参数

*path* 可选参数，它指定在开始加载之前 *contentPath* 属性的值。如果未指定值，则使用 *contentPath* 的当前值。

## 返回

无。

## 说明

方法：指示 **loader** 开始加载其内容。

## 示例

以下示例将创建一个 **Loader** 实例 *my\_ldr* 和一个 **Button** 实例，并将加载器 *autoload* 属性设置为 *false*，以便直到调用 *load()* 方法时才开始加载。接着，该示例将 *contentPath* 设置为一个图像的网络位置，并对该按钮创建 *click* 事件侦听器。当用户单击按钮时，事件处理函数会调用 *my\_ldr.load()* 以加载该图像。该事件处理函数还可以禁用该按钮。

将一个 **Loader** 组件和一个 **Button** 组件从“组件”面板拖到库中，然后向时间轴的第一帧中添加以下代码。

```
/**
 要求：
 - 库中有 Loader 组件。
 - 库中有 Button 组件。
*/

System.security.allowDomain("http://www.flash-mx.com");

// 创建加载器实例。
this.createClassObject(mx.controls.Loader, "my_ldr", 10);
this.createClassObject(mx.controls.Button, "load_button", 20, {label:"Load
 image"});

my_ldr.move(0, 30);

my_ldr.autoload = false;
my_ldr.contentPath = "http://www.flash-mx.com/images/image1.jpg";

var loadListener:Object = new Object();
loadListener.click = function (evt_obj:Object) {
 my_ldr.load();
 load_button.enabled = false;
}
load_button.addEventListener("click", loadListener);
```

# Loader.percentLoaded

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance.percentLoaded*

## 说明

属性（只读）：一个数字，指明已加载内容的百分比。通常，使用此属性是为了以一种易于阅读的形式向用户展示进度。使用以下代码将数字舍入为最接近的整数：

```
Math.round(bytesLoaded/bytesTotal*100))
```

## 示例

以下示例将创建一个 **Loader** 实例，然后创建一个带有 progress 处理函数的侦听器对象，该处理函数跟踪已加载的百分比并将其发送到“输出”面板：

```
import mx.controls.Loader;
this.createClassObject(Loader, "my_ldr", 999);
var loadListener:Object = new Object();
loadListener.progress = function(eventObj) {
 // eventObj.target 为生成进度事件的组件，
 // 即加载器。
 trace("The image is "+my_ldr.percentLoaded+"% loaded.");
 // 跟踪加载进度。
};
my_ldr.addEventListener("progress", loadListener);
my_ldr.contentPath = "http://www.flash-mx.com/images/image2.jpg";
```

# Loader.progress

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObj:Object) {
 // ...
};
loaderInstance.addEventListener("progress", listenerObject);
```

用法 2:

```
on (progress) {
 // ...
}
```

## 说明

事件：在加载内容时向所有已注册的侦听器广播。当 `autoload` 参数或对 `Loader.load()` 的调用触发加载操作时，会发生此事件。**Progress** 事件并不会始终广播；`complete` 事件可能在未调度任何 `progress` 事件的情况下广播。如果加载的内容是本地文件，就会出现这种情况。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (`loaderInstance`) 调度一个事件（在本示例中为 `progress`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **Loader** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **Loader** 实例 `myLoaderComponent` 上，并将“\_level0.myLoaderComponent”发送到“输出”面板：

```
on (progress) {
 trace(this);
}
```

## 示例

以下代码创建一个 **Loader** 实例，然后为 `progress` 事件创建一个带有事件处理函数的侦听器对象，该处理函数会将一条关于已加载内容的百分比的消息发送到“输出”面板：

```
// 创建加载器实例。
this.createClassObject(mx.controls.Loader, "my_ldr", 10);

// 创建侦听器对象。
var loaderListener:Object = new Object();
loaderListener.progress = function(evt_obj:Object){
 // evt_obj.target 为生成 progress 事件的组件，
 // 即加载器。
 trace("image is " + my_ldr.percentLoaded + "% loaded.");
}

// 添加侦听器。
my_ldr.addEventListener("progress", loaderListener);

// 指定加载器的内容路径。
my_ldr.contentPath = "http://www.flash-mx.com/images/image1.jpg";
```

# Loader.scaleContent

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*loaderInstance*.scaleContent

## 说明

属性：指示是内容进行缩放以适合加载器（**true**），还是加载器进行缩放以适合内容（**false**）。默认值为 **true**。

## 示例

以下代码指示加载器调整自身的大小以匹配内容的大小：

```
my_ldr.scaleContent = false;
```

# 媒体组件（仅限 Flash Professional）

使用流媒体组件能够很方便地将流媒体加入到 **Macromedia Flash** 演示文稿中。这些组件可让您以多种方法显示媒体。

可使用以下三种媒体组件：

- **MediaDisplay** 组件使媒体可以流入到 **Flash** 内容中，而不需要支持的用户界面。此组件可以用于处理视频和音频数据。单独使用此组件时，用户将无法控制媒体。
- **MediaController** 组件为媒体回放提供标准的用户界面控件（播放、暂停等等）。媒体绝不会加载到此组件中或由此组件播放；它仅用于控制 **MediaPlayback** 或 **MediaDisplay** 实例中的回放。**MediaController** 组件起到“抽屉”的作用，它会在鼠标位于组件上方时显示播放控件的内容。
- **MediaPlayback** 组件是 **MediaDisplay** 和 **MediaController** 组件的结合；它提供对媒体内容进行流式处理的方法。

请记住关于媒体组件的以下要点：

- 媒体组件需要 **Flash Player 6** 或更高版本。在 **Flash Player 6** 中，媒体组件只通过 **Flash Communication Server** 而不通过 **HTTP** 来支持 **FLV** 文件。
- 媒体组件不支持向前扫描和向后扫描功能。不过，您可以通过移动回放滑块得到此功能。
- 只有组件大小和控制器策略会在实时预览中反映出来。
- 媒体组件不支持辅助功能。

# 与媒体组件进行交互（仅限 Flash Professional）

流式 `MediaPlayback` 和 `MediaController` 组件会响应鼠标和键盘活动；而 `MediaDisplay` 组件不会。下表总结了 `MediaPlayback` 和 `MediaController` 组件在接收到焦点后的动作：

目标	导航	说明
给定控制器的播放控件	鼠标滑过	按钮加亮显示。
给定控制器的播放控件	单击鼠标左键	用户可以单击回放控件来控制音频和视频媒体的回放。 “暂停” / “播放”和“转到开头” / “转到结尾”按钮的行为与标准按钮相同。按下鼠标按钮时，屏幕上的按钮会加亮显示为其被按下状态，当松开鼠标按钮时，屏幕上的按钮会回到未选中的状态。在播放 FLV 媒体文件时，“转到结尾”按钮被禁用。
给定控制器的滑块控件	前后移动滑块	播放栏指示用户在媒体中的位置；回放滑块会水平（默认）移动，以便从开头（左）到结尾（右）指示回放。垂直摆放控件时，滑块从下往上移动。当滑块从左向右移动时，它会加亮显示上一个显示区，以指示此内容已回放或已选中。滑块前面的显示区会保持非加亮显示状态，直到滑块移过。用户可以拖动滑块，以改变媒体的回放位置。如果媒体正在播放，会从松开鼠标的点开始自动回放。如果媒体暂停，用户可移动并松开滑块，媒体会保持暂停。 此外还提供了一个音量滑块，可以在水平和垂直布局下从左（静音）向右（最大音量）移动。
播放控制器导航	Tab 键， Shift+Tab 键	使焦点在控制器组件内的按钮之间移动，获得焦点的元素将加亮显示。此导航适用于“暂停” / “播放”、“转到开头”、“转到结尾”、“静音”和“最大音量”控件。当用户按 tab 键在元素之间移动时，焦点会从左到右、从上到下移动。按 Shift+Tab 键会将焦点从右到左、从下到上移动。通过 Tab 键接收焦点后，控件立即将焦点传给“播放” / “暂停”按钮。当焦点位于“最大音量”按钮上且按下 Tab 键时，焦点会移动到舞台上 Tab 键索引中的下一个控件。
给定的控件按钮	空格键或 Enter/Return	选择具有焦点的元素。按下后，按钮会显示为其被按下状态。松开后，按钮会回复其具有焦点、鼠标移过的状态。

# 了解媒体组件（仅限 Flash Professional）

本节提供了有关媒体组件如何进行工作的概述。本节中列出的大多数属性都可以通过“组件”检查器设置。（请参见第 776 页的“使用“组件”检查器设置媒体组件”。）

除了本节稍后讨论的布局属性，可以为 `MediaDisplay` 和 `MediaPlayerback` 组件设置以下属性：

- 媒体类型，可以设置为 MP3 或 FLV（请参见 `Media.mediaType` 和 `Media.setMedia()`）。
- 相对或绝对内容路径，它保存了要进行流式处理的媒体文件（请参见 `Media.contentPath`）。
- 提示点对象，及其名称、时间和播放器属性（请参见 `Media.addCuePoint()` 和 `Media.cuePoints`）。可以任意设置提示点的名称；在使用侦听器 and 跟踪事件时应使用一个有意义的名称。在提示点的时间属性值等于它所关联的 `MediaPlayerback` 或 `MediaDisplay` 组件的播放头位置值时，提示点将广播 `cuePoint` 事件。播放器属性是对其所关联的 `MediaPlayerback` 实例的引用。可以通过使用 `Media.removeCuePoint()` 和 `Media.removeAllCuePoints()` 来删除提示点。

媒体流组件可广播多个相关事件。以下广播事件可以用于设置动画中的其它项目：

- `change` 事件在播放媒体时由 `MediaDisplay` 和 `MediaPlayerback` 组件连续广播。（请参见 `Media.change`。）
- `progress` 事件在加载媒体时由 `MediaDisplay` 和 `MediaPlayerback` 组件连续广播。（请参见 `Media.progress`。）
- `click` 事件在用户每次单击“暂停”/“播放”按钮时由 `MediaController` 和 `MediaPlayerback` 组件广播。在这种情况下，事件对象的 `detail` 属性会提供关于所单击按钮的信息。（请参见 `Media.click`。）
- `volume` 事件在用户调节音量控件时由 `MediaController` 和 `MediaPlayerback` 组件广播。（请参见 `Media.volume`。）
- 当用户移动回放滑块或者单击“转到开头”或“转到结尾”按钮时，`playheadChange` 事件将由 `MediaController` 和 `MediaPlayerback` 组件广播。（请参见 `Media.playheadChange`。）

`MediaDisplay` 组件与 `MediaController` 组件配合使用。配合使用时，这两个组件的行为方式类似于 `MediaPlayerback` 组件，但它们在媒体演示文稿的外观和行为方面提供了更多的灵活性。

## 了解 MediaPlayer 组件

当将 MediaPlayer 组件放置到舞台上时，该组件没有用户界面。它只是一个容纳和播放媒体的容器。以下属性影响在 MediaPlayer 组件中播放的视频媒体的外观：

- `Media.aspectRatio`
- `Media.autoSize`
- 高度（在“属性”检查器中）
- 宽度（在“属性”检查器中）



除非正在播放媒体，否则用户看不到任何内容。

`Media.aspectRatio` 属性优先于其它属性。当 `Media.aspectRatio` 设置为 `true`（默认值）时，该组件总是重新调整正在播放的媒体的大小来保持媒体的高宽比。

对于 FLV 文件，当 `Media.autoSize` 设置为 `true` 时，无论组件的大小如何，媒体都将其首选大小显示。这意味着，如果 MediaPlayer 实例的大小与该媒体的大小不同，则该媒体将溢出实例边界或无法填充实例大小。当 `Media.autoSize` 设置为 `false` 时，Flash 将会尽可能地使用实例大小，同时保持高宽比。如果 `Media.autoSize` 和 `Media.aspectRatio` 都设置为 `false`，则使用该组件的准确大小。



由于不存在可随 MP3 文件显示的图像，因此，设置 `Media.autoSize` 时不会有任何效果。对于 MP3 文件，最小的可用大小是 60 像素高乘以 256 像素宽（在默认的方向中）。

MediaPlayer 组件还支持 `Media.volume` 属性。此属性的值为介于 0（静音）和 100（最大音量）之间的整数值。默认设置为 75。

## 了解 MediaController 组件

MediaController 组件的界面取决于它的 `Media.controllerPolicy` 和 `Media.backgroundColor` 属性。`Media.controllerPolicy` 属性确定媒体控件集是始终可见、折叠还是仅当鼠标悬停在组件的控件部分时可见。折叠时，控制器会绘制一个经修改的进度栏，它是加载栏和播放栏的组合。它在栏底部显示加载的字节进度，并在其上面显示播放头的进度。处于展开状态时，控制器会绘制增强版的播放栏 / 加载栏，包含以下项目：

- 指示回放状态（正进行流式处理或暂停）的左侧文本标签，以及指示播放头位置（以秒为单位）的右侧文本标签
- 播放头位置指示器
- 滑块，用户可以拖动它在媒体中导航



MediaController 组件还提供了以下项目：

- “播放” / “暂停” 按钮
- “转到开头” 和 “转到结尾” 按钮，分别导航到媒体的开头和结尾
- 音量控件，由一个滑块、一个静音按钮和一个最大音量按钮组成

MediaController 组件的折叠和展开状态都使用 `Media.backgroundStyle` 属性。此属性确定控制器是绘制铬印染背景（默认值），还是允许媒体从控件后面显示媒体背景。

MediaController 组件有一个方向设置 (`Media.horizontal`)，可用于以水平方向（默认值）或垂直方向绘制组件。在水平方向下，播放栏从左到右跟踪播放的媒体。在垂直方向下，播放栏从下到上跟踪媒体。

可以通过使用 `Media.associateDisplay()` 和 `Media.associateController()` 方法使 MediaDisplay 与 MediaController 组件互相关联。这些方法允许 MediaController 实例根据来自 MediaDisplay 实例的事件广播更新前者的控件，并允许 MediaDisplay 组件响应 MediaController 中的用户设置。

## 了解 MediaPlayer 组件

MediaPlayer 包含 MediaController 和 MediaDisplay 子组件。MediaController 和 MediaDisplay 部分始终会根据 MediaPlayer 实例的整体大小进行缩放。

MediaPlayer 组件使用 `Media.controlPlacement` 确定控件的布局。通过将此属性设置为 top、bottom、left 或 right，可以指示控件的绘制位置（相对于屏幕）。例如，right 值使控件保持垂直方向，并将其放置在屏幕的右侧。

## 使用媒体组件（仅限 Flash Professional）

随着使用媒体向 Web 用户提供信息的案例的迅速增多，许多开发人员希望他们的用户能够对媒体进行流式处理然后加以控制。您可以在以下各种情况下使用媒体组件：

- 显示介绍公司的媒体
- 对影片或影片预览进行流式处理
- 对歌曲或歌曲片段进行流式处理
- 提供媒体形式的学习材料

## 使用 MediaPlayer 组件

假设必须开发一个 Web 站点，使用户可以在丰富媒体环境下预览您销售的 DVD 和 CD。以下示例显示涉及的步骤。（它假定您的 Web 站点已做好插入流组件的准备。）

### 创建显示 CD 或 DVD 预览效果的 Flash 文档：

1. 选择“文件”>“新建”，然后选择“Flash 文档”。
2. 打开“组件”面板并双击 MediaPlayer 组件，这会将该组件的一个实例放置到舞台上。
3. 选择该 MediaPlayer 组件实例，并在“属性”检查器中输入实例名称 **myMedia**。
4. 在“组件”检查器中，根据要进行流式处理的媒体的类型设置媒体类型（MP3 或 FLV）。
5. 如果选择了 FLV，请在“Video length”文本框中输入视频的持续时间；请使用 HH:MM:SS 格式。
6. 在“URL”文本框中输入视频的预览位置。例如，您可以输入 **www.helpexamples.com/flash/video/clouds.flv**。
7. 为“Automatically Play”、“Use Preferred Media Size”和“Respect Aspect Ratio”复选框设置所需选项。
8. 将控件位置设置到所需的 MediaPlayer 组件一侧。
9. 通过单击“添加”（+）按钮向媒体结尾添加一个提示点；此提示点将与侦听器一起使用，用于打开一个弹出式窗口，通知用户以影片出售。为提示点指定名称 **cuePointName** 以及媒体持续时间结尾附近的位置。
10. 将 Window 组件从“组件”面板拖到当前文档的库中。  
这会将名为 Window 的元件放置到库中，并使 Window 组件在运行时可供 SWF 文件使用。
11. 创建一个文本框并写入一些文本，以通知用户此影片出售。
12. 选择“修改”>“转换为元件”，以将此文本框转换为影片剪辑，然后将其命名为 **mySale\_mc**。
13. 右击 (Windows) 或者按住 Ctrl 键单击 (Macintosh) 库中的 mySale\_mc 影片剪辑，再选择“链接”，然后选择“为 ActionScript 导出”。  
这会将影片剪辑放置到您的运行时库中。
14. 将以下 ActionScript 添加到第 1 帧。此代码会创建一个侦听器，用于打开一个通知用户此影片出售的弹出式窗口。

```
// 导入动态创建弹出式窗口所需的类。
```

```
import mx.containers.Window;
import mx.managers.PopUpManager;
```

```
// 创建一个打开销售弹出式窗口的侦听器对象。
```

```

var saleListener:Object = new Object();

saleListener.cuePoint = function(eventObj:Object) {

var saleWin:MovieClip = PopUpManager.createPopUp(_root, Window, false,
 {closeButton:true, title:"Movie Sale", contentPath:"mySale_mc"});

// 放大窗口以便使内容与之相适应。

saleWin.setSize(80, 80);
var delSaleWin:Object = new Object();
delSaleWin.click = function(eventObj:Object) {
 saleWin.deletePopUp();
}
saleWin.addEventListener("click", delSaleWin);

}

myMedia.addEventListener("cuePoint", saleListener);

```

15. 选择“控制” > “测试影片”以测试 SWF 文件。

当应用程序到达 `cuePointName` 提示点的回放时间时，将弹出一个窗口显示您的消息。

## 使用 MediaDisplay 和 MediaController 组件

如果要对媒体显示的外观和行为进行较大程度的控制，您可能需要同时使用 `MediaDisplay` 和 `MediaController` 组件。以下示例创建一个显示 CD 和 DVD 预览媒体的 Flash 应用程序。

**创建显示 CD 或 DVD 预览效果的 Flash 文档：**

1. 在 Flash 中，选择“文件” > “新建”，然后选择“Flash 文档”。
2. 从“组件”面板中，将一个 `MediaController` 组件和一个 `MediaDisplay` 组件拖到舞台上。
3. 选择 `MediaDisplay` 实例并在“属性”检查器中输入实例名称 **myDisplay**。
4. 选择 `MediaController` 实例并在“属性”检查器中输入实例名称 **myController**。
5. 选择 `MediaDisplay` 实例，然后打开“组件”检查器的“参数”选项卡。请根据要进行流式处理的媒体的类型设置媒体类型（MP3 或 FLV）。
6. 如果选择了 FLV，请使用 HH:MM:SS 格式在“Video Length”文本框中输入视频的持续时间。
7. 在“URL”文本框中输入视频的预览位置。例如，您可以输入 **www.helpexamples.com/flash/video/clouds.flv**。
8. 为“Automatically Play”、“Use Preferred Media Size”和“Respect Aspect Ratio”复选框设置所需选项。

9. 选择 **MediaController** 实例，并在“组件”检查器的“参数”选项卡中将方向设置为垂直，方法是将 `horizontal` 属性设置为 `false`。
  10. 在“组件”检查器的“参数”选项卡中，将 `backgroundStyle` 设置为 `none`。  
这样会指定 **MediaController** 实例不应绘制背景，而应在控件之间填充媒体。  
接着，使用一个行为关联 **MediaController** 和 **MediaDisplay** 实例，使 **MediaController** 实例准确地反映 **MediaDisplay** 实例中的播放头移动和其它设置，并使 **MediaDisplay** 实例响应用户的单击。
  11. 在 **MediaController** 实例仍然保持选中的情况下，打开“行为”面板（“窗口” > “行为”）。
  12. 在“行为”面板中，单击“添加” (+) 按钮并选择“媒体” > “关联显示”。
  13. 在“关联显示”窗口中，选择 `_root` 下的 `myDisplay`，然后单击“确定”。
- 有关使用行为控制媒体组件的更多信息，请参见第 777 页的“通过使用行为控制媒体组件”。

## 使用“组件”检查器设置媒体组件

“组件”检查器使用户能够轻松地设置媒体组件的参数、属性等等。要使用此面板，请在舞台上单击所需的组件，并在“属性”检查器打开的情况下，单击“启动组件检查器”。“组件”检查器可用于以下用途：

- 自动播放媒体（请参见 `Media.activePlayControl` 和 `Media.autoPlay`）
- 保持或忽略媒体的高宽比（请参见 `Media.aspectRatio`）
- 确定媒体是否将自动调整大小以适合组件实例（请参见 `Media.autoSize`）
- 启用或禁用铬印染背景（请参见 `Media.backgroundStyle`）
- 以 URL 的形式指定媒体的路径（请参见 `Media.contentPath`）
- 指定回放控件的可见性（请参见 `Media.controllerPolicy`）
- 添加提示点对象（请参见 `Media.addCuePoint()`）
- 删除提示点对象（请参见 `Media.removeCuePoint()`）
- 设置 **MediaController** 实例的方向（请参见 `Media.horizontal`）
- 设置所播放的媒体的类型（请参见 `Media.setMedia()`）
- 设置 FLV 媒体的播放时间（请参见 `Media.totalTime`）
- 设置时间显示的最后几位数，以指示毫秒数或每秒帧数 (fps)

使用“组件”检查器时，了解有些概念非常重要：

- 当选择 MP3 视频类型时，视频时间控件不可用，原因是此信息会在使用 MP3 文件时自动读入。对于使用 Flash Video Exporter 1.0 创建的 FLV 文件，必须输入媒体的总时间 (`Media.totalTime`)，以便使 `MediaPlayback` 组件（或任何用于侦听的 `MediaController` 组件）的播放栏能准确地反映播放进度。使用 Flash Video Exporter 1.1 或更高版本创建的 FLV 文件会自动设置持续时间。
- 文件类型设置为 FLV 时，您将观察到一个“毫秒”选项和（如果取消选择“毫秒”）“每秒的帧” (FPS) 弹出式菜单。当选择“毫秒”时，FPS 控件不可见。在此模式下，运行时在播放栏中显示时间的格式为 HH:MM:SS.mmm（H = 小时、M = 分钟、S = 秒、m = 毫秒），而且提示点以秒为单位设置。当取消选择“毫秒”时，FPS 控件即会启用，且播放栏时间的格式为 HH:MM:SS.FF（F = 每秒的帧数），而提示点以帧为单位设置。



只能使用“组件”检查器设置 FPS 值。使用 ActionScript 设置 fps 值是无效的，而且会被忽略。

## 通过使用行为控制媒体组件

行为是预先编写的 ActionScript，为了控制该对象而添加到实例（例如 `MediaDisplay` 组件）中。行为使您可以将 ActionScript 编码的强大功能、控制能力以及灵活性添加到文档中，而不必自己创建 ActionScript 代码。

要通过行为控制媒体组件，请使用“行为”面板将行为应用到给定的媒体组件实例。请指定将会触发行为的事件（例如到达指定的提示点）、选择目标对象（将受行为影响的媒体组件），并在必要时选择行为的设置（例如要导航到的媒体内的影片剪辑）。

以下行为随附在 Flash Professional 8 中，用于控制嵌入的媒体组件。

行为	目的	参数
关联控制器	将 <code>MediaController</code> 组件与 <code>MediaDisplay</code> 组件关联	目标 <code>MediaController</code> 组件的实例名称
关联显示	将 <code>MediaDisplay</code> 组件与 <code>MediaController</code> 组件关联	目标 <code>MediaController</code> 组件的实例名称
指定帧提示点导航	将动作放到 <code>MediaDisplay</code> 或 <code>MediaPlayback</code> 实例上，通知指定的影片剪辑导航到与给定的提示点名称相同的帧	帧的名称和提示点的名称（它们的名称应该相同）
幻灯片提示点导航	使基于幻灯片的 Flash 文档导航到与给定的提示点名称相同的幻灯片	幻灯片的名称和提示点的名称（它们的名称应该相同）

### 将 MediaDisplay 组件与 MediaController 组件关联：

1. 将 MediaDisplay 实例和 MediaController 实例放置在舞台上。
2. 选择 MediaDisplay 实例，并使用“属性”检查器输入实例名称 **myMediaDisplay**。
3. 选择将触发该行为的 MediaController 实例。
4. 在“行为”面板中，单击“添加” (+) 按钮并选择“媒体” > “关联显示”。
5. 在“关联显示”窗口中，选择 \_root 下的 myMediaDisplay，然后单击“确定”。



如果已将 MediaDisplay 组件与 MediaController 组件关联，则不需要将 MediaController 组件与 MediaDisplay 组件关联。

### 将 MediaController 组件与 MediaDisplay 组件关联：

1. 将 MediaDisplay 实例和 MediaController 实例放置在舞台上。
2. 选择 MediaController 实例，并使用“属性”检查器输入实例名称 **myMediaController**。
3. 选择将触发该行为的 MediaDisplay 实例。
4. 在“行为”面板中，单击“添加” (+) 按钮并选择“媒体” > “关联控制器”。
5. 在“关联控制器”窗口中，选择 \_root 下的 myMediaController，然后单击“确定”。

### 使用“指定帧提示点导航”行为：

1. 将 MediaDisplay 或 MediaPlayer 组件实例放置在舞台上。
2. 选择希望媒体导航到的帧，并使用“属性”检查器输入帧的名称 **myLabeledFrame**。
3. 选择 MediaDisplay 或 MediaPlayer 实例。
4. 在“组件”检查器中，单击“添加” (+) 按钮并以 HH:MM:SS:mmm 或 HH:MM:SS:FF 格式输入提示点时间，然后为提示点指定名称 **myLabeledFrame**。

提示点指示在导航到选定的帧之前应经过的时间量。例如，如果要跳至 myLabeledFrame（媒体中的第 5 秒位置），请在 SS 文本框中输入 **5**，并在“名称”文本框中输入 **myLabeledFrame**。

5. 在“行为”面板中，单击“添加” (+) 按钮并选择“媒体” > “指定帧提示点导航”。
6. 在“指定帧提示点导航”窗口中，选择 \_root 剪辑并单击“确定”。

使用“幻灯片提示点导航”行为：

1. 打开新的 Flash 幻灯片演示文稿文档。
2. 将 `MediaDisplay` 或 `MediaPlayback` 组件实例放置在舞台上。
3. 在舞台左侧的“屏幕轮廓”窗格中，单击“插入屏幕” (+) 按钮以添加另一个幻灯片，然后选择第二张幻灯片并重命名为 **mySlide**。
4. 选择 `MediaDisplay` 或 `MediaController` 实例。
5. 在“组件”检查器中，单击“添加” (+) 按钮并以 `HH:MM:SS:mmm` 或 `HH:MM:SS:FF` 格式输入提示点时间，然后为提示点指定名称 **MySlide**。  
提示点指示在导航到选定的幻灯片之前应经过的时间量。例如，如果要跳至 `mySlide`（媒体中的第 5 秒位置），请在 `SS` 文本框中输入 **5**，并在“名称”文本框中输入 **mySlide**。
6. 在“行为”面板中，单击“添加” (+) 按钮并选择“媒体” > “幻灯片提示点导航”。
7. 在“幻灯片提示点导航”窗口中，选择 `_root` 剪辑下的“演示文稿”，然后单击“确定”。

# 媒体组件参数（仅限 Flash Professional）

以下各表列出了可以在“属性”检查器中为给定的媒体组件实例设置的创作参数 `MediaDisplay`、`MediaController` 和 `MediaPlayback`。

## MediaDisplay 参数

名称	类型	默认值	说明
自动播放 ( <code>Media.autoPlay</code> )	布尔值	Selected	确定是否在加载媒体后立刻播放该媒体。
使用首选媒体大小 ( <code>Media.autoSize</code> )	布尔值	Selected	确定与 <code>MediaDisplay</code> 实例关联的媒体是符合组件大小，还是仅使用其默认的大小。
FPS	整数	30	指示每秒的帧数。当选择了“毫秒”选项时，此控件禁用。
提示点 ( <code>Media.cuePoints</code> )	数组	Undefined	由提示点对象组成的数组，这些对象各自具有一个名称和时间位置，有效时间格式为 <code>HH:MM:SS:FF</code> （选择了“毫秒”选项时）或 <code>HH:MM:SS:mmm</code> 格式。
FLV 或 MP3 ( <code>Media.mediaType</code> )	FLV 或 MP3	FLV	指定要播放的媒体类型。
Milliseconds	布尔值	Unselected	确定播放栏是使用帧还是毫秒，以及提示点是使用秒还是帧。当选择此选项时，FPS 控件不可见。

名称	类型	默认值	说明
URL ( <a href="#">Media.contentPath</a> )	字符串	Undefined	一个字符串，保存要播放的媒体的路径和文件名。
视频长度 ( <a href="#">Media.totalTime</a> )	整数	Undefined	播放 FLV 媒体所需的总时间。此设置是确保播放栏正常工作所必需的。此控件仅在媒体类型设置为 FLV 时可见。

## MediaController 参数

名称	类型	默认值	说明
<a href="#">activePlayControl</a> ( <a href="#">Media.activePlayControl</a> )	字符串： pause 或 play	pause	确定播放栏在实例化时是处于播放模式还是暂停模式。此模式确定在“播放”/“暂停”按钮上显示的图像，它与控制器实际所处的播放 / 暂停状态相反。
<a href="#">backgroundStyle</a> ( <a href="#">Media.backgroundStyle</a> )	字符串： default 或 none	default	确定是否为 <code>MediaController</code> 实例绘制铬印染背景。
<a href="#">controllerPolicy</a> ( <a href="#">Media.controllerPolicy</a> )	字符串： auto、on 或 off	auto	确定控制器是根据鼠标位置打开或关闭，还是锁定在打开或关闭状态。
<a href="#">horizontal</a> ( <a href="#">Media.horizontal</a> )	布尔值	true	确定实例的控制器部分为垂直方向还是水平方向。true 值指示组件将为水平方向。
<a href="#">enabled</a>	布尔值	true	确定此控件是否可由用户修改。true 值指示可以修改此控件。
<a href="#">visible</a>	布尔值	true	确定此控件是否对用户可见。true 值指示可查看此控件。

## MediaPlayback 参数

名称	类型	默认值	说明
控件位置 ( <a href="#">Media.controlPlacement</a> )	字符串： top、 bottom、 left 或 right	bottom	控制器的位置。值与方向有关。
控件可见性 ( <a href="#">Media.controllerPolicy</a> )	布尔值	true	确定控制器是否根据鼠标的位置而打开或关闭。



名称	类型	默认值	说明
自动播放 ( <a href="#">Media.autoPlay</a> )	布尔值	true	确定是否在加载媒体后立刻播放该媒体。
使用首选媒体大小 ( <a href="#">Media.autoSize</a> )	布尔值	true	确定是调整 <code>MediaController</code> 实例的大小以与媒体相适应还是使用其它设置。
FPS	整数	30	每秒的帧数。当选择了“毫秒”选项时，此控件禁用。
提示点 ( <a href="#">Media.cuePoints</a> )	数组	undefined	由提示点对象组成的数组，这些对象各自具有一个名称和时间位置，有效时间格式为 HH:MM:SS:FF（选择了“毫秒”选项时）或 HH:MM:SS:mmm 格式。
FLV 或 MP3 ( <a href="#">Media.mediaType</a> )	字符串： FLV 或 MP3	FLV	指定要播放的媒体类型。
Milliseconds	布尔值	false	确定播放栏是使用帧还是毫秒，以及提示点是使用秒还是帧。当选择此选项时，FPS 控件禁用。
URL ( <a href="#">Media.contentPath</a> )	字符串	undefined	一个字符串，保存要播放的媒体的路径和文件名。
视频长度 ( <a href="#">Media.totalTime</a> )	整数	undefined	播放 FLV 媒体所需的总时间。此设置是确保播放栏正常工作所必需的。

# 使用媒体组件创建应用程序（仅限 Flash Professional）

使用媒体组件创建 Flash 内容非常简单，通常只需要几个步骤。此示例显示如何创建一个播放公开提供的小型媒体文件的应用程序。

## 要将媒体组件添加到应用程序：

1. 在 Flash 中，选择“文件” > “新建”，然后选择“Flash 文档”。
2. 在“组件”面板中，双击 `MediaPlayback` 组件以将其添加到舞台上。
3. 在“属性”检查器中，执行以下操作：
  - 输入实例名称 **myMedia**。
  - 单击“启动组件检查器”。
4. 在“组件”检查器中，在“URL”文本框内输入 **`http://www.helpexamples.com/flash/video/water.flv`**。
5. 选择“控制” > “测试影片”，查看媒体播放。

# 自定义媒体组件（仅限 Flash Professional）

如果要更改媒体组件的外观，可以使用外观设置。若要查看组件自定义的完整指南，请参见《使用组件》中的第 5 章“自定义组件”。

## 在媒体组件中使用样式

媒体组件不使用样式。

## 在媒体组件中使用外观

尽管可以打开媒体组件源文档并更改其资源以获得需要的外观，但媒体组件不支持动态外观设置。最好是制作此文件的一份副本并使用该副本，这样即可始终拥有可恢复的已安装的源文档。可以在以下位置找到媒体组件源文档：

- Windows: C:\Program Files\Macromedia\Flash 8\语言\ Configuration\ComponentFLA\MediaComponents fla
- Macintosh: HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/MediaComponents fla

# Media 类（仅限 Flash Professional）

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > Media

ActionScript 类名称 mx.controls.MediaController、mx.controls.MediaDisplay、mx.controls.MediaPlayback

每个组件类都有一个 version 属性，而该属性是一个类属性。类属性只能用于该类本身。version 属性会返回一个字符串，该字符串指示组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.controls.MediaPlayback.version);
```



代码 trace(myMediaInstance.version); 返回 undefined。

# Media 类的方法摘要

下表列出了 **Media** 类的方法。

方法	组件	说明
<code>Media.addCuePoint()</code>	MediaDisplay、MediaPlayback	向组件实例添加提示点对象。
<code>Media.associateController()</code>	MediaDisplay	将 MediaDisplay 实例与 MediaController 实例关联。
<code>Media.associateDisplay()</code>	MediaController	将 MediaController 实例与 MediaDisplay 实例关联。
<code>Media.displayFull()</code>	MediaPlayback	将组件实例转换为全屏播放模式。
<code>Media.displayNormal()</code>	MediaPlayback	将组件实例转换回其原始屏幕大小。
<code>Media.getCuePoint()</code>	MediaDisplay、MediaPlayback	返回提示点对象。
<code>Media.pause()</code>	MediaDisplay、MediaPlayback	在媒体时间轴中的播放头当前位置暂停播放头。
<code>Media.play()</code>	MediaDisplay、MediaPlayback	在给定的开始点播放与组件实例关联的媒体。
<code>Media.removeAllCuePoints()</code>	MediaDisplay、MediaPlayback	删除与给定的组件实例关联的所有提示点对象。
<code>Media.removeCuePoint()</code>	MediaDisplay、MediaPlayback	删除与给定的组件实例关联的指定提示点。
<code>Media.setMedia()</code>	MediaDisplay、MediaPlayback	将媒体类型和路径设置为指定的媒体类型。
<code>Media.stop()</code>	MediaDisplay、MediaPlayback	停止播放头并将其移到位置 0，即媒体的开头。

# Media 类的属性摘要

下表列出了 Media 类的属性。

属性	组件	说明
Media.activePlayControl	MediaController	确定组件在运行时加载时的状态。
Media.aspectRatio	MediaDisplay、MediaPlayback	确定组件实例是否保持其视频高宽比。
Media.autoPlay	MediaDisplay、MediaPlayback	确定组件实例是否立即开始缓冲和播放。
Media.autoSize	MediaDisplay、MediaPlayback	确定 MediaDisplay 或 MediaPlayback 组件的媒体查看部分的大小。
Media.backgroundColor	MediaController	确定组件实例是否绘制其铬印染背景。
Media.bytesLoaded	MediaDisplay、MediaPlayback	只读；已加载且可以播放的字节数。
Media.bytesTotal	MediaDisplay、MediaPlayback	要加载到组件实例中的字节数。
Media.contentPath	MediaDisplay、MediaPlayback	一个字符串，保存要进行流式处理并播放的媒体的相对路径和文件名。
Media.controllerPolicy	MediaController、MediaPlayback	确定控制器是否在实例化时隐藏，仅当用户将鼠标移过控制器的折叠状态时才出现。
Media.controlPlacement	MediaPlayback	确定组件控件所在的位置。
Media.cuePoints	MediaDisplay、MediaPlayback	已指定到给定的组件实例的提示点对象数组。
Media.horizontal	MediaController	确定组件实例的方向。
Media.mediaType	MediaDisplay、MediaPlayback	确定要播放的媒体类型。
Media.playheadTime	MediaDisplay、MediaPlayback	对于正在播放的媒体时间轴，保存播放头的当前位置（以秒为单位）。
Media.playing	MediaDisplay、MediaPlayback 和 MediaController	对于 MediaDisplay 和 MediaPlayback，此属性是只读的，并会返回一个布尔值，指示给定的组件实例是否正在播放媒体。对于 MediaController，此属性是可读 / 写的，控制在控制器的“播放” / “暂停”按钮上显示的图像（正在播放或已暂停）。
Media.preferredHeight	MediaDisplay、MediaPlayback	FLV 文件的默认高度值。

属性	组件	说明
<code>Media.preferredWidth</code>	MediaDisplay、MediaPlayback	FLV 文件的默认宽度值。
<code>Media.totalTime</code>	MediaDisplay、MediaPlayback	一个整数，指示媒体的总长度（以秒为单位）。
<code>Media.volume</code>	MediaDisplay、MediaPlayback	一个介于 0（最小值）和 100（最大值）之间的整数，代表音量级别。

## Media 类的事件摘要

下表列出了 **Media** 类的事件。

事件	组件	说明
<code>Media.change</code>	MediaDisplay、MediaPlayback	当媒体播放时连续广播。
<code>Media.click</code>	MediaController、MediaPlayback	当用户单击“播放”/“暂停”按钮时进行广播。
<code>Media.complete</code>	MediaDisplay、MediaPlayback	播放头已到达媒体结尾的通知。
<code>Media.cuePoint</code>	MediaDisplay、MediaPlayback	播放头已到达给定的提示点的通知。
<code>Media.playheadChange</code>	MediaController、MediaPlayback	当用户移动播放滑块或单击“转到开头”或“转到结尾”按钮时，由组件实例进行广播。
<code>Media.progress</code>	MediaDisplay、MediaPlayback	连续生成，直到媒体完全下载。
<code>Media.scrubbing</code>	MediaController、MediaPlayback	当拖放播放头时生成。
<code>Media.volume</code>	MediaController、MediaPlayback	当用户调整音量时进行广播。

# Media.activePlayControl

## 适用于

MediaController。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

`myMedia.activePlayControl`

## 说明

属性：一个字符串，它指定在运行时加载 **MediaController** 组件时该组件的状态。"play" 值指示播放状态；"pause" 值指示暂停状态。对此属性和 `autoPlay` 属性进行设置，使二者指示相同的状态。默认值为 "play"。

在 **MediaController** 组件中显示的按钮图像与当前的播放 / 暂停状态相反。例如，处于播放状态时 **MediaController** 显示暂停按钮，因为这就是用户单击按钮并切换状态时所得到的结果。

由于它指示加载时控制器所处的状态，因此在创建控制器之前必须通过“属性”检查器或“组件”检查器（如果组件在舞台上）设置 `activePlayControl` 属性。如果通过 **ActionScript** 代码创建组件，则必须在 `initObj` 参数中设置此属性。在创建了组件后更改此属性的值将不起作用。仅当用户单击“播放” / “暂停”按钮时才可以更改该值。

## 示例

以下示例指示首次加载时控件暂停：

```
myMedia.activePlayControl = "pause";
```

## 另请参见

[Media.autoPlay](#)

# Media.addCuePoint()

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.addCuePoint(cuePointName, cuePointTime)
```

## 参数

*cuePointName* 一个字符串，它命名提示点。

*cuePointTime* 一个数字（单位为秒），指示何时广播 cuePoint 事件。

## 返回

无。

## 说明

方法：向 MediaPlayback 或 MediaDisplay 实例添加提示点对象。当播放头时间等于提示点时间时，即会广播 cuePoint 事件。

## 示例

以下代码将名为 Homerun 的提示点添加到 myMedia 中播放头时间为 16 秒的位置。

```
myMedia.addCuePoint("Homerun", 16);
```

## 另请参见

[Media.cuePoint](#)、[Media.cuePoints](#)、[Media.getCuePoint\(\)](#)、

[Media.removeAllCuePoints\(\)](#)、[Media.removeCuePoint\(\)](#)

# Media.aspectRatio

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.aspectRatio

## 说明

属性：一个布尔值，确定 MediaDisplay 或 MediaPlayback 实例在播放时是否保持视频高宽比。true 值指示应保持高宽比；false 值指示高宽比可以在播放时变化。默认值为 true。

## 示例

以下示例指示高宽比可以在播放时变化：

```
myMedia.aspectRatio = false;
```

# Media.associateController()

## 适用于

MediaDisplay。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.associateController(*instanceName*)

## 参数

*instanceName* 一个字符串，它指定要关联的 MediaController 组件的实例名称。



## 返回

无。

## 说明

方法：将 **MediaDisplay** 实例与 **MediaController** 实例关联。

如果使用 `Media.associateDisplay()` 将 **MediaController** 实例与 **MediaDisplay** 实例关联，则无需使用 `Media.associateController()`。

## 示例

以下代码将 `myMedia` 与 `myController` 关联：

```
myMedia.associateController(myController);
```

## 另请参见

[Media.associateDisplay\(\)](#)

# Media.associateDisplay()

## 适用于

**MediaController**。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.associateDisplay(instanceName)
```

## 参数

*instanceName* 一个字符串，它指定要关联的 **MediaDisplay** 组件的实例名称。

## 返回

无。

## 说明

方法：将 **MediaController** 实例与 **MediaDisplay** 实例关联。

如果使用 `Media.associateController()` 将 **MediaDisplay** 实例与 **MediaController** 实例关联，则无需使用 `Media.associateDisplay()`。

### 示例

以下代码将 `myMedia` 与 `myDisplay` 关联：

```
myMedia.associateDisplay(myDisplay);
```

### 另请参见

[Media.associateController\(\)](#)

## Media.autoPlay

### 适用于

`MediaDisplay`、`MediaPlayback`。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*myMedia*.autoPlay

### 说明

属性：一个布尔值，确定 `MediaPlayback` 或 `MediaDisplay` 实例是否应立即开始尝试缓冲和播放。`true` 值指示控件在运行时进行缓冲和播放；`false` 值指示控件在运行时停止。此属性取决于 `contentPath` 和 `mediaType` 属性。如果未设置 `contentPath` 和 `mediaType`，则在运行时不进行回放。默认值为 `true`。

### 示例

以下示例指示首次加载时控件不启动：

```
myMedia.autoPlay = false;
```

### 另请参见

[Media.contentPath](#)、[Media.mediaType](#)

# Media.autoSize

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.autoSize

## 说明

属性：一个布尔值，它确定 MediaDisplay 或 MediaPlayback 组件的媒体查看部分的大小。

对于 MediaDisplay 组件，此属性的作用如下所示：

- 如果将此属性设置为 true，无论组件的大小如何，Flash 都将以媒体的首选大小显示媒体。这意味着，除非 MediaDisplay 实例大小与该媒体的大小相同，否则，该媒体将溢出实例边界或无法填充实例。
- 如果将此属性设置为 false，Flash 将尽可能使用实例的大小，同时保持高宽比。如果 Media.autoSize 和 Media.aspectRatio 都设置为 false，则使用该组件的准确大小。

对于 MediaPlayback 组件，此属性的作用如下所示：

- 如果将此属性设置为 true，则除非媒体的回放区小于首选大小，否则 Flash 将以媒体的首选大小显示媒体。在这种情况下，Flash 将缩小媒体以适应实例的大小，并保持高宽比。如果首选大小小于实例的媒体区，则不使用部分媒体区。
- 如果将此属性设置为 false，Flash 将尽可能使用实例的大小，同时保持高宽比。如果 Media.autoSize 和 Media.aspectRatio 都设置为 false，则填充组件的媒体区。此区域定义为控件上面的区域（在默认的布局下），不包括其周围构成组件边缘的 8 个像素的边距。

默认值为 true。

### 示例

下面的示例指示控件不会根据其媒体大小进行回放：

```
myMedia.autoSize = false;
```

### 另请参见

[Media.aspectRatio](#)

## Media.backgroundColor

### 适用于

MediaController。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
myMedia.backgroundColor
```

### 说明

属性：一个字符串值，它指示为 **MediaController** 实例绘制何种背景。"default" 值指示绘制铬印染背景，而 "none" 值指示不绘制铬印染背景。默认值为 "default"。

这不是样式属性，因此不受样式设置的影响。

### 示例

以下示例指示将不会为控件绘制铬印染背景：

```
myMedia.backgroundColor = "none";
```

# Media.bytesLoaded

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.bytesLoaded
```

## 说明

只读属性；已加载到组件且可以播放的字节数。默认值是 undefined。

## 示例

以下代码创建名为 PlaybackLoad 的变量，并使用加载的字节数设置该变量。随后该变量将用于 for 循环。

```
// 创建保存加载的字节数的变量。
var PlaybackLoad:Number = myMedia.bytesLoaded;
// 执行 some 函数，直到准备好播放。
for (PlaybackLoad < 150) {
 someFunction();
}
```

# Media.bytesTotal

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.bytesTotal
```

## 说明

只读属性；要加载到 **MediaPlayback** 或 **MediaDisplay** 组件的字节数。默认值是 `undefined`。

## 示例

下面的示例告知用户要进行流式处理的媒体大小：

```
myTextField.text = myMedia.bytesTotal;
```

# Media.change

## 适用于

**MediaDisplay**、**MediaPlayback**。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
 // 在此处插入您的代码。
}
myMedia.addEventListener("change", listenerObject)
```

## 说明

事件；当播放媒体时由 **MediaDisplay** 和 **MediaPlayback** 组件进行广播。完成的百分比可以从组件实例中获取。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

**Media.change** 事件的事件对象有两个附加属性：

**target** 对广播中的对象的引用。

**type** 指示事件类型的字符串 "change"。

有关更多信息，请参见第 461 页的 [“EventDispatcher 类”](#)。

## 示例

以下示例使用对象侦听器确定播放头位置 (`Media.playheadTime`)，从中可计算出完成的百分比：

```
var myPlayerListener:Object = new Object();
myPlayerListener.change = function(eventObj:Object) {
 var myPosition:Number = myPlayer.playheadTime;
 var myPercentPosition:Number = (myPosition/myPlayer.totalTime);
};
myPlayer.addEventListener("change", myPlayerListener);
```

## 另请参见

[Media.playing](#)、[Media.pause\(\)](#)

# Media.click

## 适用于

`MediaController`、`MediaPlayback`。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObj:Object) {
 // ...
};
myMedia.addEventListener("click", listenerObject);
```

## 说明

事件：当用户单击“播放”/“暂停”按钮时进行广播。**detail** 字段将用于确定所单击的按钮。`Media.click` 事件对象具有以下属性：

**detail** 字符串 "pause" 或 "play"。

**target** 对 `MediaController` 或 `MediaPlayback` 实例的引用。

**type** 字符串 "click"。

## 示例

对于名为 **myMedia**（并具有库中的 **Window** 组件）的 **MediaController** 组件实例，以下示例会在用户单击“播放”/“暂停”按钮时打开一个弹出式窗口：

```
var myMediaListener:Object = new Object();
myMediaListener.click = function(eventObj:Object) {
 mx.managers.PopUpManager.createPopUp(_root, mx.containers.Window, true);
};
myMedia.addEventListener("click", myMediaListener);
```

# Media.complete

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObj:Object) {
 // ...
};
myMedia.addEventListener("complete", listenerObject);
```

## 说明

事件；播放头已到达媒体结尾的通知。Media.complete 事件对象具有以下属性：

target 对 MediaDisplay 或 MediaPlayback 实例的引用。

type 字符串 "complete"。

## 示例

下面的示例使用对象侦听器确定媒体何时结束播放：

```
var myListener:Object = new Object();
myListener.complete = function(eventObj:Object) {
 trace("media is finished");
};
myMedia.addEventListener("complete", myListener);
```



# Media.contentPath

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.contentPath

## 说明

属性：一个字符串，保存要进行流式处理和 / 或播放的媒体的相对路径和文件名。设置 contentPath 属性等效于不指定 *mediaType* 参数而调用 [Media.setMedia\(\)](#) 方法。当不使用 [Media.setMedia\(\)](#) 设置任何 *mediaType* 参数时，默认类型为 **FLV**。contentPath 属性的默认值为 undefined。

## 示例

以下示例在文本框中显示正播放的媒体的名称：

```
myTextField.text = myMedia.contentPath;
```

## 另请参见

[Media.setMedia\(\)](#)

# Media.controllerPolicy

## 适用于

MediaController、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.controllerPolicy

## 说明

属性：确定 **MediaController** 组件（或 **MediaPlayback** 组件内的控制器子组件）是否在实例化时隐藏，而仅当用户将鼠标移过控制器的折叠状态时显示。

此属性的可能值如下：

- "on" 指定控件始终是展开的。
- "off" 指定控件始终是折叠的。
- "auto"（默认值）指定控件将保持折叠状态，直到用户将鼠标移过点击区。点击区与在其中绘制折叠的控件的区域匹配。控件将保持展开状态，直到鼠标移离点击区。



点击区会随着控制器扩展和收缩。

## 示例

以下示例使控制器始终保持打开状态：

```
myMedia.controllerPolicy = "on";
```

# Media.controlPlacement

## 适用于

**MediaPlayback**。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.controlPlacement
```

## 说明

属性：确定 **MediaPlayback** 组件的控制器部分相对于其显示的位置。可能的值有 "top"、"bottom"、"left" 和 "right"。默认值为 "bottom"。

## 示例

在以下示例中，**MediaPlayback** 组件的控制器部分位于右侧：

```
myMedia.controlPlacement = "right";
```

# Media.cuePoint

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObj:Object) {
 // ...
};
myMedia.addEventListener("cuePoint", listenerObject);
```

## 说明

事件：播放头已到达提示点的通知。Media.cuePoint 事件对象具有以下属性：

cuePointName 一个字符串，指示提示点的名称。

cuePointTime 一个以帧数或秒数表示的数字，指示何时到达提示点。

target 对 MediaPlayback 对象（如果存在）或 MediaDisplay 对象本身的引用。

type 字符串 "cuePoint"。

## 示例

以下示例使用对象侦听器确定何时到达提示点：

```
var myCuePointListener:Object = new Object();
myCuePointListener.cuePoint = function(eventObject:Object){
 trace("heard " + eventObject.type + ", " + eventObject.target + ", " +
 eventObject.cuePointName + ", " + eventObject.cuePointTime);
};
myPlayback.addEventListener("cuePoint", myCuePointListener);
```

## 另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoints](#)、[Media.getCuePoint\(\)](#)

# Media.cuePoints

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.cuePoints

## 或者

*myMedia*.cuePoints[N]

## 说明

属性；已分配给 **MediaPlayback** 或 **MediaDisplay** 实例的提示点对象数组。在数组内，每个提示点对象都可以具有名称、以秒数或帧数表示的时间，以及一个播放器属性（即对象所关联的组件的实例名称）。默认值为空数组 ([])。

## 示例

如果正在播放动作预览，以下示例会删除第三个提示点：

```
if (myVariable == actionPreview) {
 myMedia.removeCuePoint(myMedia.cuePoints[2]);
}
```

## 另请参见

[Media.addCuePoint\(\)](#)、[Media.getCuePoint\(\)](#)、[Media.removeCuePoint\(\)](#)

# Media.displayFull()

## 适用于

MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

### 用法

*myMedia*.displayFull()

### 返回

无。

### 说明

方法：将 **MediaPlayback** 实例设置为全屏模式。在此模式下，组件进行扩展以填充整个舞台。若要将组件恢复至其正常大小，请使用 `Media.displayNormal()`。

### 示例

以下代码使组件进行扩展以填满舞台：

```
myMedia.displayFull();
```

### 另请参见

[Media.displayNormal\(\)](#)

## Media.displayNormal()

### 适用于

**MediaPlayback**。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*myMedia*.displayNormal()

### 返回

无。

### 说明

方法：在使用 `Media.displayFull()` 方法之后，将 **MediaPlayback** 实例设置回正常大小。

### 示例

以下代码会使 **MediaPlayer** 组件回复为其原始大小：

```
myMedia.displayNormal();
```

### 另请参见

[Media.displayFull\(\)](#)

## Media.getCuePoint()

### 适用于

MediaDisplay、MediaPlayer。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
myMedia.getCuePoint(cuePointName)
```

### 参数

*cuePointName* 在使用 `Media.addCuePoint()` 时提供的字符串。

### 返回

一个提示点对象。

### 说明

方法：根据其提示点名称返回提示点对象。

### 示例

以下代码将获取名为 `myCuePointName` 的提示点。

```
myMedia.removeCuePoint(myMedia.getCuePoint("myCuePointName"));
```

### 另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoint](#)、[Media.cuePoints](#)、[Media.removeCuePoint\(\)](#)

# Media.horizontal

## 适用于

MediaController。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.horizontal
```

## 说明

属性；确定 **MediaController** 组件是以垂直方向显示还是以水平方向显示。true 值指示组件以水平方向显示；false 值指示以垂直方向显示。当设置为 false 时，播放栏和回放滑块从下往上移动。默认值为 true。

## 示例

下面的示例将以垂直方向显示 **MediaController** 组件：

```
myMedia.horizontal = false;
```

# Media.mediaType

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.mediaType
```

## 说明

属性；指示要播放的媒体的类型（FLV 或 MP3）。默认值为 "FLV"。请参见《使用 Flash》中的“使用视频”。

### 示例

以下示例确定当前播放的媒体类型：

```
var currentMedia:String = myMedia.mediaType;
```

### 另请参见

[Media.setMedia\(\)](#)

# Media.pause()

### 适用于

MediaDisplay、MediaPlayback。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
myMedia.pause()
```

### 返回

无。

### 说明

方法：在当前位置暂停播放头。

### 示例

以下代码会暂停播放。

```
myMedia.pause();
```



# Media.play()

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.play(startingPoint)
```

## 参数

*startingPoint* 一个非负整数，它指示媒体开始播放的开始点（以秒为单位）。

## 返回

无。

## 说明

方法；在给定的开始点播放与组件实例关联的媒体。默认值为 playheadTime 的当前值。

## 示例

以下代码指示媒体组件应在 120 秒处开始播放：

```
myMedia.play(120);
```

## 另请参见

[Media.pause\(\)](#)

# Media.playheadChange

## 适用于

MediaController、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.playheadChange = function(eventObject){
 // 在此处插入您的代码。
}
myMedia.addEventListener("playheadChange", listenerObject)
```

## 说明

事件：当用户移动播放滑块或单击“转到开头”或“转到结尾”按钮时，由 **MediaController** 或 **MediaPlayback** 组件进行广播。Media.playheadChange 事件对象具有以下属性：

**detail** 一个数字，指示已播放的媒体的百分比。

**type** 字符串 "playheadChange"。

## 示例

以下示例在用户停止拖动播放头时将已播放的百分比发送到“输出”面板：

```
var controlListen:Object = new Object();
controlListen.playheadChange = function(eventObj:Object) {
 trace(eventObj.detail);
};
myMedia.addEventListener("playheadChange", controlListen);
```

# Media.playheadTime

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.playheadTime

### 说明

属性；对于正在播放的媒体时间轴，保存播放头的当前位置（以秒为单位）。默认值设置为播放头的位置。

### 示例

以下示例为播放头位置设置一个变量，以秒为单位表示：

```
var myPlayhead:Number = myMedia.playheadTime;
```

## Media.playing

### 适用于

MediaDisplay、MediaPlayback 和 MediaController。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

*myMedia.playing*

### 说明

属性；返回一个布尔值，它指示媒体是正在播放 (true) 还是已经暂停 (false)。此属性对 MediaDisplay 和 MediaPlayback 组件是只读的，对 MediaController 组件是可读 / 写的。

### 示例

以下代码确定媒体是正在播放还是暂停：

```
if(myMedia.playing == true){
 some function;
}
```

### 另请参见

[Media.change](#)

# Media.preferredHeight

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.preferredHeight

## 说明

属性：根据 FLV 文件的默认高度值设置。此属性仅适用于 FLV 媒体，因为高度对于 MP3 文件是固定的。此属性可以用于设置高度和宽度属性（包括组件自身的边距）。如果未设置 FLV 媒体，则默认值为 `undefined`。

## 示例

以下示例根据正在播放的媒体调整 **MediaPlayback** 实例的大小，并考虑了组件实例所需的边距像素：

```
if (myPlayback.contentPath != undefined) {
 var mediaHeight:Number = myPlayback.preferredHeight;
 var mediaWidth:Number = myPlayback.preferredWidth;
 myPlayback.setSize((mediaWidth + 20), (mediaHeight + 70));
}
```

# Media.preferredWidth

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*myMedia*.preferredWidth

## 说明

属性：根据 FLV 文件的默认宽度值设置。默认值是 `undefined`。

## 示例

以下示例将设置变量 `mediaWidth` 的所需宽度：

```
var mediaWidth:Number = myMedia.preferredWidth;
```

# Media.progress

## 适用于

`MediaDisplay`、`MediaPlayback`。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObj:Object) {
 // ...
};
myMedia.addEventListener("progress", listenerObject);
```

## 说明

事件：连续生成，直到媒体已完全下载。`Media.progress` 事件对象具有以下属性：

`target` 对 `MediaDisplay` 或 `MediaPlayback` 实例的引用。

`type` 字符串 "progress"。

## 示例

以下示例会侦听进程：

```
var myProgressListener:Object = new Object();
myProgressListener.progress = function(eventObj:Object) {
 // 使 lightMovieClip 在显示进度时闪烁。
 var lightVisible:Boolean = lightMovieClip.visible;
 lightMovieClip.visible = !lightVisible;
};
```

以下示例侦听进度，如果 **progress** 事件持续时间超过 3000 毫秒（3 秒），则调用另一个函数：

```
// 在调用 timeout 前延迟的持续时间。
var timeout:Number = 3000;

// 如果已到达 timeout，则应执行此操作：
function callback(arg) {
 trace(arg);
}

// 侦听进度。
var myListener:Object = new Object();
myListener.progress = function(eventObj:Object) {
 setInterval(callback, timeout, "Experiencing Network Delay");
};
md.addEventListener("progress", myListener);
```

## Media.scrubbing

### 适用于

MediaController、MediaPlayback。

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObj:Object) {
 // ...
};
myMedia.addEventListener("scrubbing", listenerObject);
```

### 说明

事件：当拖动播放头时产生。

target 对 MediaController 或 MediaPlayback 实例的引用。

type 字符串 "scrubbing"。

## 示例

以下示例将侦听拖动播放头的用户：

```
my_mp.addEventListener("scrubbing", scrubbingListener);
function scrubbingListener(evt_obj:Object):Void {
 trace(evt_obj.type+" @ "+getTimer()+" ms
 (isScrubbing="+evt_obj.detail+"");
}
```

# Media.removeAllCuePoints()

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.removeAllCuePoints()
```

## 返回

无。

## 说明

方法：删除与组件实例关联的所有提示点对象。

## 示例

以下代码会删除所有提示点对象：

```
myMedia.removeAllCuePoints();
```

## 另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoints](#)、[Media.removeCuePoint\(\)](#)

# Media.removeCuePoint()

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.removeCuePoint(cuePoint)
```

## 参数

*cuePoint* 对先前已通过 `Media.addCuePoint()` 指定的提示点对象的引用。

## 返回

无。

## 说明

方法；删除与组件实例关联的提示点。

## 示例

以下代码会删除名为 `myCuePoint` 的提示点：

```
myMedia.removeCuePoint(getCuePoint("myCuePoint"));
```

## 另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoints](#)、[Media.removeAllCuePoints\(\)](#)

# Media.setMedia()

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。



## 用法

`myMedia.setMedia(contentPath [, mediaType])`

## 参数

*contentPath* 一个字符串，它指示要播放的媒体的 URL。默认值是 `undefined`。

*mediaType* 一个字符串，它用于将媒体类型设置为 **FLV** 或 **MP3**。此参数是可选的。默认值为 **FLV**。

## 返回

无。

## 说明

方法：设置媒体类型并使用 URL 参数设置到指定媒体类型的路径。

此方法提供了设置 **MediaPlayback** 和 **MediaDisplay** 组件的内容路径和媒体类型的受支持的方法。`Media.contentPath` 属性也可用于设置内容路径，但不允许设置媒体类型。

如果仅使用 **FLV** 文件，则不需要指定 *mediaType* 参数。如果独占式使用 **MP3** 文件，则必须将 *mediaType* 参数设置为 **MP3** 一次。如果在 **FLV** 和 **MP3** 文件之间来回切换，则必须在每次调用 `setMedia()` 时更改媒体类型。如果尝试在未将媒体类型显式设置为 **MP3** 时播放 **MP3** 文件，该文件不会播放。

## 示例

以下代码为要播放的组件实例提供了新媒体：

```
myMedia.setMedia("http://www.helpexamples.com/flash/video/clouds.flv",
 "FLV");
```

# Media.stop()

## 适用于

**MediaDisplay**、**MediaPlayback**。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

`myMedia.stop()`

## 返回

无。

## 说明

方法：停止播放头并将其移到位置 0，即媒体的开头。

## 示例

以下代码停止播放头，并将其移动到位置 0 处：

```
myMedia.stop()
```

# Media.totalTime

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.totalTime
```

## 说明

属性：媒体的总长度（以秒为单位）。由于 FLV 文件格式在完全加载后才向媒体组件提供播放时间，因此必须手动输入 `Media.totalTime`，这样播放栏可以准确地反映媒体的实际播放时间。MP3 文件的默认值为媒体的播放时间。对于 FLV 文件，默认值为 `undefined`。

不能为 MP3 文件设置此属性，因为信息包含在 **Sound** 对象内。

## 示例

以下示例为 FLV 媒体设置播放时间（以秒为单位）：

```
myMedia.totalTime = 151;
```

# Media.volume

## 适用于

MediaDisplay、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
myMedia.volume
```

## 说明

属性；存储指示音量设置的整数值，范围为 0 到 100。默认值为 75。

## 示例

以下示例设置媒体回放的最大音量：

```
myMedia.volume = 100;
```

## 另请参见

[Media.volume](#)、[Media.pause\(\)](#)

# Media.volume

## 适用于

MediaController、MediaPlayback。

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.volume = function(eventObj:Object) {
 // ...
};
myMedia.addEventListener("volume", listenerObject);
```

## 说明

事件：当音量值由用户调整时进行广播。Media.volume 事件对象具有以下属性：

detail 一个介于 0 到 100 之间的整数值，它表示音量级别。

type 字符串 "volume"。

## 示例

以下示例通知用户已对音量进行了调整：

```
var myVolListener:Object = new Object();
myVolListener.volume = function(eventObj:Object) {
 mytextfield.text = "Volume adjusted!";
};
myMedia.addEventListener("volume", myVolListener);
```

## 另请参见

[Media.volume](#)

# Menu 组件（仅限 Flash Professional）

**Menu** 组件使用户可以从弹出菜单中选择一个项目，这与大多数软件应用程序的“文件”或“编辑”菜单很相似。

当用户滑过或单击一个按钮状的菜单激活器时，通常会在应用程序中打开 **Menu** 组件。还可以对 **Menu** 组件编写脚本，使其在用户按下特定的键时打开。

**Menu** 组件始终在运行时动态创建。将该组件从“组件”面板拖到库中，然后通过 **ActionScript** 使用以下代码创建菜单：

```
var myMenu = mx.controls.Menu.createMenu(parent, menuDataProvider);
```

使用以下代码在应用程序中打开菜单：

```
myMenu.show(x, y);
```

`menuShow` 事件在菜单呈现的前一刻对所有 **Menu** 实例的侦听器进行广播，从而可以更新菜单项的状态。类似地，在 **Menu** 实例隐藏后，会立即广播 `menuHide` 事件。

菜单中的菜单项是以 XML 进行描述的。有关更多信息，请参见第 819 页的“[了解 Menu 组件：视图和数据](#)”。

无法使用屏幕阅读器来访问 **Menu** 组件。

菜单通常嵌套在菜单栏中。有关菜单栏的信息，请参见第 875 页的“[MenuBar 组件（仅限 Flash Professional）](#)”。

# 与 Menu 组件进行交互（仅限 Flash Professional）

可以使用鼠标和键盘与 Menu 组件进行交互。

Menu 组件打开后会保持可见，直到它由脚本关闭，或者用户在菜单外部或启用的菜单项内部单击鼠标。

进行单击即会选中菜单项，但以下菜单项类型除外：

**禁用的项或分隔符** 滚动和单击不起作用（菜单仍保持可见）。

**子菜单的锚记** 滚动操作会激活子菜单；单击操作无效；滚动到子菜单项以外的任何项都会关闭子菜单。

选中一个项目时，Menu.change 事件会发送到菜单的所有侦听器，菜单会被隐藏，而且发生以下动作（取决于项目类型）：

**check** 切换该项的 selected 属性。

**radio** 该项成为其单选按钮组的当前选项。

移动鼠标会触发 Menu.rollOut 和 Menu.rollOver 事件。

在菜单外部按鼠标会关闭菜单并触发 Menu.menuHide 事件。

在启用的项目内部松开鼠标会按以下方式影响项目类型：

**check** 切换该项的 selected 属性。

**radio** 项的 selected 属性设置为 true，而且上次在单选按钮组中所选项的 selected 属性设置为 false。会设置相应单选按钮组对象的 selection 属性，以引用选中的菜单项。

**undefined 和分层菜单的父项** 切换分层菜单的可见性。

当 Menu 实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

键	说明
向下箭头 向上箭头	在菜单行中向下和向上移动选区。选区在顶部或底部行循环。
向右箭头	打开子菜单，或者将选区移动到菜单栏（如果存在菜单栏）中的下一个菜单。
向左箭头	关闭子菜单并将焦点返回到父菜单（如果存在父菜单），或者将选区移动到菜单栏（如果存在菜单栏）中的上一个菜单。
Enter	打开子菜单。如果子菜单不存在，此键与在行上单击并释放具有相同的效果。

<b>按键</b>	如果打开了某个菜单，可以按 Tab 键移出该菜单。必须要进行选择，或者按 Escape 键退出菜单。
-----------	----------------------------------------------------

# 使用 Menu 组件（仅限 Flash Professional）

可以使用 **Menu** 组件创建具有可选择选项的菜单，此菜单类似于大多数软件应用程序的“文件”和“编辑”菜单。还可以使用 **Menu** 组件创建上下文关联菜单，这些菜单在用户单击热点或按下组合键时出现。与 **MenuBar** 组件一起使用 **Menu** 组件来创建水平菜单栏，它带有在每个菜单栏项目下扩展的菜单。

与标准的桌面菜单一样，**Menu** 组件支持功能属于以下一般类别的菜单项：

**命令激活器** 这些项会触发事件；可以编写代码来处理这些事件。

**子菜单锚记** 这些项是用于打开子菜单的锚记。

**单选按钮** 这些项成组使用；一次只能选择一项。

**复选框项** 这些项代表布尔值（true 或 false）。

**分隔符** 这些项提供简单的水平线，用于将菜单项在外观上分为不同的组。

## 了解 Menu 组件：视图和数据

从概念上来说，**Menu** 组件由数据模型和显示数据的视图组成。**Menu** 类就提供这种视图，并且包含可视的配置方法。**MenuDataProvider** 类向全局 XML 原型对象添加方法（非常类似于 **DataProvider** API 对 **Array** 对象所做的操作）；这些方法使您可以在外部构建数据提供程序并将其添加到多个菜单实例中。数据提供程序会向其所有客户端视图广播所有更改。（请参见第 864 页的“**MenuDataProvider** 类”。）

**Menu** 实例是与各个菜单项对应的 XML 元素的分层集合。属性定义相应的菜单项在屏幕上的行为和外观。集合与 XML 之间可以方便地来回转换，而 XML 可用于描述菜单（**menu** 标签）和项（**menuitem** 标签）。内置的 **ActionScript XML** 类是 **Menu** 组件底层模型的基础。

具有两个项目的简单菜单可在 XML 中以两个菜单项子元素进行描述：

```
<menu>
 <menuitem label="Up" />
 <menuitem label="Down" />
</menu>
```



XML 节点的标签名称（**menu** 和 **menuitem**）并不重要；属性及其嵌套关系在菜单中使用。

## 关于分层菜单

要创建分层菜单，应在父 XML 元素中嵌入 XML 元素，如下所示：

```
<menu>
 <menuitem label="MenuItem A" >
 <menuitem label="SubMenuItem 1-A" />
 <menuitem label="SubMenuItem 2-A" />
 </menuitem>
 <menuitem label="MenuItem B" >
 <menuitem label="SubMenuItem 1-B" />
 <menuitem label="SubMenuItem 2-B" />
 </menuitem>
</menu>
```

这会将父菜单项转换为弹出菜单锚记，因此在选中时不会生成事件。

## 关于菜单项 XML 属性

菜单项 XML 元素的属性确定显示的内容、菜单项的行为，以及如何向 **ActionScript** 公开此菜单项。下表描述了 XML 菜单项的属性：

属性名称	类型	默认值	说明
label	字符串	undefined	代表菜单项的显示文本。除 separator 外，此属性对所有项类型都是必需的。
type	separator、check、radio、normal 或 undefined	undefined	菜单项类型：separator、check box、radio button 或 normal（命令或子菜单激活器）。如果此属性不存在，则默认值为 normal。
icon	字符串	undefined	图像资源的链接标识符。此属性不是必需的，而且不适用于 check、radio 和 separator 类型。
instanceName	字符串	undefined	一个标识符，可以用于引用根菜单实例中的菜单项实例。例如，名为 <b>yellow</b> 的菜单项可以以 myMenu.yellow 形式引用。该属性不是必需的属性。
groupName	字符串	undefined	一个标识符，可以用于关联单选按钮组中的几个单选按钮项，并显示根菜单实例中的单选按钮组的状态。例如，名为 <b>colors</b> 的单选按钮组可以以 myMenu.colors 形式引用。只有 radio 类型才需要此属性。



属性名称	类型	默认值	说明
selected	布尔值 (false 或 true) 或字符串 ("false" 或 "true")	false	一个布尔值或字符串值，它指示 check 或 radio 项是处于开启状态 (true) 还是关闭状态 (false)。该属性不是必需的。
enabled	布尔值 (false 或 true) 或字符串 ("false" 或 "true")	true	一个布尔值或字符串值，它指示是 (true) 否 (false) 可以选择此菜单项。该属性不是必需的。

# 关于菜单项类型（仅限 Flash Professional）

有四种菜单项，它们由 type 属性指定：

```
<menu>
 <menuitem label="Normal Item" />
 <menuitem type="separator" />
 <menuitem label="Checkbox Item" type="check" instanceName="check_1"/>
 <menuitem label="RadioButton Item" type="radio" groupName="radioGroup_1"
/>
</menu>
```

## 普通菜单项

Normal Item 菜单项没有 type 属性，这意味着 type 属性默认为 normal。**Normal** 项目可以是命令激活器或子菜单激活器，这取决于它们是否具有嵌套的子项目。

## 分隔符菜单项

type 属性设置为 separator 的菜单项用作菜单中的可视分隔符。下面的 XML 会创建三个菜单项：**Top**、**Middle** 和 **Bottom**，各项之间带有分隔符：

```
<menu>
 <menuitem label="Top" />
 <menuitem type="separator" />
 <menuitem label="Middle" />
 <menuitem type="separator" />
 <menuitem label="Bottom" />
</menu>
```

所有分隔符项目都被禁用。单击或滑过分隔符不起作用。

## 复选框菜单项

`type` 属性设置为 `check` 的菜单项用作菜单中的复选框项；当 `selected` 属性设置为 `true` 时，在菜单项的标签旁边会出现一个复选标记。当选中某个复选框项时，其状态会自动切换，而且会对根菜单上的所有侦听器广播 `change` 事件。不过，尽管复选框菜单项的行为与 **CheckBox** 组件类似，但复选框菜单项可以可见地显示，而不在复选标记周围显示框。因此，未被选中的复选框菜单项在被选中之前其外观与 **Normal** 菜单项相似。

以下示例定义三个复选框菜单项：

```
<menu>
 <menuItem label="Apples" type="check" instanceName="buyApples"
 selected="true" />
 <menuItem label="Oranges" type="check" instanceName="buyOranges"
 selected="false" />
 <menuItem label="Bananas" type="check" instanceName="buyBananas"
 selected="false" />
</menu>
```

可以在 **ActionScript** 中使用实例名称直接从菜单本身访问菜单项，如下例所示：

```
myMenu.setMenuItemSelected(myMenu.buyapples, true);
myMenu.setMenuItemSelected(myMenu.buyoranges, false);
```



`selected` 属性只能使用 `setMenuItemSelected()` 方法进行修改。您可以直接检查 `selected` 属性，但它返回字符串值 `true` 或 `false`。

## 单选按钮菜单项

可以将 `type` 属性设置为 `radio` 的菜单项分到一组，使得一次只能选择其中一项。尽管单选按钮菜单项的行为与 **RadioButton** 组件类似，但单选按钮菜单项可以可见地显示，而不在按钮周围显示边框。因此，未被选中的单选按钮菜单项在被选中之前其外观与 **Normal** 菜单项相似。

通过为菜单项指定与其 `groupName` 属性相同的值，可以创建单选按钮组，如下例所示：

```
<menu>
 <menuItem label="Center" type="radio" groupName="alignment_group"
 instanceName="center_item"/>
 <menuItem type="separator" />
 <menuItem label="Top" type="radio" groupName="alignment_group" />
 <menuItem label="Bottom" type="radio" groupName="alignment_group" />
 <menuItem label="Right" type="radio" groupName="alignment_group" />
 <menuItem label="Left" type="radio" groupName="alignment_group" />
</menu>
```

当用户选择其中一项时，当前的选定内容会自动改变，而且会对根菜单上的所有侦听器广播 change 事件。在 **ActionScript** 中，通过 selection 属性可访问单选按钮组中当前选中的项，如下所示：

```
var selectedItem = myMenu.alignment_group.selection;
myMenu.alignment_group = myMenu.center_item;
```

每个 groupName 值在根菜单实例的范围内必须是唯一的。



selected 属性只能使用 setMenuItemSelected() 方法进行修改。您可以直接检查 selected 属性，但它返回字符串值 true 或 false。

## 通过 ActionScript 访问菜单项

可以在 instanceName 属性中为每个菜单项指定唯一的标识符，从而能从根菜单直接访问菜单项。例如，以下 XML 代码为每个菜单项提供了 instanceName 属性：

```
<menu>
 <menuItem label="Item 1" instanceName="item_1" />
 <menuItem label="Item 2" instanceName="item_2" >
 <menuItem label="SubItem A" instanceName="sub_item_A" />
 <menuItem label="SubItem B" instanceName="sub_item_B" />
 </menuItem>
</menu>
```

可以使用 **ActionScript** 直接从菜单组件访问相应的实例及其属性，如下所示：

```
var aMenuItem = myMenu.item_1;
myMenu.setMenuItemEnabled(item_2, true);
var aLabel = myMenu.sub_item_A.attributes.label;
```



每个 instanceName 属性在根菜单组件实例（包括根菜单的所有子菜单）的范围内必须是唯一的。

# 关于初始化对象属性（仅限 Flash Professional）

*initObject*（初始化对象）参数是关于创建 **Menu** 组件的布局的一个基本概念。此参数是一个具有属性的对象。每个属性代表菜单项的一个可能的 XML 属性。（有关 *initObject* 参数中所允许属性的说明，请参见第 820 页的“关于菜单项 XML 属性”。）

*initObject* 参数用于以下方法：

- `Menu.addItem()`
- `Menu.addItemAt()`
- `MenuDataProvider.addItem()`
- `MenuDataProvider.addItemAt()`

以下示例创建一个 *initObject* 参数，它带有两个属性 - `label` 和 `instanceName`：

```
var i = myMenu.addItem({label:"myMenuItem",
 instanceName:"myFirstItem"});
```

可以协同使用多个属性来创建特定类型的菜单项。通过指定特定的属性来创建特定类型的菜单项（普通、分隔符、复选框或单选按钮）。

例如，可以使用以下 *initObject* 参数对 **Normal** 菜单项进行初始化：

```
myMenu.addItem({label:"myMenuItem", enabled:true, icon:"myIcon",
 instanceName:"myFirstItem"});
```

可以使用以下 *initObject* 参数对分隔符菜单项进行初始化：

```
myMenu.addItem({type:"separator"});
```

可以使用以下 *initObject* 参数对复选框菜单项进行初始化：

```
myMenu.addItem({type:"check", label:"myMenuCheck", enabled:false,
 selected:true, instanceName:"myFirstCheckItem"})
```

可以使用以下 *initObject* 参数对单选按钮菜单项进行初始化：

```
myMenu.addItem({type:"radio", label:"myMenuRadio1", enabled:true,
 selected:false, groupName:"myRadioGroup",
 instanceName:"myFirstRadioItem"})
```

应该将菜单项的 `instanceName`、`groupName` 和 `type` 属性视为只读。应该仅在创建项（例如，在对 `addItem()` 的调用中）时设置这些属性。在创建后修改这些属性可能会产生不可预料的结果。

# Menu 参数（仅限 Flash Professional）

可以在“属性”检查器中为每个 **Menu** 组件实例设置以下创作参数：

**rowHeight** 指示每行的高度（以像素为单位）。更改字体大小不会更改行高度。默认值为 20。

可以使用 **Menu** 组件的属性、方法和事件编写 **ActionScript** 来控制 **Menu** 组件。有关更多信息，请参见第 833 页的“**Menu** 类（仅限 **Flash Professional**）”。

## 创建具有 Menu 组件的应用程序（仅限 Flash Professional）

在以下示例中，开发人员正构建一个应用程序，并使用 **Menu** 组件显示一些用户可以发出的命令，例如“打开”、“关闭”和“保存”。

**创建具有 Menu 组件的应用程序：**

1. 选择“文件” > “新建”，创建一个 **Flash** 文档。

2. 将 **Menu** 组件从“组件”面板拖到库中。

菜单是通过 **ActionScript** 动态创建的。

3. 将 **Button** 组件从“组件”面板拖到库中。

该按钮将用于激活菜单。

4. 在第一帧上的“动作”面板中，输入以下代码以添加事件侦听器，用于侦听按钮的 **click** 事件。代码还会侦听菜单的 **change** 事件，并在“输出”面板中显示所选菜单项的名称：

```
/**
 要求：
 - 库中有 Menu 组件
 - 库中有 Button 组件
*/

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "menu_button", 10, {label:"Launch Menu"});

// 创建菜单。
var my_menu:Menu = Menu.createMenu();

// 添加某些菜单项。
my_menu.addItem("Open");
my_menu.addItem("Close");
my_menu.addItem("Save");
my_menu.addItem("Revert");
```

```
// 将更改侦听器添加到 Menu 以检测选中的是哪个菜单项。
var menuListener:Object = new Object();
menuListener.change = function(evt_obj:Object) {
 var item_obj:Object = evt_obj.menuItem;
 trace("Item selected: "+item_obj.attributes.label);
};
my_menu.addEventListener("change", menuListener);

// 添加一个按钮侦听器，在单击按钮时该侦听器显示菜单。
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 var my_button:Button = evt_obj.target;
 // 在按钮底部显示菜单。
 my_menu.show(my_button.x, my_button.y + my_button.height);
};
menu_button.addEventListener("click", buttonListener);
```

##### 5. 选择“控制” > “测试影片”。

单击“启动菜单”按钮以显示菜单。选择一个菜单项时，`trace()` 语句将在“输出”面板中报告所选内容。

**若要使用服务器中的 XML 数据创建并填充菜单，请执行以下操作：**

##### 1. 选择“文件” > “新建”，创建一个 Flash 文档。

##### 2. 将 Menu 组件从“组件”面板拖到库中。

菜单是通过 **ActionScript** 动态创建的。

##### 3. 在“动作”面板中，向第一帧中添加以下代码以创建菜单，并使用 `dataProvider` 属性从网页中加载菜单项：

```
/**
 要求：
 - 库中有 Menu 组件
*/

import mx.controls.Menu;

var my_menu:Menu = Menu.createMenu();

// 导入 XML 文件。
var myDP_xml:XML = new XML();
myDP_xml.ignoreWhite = true;
myDP_xml.onLoad = function(success:Boolean) {
 // 当数据到达时，将其传递给菜单。
 if (success) {
 my_menu.dataProvider = myDP_xml.firstChild;
 }
};
myDP_xml.load("http://www.flash-mx.com/mm/xml/menu.xml");
```

```
// 显示和定位菜单。
my_menu.show(100, 20);
```

**备注：**菜单项由 XML 文档的第一个子级的子级进行描述。

#### 4. 选择 “控制” > “测试影片”。

此处提供了网页中的 XML 菜单定义以供参考：

```
<?xml version="1.0" ?>
<menu>
 <menuitem label="Undo" />
 <menuitem type="separator" />
 <menuitem label="Cut" />
 <menuitem label="Copy" />
 <menuitem label="Paste" />
 <menuitem label="Clear" />
 <menuitem type="separator" />
 <menuitem label="Select All" />
</menu>
```

若要使用正确格式的 XML 字符串创建并填充菜单，请执行以下操作：

#### 1. 选择 “文件” > “新建”，创建一个 Flash 文档。

#### 2. 将 Menu 组件从 “组件” 面板拖到库中。

菜单是通过 **ActionScript** 动态创建的。

#### 3. 在 “动作” 面板中，将以下代码添加到第一帧，以创建菜单和添加一些项：

```
/**
 要求：
 - 库中有 Menu 组件
*/

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var theMenuElement_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
theMenuElement_obj.addMenuItem({label:"Undo"});
theMenuElement_obj.addMenuItem({type:"separator"});
theMenuElement_obj.addMenuItem({label:"Cut"});
theMenuElement_obj.addMenuItem({label:"Copy"});
```

```

theMenuElement_obj.addItem({label:"Paste"});
theMenuElement_obj.addItem({label:"Clear", enabled:"false"});
theMenuElement_obj.addItem({type:"separator"});
theMenuElement_obj.addItem({label:"Select All"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, theMenuElement_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

```

4. 选择 “控制” > “测试影片”。

若要使用 **MenuDataProvider** 类创建并填充菜单，请执行以下操作：

1. 选择 “文件” > “新建”，创建一个 Flash 文档。
2. 将 Menu 组件从 “组件” 面板拖到库中。

菜单是通过 **ActionScript** 动态创建的。

3. 在 “动作” 面板中，将以下代码添加到第一帧，以创建菜单和添加一些项：

```

/**
 要求：
 - 库中有 Menu 组件
*/

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var xml = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var theMenuElement = xml.addItem("XXXXX");

// 添加菜单项。
theMenuElement.addItem({label:"Undo"});
theMenuElement.addItem({type:"separator"});
theMenuElement.addItem({label:"Cut"});
theMenuElement.addItem({label:"Copy"});
theMenuElement.addItem({label:"Paste"});
theMenuElement.addItem({label:"Clear", enabled:"false"});
theMenuElement.addItem({type:"separator"});
theMenuElement.addItem({label:"Select All"});
// 创建 Menu 对象。
var my_menu = mx.controls.Menu.createMenu(_root, theMenuElement);

my_menu.show(100, 20);

```

4. 选择 “控制” > “测试影片”。



# 自定义 Menu 组件（仅限 Flash Professional）

菜单会在水平方向上调整其自身的大小，以适合其最宽的文本。还可以调用 `setSize()` 方法调整组件大小。图标应调整到 16 x 16 像素的最大大小。

## 在 Menu 组件中使用样式

可以调用 `setStyle()` 方法来更改菜单及其菜单项和子菜单的样式。**Menu** 组件支持以下样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
<code>alternatingRowColors</code>	光晕和范例	指定交替模式中行的颜色。它的值可以是两个或多个颜色（如 <code>0xFF00FF</code> 、 <code>0xCC6699</code> 和 <code>0x996699</code> ）组成的数组。与单值颜色样式不同， <code>alternatingRowColors</code> 不接受颜色名称；颜色的值必须为数字颜色代码。默认情况下不设置此样式，而是对所有的行使用 <code>backgroundColor</code> 来代替此样式。
<code>backgroundColor</code>	光晕和范例	菜单的背景颜色。默认的颜色为白色，在类样式声明中定义。如果指定了 <code>alternatingRowColors</code> ，则忽略此样式。
<code>backgroundDisabledColor</code>	光晕和范例	当组件的 <code>enabled</code> 属性设置为 "false" 时的背景颜色。默认值为 <code>0xDDDDDD</code> （中度灰）。
<code>borderStyle</code>	光晕和范例	<b>Menu</b> 组件使用 <code>RectBorder</code> 实例作为其边框，并对该类定义的样式做出响应。请参见第 985 页的 <a href="#">“RectBorder 类”</a> 。  默认的边框样式为 "menuBorder"。
<code>color</code>	光晕和范例	文本颜色。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>0x848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	文本的字体名称。默认值为 "_sans"。

样式	主题	说明
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式: "normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textAlign	光晕和范例	文本对齐方式: "left"、"right" 或 "center"。默认值为 "left"。
textDecoration	光晕和范例	文本修饰: "none" 或 "underline"。默认值为 "none"。
textIndent	光晕和范例	表示文本缩进的数字。默认值为 0。
defaultIcon	光晕和范例	在每行上显示的默认图标的名称。默认值为 <code>undefined</code> , 表示不显示图标。不需要图标属性, 不适用于“复选”、“单选”或“分隔符”项目, 并使用图像资源的链接标识符做为值参数。所有菜单项显示同一个图标。
popupDuration	光晕和范例	当菜单打开时, 过渡的持续时间。该值以毫秒为单位指定; 0 指示没有过渡。默认值为 150。
rollOverColor	光晕和范例	滑过的行的背景颜色。“光晕”主题的默认值为 <code>0xE3FFD6</code> (亮绿), “范例”主题的默认值为 <code>0xA8AAAA</code> (浅灰)。  当通过调用 <code>setStyle()</code> 更改 <code>themeColor</code> 时, 框架会将 <code>rollOverColor</code> 设置为一个与所选 <code>themeColor</code> 相关的值。
selectionColor	光晕和范例	所选行的背景颜色。“光晕”主题的默认值为 <code>0xCDFFC1</code> (浅绿), “范例”主题的默认值为 <code>0xEEEEEE</code> (极浅灰)。  当通过调用 <code>setStyle()</code> 更改 <code>themeColor</code> 时, 框架会将 <code>selectionColor</code> 设置为一个与所选 <code>themeColor</code> 相关的值。
selectionDuration	光晕和范例	从正常状态到选中状态进行过渡的时间长度 (以毫秒为单位)。默认值为 200。
selectionEasing	光晕和范例	对用于控制选择状态间过渡的扩大等式的引用。默认等式使用正弦输入 / 输出公式。有关更多信息, 请参见《使用组件》中的“自定义组件动画”。

样式	主题	说明
textRollOverColor	光晕和范例	鼠标指针滑过时文本的颜色。默认值为 0x2B333C（深灰）。设置 rollOverColor 时此样式非常重要，因为这两个设置必须相辅相成，才能使文本在指针滑过期间易于查看。
textSelectedColor	光晕和范例	所选行的文本颜色。默认值为 0x005F33（深灰）。设置 selectionColor 时此样式非常重要，因为这两个设置必须相互补充，才能使选中的文本易于查看。
useRollOver	光晕和范例	确定滑过一行时是否激活亮显示该行。默认值为 true。

## 为文档中的所有 Menu 组件设置样式

**Menu** 类继承自 **ScrollSelectList** 类。默认的分类样式属性在 **ScrollSelectList** 类上定义，该类由所有基于列表的组件共享。可以直接为此类设置新的默认样式值，新的设置将反映在所有受影响的组件中。

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

若要仅设置 **Menu** 组件的样式属性，您可以创建一个新的 **CSSStyleDeclaration**，并将其存储在 `_global.styles.Menu` 中。

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.Menu == undefined) {
 _global.styles.Menu = new CSSStyleDeclaration();
}
_global.styles.Menu.setStyle("backgroundColor", 0xFF00AA);
```

当创建新的类级别样式声明时，由 **ScrollSelectList** 声明提供的所有默认值都将丢失。这包括支持鼠标事件所需的 `backgroundColor`。若要创建类级别样式声明并保留默认值，请使用 `for..in` 循环，将旧的设置复制到新的声明中。

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.Menu;
for (var style in source) {
 target.setStyle(style, source.getStyle(style));
}
```

有关类级别样式的更多信息，请参见《使用组件》中的“为组件类设置样式”。

## 在 Menu 组件中使用外观

Menu 组件使用 `RectBorder` 的实例作为其边框（请参见第 985 页的“`RectBorder` 类”）。

Menu 组件具有分支、复选标记、单选点和分隔符图形的可视资源。这些资源不能动态设置外观，但是在两种主题中都可以从 `Flash UI Components 2/Themes/MMDefault/Menu Assets/States` 文件夹复制并根据需要修改这些资源。链接标识符不能更改，而且所有 Menu 实例都必须使用相同的元件。

### 创建 Menu 资源的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件” > “导入” > “打开外部库”，然后选择 `HaloTheme.fla` 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在您的操作系统上的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 `Flash UI Components 2/Themes/MMDefault` 文件夹，然后将 `Menu Assets` 文件夹拖到您的文档库中。
4. 在文档的库中展开 `Menu Assets/States` 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如，打开 `MenuCheckEnabled` 元件。
6. 按需要自定义元件。  
例如，将图像更改为 X 来代替复选标记。
7. 对所有要自定义的元件重复步骤 6-7。
8. 单击“返回”按钮返回主时间轴。
9. 将 Menu 组件从“组件”面板中拖到当前文档的库中。  
此操作会将 Menu 组件添加到库中并使其在运行时可用。
10. 将 `ActionScript` 添加到主时间轴，以在运行时创建一个 Menu 实例：

```
var myMenu = mx.controls.Menu.createMenu();
myMenu.addMenuItem({label: "One", type: "check", selected: true});
myMenu.addMenuItem({label: "Two", type: "check"});
myMenu.addMenuItem({label: "Three", type: "check"});
myMenu.show(0, 0);
```

11. 选择“控制” > “测试影片”。

# Menu 类（仅限 Flash Professional）

继承 MovieClip > UIObject 类 > UIComponent 类 > View > ScrollView > ScrollSelectList > Menu

ActionScript 类名称 mx.controls.Menu

Menu 类的方法和属性允许您在运行时创建并编辑菜单。

使用 ActionScript 设置菜单类的属性将会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

每个组件类都有一个 version 属性，而该属性是一个类属性。类属性只能用于该类本身。version 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.Menu.version);
```

提醒

代码 trace(myMenuInstance.version); 返回 undefined。

## Menu 类的方法摘要

下表列出了 Menu 类的方法。

方法	说明
<code>Menu.addItem()</code>	向菜单添加菜单项。
<code>Menu.addItemAt()</code>	将菜单项添加到菜单的特定位置。
<code>Menu.createMenu()</code>	创建 Menu 类的实例。这是静态方法。
<code>Menu.getItemAt()</code>	获取对指定位置的菜单项的引用。
<code>Menu.hide()</code>	关闭菜单。
<code>Menu.indexOf()</code>	返回给定菜单项的索引。
<code>Menu.removeAll()</code>	删除菜单中的所有项目。
<code>Menu.removeItem()</code>	删除指定的菜单项。
<code>Menu.removeItemAt()</code>	删除菜单中指定位置处的菜单项。
<code>Menu.setMenuItemEnabled()</code>	指示是 (true) 否 (false) 启用菜单项。
<code>Menu.setMenuItemSelected()</code>	指示菜单项是 (true) 否 (false) 被选中。
<code>Menu.show()</code>	在特定的位置或上次的位置打开菜单。

## 从 UIObject 类继承的方法

下表列出了 **Menu** 类从 **UIObject** 类继承的方法。从 **Menu** 对象调用这些方法时，请使用 *MenuInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **Menu** 类从 **UIComponent** 类继承的方法。从 **Menu** 对象调用这些方法时，请使用 *MenuInstance.methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# Menu 类的属性摘要

下表列出了 Menu 类的属性。

属性	说明
<code>Menu.dataProvider</code>	菜单的数据源。

## 从 UIObject 类继承的属性

下表列出了 Menu 类从 UIObject 类继承的属性。从 Menu 对象访问这些属性时，请使用 `MenuInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 Menu 类从 UIComponent 类继承的属性。从 Menu 对象访问这些属性时，请使用 `MenuInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

# Menu 类的事件摘要

下表列出了 Menu 类的事件。

事件	说明
<code>Menu.change</code>	当用户造成菜单更改时进行广播。
<code>Menu.menuHide</code>	当菜单关闭时进行广播。
<code>Menu.menuShow</code>	当菜单打开时进行广播。
<code>Menu.rollOut</code>	当指针滑离项目时进行广播。
<code>Menu.rollOver</code>	当指针滑过项目时进行广播。

## 从 UIObject 类继承的事件

下表列出了 Menu 类从 UIObject 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 Menu 类从 UIComponent 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。



# Menu.addItem()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
menuInstance.addItem(initObject)
```

用法 2:

```
menuInstance.addItem(childMenuItem)
```

## 参数

*initObject* 一个对象，它包含对菜单项的属性进行初始化的属性。请参见第 820 页的“关于菜单项 XML 属性”。

*childMenuItem* 一个 XML 节点对象。

## 返回

对添加的 XML 节点的引用。

## 说明

方法：用法 1 在菜单末尾添加菜单项。该菜单项是根据在 *initObject* 参数中提供的值构建的。用法 2 在菜单末尾添加菜单项，该菜单项是预先建立的 XML 节点（为 XML 对象形式）。添加预先存在的节点会将该节点从其先前的位置上删除。

## 示例

以下示例创建两个菜单，初始时为每个菜单添加一个菜单项。然后示例又将两个菜单项添加到第一个菜单中，方法为调用 addItem()，以通过指定其属性添加第一个菜单项。然后使用第二个菜单中预先构建的菜单项节点添加第二个菜单项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建 Menu 对象。
```

```
var first_menu:Menu = Menu.createMenu();
first_menu.addItem({label:"1st Item"});
var second_menu:Menu = Menu.createMenu();
second_menu.addItem({label:"1st Item 2nd Menu"});

// 第一种用法
first_menu.addItem({label:"Radio Item", instanceName:"radioItem1",
 type:"radio", selected:false, enabled:true, groupName:"myRadioGroup"});

// 第二种用法
first_menu.addItem(second_menu.getMenuItemAt(0));

// 显示菜单。
first_menu.show();
```

## Menu.addItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

#### 用法 1:

```
menuInstance.addItemAt(index, initObject)
```

#### 用法 2:

```
menuInstance.addItemAt(index, childMenuItem)
```

### 参数

*index* 一个整数，它指示添加项的索引位置（在多个子节点之间）。

*initObject* 一个对象，它包含对菜单项的属性进行初始化的属性。请参见第 820 页的“关于菜单项 XML 属性”。

*childMenuItem* 一个 XML 节点对象。

### 返回

对添加的 XML 节点的引用。

## 说明

方法；用法 1 在菜单中的指定位置添加菜单项（子节点）。该菜单项是根据在 *initObject* 参数中提供的值构建的。用法 2 在菜单中的指定位置添加菜单项，该菜单项是预先建立的 XML 节点（为 XML 对象形式）。添加预先存在的节点会将该节点从其先前的位置上删除。

## 示例

以下示例创建两个菜单，初始时为每个菜单添加一个菜单项。然后示例又将两个菜单项添加到第一个菜单中，方法为调用 `addMenuItemAt()`，以通过指定属性在第二个位置添加菜单项。然后使用第二个菜单中预先构建的菜单项节点在第三个位置添加菜单项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建 Menu 对象。
var first_menu:Menu = Menu.createMenu();
first_menu.addMenuItem({label:"1st Item"});
var second_menu:Menu = Menu.createMenu();
second_menu.addMenuItem({label:"1st Item 2nd Menu"});

// 第一种用法
first_menu.addMenuItemAt(1, {label:"Radio Item", instanceName:"radioItem1",
 type:"radio", selected:false, enabled:true, groupName:"myRadioGroup"});

// 第二种用法
first_menu.addMenuItemAt(2, second_menu.getMenuItemAt(0));

// 显示菜单。
first_menu.show();
```

# Menu.change

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 // 在此处插入您的代码。
};
menuInstance.addEventListener("change", listenerObject);
```

用法 2:

```
on (change) {
 // 在此处插入您的代码。
}
```

## 说明

事件：每当用户造成菜单更改时对所有注册的侦听器进行广播。

第 2 版 **Macromedia Component Architecture** 组件使用调度程序 - 侦听器事件模型。当 **Menu** 组件广播 `change` 事件时，该事件由附加到您创建的侦听器对象 (`listenerObject`) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Menu.change` 事件的事件对象具有以下附加属性：

- `menuBar` 对作为目标菜单的父菜单的 **MenuBar** 实例的引用。当目标菜单不属于 **MenuBar** 实例时，此值为 `undefined`。
- `menu` 对目标项所在的 **Menu** 实例的引用。
- `menuItem` 一个 XML 节点，它是选中的菜单项。
- `groupName` 一个字符串，指示项所属的单选按钮组的名称。如果项不在单选按钮组中，则此值为 `undefined`。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例创建一个菜单 `my_menu`，然后为其定义一个事件侦听器 `menulistener`，该侦听器侦听 `change` 事件。当用户单击某个菜单项而引发 `change` 事件时，示例会在“输出”面板中显示其标签属性。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("Edit");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);
my_menu.show();

var menuListener:Object = new Object();
menuListener.change = function(evt_obj:Object) {
 trace("Menu item chosen: " + evt_obj.menuItem.attributes.label);
};
my_menu.addEventListener("change", menuListener);
```

# Menu.createMenu()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
Menu.createMenu([parent [, mdp]])
```

## 参数

*parent* **MovieClip** 实例。影片剪辑是包含新 **Menu** 实例的父组件。此参数是可选的。

*mdp* 描述此 **Menu** 实例的 **MenuDataProvider** 实例。此参数是可选的。

## 返回

对新的 **Menu** 实例的引用。

## 说明

方法（静态）：对 **Menu** 实例进行实例化，并且选择性地将其附加到指定的父项，以指定的 **MenuDataProvider** 作为菜单项的数据源。

如果 *parent* 参数被省略或为 **null**，则 **Menu** 被附加到 **\_root** 时间轴。

如果 *mdp* 参数被省略或为 **null**，则菜单不含任何菜单项；必须调用 **addMenuItem()** 或 **setDataProvider()** 来填充菜单。

## 示例

以下示例创建一个菜单，该菜单的“新建”菜单项具有一个子菜单。该示例通过创建一个 **XML** 对象 **my\_xml**，并通过调用 **addMenuItem()** 向其中添加菜单项来创建该菜单。然后通过调用 **createMenu()** 并将该 **XML** 对象作为数据提供程序进行传递来创建菜单。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

var my_xml:XML = new XML();
var newItem_obj:Object = my_xml.addMenuItem({label:"New"});

// 创建主菜单的其它子菜单项。
```

```

my_xml.addItem({label:"Open", instanceName:"miOpen"});
my_xml.addItem({label:"Save", instanceName:"miSave"});
my_xml.addItem({type:"separator"});
my_xml.addItem({label:"Quit", instanceName:"miQuit"});

// 创建“新建”子菜单的子菜单项。
newItem_obj.addItem({label:"File..."});
newItem_obj.addItem({label:"Project..."});
newItem_obj.addItem({label:"Resource..."});

// 创建菜单。
var my_menu:Menu = Menu.createMenu(myParent_mc, my_xml);
my_menu.show(100, 20);

```

## Menu.dataProvider

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*menuItem*.dataProvider

### 说明

属性：**Menu** 组件中项目的数据源。

**Menu.dataProvider** 是 XML 节点对象。设置此属性会替换菜单的现有数据源。

默认值为 undefined。



所有 XML 或 XMLNode 实例在与 **Menu** 组件一起使用时，会自动接收 **MenuDataProvider** 类的方法和属性。

### 示例

以下示例创建一个菜单 (**my\_menu**)，并将网页中的菜单项加载到 **XML** 对象，然后通过将 **XML** 对象的子节点分配给菜单 (**my\_menu.dataProvider**) 的 **dataProvider** 属性来用菜单项填充菜单。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```

/**
 * 要求：
 * - 库中有 Menu 组件
 */

```

```
import mx.controls.Menu;

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu();

var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean){
 if (success) {
 my_menu.dataProvider = my_xml.firstChild;
 }
}
my_xml.load("http://www.flash-mx.com/mm/xml/menu.xml");

// 显示和定位菜单。
my_menu.show(100, 20);
```

## Menu.getItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*menuInstance*.getItemAt(*index*)

### 参数

*index* 一个整数，指示菜单中节点的索引。

### 返回

对指定节点的引用。

### 说明

方法：返回对菜单的指定子节点的引用。

### 示例

以下示例初始时创建两个菜单，每个菜单各有一个菜单项。然后将第二个菜单项添加到第一个菜单，方法是调用 getItemAt() 方法以获取第二个菜单的菜单项，然后将其添加到第一个菜单中。



首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 要求：
 - 库中有 Menu 组件
*/

import mx.controls.Menu;

// 创建 Menu 对象。
var first_menu:Menu = Menu.createMenu();
first_menu.addItem({label:"1st Item"});
var second_menu:Menu = Menu.createMenu();
second_menu.addItem({label:"1st Item 2nd Menu"});

// 将第二个菜单中的项添加到第一个菜单的第二个位置。
first_menu.addItemAt(1, second_menu.getMenuItemAt(0));

// 显示菜单。
first_menu.show();
```

## Menu.hide()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*menuInstance*.hide()

### 返回

无。

### 说明

方法：关闭菜单。

### 示例

以下示例创建一个按钮和一个带有两个菜单项的菜单，并以 **2000** 毫秒的时间间隔显示菜单。当时间间隔结束时，closeMenu() 函数调用 menu.hide() 方法来关闭该菜单。单击“重置菜单”按钮将触发 resetMenu() 侦听器，该侦听器会重新显示菜单并重置时间间隔。

首先将一个 **Menu** 组件和一个 **Button** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 要求：
 - 库中有 Menu 组件
 - 库中有 Button 组件
*/

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "my_button", 10, {label:"Reset Menu", _x:100,
 _y:50});

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("Edit");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

my_menu.show(100, 100);
// 2000 毫秒后调用 closeMenu。
var interval_id:Number = setInterval(closeMenu, 2000, my_menu);
function closeMenu(the_menu:Menu):Void {
 the_menu.hide();
 clearInterval(interval_id);
}
// 用于侦听按钮单击活动的侦听器；显示菜单并重置时间间隔。
function resetMenu(evt_obj:Object):Void {
 clearInterval(interval_id);
 my_menu.show(100, 100);
 interval_id = setInterval(closeMenu, 2000, my_menu);
}
my_button.addEventListener("click", resetMenu);
```

## 另请参见

[Menu.show\(\)](#)

# Menu.indexOf()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*menuInstance.indexOf(item)*

## 参数

*item* 对描述菜单项的 XML 节点的引用。

## 返回

指定的菜单项的索引，或者如果菜单项不属于此菜单，则为 undefined。

## 说明

方法；返回此菜单实例中的指定菜单项的索引。

## 示例

以下示例创建一个带有两个菜单项的菜单（这两个菜单项由 XML 数据提供程序提供），然后向菜单中添加第三个菜单项并保存由 addItem() 方法返回的引用。然后，该示例使用该引用调用 indexOf() 方法，以获取项的索引并将其显示在“输出”面板中。

首先将一个 **Menu** 组件和一个 **Button** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addItem("Edit");

// 添加菜单项。
menuDP_obj.addItem({label:"1st Item"});
```

```
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);
var myItem_obj:Object = my_menu.addMenuItem({label:"That item"});

// 显示和定位菜单。
my_menu.show(100, 20);

var myIndex_num:Number = my_menu.indexOf(myItem_obj);
trace("Index of 'That Item': " + myIndex_num);
```

## Menu.menuHide

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.menuHide = function(eventObject:Object) {
 // 在此处插入您的代码。
};
menuInstance.addEventListener("menuHide", listenerObject);
```

用法 2:

```
on (menuHide) {
 // 在此处插入您的代码。
}
```

### 说明

事件：每当菜单关闭时，向所有已注册的侦听器广播。

第 2 版组件使用调度程序 - 侦听器事件模型。当 **Menu** 组件调度 menuHide 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 addEventListener() 方法，并将处理函数的名称和侦听器对象的名称作为参数传递给它。

该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Menu.menuHide` 事件的事件对象具有两个附加属性：

- `menuBar` 对 **MenuBar** 实例的引用，而该实例是目标菜单的父菜单。当目标菜单不属于 **MenuBar** 实例时，此值为 `undefined`。
- `menu` 对隐藏的 **Menu** 实例的引用。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例创建一个按钮和一个带有两个菜单项的菜单。当用户单击按钮时，按钮 `click` 事件的侦听器显示该菜单。当用户再次单击时，会隐藏菜单，并且 `menuHide` 事件的侦听器 `menuListener` 会在“输出”面板中显示“**menu closed**”（菜单已关闭）。

首先将一个 **Menu** 组件和一个 **Button** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 要求:
 - 库中有 Menu 组件
 - 库中有 Button 组件
*/

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "my_button", 10);

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 添加一个按钮，在单击该按钮时显示菜单。
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 // 获取对按钮的引用
```

```

var the_button:Button = evt_obj.target;
// 在按钮底部显示菜单。
my_menu.show(the_button.x, the_button.y + the_button.height);
};
my_button.addEventListener("click", buttonListener);

// 创建侦听器对象。
var menuListener:Object = new Object();
menuListener.menuHide = function(evt_obj:Object) {
 trace("Menu closed.");
};

// 添加侦听器。
my_menu.addEventListener("menuHide", menuListener);

```

### 另请参见

[Menu.menuShow](#)

# Menu.menuShow

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

### 用法 1:

```

var listenerObject:Object = new Object();
listenerObject.menuShow = function(eventObject:Object) {
 // 在此处插入您的代码。
};
menuInstance.addEventListener("menuShow", listenerObject);

```

### 用法 2:

```

on (menuShow) {
 // 在此处插入您的代码。
}

```

## 说明

事件：当菜单打开时，向所有已注册的侦听器广播。所有父节点打开菜单以显示其子项。

第 2 版组件使用调度程序 - 侦听器事件模型。当 **Menu** 组件调度 `menuShow` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法，并将处理函数的名称和侦听器对象作为参数传递给它。

该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Menu.menuShow` 事件的事件对象具有两个附加属性：

- `menuBar` 对 **MenuBar** 实例的引用，该实例是目标菜单的父菜单。当目标菜单不属于 **MenuBar** 实例时，此值为 `undefined`。
- `menu` 对显示的 **Menu** 实例的引用。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例创建一个按钮和一个带有两个菜单项的菜单。当用户单击按钮时，按钮 `click` 事件的侦听器显示该菜单。`menuShow` 事件的侦听器 `menuListener` 在“输出”面板中显示“Menu open”（菜单已打开）。

首先将一个 **Menu** 组件和一个 **Button** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 要求：
 - 库中有 Menu 组件
 - 库中有 Button 组件
*/

import mx.controls.Button;
import mx.controls.Menu;

this.createClassObject(Button, "my_button", 10);

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("Edit");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});
```

```
// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 添加一个按钮，在单击该按钮时显示菜单。
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 // 获取对按钮的引用
 var the_button:Button = evt_obj.target;
 // 在按钮底部显示菜单。
 my_menu.show(the_button.x, the_button.y + the_button.height);
};
my_button.addEventListener("click", buttonListener);

// 创建侦听器对象。
var menuListener:Object = new Object();
menuListener.menuShow = function(evt_obj:Object) {
 trace("Menu open.");
};

// 添加侦听器。
my_menu.addEventListener("menuShow", menuListener);
```

### 另请参见

[Menu.menuHide](#)

## Menu.removeAll()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
menuInstance.removeAll()
```

### 返回

无。

### 说明

方法：删除所有项目并刷新菜单。



## 示例

以下示例创建一个带有两个菜单项的菜单，并在两秒钟（2000 毫秒）的时间间隔后，删除菜单中的所有节点。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);
var interval_id:Number = setInterval(remove, 2000, my_menu);
function remove(the_menu:Menu):Void {
 // 删除所有菜单项。
 the_menu.removeAll();
 clearInterval(interval_id);
 the_menu.show(100, 20);
}
```

# Menu.removeItem()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
menuInstance.removeMenuItem()
```

## 返回

对返回的菜单项（XML 节点）的引用。如果该位置没有项目，此值为 `undefined`。

## 说明

方法：删除指定菜单项及其所有子项，并刷新菜单。

## 示例

以下示例创建一个带有三个菜单项的菜单，并设置时间间隔以使菜单显示两秒钟（2000 毫秒）。当时间间隔结束时，该示例调用 `removeItem()` 函数，该函数调用 `removeMenuItem()` 方法删除菜单中的第一项并重新显示菜单。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"first Item"});
menuDP_obj.addMenuItem({label:"second Item"});
menuDP_obj.addMenuItem({label:"third Item"});
```

```
// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 2000 毫秒后调用 closeMenu。
var interval_id:Number = setInterval(removeItem, 2000, my_menu);
function removeItem(the_menu:Menu):Void {
 // 删除第一个节点项。
 var myItem_obj:Object = my_menu.getMenuItemAt(0);
 myItem_obj.removeMenuItem();
 clearInterval(interval_id);
 my_menu.show(100, 20);
}
```

## Menu.removeItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*menuInstance.removeItemAt(index)*

### 参数

*index* 要删除的菜单项的索引。

### 返回

对返回的菜单项（XML 节点）的引用。如果该位置没有项，则此值为 `undefined`。

### 说明

方法：删除指定索引处的菜单项及其所有子项。如果在该索引处不存在任何菜单项，则调用此方法没有效果。

## 示例

以下示例创建一个带有两个菜单项的菜单，并在两秒钟（2000 毫秒）时间间隔后，删除第二项（在索引 1 位置）。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);
var interval_id:Number = setInterval(remove, 2000, my_menu);
function remove(the_menu:Menu):Void {
 // 删除第二个节点项。
 var item_obj:Object = my_menu.removeMenuItemAt(1);
 trace("Item removed: " + item_obj);
 clearInterval(interval_id);
 the_menu.show(100, 20);
}
```

# Menu.rollOut

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.rollOut = function(eventObject:Object) {
 // 在此处插入您的代码。
};
menuInstance.addEventListener("rollOut", listenerObject);
```

用法 2:

```
on (rollOut) {
 // 在此处插入您的代码。
}
```

## 说明

事件：当指针滑离菜单项时，向所有已注册的侦听器广播。

第 2 版组件使用调度程序 - 侦听器事件模型。当 **Menu** 组件广播 rollOut 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Menu.rollOut` 事件的事件对象具有一个附加属性：`menuItem`，它是对指针滑离的菜单项（XML 节点）的引用。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例创建一个带有两个菜单项的菜单和一个 rollOut 事件侦听器。当广播 rollOut 事件时，事件处理函数 menuListener 中的一个 trace() 函数显示导致事件发生的菜单项的名称。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 创建侦听器对象。
var menuListener:Object = new Object();
menuListener.rollOut = function(evt_obj:Object) {
 trace("Menu rollOut: " + evt_obj.menuItem.attributes.label);
};

// 添加侦听器。
my_menu.addEventListener("rollOut", menuListener);
```

# Menu.rollOver

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.rollOver = function(eventObject:Object) {
 // 在此处插入您的代码。
};
menuInstance.addEventListener("rollOver", listenerObject);
```

用法 2:

```
on (rollOver) {
 // 在此处插入您的代码。
}
```

## 说明

事件；当指针滑过菜单项时，向所有已注册的侦听器广播。

第 2 版组件使用调度程序 - 侦听器事件模型。当 **Menu** 组件广播 `rollOver` 事件时，该事件由附加到您创建的侦听器对象 (`listenerObject`) 的函数（也称作“处理函数”）处理。您需要调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Menu.rollOver` 事件的事件对象具有一个附加属性：`menuItem`，它是对指针滑过的菜单项（XML 节点）的引用。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例创建一个带有两个菜单项的菜单和一个 `rollOver` 事件侦听器。当广播 `rollOver` 事件时，事件处理函数 `menuListener` 中的一个 `trace()` 函数显示导致事件发生的菜单项的名称。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */
```

```

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 创建侦听器对象。
var menuListener:Object = new Object();
menuListener.rollOver = function(evt_obj:Object) {
 trace("Menu rollOver: "+evt_obj.menuItem.attributes.label);
};

// 添加侦听器。
my_menu.addEventListener("rollOver", menuListener);

```

## Menu.setMenuItemEnabled()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
menuInstance.setMenuItemEnabled(item, enable)
```

### 参数

*item* 一个 XML 节点；数据提供程序中目标菜单项的节点。

*enable* 一个布尔值，它指示是 (true) 否 (false) 启用菜单项。



## 返回

无。

## 说明

方法：将目标项的 `enabled` 属性更改为 `enable` 参数中指定的状态。如果此调用导致状态更改，则用新状态重绘此项。

## 示例

以下示例创建一个带有两个菜单项的菜单，并调用 `setMenuItemEnabled()` 方法来禁用第一个菜单项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 选择第一个菜单项并禁用该菜单项。
var item_obj:Object = my_menu.getMenuItemAt(0);
my_menu.setMenuItemEnabled(item_obj, false);

// 显示和定位菜单。
my_menu.show(100, 20);
```

## 另请参见

[Menu.setSelected\(\)](#)

# Menu.setMenuItemSelected()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
menuInstance.setMenuItemSelected(item, select)
```

## 参数

*item* 一个 XML 节点。数据提供程序中目标菜单项的节点。

*select* 一个布尔值，它指示项是 (true) 否 (false) 被选中。如果项目是复选框，则其复选标记为可见或不可见。如果选择的项目为单选按钮，则该项目成为单选按钮组中的当前选中项。

## 返回

无。

## 说明

方法；将项的 `selected` 属性更改为 `select` 参数指定的状态。如果此调用导致状态更改，则用新状态重绘此项。这仅对 `type` 属性设置为 "radio" 或 "check" 的项有意义，原因是这会使它们的点或复选标记出现或消失。对类型为 "normal" 或 "separator" 的项调用此方法是无效的。

## 示例

以下示例创建一个带有两个菜单项的菜单，第二个菜单项是一个复选框菜单项。该示例调用 `setMenuItemSelected()` 方法使复选框菜单项处于选中状态。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
```

```
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({type:"check", label:"2nd Item"})

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

var myItem = my_menu.getMenuItemAt(1);
my_menu.setMenuItemSelected(myItem, true);

// 显示和定位菜单。
my_menu.show(100, 20);
```

## Menu.show()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*menuInstance*.show(*x*, *y*)

### 参数

*x* *x* 坐标。

*y* *y* 坐标。

### 返回

无。

### 说明

方法：打开指定位置的菜单。菜单会自动重新调整大小，使其所有顶层项目都可见，而且左上角放在由组件的父项提供的坐标系统中的指定位置。

如果省略了 *x* 和 *y* 参数，则菜单在其上次的位置显示。

## 示例

以下示例通过 XML 菜单对象创建一个菜单，并调用 `menu.show()` 方法显示该菜单。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

var my_xml:XML = new XML();

// 创建菜单项。
var newItem_obj:Object = my_xml.addMenuItem({label:"New"});
my_xml.addMenuItem({label:"Open", instanceName:"miOpen"});
my_xml.addMenuItem({label:"Save", instanceName:"miSave"});
my_xml.addMenuItem({type:"separator"});
my_xml.addMenuItem({label:"Quit", instanceName:"miQuit"});

// 创建和显示菜单。
var my_menu:Menu = Menu.createMenu(myParent_mc, my_xml);
my_menu.show(100, 20);
```

## 另请参见

[Menu.hide\(\)](#)

# MenuDataProvider 类

ActionScript 类名称 `mx.controls.menuclasses.MenuDataProvider`

**MenuDataProvider** 类是修饰器（混合）类，它向 **XMLNode** 全局类添加功能。此功能使分配给 `Menu.dataProvider` 属性的 XML 实例使用 **MenuDataProvider** 方法和属性来处理其数据以及关联的菜单视图。

请记住这些关于 **MenuDataProvider** 类的概念：

- **MenuDataProvider** 是修饰器（混合）类。无需对其进行实例化即可使用。
- 菜单以本机方式接受 XML 作为 `dataProvider` 属性值。
- 如果实例化 **Menu** 类，SWF 文件中的所有 XML 实例会由 **MenuDataProvider** 类进行修饰。
- 只有 **MenuDataProvider** 方法向 **Menu** 组件广播事件。您仍然可以使用本机 XML 方法，但它们不广播刷新 **Menu** 视图的事件。要控制数据模型，请使用 **MenuDataProvider** 方法。对于诸如在 **Menu** 层次结构中移动的只读操作，请使用 XML 方法。
- **Menu** 组件中的所有项目都是使用 **MenuDataProvider** 类修饰的 XML 对象。
- 对项目属性的更改在重绘之前不会在屏幕菜单中反映出来。

# MenuDataProvider 类的方法摘要

下表列出了 MenuDataProvider 类的方法。

方法	说明
<code>MenuDataProvider.addItem()</code>	添加子项目。
<code>MenuDataProvider.addItemAt()</code>	在指定的位置添加子项目。
<code>MenuDataProvider.getItemAt()</code>	获取对指定位置的菜单项的引用。
<code>MenuDataProvider.indexOf()</code>	返回指定菜单项的索引。
<code>MenuDataProvider.removeItem()</code>	删除菜单项。
<code>MenuDataProvider.removeItemAt()</code>	删除指定位置处的菜单项。

## MenuDataProvider.addItem()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

`myMenuDataProvider.addItem(initObject)`

用法 2:

`myMenuDataProvider.addItem(childMenuItem)`

### 参数

*initObject* 一个对象，它包含对菜单项的属性进行初始化的属性。有关更多信息，请参见第 820 页的“关于菜单项 XML 属性”。

*childMenuItem* 一个 XML 节点。

### 返回

对 XMLNode 对象的引用。

## 说明

方法；用法 1 将子项目添加到父菜单项（可以是菜单本身）的末尾。该菜单项根据在 *initObject* 参数中传递的值进行构建。用法 2 将在指定的 XML *childMenuItem* 参数中定义的子菜单项添加到父菜单项的末尾。

**MenuDataProvider** 实例中的任何节点或菜单项都可以调用 **MenuDataProvider** 类的方法。

## 示例

以下示例通过 XML 数据提供程序创建一个菜单。它调用 `addMenuItem()` 方法向主菜单中添加两项，另外还向主菜单的第一个菜单项的子菜单中添加两项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"Folders"});
menuDP_obj.addMenuItem({label:"Radio Edit", type:"radio"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 检索第一个菜单项并向其中添加项。
var item_obj:Object = menuDP_obj.getMenuItemAt(0);
item_obj.addMenuItem({label:"First item", instanceName:"firstItem1"});
item_obj.addMenuItem({label:"Second item", instanceName:"secondItem1"});
```

# MenuDataProvider.addItemAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
myMenuDataProvider.addItemAt(index, initObject)
```

用法 2:

```
myMenuDataProvider.addItemAt(index, childMenuItem)
```

## 参数

*index* 一个整数。

*initObject* 一个对象，包含用于初始化菜单项的属性的特定属性。有关更多信息，请参见第 820 页的“关于菜单项 XML 属性”。

*childMenuItem* 一个 XML 节点。

## 返回

对添加的 XML 节点的引用。

## 说明

方法：用法 1 将子项目添加到父菜单项（可以是菜单本身）中的指定索引位置。该菜单项根据在 *initObject* 参数中传递的值进行构建。用法 2 将在指定的 XML *childMenuItem* 参数中定义的子菜单项添加到父菜单项的指定索引处。

**MenuDataProvider** 实例中的任何节点或菜单项都可以调用 **MenuDataProvider** 类的方法。

## 示例

以下示例创建一个带有一个菜单项的菜单，然后调用 `addItemAt()` 方法来添加第二个菜单项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */
```

```
import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"Edit"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 添加菜单项。
menuDP_obj.addMenuItemAt(1, {label:"Save", instanceName:"saveItem1"});
```

## MenuDataProvider.getMenuItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*myMenuDataProvider.getMenuItemAt(index)*

### 参数

*index* 一个整数，指示菜单的位置。

### 返回

对指定的 XML 节点的引用。



## 说明

方法：返回对当前菜单项的指定子菜单项的引用。

**MenuDataProvider** 实例中的任何节点或菜单项都可以调用 **MenuDataProvider** 类的方法。

## 示例

以下示例创建一个菜单，并向其中添加一个菜单项，然后调用 `getMenuItemAt()` 方法来访问它的节点对象以向其中添加子菜单项。它还调用 `getMenuItemAt()` 方法在“输出”面板中显示子菜单项的标签。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
var menuItem_obj:Object = menuDP_obj.getMenuItemAt(0);
menuItem_obj.addMenuItem({label:"Submenu Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 在第一个菜单项中检索子菜单项。
var myMenuItem_obj:Object = menuDP_obj.firstChild;
trace(myMenuItem_obj.getMenuItemAt(0));
```

# MenuDataProvider.indexOf()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myMenuDataProvider.indexOf(item)
```

## 参数

*item* 对描述菜单项的 XML 节点的引用。

## 返回

指定的菜单项的索引；如果菜单项不属于此菜单，则返回 undefined。

## 说明

方法；返回此父菜单项中的指定菜单项的索引。

MenuDataProvider 实例中的任何节点或菜单项都可以调用 MenuDataProvider 类的方法。

## 示例

以下示例向菜单中添加一个菜单项，并调用 indexOf() 方法在“输出”面板中显示该菜单项的索引。

首先将一个 Menu 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");
```

```
// 添加菜单项。
menuDP_obj.addItem({label:"1st Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);

// 添加一项并跟踪该项的位置。
var myItem_obj:Object = menuDP_obj.addItem({label:"That item"});
var myIndex_num:Number = menuDP_obj.indexOf(myItem_obj);
trace("Position: " + myIndex_num);
```

## MenuDataProvider.removeItem()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
myMenuDataProvider.removeMenuItem()
```

### 返回

对删除的菜单项（XML 节点）的引用；如果出错则返回 `undefined`。

### 说明

方法：删除目标项目和任何子节点。

**MenuDataProvider** 实例中的任何节点或菜单项都可以调用 **MenuDataProvider** 类的方法。

### 示例

以下示例创建一个带有三个菜单项的菜单，并在两秒钟（2000 毫秒）时间间隔后调用 `removeMenuItem()` 来删除第一个菜单项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;
```

```

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();
d
// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});
menuDP_obj.addMenuItem({label:"3rd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);
// 2000 毫秒后调用 removeItem。
var interval_id:Number = setInterval(removeItem, 2000, my_menu);
function removeItem(the_menu:Menu):Void {
 // 删除位置 0 处的项。
 var myItem_obj:Object = menuDP_obj.getMenuItemAt(0);
 myItem_obj.removeMenuItem();
 clearInterval(interval_id);
}

```

## MenuDataProvider.removeItemAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*myMenuDataProvider.removeItemAt(index)*

### 参数

*index* 菜单项的索引。

### 返回

对删除的菜单项的引用。如果该位置没有菜单项，则此值为 `undefined`。

## 说明

方法：删除 *index* 参数指定的菜单项的子菜单项。如果在该索引处不存在任何菜单项，则调用此方法没有效果。

**MenuDataProvider** 实例中的任何节点或菜单项都可以调用 **MenuDataProvider** 类的方法。

## 示例

以下示例创建一个带有三个菜单项的菜单，并在两秒钟（2000 毫秒）时间间隔后调用 `removeMenuItemAt()` 来删除第一个菜单项。

首先将一个 **Menu** 组件拖到库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Menu 组件
 */

import mx.controls.Menu;

// 创建用作出厂设置的 XML 对象。
var my_xml:XML = new XML();

// 接下来创建的项不会出现在菜单中。
// createMenu() 方法调用（如下所示）预期会
// 接收到根元素，该元素的子元素将成为
// 菜单项。这只是一个创建该
// 根元素并为其指定一个便利名称的简单方法。
var menuDP_obj:Object = my_xml.addMenuItem("XXXXX");

// 添加菜单项。
menuDP_obj.addMenuItem({label:"1st Item"});
menuDP_obj.addMenuItem({label:"2nd Item"});
menuDP_obj.addMenuItem({label:"3rd Item"});

// 创建 Menu 对象。
var my_menu:Menu = Menu.createMenu(this, menuDP_obj);

// 显示和定位菜单。
my_menu.show(100, 20);
// 2000 毫秒后调用 removeItem。
var interval_id:Number = setInterval(removeItem, 2000, my_menu);
function removeItem(the_menu:Menu):Void {
 // 删除位置 0 处的项。
 menuDP_obj.removeMenuItemAt(0);
 clearInterval(interval_id);
}
```



# MenuBar 组件（仅限 Flash Professional）

使用 **MenuBar** 组件可以创建带有弹出菜单和命令的水平菜单栏，就像常见的软件应用程序中包含“文件”菜单和“编辑”菜单的菜单栏一样。**MenuBar** 组件对 **Menu** 组件进行了补充，方法是通过提供可单击的界面来显示和隐藏菜单，而这些菜单起到了组合鼠标和键盘交互性操作的作用。

**MenuBar** 组件使您可以通过几个步骤创建应用程序菜单。若要构建菜单栏，可以向描述一系列菜单的菜单栏指定 XML 数据提供程序，或者使用 `MenuBar.addMenu()` 方法一次添加数个菜单实例。

菜单栏中的每个菜单都由两部分组成：菜单和使菜单打开的按钮（称为菜单激活器）。这些可单击的菜单激活器作为文本标签出现在菜单栏中，并带有边框凹下和凸起的加亮显示状态，这些状态响应来自鼠标和键盘的交互性操作。

单击菜单激活器之后，相应的菜单会在其下面打开。菜单会保持活动状态，直到再次单击激活器，或选择了某个菜单项或者在菜单区域外进行了单击。

除了创建显示和隐藏菜单的菜单激活器外，**MenuBar** 组件还在一系列菜单之间创建组行为。这使用户可以浏览许多命令选项，方法是滑过一系列激活器或使用箭头键在列表中移动。鼠标和键盘通过交互性操作的方式协同工作，使用户可以在菜单栏内的菜单之间跳转。

用户不能在菜单栏上的菜单之间滚动。如果菜单超过了菜单栏的宽度，则会被遮盖。

不能使用屏幕阅读器访问 **MenuBar** 组件。

菜单通常嵌套在菜单栏中。有关菜单的信息，请参见第 817 页的“[Menu 组件（仅限 Flash Professional）](#)”。


# 与 MenuBar 组件进行交互（仅限 Flash Professional）

可以使用鼠标和键盘与 MenuBar 组件进行交互。

滑过菜单激活器会在激活器标签周围显示凸起的边框高亮区。

当 MenuBar 实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

键	说明
向下箭头	将选区下移一个菜单行。
向上箭头	将选区上移一个菜单行。
向右箭头	将选区移到下一个按钮。
向左箭头	将选区移到上一个按钮。
Enter/Escape 键	关闭打开的菜单。

如果菜单打开，不能通过按 Tab 键将其关闭。必须要进行选择，或者按 Escape 键关闭菜单。

# 使用 MenuBar 组件（仅限 Flash Professional）

可以使用 MenuBar 组件将一组菜单（如“文件”、“编辑”、“特殊”、“窗口”）添加到应用程序的顶部。

## MenuBar 参数

可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 MenuBar 组件实例设置以下创作参数：

**Labels** 一个数组，它将带有指定标签的菜单激活器添加到 MenuBar 组件。默认值为 []（空数组）。

可以在“组件”检查器（“窗口” > “组件检查器”）中设置每个 MenuBar 组件实例的以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 true。



**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。

提醒

**minHeight** 和 **minWidth** 属性由内部的大小调整例程使用。它们在 **UIObject** 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

不能使用 **ActionScript** 来访问 **Labels** 参数。不过，您可以编写 **ActionScript**，以便使用 **MenuBar** 组件的属性、方法和事件来控制该组件的其它选项。有关更多信息，请参见第 880 页的“**MenuBar** 类（仅限 **Flash Professional**）”。

## 创建具有 MenuBar 组件的应用程序

在本示例中，将 **MenuBar** 组件拖到舞台上，添加代码以向其中添加菜单项，然后将一个侦听器附加到菜单中以响应菜单项的选择。

在应用程序中使用 **MenuBar** 组件：

1. 选择“文件”>“新建”，然后创建新的 **Flash** 文档。
2. 将 **MenuBar** 组件从“组件”面板拖到舞台中。
3. 将菜单放置在舞台的顶部，形成标准布局。
4. 选择 **MenuBar** 实例，并在“属性”检查器中输入实例名称 **my\_mb**。
5. 在“动作”面板中的第 1 帧上输入以下代码：

```
import mx.controls.Menu;
import mx.controls.MenuBar;
```

```
var my_mb:MenuBar;
```

```
var my_menu:Menu = my_mb.addMenu("File");
my_menu.addMenuItem({label:"New", instanceName:"newInstance"});
my_menu.addMenuItem({label:"Open", instanceName:"openInstance"});
my_menu.addMenuItem({label:"Close", instanceName:"closeInstance"});
```

这段代码向 **MenuBar** 实例添加“**File**”（文件）菜单。然后，它使用 **Menu** 方法添加三个菜单项：“**New**”（新建）、“**Open**”（打开）和“**Close**”（关闭）。


6. 在“动作”面板中的第 1 帧上输入以下代码：

```
// 创建侦听器对象。
var mListener:Object = new Object();
mListener.change = function(evt_obj:Object) {
 var menuItem_obj:Object = evt_obj.menuItem;
 switch (menuItem_obj.attributes.instanceName) {
 case "newInstance":
 trace("New menu item");
 break;
 case "openInstance":
```

```
 trace("Open menu item");
 break;
 case "closeInstance":
 trace("Close menu item");
 break;
 }
 trace(menuItem_obj);
};

// 添加侦听器。
my_menu.addEventListener("change", mListener);
```

这段代码创建一个侦听器对象 `mListener`，它捕获菜单项选择并显示其名称和该菜单项对象的值。



您必须调用 `addEventListener()` 方法将侦听器注册到菜单实例（而不是菜单栏实例）。

7. 选择“控制”>“测试影片”，测试 `MenuBar` 组件。

# 自定义 MenuBar 组件（仅限 Flash Professional）

此组件根据通过 `MenuBar` 类的 `dataProvider` 属性或方法提供的激活器标签调整自身的大小。当激活器按钮在菜单栏中时，它会保持固定的大小（取决于字体样式和文本长度）。

## 在 MenuBar 组件中使用样式

`MenuBar` 组件为组中的每个菜单创建一个激活器标签。可以使用样式更改激活器标签的外观。`MenuBar` 组件支持下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题默认值为 0x0B333C，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。

样式	主题	说明
fontFamily	光晕和范例	文本的字体名称。默认值为 "_sans"。
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式: "normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textDecoration	光晕和范例	文本修饰: "none" 或 "underline"。默认值为 "none"。

**MenuBar** 组件还将 **Menu** 样式属性的所有样式设置进一步应用到所包含的 **Menu** 实例。有关 **Menu** 样式属性的列表, 请参见第 829 页的“在 **Menu** 组件中使用样式”。

## 在 MenuBar 组件中使用外观

**MenuBar** 组件使用三种外观来表示其背景, 使用影片剪辑元件来加亮显示单个项目, 并包含一个 **Menu** 组件作为其本身可以使用外观的弹出组件。下表中描述了 **MenuBar** 外观。有关设置 **Menu** 组件外观的信息, 请参见第 832 页的“在 **Menu** 组件中使用外观”。

**MenuBar** 组件支持以下外观属性。

属性	说明
menuBarBackLeftName	弹出图标弹起状态。
menuBarBackRightName	弹出图标按下状态。
menuBarBackMiddleName	弹出图标禁用状态。

### 创建 MenuBar 外观的影片剪辑元件:

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”, 然后选择 **HaloTheme.fla** 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在操作系统上的确切位置, 请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中, 展开 **Flash UI Components 2/Themes/MMDefault** 文件夹, 然后将 **MenuBar Assets** 文件夹拖到您的文档库中。
4. 在文档库中展开 **MenuBar Assets/Elements** 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如, 打开 **MenuBarBackLeft** 元件。
6. 按需要自定义元件。  
例如, 将外边缘更改为空白。

- 7. 对所有要自定义的元素重复步骤 5-6。  
例如，将中间元素和右侧元素的外边缘设置为黑色。
- 8. 单击“返回”按钮返回主时间轴。
- 9. 将 MenuBar 组件拖到舞台上。
- 10. 设置 MenuBar 属性，以便这些属性在菜单栏上显示项目。
- 11. 选择“控制”>“测试影片”。

碎嘴

在 MenuBar 组件中用于加亮显示单个项的边框是位于 Flash UI Components 2/Themes/MMDefault/Button Assets 文件夹中的一个 ActivatorSkin 实例。可将此元件自定义为指向不同的类，以提供不同的边框。但是不能修改元件名称，也不能在单个文档中为不同的 MenuBar 实例使用不同的元件。

# MenuBar 类（仅限 Flash Professional）

继承 MovieClip > UIObject 类 > UIComponent 类 > MenuBar

ActionScript 类名称 mx.controls.MenuBar

使用 MenuBar 类的方法和属性可以创建带有弹出菜单和命令的水平菜单栏。这些方法和属性补充了 Menu 类的方法和属性，所采用的方式是允许您创建可单击的界面来显示和隐藏菜单，而这些菜单起到了组合鼠标和键盘交互性操作的作用。

## MenuBar 类的方法摘要

下表列出了 MenuBar 类的方法。

方法	说明
<code>MenuBar.addMenu()</code>	将菜单添加到菜单栏。
<code>MenuBar.addMenuAt()</code>	将指定位置的菜单添加到菜单栏中。
<code>MenuBar.getMenuAt()</code>	获取对位于指定位置的菜单的引用。
<code>MenuBar.getMenuEnabledAt()</code>	返回一个布尔值，指示是 (true) 否 (false) 启用菜单。
<code>MenuBar.removeMenuAt()</code>	删除位于菜单栏特定位置的菜单。
<code>MenuBar.removeAll()</code>	删除菜单栏中的所有菜单项。
<code>MenuBar.setMenuEnabledAt()</code>	一个布尔值，指示是 (true) 否 (false) 可以选择此菜单。

## 从 UIObject 类继承的方法

下表列出了 `MenuBar` 类从 `UIObject` 类继承的方法。从 `MenuBar` 对象调用这些方法时，请使用 `MenuBar.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 `MenuBar` 类从 `UIComponent` 类继承的方法。从 `MenuBar` 对象调用这些方法时，请使用 `MenuBar.methodName` 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# MenuBar 类的属性摘要

下表列出了 MenuBar 类的属性。

属性	说明
<code>MenuBar.dataProvider</code>	菜单栏的数据模型。
<code>MenuBar.labelField</code>	一个字符串，它确定将各个 XMLNode 的哪个属性用作菜单的标签文本。
<code>MenuBar.labelFunction</code>	一个函数，它确定在每个菜单标签中显示的内容。

## 从 UIObject 类继承的属性

下表列出了 MenuBar 类从 UIObject 类继承的属性。从 MenuBar 对象调用这些属性时，请使用 `MenuBar.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 MenuBar 类从 UIComponent 类继承的属性。从 MenuBar 对象调用这些属性时，请使用 `MenuBar.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指示组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指示文档中组件的 Tab 键顺序。

# MenuBar 类的事件摘要

没有 MenuBar 类专用的事件。

## 从 Menu 类继承的事件

下表列出了 MenuBar 类从 Menu 类继承的事件。从 MenuBar 对象调用这些事件时，请使用 MenuBar.*eventName* 的形式。

事件	说明
<a href="#">Menu.change</a>	当用户造成菜单更改时进行广播。
<a href="#">Menu.menuHide</a>	当菜单关闭时进行广播。
<a href="#">Menu.menuShow</a>	当菜单打开时进行广播。
<a href="#">Menu.rollOut</a>	当指针滑离项目时进行广播。
<a href="#">Menu.rollOver</a>	当指针滑过项目时进行广播。

## 从 UIObject 类继承的事件

下表列出了 MenuBar 类从 UIObject 对象继承的事件。从 MenuBar 对象调用这些事件时，请使用 MenuBar.*eventName* 的形式。

事件	说明
<a href="#">UIObject.draw</a>	当对象将要绘制它的图形时进行广播。
<a href="#">UIObject.hide</a>	在对象的状态从可见变为不可见时广播。
<a href="#">UIObject.load</a>	创建子对象时广播。
<a href="#">UIObject.move</a>	移动了对象时广播。
<a href="#">UIObject.resize</a>	在调整对象大小后广播。
<a href="#">UIObject.reveal</a>	在对象的状态从不可见变为可见时广播。
<a href="#">UIObject.unload</a>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 `MenuBar` 类从 `UIComponent` 类继承的事件。从 `MenuBar` 对象调用这些事件时，请使用 `MenuBar.eventName` 的形式。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## MenuBar.addMenu()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
menuBarInstance.addMenu(label)
```

用法 2:

```
menuBarInstance.addMenu(label, menuDataProvider)
```

### 参数

*label* 一个字符串，它指示新菜单的标签。

*menuDataProvider* 描述菜单及其项目的 XML 或 XMLNode 实例。如果该值是 XML 实例，则使用该实例的第一个子实例。

### 返回

指向新的 `Menu` 对象的引用。

### 说明

方法；用法 1 将一个菜单和菜单激活器添加到菜单栏末尾，并使用指定的标签。用法 2 添加在指定的 XML *menuDataProvider* 参数中定义的一个菜单和菜单激活器。



## 示例

用法 1: 以下示例添加 “File”（文件）菜单，然后使用 `MenuBar.addItem()` 添加菜单项 “New”（新建）和 “Open”（打开）。

将 `MenuBar` 组件的一个实例拖到舞台上，然后在 “属性” 检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上 有 MenuBar 组件（实例名称：my_mb）
 */
var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", instanceName:"newInstance"});
my_menu.addItem({label:"Open", instanceName:"openInstance"});
```

用法 2: 以下示例添加 “Font”（字体）菜单，该菜单具有在 XML 数据提供程序 `myDP_xml` 中定义的菜单项 “Bold”（粗体）和 “Italic”（斜体）。

将 `MenuBar` 组件的一个实例拖到舞台上，然后在 “属性” 检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上 有 MenuBar 组件（实例名称：my_mb）
 */

var my_mb:mx.controls.MenuBar;

var myDP_xml:XML = new XML();
myDP_xml.addItem({type:"check", label:"Bold", instanceName:"check1"});
myDP_xml.addItem({type:"check", label:"Italic", instanceName:"check2"});

var my_menu:mx.controls.Menu = my_mb.addMenu("Font", myDP_xml);
```

# MenuBar.addMenuAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
menuBarInstance.addMenuAt(index, label)
```

用法 2:

```
menuBarInstance.addMenuAt(index, label, menuDataProvider)
```

## 参数

*index* 一个整数，它指示应在何处插入菜单。第一个位置为 **0**。若要附加到菜单的末尾，请调用 `MenuBar.addMenu(label)`。

*label* 一个字符串，它指示新菜单的标签。

*menuDataProvider* 描述菜单的 **XML** 或 **XMLNode** 实例。如果该值是 **XML** 实例，则使用该实例的第一个子实例。

## 返回

指向新的 **Menu** 对象的引用。

## 说明

方法；用法 1 使用指定的标签，在指定的索引处添加一个菜单和菜单激活器。用法 2 在指定的索引处添加一个菜单和带标签的菜单激活器。菜单的内容在 *menuDataProvider* 参数中定义。

## 示例

用法 1：以下示例在 **MenuBar** 实例 `my_mb` 上的第一个位置放置一个菜单。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 MenuBar 组件（实例名称：my_mb）
 */

var my_mb:mx.controls.MenuBar;
```

```
var my_menu:mx.controls.Menu = my_mb.addMenuAt(0, "Flash");
my_menu.addItem({label:"About Macromedia Flash",
 instanceName:"aboutInst"});
my_menu.addItem({label:"Preferences", instanceName:"PrefInst"});
```

用法 2: 以下示例添加 “Edit”（编辑）菜单，该菜单具有在 XML 数据提供程序 myDP\_xml 中定义的菜单项 “Undo”（撤消）、“Redo”（重做）、“Cut”（剪切）和 “Copy”（复制）。它将菜单添加到 MenuBar 实例 my\_mb 的第一个位置。

将 MenuBar 组件的一个实例拖到舞台上，然后在 “属性” 检查器中输入实例名称 my\_mb。将以下代码添加到时间轴的第 1 帧中：

```
/**
 要求:
 - 舞台上 有 MenuBar 组件（实例名称: my_mb）
*/

var my_mb:mx.controls.MenuBar;

var myDP_xml:XML = new XML();
myDP_xml.addItem({label:"Undo", instanceName:"undoInst"});
myDP_xml.addItem({label:"Redo", instanceName:"redoInst"});
myDP_xml.addItem({type:"separator"});
myDP_xml.addItem({label:"Cut", instanceName:"cutInst"});
myDP_xml.addItem({label:"Copy", instanceName:"copyInst"});

my_mb.addMenuAt(0, "Edit", myDP_xml);
```

## MenuBar.dataProvider

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
menuBarInstance.dataProvider
```

## 说明

属性：**MenuBar** 组件中的项目的数据模型。

`menuBar.dataProvider` 是 XML 节点对象。设置此属性会替换 **MenuBar** 组件的现有数据模型。数据提供程序可能具有的任何子节点用作菜单栏自身的项目；这些子节点的任何子节点用作其相应菜单的项目。

默认值为 `undefined`。



所有 XML 或 XMLNode 实例在与 **MenuBar** 组件一起使用时，会自动接收 `MenuDataProvider` 类的方法和属性。

## 示例

以下示例从网页中加载一个 XML 菜单文件，然后使用 `onLoad` 事件处理函数将其分配给 **MenuBar** 实例 `my_mb` 的 `dataProvider` 属性。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 MenuBar 组件（实例名称：my_mb）
 */

var my_mb:mx.controls.MenuBar;

var myDP_xml:XML = new XML();
myDP_xml.ignoreWhite = true;
myDP_xml.onLoad = function(success:Boolean) {
 if (success) {
 my_mb.dataProvider = myDP_xml.firstChild;
 } else {
 trace("error loading XML file");
 }
};
myDP_xml.load("http://www.flash-mx.com/mm/xml/menubar.xml");
```

# MenuBar.getMenuAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
menuBarInstance getMenuAt(index)
```

## 参数

*index* 一个整数，它指示菜单的位置。

## 返回

对指定索引处的菜单的引用。如果该位置没有菜单，则此值为 `undefined`。

## 说明

方法；返回对指定索引处的菜单的引用。由于 `getMenuAt()` 返回一个引用，因此可以将菜单项添加到菜单的指定索引处。

## 示例

以下示例创建一个“**File**”（文件）菜单，然后调用 `getMenuAt()` 以创建对它的引用。然后使用该引用将两个菜单项“**New**”（新建）和“**Open**”（打开）添加到“**File**”（文件）菜单中。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上 有 MenuBar 组件（实例名称：my_mb）
 */

var my_mb:mx.controls.MenuBar;

my_mb.addMenu("File");

var my_menu:mx.controls.Menu = my_mb.getMenuAt(0);
my_menu.addItem({label:"New",instanceName:"newInst"});
my_menu.addItem({label:"Open",instanceName:"openInst"});
```

# MenuBar.getMenuEnabledAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
menuBarInstance.getMenuEnabledAt(index)
```

## 参数

*index* 菜单栏中菜单的索引。

## 返回

一个布尔值，它指示是 (true) 否 (false) 可以选择此菜单。

## 说明

方法；返回一个布尔值，它指示是 (true) 否 (false) 可以选择此菜单。

## 示例

以下示例创建一个带有两个菜单项的“File”（文件）菜单，然后调用值为 `false` 的 `setMenuEnabledAt()` 以禁用该菜单。它还调用 `getMenuEnabledAt()`，并显示结果以指示如何确定菜单是否启用。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上要有 MenuBar 组件（实例名称：my_mb）
 */

var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", instanceName:"newInstance"});
my_menu.addItem({label:"Open", instanceName:"openInstance"});

// 禁用“File”（文件）菜单。
my_mb.setMenuEnabledAt(0, false);

// 检查能否选择“File”（文件）菜单。
trace("Menu can be selected: " + my_mb.getMenuEnabledAt(0));
```

# MenuBar.labelField

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`menuBarInstance.labelField`

## 说明

属性：一个字符串，指定将每个 **XML** 节点的哪个属性用作菜单的标签文本。此属性值还会传递到从菜单栏创建的任何菜单。默认值为 "label"。

在设置 `dataProvider` 属性后，此属性为只读。

## 示例

以下示例指定由每个 **XML** 节点的 `name` 属性提供菜单项的标签文本。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。

将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 MenuBar 组件（实例名称：my_mb）
 */
var my_mb:mx.controls.MenuBar;

// 将标签文本更改为从 "name" 读取。
my_mb.labelField = "name";

var my_menu:mx.controls.Menu = my_mb.addMenu({name:"File"});
my_menu.addItem({name:"New", instanceName:"newInstance"});
my_menu.addItem({name:"Open", instanceName:"openInstance"});
```

# MenuBar.labelFunction

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`menuBarInstance.labelFunction`

## 说明

属性：一个函数，确定在每个菜单的标签文本中显示什么内容。此函数接收与项目关联的 XML 节点作为参数，并返回要用作标签文本的字符串。此属性会传递到从菜单栏创建的任何菜单。默认值为 `undefined`。

在设置 `dataProvider` 属性后，此属性为只读。

## 示例

以下示例使用标签函数通过节点属性构建并返回一个自定义标签，如“新建” (Ctrl +N)。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 MenuBar 组件（实例名称：my_mb）
 */

var my_mb:mx.controls.MenuBar;

var my_menu:mx.controls.Menu = my_mb.addMenu("File");
my_menu.addItem({label:"New", data:"Control+N",
 instanceName:"newInstance"});
my_menu.addItem({label:"Open", data:"Control+O",
 instanceName:"openInstance"});
my_menu.addItem({label:"Close", data:"Control+W",
 instanceName:"closeInstance"});

// 为提供给菜单的 XML 数据设置格式。
my_menu.labelFunction = function(node:XMLNode):String {
 var attrb:Object = node.attributes;
 return (attrb.label + " (" + attrb.data + ")");
};
```



# MenuBar.removeAll()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
menuBarInstance.removeAll()
```

## 参数

无。

## 返回

无。

## 说明

方法；删除菜单栏上所有菜单项。

## 示例

以下示例在菜单栏上创建“File”（文件）、“Edit”（编辑）、“Tools”（工具）和“Window”（窗口）菜单。然后在单击按钮时，脚本调用 `removeAll()` 来删除这些菜单项。

将 **MenuBar** 组件拖动到舞台上，然后在“属性”检查器中输入实例名称 `myMenuBar`。还需将 **Button** 组件拖到舞台上，然后输入实例名称 `remBtn`。将以下代码添加到时间轴的第 1 帧中：

```
var menu = myMenuBar.addMenu("File");
var menu = myMenuBar.addMenu("Edit");
var menu = myMenuBar.addMenu("Tools");
var menu = myMenuBar.addMenu("Window");
// 添加一个可删除菜单项的按钮。
var rem_listener = new Object();
rem_listener.click = function() {
 myMenuBar.removeAll();
};
remBtn.addEventListener("click", rem_listener);
```

# MenuBar.removeMenuAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
menuBarInstance.removeMenuAt(index)
```

## 参数

*index* 要从菜单栏中删除的菜单的索引。

## 返回

对菜单栏中指定索引处的菜单的引用。如果在菜单栏中的该位置没有菜单，则此值为 `undefined`。

## 说明

方法；删除指定索引处的菜单。如果在该索引不存在任何菜单项，则调用此方法没有效果。而且，当删除多个菜单时，索引分配会随着每个菜单的删除相应地移动。

## 示例

以下示例在菜单栏上创建一个“File”（文件）菜单和一个“Edit”（编辑）菜单。然后调用 `removeMenuAt()` 来删除位置 `0` 处的菜单（即“File”（文件）菜单），留下“Edit”（编辑）菜单。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_mb`。将以下代码添加到时间轴的第 `1` 帧中：

```
/**
 * 要求：
 * - 舞台上有 MenuBar 组件（实例名称：my_mb）
 */

import mx.controls.Menu;
import mx.controls.MenuBar;

var my_mb:MenuBar;

var file_menu:Menu = my_mb.addMenu("File");
file_menu.addItem({label:"New", instanceName:"newInstance"});
file_menu.addItem({label:"Open", instanceName:"openInstance"});

var edit_menu:Menu = my_mb.addMenu("Edit");
```

```
edit_menu.addItem({label:"Cut", instanceName:"cutInstance"});
edit_menu.addItem({label:"Copy", instanceName:"copyInstance"});
edit_menu.addItem({label:"Paste", instanceName:"pasteInstance"});
```

```
// 删除“File”（文件）菜单。
my_mb.removeMenuAt(0);
```

## MenuBar.setEnabledAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
menuItemInstance.setEnabledAt(index, boolean)
```

### 参数

*index* 要在 **MenuBar** 实例中设置的菜单项的索引。

*boolean* 一个布尔值，它指示指定索引处的菜单项是 (true) 否 (false) 启用。

### 返回

无。

### 说明

方法：启用指定索引处的菜单。如果在该索引处不存在任何菜单，则调用此方法无效。

### 示例

以下示例将“File”（文件）菜单添加到菜单栏上，然后调用 `setEnabledAt()` 方法来启用或禁用该菜单，具体取决于 `menuEnabled_ch` 复选框处于选中状态还是清除状态。

将 **MenuBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 **my\_mb**。将一个 **CheckBox** 组件拖到舞台上，然后为其指定实例名称 **menuEnabled\_ch**。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 MenuBar 组件（实例名称：my_mb）
 * - 舞台上有 CheckBox 组件（实例名称：menuEnabled_ch）
 */
```

```
import mx.controls.CheckBox;
import mx.controls.Menu;
```

```
import mx.controls.MenuBar;

var my_mb:MenuBar;
var menuEnabled_ch:CheckBox;

menuEnabled_ch.selected = true;
var my_menu:Menu = my_mb.addMenu("File");
my_menu.addMenuItem({label:"New", instanceName:"newInstance"});
my_menu.addMenuItem({label:"Open", instanceName:"openInstance"});

var chListener:Object = new Object();
chListener.click = function(evt_obj:Object) {
 // 切换“File”（文件）菜单。
 my_mb.setMenuEnabledAt(0, evt_obj.target.selected);
}
menuEnabled_ch.addEventListener("click", chListener);
```

# NumericStepper 组件

**NumericStepper** 组件允许用户逐个通过一组经过排序的数字。该组件由显示在小上下箭头按钮旁边的文本框中的数字组成。用户按下按钮时，数字将根据 `stepSize` 参数中指定的单位递增或递减，直到用户释放按钮或达到最大或最小值为止。**NumericStepper** 组件的文本框中的文本也是可编辑的。

**NumericStepper** 组件只处理数值数据。此外，要显示两个以上的数值位置（如数字 5246 或 1.34），您在创作时必须调整步进器的大小。

在应用程序中，可以启用或禁用步进器。在禁用状态下，步进器不接收鼠标或键盘输入。如果您单击或按 **Tab** 键切换到启用的步进器，则它将接收焦点并且其内部焦点会设置为文本框。当 **NumericStepper** 实例有焦点时，您可以使用以下按键来控制它：

键	说明
向下箭头	值一次变化一个单位。
向左箭头	在文本框中将插入点移动到左侧。
向右箭头	在文本框中将插入点移动到右侧。
Shift+Tab	将焦点移到前一个对象。
Tab	将焦点移到下一个对象。
向上箭头	值一次变化一个单位。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

每个步进器实例的实时预览会反映创作过程中“属性”检查器或“组件”检查器中的值参数的设置。但是，在实时预览中，鼠标或键盘与步进器箭头按钮之间不能进行交互操作。

将 **NumericStepper** 组件添加到应用程序时，可以使用“辅助功能”面板使其可由屏幕读取器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.NumericStepperAccImpl.enableAccessibility();
```

不管组件有多少实例，都只对组件启用一次辅助功能。有关更多信息，请参见《使用 Flash》中的第 19 章“创建辅助内容”。

# 使用 NumericStepper 组件

NumericStepper 可用于任何您想让用户选择数值的场合。例如，您可以在表单中使用 NumericStepper 组件来允许用户设置信用卡到期时间。还可以使用 NumericStepper 组件来允许用户改变字体大小。

## NumericStepper 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 NumericStepper 实例设置以下创作参数：

**maximum** 设置可在步进器中显示的最大值。默认值为 10。如果您设置一个 **stepSize**，并使某一点处的最小值加上 **stepSize** 值不等于最大值（**minimum** + **stepSize** + **stepSize** + **stepSize**，依此类推），则步进器超过 **maximum**（最大值）时，将会显示最大值。

**minimum** 设置可在步进器中显示的最小值。默认值为 0。

**stepSize** 设置每次单击时步进器增大或减小的单位。默认值为 1。

**value** 设置在步进器的文本区域中显示的值。默认值为 0。

您可以在“组件”检查器（“窗口” > “组件检查器”）中设置每个 NumericStepper 组件实例的以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 `true`。

**visible** 是一个布尔值，它指示对象是 (`true`) 否 (`false`) 可见。默认值为 `true`。

提醒

`minHeight` 和 `minWidth` 属性由内部的大小调整例程使用。它们在 `UIObject` 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以编写 `ActionScript`，以便使用 NumericStepper 组件的属性、方法和事件来控制该组件的这些和其它选项。有关更多信息，请参见第 902 页的“NumericStepper 类”。

## 创建具有 NumericStepper 组件的应用程序

以下过程解释了如何在创作时将 NumericStepper 组件添加到应用程序。该示例将一个 NumericStepper 组件和一个 Label 组件放置到舞台上，然后在 NumericStepper 实例上创建一个 `change` 事件侦听器。当数字步进器中的值更改时，该示例会在 Label 实例中显示新值。

**创建具有 NumericStepper 组件的应用程序：**

1. 将 NumericStepper 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，输入实例名称 `my_nstep`。
3. 将 Label 组件从“组件”面板拖至舞台。

4. 在“属性”检查器中，输入实例名称 **my\_label**。
5. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
/**
 要求：
 - 舞台上 有 NumericStepper 组件（实例名称: my_nstep）
 - 舞台上 有 Label 组件（实例名称: my_label）
*/

var my_nstep:mx.controls.NumericStepper;
var my_label:mx.controls.Label;

my_label.text = "value = " + my_nstep.value;

// 创建侦听器对象。
var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object) {
 my_label.text = "value = " + evt_obj.target.value;
};

// 添加侦听器。
my_nstep.addEventListener("change", nstepListener);

最后一行代码将 change 事件处理函数添加到 my_nstep 实例。处理函数
(nstepListener) 将数字步进器中的当前值分配给 Label 实例的 text 属性。
```

## 自定义 NumericStepper 组件

在创作过程中以及在运行时，您都可以在水平和垂直方向上改变 **NumericStepper** 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，应使用 `setSize()` 方法（请参见 `UIObject.setSize()`）或任何适用的 **NumericStepper** 类的属性和方法。（请参见第 902 页的“**NumericStepper 类**”。）

调整 **NumericStepper** 组件的大小不会改变上下箭头按钮的大小。如果将步进器的大小调整为大于默认的高度，则上下箭头按钮将被固定在组件的顶部和底部。箭头按钮会始终出现在文本框的右侧。

# 对 NumericStepper 组件使用样式

您可以设置样式属性来更改 `NumericStepper` 实例的外观。如果样式属性的名称以“`Color`”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

`NumericStepper` 组件支持下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 <code>"haloGreen"</code> 、 <code>"haloBlue"</code> 和 <code>"haloOrange"</code> 。默认值为 <code>"haloGreen"</code> 。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题的默认值为 <code>0x0B333C</code> ，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>0x848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	文本的字体名称。默认值为 <code>"_sans"</code> 。
<code>fontSize</code>	光晕和范例	字体的磅值。默认值为 <code>10</code> 。
<code>fontStyle</code>	光晕和范例	字体样式： <code>"normal"</code> 或 <code>"italic"</code> 。默认值为 <code>"normal"</code> 。
<code>fontWeight</code>	光晕和范例	字体粗细： <code>"none"</code> 或 <code>"bold"</code> 。默认值为 <code>"none"</code> 。在调用 <code>setStyle()</code> 期间，所有组件还可以接受值 <code>"normal"</code> 来代替 <code>"none"</code> ，但随后对 <code>getStyle()</code> 的调用将返回 <code>"none"</code> 。
<code>textAlign</code>	光晕和范例	文本对齐方式： <code>"left"</code> 、 <code>"right"</code> 或 <code>"center"</code> 。默认值为 <code>"center"</code> 。
<code>textDecoration</code>	光晕和范例	文本修饰： <code>"none"</code> 或 <code>"underline"</code> 。默认值为 <code>"none"</code> 。
<code>repeatDelay</code>	光晕和范例	用户第一次按下按钮与操作开始重复之间延迟的毫秒数。默认值为 <code>500</code> （半秒）。
<code>repeatInterval</code>	光晕和范例	用户在某个按钮上按下鼠标按钮时两次自动单击之间延迟的毫秒数。默认值为 <code>35</code> 。
<code>symbolColor</code>	范例	箭头颜色。默认值为 <code>0x2B333C</code> （深灰）。



# 对 NumericStepper 组件使用外观

NumericStepper 组件使用外观来表示其向上和向下按钮的状态。要在创作时设置 NumericStepper 组件的外观，请在库中修改 Flash UI Components 2/Themes/MMDefault/Stepper Assets/States 文件夹内的外观元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。

如果启用了步进器，当指针移到向上按钮和向下按钮的上方时，这些按钮会显示其悬停状态。这些按钮在被按下时，会显示为按下状态。当释放鼠标后，这些按钮又会返回其悬停状态。如果鼠标按下时指针移离按钮，按钮会恢复到其原始状态。

如果禁用了步进器，不论用户进行什么交互性操作，它都会显示其禁用状态。

NumericStepper 组件使用以下外观属性：

属性	说明
upArrowUp	向上箭头按钮的弹起状态。默认值为 StepUpArrowUp。
upArrowDown	向上箭头按钮的按下状态。默认值为 StepUpArrowDown。
upArrowOver	向上箭头按钮的悬停状态。默认值为 StepUpArrowOver。
upArrowDisabled	向上箭头按钮的禁用状态。默认值为 StepUpArrowDisabled。
downArrowUp	向下箭头按钮的弹起状态。默认值为 StepDownArrowUp。
downArrowDown	向下箭头按钮的按下状态。默认值为 StepDownArrowDown。
downArrowOver	向下箭头按钮的悬停状态。默认值为 StepDownArrowOver。
downArrowDisabled	向下箭头按钮的禁用状态。默认值为 StepDownArrowDisabled。

## 创建 NumericStepper 外观的影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 HaloTheme.fla 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在您的操作系统上的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 Flash UI Components 2/Themes/MMDefault 文件夹，然后将 Stepper Assets 文件夹拖到您的文档库中。
4. 在文档库中展开 Stepper Assets 文件夹。
5. 在文档库中展开 Stepper Assets/States 文件夹。
6. 打开要自定义的元件以进行编辑。  
例如，打开 StepDownArrowDisabled 元件。
7. 按需要自定义元件。  
例如，将白色的内部图形更改为浅灰色。

8. 对所有要自定义的元素重复步骤 6-7。

例如，对向上箭头重复相同的更改。

9. 单击“返回”按钮返回主时间轴。

10. 将 **NumericStepper** 组件拖动到舞台上。

此示例自定义了禁用的外观，因此请使用 **ActionScript** 将 **NumericStepper** 实例设置为禁用，以便查看修改后的外观。

11. 选择“控制” > “测试影片”。



Stepper Assets/States 文件夹还包含一个 **StepTrack** 元件，如果 **NumericStepper** 实例的总高度大于两个箭头的高度之和，则该元件用作上下外观的分隔栏。不能通过外观属性修改此元件链接标识符，但只要链接标识符保持不变，就可以修改库元件。

## NumericStepper 类

继承 **MovieClip** > **UIObject** 类 > **UIComponent** 类 > **NumericStepper**

**ActionScript** 类名称 **mx.controls.NumericStepper**

**NumericStepper** 类的属性允许您在运行时做如下设置：步进器中显示的最小和最大值，步进器响应每次单击时增大或减小的单位，以及步进器中显示的当前值。

使用 **ActionScript** 设置 **NumericStepper** 类的属性会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

**NumericStepper** 组件会使用焦点管理器覆盖默认的 **Flash Player** 焦点矩形，并画出一个带有圆角的自定义焦点矩形。有关更多信息，请参见《使用组件》中的“创建自定义焦点导航”。

每个组件类都有一个 **version** 属性，该属性是一个类属性。类属性只能用于该类本身。**version** 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.NumericStepper.version);
```



代码 `trace(myNumericStepperInstance.version);` 返回 `undefined`。

# NumericStepper 类的方法摘要

没有 NumericStepper 类专用的方法。

## 从 UIObject 类继承的方法

下表列出了 NumericStepper 类从 UIObject 类继承的方法。从 NumericStepper 对象调用这些方法时，请使用 NumericStepper.*methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 NumericStepper 类从 UIComponent 类继承的方法。从 NumericStepper 对象调用这些方法时，请使用 NumericStepper.*methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# NumericStepper 类的属性摘要

下表列出了 NumericStepper 类的属性。

属性	说明
<code>NumericStepper.maximum</code>	一个指明最大范围值的数字。
<code>NumericStepper.minimum</code>	一个指明最小范围值的数字。
<code>NumericStepper.nextValue</code>	一个指明下一个连续值的数字。该属性为只读。
<code>NumericStepper.previousValue</code>	一个指明前一个连续值的数字。该属性为只读。
<code>NumericStepper.stepSize</code>	一个数字，它指示每次单击的变化单位。
<code>NumericStepper.value</code>	一个显示步进器当前值的数字。

## 从 UIObject 类继承的属性

下表列出了 NumericStepper 类从 UIObject 类继承的属性。从 NumericStepper 对象调用这些属性时，请使用 `NumericStepper.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 <b>x</b> 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 <b>y</b> 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的(true)还是不可见的(false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 UIComponent 类继承的属性

下表列出了 `NumericStepper` 类从 `UIComponent` 类继承的属性。从 `NumericStepper` 对象调用这些属性时，请使用 `NumericStepper.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指示组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指示文档中组件的 Tab 键顺序。

## NumericStepper 类的事件摘要

下表列出了 `NumericStepper` 类的事件。

事件	说明
<code>NumericStepper.change</code>	当步进器的值更改时触发。

## 从 UIObject 类继承的事件

下表列出了 `NumericStepper` 类从 `UIObject` 类继承的事件。从 `NumericStepper` 对象调用这些事件时，请使用 `NumericStepper.eventName` 的形式。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 `NumericStepper` 类从 `UIComponent` 类继承的事件。从 `NumericStepper` 对象调用这些事件时，请使用 `NumericStepper.eventName` 的形式。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## NumericStepper.change

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 //...
};
numericStepperInstance.addEventListener("change", listenerObject);
```

用法 2:

```
on (change) {
 // ...
}
```

## 说明

事件；当步进器的值更改时向所有已注册的侦听器广播。

组件实例 (*stepperInstance*) 调度一个事件（在本例中为 *change*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例在名为 *my\_nstep* 的数字步进器上创建一个 *change* 事件侦听器。当在数字步进器中更改该值时，侦听器会在“输出”面板中显示该值 (*value* 属性)。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 *my\_nstep*。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 NumericStepper 组件（实例名称: my_nstep）
 */

var my_nstep:mx.controls.NumericStepper;

// 创建侦听器对象。
var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object){
 // evt_obj.target 为生成 change 事件的组件，
 // 即数字步进器。
 trace("Value changed to " + evt_obj.target.value);
}
// 添加侦听器。
my_nstep.addEventListener("change", nstepListener);
```

# NumericStepper.maximum

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*numericStepperInstance*.maximum

## 说明

属性；步进器的最大范围值。此属性可以包含一个最多可达三位的十进制数字。默认值为 10。

## 示例

以下示例将步进器范围的最大值设置为 20。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_nstep。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上 有 NumericStepper 组件（实例名称：my_nstep）
 */
var my_nstep:mx.controls.NumericStepper;

my_nstep.maximum = 20;
```

## 另请参见

[NumericStepper.minimum](#)



# NumericStepper.minimum

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*numericStepperInstance.minimum*

## 说明

属性；步进器的最小范围值。此属性可以包含一个最多可达三位的十进制数字。默认值为 0。

## 示例

以下示例将 **NumericStepper** 实例的最小值和初始值设置为 100，并将最大值设置为 120。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_nstep。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上 有 NumericStepper 组件（实例名称：my_nstep）
 */
```

```
var my_nstep:mx.controls.NumericStepper;
```

```
my_nstep.minimum = 100;
my_nstep.maximum = 120;
my_nstep.value = my_nstep.minimum;
```

## 另请参见

[NumericStepper.maximum](#)

# NumericStepper.nextValue

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*numericStepperInstance*.nextValue

## 说明

属性（只读），下一个连续的值。此属性可以包含一个最多可达三位的十进制数字。

## 示例

以下示例将 **NumericStepper** 组件实例的初始值设置为 -6，并将 `stepSize` 属性设置为 3。然后在“输出”面板中显示 `nextValue` 属性值。在单击步进器上的向上箭头时，应该能够看到相同的值。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_nstep`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有一个 NumericStepper 组件（实例名称: my_nstep）
 */
var my_nstep:mx.controls.NumericStepper;

my_nstep.stepSize = 3;
my_nstep.minimum = -6;
my_nstep.maximum = 12;
my_nstep.value = my_nstep.minimum;
trace(my_nstep.nextValue); // -3
```

## 另请参见

[NumericStepper.previousValue](#)

# NumericStepper.previousValue

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*numericStepperInstance.previousValue*

## 说明

属性（只读），前一个连续的值。此属性可以包含一个最多可达三位的十进制数字。

## 示例

以下示例将 **NumericStepper** 实例的初始值设置为等于最小值 **6**，并将 **stepSize** 值设置为 **3**，然后创建一个 **change** 事件的侦听器对象。当发生 **change** 事件时，该示例会在“输出”面板中显示 **previousValue** 属性。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 **my\_nstep**。将以下代码添加到时间轴的第 **1** 帧中：

```
/**
 * 要求：
 * - 舞台上要有 NumericStepper 组件（实例名称: my_nstep）
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.minimum = 6;
my_nstep.value = my_nstep.minimum;
my_nstep.maximum = 120;
my_nstep.stepSize = 3;

// 创建侦听器对象。
var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object) {
 trace("previous value = " + evt_obj.target.previousValue);
}

// 添加侦听器。
my_nstep.addEventListener("change", nstepListener);
```

## 另请参见

[NumericStepper.nextValue](#)

# NumericStepper.stepSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*numericStepperInstance.stepSize*

## 说明

属性；要从当前值改变的单位数量。默认值为 1，该值不能为 0。该属性可以包含一个最多可达三位的十进制数字。

## 示例

以下示例将 **NumericStepper** 实例的初始值设置为等于最小值 3，还将 `stepSize` 值设置为 3，以使数字步进器在用户单击向上箭头时增加 3，在用户单击向下箭头时减少 3。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_nstep`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上要有 NumericStepper 组件（实例名称: my_nstep）
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.minimum = 3;
my_nstep.maximum = 120;
my_nstep.value = my_nstep.minimum;
my_nstep.stepSize = 3;
```

# NumericStepper.value

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*numericStepperInstance.value*

## 说明

属性：步进器的文本区域显示的当前值。如果该值与 `stepSize` 属性中定义的步进器的范围和步进增量不符，将不被赋值。此属性可以包含一个最多可达三位的十进制数字。

## 示例

以下示例将 **NumericStepper** 实例的当前值设置为 10，并将该值发送到“输出”面板。

将 **NumericStepper** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_nstep`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 NumericStepper 组件（实例名称: my_nstep）
 */

var my_nstep:mx.controls.NumericStepper;

my_nstep.value = 10;
my_nstep.maximum = 100;
trace(my_nstep.value); // 10
```



# PopUpManager 类

ActionScript 类名称 `mx.managers.PopUpManager`

使用 `PopUpManager` 类可以创建模式的或非模式的重叠窗口。（模式窗口在处于活动状态时不允许与其它窗口进行交互操作。）使用此类的方法可创建和破坏弹出窗口。

## PopUpManager 类的方法摘要

下表列出了 `PopUpManager` 类的方法。

方法	说明
<code>PopUpManager.createPopUp()</code>	创建弹出窗口。
<code>PopUpManager.deletePopUp()</code>	删除由调用 <code>PopUpManager.createPopUp()</code> 而创建的弹出窗口。

# PopUpManager.createPopUp()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

`PopUpManager.createPopUp(parent, class, modal [, initobj, outsideEvents])`

## 参数

*parent* 对弹出窗口所基于的窗口的引用。

*class* 对要创建的对象类的引用。

*modal* 一个布尔值，它指示该窗口是 (`true`) 否 (`false`) 是模式的。

*initobj* 一个包含初始化属性的对象。此参数是可选的。

*outsideEvents* 一个布尔值，指示在用户单击窗口以外的区域时是 (`true`) 否 (`false`) 触发事件。此参数是可选的。

## 返回

一个引用，引用所创建的对象。

如果 *class* 参数为 **Window** 并且库中有一个 **Window** 组件，则返回的引用是一个 **Window**。

## 说明

方法：如果为模式窗口，对 `createPopUp()` 的调用会找到以父级开始的最顶层的父窗口，然后创建一个类的实例。如果是非模式的，对 `createPopUp()` 的调用会创建一个类的实例作为父窗口的子窗口。

## 示例

以下代码在单击按钮时创建一个模式窗口：

```
lo = new Object();
lo.click = function(){
 mx.managers.PopUpManager.createPopUp(_root, mx.containers.Window, true);
}
button.addEventListener("click", lo);
```



# PopUpManager.deletePopUp()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004

## 用法

*windowInstance.deletePopUp()*;

## 参数

无。

## 返回

无。

## 说明

方法：删除一个弹出窗口并移除模式状态。当窗口被破坏时，被重叠的窗口负责调用 [PopUpManager.deletePopUp\(\)](#)。

## 示例

以下代码创建一个名为 win 的模式窗口，该窗口具有一个关闭按钮，当单击此关闭按钮时会删除该窗口：

```
import mx.managers.PopUpManager
import mx.containers.Window
win = PopUpManager.createPopUp(_root, Window, true, {closeButton:true});
lo = new Object();
lo.click = function(){
 win.deletePopUp();
}
win.addEventListener("click", lo);
```



## ProgressBar 组件

**ProgressBar** 组件显示加载内容的进度。**ProgressBar** 可用于显示加载图像和部分应用程序的状态。加载进程可以是确定的也可以是不确定的。当要加载的内容量是已知时，请使用确定的进度栏。确定的进度栏是一段时间内任务进度的线性表示。当要加载的内容量是未知时，请使用不确定的进度栏。您可以添加标签来显示加载内容的进度。

**ProgressBar** 组件包含左端、右端和进度跟踪。所谓“端”就是进度栏的末端，在此处直观地表示进度跟踪的结束。每个 **ProgressBar** 实例的实时预览都会反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。实时预览中会反映以下参数：**conversion**、**direction**、**label**、**labelPlacement**、**mode** 和 **source**。

### 使用 ProgressBar 组件

进度栏允许您在内容加载过程中显示内容的进度。当用户与应用程序交互操作时，这是必需的反馈信息。

使用 **ProgressBar** 组件有几种模式：您可以使用模式参数来设置模式。最常用的模式是事件模式和轮询模式。这两种模式使用 **source** 参数来指定一个加载进程，该进程发出 **progress** 和 **complete** 事件（事件模式和轮询模式）或者公开 **getBytesLoaded** 和 **getBytesTotal** 方法（轮询模式）。您也可以在手动模式下使用 **ProgressBar** 组件，方法是：手动设置 **maximum**、**minimum** 和 **indeterminate** 属性，并调用 **ProgressBar.setProgress()** 方法。

## ProgressBar 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 **ProgressBar** 实例设置下列创作参数：

**conversion** 是一个数字，在显示标签字符串中的 %1 和 %2 的值之前，用这些值除以该数字。默认值为 1。

**direction** 指示进度栏填充的方向。该值可以是 right 或 left，默认值为 right。

**label** 是指示加载进度的文本。此参数是一个字符串，其格式是“已加载 %1，共 %2 (%3%%)”。在此字符串中，%1 是当前已加载字节数的占位符，%2 是总共要加载的字节数的占位符，%3 是已加载内容的百分比的占位符。字符“%%”是字符“%”的占位符。如果 %2 的值为未知，它将被替换为两个问号(??)。如果值为 **undefined**，则标签不显示。

**labelPlacement** 指示与进度栏相关的标签的位置。此参数可以是下列值之一：top、bottom、left、right 和 center。默认值为 bottom。

**mode** 是进度栏运行的模式。此值可以是下列之一：event、polled 或 manual。默认值为 event。

**source** 是一个要转换为对象的字符串，它表示源的实例名称。

而且，您可以在“组件”检查器（通过“窗口” > “组件检查器”菜单选项打开）中为每个 **ProgressBar** 组件实例设置以下附加参数：

**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。

提醒

minHeight 和 minWidth 属性由内部的大小调整例程使用。它们在 UIObject 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以编写 **ActionScript**，以便使用其属性、方法和事件来控制 **ProgressBar** 组件的这些和其它选项。有关更多信息，请参见第 926 页的“**ProgressBar 类**”。

## 创建具有 ProgressBar 组件的应用程序

以下过程解释了如何在创作时将 **ProgressBar** 组件添加到应用程序。在该示例中，进度栏用于事件模式。在事件模式中，要加载的内容必须发出 `progress` 事件和 `complete` 事件，进度栏使用这两个事件来显示进度。（这两个事件由加载器组件发出。有关更多信息，请参见第 751 页的“**Loader 组件**”。）

在事件模式下创建带有 **ProgressBar** 组件的应用程序：

1. 将一个 **ProgressBar** 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，执行以下操作：
  - 输入实例名称 **my\_pb**。
  - 为 `mode` 参数选择“Event”。
3. 将 **Loader** 组件从“组件”面板上拖到舞台上。
4. 在“属性”检查器中，输入实例名称 **my\_ldr**。
5. 在舞台上选择进度栏，并在“属性”检查器中为 `source` 参数输入 **my\_ldr**。
6. 在时间轴上选择第一帧，打开“动作”面板，输入以下代码，该代码会将一个 JPEG 文件加载到 **Loader** 组件中：

```
/**
 要求：
 - 舞台上有 Loader 组件（实例名称：my_ldr）
 - 舞台上有 ProgressBar 组件（实例名称：my_pb）
*/

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.source = my_ldr;
my_ldr.autoLoad = false;
my_ldr.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// 如果 autoLoad 为 false，则直到调用 load() 时才开始加载
my_ldr.load();
```

在以下示例中，在轮询模式下使用进度栏。在轮询模式中，**ProgressBar** 使用源对象的 `getBytesLoaded()` 和 `getBytesTotal()` 方法来显示其进度。

在轮询模式下创建带有 ProgressBar 组件的应用程序：

1. 将一个 ProgressBar 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，输入实例名称 **my\_pb**。
3. 在时间轴上选择第一帧，并打开“动作”面板，然后输入以下代码。这段代码会创建一个名为 my\_sound 的 Sound 对象，并调用 loadSound()，以将一个声音加载到 Sound 对象中：

```
/**
 * 要求：
 * - 舞台上有一个 ProgressBar 组件（实例名称：my_pb）
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = "my_sound";

var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
 trace("Sound loaded");
}
my_pb.addEventListener("complete", pbListener);

var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/disco.mp3",
 true);
```

在以下示例中，在手动模式下使用进度栏。在手动模式中，您必须设置 maximum、minimum 和 indeterminate 属性，并使用 setProgress() 方法来显示进度。在手动模式中不需要设置 source 属性。

在手动模式下创建带有 ProgressBar 组件的应用程序：

1. 将一个 ProgressBar 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，执行以下操作：
  - 输入实例名称 **my\_pb**。
  - 为 mode 参数选择“Manual”。
3. 在时间轴中选择第 1 帧，并打开“动作”面板，然后输入以下代码。这段代码会通过调用 setProgress() 对每个文件下载过程手动更新进度栏：

```
for (var i:Number = 1; i <= total; i++){
 // 插入代码以加载文件
 my_pb.setProgress(i, total);
}
```

随后是另外两个示例。

在手动模式下创建带有 ProgressBar 组件的应用程序（示例 2）：

1. 将一个 Label 组件拖到舞台上，然后为它指定实例名称 **my\_label**。
2. 将一个 ProgressBar 组件拖到舞台上，然后为它指定实例名称 **my\_pb**。
3. 在舞台上选择 **my\_pb** ProgressBar，然后在“属性”检查器中将组件的模式参数设置为“manual”。
4. 在时间轴中选择第 1 帧，然后在“动作”面板中添加下面的 ActionScript：

```
var feed_xml:XML = new XML();
feed_xml.onLoad = function(success:Boolean):Void {
 clearInterval(timer);
 my_label.text = "XML Loaded";
 my_pb.setProgress(feed_xml.getBytesLoaded(),
 feed_xml.getBytesTotal());
};
function updatePB(local_xml:XML):Void {
 my_pb.setProgress(local_xml.getBytesLoaded(),
 local_xml.getBytesTotal());
}
var timer:Number = setInterval(updatePB, 100, feed_xml);
feed_xml.load("http://www.helpexamples.com/flash/xml/menu.xml");
```

5. 按 **Ctrl+Enter** 键以进行测试。

在手动模式下创建带有 ProgressBar 组件的应用程序（示例 3）：

1. 将一个 ProgressBar 组件拖到舞台上，然后为它指定实例名称 **my\_pb**。
2. 在舞台上选择 **my\_pb** ProgressBar，然后在“属性”检查器中将组件的模式参数设置为“manual”。
3. 在时间轴中选择第 1 帧，然后在“动作”面板中添加下面的 ActionScript：

```
var img_mcl:MovieClipLoader = new MovieClipLoader();
var mcListener:Object = new Object();
mcListener.onLoadProgress = function(target_mc:MovieClip,
 numBytesLoaded:Number, numBytesTotal:Number) {
 my_pb.setProgress(numBytesLoaded, numBytesTotal);
};
mcListener.onLoadComplete = function(target_mc:MovieClip) {
 //my_pb._visible = false;
};
img_mcl.addListener(mcListener);
this.createEmptyMovieClip("image_mc", 20);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
 image_mc);
```



如果您要在加载内容后隐藏组件，可以注释掉 `//my_pb._visible = false;` 代码行。

4. 按 **Ctrl+Enter** 键以进行测试。

# 自定义 ProgressBar 组件

在创作过程中和运行时，您都可以在水平方向上改变 **ProgressBar** 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `UIObject.setSize()`。

进度栏的左右两端以及跟踪图形都设置为固定大小。当您调整进度栏的大小时，会调整进度栏中间部分的大小，以适合左右两端。如果进度栏太小，则可能会无法正确呈现。

## 对 ProgressBar 组件使用样式

您可以设置样式属性来改变进度栏实例的外观。如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

**ProgressBar** 组件支持下列样式：

样式	主题	说明
themeColor	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
color	光晕和范例	文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。
disabledColor	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。
embedFonts	光晕和范例	一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。
fontFamily	光晕和范例	文本的字体名称。默认值为 "_sans"。
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式："normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细："none" 或 "bold"。默认值为 "none"。在调用 setStyle() 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 getStyle() 的调用将返回 "none"。
textDecoration	光晕和范例	文本修饰："none" 或 "underline"。默认值为 "none"。
barColor	范例	表示已完成百分比的前景色。默认颜色为白色。若要对以光晕为主题的组件设置进度栏颜色，请设置 themeColor 样式属性。
trackColor	范例	进度栏的背景色。默认值为 0x666666（深灰）。



# 对 ProgressBar 组件使用外观

ProgressBar 组件使用外观来表示进度栏跟踪、已完成栏和不确定栏。若要在创作 ProgressBar 组件时设置其外观，请修改 Flash UI Components 2/Themes/MMDefault/ProgressBar Elements 文件夹中的元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。

跟踪图形和栏图形均由对应于左、右端和中间部分的三个外观组成。两端均“按原样”使用，但会沿水平方向调整中间部分的大小，以适合 ProgressBar 实例的宽度。

在 ProgressBar 实例的 indeterminate 属性设置为 true 时，会使用不确定栏。外观会沿水平方向调整大小以适合进度栏的宽度。

ProgressBar 组件支持以下外观属性：

属性	说明
progTrackMiddleName	跟踪的可展开的中部。默认值为 ProgTrackMiddle。
progTrackLeftName	固定大小的左端。默认值为 ProgTrackLeft。
progTrackRightName	固定大小的右端。默认值为 ProgTrackRight。
progBarMiddleName	可展开的中间栏图形。默认值为 ProgBarMiddle。
progBarLeftName	固定大小的左栏端。默认值为 ProgBarLeft。
progBarRightName	固定大小的右栏端。默认值为 ProgBarRight。
progIndBarName	不确定的栏图形。默认值为 ProgIndBar。

## 为 ProgressBar 外观创建影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 HaloTheme.fla 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在您的操作系统中的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 Flash UI Components 2/Themes/MMDefault 文件夹，并将 ProgressBar Assets 文件夹拖到您的文档库中。
4. 在文档库中展开 ProgressBar Assets/Elements 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如，打开 ProgIndBar 元件。
6. 按需要自定义元件。  
例如，沿水平方向翻转跟踪图形。
7. 对所有要自定义的元件重复步骤 5-6。
8. 单击“返回”按钮返回主时间轴。

9. 将 `ProgressBar` 组件拖到舞台上。
- 若要查看本示例中所修改的外观，请使用 `ActionScript` 将 `indeterminate` 属性设置为 `true`。
10. 选择 “控制” > “测试影片”。

# ProgressBar 类

继承 `MovieClip` > `UIObject` 类 > `ProgressBar`

`ActionScript` 类名称 `mx.controls.ProgressBar`

使用 `ActionScript` 设置 `ProgressBar` 类的属性会覆盖在 “属性” 检查器或 “组件” 检查器中设置的同名参数。

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.ProgressBar.version);
```

 代码 `trace(myProgressBarInstance.version);` 返回 `undefined`。

## ProgressBar 类的方法摘要

下表列出了 `ProgressBar` 类的方法。

方法	说明
<code>ProgressBar.setProgress()</code>	在进度栏处于手动模式时，设置进度栏的状态以反映已经完成的进度量

## 从 `UIObject` 类继承的方法

下表列出了 `ProgressBar` 类从 `UIObject` 类继承的方法。从 `ProgressBar` 对象调用这些方法时，请使用 `ProgressBar.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在 “属性” 检查器和 “组件” 检查器中设置了参数之后，调用一个函数。

方法	说明
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## ProgressBar 类的属性摘要

下表列出了 `ProgressBar` 类的属性。

属性	说明
<code>ProgressBar.conversion</code>	一个数字，用于转换当前所加载字节的值和所加载字节总值。
<code>ProgressBar.direction</code>	进度栏填充的方向。
<code>ProgressBar.indeterminate</code>	指示加载源的大小是否是未知的。
<code>ProgressBar.label</code>	进度栏随附的文本。
<code>ProgressBar.labelPlacement</code>	与进度栏相关的标签位置。
<code>ProgressBar.maximum</code>	手动模式中进度栏的最大值。
<code>ProgressBar.minimum</code>	手动模式中进度栏的最小值。
<code>ProgressBar.mode</code>	进度栏加载内容的模式。
<code>ProgressBar.percentComplete</code>	只读；指示加载百分比的数字。
<code>ProgressBar.source</code>	要加载的内容。
<code>ProgressBar.value</code>	只读；指示已完成的进度量。

## 从 UIObject 类继承的属性

下表列出了 `ProgressBar` 类从 `UIObject` 类继承的属性。从 `ProgressBar` 对象调用这些属性时，请使用 `ProgressBar.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## ProgressBar 类的事件摘要

下表列出了 `ProgressBar` 类的事件。

事件	说明
<code>ProgressBar.complete</code>	加载完成时触发。
<code>ProgressBar.progress</code>	在手动模式或轮询模式中加载内容时触发。

## 从 UIObject 类继承的事件

下表列出了 `ProgressBar` 类从 `UIObject` 类继承的事件。从 `ProgressBar` 对象调用这些事件时，请使用 `ProgressBar.eventName` 的形式。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。

事件	说明
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## ProgressBar.complete

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object) {
 // ...
};
progressBarInstance.addEventListener("complete", listenerObject);
```

用法 2:

```
on (complete) {
 // ...
}
```

### 事件对象

除了标准的事件对象属性之外，还为 `ProgressBar.complete` 事件定义了另外两个属性：`current`（加载的值等于 **total**）和 `total`（总值）。

### 说明

事件：内容加载结束时，向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*progressBarInstance*) 调度一个事件 (在本例中为 *complete*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作“处理函数”) 处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **ProgressBar** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到实例 *progressBarInstance*，它将“\_level0.progressBarInstance”发送到“输出”面板：

```
on (complete) {
 trace(this);
}
```

## 示例

此示例创建一个 **Loader** 组件，并为该组件创建一个 **ProgressBar** (*my\_pb*) 以及一个侦听器，而该侦听器可在 *complete* 事件发生时使进度栏不可见。该示例将图像加载到加载器 *my\_ldr* 中。您必须首先将一个 **Loader** 组件和一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 要求:
 - 库中有 ProgressBar 组件
 - 库中有 Loader 组件
*/

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.Loader, "my_ldr", 10, {autoLoad:false});
this.createClassObject(mx.controls.ProgressBar, "my_pb", 20,
 {indeterminate:true, source:my_ldr, mode:"polled"});

// 创建侦听器对象
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
 my_pb.visible = false;
};
// 添加侦听器
my_pb.addEventListener("complete", pbListener);

my_ldr.load("http://www.helpexamples.com/flash/images/image2.jpg");
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# ProgressBar.conversion

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.conversion*

## 说明

属性：一个数字，设置进入值的转换值。它除当前值和总值，将它们向下取整，然后在 `label` 属性中显示转换后的值。默认值为 1。



下限值是小于或等于指定值的最接近的整数。例如，数字 4.6 转换为 4。

## 示例

以下代码通过用已加载的字节数除以转换值 1024 生成以千字节为单位的值来显示加载声音对象的进度。

您必须首先将一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 20);

// 设置进度栏属性
my_pb.mode = "polled";
my_pb.source = "my_sound";
my_pb.label = "%1 kb loaded";
my_pb.conversion = 1024;

// 加载声音
var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/disco.mp3",
 true);
```

# ProgressBar.direction

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.direction*

## 说明

属性：指示进度栏的填充方向。right 值指定该栏将从左向右填充。left 值指定该栏将从右向左填充。默认值为 right。

## 示例

以下代码加载一个声音对象，然后以向左填充的进度栏指示进度。

您必须首先将一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 20);

// 设置进度栏属性
my_pb.mode = "polled";
my_pb.source = "my_sound";
my_pb.direction = "left";

// 加载声音
var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/disco.mp3",
 true);
```



# ProgressBar.indeterminate

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.indeterminate*

## 说明

属性：一个布尔值，它指示进度栏是填充条纹图案并且加载源的大小未知 (*true*)，还是实心填充并且加载源大小已知 (*false*)。例如，如果您正将一个大的数据集加载到 SWF 文件中，而且并不知道正在加载的数据的大小时，则可能会使用此属性。

## 示例

以下代码创建一个不确定的进度栏，该进度栏为条纹图案填充、从左向右移动。

您必须首先将一个 **Loader** 组件和一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 * - 库中有 Loader 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

// 创建侦听器对象
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
 trace("Height: " + evt_obj.target.height + ", Width: " +
 evt_obj.target.width);};
// 添加侦听器
my_pb.addEventListener("complete", pbListener);

// 设置进度栏设置
my_pb.mode = "polled";
my_pb.indeterminate = true;
my_pb.source = my_ldr;
```

```
// 设置加载器设置
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.move(100, 100)
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

# ProgressBar.label

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.label*

## 说明

属性；指示加载进度的文本。此属性是一个字符串，其格式是 "已加载 %1, 共 %2 (%3%%)"。在此字符串中，%1 是当前已加载字节数的占位符，%2 是总共要加载的字节数的占位符，%3 是已加载内容所占百分比的占位符。（字符 %% 允许 **Flash** 显示单个 % 字符。）如果 %2 的值为未知，则用 ?? 替换。如果值为 undefined，则不显示标签。默认值为 "LOADING %3%%"。

## 示例

以下代码将图像加载到加载器中，然后用进度栏指示进度，进度栏标签指定已加载的总千字节数百分比。

您必须首先将一个 **Loader** 组件和一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 * - 库中有 Loader 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

my_ldr.move(0, 30);

// 设置进度栏设置
my_pb.mode = "polled";
```

```
my_pb.source = my_ldr;
my_pb.label = "%1 of %2 KB loaded";
my_pb.conversion = 1024; // 1024 字节为 1 KB

// 设置加载器设置
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg")
```

### 另请参见

[ProgressBar.labelPlacement](#)

## ProgressBar.labelPlacement

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*progressBarInstance*.labelPlacement

### 说明

属性；设置标签相对于进度栏的位置。可能的值有 "left"、"right"、"top"、"bottom" 和 "center"。

### 示例

以下示例将图像加载到加载器中，并用进度栏指示进度。它将 labelPlacement 属性设置为 top，以将标签放置在进度栏的上方。

您必须首先将一个 **Loader** 组件和一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 * - 库中有 Loader 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);
```

```
my_ldr.move(0, 30);

// 设置进度栏设置
my_pb.mode = "polled";
my_pb.source = my_ldr;
my_pb.label = "%1 of %2 KB loaded";
my_pb.conversion = 1024; // 1024 字节为 1 KB
my_pb.labelPlacement = "top";

// 设置加载器设置
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

另请参见

[ProgressBar.label](#)

## ProgressBar.maximum

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*progressBarInstance*.maximum

说明

属性：[ProgressBar.mode](#) 属性设置为 "manual" 时进度栏的最大值。

示例

以下示例将一个 **ProgressBar** 组件手动增加至 maximum 值 200，并在该点停止。它在值增加时，在“输出”面板中显示增量。

将 **ProgressBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_pb。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有一个 ProgressBar 组件（实例名称：my_pb）
 */

var my_pb:mx.controls.ProgressBar;
```

```
// 设置进度栏模式
my_pb.mode = "manual";
my_pb.label = "%1 out of %2 loaded";

// 进度栏增加前的最小数值
my_pb.minimum = 100;

// 进度栏停止前的最大值
my_pb.maximum = 200;

var increment_num:Number = my_pb.minimum;
this.onEnterFrame = function() {
 if (increment_num < my_pb.maximum) {
 increment_num++;
 // 更新数字增加的进度
 my_pb.setProgress(increment_num, my_pb.maximum);
 trace(increment_num);
 } else {
 delete this.onEnterFrame;
 }
};
```

另请参见

[ProgressBar.minimum](#)、[ProgressBar.mode](#)

# ProgressBar.minimum

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*progressBarInstance.minimum*

说明

属性；[ProgressBar.mode](#) 属性设置为 "manual" 时进度栏的最小值。

## 示例

以下示例手动增加一个 **ProgressBar** 组件，以最小值 100 开始。它在值增加时，在“输出”面板中显示增量。

将 **ProgressBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_pb。将以下代码添加到时间轴的第 1 帧中：

```
/**
 要求：
 - 舞台上 有 ProgressBar 组件（实例名称：my_pb）
*/

var my_pb:mx.controls.ProgressBar;

// 设置进度栏模式
my_pb.mode = "manual";
my_pb.label = "%1 out of %2 loaded";

// 进度栏增加前的最小数值
my_pb.minimum = 100;

// 进度栏停止前的最大值
my_pb.maximum = 200;

var increment_num:Number = my_pb.minimum;
this.onEnterFrame = function() {
 if (increment_num < my_pb.maximum) {
 increment_num++;
 // 更新数字增加的进度
 my_pb.setProgress(increment_num, my_pb.maximum);
 trace(increment_num);
 } else {
 delete this.onEnterFrame;
 }
};
```

## 另请参见

[ProgressBar.maximum](#)、[ProgressBar.mode](#)

# ProgressBar.mode

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.mode*

## 说明

属性；进度栏加载内容的模式。此值可以是 "event"（事件模式）、"polled"（轮询模式）或 "manual"（手动模式）。

事件模式和轮询模式是最常用的模式。在事件模式下，source 属性指定发出 progress 和 complete 事件的加载内容；在此模式下，应使用 **Loader** 对象。在轮询模式下，source 属性指定公开 getBytesLoaded() 和 getBytesTotal() 方法的加载内容（如 **MovieClip** 对象）。在轮询模式下，任何公开这些方法的对象均可以用作源（包括自定义对象或根时间轴）。

您也可以在手动模式下使用 **ProgressBar** 组件，方法是：手动设置 maximum、minimum 和 indeterminate 属性，并调用 [ProgressBar.setProgress\(\)](#) 方法。

## 示例

以下示例将图像加载到加载器中，然后用设置为 event 模式的进度栏指示加载进度。当加载完成后，complete 事件的侦听器将显示加载器对象的名称。

您必须首先将一个 **Loader** 组件和一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 * - 库中有 Loader 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

// 创建侦听器对象
var ldrListener:Object = new Object();
ldrListener.complete = function(evt_obj:Object) {
 trace("Event complete for " + evt_obj.target);
};
```

```
// 添加侦听器
my_ldr.addEventListener("complete", ldrListener);

// 设置进度栏设置
my_pb.mode = "event";
my_pb.indeterminate = true;
my_pb.source = my_ldr;

// 设置加载器设置
my_ldr.move(0,30);
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

# ProgressBar.percentComplete

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.percentComplete*

## 说明

属性（只读）：表示已加载的内容的百分比。该值已向下取整。（下限值是小于或等于指定值的最接近的整数。例如，数字 7.8 转换为 7。）以下是用于计算百分比的公式：

$$100 * (\text{value} - \text{minimum}) / (\text{maximum} - \text{minimum})$$

## 示例

以下示例将图像加载到与进度栏相关联的加载器中。progress 事件的侦听器和 complete 事件的侦听器都会访问 percentComplete 属性，以显示已完成的加载百分比。

将 **ProgressBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_pb。将 **Loader** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_ldr。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上 有 Loader 组件实例（实例名称: my_ldr）
 * - 舞台上 有 Progress 组件实例（实例名称: my_pb）
 */

System.security.allowDomain("http://www.helpexamples.com");
```



```

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = my_ldr;
my_ldr.autoLoad = false;

var pbListener:Object = new Object();
pbListener.progress = function(evt_obj:Object) {
 trace("progress = " + my_pb.percentComplete + "%");
}
pbListener.complete = function(evt_obj:Object) {
 trace("complete = " + my_pb.percentComplete + "%");
}
my_pb.addEventListener("progress", pbListener);
my_pb.addEventListener("complete", pbListener);

// 如果 autoLoad 为 false, 则直到调用 load() 时才开始加载
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");

```

## ProgressBar.progress

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

用法 1:

```

var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object) {
 // ...
};
progressBarInstance.addEventListener("progress", listenerObject);

```

用法 2:

```

on (progress) {
 // ...
}

```

### 事件对象

除了标准的事件对象属性外, 还为 ProgressBar.progress 事件定义了另外两个属性: current (加载的值等于 total) 和 total (总值)。

## 说明

事件：当进度栏的值更改时向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*progressBarInstance*) 调度一个事件（在本例中为 *progress*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **ProgressBar** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到实例 `progressBarInstance`，它将“`_level0.progressBarInstance`”发送到“输出”面板：

```
on (progress) {
 trace(this);
}
```

## 示例

此示例将图像加载到具有关联进度栏的加载器中，然后为 *progress* 事件创建一个侦听器。当 **progress** 事件发生时，该示例显示 *value* 属性。该属性的值介于 `ProgressBar.minimum` 和 `ProgressBar.maximum` 之间。

将 **ProgressBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_pb`。将 **Loader** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_ldr`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 要求：
 - 舞台上有 Loader 组件实例（实例名称：my_ldr）
 - 舞台上有 Progress 组件实例（实例名称：my_pb）
*/

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
my_pb.source = my_ldr;
my_ldr.autoLoad = false;

// 创建侦听器对象
var pbListener:Object = new Object();
```

```
pbListener.progress = function(evt_obj:Object) {
 // evt_obj.target 是生成了 progress 事件的组件，
 // 即进度栏。
 trace("Current progress value = " + evt_obj.target.value);
};
// 添加侦听器
my_pb.addEventListener("progress", pbListener);

// 如果 autoLoad 为 false, 则直到调用 load() 时才开始加载
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

另请参见

[EventDispatcher.addEventListener\(\)](#)

## ProgressBar.setProgress()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

```
progressBarInstance.setProgress(completed, total)
```

参数

*completed* 一个数字，指示已完成的进度量。您可以使用 [ProgressBar.label](#) 和 [ProgressBar.conversion](#) 属性以百分比形式或所选择的任何单位来显示该数字，具体取决于进度栏的源。

*total* 一个数字，指示要达到 100% 必须完成的总进度。

返回

一个数字，指示已完成的进度。

说明

方法： [ProgressBar.mode](#) 属性设置为 "manual" 时，设置进度栏的状态以反映完成的进度量。您可以调用该方法，使栏除了反映加载的状态外还反映进度的状态。例如，您可能想显式将进度栏设置为零进度。

将 *completed* 参数分配给 *value* 属性，并将 *total* 参数分配给 *maximum* 属性。*minimum* 属性不变。

## 示例

以下示例将进度栏模式设置为 `manual`，然后从 `onEnterFrame()` 函数调用 `setProgress()`，该函数会以 SWF 文件的帧频重复地调用。此示例将进度栏的最小值设置为 `100`，最大值为 `200`，并以增量值 `1` 来指示进度。

将 **ProgressBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 `my_pb`。将以下代码添加到时间轴的第 1 帧中：

```
/**
 * 要求：
 * - 舞台上有 ProgressBar（实例名称：my_pb）
 */

var my_pb:mx.controls.ProgressBar;

// 设置进度栏模式
my_pb.mode = "manual";
my_pb.label = "%1 out of %2 loaded";

// 进度栏增加前的最小数值
my_pb.minimum = 100;

// 进度栏停止前的最大值
my_pb.maximum = 200;

var increment_num:Number = my_pb.minimum;
this.onEnterFrame = function() {
 if (increment_num < my_pb.maximum) {
 increment_num++;
 // 更新数字增加的进度
 my_pb.setProgress(increment_num, my_pb.maximum);
 trace(increment_num);
 } else {
 delete this.onEnterFrame;
 }
};
```

# ProgressBar.source

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*progressBarInstance.source*

## 说明

属性：对要加载的实例的引用，该实例的加载进程将会显示。加载内容应该发出一个 progress 事件，从该事件可以检索当前值和总值。仅当 `ProgressBar.mode` 设置为 "event" 或 "polled" 时才使用该属性。默认值为 undefined。

**ProgressBar** 组件可与应用程序（包括 `_root`）中的内容一起使用。

## 示例

以下示例将图像加载到加载器中，并用进度栏指示进度。该示例将 `source` 属性设置为 **Loader** 组件 (`my_ldr`) 的名称，以使内容与进度栏相关联。

您必须首先将一个 **Loader** 组件和一个 **ProgressBar** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 ProgressBar 组件
 * - 库中有 Loader 组件
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.ProgressBar, "my_pb", 10);
this.createClassObject(mx.controls.Loader, "my_ldr", 20);

// 创建侦听器对象
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object) {
 evt_obj.target.visible = false;
};
// 添加侦听器
my_pb.addEventListener("complete", pbListener);

// 设置进度栏设置
my_pb.mode = "polled";
my_pb.indeterminate = true;
```

```
my_pb.source = my_ldr;

// 设置加载器设置
my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

另请参见

[ProgressBar.mode](#)

## ProgressBar.value

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*progressBarInstance.value*

说明

属性（只读）；指明已完成的进度。此属性为介于 [ProgressBar.minimum](#) 和 [ProgressBar.maximum](#) 值之间的一个数字。默认值为 0。

示例

以下示例将图像加载到 **Loader** 组件中，并用进度栏指示进度。当加载完成后，该示例显示进度栏的最小、最大和当前值。

将 **ProgressBar** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_pb。将 **Loader** 组件的一个实例拖到舞台上，然后在“属性”检查器中输入实例名称 my\_ldr。将以下代码添加到时间轴的第 1 帧中：

```
/**
 要求：
 - 舞台上有 Loader 组件实例（实例名称：my_ldr）
 - 舞台上有 Progress 组件实例（实例名称：my_pb）
*/

System.security.allowDomain("http://www.helpexamples.com");

var my_ldr:mx.controls.Loader;
var my_pb:mx.controls.ProgressBar;

my_pb.mode = "polled";
```

```
my_pb.source = my_ldr;
my_ldr.autoLoad = false;

// 创建侦听器对象
var pbListener:Object = new Object();
pbListener.complete = function(evt_obj:Object){
 // event_obj.target 是生成了 complete 事件的组件,
 // 即进度栏。
 trace("Minimum value is: " + evt_obj.target.minimum + " bytes");
 trace("Maximum value is: " + evt_obj.target.maximum + " bytes");
 trace("Current ProgressBar value = " + evt_obj.target.value + " bytes");
}
// 添加侦听器
my_pb.addEventListener("complete", pbListener);

// 如果 autoLoad 为 false, 则直到调用 load() 时才开始加载
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```





使用 **RadioButton** 组件可以强制用户只能选择一组选项中的一项。该组件必须用于至少有两个 **RadioButton** 实例的组。在任何给定的时刻，都只有一个组成员被选中。选择组中的一个单选按钮将取消选择组内当前选定的单选按钮。您可以设置 `groupName` 参数，以指示单选按钮属于哪个组。

可以启用或禁用单选按钮。禁用的单选按钮不接收鼠标或键盘输入。当用户单击或使用 **Tab** 键切换到 **RadioButton** 组件组时，只有选定的单选按钮会接收焦点。然后用户可以使用以下按键来控制它：

键	说明
向上箭头 / 向左箭头	所选项会移至单选按钮组内的前一个单选按钮。
向下箭头 / 向右箭头	选择将移到单选按钮组的下一个单选按钮。
Tab	将焦点从单选按钮组移动到下一个组件。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager 类**”或“创建自定义焦点导航”。

每个 **RadioButton** 实例在舞台上的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。但是，实时预览中不会显示互斥的所选项。如果将同组的两个单选按钮的 `selected` 参数设置为 `true`，它们都会显示为选中状态，尽管在运行时只有最后创建的实例才显示选中状态。有关更多信息，请参见第 950 页的“**RadioButton 参数**”。

将 **RadioButton** 组件添加到应用程序时，您可以使用“辅助功能”面板，以便让屏幕阅读器能够访问到该组件。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.RadioButtonAccImpl.enableAccessibility();
```

不管组件有多少实例，都只对组件启用一次辅助功能。有关更多信息，请参见《使用 Flash》中的第 19 章“创建辅助内容”。

# 使用 RadioButton 组件

单选按钮是任何表单或 Web 应用程序中的一个基础部分。如果您需要让用户从一组选项中做出一个选择，可以使用单选按钮。例如，在表单上询问客户要使用哪种信用卡时，您就可以使用单选按钮。

## RadioButton 参数

您可以在“属性”检查器或“组件”检查器中为每个 **RadioButton** 组件实例设置以下创作参数：

**data** 是与单选按钮相关的值。没有默认值。

**groupName** 是单选按钮的组名称。默认值为 `radioGroup`。

**label** 设置按钮上的文本值。默认值为 `Radio Button`（单选按钮）。

**labelPlacement** 确定按钮上标签文本的方向。该参数可以是下列四个值之一：`left`、`right`、`top` 或 `bottom`。默认值为 `right`。有关更多信息，请参见 [RadioButton.labelPlacement](#)。

**selected** 将单选按钮的初始值设置为被选中 (`true`) 或取消选中 (`false`)。被选中的单选按钮中会显示一个圆点。一个组内只有一个单选按钮可以有表示被选中的值 `true`。如果组内有多个单选按钮被设置为 `true`，则会选中最后实例化的单选按钮。默认值为 `false`。

您可以编写 **ActionScript**，通过利用 **RadioButton** 类的方法、属性和事件来设置 **RadioButton** 实例的其它选项。有关更多信息，请参见第 954 页的“[RadioButton 类](#)”。

## 创建具有 RadioButton 组件的应用程序

以下过程解释了如何在创作时将 **RadioButton** 组件添加到应用程序。在本示例中，单选按钮用于显示是非问题“Are you a Flashist?”（您是 **Flash** 专业人员吗？）。单选按钮组的数据和实例名称 `theVerdict` 一起显示在 **TextArea** 组件中。

**创建具有 RadioButton 组件的应用程序：**

1. 将两个 **RadioButton** 组件从“组件”面板拖到舞台上。
2. 选择一个单选按钮。在“组件”检查器中，执行以下操作：
  - 为 **label** 参数输入 **Yes**。
  - 为 **data** 参数输入 **Flashist**。
3. 选择另一个单选按钮。在“组件”检查器中，执行以下操作：
  - 为 **label** 参数输入 **No**。
  - 为 **data** 参数输入 **Anti-Flashist**。

4. 将一个 **TextArea** 组件从“组件”面板拖到舞台上，并为其指定实例名称 `theVerdict`。
5. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
flashistListener = new Object();
flashistListener.click = function (evt){
 theVerdict.text = evt.target.selection.data
}
radioGroup.addEventListener("click", flashistListener);
```

最后一行代码将一个 `click` 事件处理函数添加到 `radioGroup` 单选按钮组。该处理函数会将 `theVerdict` (**TextArea** 组件实例) 的 `text` 属性设置为 `radioGroup` 单选按钮组中所选单选按钮的 `data` 属性值。有关更多信息，请参见 [RadioButton.click](#)。

# 自定义 RadioButton 组件

在创作过程中和运行时，可以在水平和垂直方向上改变 **RadioButton** 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参见 [UIObject.setSize\(\)](#)）。

**RadioButton** 组件的边框是不可见的，它同时也指定了组件的点击区。如果您增加组件的大小，也就增加了点击区的大小。

如果组件边框太小而无法容纳组件标签，标签将被裁剪以适合边框。

## 对 RadioButton 组件使用样式

您可以设置样式属性以更改 **RadioButton** 的外观。如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

**RadioButton** 组件使用下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。

样式	主题	说明
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	文本的字体名称。默认值为 <code>"_sans"</code> 。
<code>fontSize</code>	光晕和范例	字体的磅值。默认值为 10。
<code>fontStyle</code>	光晕和范例	字体样式： <code>"normal"</code> 或 <code>"italic"</code> 。默认值为 <code>"normal"</code> 。
<code>fontWeight</code>	光晕和范例	字体粗细： <code>"none"</code> 或 <code>"bold"</code> 。默认值为 <code>"none"</code> 。在调用 <code>setStyle()</code> 期间，所有组件还可以接受值 <code>"normal"</code> 来代替 <code>"none"</code> ，但随后对 <code>getStyle()</code> 的调用将返回 <code>"none"</code> 。
<code>textDecoration</code>	光晕和范例	文本修饰： <code>"none"</code> 或 <code>"underline"</code> 。默认值为 <code>"none"</code> 。
<code>symbolBackgroundColor</code>	范例	单选按钮的背景颜色。默认值为 <code>0xFFFFFFFF</code> （白色）。
<code>symbolBackgroundDisabledColor</code>	范例	单选按钮在禁用时的背景颜色。默认值为 <code>0xEFEFEF</code> （浅灰）。
<code>symbolBackgroundPressedColor</code>	范例	单选按钮在按下时的背景颜色。默认值为 <code>0xFFFFFFFF</code> （白色）。
<code>symbolColor</code>	范例	单选按钮中的点的颜色。默认值为 <code>0x000000</code> （黑色）。
<code>symbolDisabledColor</code>	范例	组件禁用时单选按钮中的点的颜色。默认值为 <code>0x848384</code> （深灰）。

# 对 RadioButton 组件使用外观

通过修改库中的组件元件，可以在创作时设置 **RadioButton** 组件的外观。**RadioButton** 组件的外观位于 **HaloTheme.fla** 或 **SampleTheme.fla** 中，这些文件位于库中的下列文件夹中：**Flash UI Components 2/Themes/MMDefault/RadioButton Assets/States**。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。

如果单选按钮已启用但未被选中，当用户将指针移到它上方时，它会显示其滑过状态。当用户单击未被选中的单选按钮时，该单选按钮将接收输入焦点并显示其“false”按下状态。当用户松开鼠标时，单选按钮显示其“true”状态，组内先前被选中的单选按钮显示其“false”状态。如果用户在按下鼠标时将指针移离单选按钮，单选按钮的外观会恢复到其“false”状态并保留输入焦点。

如果单选按钮或单选按钮组被禁用，则不论用户进行什么交互操作，它都会显示禁用状态。

**RadioButton** 组件使用以下外观属性：

名称	说明
falseUpIcon	未选中的状态。默认值为 RadioFalseUp。
falseDownIcon	按下且未选中的状态。默认值为 RadioFalseDown。
falseOverIcon	悬停且未选中的状态。默认值为 RadioFalseOver。
falseDisabledIcon	禁用且未选中的状态。默认值为 RadioFalseDisabled。
trueUpIcon	选中状态。默认值为 RadioTrueUp。
trueDisabledIcon	禁用且选中的状态。默认值为 RadioTrueDisabled。

这些外观中的每一个都对应于指示 **RadioButton** 状态的图标。**RadioButton** 不具有边框或背景。

## 创建 RadioButton 外观的影片剪辑元件：

1. 创建一个新的 **FLA** 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 **HaloTheme.fla** 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在您的操作系统中的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 **Flash UI Components 2/Themes/MMDefault** 文件夹，然后将 **RadioButton Assets** 文件夹拖动到文档的库中。
4. 在文档的库中展开 **RadioButton Assets/States** 文件夹。
5. 打开要自定义的元件以进行编辑。  
例如，打开 **RadioFalseDisabled** 元件。

6. 按需要自定义元件。  
例如，将白色的圆的内侧更改为浅灰色。
7. 对所有要自定义的元件重复步骤 5-6。  
例如，重复更改 `RadioTrueDisabled` 元件的内部圆的颜色。
8. 单击“返回”按钮返回主时间轴。
9. 将 `RadioButton` 组件拖动到舞台上。  
对于此示例，拖动两个实例以显示两个新的外观元件。
10. 根据需要设置 `RadioButton` 实例属性。  
对于此示例，将一个 `RadioButton` 设置为选中，并使用 `ActionScript` 将两个 `RadioButton` 实例都设置为禁用。
11. 选择“控制” > “测试影片”。

## RadioButton 类

继承 `MovieClip` > `UIObject` 类 > `UIComponent` 类 > `SimpleButton` 类 > `Button` 组件 > `RadioButton`

**ActionScript 包名称** `mx.controls.RadioButton`

`RadioButton` 类的属性允许您在运行时创建文字标签并确定它相对于单选按钮的位置。您还可以为单选按钮指定数据值，将单选按钮分组，并根据数据值或实例名称进行选择。

使用 `ActionScript` 设置 `RadioButton` 类的属性将会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

`RadioButton` 组件使用焦点管理器覆盖默认的 `Flash Player` 焦点矩形，并绘制一个带圆角的自定义焦点矩形。有关创建焦点导航的信息，请参见《使用组件》中的“创建自定义焦点导航”。

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.RadioButton.version);
```



代码 `trace(myRadioButtonInstance.version);` 返回 `undefined`。

# RadioButton 类的方法摘要

没有 `RadioButton` 类专用的方法。

## 从 `UIObject` 类继承的方法

下表列出了 `RadioButton` 类从 `UIObject` 类继承的方法。从 `RadioButton` 对象调用这些方法时，请使用 `RadioButtonInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 `UIComponent` 类继承的方法

下表列出了 `RadioButton` 类从 `UIComponent` 类继承的方法。从 `RadioButton` 对象调用这些方法时，请使用 `RadioButtonInstance.methodName` 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# RadioButton 类的属性摘要

下表列出了 `RadioButton` 类的属性。

属性	说明
<code>RadioButton.data</code>	与单选按钮实例相关的值。
<code>RadioButton.groupName</code>	单选按钮组实例的组名或单选按钮实例。
<code>RadioButton.label</code>	单选按钮旁边显示的文本。
<code>RadioButton.labelPlacement</code>	标签文本相对于单选按钮或单选按钮组的方向。
<code>RadioButton.selected</code>	选择单选按钮，并取消选择先前选中的单选按钮。该属性可用于 <code>RadioButton</code> 实例或 <code>RadioButtonGroup</code> 实例。
<code>RadioButton.selectedData</code>	选择单选按钮组中具有指定数据值的单选按钮。
<code>RadioButton.selection</code>	对单选按钮组中当前选中的单选按钮的引用。该属性可用于 <code>RadioButton</code> 实例或 <code>RadioButtonGroup</code> 实例。

## 从 `UIObject` 类继承的属性

下表列出了 `RadioButton` 类从 `UIObject` 类继承的属性。从 `RadioButton` 对象访问这些属性时，请使用 `RadioButtonInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。



## 从 UIComponent 类继承的属性

下表列出了 `RadioButton` 类从 `UIComponent` 类继承的属性。从 `RadioButton` 对象访问这些属性时，请使用 `RadioButtonInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## 从 SimpleButton 类继承的属性

下表列出了 `RadioButton` 类从 `SimpleButton` 类继承的属性。从 `RadioButton` 对象访问这些属性时，请使用 `RadioButtonInstance.propertyName` 的形式。

属性	说明
<code>SimpleButton.emphasized</code>	指示按钮是否具有默认普通按钮的外观。
<code>SimpleButton.emphasizedStyleDeclaration</code>	当 <code>emphasized</code> 属性设置为 <code>true</code> 时的样式声明。
<code>SimpleButton.selected</code>	一个布尔值，它指示按钮是 ( <code>true</code> ) 否 ( <code>false</code> ) 处于选中状态。默认值为 <code>false</code> 。
<code>SimpleButton.toggle</code>	一个布尔值，它指示按钮的行为与切换开关相同 ( <code>true</code> ) 还是不同 ( <code>false</code> )。默认值为 <code>false</code> 。

## 从 Button 类继承的属性

下表列出了 `RadioButton` 类从 `Button` 类继承的属性。从 `RadioButton` 对象访问这些属性时，请使用 `RadioButtonInstance.propertyName` 的形式。

属性	说明
<code>Button.icon</code>	指定按钮实例的图标。
<code>Button.label</code>	指定在按钮上显示的文本。
<code>Button.labelPlacement</code>	指定标签文本相对于图标的方向。

# RadioButton 类的事件摘要

下表列出了 `RadioButton` 类的事件。

事件	说明
<code>RadioButton.click</code>	在单选按钮或单选按钮组上按下鼠标按钮时触发。

## 从 `UIObject` 类继承的事件

下表列出了 `RadioButton` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 `UIComponent` 类继承的事件

下表列出了 `RadioButton` 类从 `UIComponent` 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## 从 `SimpleButton` 类继承的事件

下表列出了 `RadioButton` 类从 `SimpleButton` 类继承的事件。

事件	说明
<code>SimpleButton.click</code>	在按钮上单击（释放）鼠标或按钮具有焦点并按下空格键时广播。

# RadioButton.click

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObj:Object) {
 // ...
};
radioButtonGroup.addEventListener("click", listenerObject);
```

用法 2:

```
on (click) {
 // ...
}
```

## 说明

事件：在单选按钮上单击鼠标（按下然后松开）或使用箭头键选中单选按钮时，向所有已注册的侦听器广播。当单选按钮组具有焦点，但组内没有单选按钮被选中时，如果按空格键或箭头键，该事件也会广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*radioButtonInstance*) 调度一个事件（在本例中为 `click`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **RadioButton** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，附加到单选按钮 `myRadioButton` 的以下代码将“\_level0.myRadioButton”发送到“输出”面板：

```
on (click) {
 trace(this);
}
```

## 示例

以下示例创建三个单选按钮，将它们放到舞台上，并创建一个 **click** 事件的侦听器。当用户单击三个按钮之一时，侦听器将显示被选中的按钮的实例名称。

首先将一个 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求:
 * - 库中有 RadioButton 组件
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
 groupName:"radioGroup"});

// 在舞台上定位单选按钮。
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

// 创建侦听器对象。
var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
 trace("The selected radio instance is " + evt_obj.target.selection);
}
// 添加侦听器。
radioGroup.addEventListener("click", rbListener);
```

# RadioButton.data

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*radioButtonInstance.data*

## 说明

属性：指定与 **RadioButton** 实例关联的数据。设置该属性会覆盖创作过程中设置的数据参数值。data 属性可以为任何数据类型。

## 示例

以下示例将数据值 0xFF00FF 和标签 #FF00FF 分配给单选按钮实例 my\_rb。然后创建一个 **click** 事件的侦听器，并在用户单击该按钮时显示按钮的数据值。

首先将一个 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 RadioButton 组件
 */

this.createClassObject(mx.controls.RadioButton, "my_rb", 10, {label:"first",
 groupName:"radioGroup"});

my_rb.data = 0xFF00FF;
my_rb.label = "#FF00FF";

var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
 trace("The data value for my_rb is " + my_rb.data);
}
// 添加侦听器。
my_rb.addEventListener("click", rbListener);
```

# RadioButton.groupName

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*radioButtonInstance.groupName*

*radioButtonGroup.groupName*

## 说明

属性：为单选按钮实例或单选按钮组设置组名。您可以使用该属性来获取或设置单选按钮实例的组名或单选按钮组的组名。调用此方法会覆盖在创作过程中设置的组件名参数值。默认值为 "radioGroup"。

## 示例

以下示例将具有三个单选按钮的按钮组名称设置为 `myrbGroup`。它定位这些按钮，然后在该单选按钮组上创建一个 `click` 事件侦听器。当用户单击某个单选按钮时，该示例将显示被单击的按钮的 `groupName` 属性。

首先将一个 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码：

```
/**
 * 要求：
 * - 库中有 RadioButton 组件
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
 groupName:"myrbGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
 groupName:"myrbGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
 groupName:"myrbGroup"});

// 在舞台上定位单选按钮。
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

// 创建侦听器对象。
var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
 trace("The selected radio button group name is " +
 evt_obj.target.groupName);
}

// 添加侦听器。
myrbGroup.addEventListener("click", rbListener);
```

# RadioButton.label

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*radioButtonInstance*.label

## 说明

属性；为单选按钮指定文字标签。默认情况下，标签出现在单选按钮的右侧。调用此方法会覆盖在创作中指定的 **label** 参数。如果标签文字太长无法匹配组件的边界框，则缩短文字。您可以调用 `RadioButton.setSize()` 来增加标签区域的大小，但文本不会换行到下一行。

要提供一个文本可以换行的标签，您可以将一个无标签的 **RadioButton** 和一个 **TextArea** 组合起来用做可以使文本换行的单个 **RadioButton**。以下示例可以创建这样一个单选按钮。示例假定库中已有 **RadioButton** 组件和 **TextArea** 组件，而且关闭了 **TextArea** 的边框。在本示例中，如果您访问过 `label` 属性，它将为 **undefined**。

```
this.createClassObject(mx.controls.RadioButton, "sameas_rb", 1,
 {groupName:"myGroup"});
sameas_rb.move(0,30)
this.createClassObject(mx.controls.TextArea, "message_ta", 2);
message_ta.setSize(200, 60);
// 关闭 TextArea 的边框。
message_ta.borderStyle = "none";
message_ta.wordWrap = true;
message_ta.text = "Click here if your shipping information is the same as
 your billing information.";
message_ta.move(20, 30);
```

## 示例

以下示例创建一个单选按钮，并为其分配标签 “Remove from list”。

首先将一个 **RadioButton** 组件从“组件”面板添加到当前文档的库中，然后在主时间轴的第一帧中添加以下代码。

```
/**
 要求:
 - 库中有 RadioButton 组件
*/

this.createClassObject(mx.controls.RadioButton, "my_rb", 10);

// 调整 RadioButton 组件的大小。
my_rb.setSize(200, my_rb.height);
my_rb.label = "Remove from list";
```

# RadioButton.labelPlacement

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*radioButtonInstance*.labelPlacement

*radioButtonGroup*.labelPlacement

## 说明

属性；一个字符串，它指明该标签相对于某个单选按钮的位置。您可以为某个单独的实例或为某个单选按钮组设置此属性。如果您为一个组设置该属性，标签就会放置在组中各个单选按钮的相应的位置。

以下是四个可能的值：

- "right" 单选按钮固定于边界区域的左上角。标签放置在单选按钮的右边。
- "left" 单选按钮固定于边界区域的右上角。标签放置在单选按钮的左边。
- "bottom" 标签放置在单选按钮的下方。单选按钮和标签组合在水平和垂直方向上居中。如果单选按钮的边界框不够大，标签就会被裁剪。
- "top" 标签放置在单选按钮的上方。单选按钮和标签组合在水平和垂直方向上居中。如果单选按钮的边界框不够大，标签就会被裁剪。

## 示例

以下代码创建三个单选按钮，并使用 labelPlacement 属性将第二个按钮的标签放置在按钮的左侧。

首先将一个 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码。

```
/**
 * 要求：
 * - 库中有 RadioButton 组件
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
 groupName:"radioGroup"});
```



```
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
 groupName:"radioGroup"});
```

```
// 在舞台上定位单选按钮。
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

second_rb.labelPlacement = "left";
```

## RadioButton.selected

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
radioButtonInstance.selected
radioButtonGroup.selected
```

### 说明

属性：一个布尔值，它将单选按钮的状态设置为被选中 (true)，并取消选中先前选中的单选按钮，或者将单选按钮的状态设置为取消选中 (false)。

### 示例

以下示例在一个单选按钮组中创建三个单选按钮，定位这些按钮，并将 selected 属性设置为 true 以使其处于选中状态。

首先将一个 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码。

```
/**
 要求：
 - 库中有 RadioButton 组件
*/

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
 groupName:"radioGroup"});
```

```
// 在舞台上定位单选按钮。
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

first_rb.selected = true;
```

## RadioButton.selectedData

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*radioButtonGroup.selectedData*

### 说明

属性：选中具有指定数据值的单选按钮，并取消选中先前被选中的单选按钮。如果所选实例没有指定 `data` 属性，则会选择并返回所选实例的 `label` 值。`selectedData` 属性可以是任何数据类型。

### 示例

以下示例在一个单选按钮组中创建三个单选按钮，定位这些按钮，并选择数据值为 **10** 的按钮，即第二个按钮。

首先将 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码。

```
/**
 * 要求：
 * - 库中有 RadioButton 组件
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first", data:5,
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
 data:10, groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third", data:15,
 groupName:"radioGroup"});
```

```
// 在舞台上定位单选按钮。
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

radioGroup.selectedData = 10;
```

## RadioButton.selection

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*radioButtonInstance.selection*

*radioButtonGroup.selection*

### 说明

属性：根据您是获取还是设置该属性，采用不同的行为。如果您获取该属性，它会返回单选按钮组中当前被选单选按钮的对象引用。如果您设置该属性，它会选择单选按钮组中指定的单选按钮（传递为对象引用），并取消选中先前被选中的单选按钮。

### 示例

以下示例在一个单选按钮组中创建三个单选按钮，定位这些按钮，并在单选按钮组上创建一个 **click** 事件侦听器。当用户单击某个单选按钮时，侦听器使用 *selection* 属性来显示被单击按钮的实例名称。

首先将一个 **RadioButton** 组件从“组件”面板拖到当前文档的库中，然后在主时间轴的第一帧中添加以下代码。

```
/**
 * 要求：
 * - 库中有 RadioButton 组件
 */

import mx.controls.RadioButton;

this.createClassObject(RadioButton, "first_rb", 10, {label:"first",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "second_rb", 20, {label:"second",
 groupName:"radioGroup"});
this.createClassObject(RadioButton, "third_rb", 30, {label:"third",
 groupName:"radioGroup"});
```

```
// 在舞台上定位单选按钮。
second_rb.move(0, first_rb.y + first_rb.height);
third_rb.move(0, second_rb.y + second_rb.height);

// 创建侦听器对象。
var rbListener:Object = new Object();
rbListener.click = function(evt_obj:Object){
 trace("The selected radio instance is " + radioGroup.selection);
}
// 添加侦听器。
radioGroup.addEventListener("click", rbListener);
```

有关 `RadioButtonGroup` 类的信息，请参见 [RadioButton 组件](#)。



# RDBMSResolver 组件（仅限 Flash Professional）

Resolver 组件和 DataSet 组件（Flash 数据结构中的数据管理功能的一部分）一起使用，用于保存对外部数据源所做的更改。解析程序包括 RDBMSResolver 组件和 XUpdateResolver 组件。解析程序获取一个增量数据包（由 `DataSet.deltaPacket` 返回），并将其转换为格式适合于解析程序类型的更新数据包。然后，此更新包通过一个连接器组件传送到外部数据源。Resolver 组件在运行时没有可视外观。

RDBMSResolver 组件创建一个 XML 更新数据包，该更新数据包可以轻松解析为用于更新关系数据库的 SQL 语句。RDBMSResolver 组件连接到 DataSet 组件的 `DeltaPacket` 属性，将其自己的更新数据包发送到连接器，从连接器接收服务器错误，并且将这些错误传送回 DataSet 组件 - 全都使用可绑定的属性。

有关 Flash 数据结构的更多信息，请参见《使用 Flash》中的“数据解析（仅限 Flash Professional）”。有关关系数据库的更多信息，请参见《使用 Flash》中的“将数据解析到关系数据库（仅限 Flash Professional）”。若要查看使用 RDBMSResolver 组件更新数据的应用程序的完整示例，请参见 [www.macromedia.com/devnet/mx/flash/articles/delta\\_packet.html](http://www.macromedia.com/devnet/mx/flash/articles/delta_packet.html)。



可以使用 RDBMSResolver 组件将数据更新发送到您编写的可解析 XML 并依据数据库生成 SQL 语句的任何对象（例如，ASP 页、Java servlet 或 ColdFusion 组件）。

# 使用 RDBMSResolver 组件（仅限 Flash Professional）

仅当 Flash 应用程序包含 **DataSet** 组件且必须将更新发送回数据源时，才使用此 **RDBMSResolver** 组件。此组件对您要返回到关系数据库的数据进行解析。

## RDBMSResolver 参数

您可以使用“组件”检查器的“参数”选项卡为每个 **RDBMSResolver** 实例设置以下创作参数：

**TableName** 是一个字符串，它表示要更新的数据库表的名称（在 XML 中）。该字符串应与要更新的 **RDBMSResolver.fieldInfo** 项的名称匹配。如果没有对此字段的更新，此参数应该为空（默认值）。

**UpdateMode** 是一个枚举数，它确定在生成 XML 更新数据包时标识关键字段的方式。可能值如下：

- **umUsingAll** 使用修改过的所有字段的旧值标识要更新的记录。这是用于更新的最安全的值，原因是，它确保了其他用户在您检索记录后没有修改过该记录。然而，这种方法很耗时，而且会生成较大的更新包。
- **umUsingModified** 使用修改过的所有字段的旧值标识要更新的记录。该值确保其他用户在您获取记录后没有修改记录中的相同字段。
- **umUsingKey** 默认值。此设置使用关键字段的旧值。这意味着“乐观并发”模型（当今的大部分数据库系统都采用该模型），并确保您正在修改的记录是从数据库中检索的记录。您的更改会覆盖任何其他用户对相同数据的更改。

**NullValue** 是一个字符串，它代表空字段值。可以自定义此参数以防止与空字符串（""）或另一个有效值混淆。默认值为 {\_NULL\_}。

**FieldInfo** 是一个集合，它代表唯一标识记录的一个或多个关键字段。如果数据源是数据库表，则该表应该有一个或多个字段唯一地标识其中的记录。另外，可能已通过其它表计算或联接了一些字段。必须标识这些字段，以便可以在 XML 更新包中设置关键字段，而且在 XML 更新包中忽略任何不应更新的字段。



**FieldInfo** 参数允许您使用属性来指定需要特殊处理的字段。集合中的每个项目都包含三个属性：

- **FieldName** 字段的名称。此名称应该与 **DataSet** 组件中的字段名称匹配。
- **OwnerName** 一个可选值，用于标识不“属于”在 **RDBMSResolver** 组件的 **TableName** 参数中定义的同一个表的字段。如果此属性与 **TableName** 参数的值相同或为空，则此字段通常会包含在 XML 更新包中。如果值不相同，则此字段会被排除在更新包之外。
- **IsKey** 一个布尔值属性，应将其设置为 **true**，以便更新表的所有关键字段。

以下示例显示为了更新客户表中的字段而创建的 **FieldInfo** 项。必须标识客户表中的关键字段。客户表具有一个关键字段 **id**；因此，您应该创建具有以下值的字段项：

```
FieldName = "id"
OwnerName = <--! leave this value blank -->
IsKey = "true"
```

另外，在查询中使用接合操作来添加 **custType** 字段。因为此字段应被排除在更新之外，所以创建具有以下值的字段项目：

```
FieldName = "custType"
OwnerName = "JoinedField"
IsKey = "false"
```

定义了字段项之后，**Flash Player** 可以使用它们自动生成用于更新表的完整 XML。

提醒

FieldInfo 参数使用称为“集合编辑器”的 Flash 功能。当您选择 FieldInfo 参数时，可以使用“集合编辑器”对话框添加新的 FieldInfo 项，并从同一位置设置其 **fieldName**、**ownerName** 和 **isKey** 属性。

## RDBMSResolver 组件的通用工作流程

以下步骤描述了 **RDBMSResolver** 组件的典型工作流程。

**要使用 RDBMSResolver 组件：**

1. 将 **WebServiceConnector** 组件的两个实例以及 **DataSet** 和 **RDBMSResolver** 组件的一个实例添加到应用程序，并为它们指定实例名称。
2. 选择第一个 **WebServiceConnector** 组件。然后使用“组件”检查器的“参数”选项卡为从外部数据源公开数据的 Web 服务输入 Web 服务定义语言 (WSDL) URL。

提醒

Web 服务必须返回要绑定到数据集的记录数组。

3. 使用“组件”检查器的“绑定”选项卡将第一个 **WebServiceConnector** 组件的结果属性绑定到 **DataSet** 组件的 **dataProvider** 属性。
4. 选择 **DataSet** 组件，然后使用“组件”检查器的“绑定”选项卡将数据元（**DataSet** 字段）绑定到应用程序中的可视组件。

5. 将 `DataSet` 的 `deltaPacket` 属性绑定到 `RDBMSResolver` 的 `deltaPacket` 属性。  
在调用 `DataSet.applyUpdates()` 方法时，更新指令将从 `DataSet` 组件发送到 `RDBMSResolver` 组件。
6. 将 `RDBMSResolver` 的 `updatePacket` 属性绑定到第二个 `WebServiceConnector` 的 `params` 属性，以将数据发送回将分析 XML 更新数据包的方法。将此类 `params` 属性设置为自动触发，这样，一旦数据绑定进行复制，连接器便发送更新数据包。
7. 添加触发器以启动数据绑定操作：使用附加至按钮的“触发器数据源”行为，或添加 `ActionScript`。

除这些步骤之外，您还可以使用 `RDBMSResolver` 组件创建绑定以应用从服务器发送回数据集的结果数据包。

若要查看使用 `RDBMSResolver` 组件将数据解析到关系数据库的分步示例，请参见 [www.macromedia.com/devnet/mx/flash/data\\_integration.html](http://www.macromedia.com/devnet/mx/flash/data_integration.html) 中关于 DevNet 的教程。

## RDBMSResolver 类（仅限 Flash Professional）

继承 MovieClip > RDBMSResolver

ActionScript 包名称 mx.data.components.RDBMSResolver

`RDBMSResolver` 类的方法、属性和事件允许您连接 `DataSet` 组件并对外部数据源进行更改。

### RDBMSResolver 组件的方法摘要

下表列出了 `RDBMSResolver` 类的方法。

方法	说明
<code>RDBMSResolver.addFieldInfo()</code>	将一个新项添加到 <code>fieldInfo</code> 集合中，而该集合用于在运行时动态设置 <code>RDBMSResolver</code> 组件。

# RDBMSResolver 组件的属性摘要

下表列出了 RDBMSResolver 类的属性。

属性	说明
<code>RDBMSResolver.deltaPacket</code>	DataSet 对象的 <code>deltaPacket</code> 属性应绑定到此属性，以便在调用 <code>DataSet.applyUpdates()</code> 时，该绑定将复制该属性，并且解析程序将创建更新数据包。
<code>RDBMSResolver.fieldInfo</code>	字段的集合，这些字段具有标识需要特殊处理（由于字段是关键字段或不可更新字段）的 DataSet 字段的属性。
<code>RDBMSResolver.nullValue</code>	一个字符串，它置于更新数据包中以指示字段值为 null。
<code>RDBMSResolver.tableName</code>	标识要更新的数据库表。
<code>RDBMSResolver.updateMode</code>	在生成 XML 更新包时，确定如何标识关键字段的值。
<code>RDBMSResolver.updatePacket</code>	由此解析程序生成的 XML 包，其中包含来自数据集的变量增量包的更改。
<code>RDBMSResolver.updateResults</code>	变量增量包，其中包含通过连接器从服务器返回的更新的结果。

# RDBMSResolver 组件的事件摘要

下表列出了 RDBMSResolver 类的事件。

事件	说明
<code>RDBMSResolver.beforeApplyUpdates</code>	在应用程序中定义；由 RDBMSResolver 组件调用，用于在将 <code>updatePacket</code> 属性的 XML 绑定到连接器之前对其进行自定义修改。
<code>RDBMSResolver.reconcileResults</code>	在应用程序中定义；由 RDBMSResolver 组件调用，用于在从服务器接收到结果并应用到变量增量包之后比较两个包。
<code>RDBMSResolver.reconcileUpdates</code>	在应用程序中定义；在应用变量增量包中的更新后从服务器接收到结果时，由 RDBMSResolver 组件调用。

# RDBMSResolver.addFieldInfo()

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
resolveData.addFieldInfo("fieldName", "ownerName", "isKey")
```

## 参数

*fieldName* 字符串；提供此信息对象所描述的字段的名称。

*ownerName* 字符串；提供拥有此字段的表的名称。如果此名称与 RDBMSResolver 实例的 *tableName* 属性相同，则可以将此参数保留为空 ("")。

*isKey* 布尔值；它指示此字段是否是关键字段。

## 返回

无。

## 说明

方法；将新项添加到更新数据包中的 XML *fieldInfo* 集合。如果您必须在运行时动态设置 RDBMSResolver 组件，请使用此方法，而不使用创作环境中的“组件”检查器。

## 示例

以下示例创建了一个 RDBMSResolver 组件，并提供了表的名称、关键字段的名称，而且防止 *personTypeName* 字段被更新：

```
var myResolver:RDBMSResolver = new RDBMSResolver();
myResolver.tableName = "Customers";
// 将 ID 字段设置为关键字段
// 并设置 personTypeName 字段，使其不会被更新。
myResolver.addFieldInfo("id", "", true);
myResolver.addFieldInfo("personTypeName", "JoinedField", false);
// 设置数据绑定
//...
```

# RDBMSResolver.beforeApplyUpdates

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004。

### 用法

`resolveData.beforeApplyUpdates(eventObject)`

### 参数

*eventObject* 解析程序事件对象；描述在通过连接器将更新发送到数据库之前对 XML 数据包进行的自定义。此事件对象应包含以下属性：

属性	说明
target	对象；产生此事件的解析程序。
type	字符串；事件的名称。
updatePacket	XML 对象；将要应用的 XML 对象。

### 返回

无。

### 说明

属性： `deltaPacket` 类型的属性。它接收要转换为更新数据包的增量数据包，并输出来自 `updateResults` 属性中放置的任何服务器结果的增量数据包。此事件处理函数提供一个方法，您可以通过此方法在将更新的数据发送到连接器前对 XML 进行自定义修改。

`updateResults` 属性中的消息视为错误。这意味着，具有消息的增量被再次添加到增量数据包中，这样，下次将增量数据包发送到服务器时可以重新发送该增量。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个增量数据包前将这些消息提供给用户并可以修改增量。

### 示例

以下范例将用户身份验证数据添加到 XML 包：

```
on (beforeApplyUpdates) {
 // 添加用户身份验证数据。
 var userInfo = new XML("" + getUserId() + ""+getPassword() + "");
 updatePacket.firstChild.appendChild(userInfo);
}
```

# RDBMSResolver.deltaPacket

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.deltaPacket*

## 说明

属性： *deltaPacket* 类型的属性。它接收要转换为更新数据包的增量数据包，并输出来自 *updateResults* 属性中放置的任何服务器结果的增量数据包。

*updateResults* 属性中的消息视为错误。这意味着，具有消息的增量被再次添加到增量数据包中，这样，下次将增量数据包发送到服务器时可以重新发送该增量。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个增量数据包前将这些消息提供给用户并可以修改增量。

# RDBMSResolver.fieldInfo

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.fieldInfo*

说明

属性：它指定不限数量的字段的集合，而这些字段具有标识需要特殊处理（由于字段是关键字段或不可更新字段）的 **DataSet** 字段的属性（有关添加字段的信息，请参见 [RDBMSResolver.addFieldInfo\(\)](#)）。集合中的每个 `fieldInfo` 项都包含三个属性：

属性	说明
<code>fieldName</code>	需特殊处理的字段的名称。此字段名称应该与 <b>DataSet</b> 组件中的字段名称匹配。
<code>ownerName</code>	可选属性。如果此字段不“属于”在 <code>RDBMSResolver.tableName</code> 属性中定义的表，则 <code>OwnerName</code> 将为此字段所有者的名称。如果 <code>OwnerName</code> 与 <code>RDBMSResolver.tableName</code> 具有相同的值或为空，则此字段通常会包含在 XML 更新数据包中。如果 <code>OwnerName</code> 不具有以上任何值，则此字段将排除在更新数据包之外。
<code>isKey</code>	布尔值；如果为 <code>true</code> ，则更新表的所有关键字段。

# RDBMSResolver.nullValue

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`resolveData.nullValue`

说明

属性：一个字符串，用于为字段的值提供空值。可以自定义此属性以防止与空字符串 ("" ) 或另一个有效值混淆。默认字符串为 `{_NULL_}`。

# RDBMSResolver.reconcileResults

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.reconcileResults(eventObject)*

## 参数

*eventObject* 解析程序事件对象；它描述用于比较两个更新数据包的事件对象。此事件对象应包含以下属性：

属性	说明
target	对象；广播此事件的解析程序。
type	字符串；事件的名称。

## 返回

无。

## 说明

事件；由 **RDBMSResolver** 组件广播，用于在服务器接收到结果并应用到变量增量包之后比较两个包。

一个 `updateResults` 数据包可能包含增量数据包中的操作结果以及有关其它客户端所执行更新的信息。接收到新的更新数据包时，操作结果和数据库更新被分为两个更新数据包，并分别放入 `deltaPacket` 属性中。`reconcileResults` 事件就在使用数据绑定发送包含操作结果的增量数据包之前广播。

## 示例

下面的示例会协调两个更新包，并在成功后返回和清除更新：

```
on (reconcileResults) {
 // 检查结果。
 if (examine(updateResults)) {
 myDataSet.purgeUpdates();
 } else {
 displayErrors(results);
 }
}
```



# RDBMSResolver.reconcileUpdates

## 可用性

Flash Player 7。

## 版本

Flash Professional 8。

## 用法

*resolveData.reconcileUpdates(eventObject)*

## 参数

*eventObject* 解析程序事件对象；描述在通过连接器将更新发送到数据库之前对 XML 数据包进行的自定义。此事件对象应包含以下属性：

属性	说明
target	对象；广播此事件的解析程序。
type	字符串；事件的名称。

## 返回

无。

## 说明

事件；在应用变量增量包中的更新后从服务器接收到结果时，由 **RDBMSResolver** 组件广播。一个 `updateResults` 数据包可能包含增量数据包中的操作结果以及有关其它客户端所执行更新的信息。接收到新的更新数据包时，操作结果和数据库更新被分为两个增量数据包，并分别放入 `deltaPacket` 属性中。`reconcileUpdates` 事件就在使用数据绑定发送包含任何数据库更新的增量数据包之前广播。

## 示例

下面的范例会协调两个结果并在成功后清除更新：

```
on (reconcileUpdates) {
 // 检查结果。
 if (examine(updateResults)) {
 myDataSet.purgeUpdates();
 } else {
 displayErrors(results);
 }
}
```

# RDBMSResolver.tableName

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.tableName*

## 说明

属性：一个字符串，在 XML 中表示要更新的数据库表的表名称。此属性还确定在更新包中发送哪个字段。为了进行此确定，RDBMSResolver 组件会将此属性的值与为 `fieldInfo.ownerName` 属性提供的值进行比较。如果某个字段在 `fieldInfo` 集合属性中没有条目，则此字段会放入更新数据包中。如果某个字段在 `fieldInfo` 集合属性中有条目，并且其 `ownerName` 值为空或者与 RDBMSResolver 组件的 `tableName` 属性相同，则此字段会放入更新数据包中。如果某个字段在 `fieldInfo` 集合属性中有条目，其 `ownerName` 值不为空，并且与 RDBMSResolver 组件的 `tableName` 属性不相同，则此字段不会放入更新数据包中。

# RDBMSResolver.updateMode

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.updateMode*

说明

属性；它包含一些值，这些值在生成 XML 更新包时确定如何标识关键字段。此属性的值可以是以下字符串：

值	说明
"umUsingAll"	使用修改过的所有字段的旧值以标识要更新的记录。这是用于更新的最安全的值，原因是它确保了其他用户在您检索记录后没有修改过记录的任何字段。然而，这种方法较为耗时，而且会生成较大的更新包。
"umUsingModified"	使用修改过的所有字段的旧值以标识要更新的记录。该值确保其他用户在您获取记录后没有修改记录中的相同字段。
"umUsingKey"	默认值。此设置使用关键字段的旧值。这意味着“乐观并发”模型（当今的大部分数据库系统都采用该模型），并确保您正在修改的记录是从数据库中检索的记录。您所做的更改将覆盖任何其他用户对相同数据进行的更改。

# RDBMSResolver.updatePacket

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

*resolveData.updatePacket*

说明

属性；XML 类型属性，包含用于绑定到某个连接器属性的 XML 包，连接器属性将更改的已转换更新包传送回服务器，以便可以更新数据源。这是包含 DataSet 更改的包的 XML 文档。

# RDBMSResolver.updateResults

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.updateResults*

## 说明

属性：变量增量包，其中包含通过连接器从服务器返回的更新的结果。使用此属性可将错误和更新的数据从服务器传送到数据集；例如，当服务器为自动分配的字段分配新 ID 时。将此属性绑定到连接器的 `results` 属性，以便它可以接收更新的结果并将结果传送回数据集。

`updateResults` 属性中的消息视为错误。这意味着，具有消息的增量被再次添加到增量数据包中，这样，下次将增量数据包发送到服务器时可以重新发送该增量。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个增量数据包前将这些消息提供给用户并可以修改增量。

## RectBorder 类

**RectBorder** 类用作大多数组件的边框。每个主题都提供了此类的单独实现，每个主题都各自具有一组支持的边框样式和属性。

您主要通过设置其它组件的样式来与 **RectBorder** 类进行交互。例如，在文档中包含 **List** 组件并设置 `borderStyle` 样式属性时，**List** 组件将创建一个 **RectBorder** 实例，该实例使用列表的 `borderStyle` 设置。您也可以创建自定义的 **RectBorder** 实现来设置使用 **RectBorder** 的所有组件的边框的外观。

**RectBorder** 类有四种标准显示样式：`none`、`inset`、`outset` 和 `solid`。

“光晕”主题还添加了供特定组件使用的四种特殊的显示样式。

特殊样式	使用该样式的组件
<code>default</code>	窗口
<code>alert</code>	<b>Alert</b>
<code>dropDown</code>	<b>ComboBox</b> 和 <b>DateField</b>
<code>menuBorder</code>	<b>Menu</b> 和 <b>MenuBar</b>

此处描述的 **RectBorder** 行为和样式属性对于使用 **RectBorder** 类的所有组件都是一致的。

# 对 RectBorder 类使用样式

您可以设置样式属性来更改 **RectBorder** 实例的外观。 **RectBorder** 实例使用以下样式：

- `borderCapColor`
- `borderColor`
- `buttonColor`
- `highlightColor`
- `shadowCapColor`
- `shadowColor`
- `themeColor`

特定 **RectBorder** 实例可以使用的样式取决于使用的主题和为组件设置的边框样式。有关显示主题、边框样式和可用的颜色样式属性之间关系的交互演示，请参见《使用组件》帮助。

四种特殊的“光晕”样式（`default`、`alert`、`dropDown` 和 `menuBorder`）有一些无法通过样式设置颜色的线条。您只能通过创建自定义主题并在自定义的 **RectBorder** 实现内修改相应的 **ActionScript** 来修改这些颜色。

## 使用 `setStyle` 设置边框样式：

1. 选择“文件” > “新建”，然后创建新的 **Flash** 文档。
2. 将一个 **TextArea** 组件拖到舞台上，然后为其指定实例名称 **my\_ta**。
3. 在主时间轴的第一帧中，将以下 **ActionScript** 代码添加到“动作”面板中：

```
my_ta.setStyle("borderStyle", "alert");
```



您可以将 `borderStyle` 设置为“`alert`”，因为您在使用默认的主题（光晕）。如果您在使用一个不同的主题，则这四个“特殊”光晕样式，包括“`alert`”，将不可用。

4. 选择“控制” > “测试影片”对 **SWF** 文件进行测试。

## 将多个边框样式设置为 `createClassObject` 方法的参数：

1. 选择“文件” > “新建”，然后创建新的 **Flash** 文档。
2. 在主时间轴的第一帧中，将以下 **ActionScript** 添加到“动作”面板中：

```
createClassObject(mx.controls.TextArea, "my_ta", 1, {borderStyle: "menuBorder", themeColor: "0x990000"});
```

有关更多信息，请参见 `UIObject.createClassObject()`。或者，如果您想设置多个样式，并将它们应用到多个组件实例，您可以建立一个包含样式设置的新样式声明，然后将该样式声明附加到这些组件实例（请参见《使用组件》中的“为成组的组件设置自定义样式”）。

3. 选择“控制” > “测试影片”对 **SWF** 文件进行测试。

使用“范例”主题设置边框样式：

1. 选择“文件”>“新建”，然后创建新的 Flash 文档。

2. 将一个 Button 组件拖到舞台上，然后为其指定实例名称 **my\_btn**。

您也可以使用 **ActionScript** 创建该实例，如下所示（一定要先将 Button 组件拖到库中）：

```
createClassObject(mx.controls.Button, "my_btn", 1);
```

3. 选择“文件”>“导入”>“打开外部库”。

4. 打开 **SampleTheme.fla** 文件，它位于：

- Windows: \Program Files\Macromedia\Flash 8\语言  
  \Configuration\ComponentFLA\
- Macintosh: HD/Applications/Macromedia Flash 8/Configuration/  
  ComponentFLA/

5. 在 **SampleTheme.fla** 库中，在“Flash UI components”>“Themes”>“MMDefault”>“Button Assets”>“ButtonSkin”中找到按钮资源影片剪辑，并将其拖到当前文档的库中。

6. 在主时间轴的第一帧中，将以下 **ActionScript** 代码添加到“动作”面板中：

```
my_btn.setStyle("buttonColor", "0xFFFFFF");
my_btn.setStyle("borderStyle", "solid");
my_btn.setStyle("borderColor", "none");
```



如果您计划设置多个样式，并需要提高组件的运行时性能，您可以设置一个包含这些样式的自定义样式声明，然后将该自定义样式声明附加到组件实例（请参见《使用组件》中的“为成组的组件设置自定义样式”）。

或者，您可以将这些设置追加到 **createClassObject**，如下所示：

```
createClassObject(mx.controls.Button, "my_btn", 1, {buttonColor:
 "0xFFFFFF", borderStyle: "solid", borderColor: "none"});
```

7. 选择“控制”>“测试影片”以测试 SWF 文件。

请注意，即使对于为“none”的“borderColor”，按钮也有一个灰色的边框。在这种情况下，“none”并不表示透明，而是表示中性灰。

# 创建自定义的 RectBorder 实现

在大多数 `ActionScript 2.0` 版组件中，`RectBorder` 类都被用作边框外观。“光晕”主题和“范例”主题中的默认实现都使用 `ActionScript` 绘制边框。自定义实现必须使用 `ActionScript` 将其自身注册为 `RectBorder` 实现并提供调整大小的功能，但可以使用 `ActionScript` 或图形元素来表示可视状态。

每个 `RectBorder` 实现都必须符合以下要求：

- 它必须扩展 `mx.skins.RectBorder` 或它的一个子类。
- 它必须提供 `offset` 属性值或者实现 `getBorderMetrics` 方法以返回大小调整信息。
- 它必须实现 `drawBorder()` 方法以绘制边框或设置边框大小。
- 它必须支持所有四种标准样式和四种特殊样式。  
该实现可以重新对特殊边框使用标准边框，与“范例”主题一样。
- 它必须将自身注册为 `RectBorder` 实现。

## RectBorder 全局注册

所有组件都会搜索一个集中位置，以便为文档查找对正在使用的 `RectBorder` 类的引用：`_global.styles.rectBorderClass`。您不能指定单个组件应使用不同的 `RectBorder` 实现。若要为组件自定义 `RectBorder`，必须依赖于 `borderStyle` 样式属性。

## 自定义 RectBorder 示例

“光晕”主题和“范例”主题提供的 `RectBorder` 实现都使用 `ActionScript` 绘图 API 来绘制不同样式的边框。下面的示例演示如何创建使用图形元件进行显示的自定义 `RectBorder` 实现。

### 创建自定义 RectBorder 实现：

1. 在 `Classes/mx/skins` 文件夹中创建一个新文件夹，其名称与要用于自定义边框的自定义包的名称相对应。

对于本示例，使用 `myTheme`。

2. 在新文件夹中创建一个新的 `AS` 文件，并将其保存为 `RectBorder.as`。
3. 将以下 `ActionScript` 代码复制到新 `AS` 文件中：

```
import mx.core.ext.UIObjectExtensions;

class mx.skins.myTheme.RectBorder extends mx.skins.RectBorder
{
 static var symbolName:String = "RectBorder";
 static var symbolOwner:Object = RectBorder;
 var className:String = "RectBorder";
```



```
#include "../..core/ComponentVersion.as"

 // 所有这些边框都具有相同大小的边缘，一个像素。
 var offset:Number = 4;

 function init(Void):Void
 {
 super.init();
 }

 function drawBorder(Void):Void
 {
 // 图形位于元件的时间轴上，
 // 所以，在此您需要做的只是设置边框大小。
 _width = __width;
 _height = __height;
 }

 // 将类注册为 RectBorder 以供所有组件使用。
 static function classConstruct():Boolean
 {
 UIObjectExtensions.Extensions();
 _global.styles.rectBorderClass = RectBorder;
 _global.skinRegistry["RectBorder"] = true;
 return true;
 }
 static var classConstructed:Boolean = classConstruct();
 static var UIObjectExtensionsDependency = UIObjectExtensions;
}
```

如果您使用的不是 myTheme 包，请根据需要更改类声明。

4. 保存 AS 文件。
5. 创建一个新的 FLA 文件。
6. 使用“插入”>“新建元件”创建一个新的影片剪辑元件。
7. 将名称设置为 RectBorder。
8. 如果未显示高级字段，请单击“高级”。
9. 选择“为 ActionScript 导出”。  
标识符将自动填写为 RectBorder。
10. 将 AS 2.0 类设置为自定义边框实现的完整类名。  
此示例使用 mx.skins.myTheme.RectBorder。
11. 确认“在第一帧导出”处于选中状态，然后单击“确定”。
12. 打开 RectBorder 元件以进行编辑。

**13.** 为元件绘制图形。

例如，绘制一个无填充的极细方框。若要使自定义边框易于观看，请将线条颜色设置为亮红色。

**14.** 确保该图形与左上角（**x** 和 **y** 坐标设置为 (0,0)）对齐。

您的自定义 `drawBorder` 实现将根据组件要求设置宽度和高度。

**15.** 单击“返回”返回主时间轴。

**16.** 将使用 `RectBorder` 的几个组件拖动到舞台上。

例如，将 `List`、`TextArea` 和 `TextInput` 组件拖动到舞台上。

**17.** 选择“控制”>“测试影片”。

此示例创建一个具有静态颜色和图形的非常简单的边框实现。它对不同的 `borderStyle` 设置不会做出响应；它总是使用相同的图形，而无论 `borderStyle` 是什么。有关更多完整边框实现的示例，请查看为“光晕”和“范例”主题提供的示例。

# Screen 类（仅限 Flash Professional）

# 39

**Screen** 类是在 **Flash Professional 8** 的“屏幕大纲”窗格中创建的屏幕的基类。屏幕是用于创建应用程序和演示文稿的高级容器。有关使用屏幕的概述，请参见《使用 **Flash**》中的第 14 章“使用屏幕（仅限 **Flash Professional**）”。

**Screen** 类有两个主要的子类：**Slide** 和 **Form**。

**Slide** 类提供幻灯片演示文稿的运行时行为。**Slide** 类提供了内置的导航和排序功能，并允许您使用行为轻松地在幻灯片之间附加过渡。幻灯片对象含有“状态”，因此用户可以进入到下一或上一幻灯片 / 状态：显示下一张幻灯片时，上一张幻灯片即会隐藏。有关使用 **Slide** 类控制幻灯片演示文稿的更多信息，请参见第 1047 页的“**Slide** 类（仅限 **Flash Professional**）”。

**Form** 类为表单应用程序提供运行时环境。表单可以覆盖其它组件、包含其它组件或被其它组件包含。与幻灯片不同，表单不提供任何排序或导航功能。有关更多信息，请参见第 677 页的“**Form** 类（仅限 **Flash Professional**）”。

**Screen** 类提供幻灯片和表单共有的功能。

**屏幕知道如何管理其子屏幕** 每个屏幕包含一个内置的属性，此属性中包含该屏幕的子屏幕的列表（称为集合）。此集合由“屏幕轮廓”窗格中的屏幕层次结构确定。屏幕可以具有任意数量的子项（或没有子项），这些子项自身也可以有子项。

**屏幕可以隐藏和显示其子项** 因为屏幕本质上是嵌套的影片剪辑的集合，所以屏幕可以控制其子屏幕的可见性。对于表单应用程序，在默认情况下，屏幕的所有子屏幕同时可见；对于幻灯片演示文稿，通常是一次显示一个屏幕。

**屏幕广播事件** 例如，当特定的屏幕变得可见时，可以触发声音的播放或者开始播放一段视频。

# 将外部内容加载到屏幕（仅限 Flash Professional）

Screen 类扩展了 Loader 类（请参见第 751 页的“Loader 组件”），从而使您可以轻松地管理和加载外部的 SWF 和 JPEG 文件。Loader 类包含一个 contentPath 属性，此属性指定外部的 SWF 或 JPEG 文件的 URL，或者指定库中影片剪辑的链接标识符。

使用此功能，可以将外部屏幕树（或任何外部 SWF 文件）作为任何屏幕节点的子项加载。这提供了一种有用的方法，利用此方法可将基于屏幕的媒体模块化，并将此媒体分为多个单独的 SWF 文件。

例如，假如您有一个幻灯片演示文稿，有三个人各自提供了其中的一个部分。您可以要求每个演示者创建一个单独的幻灯片演示文稿（SWF 文件）。然后，您可以创建一个“主幻灯片演示文稿”，其中包含三个占位符幻灯片，分别用于演示者创建的各个幻灯片演示文稿。对于每个占位符幻灯片，可以将它的 contentPath 属性指向一个 SWF 文件。可以按下图所示安排主幻灯片演示文稿：



“主” SWF 文件幻灯片演示文稿结构

假设演示者提供了三个 SWF 文件：speaker\_1.swf、speaker\_2.swf 和 speaker\_3.swf。通过使用“属性”检查器或 ActionScript 设置每个占位符幻灯片的 contentPath 属性，即可轻松地组合整个演示文稿，如下代码所示：

```
Speaker_1.contentPath = speaker_1.swf;
Speaker_2.contentPath = speaker_2.swf;
Speaker_3.contentPath = speaker_3.swf;
```

默认情况下，在“属性”检查器中进行创作或者使用 **ActionScript** 的过程中，当您设置幻灯片的 `contentPath` 属性时（如上所示），加载完“主演示文稿”**SWF** 文件后就会立即加载指定的 **SWF** 文件。为了缩短最初加载时间，可以考虑设置附加到每张幻灯片的 `on(reveal)` 处理函数中的 `contentPath` 属性。

```
// 附加到 Speaker_1 幻灯片
on(reveal) {
 this.contentPath="speaker_1.swf";
}
```

或者，您也可以将幻灯片的 `autoLoad` 属性设置为 `false`。然后当幻灯片显示时，您便可以调用幻灯片的 `load()` 方法。（`autoLoad` 属性和 `load()` 方法继承自 **Loader** 类。）

```
// 附加到 Speaker_1 幻灯片
on(reveal) {
 this.load();
}
```

## 使用 **ActionScript** 引用加载的屏幕

**Loader** 类创建一个名为 `contentNode` 的内部影片剪辑，该类将 `contentPath` 属性所指定的 **SWF** 或 **JPEG** 文件加载到其中。实际上，该影片剪辑会在“占位符”幻灯片（在上面的“主”演示文稿中创建）和加载的幻灯片演示文稿中的第一张幻灯片之间添加额外的屏幕节点。

例如，假设为上图所示的 **Speaker\_1** 幻灯片占位符创建的 **SWF** 文件具有以下结构（如“屏幕轮廓”窗格中所示）：



“Speaker 1” **SWF** 文件幻灯片演示文稿结构

在运行时，当 **Speaker 1 SWF** 文件加载到占位符幻灯片中时，整个幻灯片演示文稿将会具有以下结构：



“主”演示文稿和“speaker”演示文稿的结构（运行时）

**Screen**、**Slide** 和 **Form** 类的属性和方法会尽可能地“忽略”此 **contentHolder** 节点。也就是说，名为 **MyPresentation** 的幻灯片（及其子幻灯片）是以“演示文稿”幻灯片为根的连续幻灯片树的一部分，而且不被视为单独的子树。

# Screen 类 (API)（仅限 Flash Professional）

继承 **MovieClip** > **UIObject** 类 > **UIComponent** 类 > **View** > **Loader 组件** > **Screen**

ActionScript 类名称 `mx.screens.Screen`

**Screen** 类的方法、属性和事件允许您在运行时创建和处理屏幕。

## Screen 类的方法摘要

下表列出了 **Screen** 类的方法。

方法	说明
<a href="#">Screen.getChildScreen()</a>	返回此屏幕位于特定索引处的子屏幕。

## 从 UIObject 类继承的方法

下表列出了 **Screen** 类从 **UIObject** 类继承的方法。从 **Screen** 对象调用这些方法时，请使用 *ScreenInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **Screen** 类从 **UIComponent** 类继承的方法。从 **Screen** 对象调用这些方法时，请使用 *ScreenInstance.methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## 从 Loader 类继承的方法

下表列出了 **Screen** 类从 **Loader** 类继承的方法。从 **Screen** 对象调用此方法时，请使用 *ScreenInstance.methodName* 的形式。

方法	说明
<code>Loader.load()</code>	加载由 <code>contentPath</code> 属性指定的内容。

# Screen 类的属性摘要

下表列出了 Screen 类的属性。

属性	说明
<code>Screen.currentFocusedScreen</code>	只读；返回包含全局当前焦点的屏幕。
<code>Screen.indexInParent</code>	只读；返回该屏幕在其父屏幕的子屏幕列表中的索引（从零开始）。
<code>Screen.numChildScreens</code>	只读；返回屏幕包含的子屏幕的数量。
<code>Screen.parentIsScreen</code>	只读；返回一个布尔值（true 或 false），该值指示屏幕的父对象本身是否是屏幕。
<code>Screen.parentScreen</code>	只读；返回包含指定屏幕的屏幕。
<code>Screen.rootScreen</code>	只读；返回包含此屏幕的树或子树的根屏幕。

## 从 UIObject 类继承的属性

下表列出了 Screen 类从 UIObject 类继承的属性。从 Screen 对象访问这些属性时，请使用 `ScreenInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。



## 从 UIComponent 类继承的属性

下表列出了 `Screen` 类从 `UIComponent` 类继承的属性。从 `Screen` 对象访问这些属性时，请使用 `ScreenInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## 从 Loader 类继承的属性

下表列出了 `Screen` 类从 `Loader` 类继承的属性。从 `Screen` 对象访问这些属性时，请使用 `ScreenInstance.propertyName` 的形式。

属性	说明
<code>Loader.autoLoad</code>	一个布尔值，它指示内容是自动加载 ( <code>true</code> ) 还是只在您调用 <code>Loader.load()</code> 时才进行加载 ( <code>false</code> )。
<code>Loader.bytesLoaded</code>	只读属性，指明已经加载的字节数。
<code>Loader.bytesTotal</code>	指明内容中的总字节数的只读属性。
<code>Loader.content</code>	对加载器内容的引用。该属性为只读。
<code>Loader.contentPath</code>	一个字符串，它指明要加载的内容的 URL。
<code>Loader.percentLoaded</code>	一个数字，它指明已加载内容的百分比。该属性为只读。
<code>Loader.scaleContent</code>	一个布尔值，它指示是内容会进行缩放以适合加载器 ( <code>true</code> )，还是加载器会进行缩放以适合内容 ( <code>false</code> )。

## Screen 类的事件摘要

下表列出了 `Screen` 类的事件。

事件	说明
<code>Screen.allTransitionsInDone</code>	当应用到屏幕的所有“输入”过渡结束时进行广播。
<code>Screen.allTransitionsOutDone</code>	当应用到屏幕的所有“输出”过渡结束时进行广播。
<code>Screen.mouseDown</code>	当在直接属于屏幕的对象（形状或影片剪辑）上按下鼠标按钮时进行广播。
<code>Screen.mouseDownSomewhere</code>	当在舞台上的某处（不一定要在属于此屏幕的对象上）按下鼠标按钮时进行广播。
<code>Screen.mouseMove</code>	当鼠标在屏幕上移动时进行广播。
<code>Screen.mouseOut</code>	当鼠标从屏幕内移出时进行广播。

事件	说明
<code>Screen.mouseOver</code>	当鼠标从屏幕外移入时进行广播。
<code>Screen.mouseUp</code>	当在直接属于屏幕的对象（形状或影片剪辑）上松开鼠标按钮时进行广播。
<code>Screen.mouseUpSomewhere</code>	当在舞台上的某处（不一定要在属于此屏幕的对象上）松开鼠标按钮时进行广播。

## 从 UIObject 类继承的事件

下表列出了 `Screen` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 `Screen` 类从 `UIComponent` 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## 从 Loader 类继承的事件

下表列出了 `Screen` 类从 `Loader` 类继承的事件。

事件	说明
<code>Loader.complete</code>	当内容加载完成时触发。
<code>Loader.progress</code>	在内容加载过程中触发。

# Screen.allTransitionsInDone

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(allTransitionsInDone) {
 // 此处是您的代码。
}

listenerObject = new Object();
listenerObject.allTransitionsInDone = function(eventObject){
 // 在此处插入您的代码。
}

screenObj.addEventListener("allTransitionsInDone", listenerObject)
```

## 说明

事件：当应用到此屏幕的所有“进入”过渡结束时进行广播。allTransitionsInDone 事件由与 *screenObj* 关联的过渡管理器广播。

## 示例

在以下示例中，当应用到名为 *mySlide* 的幻灯片的所有“进入”过渡结束后，*mySlide* 所包含的按钮 (*nextSlide\_btn*) 即可见。

```
// 附加到 mySlide:
on(allTransitionsInDone) {
 this.nextSlide_btn._visible = true;
}
```

## 另请参见

[Screen.allTransitionsOutDone](#)

# Screen.allTransitionsOutDone

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(allTransitionsOutDone) {
 // 此处是您的代码。
}

listenerObject = new Object();
listenerObject.allTransitionsOutDone = function(eventObject){
 // 在此处插入您的代码。
}

screenObj.addEventListener("allTransitionsOutDone", listenerObject)
```

## 说明

事件：当应用到此屏幕的所有“输出”过渡结束时进行广播。allTransitionsOutDone 事件由与 *screenObj* 关联的过渡管理器广播。

## 另请参见

[Screen.currentFocusedScreen](#)

# Screen.currentFocusedScreen

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myScreen.currentFocusedScreen
```

## 说明

静态属性（只读）；返回对包含全局当前焦点的“最内叶”屏幕对象的引用。最内叶是指在屏幕层次结构中距离根屏幕最远的屏幕。焦点可以在屏幕自身上，或者在影片剪辑、文本对象或者该屏幕内的组件上。如果没有当前焦点，则此属性默认为 `null`。

例如，假设您有一个如下所示的运行时屏幕层次结构：

```
presentation
├── screen1
│ ├── subscreen1_1
│ │ ├── mymovieclip
│ │ └── myUIButton
│ └── screen2
│ └── subscreen1_2
```

如果 `myUIButton` 具有焦点，则包含焦点的最内叶屏幕为 `subscreen1_1`，这也是 `currentFocusedScreen` 将返回的内容。在本例中，`presentation`、`screen1` 和 `subscreen1_1` 都包含焦点，但在屏幕层次结构中距离树的叶“最近的”（即距离根最远的）是 `subscreen1_1`。

## 示例

以下示例在“输出”面板中显示当前具有焦点的屏幕的名称。

```
var currentFocus:mx.screens.Screen = mx.screens.Screen.currentFocusedScreen;
trace("Current screen is: " + currentFocus._name);
```

# Screen.getChildScreen()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myScreen.getChildScreen(childIndex)
```

## 参数

*childIndex* 指示要返回的子屏幕索引（从零开始）的一个数字。

## 返回

屏幕对象。

### 说明

方法；返回 *myScreen* 的子屏幕（其索引为 *childIndex*）。

### 示例

以下示例将属于名为 *Presentation* 的根屏幕的所有子屏幕的名称发送到“输出”面板。

```
for (var i:Number = 0; i < _root.Presentation.numChildScreens; i++) {
 var childScreen:mx.screens.Screen =
 _root.Presentation.getChildScreen(i);
 trace(childScreen._name);
}
```

## Screen.indexOfParent

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*myScreen*.indexOfParent

### 说明

属性（只读）；包含 *myScreen* 在其父屏幕的子屏幕列表中的索引（从零开始）。

### 示例

以下示例显示了屏幕 *myScreen* 在其父屏幕的子屏幕列表中的相对位置。

```
var numChildren:Number = myScreen._parent.numChildScreens;
var myIndex:Number = myScreen.indexOfParent;
trace("I child slide # " + myIndex + " out of " + numChildren + "
 screens.");
```

# Screen.mouseDown

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(mouseDown) {
 // 此处是您的代码。
}

listenerObject = new Object();
listenerObject.mouseDown = function(eventObj){
 // 在此处插入您的代码。
}

screenObj.addEventListener("mouseDown", listenerObject)
```

## 说明

事件：当在直接属于屏幕的对象（如形状或影片剪辑）上按下鼠标按钮时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下代码在“输出”面板中显示曾捕获鼠标事件的屏幕的名称。

```
on(mouseDown) {
 trace("Mouse down event on: " + eventObj.target._name);
}
```

# Screen.mouseDownSomewhere

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(mouseDownSomewhere) {
 // 此处是您的代码。
}

listenerObject = new Object();
listenerObject.mouseDownSomewhere = function(eventObject){
 // 在此处插入您的代码。
}

screenObj.addEventListener("mouseDownSomewhere", listenerObject)
```

## 说明

事件：当按下鼠标按钮（但不一定在指定的屏幕上）时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

# Screen.mouseMove

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(mouseMove) {
 // 此处是您的代码。
}

listenerObject = new Object();
listenerObject.mouseMove = function(eventObject){
 // 在此处插入您的代码。
}

screenObj.addEventListener("mouseMove", listenerObject)
```



## 说明

事件：当鼠标在屏幕上移动时进行广播。此事件仅当鼠标移到此屏幕的边框上时才发送。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。



此事件可能会影响系统性能，因此使用时应小心。

# Screen.mouseOut

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(mouseOut) {
 // 此处是您的代码。
}

listenerObject = new Object();
listenerObject.mouseOut = function(eventObject){
 // 在此处插入您的代码。
}

screenObj.addEventListener("mouseOut", listenerObject)
```

## 说明

事件：当鼠标从屏幕的边框内移到边框外时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。



此事件可能会影响系统性能，因此使用时应小心。

# Screen.mouseOver

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(mouseOver) {
 // 此处是您的代码。
}
listenerObject = new Object();
listenerObject.mouseOver = function(eventObject){
 // 在此处插入您的代码。
}
screenObj.addEventListener("mouseOver", listenerObject)
```

## 说明

事件：当鼠标从屏幕的边框内移到边框外时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。



此事件可能会影响系统性能，因此使用时应小心。

# Screen.mouseUp

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(mouseUp) {
 // 此处是您的代码。
}
listenerObject = new Object();
listenerObject.mouseUp = function(eventObject){
 // 在此处插入您的代码。
}
```

```
}
screenObj.addEventListener("mouseUp", listenerObject)
```

### 说明

事件：在屏幕上松开鼠标时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## Screen.mouseUpSomewhere

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
on(mouseUpSomewhere) {
 // 此处是您的代码。
}
listenerObject = new Object();
listenerObject.mouseUpSomewhere = function(eventObject){
 // 在此处插入您的代码。
}
screenObj.addEventListener("mouseUpSomewhere", listenerObject)
```

### 说明

事件：当释放鼠标按钮（但不一定在指定的屏幕上）时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

# Screen.numChildScreens

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myScreen*.numChildScreens

## 说明

属性（只读）：返回 *myScreen* 包含的子屏幕数量。

## 示例

以下示例显示属于 *myScreen* 的所有子屏幕的名称。

```
var howManyKids:Number = myScreen.numChildScreens;
for(i=0; i<howManyKids; i++) {
 var childScreen = myScreen.getChildScreen(i);
 trace(childScreen._name);
}
```

## 另请参见

[Screen.getChildScreen\(\)](#)

# Screen.parentIsScreen

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myScreen*.parentIsScreen

## 说明

属性（只读）：返回一个布尔值，指示所指定屏幕的父对象是 (true) 否 (false) 也是屏幕。如果此属性为 false，则 *myScreen* 位于屏幕层次结构的根位置。

## 示例

以下代码确定屏幕 `myScreen` 的父对象是否也是屏幕。如果 `myScreen.parentIsScreen` 为 `true`，则 `trace()` 语句将在“输出”面板中显示 `myScreen` 的同级幻灯片的数量。如果 `myScreen` 的父屏幕并不是屏幕，则 **Flash** 将假设 `myScreen` 为演示文稿的根（主）幻灯片，因而没有同级幻灯片。

```
if (myScreen.parentIsScreen) {
 trace("I have "+myScreen._parent.numChildScreens+" sibling screens");
} else {
 trace("I am the root screen and have no siblings");
}
```

# Screen.parentScreen

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`myScreen.parentScreen`

## 说明

属性（只读）：返回包含 `myScreen` 的屏幕。如果 `myScreen` 为根屏幕，则返回 `null`。

## 示例

以下示例显示包含屏幕 `myScreen` 的屏幕的名称。

```
var myParent:mx.screens.Screen = myScreen.rootScreen;
```

# Screen.rootScreen

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*myScreen*.rootScreen

## 说明

属性（只读）：返回位于屏幕层次结构顶层的包含 *myScreen* 的屏幕。

## 示例

以下示例显示包含屏幕 *myScreen* 的根屏幕的名称。

```
var myRoot:mx.screens.Screen = myScreen.rootScreen;
```

**ScrollPane** 组件在一个可滚动区域中显示影片剪辑、JPEG 文件和 SWF 文件。通过使用滚动窗格，可以限制这些媒体类型所占用的屏幕区域的大小。滚动窗格可以显示从本地磁盘或 Internet 加载的内容。在创作过程中和运行时，您都可以使用 **ActionScript** 设置此内容。

一旦滚动窗格具有焦点，如果其内容具有有效的制表位，那些标记将接收焦点。在内容中的最后一个制表位之后，焦点将切换到下一个组件。滚动窗格中的垂直和水平滚动条从不接收焦点。

如果用户单击或切换到 **ScrollPane** 实例，该实例将接收焦点。当 **ScrollPane** 实例具有焦点时，您可以使用下列按键来控制它：

键	说明
向下箭头	内容向上移动一垂直滚动行。
End	内容移动到滚动窗格的底部。
向左箭头	内容向右移动一水平滚动行。
Home	内容移动到滚动窗格的顶部。
Page Down	内容向上移动一垂直滚动页。
Page Up	内容向下移动一垂直滚动页。
向右箭头	内容向左移动一水平滚动行。
向上箭头	内容向下移动一垂直滚动行。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

每个 **ScrollPane** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。

# 使用 ScrollPane 组件

如果某些内容对于它们要加载到其中的区域而言过大，您可以使用滚动窗格来显示这些内容。例如，如果您有一幅大图像，而在应用程序中只有很小的空间来显示它，则可以将其加载到滚动窗格中。

您可以通过将 `scrollDrag` 参数设为 `true` 来允许用户在窗格中拖动内容；一个手形光标会出现在内容上。与其它大多数组件不同的是，当按下鼠标按键时，事件开始广播，一直到松开按键才结束。如果滚动窗格的内容具有有效的 **Tab** 键停靠位，您必须将 `scrollDrag` 设为 `false`，否则每次鼠标与内容进行的交互操作都将会调用滚动拖动。

**Loader**、**ScrollPane** 和 **Window** 等组件具有用于确定内容何时完成加载的事件。因此，如果希望设置 **Loader**、**ScrollPane** 或 **Window** 组件内容上的属性，请在 “complete” 事件处理函数中添加属性语句。考虑以下示例：

```
loadtest = new Object();
loadtest.complete = function(eventObject){
 content_mc._width= 100;
}
my_scrollpane.addEventListener("complete", loadtest)
```

有关更多信息，请参见第 1021 页的 “[ScrollPane.content](#)”。

## ScrollPane 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 **ScrollPane** 实例设置以下创作参数：

**contentPath** 指示要加载到滚动窗格中的内容。该值可以是本地 **SWF** 或 **JPEG** 文件的相对路径，或 **Internet** 上的文件的相对或绝对路径。它也可以是设置为“为 **ActionScript** 导出”的库中的影片剪辑元件的链接标识符。

**hLineScrollSize** 指示每次单击箭头按钮时水平滚动条移动多少个单位。默认值为 5。

**hPageScrollSize** 指示每次单击轨道时水平滚动条移动多少个单位。默认值为 20。

**hScrollPolicy** 显示水平滚动条。该值可以是 `on`、`off` 或 `auto`。默认值为 `auto`。

**scrollDrag** 是一个布尔值，它确定当用户在滚动窗格中拖动内容时是 (`true`) 否 (`false`) 发生滚动。默认值为 `false`。

**vLineScrollSize** 指示每次单击滚动箭头时垂直滚动条移动多少个单位。默认值为 5。

**vPageScrollSize** 指示每次单击滚动条轨道时垂直滚动条移动多少个单位。默认值为 20。

**vScrollPolicy** 显示垂直滚动条。该值可以是 `on`、`off` 或 `auto`。默认值为 `auto`。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **ScrollPane** 组件实例设置以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 `true`。



**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。

提醒

**minHeight** 和 **minWidth** 属性由内部的大小调整例程使用。它们在 **UIObject** 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以编写 **ActionScript**，通过利用其属性、方法和事件来控制 **ScrollPane** 组件的这些选项以及其它选项。有关更多信息，请参见第 1015 页的“**ScrollPane 类**”。

## 创建具有 ScrollPane 组件的应用程序

以下过程解释了如何在创作时将 **ScrollPane** 组件添加到应用程序。在本示例中，滚动窗格会从 **contentPath** 属性指定的路径中加载一个图片。

### 创建具有 ScrollPane 组件的应用程序：

1. 将 **ScrollPane** 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，输入实例名称 **my\_sp**。
3. 在主时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
/**
 * 要求：
 * - 库中有 ScrollPane
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(320, 240);

// 为滚动垂直位置创建侦听器对象。
var scrollListener:Object = new Object();
scrollListener.scroll = function(evt_obj:Object) {
 trace("hPosition: " + my_sp.hPosition + ", vPosition = " +
 my_sp.vPosition);
};
// 添加侦听器。
my_sp.addEventListener("scroll", scrollListener);

// 为加载完成创建侦听器对象。
var completeListener:Object = new Object();
completeListener.complete = function(evt_obj:Object) {
 trace(evt_obj.target.contentPath + " has completed loading.");
};
// 添加侦听器。
my_sp.addEventListener("complete", completeListener);
```

```
my_sp.contentPath = "http://www.helpexamples.com/flash/images/
 image1.jpg";
```

上面这些示例创建一个滚动窗格、设置其大小并使用 `contentPath` 属性向其中加载一个图像。它还创建两个侦听器。第一个侦听器在用户垂直或水平滚动时侦听滚动事件并显示图像的位置。第二个侦听器侦听 `complete` 事件，并在“输出”面板中显示说明图像已加载完成的消息。

## 自定义 ScrollPane 组件

在创作过程中和运行时，可以在水平和垂直方向上改变 `ScrollPane` 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参见 `UIObject.setSize()`）或任何适用的 `ScrollPane` 类的属性和方法。

请记住关于 `ScrollPane` 组件的以下要点：

- `ScrollPane` 将其内容的注册点放置在窗格的左上角。
- 当关闭水平滚动条时，垂直滚动条沿滚动窗格右侧从上到下显示。当关闭垂直滚动条时，水平滚动条沿滚动窗格底部从左到右显示。也可以同时关闭两个滚动条。
- 如果滚动窗格太小，则内容可能无法正确显示。
- 调整滚动窗格的大小时，按钮会保持相同的大小。滚动轨道和滚动框（滑块）会扩展或收缩，其点击区也会调整大小。

## 对 ScrollPane 组件使用样式

`ScrollPane` 支持以下样式：

样式	主题	说明
themeColor	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
borderStyle	光晕和范例	<code>ScrollPane</code> 组件使用 <code>RectBorder</code> 实例作为其边框，并对该类定义的样式做出响应。请参见第 985 页的“ <code>RectBorder</code> 类”。  默认边框样式为 "inset"。
scrollTrackColor	范例	滚动轨道的背景色。默认值为 0xCCCCCC（浅灰）。
symbolColor	范例	滚动条按钮上箭头的颜色。默认值为 0x000000（黑色）。
symbolDisabledColor	范例	滚动条按钮上禁用箭头的颜色。默认值为 0x848384（深灰）。

## 对 ScrollPane 组件使用外观

ScrollPane 组件使用一个 `RectBorder` 实例作为其边框，并使用滚动条滚动资源。有关设置这些可视元素的外观的更多信息，请参见第 985 页的“[RectBorder 类](#)”和第 1283 页的“[对 UIScrollBar 组件使用外观](#)”。

## ScrollPane 类

继承 MovieClip > [UIObject 类](#) > [UIComponent 类](#) > View > ScrollView > ScrollPane  
ActionScript 类名称 `mx.containers.ScrollPane`

ScrollPane 类的属性使您可以在运行时执行以下操作：设置内容、监视加载进度和调整滚动量。

使用 ActionScript 设置 ScrollPane 类的属性将会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

您可以设置滚动窗格，以使用户可以在窗格内拖动内容。为此，可将 `scrollDrag` 属性设置为 `true`；一个手形光标会出现在内容上。与其它大多数组件不同的是，当按下鼠标按键时，事件开始广播，一直到松开按键才结束。如果滚动窗格的内容具有有效的 `Tab` 键停靠位，您必须将 `scrollDrag` 设为 `false`，否则每次鼠标与内容进行的交互操作都将会调用滚动拖动。

每个组件类都有一个 `version` 属性，该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指明组件的版本。要访问此属性，请使用以下代码：

```
trace(mx.containers.ScrollPane.version);
```



代码 `trace(myScrollPaneInstance.version);` 返回 `undefined`。

# ScrollPane 类的方法摘要

下表列出了 ScrollPane 类的方法。

方法	说明
<a href="#">ScrollPane.getBytesLoaded()</a>	返回已加载的内容的字节数。
<a href="#">ScrollPane.getBytesTotal()</a>	返回要加载的内容的总字节数。
<a href="#">ScrollPane.refreshPane()</a>	重新加载滚动窗格的内容（但不重绘滚动条）。

## 从 UIObject 类继承的方法

下表列出了 ScrollPane 类从 UIObject 类继承的方法。从 ScrollPane 对象调用这些方法时，请使用 *ScrollPaneInstance.methodName* 的形式。

方法	说明
<a href="#">UIObject.createClassObject()</a>	创建指定类的对象。
<a href="#">UIObject.createObject()</a>	创建对象的子对象。
<a href="#">UIObject.destroyObject()</a>	破坏组件实例。
<a href="#">UIObject.doLater()</a>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<a href="#">UIObject.getStyle()</a>	从样式声明或对象获取样式属性。
<a href="#">UIObject.invalidate()</a>	标记对象使其在到达下一个帧间隔时进行重绘。
<a href="#">UIObject.move()</a>	将对象移动到要求的位置。
<a href="#">UIObject.redraw()</a>	迫使对象有效以便能在当前帧中绘制。
<a href="#">UIObject.setSize()</a>	将对象调整为所要求的大小。
<a href="#">UIObject.setSkin()</a>	设置对象的外观。
<a href="#">UIObject.setStyle()</a>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 ScrollPane 类从 UIComponent 类继承的方法。从 ScrollPane 对象调用这些方法时，请使用 *ScrollPaneInstance.methodName* 的形式。

方法	说明
<a href="#">UIComponent.getFocus()</a>	返回对具有焦点的对象的引用。
<a href="#">UIComponent.setFocus()</a>	将焦点设置到组件实例中。

# ScrollPane 类的属性摘要

下表列出了 ScrollPane 类的属性。

方法	说明
<a href="#">ScrollPane.content</a>	对加载到滚动窗格中的内容的引用（只读）。
<a href="#">ScrollPane.contentPath</a>	一个字符串，它指示加载到滚动窗格中的 SWF 或 JPEG 文件的绝对 URL 或相对 URL，或者是当前文档的库面板中影片剪辑的链接标识符。
<a href="#">ScrollPane.hLineScrollSize</a>	单击滚动箭头时要水平滚动的内容量。
<a href="#">ScrollPane.hPageScrollSize</a>	单击滚动轨道时要水平滚动的内容量。
<a href="#">ScrollPane.hPosition</a>	滚动窗格的水平滚动条的水平像素位置。
<a href="#">ScrollPane.hScrollPolicy</a>	水平滚动条的状态。它可以为始终打开 ("on")、始终关闭 ("off")，或者只是在需要时才打开 ("auto")。默认值为 "auto"。
<a href="#">ScrollPane.scrollDrag</a>	指示当用户在滚动窗格中拖动内容时是 (true) 否 (false) 发生滚动。默认值为 false。
<a href="#">ScrollPane.vLineScrollSize</a>	单击滚动箭头时要垂直滚动的内容量。
<a href="#">ScrollPane.vPageScrollSize</a>	单击滚动轨道时要垂直滚动的内容量。
<a href="#">ScrollPane.vPosition</a>	滚动窗格的垂直滚动条的像素位置。
<a href="#">ScrollPane.vScrollPolicy</a>	垂直滚动条的状态。它可以为始终打开 ("on")、始终关闭 ("off")，或者只是在需要时才打开 ("auto")。默认值为 "auto"。

## 从 UIObject 类继承的属性

下表列出了 ScrollPane 类从 UIObject 类继承的属性。从 ScrollPane 对象访问这些属性时，请使用 *ScrollPaneInstance.propertyName* 的形式。

属性	说明
<a href="#">UIObject.bottom</a>	只读；对象的底边缘位置（相对于其父对象的底边缘）。
<a href="#">UIObject.height</a>	只读；对象的高度，以像素为单位。
<a href="#">UIObject.left</a>	只读；对象的左边缘（以像素为单位）。
<a href="#">UIObject.right</a>	只读；对象的右边缘位置（相对于其父对象的右边缘）。
<a href="#">UIObject.scaleX</a>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<a href="#">UIObject.scaleY</a>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<a href="#">UIObject.top</a>	只读；对象上边缘的位置（相对于其父对象）。
<a href="#">UIObject.visible</a>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。

属性	说明
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

## 从 `UIComponent` 类继承的属性

下表列出了 `ScrollPane` 类从 `UIComponent` 类继承的属性。从 `ScrollPane` 对象访问这些属性时，请使用 `ScrollPaneInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## ScrollPane 类的事件摘要

下表列出了 `ScrollPane` 类的事件。

事件	说明
<code>ScrollPane.complete</code>	加载了滚动窗格内容时广播。
<code>ScrollPane.progress</code>	在加载滚动窗格内容时广播。
<code>ScrollPane.scroll</code>	在单击滚动条时广播。

## 从 `UIObject` 类继承的事件

下表列出了 `ScrollPane` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 `ScrollPane` 类从 `UIComponent` 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## ScrollPane.complete

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.complete = function(eventObject:Object) {
 // ...
};
scrollPaneInstance.addEventListener("complete", listenerObject);
```

用法 2:

```
on (complete) {
 //...
}
```

### 说明

事件：当加载完内容时向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*ScrollPaneInstance*) 调度一个事件 (在本例中为 *complete*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作“处理函数”) 处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **ScrollPane** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **ScrollPane** 组件实例 `myScrollPaneComponent`，它将

“\_level0.myScrollPaneComponent” 发送到“输出”面板：

```
on (complete) {
 trace(this);
}
```

### 示例

以下示例为 `ScrollPane` 实例创建一个具有 `complete` 事件处理函数的侦听器对象。当滚动窗格的内容加载完成时，侦听器会在“输出”面板中显示一条消息。

首先将 **ScrollPane** 组件从“组件”面板中拖到库中，然后将以下代码添加到第一帧中：

```
/**
 要求：
 - 库中有 ScrollPane
*/

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(320, 240);

// 为加载完成创建侦听器对象。
var completeListener:Object = new Object();
completeListener.complete = function(evt_obj:Object) {
 trace(evt_obj.target.contentPath + " has completed loading.");
};
// 添加侦听器。
my_sp.addEventListener("complete", completeListener);

my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```



# ScrollPane.content

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ScrollPaneInstance.content*

## 说明

只读属性；对滚动窗格内容的引用。在加载开始前该值为 `undefined`。

## 示例

本示例设置 `contentPath` 属性在滚动窗格中加载图片（或从专业角度来说，加载包含 JPEG 图像的影片剪辑）。它还创建一个数字步进器，用户可以以每 10 个增量单位进行增加或减少，最大值为 100。当用户更改 `NumericStepper` 中的值时，侦听器会将图像的透明度 (`content._alpha`) 设置为指定的百分比。注意，`_alpha` 是一个 `MovieClip` 属性。

首先将 `ScrollPane` 和 `NumericStepper` 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane
 * - 库中有 NumericStepper
 */

System.security.allowDomain("http://www.helpexamples.com");

this.createClassObject(mx.controls.NumericStepper, "my_nstep", 10,
 {minimum:10, maximum:100, stepSize:10});
my_nstep.value = my_nstep.maximum;

this.createClassObject(mx.containers.ScrollPane, "my_sp", 20);
my_sp.move(0, 30);
my_sp.setSize(180, 160);
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image2.jpg";

var nstepListener:Object = new Object();
nstepListener.change = function(evt_obj:Object) {
 my_sp.content._alpha = my_nstep.value;
}
my_nstep.addEventListener("change", nstepListener);
```

另请参见

[ScrollPane.contentPath](#)

# ScrollPane.contentPath

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ScrollPaneInstance*.contentPath

## 说明

属性：一个字符串，它指明加载到滚动窗格中的 SWF 或 JPEG 文件的绝对 URL 或相对 URL。相对路径必须相对于加载内容的 SWF 文件。

如果使用相对 URL 加载内容，加载的内容必须相对于包含滚动窗格的 SWF 文件的位置。例如，使用位于 `/scrollpane/nav/example.swf` 目录中的 ScrollPane 组件的应用程序可通过使用以下 contentPath 属性从目录 `/scrollpane/content/flash/logo.swf` 加载内容：`"../content/flash/logo.swf"`

## 示例

以下示例显示如何设置 contentPath 属性以便从三个不同的源中加载 ScrollPane：1) Internet 上的图像；2) 库中的影片剪辑；3) 当前工作目录中的 SWF 文件。一次只能使用一个源。

首先将 ScrollPane 组件从“组件”面板拖到当前文档的库中。要试用第 2 种选择，必须在库中创建一个影片剪辑并引用其名称。若要试用第 3 种选择，应在当前工作目录中创建一个 SWF 文件并指定其名称。然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 舞台上有一个 ScrollPane（实例名称：my_sp）
 * - 库中有一个链接 ID 为 "movieClip_Name" 的元件 ** 可选
 * - 工作目录中有一个 logo.swf 文件 ** 可选
 */

System.security.allowDomain("http://www.helpexamples.com");

var my_sp:mx.containers.ScrollPane;
```

```
// 方法 1: JPEG 图像
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

```
// 方法 2: 库中的元件
my_sp.contentPath = "movieClip_Name";
```

```
// 方法 3: SWF 文件
my_sp.contentPath = "logo.swf";
```

另请参见

[ScrollPane.content](#)

## ScrollPane.getBytesLoaded()

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

```
scrollPaneInstance.getBytesLoaded()
```

参数

无。

返回

滚动窗格中已加载的字节数。

说明

方法：返回 **ScrollPane** 实例中已加载的字节数。加载内容时可按固定时间间隔调用该方法以查看其进度。

## 示例

本示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，并定义一个具有 `progress` 事件处理函数的名为 `loadListener` 的侦听器对象。该事件处理函数调用 `getBytesLoaded()` 和 `getBytesTotal()` 函数，以在“输出”面板中显示加载的进度。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

var loadListener:Object = new Object();
loadListener.progress = function(evt_obj:Object) {
 trace(my_sp.getBytesLoaded() + " of " + my_sp.getBytesTotal() + " bytes
 loaded.");
};
my_sp.addEventListener("progress", loadListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

# ScrollPane.getBytesTotal()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
scrollPaneInstance.getBytesTotal()
```

## 参数

无。

## 返回

一个数字。

## 说明

方法：返回要加载到 **ScrollPane** 实例中的总字节数。

## 示例

本示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，并定义一个具有 `progress` 事件处理函数的名为 `loadListener` 的侦听器对象。该事件处理函数调用 `getBytesLoaded()` 和 `getBytesTotal()` 函数以在“输出”面板中显示加载的进度。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

var loadListener:Object = new Object();
loadListener.progress = function(evt_obj:Object) {
 trace(my_sp.getBytesLoaded() + " of " + my_sp.getBytesTotal() + " bytes
 loaded.");
};
my_sp.addEventListener("progress", loadListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

## 另请参见

[ScrollPane.getBytesLoaded\(\)](#)

# ScrollPane.hLineScrollSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*scrollPaneInstance.hLineScrollSize*

## 说明

属性；水平滚动条中的箭头被单击时内容要移动的像素数。默认值为 5。

## 示例

本示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，加载一个图像，并将 `hLineScrollSize` 属性设置为用户单击水平滚动条上的箭头时滚动 **100** 像素。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// 单击水平滚动条上的箭头时滚动 100 像素。
my_sp.hLineScrollSize = 100;
```

# ScrollPane.hPageScrollSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`scrollPaneInstance.hPageScrollSize`

## 说明

属性；水平滚动条中的轨道被单击时内容要移动的像素数。默认值为 **20**。

## 示例

本示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，加载一个图像，并将 `hPageScrollSize` 属性设置为用户单击水平滚动条中的轨道时滚动 **100** 像素。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */
```

```
this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// 单击水平滚动条时滚动 100 像素。
my_sp.hPageScrollSize = 100;
```

## ScrollPane.hPosition

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*ScrollPaneInstance.hPosition*

### 说明

属性；以像素为单位调整滚动窗格的内容，并按比例调整水平滚动框（缩略图）。0 位置位于滚动轨道的最左端，当滑块处于此位置时滚动窗格最左边的内容在滚动窗格内可见。

### 示例

本示例创建一个名为 my\_sp 的 **ScrollPane** 实例，加载一个图像，创建一个侦听器来处理滚动事件，并在用户单击滚动条时显示水平 (hPosition) 和垂直 (vPosition) 滚动位置。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// 单击水平滚动条时滚动 100 像素。
my_sp.hPageScrollSize = 100;
```

```
// 创建侦听器对象。
var spListener:Object = new Object();
spListener.scroll = function(evt_obj:Object) {
 trace("hPosition = " + my_sp.hPosition + ", vPosition = " +
 my_sp.vPosition);
}
// 添加侦听器。
my_sp.addEventListener("scroll", spListener);
```

## ScrollPane.hScrollPolicy

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*ScrollPaneInstance.hScrollPolicy*

### 说明

属性；确定水平滚动条是始终显示 ("on")、从不显示 ("off")，还是根据图像大小自动显示 ("auto")。默认值为 "auto"。

### 示例

以下示例创建一个名为 my\_sp 的 **ScrollPane** 实例，将 hScrollPolicy 设置为 off 以阻止显示水平滚动条，并为其加载一个图像。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);
my_sp.hScrollPolicy = "off";

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```



# ScrollPane.progress

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.progress = function(eventObject:Object) {
 // ...
};
scrollPaneInstance.addEventListener("progress", listenerObject);
```

用法 2:

```
on (progress) {
 // ...
}
```

## 说明

事件：在加载内容时向所有已注册的侦听器广播。**progress** 事件并不会始终广播；complete 事件可能在未调度任何 progress 事件的情况下广播。如果加载的内容是本地文件，尤其会出现这种情况。通过设置 contentPath 属性的值可使应用程序在内容开始加载时触发 progress 事件。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*ScrollPaneInstance*) 调度一个事件（在本例中为 progress），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **ScrollPane** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **ScrollPane** 组件实例 `mySPComponent`，它将 “\_level0.mySPComponent” 发送到 “输出” 面板：

```
on (progress) {
 trace(this);
}
```

## 示例

本示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，并定义一个具有 `progress` 事件处理函数的名为 `spListener` 的侦听器对象。该事件处理函数调用 `getBytesLoaded()` 和 `getBytesTotal()` 函数，以在 “输出” 面板中显示加载的进度。

首先将 **ScrollPane** 组件从 “组件” 面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 要求:
 - 库中有 ScrollPane 组件
*/

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

var spListener:Object = new Object();
spListener.progress = function(evt_obj:Object):Void {
 trace("Loading " + my_sp.contentPath);
 trace(my_sp.getBytesLoaded() + " of " + my_sp.getBytesTotal() + " bytes
 loaded");
};
my_sp.addEventListener("progress", spListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

# ScrollPane.refreshPane()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
scrollPaneInstance.refreshPane()
```

## 参数

无。

## 返回

无。

## 说明

方法：加载内容以后刷新滚动窗格。此方法重新加载内容，但不重置滚动条。例如，如果您已经将一个表单加载到滚动窗格中，并且已经使用 **ActionScript** 更改了一个输入属性（例如一个文本字段），那么，您就可以使用该方法。在本例中，将调用 `refreshPane()` 来重新加载具有这些输入属性的新值的同一个表单。

## 示例

本示例创建一个“刷新”按钮和一个名为 `my_sp` 的 **ScrollPane** 实例。它向 **ScrollPane** 中加载一个图像，并为按钮上的单击事件创建一个侦听器。发生单击事件时，该示例会调用 `refreshPane()` 函数，此函数会重新加载滚动窗格的内容。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.controls.Button, "my_button", 10,
 {label:"Refresh"});
this.createClassObject(mx.containers.ScrollPane, "my_sp", 20);
my_sp.move(0, 30);
my_sp.setSize(360, 280);

var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 my_sp.refreshPane();
}
my_button.addEventListener("click", buttonListener);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

# ScrollPane.scroll

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object):Void {
 // ...
};
scrollPaneInstance.addEventListener("scroll", listenerObject);
```

用法 2:

```
on (scroll) {
 // ...
}
```

## 事件对象

除了标准的事件对象属性之外，还为 scroll 事件定义了另外两个属性：一个 type 属性，其值为 "scroll"，一个 direction 属性，其值可以为 "vertical" 或 "horizontal"。

除了标准的事件对象属性外，还为 ProgressBar.progress 事件定义了另外两个属性：current（加载的值等于 total）和 total（总值）。

## 说明

事件：当用户按下滚动条按钮、滚动框（滑块）或滚动轨道时，向所有已注册的侦听器广播。与其它事件不同的是，当用户在滚动条上按下鼠标按钮时，scroll 事件开始持续广播，直到松开鼠标按钮。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*scrollPaneInstance*) 调度一个事件（在本例中为 scroll），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **ScrollPane** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到实例 `my_sp`，它将 “\_level0.my\_sp” 发送到 “输出” 面板：

```
on (scroll) {
 trace(this);
}
```

## 示例

本示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，加载一个图像，并为滚动事件创建一个侦听器。滚动事件发生时，本示例会在 “输出” 面板中显示水平 (`hPosition`) 和垂直 (`vPosition`) 滚动位置。

首先将 **ScrollPane** 组件从 “组件” 面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 要求：
 - 库中有 ScrollPane 组件
*/

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// 单击水平滚动条时滚动 100 像素。
my_sp.hPageScrollSize = 100;

// 创建侦听器对象。
var spListener:Object = new Object();
spListener.scroll = function(evt_obj:Object):Void {
 trace("hPosition = " + my_sp.hPosition + ", vPosition = " +
 my_sp.vPosition);
};
// 添加侦听器。
my_sp.addEventListener("scroll", spListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# ScrollPane.scrollDrag

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ScrollPaneInstance.scrollDrag*

## 说明

属性；一个布尔值，它指示当用户在滚动窗格内拖动时是 (true) 否 (false) 发生滚动。默认值为 false。

## 示例

本示例创建一个名为 my\_sp 的 **ScrollPane** 实例，加载一个图像，并将 scrollDrag 属性设置为 true，以便允许用户通过拖动滚动窗格内的图像进行滚动。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// 启用通过拖动滚动窗格进行滚动。
my_sp.scrollDrag = true;
```

# ScrollPane.vLineScrollSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*scrollPaneInstance.vLineScrollSize*

## 说明

属性：表示当用户单击垂直滚动条中的滚动箭头时，显示区域中的内容将移动的像素数。默认值为 5。

## 示例

以下示例创建一个名为 `my_sp` 的 **ScrollPane** 实例，加载一个图像，并将 `vLineScrollSize` 属性设置为当用户单击垂直滚动条上的箭头时滚动 20 像素。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档面板中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// 单击垂直滚动条上的箭头时滚动 20 像素。
my_sp.vLineScrollSize = 20;
```

# ScrollPane.vPageScrollSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*scrollPaneInstance.vPageScrollSize*

## 说明

属性；表示当用户单击垂直滚动条中的轨道时，显示区域中的内容将移动的像素数。默认值为 20。

## 示例

本示例创建一个名为 my\_sp 的 **ScrollPane** 实例，加载一个图像，并将 vPageScrollSize 属性设置为用户单击垂直滚动条中的轨道时滚动 30 像素。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";

// 单击垂直滚动条时滚动 30 像素。
my_sp.vPageScrollSize = 30;
```



# ScrollPane.vPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ScrollPaneInstance.vPosition*

## 说明

属性：以像素为单位调整滚动窗格的内容，并按比例调整垂直滚动框（缩略图）。0 位置位于滚动轨道的最左端，当滑块处于此位置时滚动窗格最左边的内容在滚动窗格内可见。默认值为 0。

## 示例

本示例创建一个名为 my\_sp 的 **ScrollPane** 实例，加载一个图像，创建一个侦听器来处理滚动事件，并在用户单击滚动条时显示水平 (hPosition) 和垂直 (vPosition) 滚动位置。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */

this.createClassObject(mx.containers.ScrollPane, "my_sp", 10);
my_sp.setSize(360, 280);

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
// 单击水平滚动条时滚动 100 像素。
my_sp.hPageScrollSize = 100;

// 创建侦听器对象。
var spListener:Object = new Object();
spListener.scroll = function(evt_obj:Object):Void {
 trace("hPosition = " + my_sp.hPosition + ", vPosition = " +
 my_sp.vPosition);
};
// 添加侦听器。
my_sp.addEventListener("scroll", spListener);
```

# ScrollPane.vScrollPolicy

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*ScrollPaneInstance.vScrollPolicy*

## 说明

属性；确定垂直滚动条是始终显示 ("on")、从不显示 ("off")，还是根据图像大小自动显示 ("auto")。默认值为 "auto"。

## 示例

以下示例创建一个名为 my\_sp 的 **ScrollPane** 实例，将 vScrollPolicy 设置为 off 以阻止显示垂直滚动条，并为 **ScrollPane** 加载一个图像。

首先将 **ScrollPane** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 ScrollPane 组件
 */
import mx.containers.ScrollPane;

this.createClassObject(ScrollPane, "my_sp", 30);
my_sp.setSize(360, 280);
my_sp.vScrollPolicy = "off";

System.security.allowDomain("http://www.helpexamples.com");
my_sp.contentPath = "http://www.helpexamples.com/flash/images/image1.jpg";
```

# SimpleButton 类

继承 MovieClip > **UIObject 类** > **UIComponent 类** > SimpleButton

ActionScript 类名称 mx.controls.SimpleButton

使用 SimpleButton 类的属性可以在运行时控制以下操作：

- 按钮是否具有默认普通按钮的强调外观
- 按钮是用作普通按钮还是切换开关
- 按钮是否被选中

## SimpleButton 类的方法摘要

没有 SimpleButton 类专用的方法。

### 从 UIObject 类继承的方法

下表列出了 SimpleButton 类从 UIObject 类继承的方法。从 SimpleButton 类调用这些方法时，请使用 *buttonInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。

方法	说明
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 `SimpleButton` 类从 `UIComponent` 类继承的方法。从 `SimpleButton` 对象调用这些方法时，请使用 `buttonInstance.methodName` 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## SimpleButton 类的属性摘要

下表列出了 `SimpleButton` 类的属性。

属性	说明
<code>SimpleButton.emphasized</code>	指示按钮是否具有默认普通按钮的外观。
<code>SimpleButton.emphasizedStyleDeclaration</code>	当 <code>emphasized</code> 属性设置为 <code>true</code> 时的样式声明。
<code>SimpleButton.selected</code>	一个布尔值，它指示按钮是 ( <code>true</code> ) 否 ( <code>false</code> ) 处于选中状态。默认值为 <code>false</code> 。
<code>SimpleButton.toggle</code>	一个布尔值，它指示按钮的行为与切换开关相同 ( <code>true</code> ) 还是不同 ( <code>false</code> )。默认值为 <code>false</code> 。

## 从 UIObject 类继承的属性

下表列出了 `SimpleButton` 类从 `UIObject` 类继承的属性。从 `SimpleButton` 对象访问这些属性时，请使用 `buttonInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。

属性	说明
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## 从 `UIComponent` 类继承的属性

下表列出了 `SimpleButton` 类从 `UIComponent` 类继承的属性。从 `SimpleButton` 对象访问这些属性时，请使用 `buttonInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## SimpleButton 类的事件摘要

下表列出了 `SimpleButton` 类的事件。

事件	说明
<code>SimpleButton.click</code>	单击一个按钮时广播。

## 从 `UIObject` 类继承的事件

下表列出了 `SimpleButton` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 SimpleButton 类从 UIComponent 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## SimpleButton.click

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObj:Object){
 // ...
};
buttonInstance.addEventListener("click", listenerObject);
```

用法 2:

```
on (click) {
 // ...
}
```

## 说明

事件；当在按钮上单击（释放）鼠标，或者当按钮具有焦点并按下空格键时，对所有已注册的侦听器进行广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*buttonInstance*) 调度一个事件（在本例中为 `click`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件发生时，就会调用该方法。该事件发生时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()`（请参见 [EventDispatcher.addEventListener\(\)](#)），以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用一个 `on()` 处理函数，并且必须直接附加到一个 **Button** 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **Button** 组件实例 `myButtonComponent`，它将“\_level0.myButtonComponent”发送到“输出”面板：

```
on (click) {
 trace(this);
}
```

当在附加到常规 **Flash** 按钮元件的 `on()` 处理函数中使用 `this` 的行为是不同的；在此示例中，`this` 指的是包含按钮的对象。例如，以下代码附加到按钮元件实例 `myButton`，它将“\_level0”发送到“输出”面板：

```
on (release) {
 trace(this);
}
```



内置的 `ActionScript Button` 对象没有 `click` 事件；最接近的事件是 `release`。

## 示例

此示例是在时间轴上的某一帧上编写的，当单击名为 `buttonInstance` 的按钮时，这段代码会向“输出”面板发送一条消息。第一行代码指定该按钮用作切换开关。第二行代码创建一个名为 `form` 的侦听器对象。第三行代码为侦听器对象上的 `click` 事件定义一个函数。该函数内部有一条 `trace()` 语句，此语句使用自动传递到该函数的事件对象（在本例中是 `eventObj`）来生成消息。事件对象的 `target` 属性是生成了该事件的组件（在本例中是 `buttonInstance`）。从事件对象的 `target` 属性中可以访问 `SimpleButton.selected` 属性。最后一行代码从 `buttonInstance` 调用 `addEventListener()` 方法，并将 `click` 事件和 `form` 侦听器对象作为参数传递给该方法。

```
buttonInstance.toggle = true;
var form:Object = new Object();
```

```
form.click = function(eventObj:Object) {
 trace("The selected property has changed to " + eventObj.target.selected);
};
buttonInstance.addEventListener("click", form);
```

以下代码还会在单击 `buttonInstance` 时向“输出”面板发送一条消息。`on()` 处理函数必须直接附加到 `buttonInstance`。

```
on (click) {
 trace("button component was clicked");
}
```

## SimpleButton.emphasized

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

`buttonInstance.emphasized`

### 说明

属性；指示按钮是 (`true`) 否 (`false`) 处于强调状态。强调状态相当于默认普通按钮外观。一般来说，应使用 `FocusManager.defaultPushButton` 属性，而不是直接设置 `emphasized` 属性。默认值为 `false`。

如果您没有使用 `FocusManager.defaultPushButton`，则您可能只想将按钮设置为强调状态，或者使用强调状态来更改文本颜色。以下示例设置按钮实例 `myButton` 的 `emphasized` 属性：

```
_global.styles.foo = new CSSStyleDeclaration();
_global.styles.foo.color = 0xFF0000;
SimpleButton.emphasizedStyleDeclaration = "neutralStyle";
myButton.emphasized = true;
```

### 另请参见

[SimpleButton.emphasizedStyleDeclaration](#)



# SimpleButton.emphasizedStyleDeclaration

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*buttonInstance*.emphasizedStyleDeclaration

## 说明

属性（静态）：一个指示样式声明的字符串，该样式声明会在 `emphasized` 属性设置为 `true` 时格式化按钮。

`emphasizedStyleDeclaration` 属性是 **SimpleButton** 类的静态属性。因此，必须直接从 **SimpleButton** 而不是从 *buttonInstance* 访问它，如下所示：

```
SimpleButton.emphasizedStyleDeclaration = "3dEmphStyle";
```

## 另请参见

[SimpleButton.emphasized](#)

# SimpleButton.selected

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*buttonInstance*.selected

## 说明

属性；一个布尔值，它指示按钮是 (`true`) 否 (`false`) 已选中。默认值为 `false`。

# SimpleButton.toggle

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*buttonInstance.toggle*

## 说明

属性；一个布尔值，它指示按钮的行为与切换开关相同 (true) 还是不同 (false)。默认值为 false。

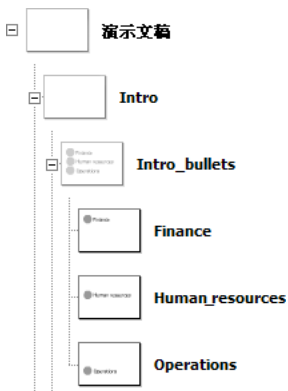
如果按钮的行为与切换开关相同，则它将保持按下状态，直到再次按下释放它为止。

# Slide 类（仅限 Flash Professional）

**Slide** 类对应于分层的幻灯片演示文稿中的节点。在 **Flash Professional 8** 中，可以使用“屏幕大纲”窗格创建幻灯片演示文稿。有关使用屏幕的概述，请参见《使用 **Flash**》中的第 14 章“使用屏幕（仅限 **Flash Professional**）”。

**Slide** 类扩展了 **Screen** 类（请参见第 991 页的“**Screen** 类（仅限 **Flash Professional**）”），并提供在幻灯片之间导航和排序的内置功能，它还能够使用行为轻松地在幻灯片之间附加过渡。幻灯片含有“状态”，因此当用户进入到相邻的幻灯片时，上一张幻灯片会隐藏。

请注意，您只可以导航到（或“停止于”）不包含任何子幻灯片的幻灯片（也称为“叶幻灯片”）。例如，以下图示显示幻灯片演示文稿示例的“屏幕大纲”窗格的内容：



当此演示文稿开始后，默认情况下，将在名为 **Finance** 的幻灯片（即演示文稿中第一张不包含任何子幻灯片的幻灯片）处“停止”。

还应注意，子幻灯片会“继承”其父幻灯片的可视外观（图形和其它内容）。例如，在上面的图示中，除了 **Finance** 幻灯片上的内容，用户还会看到 **Intro** 和“演示文稿”幻灯片上的任何内容。

提醒

**Slide** 类继承于 **Loader** 类，它可让您轻松地将外部 **SWF** 或 **JPEG** 文件加载到给定的幻灯片中。这就提供了一种模块化幻灯片演示文稿并降低初始下载时间的方法。有关更多信息，请参见第 992 页的“将外部内容加载到屏幕（仅限 **Flash Professional**）”。

# 使用 Slide 类（仅限 Flash Professional）

使用 **Slide** 类的方法和属性来控制您使用 **Flash** 幻灯片演示文稿的“屏幕大纲”窗格创建的幻灯片演示文稿，以便获取有关幻灯片演示文稿的信息（例如，确定父幻灯片所包含的子幻灯片的数量），或者在幻灯片演示文稿中的幻灯片之间导航（例如，创建“下一张幻灯片”和“上一张幻灯片”按钮）。

也可以使用“行为”面板中提供的内置行为来控制幻灯片演示文稿。有关更多信息，请参见《使用 Flash》中的“使用行为将控件添加到屏幕（仅限于 Flash Professional）”。

## 幻灯片参数

您可以在“属性”检查器或“组件”检查器中为每张幻灯片设置以下创作参数：

**autoKeyNav** 确定幻灯片如何或是否响应默认的键盘导航。有关更多信息，请参见 [Slide.autoKeyNav](#)。

**autoload** 指示 **contentPath** 参数所指定的内容是应该自动加载 (**true**)，还是应该等到调用 [Loader.load\(\)](#) 方法时再进行加载 (**false**)。默认值为 **true**。

**contentPath** 指定幻灯片的内容。该参数可以是一个影片剪辑的链接标识符，也可以是要加载到幻灯片中的 **SWF** 或 **JPEG** 文件的绝对或相对 **URL**。默认情况下，加载的内容会进行剪辑以适合幻灯片的大小。

**overlayChildren** 指定在从一张子幻灯片导航到下一张子幻灯片时，幻灯片的子幻灯片是保持可见 (**true**) 还是不可见 (**false**)。

**playHidden** 指定幻灯片在隐藏时是继续播放 (**true**) 还是停止播放 (**false**)。

## 使用 Slide 类创建幻灯片演示文稿

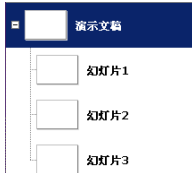
使用 **Slide** 类的方法和属性来控制您在 **Flash** 创作环境的 **Flash** 幻灯片演示文稿的“屏幕大纲”窗格中创建的幻灯片演示文稿。（“行为”面板也包含一些用于创建幻灯片导航的行为。）在本例中，您将编写自己的 **ActionScript**，以便为幻灯片演示文稿创建“下一张”和“上一张”按钮。

**要创建带有导航的幻灯片演示文稿：**

1. 在 **Flash** 中，选择“文件”>“新建”。
2. 在“常规”选项卡上选择“Flash 幻灯片演示文稿”。

3. 在“屏幕大纲”窗格中，单击“插入屏幕” (+) 按钮两次，以在“演示文稿”幻灯片下面创建两张新幻灯片。

“屏幕大纲”窗格应如下所示：



4. 在“屏幕大纲”窗格中选择幻灯片 1，并使用“文本”工具添加一个文本字段，内容为这是第一张幻灯片。
5. 对幻灯片2和幻灯片3重复上述步骤，分别在每张幻灯片上创建内容为这是第二张幻灯片和这是第三张幻灯片的文本字段。
6. 选择“演示文稿”幻灯片并打开“组件”面板。
7. 将 Button 组件从“组件”面板拖到舞台底部。
8. 在“属性”检查器中，为 Button 组件的 Label 属性输入下一张幻灯片。
9. 在“动作”面板中键入以下代码：

```
on(click) {
 _parent.currentSlide.gotoNextSlide();
}
```
10. 测试 SWF 文件（“控制” > “测试影片”）并单击“下一张幻灯片”按钮进入到下一张幻灯片。

## Slide 类 (API)（仅限 Flash Professional）

继承 [MovieClip](#) > [UIObject](#) 类 > [UIComponent](#) 类 > View > [Loader](#) 组件 > [Screen](#) 类（仅限 [Flash Professional](#)）> Slide

ActionScript 类名称 `mx.screens.Slide`

Slide 类的方法、属性和事件使您可以管理和处理幻灯片。

# Slide 类的方法摘要

下表列出了 Slide 类的方法：

方法	说明
<code>Slide.getChildSlide()</code>	返回指定的子幻灯片。
<code>Slide.gotoFirstSlide()</code>	导航到幻灯片的子幻灯片层次结构中的第一个叶节点。
<code>Slide.gotoLastSlide()</code>	导航到幻灯片的子幻灯片层次结构中的最后一个叶节点。
<code>Slide.gotoNextSlide()</code>	导航到下一张幻灯片。
<code>Slide.gotoPreviousSlide()</code>	导航到上一张幻灯片。
<code>Slide.gotoSlide()</code>	导航到一张指定的幻灯片。

## 从 UIObject 类继承的方法

下表列出了 Slide 类从 UIObject 类继承的方法。从 Slide 对象调用这些方法时，请使用 *SlideInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 Slide 类从 UIComponent 类继承的方法。从 Slide 对象调用这些方法时，请使用 *SlideInstance.methodName* 的形式。

方法	说明
<a href="#">UIComponent.setFocus()</a>	返回对具有焦点的对象的引用。
<a href="#">UIComponent.setFocus()</a>	将焦点设置到组件实例中。

## 从 Loader 类继承的方法

下表列出了 Slide 类从 Loader 类继承的方法。从 Slide 对象调用此方法时，请使用 *SlideInstance.methodName* 的形式。

方法	说明
<a href="#">Loader.load()</a>	加载由 <code>contentPath</code> 属性指定的内容。

## 从 Screen 类继承的方法

下表列出了 Slide 类从 Screen 类继承的方法。从 Slide 对象调用此方法时，请使用 *SlideInstance.methodName* 的形式。

方法	说明
<a href="#">Screen.getChildScreen()</a>	返回此屏幕位于特定索引处的子屏幕。

## Slide 类的属性摘要

下表列出了 Slide 类的属性：

属性	说明
<a href="#">Slide.autoKeyNav</a>	确定幻灯片是否使用默认的键盘处理来导航到下一张 / 上一张幻灯片。
<a href="#">Slide.currentChildSlide</a>	只读；返回包含当前活动幻灯片的幻灯片的直接子幻灯片。
<a href="#">Slide.currentFocusedSlide</a>	只读；返回包含全局当前焦点的“最内叶”幻灯片（离幻灯片树的根最远的幻灯片）。
<a href="#">Slide.currentSlide</a>	只读；返回当前活动的幻灯片。
<a href="#">Slide.defaultKeydownHandler</a>	一种回调函数，它覆盖默认的键盘导航（向左和向右箭头键）。
<a href="#">Slide.firstSlide</a>	只读；返回幻灯片的第一张没有子幻灯片的子幻灯片。

属性	说明
<code>Slide.indexInParentSlide</code>	只读；返回幻灯片在其父幻灯片的子幻灯片列表中的索引（从零开始）。
<code>Slide.lastSlide</code>	只读；返回幻灯片的最后一张没有子幻灯片的子幻灯片。
<code>Slide.nextSlide</code>	只读；返回在调用了 <code>mySlide.gotoNextSlide()</code> 时应到达但实际不导航到的幻灯片。
<code>Slide.numChildSlides</code>	只读；返回幻灯片所包含的子幻灯片的数量。
<code>Slide.overlayChildren</code>	确定当控制从一张子幻灯片转到下一张子幻灯片时，幻灯片的子幻灯片是否可见。
<code>Slide.parentIsSlide</code>	只读；返回一个布尔值，它指示幻灯片的父对象也是 (true) 幻灯片或不是 (false) 幻灯片。
<code>Slide.parentSlide</code>	只读；包含当前幻灯片的幻灯片。对于根幻灯片，它可能为 null。
<code>Slide.playHidden</code>	确定当幻灯片隐藏时是否继续播放。
<code>Slide.previousSlide</code>	只读；返回在调用了 <code>mySlide.gotoPreviousSlide()</code> 时应到达但实际不导航到的幻灯片。
<code>Slide.rootSlide</code>	只读；返回包含此幻灯片的幻灯片树的根。

## 从 UIObject 类继承的属性

下表列出了 `Slide` 类从 `UIObject` 类继承的属性。从 `Slide` 对象访问这些属性时，请使用 `SlideInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。



## 从 UIComponent 类继承的属性

下表列出了 Slide 类从 UIComponent 类继承的属性。从 Slide 对象访问这些属性时，请使用 *SlideInstance.propertyName* 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## 从 Loader 类继承的属性

下表列出了 Slide 类从 Loader 类继承的属性。从 Slide 对象访问这些属性时，请使用 *SlideInstance.propertyName* 的形式。

属性	说明
<code>Loader.autoLoad</code>	一个布尔值，它指示内容是自动加载 (true) 还是只在您调用 <code>Loader.load()</code> 时才进行加载 (false)。
<code>Loader.bytesLoaded</code>	只读属性，指明已经加载的字节数。
<code>Loader.bytesTotal</code>	指明内容中的总字节数的只读属性。
<code>Loader.content</code>	对加载器内容的引用。该属性为只读。
<code>Loader.contentPath</code>	一个字符串，它指明要加载的内容的 URL。
<code>Loader.percentLoaded</code>	一个数字，它指明已加载内容的百分比。该属性为只读。
<code>Loader.scaleContent</code>	一个布尔值，它指示是内容会进行缩放以适合加载器 (true)，还是加载器会进行缩放以适合内容 (false)。

## 从 Screen 类继承的属性

下表列出了 Slide 类从 Screen 类继承的属性。从 Slide 对象访问这些属性时，请使用 *SlideInstance.propertyName* 的形式。

属性	说明
<code>Screen.currentFocusedScreen</code>	只读；返回包含全局当前焦点的屏幕。
<code>Screen.indexInParent</code>	只读；返回该屏幕在其父屏幕的子屏幕列表中的索引（从零开始）。
<code>Screen.numChildScreens</code>	只读；返回屏幕包含的子屏幕的数量。
<code>Screen.parentIsScreen</code>	只读；返回一个布尔值（true 或 false），该值指示屏幕的父对象本身是否是屏幕。
<code>Screen.rootScreen</code>	只读；返回包含此屏幕的树或子树的根屏幕。

# Slide 类的事件摘要

下表列出了 Slide 类的事件。

事件	说明
<a href="#">Slide.hideChild</a>	每次当幻灯片的子幻灯片从可见变为不可见时进行广播。
<a href="#">Slide.revealChild</a>	每次当幻灯片对象的子幻灯片从不可见变为可见时进行广播。

## 从 UIObject 类继承的事件

下表中列出了 Slide 类从 UIObject 类继承的事件。

事件	说明
<a href="#">UIObject.draw</a>	当对象将要绘制它的图形时进行广播。
<a href="#">UIObject.hide</a>	在对象的状态从可见变为不可见时广播。
<a href="#">UIObject.load</a>	创建子对象时广播。
<a href="#">UIObject.move</a>	移动了对象时广播。
<a href="#">UIObject.resize</a>	在调整对象大小后广播。
<a href="#">UIObject.reveal</a>	在对象的状态从不可见变为可见时广播。
<a href="#">UIObject.unload</a>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 Slide 类从 UIComponent 类继承的事件。

事件	说明
<a href="#">UIComponent.focusIn</a>	当对象收到焦点时进行广播。
<a href="#">UIComponent.focusOut</a>	当对象失去焦点时进行广播。
<a href="#">UIComponent.keyDown</a>	当按下按键时进行广播。
<a href="#">UIComponent.keyUp</a>	当松开按键时进行广播。

## 从 Loader 类继承的事件

下表列出了 Slide 类从 Loader 类继承的事件。

事件	说明
<a href="#">Loader.complete</a>	当内容加载完成时触发。
<a href="#">Loader.progress</a>	在内容加载过程中触发。

## 从 Screen 类继承的事件

下表列出了 Slide 类从 Screen 类继承的事件。

事件	说明
<a href="#">Screen.allTransitionsInDone</a>	当应用到屏幕的所有“输入”过渡结束时进行广播。
<a href="#">Screen.allTransitionsOutDone</a>	当应用到屏幕的所有“输出”过渡结束时进行广播。
<a href="#">Screen.mouseDown</a>	当在直接属于屏幕的对象（形状或影片剪辑）上按下鼠标按钮时进行广播。
<a href="#">Screen.mouseDownSomewhere</a>	当在舞台上的某处（不一定要在属于此屏幕的对象上）按下鼠标按钮时进行广播。
<a href="#">Screen.mouseMove</a>	当鼠标在屏幕上移动时进行广播。
<a href="#">Screen.mouseOut</a>	当鼠标从屏幕内移出时进行广播。
<a href="#">Screen.mouseOver</a>	当鼠标从屏幕外移入时进行广播。
<a href="#">Screen.mouseUp</a>	当在直接属于屏幕的对象（形状或影片剪辑）上松开鼠标按钮时进行广播。
<a href="#">Screen.mouseUpSomewhere</a>	当在舞台上的某处（不一定要在属于此屏幕的对象上）松开鼠标按钮时进行广播。

# Slide.autoKeyNav

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*mySlide*.autoKeyNav

### 说明

属性；确定当 *mySlide* 具有焦点时，幻灯片是否使用默认的键盘处理来导航到下一张 / 上一张幻灯片。该属性接受字符串值 `"true"`、`"false"` 和 `"inherit"`。您可以使用 [Slide.defaultKeyDownHandler](#) 属性覆盖此默认的键盘处理行为。

如果该属性的值为 `"true"`，则在 *mySlide* 具有焦点时，按向右箭头键 (`Key.RIGHT`) 或空格键 (`Key.SPACE`) 会前进到下一张幻灯片，按向左箭头键 (`Key.Left`) 会移到上一张幻灯片。

如果该属性设置为 `"false"`，则在 *mySlide* 具有焦点时不发生默认的键盘处理操作。

如果该属性设置为 "inherit", 则 *mySlide* 会检查其父幻灯片的 `autoKeyNav` 属性。如果父幻灯片的该属性也设置为 "inherit", 则 **Flash** 会查找此幻灯片的继承链, 直到找到一个 `autoKeyNav` 属性设置为 "true" 或 "false" 的父幻灯片为止。

如果 *mySlide* 没有父幻灯片 (即, 如果 `(mySlide.parentIsSlide == false)` 为 true), 则它会表现出仿佛 `autoKeyNav` 设置为 "true" 的行为。

### 示例

此示例为名为 `loginSlide` 的幻灯片关闭自动键盘导航。

```
_root.Presentation.loginSlide.autoKeyNav = "false";
```

### 另请参见

[Slide.defaultKeyDownHandler](#)

## Slide.currentChildSlide

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
mySlide.currentChildSlide
```

### 说明

属性 (只读): 返回包含当前活动幻灯片的 *mySlide* 的直接子幻灯片; 如果 *mySlide* 包含的子幻灯片都不具有当前焦点, 则返回 null。

### 示例

考虑以下屏幕大纲:

```
Presentation
 Slide_1
 Bullet1_1
 SubBullet1_1_1
 Bullet1_2
 SubBullet1_2_1
 Slide_2
```

假设 SubBullet1\_1\_1 为当前幻灯片，则以下语句全部为 **true**：

```
Presentation.currentChildSlide == Slide_1;
Slide_1.currentChildSlide == Bullet_1_1;
SubBullet_1_1_1.currentChildSlide == null;
Slide_2.currentChildSlide == null;
```

**另请参见**

[Slide.currentSlide](#)

## Slide.currentFocusedSlide

**可用性**

Flash Player 6 (6.0.79.0)。

**版本**

Flash MX Professional 2004。

**用法**

```
mx.screens.Slide.currentFocusedSlide
```

**说明**

属性（只读）：返回包含全局当前焦点的“最内叶”幻灯片（离幻灯片树的根最远的幻灯片）。实际的焦点可能在幻灯片自身上，或者在该幻灯片内的影片剪辑、文本对象或组件上；如果没有当前焦点，则该方法返回 `null`。

**示例**

```
var focusedSlide = mx.screens.Slide.currentFocusedSlide;
```

## Slide.currentSlide

**可用性**

Flash Player 6 (6.0.79.0)。

**版本**

Flash MX Professional 2004。

**用法**

```
mySlide.currentSlide
```

## 说明

属性（只读）：返回当前活动的幻灯片。返回的始终是“叶”幻灯片，即不包含子幻灯片的幻灯片。

## 示例

以下代码附加到演示文稿根幻灯片上的一个按钮，并会在每次单击此按钮时进入到幻灯片演示文稿的下一张幻灯片。

```
// 附加到演示文稿幻灯片所包含的按钮实例：
on(press) {
 _parent.currentSlide.gotoNextSlide();
}
```

## 另请参见

[Slide.gotoNextSlide\(\)](#)

# Slide.defaultKeyDownHandler

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.defaultKeyDownHandler = function (eventObj) {
 // 此处是您的代码。
}
```

## 参数

*eventObj* 具有以下属性的事件对象：

- **type** 一个指示事件类型的字符串。可能值为 "keyUp" 和 "keyDown"。
- **ascii** 一个整数，它表示上次按下的按键的 ASCII 值；与 `Key.getAscii()` 返回的值对应。
- **code** 一个整数，它表示上次按下的按键的按键代码；与 `Key.getCode()` 所返回的值对应。
- **shiftKey** 一个布尔值，指示当前是 (true) 否按下了 (false) Shift 键。
- **ctrlKey** 一个布尔值，指示当前是 (true) 否按下了 (false) Ctrl 键。

## 返回

无。

## 说明

回调函数：可让您用所创建的自定义键盘处理函数覆盖默认的键盘导航。例如，您可以不使用向左和向右箭头键分别导航到演示文稿中的上一张和下一张幻灯片，而使用向上和向下箭头键执行这些功能。有关默认的键盘处理行为的论述，请参见 [Slide.autoKeyNav](#)。

## 示例

在该示例中，对于 `on(load)` 处理函数所附加到的幻灯片的子幻灯片，默认的键盘处理发生了改变。此处理函数使用向上和向下箭头键而不是向左和向右箭头键进行导航。

```
on (load) {
 this.defaultKeyDownHandler = function(eventObj:Object) {
 switch (eventObj.code) {
 case Key.DOWN :
 this.currentSlide.gotoNextSlide();
 break;
 case Key.UP :
 this.currentSlide.gotoPreviousSlide();
 break;
 default :
 break;
 }
 };
}
```

## 另请参见

[Slide.autoKeyNav](#)

# Slide.firstSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.firstSlide

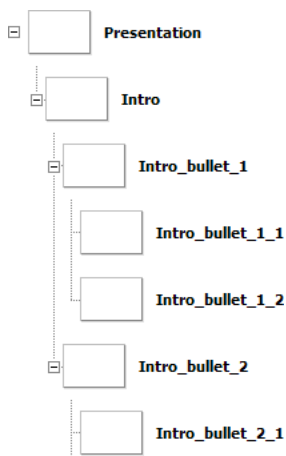
## 说明

属性（只读）：返回 *mySlide* 的第一个没有子幻灯片的子幻灯片。

## 示例

在下面显示的幻灯片层次结构中，以下语句全部为 **true**：

```
Presentation.Intro.firstSlide == Intro_bullet_1_1;
Presentation.Intro_bullet_1.firstSlide == Intro_bullet_1_1;
```



# Slide.getChildSlide()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.getChildSlide(childIndex)
```

## 参数

*childIndex* 要返回的子幻灯片的索引（从零开始）。

## 返回

一个幻灯片对象。

## 说明

方法；返回 *mySlide* 的子幻灯片（其索引为 *childIndex*）。此方法可在重复一组已知索引的子幻灯片时使用。



## 示例

以下代码会使“输出”面板显示演示文稿根幻灯片的所有子幻灯片的名称。

```
var numSlides = _root.Presentation.numChildSlides;
for(var slideIndex=0; slideIndex < numSlides; slideIndex++) {
 var childSlide = _root.Presentation.getChildSlide(slideIndex);
 trace(childSlide._name);
}
```

## 另请参见

[Slide.numChildSlides](#)

# Slide.gotoFirstSlide()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.gotoFirstSlide()

## 参数

无。

## 返回

无。

## 说明

方法；导航到 *mySlide* 下面的子幻灯片树中的第一张叶幻灯片。从某张幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数内进行调用时，如果该事件是幻灯片导航的结果，则会忽略此方法。

若要转到演示文稿中的第一张幻灯片，请调用 `mySlide.rootSlide.gotoFirstSlide()`。（有关 `rootSlide` 的更多信息，请参见 [Slide.revealChild](#)。）

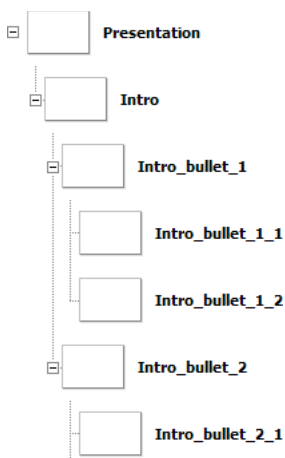
## 示例

在下面显示的幻灯片层次结构中，以下方法调用都会导航到名为 `Intro_bullet_1_1` 的幻灯片：

```
Presentation.gotoFirstSlide();
Presentation.Intro.gotoFirstSlide();
Presentation.Intro.Intro_bullet_1.gotoFirstSlide();
```

此方法调用将导航到名为 `Intro_bullet_2_1` 的幻灯片：

```
Presentation.Intro.Intro_bullet_2.gotoFirstSlide();
```



另请参见

[Slide.firstSlide](#)、[Slide.revealChild](#)

# Slide.gotoLastSlide()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.gotoLastSlide()
```

## 参数

无。

## 返回

无。

## 说明

方法：导航到 *mySlide* 下面的子幻灯片树中的最后一张叶幻灯片。从某张幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数内进行调用时，如果该事件是另一个幻灯片导航的结果，则会忽略此方法。

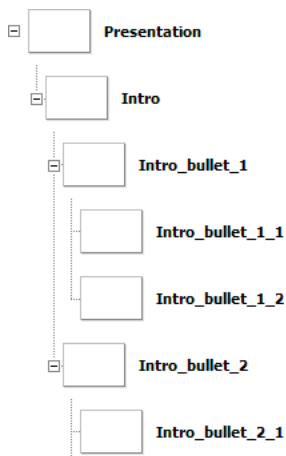
## 示例

在下面显示的幻灯片层次结构中，以下方法调用都会导航到名为 `Intro_bullet_1_2` 的幻灯片：

```
Presentation.Intro.gotoLastSlide();
Presentation.Intro.Intro_bullet_1.gotoLastSlide();
```

这些方法调用将导航到名为 `Intro_bullet_2_1` 的幻灯片：

```
Presentation.gotoLastSlide();
Presentation.Intro.gotoLastSlide();
```



## 另请参见

[Slide.gotoSlide\(\)](#)、[Slide.lastSlide](#)

# Slide.gotoNextSlide()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.gotoNextSlide()
```

## 参数

无。

## 返回

一个布尔值，或 `null`。如果此方法成功地导航到下一张幻灯片，则返回 `true`；如果在调用此方法时，演示文稿已经处于最后一张幻灯片（即，如果 `currentSlide.nextSlide` 为 `null`），则返回 `false`。如果在不包含当前幻灯片的幻灯片上调用此方法，则返回 `null`。

## 说明

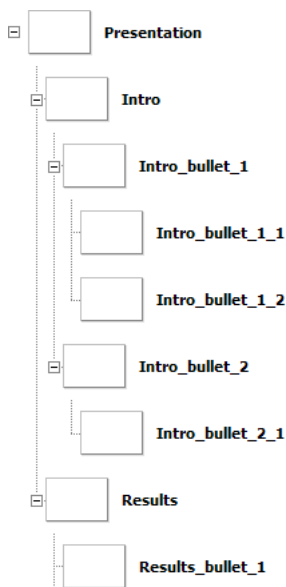
方法：导航到幻灯片演示文稿中的下一张幻灯片。当控制从一张幻灯片传到下一张幻灯片时，会隐藏传出的幻灯片并显示传入的幻灯片。如果传出和传入幻灯片位于不同的幻灯片子树中，则所有始祖幻灯片（从传出幻灯片开始，一直到传入和传出幻灯片的公共始祖）都被隐藏起来并接收 `hide` 事件。紧接着，从传入幻灯片的所有始祖幻灯片一直到传出和传入幻灯片的公共始祖都会可见，并且接收 `reveal` 事件。

通常，`gotoNextSlide()` 是在代表当前幻灯片的叶节点上调用的。如果在非叶节点 `someNode` 上调用，则 `someNode.gotoNextSlide()` 会进入到下一张幻灯片或“部分”中的第一个叶节点。

在不包含当前幻灯片的幻灯片上调用时，此方法没有效果。从附加到幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数中调用此方法时，如果该处理函数是作为幻灯片导航的结果调用的，则此方法也无效。

## 示例

假设在下面的幻灯片层次结构中，名为 `Intro_bullet_1_1` 的幻灯片是当前查看的幻灯片（即 `_root.Presentation.currentSlide._name == Intro_bullet_1_1`）。



在这种情况下，调用 `Intro_bullet_1_1.gotoNextSlide()` 时将导航到 `Intro_bullet_1_2`，它是 `Intro_bullet_1_1` 的同级幻灯片。

然而，调用 `Intro_bullet_1.gotoNextSlide()` 时将导航到 `Intro_bullet_2_1`，它是 `Intro_bullet_2` 包含的第一张叶幻灯片，而后者是 `Intro_bullet_1` 的下一张同级幻灯片。类似地，调用 `Intro.gotoNextSlide()` 时将导航到 `Results_bullet_1`，它是 `Results` 幻灯片所包含的第一张叶幻灯片。

另外，仍然假设当前幻灯片是 `Intro_bullet_1_1`，对 `Results.gotoNextSlide()` 的调用将无效，因为 `Results` 不包含当前幻灯片（即 `Results.currentSlide` 为 `null`）。

## 另请参见

[Slide.currentSlide](#)、[Slide.gotoPreviousSlide\(\)](#)、[Slide.nextSlide](#)

# Slide.gotoPreviousSlide()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.gotoPreviousSlide()
```

## 参数

无。

## 返回

一个布尔值，或 `null`。如果此方法成功地导航到上一张幻灯片，则返回 `true`；如果在调用此方法时，演示文稿已经处于第一张幻灯片（即，如果 `currentSlide.nextSlide` 为 `null`），则返回 `false`。如果在不包含当前幻灯片的幻灯片上调用此方法，则返回 `null`。

## 说明

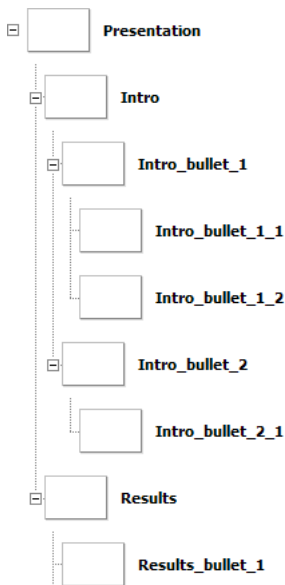
方法：导航到幻灯片演示文稿中的上一张幻灯片。当控制从一张幻灯片传到上一张幻灯片时，会隐藏传出的幻灯片并显示传入的幻灯片。如果传出和传入幻灯片位于不同的幻灯片子树中，则所有始祖幻灯片（从传出幻灯片开始，一直到传入和传出幻灯片的公共始祖）都被隐藏起来并接收 `hide` 事件。紧接着，从传入幻灯片的所有始祖幻灯片一直到传出和传入幻灯片的公共始祖都会可见，并且接收 `reveal` 事件。

通常，`gotoPreviousSlide()` 是在代表当前幻灯片的叶节点上调用的。如果在非叶节点 `someNode` 上调用，则 `someNode.gotoPreviousSlide()` 会进入到上一张幻灯片或“部分”中的第一个叶节点。

在不包含当前幻灯片的幻灯片上调用时，此方法没有效果。从附加到幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数中调用此方法时，如果该处理函数是作为幻灯片导航的结果调用的，则此方法也无效。

## 示例

假设在下面的幻灯片层次结构中，名为 `Intro_bullet_1_2` 的幻灯片是当前查看的幻灯片（即 `_root.Presentation.currentSlide._name == Intro_bullet_1_2`）。



在这种情况下，调用 `Intro_bullet_1_2.gotoPreviousSlide()` 时将导航到 `Intro_bullet_1_1`，它是 `Intro_bullet_1_2` 的上一张同级幻灯片。

然而，调用 `Intro_bullet_2.gotoPreviousSlide()` 时将导航到 `Intro_bullet_1_1`，它是 `Intro_bullet_1` 包含的第一张叶幻灯片，而后者是 `Intro_bullet_2` 的上一张同级幻灯片。类似地，调用 `Results.gotoPreviousSlide()` 时将导航到 `Intro_bullet_1_1`，它是 `Intro` 幻灯片包含的第一张叶幻灯片。

另外，如果当前幻灯片是 `Intro_bullet_1_1`，则对 `Results.gotoPreviousSlide()` 的调用将无效，原因是 `Results` 不包含当前幻灯片（即 `Results.currentSlide` 为 `null`）。

## 另请参见

[Slide.currentSlide](#)、[Slide.gotoNextSlide\(\)](#)、[Slide.previousSlide](#)

# Slide.gotoSlide()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.gotoSlide(newSlide)
```

## 参数

*newSlide* 要导航到的幻灯片。

## 返回

一个布尔值，它指示导航是 (true) 否 (false) 成功。

## 说明

方法；导航到由 *newSlide* 指定的幻灯片。要使导航成功，必须满足以下条件：

- 当前幻灯片必须是 *mySlide* 的子幻灯片。
- 由 *newSlide* 指定的幻灯片和当前幻灯片必须具有共同的始祖幻灯片，也即当前幻灯片和 *newSlide* 必须位于同一个幻灯片子树中。

如果任何一个条件得不到满足，导航就会失败，且方法返回 `false`；否则，方法就会导航到指定的幻灯片并返回 `true`。

例如，考虑以下幻灯片层次结构：

```
Presentation
 Slide1
 Slide1_1
 Slide1_2
 Slide2
 Slide2_1
 Slide2_2
```

如果当前幻灯片为 `Slide1_2`，则下面对 `gotoSlide()` 的调用将会失败，因为当前幻灯片不是 `Slide2` 的后代：

```
Slide2.gotoSlide(Slide2_1);
```

也请考虑下面的屏幕层次结构，在该结构中，表单对象是两个单独的幻灯片树的父屏幕：

```
Form_1
 Slide1
 Slide1_1
 Slide1_2
```



```
Slide2
 Slide2_1
 Slide2_2
```

如果当前幻灯片是 `Slide1_2`，则下面的方法调用也会失败，因为 `Slide1` 和 `Slide2` 位于不同的幻灯片子树中：

```
Slide1_2.gotoSlide(Slide2_2);
```

### 示例

以下附加到 **Button** 组件的代码使用 `Slide.currentSlide` 属性和 `gotoSlide()` 方法显示演示文稿中的下一张幻灯片。

```
on(click) {
 _parent.gotoSlide(_parent.currentSlide.nextSlide);
}
```

上面的语句等同于下面使用 `Slide.gotoNextSlide()` 方法的代码：

```
on(click) {
 _parent.currentSlide.gotoNextSlide();
}
```

### 另请参见

[Slide.currentSlide](#)、[Slide.gotoNextSlide\(\)](#)

## Slide.hideChild

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
on(hideChild) {
 // 此处是您的代码。
}
```

### 说明

事件：每次当幻灯片的子幻灯片从可见变为不可见时进行广播。此事件仅由幻灯片（而不是表单）来进行广播。`hideChild` 事件的主要用途是将“输出”过渡应用于某个幻灯片的所有子幻灯片。

## 示例

当附加到根幻灯片（例如，演示文稿幻灯片）时，以下代码将在属于根幻灯片的每张子幻灯片出现时显示其名称。

```
on(hideChild) {
 var child = eventObj.target._name;
 trace(child + " has just been hidden");
}
```

## 另请参见

[Slide.revealChild](#)

# Slide.indexInParentSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.indexInParent

## 说明

属性（只读）：返回 *mySlide* 在其父幻灯片的子幻灯片列表中的索引（从零开始）。

## 示例

以下代码使用 `indexInParentSlide` 和 [Slide.numChildSlides](#) 属性显示当前正在查看的幻灯片的索引，以及其父幻灯片所包含的幻灯片的总数。要使用这段代码，请将其附加到包含一张或多张子幻灯片的父幻灯片。

```
on (revealChild) {
 trace("Displaying "+(currentSlide.indexInParentSlide+1)+" of
 "+currentSlide._parent.numChildSlides);
}
```

请注意，因为此属性是从零开始的索引，所以其值以 1 为增量 (`currentSlide.indexInParent+1`)，以显示更多有意义的值。

## 另请参见

[Slide.numChildSlides](#)、[Slide.revealChild](#)

# Slide.lastSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.lastSlide

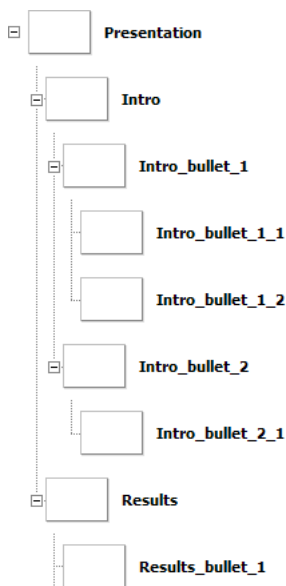
## 说明

属性（只读）：返回 *mySlide* 的最后一张没有子幻灯片的子幻灯片。

## 示例

对于下面显示的幻灯片层次结构，以下语句全部为 **true**：

```
Presentation.lastSlide._name == Results_bullet_1;
Intro.lastSlide._name == Intro_bullet_1_2;
Intro_bullet_1.lastSlide._name == Intro_bullet_1_2;
Results.lastSlide._name == Results_bullet_1;
```



# Slide.nextSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.nextSlide

## 说明

属性（只读）；返回在调用 *mySlide*.gotoNextSlide() 时应到达但实际不导航到的幻灯片。例如，您可以使用此属性显示演示文稿中的下一张幻灯片的名称，并让用户选择是否要导航到该幻灯片。

## 示例

在此例中，名为 nextButton 的 **Button** 组件的标签会显示演示文稿中的下一张幻灯片的名称。如果没有下一张幻灯片（即，如果 *mySlide*.nextSlide 为 null），则该按钮的标签会被更新，以指示用户位于此幻灯片演示文稿的末尾。

```
if (mySlide.nextSlide != null) {
 nextButton.label = "Next slide: " + mySlide.nextSlide._name + " > ";
} else {
 nextButton.label = "End of this slide presentation.";
}
```

## 另请参见

[Slide.gotoNextSlide\(\)](#)、[Slide.previousSlide](#)

# Slide.numChildSlides

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.numChildSlides

## 说明

属性（只读）；返回 *mySlide* 包含的子幻灯片的数量。幻灯片可以包含表单或其它幻灯片；如果 *mySlide* 既包含幻灯片又包含表单，则此属性仅返回幻灯片的数量，而不对表单计数。

## 示例

本示例使用 `Slide.numChildSlides` 和 `Slide.getChildSlide()` 方法来循环通过演示文稿根幻灯片的所有子幻灯片。然后在“输出”面板中显示它们的名称。

```
var numSlides = _root.Presentation.numChildSlides;
for(var slideIndex=0; slideIndex < numSlides; slideIndex++) {
 var childSlide = _root.Presentation.getChildSlide(slideIndex);
 trace(childSlide._name);
}
```

## 另请参见

[Slide.getChildSlide\(\)](#)

# Slide.overlayChildren

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.overlayChildren

## 说明

属性；确定在从一张子幻灯片导航到下一张时，*mySlide* 的子幻灯片是否保持可见。如果此属性设置为 `true`，则当控制传递到下一张同级幻灯片时，上一张幻灯片保持可见；如果此属性设置为 `false`，则当控制传递到下一张同级幻灯片时，上一张幻灯片不可见。

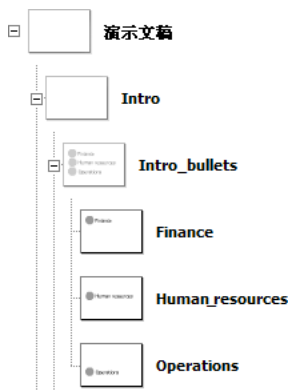
将此属性设置为 `true` 很有用，例如在这种情况下：给定的幻灯片包含几张分别显示的子“项目符号”幻灯片（很可能使用过渡），但都需要在新项目符号出现时保持可见。



此属性仅适用于 *mySlide* 的直接后代，而并不适用于所有（嵌套的）子幻灯片。

## 示例

以下图示中的 **Intro\_bullets** 幻灯片包含三张子幻灯片（**Finance**、**Human\_resources** 和 **Operations**），而每张子幻灯片均显示一个单独的项目符号。通过将 `Intro_bullets.overlayChildren` 设置为 `true`，每张项目符号幻灯片将在其它项目符号出现时仍保留在舞台上。



# Slide.parentIsSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`mySlide.parentIsSlide`

## 说明

属性（只读）：一个布尔值，它指示 `mySlide` 的父对象是否也是幻灯片。如果 `mySlide` 的父对象是幻灯片，或者属于 **Slide** 的子类，则此属性将返回 `true`；否则返回 `false`。

如果 `mySlide` 是演示文稿中的根幻灯片，则此属性将返回 `false`，因为演示文稿幻灯片的父对象是主时间轴（`_level0`）而不是幻灯片。如果 `mySlide` 的父对象是表单，此属性也会返回 `false`。

## 示例

以下代码确定幻灯片 `mySlide` 的父对象本身是否也是幻灯片。如果 `mySlide.parentIsSlide` 为 `true`，则会在“输出”面板中显示 `mySlide` 的同级幻灯片数量。如果父对象不是幻灯片，则 **Flash** 将假设 `mySlide` 为演示文稿的根（主）幻灯片，因而没有同级幻灯片。

```
if (mySlide.parentIsSlide) {
 trace("I have " + mySlide._parent.numChildSlides+" sibling slides");
} else {
 trace("I am the root slide and have no siblings");
}
```

## 另请参见

[Slide.numChildSlides](#)

# Slide.parentSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.parentSlide

## 说明

属性（只读）：对包含当前幻灯片的幻灯片的引用。

# Slide.playHidden

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.playHidden

## 说明

属性：一个布尔值，指定 *mySlide* 在隐藏时是否应继续播放。当此属性为 `true` 时，*mySlide* 将在隐藏时继续播放。设置为 `false` 时，*mySlide* 将在隐藏时停止播放；而当显示时，则在 *mySlide* 的第 1 帧处重新开始播放。

# Slide.previousSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*mySlide*.previousSlide

## 说明

属性（只读）：返回在调用 *mySlide*.gotoPreviousSlide() 时应到达但实际不导航到的幻灯片。例如，您可以使用此属性显示演示文稿中上一张幻灯片的名称，并让用户选择是否要导航到该幻灯片。

## 示例

在此例中，名为 `previousButton` 的 **Button** 组件的标签会显示演示文稿中上一张幻灯片的名称。如果没有上一张幻灯片（即，如果 *mySlide*.previousSlide 为 `null`），则该按钮的标签会被更新，以指示用户位于此幻灯片演示文稿的最开始处。

```
if (mySlide.previousSlide != null) {
 previousButton.label = "Previous slide: " + mySlide.previous._name + ">
 ";
} else {
 previousButton.label = "You're at the beginning of this slide
 presentation.";
}
```

## 另请参见

[Slide.gotoPreviousSlide\(\)](#)、[Slide.nextSlide](#)



# Slide.revealChild

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
on(revealChild) {
 // 此处是您的代码。
}
```

## 说明

事件：每次当幻灯片对象的子幻灯片从不可见变为可见时进行广播。此事件主要用于将“输入”过渡附加到给定幻灯片的所有子幻灯片。

## 示例

当附加到根幻灯片（如演示文稿幻灯片）时，以下代码将在每张子幻灯片出现时显示其名称。

```
on(revealChild) {
 var child = eventObj.target._name;
 trace(child + " has just appeared");
}
```

## 另请参见

[Slide.hideChild](#)

# Slide.rootSlide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
mySlide.rootSlide
```

## 说明

属性（只读）：返回包含 *mySlide* 的幻灯片树或幻灯片子树的根幻灯片。

## 示例

假设在幻灯片上有一个影片剪辑，单击它会转到演示文稿中的第一张幻灯片。要达到此目的，可将以下代码附加到影片剪辑：

```
on(press) {
 _parent.rootSlide.gotoFirstSlide();
}
```

在本例中，\_parent 指包含影片剪辑对象的幻灯片。

# StyleManager 类

ActionScript 类名称 `mx.styles.StyleManager`

`StyleManager` 类会跟踪已知的继承样式和颜色。只有在您要创建组件并希望添加新的继承样式或颜色时，才需要使用该类。

若要确定哪些样式是继承样式，请查看 W3C 网站，网址是：[www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)。

## StyleManager 类的方法摘要

下表列出了 `StyleManager` 类的方法。

方法	说明
<code>StyleManager.registerColorName()</code>	使用样式管理器注册新的颜色名称。
<code>StyleManager.registerColorStyle()</code>	将新的颜色样式添加到样式管理器。
<code>StyleManager.registerInheritingStyle()</code>	使用样式管理器注册新的继承样式。

## StyleManager.registerColorName()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
StyleManager.registerColorName(colorName, value)
```

### 参数

*colorName* 指示颜色名称的字符串（如 "gray"、"darkGrey" 等）。

*value* 指示颜色的十六进制数（如 0x808080、0x404040 等）。

## 返回

无。

## 说明

方法：将颜色名称与十六进制数值相关联并使用样式管理器注册颜色名称。

## 示例

以下示例将 "gray" 注册为颜色的名称，该颜色是以十六进制数值 0x808080 表示的：

```
StyleManager.registerColorName("gray", 0x808080);
```

# StyleManager.registerColorStyle()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
StyleManager.registerColorStyle(colorStyle)
```

## 参数

*colorStyle* 指示颜色名称的字符串（如 "highlightColor"、"shadowColor"、"disabledColor" 等）。

## 返回

无。

## 说明

方法：将新的颜色样式添加到样式管理器。

## 示例

以下示例将 "highlightColor" 注册为颜色样式：

```
StyleManager.registerColorStyle("highlightColor");
```

# StyleManager.registerInheritingStyle()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
StyleManager.registerInheritingStyle(propertyName)
```

## 参数

*propertyName* 指示样式属性名称的字符串（如 "newProp1"、"newProp2" 等）。

## 返回

无。

## 说明

方法：将此样式属性标记为继承。使用该方法注册未在 CSS 规范中列出的样式属性。请不要使用该方法将非继承样式属性更改为继承。

当某一样式的值不是继承的时，只能在实例上设置其样式，而不能在自定义或全局样式表上设置。不继承样式值的样式是在类样式表中设置的，因此，在自定义或全局样式表中设置此样式是不起作用的。

## 示例

以下示例将 newProp1 注册为继承样式：

```
StyleManager.registerInheritingStyle("newProp1");
```



# SystemManager 类

ActionScript 类名称 `mx.managers.SystemManager`

`SystemManager` 类与 `FocusManager` 类自动配合工作，用来处理在包含第 2 版组件的应用程序中激活哪个顶层窗口。它还提供一个 `screen` 属性，而该属性允许组件和影片剪辑访问舞台坐标。

## SystemManager 类的属性摘要

下表列出了 `SystemManager` 类的属性。

属性	说明
<code>SystemManager.screen</code>	只读；包含舞台位置和大小对象。

## SystemManager.screen

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

`SystemManager.screen`

### 说明

属性：一个对象，带有指示舞台大小和位置的属性：`x`、`y`、`width` 和 `height`。

## 示例

如果将 `Stage.align` 设置为 "LT" 以外的其它对象，将很难知道实际上哪些坐标是可查看的。

假设您要将一个水印影片剪辑置于舞台右下角（类似于许多电视频道使用的水印）。以下代码将在影片剪辑实例 `watermark` 的所有舞台对齐方式中使用：

```
import mx.managers.SystemManager;

var p1:Number = SystemManager.screen.width + SystemManager.screen.x -
 watermark._width;
var p2:Number = SystemManager.screen.height + SystemManager.screen.y -
 watermark._height;

watermark._x = p1;
watermark._y = p2;
```



## TextArea 组件

TextArea 组件的效果等于将 `ActionScript` 的 `TextField` 对象进行换行。您可以使用样式自定义 `TextArea` 组件；当实例被禁用时，其内容以 `disabledColor` 样式所指示的颜色显示。`TextArea` 组件也可以采用 HTML 格式，或者作为掩饰文本的密码字段。请参见《学习 Flash 中的 `ActionScript 2.0`》中的“将样式表应用于 `TextArea` 组件”。

在应用程序中可以启用或禁用 `TextArea` 组件。在禁用状态下，它不接收鼠标或键盘输入。当启用时，它遵循与 `ActionScript TextField` 对象相同的焦点、选择和导航规则。当 `TextArea` 实例具有焦点时，您可以使用下列按键对其进行控制：

键	说明
箭头键	将插入点向上、向下、向左或向右移动一行。
Page Down	向下移动一屏。
Page Up	向上移动一屏。
Shift+Tab	将焦点移到前一个对象。
Tab	将焦点移到下一个对象。

有关控制焦点的更多信息，请参见《使用组件》中的“创建自定义焦点导航”或第 663 页的“`FocusManager` 类”。

每个 `TextArea` 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。如果需要滚动条，它会出现实时预览中，但并不起作用。在实时预览中，文本是不可选定的，并且无法在舞台上的组件实例中输入文本。

在将 `TextArea` 组件添加到应用程序中时，您可以使用“辅助功能”面板使其可由屏幕读取器访问。

# 使用 TextArea 组件

在需要多行文本字段的任何地方都可使用 **TextArea** 组件。如果需要单行文本字段，请使用 **TextInput** 组件。例如，您可以在表单中使用 **TextArea** 组件作为注释字段。您可以设置侦听器来检查当用户切换到字段外时，字段是否为空。侦听器可能会显示错误信息，以指明必须在该字段中输入注释。

## TextArea 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 **TextArea** 组件实例设置以下创作参数：

**editable** 指示 **TextArea** 组件是 (true) 否 (false) 可编辑。默认值为 true。

**html** 指示文本是 (true) 否 (false) 采用 HTML 格式。如果 HTML 设置为 true，则可以使用字体标签来设置文本格式。默认值为 false。

**text** 指示 **TextArea** 组件的内容。您无法在“属性”检查器或“组件”检查器中输入回车。默认值为 ""（空字符串）。

**wordWrap** 指示文本是 (true) 否 (false) 自动换行。默认值为 true。



如果您使用 `createClassObject()` 方法创建 **TextArea**，则 **wordWrap** 的默认值为 false。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 **TextArea** 组件实例设置以下附加参数：

**maxChars** 是文本区域最多可以容纳的字符数。默认值为 null（表示无限制）。

**restrict** 指示用户可输入文本区域中的字符集。默认值为 undefined。请参见第 1105 页的“**TextArea.restrict**”。

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 true。

**password** 是一个布尔值，它指示输入的是密码还是其它在键入时应该隐藏起来的文本。**Flash** 以星号形式隐藏输入字符。默认值为 false。

**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。



**minHeight** 和 **minWidth** 属性由内部的大小调整例程使用。它们在 **UIObject** 中定义，必要时，其它组件将会覆盖这两个属性。如果要为应用程序创建一个自定义布局管理器，则可以使用这两个属性。否则，在“组件”检查器中设置这两个属性将不会有明显的效果。

您可以编写 **ActionScript**，以便使用 **TextArea** 组件的属性、方法和事件来控制该组件的这些选项和其它选项。有关更多信息，请参见第 1090 页的“**TextArea 类**”。

## 创建具有 TextArea 组件的应用程序

以下过程解释了如何在创作时将 **TextArea** 组件添加到应用程序。该示例在 **TextArea** 实例上设置了一个 `focusOut` 事件处理函数，用来验证用户在将焦点移到界面其它部分前是否在文本区域中键入了内容。

### 创建具有 TextArea 组件的应用程序：

1. 将一个 **TextArea** 组件从“组件”面板拖到舞台上，然后为其指定实例名 **my\_ta**。
2. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
/**
 要求：
 - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

var taListener:Object = new Object();
taListener.focusOut = function(evt_obj:Object) {
 if (my_ta.length < 1) {
 trace("Please enter a comment");
 }
};
my_ta.addEventListener("focusOut", taListener);
```

这段代码在 **TextArea** 组件实例上设置了一个 `focusOut` 事件处理函数，该事件处理函数用来验证用户是否在文本区域中键入了内容。

您可以获取在 **TextArea** 实例中输入的文本值，如下所示：

```
var ta_text:String = my_ta.text;
```

# 自定义 TextArea 组件

在创作过程中和运行时，可以在水平和垂直方向上改变 `TextArea` 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `UIObject.setSize()` 或者 `TextArea` 类中任何适当的属性和方法。

调整了 `TextArea` 组件的大小后，边框大小就会调整到新的边框。如果需要滚动条，它们会被放置在下边缘和右边缘。然后，文本区域的大小会在其余区域内调整；`TextArea` 组件中没有大小固定的元素。如果 `TextArea` 组件太小不能显示文本，文本就会被裁剪。

## 对 TextArea 组件使用样式

`TextArea` 组件在类样式声明中定义了它的 `backgroundColor` 和 `borderStyle` 样式属性。类样式覆盖全局样式，因此，如果要设置 `backgroundColor` 和 `borderStyle` 样式属性，必须在实例上创建一个不同的自定义样式声明。

如果样式属性的名称以“**Color**”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。

`TextArea` 组件支持下列样式：

样式	主题	说明
<code>backgroundColor</code>	光晕和范例	背景色。默认颜色为白色。
<code>borderStyle</code>	光晕和范例	<code>TextArea</code> 组件使用 <code>RectBorder</code> 实例作为其边框并对在该类上定义的样式做出响应。请参见第 985 页的“ <code>RectBorder</code> 类”。  默认边框样式为 <code>"inset"</code> 。
<code>marginLeft</code>	光晕和范例	表示文本左边距的数字。默认值为 0。
<code>marginRight</code>	光晕和范例	表示文本右边距的数字。默认值为 0。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题的默认值为 <code>0x0B333C</code> ，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>0x848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	文本的字体名称。默认值为 <code>"_sans"</code> 。
<code>fontSize</code>	光晕和范例	字体的磅值。默认值为 10。
<code>fontStyle</code>	光晕和范例	字体样式： <code>"normal"</code> 或 <code>"italic"</code> 。默认值为 <code>"normal"</code> 。

样式	主题	说明
fontWeight	光晕和范例	字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textAlign	光晕和范例	文本对齐方式: "left"、"right"、"center" 或 "justify"。(只有 <b>Flash Player 8</b> 中支持 "justify" 参数)。默认值为 "left"。
textIndent	光晕和范例	表示文本缩进的数字。默认值为 0。
textDecoration	光晕和范例	文本修饰: "none" 或 "underline"。默认值为 "none"。

**TextArea** 和 **TextInput** 组件使用完全相同的样式, 并且通常使用方式也相同。因此, 默认情况下它们共享同一个类级别样式声明。

例如, 下面的代码在 **TextInput** 声明时设置样式, 但它同时影响 **TextInput** 和 **TextArea** 组件。

```
_global.styles.TextInput.setStyle("disabledColor", 0xBBBFFF);
```

若要分离这两个组件并且只为其中一个提供类级别样式, 请创建新的样式声明。

```
import mx.styles.CSSStyleDeclaration;
_global.styles.TextArea = new CSSStyleDeclaration();
_global.styles.TextArea.setStyle("disabledColor", 0xFFBBB);
```

此示例在覆盖 `_global.styles.TextArea` 之前, 不会检查它是否存在; 而是假定您知道它的存在并要将其覆盖。

您可以通过在全局范围内将 `backgroundColor` 样式设置为值 `undefined`, 使得 **TextArea** 组件具有透明背景。然后将所有不希望透明的 **TextArea** 组件的 `backgroundColor` 样式单独设置为一种颜色。

```
// 给所有 TextArea 组件以透明背景
_global.styles.TextArea.backgroundColor = undefined;
```

```
// 使此特定组件实例具有白色背景。
myTextArea2.setStyle("backgroundColor", "white");
```

对于字段中的所有文本来说, **TextArea** 组件支持一组组件样式。但是, 您也可以显示与 **Flash Player** 的 **HTML** 呈现兼容的 **HTML**。若要显示 **HTML** 文本, 请将 `TextArea.html` 设置为 `true`。

如果确实将 `TextArea` 设置为显示 HTML 文本，则使用 `TextField.StyleSheet` 类来设置文本样式（有关该类的详细信息，请参见《[ActionScript 2.0 语言参考](#)》）。例如：

1. 将一个 `TextArea` 组件拖到舞台上，然后为其指定实例名称 `my_ta`。

2. 在时间轴第 1 帧的“动作”面板中输入以下代码：

```
var my_styles = new TextField.StyleSheet();
my_styles.setStyle("p", {fontFamily:'Arial,Helvetica,sans-serif',
 fontSize:'12px', color:'#CC6699'});
my_ta.styleSheet = my_styles;
my_ta.html = true;
my_ta.text = "<p>This is some text</p>";
```

## 对 `TextArea` 组件使用外观

`TextArea` 组件使用一个 `RectBorder` 实例作为其边框，并使用滚动条滚动图像。有关设置这些可视元素的外观的更多信息，请参见第 985 页的“[RectBorder 类](#)”和第 1283 页的“[UIScrollBar 组件使用外观](#)”。

# TextArea 类

继承 `MovieClip` > [UIObject 类](#) > [UIComponent 类](#) > `View` > `ScrollView` > `TextArea`

ActionScript 类名称 `mx.controls.TextArea`

`TextArea` 类的属性使您能够在运行时设置文本内容、格式以及水平和垂直位置。您也可以指明该字段是否可编辑，以及该字段是否为“密码”字段。您还可以限制用户可以输入的字符。

使用 `ActionScript` 设置 `TextArea` 类的属性将会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

`TextArea` 组件会覆盖默认的 `Flash Player` 焦点矩形，并绘制一个带圆角的自定义焦点矩形。

`TextArea` 组件支持 `CSS` 样式和 `Flash Player` 所支持的任何其它 `HTML` 样式。

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.TextArea.version);
```



代码 `trace(myTextAreaInstance.version);` 返回 `undefined`。

# TextArea 类的方法摘要

没有 TextArea 类专用的方法。

## 从 UIObject 类继承的方法

下表列出了 TextArea 类从 UIObject 类继承的方法。从 TextArea 对象调用这些方法时，请使用 *TextAreaInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定的类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 TextArea 类从 UIComponent 类继承的方法。从 TextArea 对象调用这些方法时，请使用 *TextAreaInstance.methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# TextArea 类的属性摘要

下表列出了 `TextArea` 类的属性。

属性	说明
<code>TextArea.editable</code>	一个布尔值，它指示该字段是 (true) 否 (false) 可编辑。
<code>TextArea.hPosition</code>	定义文本在字段中的水平位置。
<code>TextArea.hScrollPolicy</code>	指示水平滚动条是始终打开 ("on")、从不打开 ("off") 还是在需要时打开 ("auto")。
<code>TextArea.html</code>	一个布尔值，它指示文本区域的内容是否可以采用 HTML 格式。
<code>TextArea.length</code>	只读；文本区域中的字符数。
<code>TextArea.maxChars</code>	文本区域最多可以容纳的字符数。
<code>TextArea.maxHPosition</code>	只读； <code>TextArea.hPosition</code> 的最大值。
<code>TextArea.maxVPosition</code>	只读； <code>TextArea.vPosition</code> 的最大值。
<code>TextArea.password</code>	一个布尔值，它指示字段是 (true) 否 (false) 为密码字段。
<code>TextArea.restrict</code>	用户可在文本区域中输入的字符集。
<code>TextArea.styleSheet</code>	将样式表附加到指定的 <code>TextArea</code> 组件。
<code>TextArea.text</code>	<code>TextArea</code> 组件的文本内容。
<code>TextArea.vPosition</code>	一个数字，它指示垂直滚动位置。
<code>TextArea.vScrollPolicy</code>	指示垂直滚动条是始终打开 ("on")、从不打开 ("off") 还是在需要时打开 ("auto")。
<code>TextArea.wordWrap</code>	一个布尔值，它指示文本是 (true) 否 (false) 自动换行。

## 从 UIObject 类继承的属性

下表列出了 `TextArea` 类从 `UIObject` 类继承的属性。从 `TextArea` 对象访问这些属性时，请使用 `TextAreaInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	只读；对象的底边缘位置（相对于其父对象的底边缘）。
<code>UIObject.height</code>	只读；对象的高度，以像素为单位。
<code>UIObject.left</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.right</code>	只读；对象的右边缘位置（相对于其父对象的右边缘）。
<code>UIObject.scaleX</code>	只读；一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。



属性	说明
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	只读；对象上边缘的位置（相对于其父对象）。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

## 从 UIComponent 类继承的属性

下表列出了 `TextArea` 类从 `UIComponent` 类继承的属性。从 `TextArea` 对象访问这些属性时，请使用 `TextAreaInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## TextArea 类的事件摘要

下表列出了 `TextArea` 类的事件。

事件	说明
<code>TextArea.change</code>	通知侦听器文本已更改。
<code>TextArea.scroll</code>	通知侦听器文本已滚动。

## 从 UIObject 类继承的事件

下表列出了 `TextArea` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。

事件	说明
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 `TextArea` 类从 `UIComponent` 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

# TextArea.change

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 // ...
};
textAreaInstance.addEventListener("change", listenerObject);
```

用法 2:

```
on (change) {
 // ...
}
```

## 说明

事件；通知侦听器文本已更改。在文本更改后广播该事件。不能使用该事件防止将某些字符添加到组件的文本区域；为实现该目的，应使用 `TextArea.restrict`。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*textAreaInstance*) 调度一个事件（在本例中为 `change`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“`EventDispatcher` 类”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 `TextArea` 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到实例 `myTextArea`，它将“`_level0.myTextArea`”发送到“输出”面板：

```
on (change) {
 trace(this);
}
```

## 示例

此示例使用调度程序 / 侦听器事件模型跟踪一个名为 `my_ta` 的 `TextArea` 组件中文本区域的总更改次数：

必须首先将 `TextArea` 组件的实例添加到舞台上并将其命名为 **`my_ta`**，然后将以下代码添加到第 1 帧中。

```
/**
 要求：
 - 舞台上的 TextArea 实例（实例名：my_ta）
*/

var my_ta:mx.controls.TextArea;

// 创建一个 Number 变量来跟踪 TextArea 的更改次数。
var changeCount_num:Number = 0;

// 定义侦听器对象。
var taListener:Object = new Object();
// 定义一个函数，只要侦听器接收到 TextArea 组件发生更改的通知，
// 该函数就会执行。
taListener.change = function(evt_obj:Object) {
 changeCount_num++;
 trace("Text has changed " + changeCount_num + " times now!");
 trace("It now contains: " + evt_obj.target.text);
}
```

```
 trace("");
 };
 // 将侦听器对象注册到 TextArea 组件实例。
 my_ta.addEventListener("change", taListener);
```

### 另请参见

[EventDispatcher.addEventListener\(\)](#)

## TextArea.editable

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*textAreaInstance*.editable

### 说明

属性：一个布尔值，指示组件是 (true) 否 (false) 可编辑。默认值为 true。

### 示例

以下示例将 editable 属性设置为 false，以防止用户编辑它加载到 **TextArea** 实例（名称为 my\_ta）的文本。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.editable = false;

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 } else {
```

```
 my_ta.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

# TextArea.hPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*TextAreaInstance.hPosition*

## 说明

属性：定义文本在字段中的水平位置。默认值为 0。

## 示例

以下示例在用户在它加载到 **TextArea** 实例（名称为 my\_ta）的文本中前后滚动时，使用侦听器在“输出”面板中显示当前的水平位置。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;

var taListener:Object = new Object();
taListener.scroll = function(evt_obj:Object) {
 trace("hPosition = " + my_ta.hPosition);
}
my_ta.addEventListener("scroll", taListener);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
```

```
 my_ta.text = src;
my_ta.hPosition = 200;
 } else {
 my_ta.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

# TextArea.hScrollPolicy

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*TextAreaInstance.hScrollPolicy*

## 说明

属性；确定水平滚动条是始终显示 ("on")、从不显示 ("off") 还是根据字段大小自动显示 ("auto")。默认值为 "auto"。

## 示例

以下示例将关闭 hScrollPolicy 属性，以使 **TextArea** 实例没有滚动条。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;
my_ta.hScrollPolicy = "off";

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 } else {
```

```
 my_ta.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

# TextArea.html

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*TextAreaInstance.html*

## 说明

属性：一个布尔值，指示此文本区域的内容是 (true) 否 (false) 采用 **HTML** 格式。如果 `html` 属性为 `true`，则此文本区域的内容采用 **HTML** 格式。如果 `html` 为 `false`，则此文本区域为非 **HTML** 文本区域。默认值为 `false`。

## 示例

以下示例使名为 `my_ta` 的 **TextArea** 成为 **HTML** 文本区域，然后使用 **HTML** 标签设置文本格式。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 **my\_ta**，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.html = true;
my_ta.text = "The Royal Nonesuch";
```

# TextArea.length

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textAreaInstance*.length

## 说明

属性（只读）：指示文本区域中的字符数。此属性与 **ActionScript** `text.length` 属性都返回相同的值，但速度更快。制表符 ("`\t`") 这样的字符会被算作一个字符。默认值为 **0**。

## 示例

以下示例访问 **length** 属性以显示用户在名为 `my_ta` 的 **TextArea** 中键入的字符数。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 `my_ta`，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

// 定义侦听器对象。
var taListener:Object = new Object();
taListener.change = function(evt_obj:Object) {
 trace("my_ta.length is now: " + my_ta.length + " characters");
};
my_ta.addEventListener("change", taListener);
```



# TextArea.maxChars

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textAreaInstance*.maxChars

## 说明

属性：此文本区域最多可容纳的字符数。脚本插入的文本可能会比 maxChars 属性允许的字符数多；该属性只是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量没有限制。默认值为 null。

## 示例

以下示例设置 maxchars 属性以将用户可以输入的字符数限制为 24 个。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 **my\_ta**，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.maxChars = 24;

// 定义侦听器对象。
var taListener:Object = new Object();
taListener.change = function(evt_obj:Object) {
 trace("my_ta.length is now: " + my_ta.length + " characters");
};
my_ta.addEventListener("change", taListener);
```

# TextArea.maxHPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textAreaInstance*.maxHPosition

## 说明

只读； [TextArea.hPosition](#) 的最大值。默认值为 0。

## 示例

以下示例访问 maxHPosition 属性，以将 **TextArea**（名称为 my\_ta）的初始位置设置为最右边。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;

var taListener:Object = new Object();
taListener.scroll = function(evt_obj:Object) {
 trace("hPosition = " + my_ta.hPosition);
}
my_ta.addEventListener("scroll", taListener);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 my_ta.hPosition = my_ta.maxHPosition;
 } else {
 my_ta.text = "Error loading text.";
 }
}
```

```
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

另请参见

[TextArea.vPosition](#)

## TextArea.maxVPosition

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*textAreaInstance*.maxVPosition

说明

只读属性；指明 [TextArea.vPosition](#) 的最大值。默认值为 0。

示例

以下示例访问 `maxVPosition` 属性，以将名为 `my_ta` 的 **TextArea** 的初始垂直位置设置为底部。它还在用户向上和向下滚动时跟踪当前的垂直位置。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 `my_ta`，然后将以下代码添加到第 1 帧中。

```
/**
 要求：
 - 舞台上的 TextArea 实例（实例名：my_ta）
*/

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = true;

var taListener:Object = new Object();
taListener.scroll = function(evt_obj:Object) {
 trace("vPosition = " + my_ta.vPosition);
}
my_ta.addEventListener("scroll", taListener);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
```

```

my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 my_ta.vPosition = my_ta.maxVPosition;
 } else {
 my_ta.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

```

另请参见

[TextArea.hPosition](#)

# TextArea.password

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*textAreaInstance.password*

说明

属性：一个布尔值，指示文本区域是 (true) 否 (false) 为密码字段。如果 password 为 true，则此文本区域为密码文本区域，并且会用星号隐藏输入字符。如果 password 为 false，则此文本区域不是密码文本区域。默认值为 false。

示例

以下示例在选中复选框（名称为 my\_ch）时，将 **TextArea**（名称为 my\_ta）中的文本作为密码字段处理；否则作为普通文本处理。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，再添加一个复选框并将其命名为 my\_ch，然后将以下代码添加到第 1 帧中。

```

/**
 要求：
 - 舞台上的 TextArea 实例（实例名: my_ta）
 - 舞台上的 CheckBox 实例（实例名: my_ch）
*/
var my_ta:mx.controls.TextArea;
var my_ch:mx.controls.CheckBox;

my_ta.wordWrap = false;

```

```
my_ta.password = true;
my_ch.selected = my_ta.password;

var chListener:Object = new Object();
chListener.click = function(evt_obj:Object) {
 my_ta.password = my_ch.selected;
}
my_ch.addEventListener("click", chListener);
```

## TextArea.restrict

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*textAreaInstance.restrict*

### 说明

属性；指示用户可输入文本区域中的字符集。默认值为 `undefined`。如果此属性为 `null`，则用户可以输入任何字符。如果此属性为空字符串，则不能输入任何字符。如果此属性为一个字符串，则用户只能输入该字符串中的字符；该字符串是从左向右扫描的。可以使用短划线 (-) 指定一个范围。

如果字符串以 ^ 开头，则 ^ 后跟随的所有字符都视为不可接受的字符。如果字符串不以 ^ 开头，则该字符串中的字符都将视为可接受。^ 也可以用作可接受和不可接受字符之间的切换标记。

例如，下面的代码允许除 X 和 Q 之外的 A-Z 字符：

```
Ta.restrict = "A-Z^XQ";
```

将输入内容限制为大写字符会将输入的小写字母字符转换为大写。同样地，将输入内容限制为小写字符会将输入的大写字母字符转换为小写。

`restrict` 属性只限制用户交互；脚本可将任何文本放入文本区域中。此属性不与“属性”检查器中的“嵌入字体轮廓”复选框同步。

## 示例

以下示例首先将 **restrict** 属性设置为将文本区域限制为大写字母、数字和空格，然后将其设置为允许使用除小写字母外的所有字符。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 **my\_ta**，然后将以下代码添加到第 1 帧中。一次只能使用一个 **restrict** 属性设置。

```
/**
 * 要求:
 * - 舞台上的 TextArea 实例 (实例名: my_ta)
 */
var my_ta:mx.controls.TextArea;

my_ta.wordWrap = true;

// 将控件限定为大写字母、数字和空格。
my_ta.restrict = "A-Z 0-9";

// 允许将除小写字母之外的所有
// 字符转换为大写
my_ta.restrict = "^a-z";
```

# TextArea.scroll

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
 // ...
};
textAreaInstance.addEventListener("scroll", listenerObject);
```

用法 2:

```
on (scroll) {
 // ...
}
```

## 说明

事件：在滚动条上单击（松开）鼠标时向所有已注册的侦听器广播。在广播此事件前，会更新 `UIScrollBar.scrollPosition` 属性和滚动条的屏幕上图像。

第一个用法示例使用了一个调度程序 / 侦听器事件模型，在其中，脚本放置在时间轴中包含组件实例的帧上。组件实例 (`textAreaInstance`) 调度一个事件（在本例中为 `scroll`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件发生时，就会调用该方法。该事件发生时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()`（请参见 `EventDispatcher.addEventListener()`），以便将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

除了事件对象的常规属性（`type` 和 `target`）外，`scroll` 事件的事件对象还包含一个名为 `direction` 的第三个属性。`direction` 属性包含一个字符串，用于描述滚动条的方向。`direction` 属性的可能值为 `vertical`（默认）和 `horizontal`。

有关 `type` 和 `target` 事件对象属性的更多信息，请参见第 461 页的“事件对象”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到 `TextArea` 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 `TextArea` 组件实例 `myTextAreaComponent`，它将“\_level0.myTextAreaComponent”发送到“输出”面板：

```
on (scroll) {
 trace(this);
}
```

## 示例

此示例使用调度程序 / 侦听器事件模型来跟踪何时用户使用 `TextArea` 实例的滚动条或滚动块来滚动 `TextArea`。

首先将 `TextArea` 组件的实例添加到舞台上并将其命名为 **my\_ta**，然后将以下代码添加到第 1 帧中：

```
/**
 要求：
 - 舞台上的 TextArea 实例（实例名：my_ta）
*/

my_ta.setSize(320, 240);
my_ta.move(10, 10);

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function(src:String):Void {
 my_ta.text = src;
}
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

```
my_ta.addEventListener("scroll", doScroll);
function doScroll(evt_obj:Object):Void {
 trace("target: " + evt_obj.target);
 trace("type: " + evt_obj.type);
 trace("direction: " + evt_obj.direction);
 trace("position: " + evt_obj.position);
 trace("");
}
```

另请参见

[EventDispatcher.addEventListener\(\)](#)

## TextArea.styleSheet

可用性

Flash Player 7。

用法

```
textAreaInstance.styleSheet = TextFieldStyleSheetObject
```

说明

属性；将一个样式表附加到由 *TextAreaInstance* 指定的 **TextArea** 组件。有关创建样式表的信息，请参见《学习 Flash 中的 ActionScript 2.0》中的“使用层叠样式表设置文本格式”。

与 **TextArea** 组件关联的样式表可以随时更改。如果使用中的样式表发生更改，则会用此新样式表重绘 **TextArea** 组件。可以将样式表设置为 `null` 或 `undefined` 来删除样式表。如果使用中的样式表被删除，则不使用任何样式表重绘 **TextArea** 组件。如果样式表被删除，则该样式表进行的格式设置将不保留。

示例

以下代码用 `new TextField.StyleSheet` 构造函数创建一个名为 `my_styles` 的新 **StyleSheet** 对象，然后定义了 `html` 和 `body` 标签的样式。接下来，它通过将 `my_styles` 分配给 **TextArea** 实例 `my_ta` 的 `styleSheet` 属性，来应用该样式。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 `my_ta`，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求:
 * - 舞台上的 TextArea 实例（实例名: my_ta）
 */

var my_ta:mx.controls.TextArea;
```



```

my_ta.setSize(320, 240);

// 创建新的 StyleSheet 对象。
var my_styles:TextField.StyleSheet = new TextField.StyleSheet();
my_styles.setStyle("html", {fontFamily:"Arial,Helvetica,sans-serif",
 fontSize:"12px", color:"#0000FF"});
my_styles.setStyle("body", {color:"#00CCFF", textDecoration:"underline"});

// 将 TextAreaInstance.styleSheet 属性设置为新定义的、
// 名为 styles 的 styleSheet 对象。
my_ta.styleSheet = my_styles;
my_ta.html = true;

// 加载要显示的文本，并定义 onLoad 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 } else {
 my_ta.text = "Error loading HTML document.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

```

# TextArea.text

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textAreaInstance.text*

## 说明

属性： **TextArea** 组件的文本内容。默认值为 ""（空字符串）。

## 示例

以下示例在 my\_ta TextArea 实例的 text 属性中放置一个字符串，然后在“输出”面板输出该字符串。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，然后将以下代码添加到第 1 帧中。

```
/**
```

```
要求：
- 舞台上的 TextArea 实例（实例名：my_ta）
*/

var my_ta:mx.controls.TextArea;

my_ta.text = "The Royal Nonesuch";
trace(my_ta.text); // 输出结果：“The Royal Nonesuch”
```

## TextArea.vPosition

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*textAreaInstance.vPosition*

### 说明

属性；定义文本在文本区域中的垂直滚动位置。该属性很有用，可以帮助用户定位到长篇文章的特定段落，还可用于创建滚动文本区域。您可以获取并设置该属性。默认值为 **0**。

### 示例

以下示例将文本加载到 **TextArea**（名称为 my\_ta）中，然后设置 **vPosition** 属性以便在底部显示该文本。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 my\_ta，然后将以下代码添加到第 1 帧中。

```
/**
要求：
- 舞台上的 TextArea 实例（实例名：my_ta）
*/

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 my_ta.vPosition = my_ta.maxVPosition;
 }
}
```

```
 } else {
 trace("Error loading text.");
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt")
```

# TextArea.vScrollPolicy

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textAreaInstance.vScrollPolicy*

## 说明

属性；确定垂直滚动条是始终显示 ("on")、从不显示 ("off") 还是根据字段大小自动显示 ("auto")。默认值为 "auto"。

## 示例

以下示例关闭 **TextArea**（名称为 **my\_ta**）的垂直滚动条，因此没有可用的滚动条来滚动示例载入的文本。

必须首先将 **TextArea** 组件的实例添加到舞台上并将其命名为 **my\_ta**，然后将以下代码添加到第 1 帧中。

```
/**
 要求:
 - 舞台上的 TextArea 实例（实例名: my_ta）
*/
var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = true;
my_ta.vScrollPolicy = "off";
```

```
// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 } else {
 my_ta.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

## TextArea.wordWrap

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*textAreaInstance*.wordWrap

### 说明

属性：一个布尔值，指示文本是 (true) 否 (false) 自动换行。默认值为 true。



如果您使用 `createClassObject()` 方法来创建 `TextArea` 实例，则 `wordWrap` 的默认值为 `false`。

### 示例

以下示例将 `TextArea`（名称为 `my_ta`）的 `wordwrap` 属性设置为 `false`，使其有一个水平滚动条来访问两侧边界外的文本。

必须首先将 `TextArea` 组件的实例添加到舞台上并将其命名为 **my\_ta**，然后将以下代码添加到第 1 帧中。

```
/**
 * 要求：
 * - 舞台上的 TextArea 实例（实例名：my_ta）
 */

var my_ta:mx.controls.TextArea;

my_ta.setSize(320, 240);
my_ta.wordWrap = false;
//my_ta.vScrollPolicy = "off";
```

```
// 加载要显示的文本, 并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_ta.text = src;
 } else {
 my_ta.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```



## TextInput 组件

**TextInput** 组件是单行文本组件，该组件是本机 **ActionScript TextField** 对象的包装。您可以使用样式自定义 **TextInput** 组件；当实例被禁用时，它的内容会显示为 **disabledColor** 样式表示的颜色。**TextInput** 组件也可以采用 **HTML** 格式，或作为掩饰文本的密码字段。

在应用程序中，**TextInput** 组件可以被启用或者禁用。在禁用状态下，它不接收鼠标或键盘输入。当启用时，它遵循与 **ActionScript TextField** 对象相同的焦点、选择和导航规则。当一个 **TextInput** 实例有焦点时，您还可以使用以下按键来控制它：

键	说明
箭头键	将插入点向左或向右移动一个字符。
Shift+Tab	将焦点移到前一个对象。
Tab	将焦点移到下一个对象。

有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

每个 **TextInput** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的参数所做的更改。在实时预览中，文本是不可选定的，并且无法在舞台上的组件实例中输入文本。

在将 **TextInput** 组件添加到应用程序时，您可以使用“辅助功能”面板来使其可由屏幕读取器访问。

# 使用 TextInput 组件

在任何需要单行文本字段的地方，都可以使用 `TextInput` 组件。如果您需要多行文本字段，请使用 [TextArea 组件](#)。例如，您可以在表单中将 `TextInput` 组件用作密码字段。您也可以设置一个侦听器，检查用户按 `Tab` 键切换到字段之外时，该字段是否有足够的字符。该侦听器可以显示一条错误信息，指明必须输入正确的字符数。

## TextInput 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 `TextInput` 组件实例设置以下创作参数：

**editable** 指示 `TextInput` 组件是 (true) 否 (false) 可编辑。默认值为 `true`。

**password** 指示字段是 (true) 否 (false) 为密码字段。默认值为 `false`。

**text** 指定 `TextInput` 组件的内容。您无法在“属性”检查器或“组件”检查器中输入回车。默认值为 `""`（空字符串）。

您可以在“属性”检查器中（“窗口” > “组件检查器”）为每个 `TextInput` 组件实例设置以下附加参数：

**maxChars** 是文本输入字段最多可以容纳的字符数。默认值为 `null`（表示无限制）。

**restrict** 指示用户可输入到文本输入字段中的字符集。默认值为 `undefined`。请参见[第 1131 页的“`TextInput.restrict`”](#)。

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 `true`。

**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 `true`。

提醒

`minHeight` 和 `minWidth` 属性由内部的大小调整例程使用。这两个属性在 `UIObject` 中定义，并可以根据需要被不同的组件覆盖。在为应用程序创建自定义布局管理器时，可以使用这两个属性。否则，在“组件”检查器中设置这两个属性会不具有明显效果。

您可以编写 `ActionScript`，以便使用 `TextInput` 组件的属性、方法和事件来控制该组件的这些和其它选项。有关更多信息，请参见[第 1119 页的“`TextInput` 类”](#)。



## 创建具有 TextInput 组件的应用程序

以下过程解释了如何在创作时将 **TextInput** 组件添加到应用程序。在本示例中，组件是带有事件侦听器的密码字段，事件侦听器确定输入的字符数是否正确。

### 创建具有 TextInput 组件的应用程序：

1. 将 **TextInput** 组件从“组件”面板拖到舞台上。
2. 在“属性”检查器中，执行以下操作：
  - 输入实例名称 **my\_ti**。
  - 将 **text** 参数保留为空。
  - 将 **editable** 参数设置为 **true**。
3. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
/**
 要求：
 - 舞台上有 TextInput 实例（实例名称：my_ti）
 */

var my_ti:mx.controls.TextInput;

// 创建侦听器对象。
var tiListener:Object = new Object();
tiListener.handleEvent = function (evt_obj:Object){
 if (evt_obj.type == "enter"){
 if (my_ti.length < 8) {
 trace("You must enter at least 8 characters");
 } else {
 trace("Thanks");
 }
 }
}
// 添加侦听器。
my_ti.addEventListener("enter", tiListener);
```

此代码在名为 **my\_ti** 的 **TextInput** 实例上设置 **enter** 事件处理函数。如果用户输入少于 8 个字符，则该示例显示消息：您必须至少输入 8 个字符。如果用户输入 8 个或更多字符，该示例显示：Thanks。

4. 在 **my\_ti** 实例中输入文本后，您可以获取它的值，如下所示：

```
var my_text:String = my_ti.text;
```

# 自定义 TextInput 组件

在创作过程中和运行时，您都可以在水平方向上改变 `TextInput` 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `UIObject.setSize()` 或 `TextInput` 类的任何适用的属性和方法。

在调整 `TextInput` 组件大小时，边框将相应调整为新边框。`TextInput` 组件不使用滚动条，但当用户与文本交互操作时插入点会自动滚动。然后在剩余区域中调整文本字段大小，在 `TextInput` 组件中没有固定大小的元素。如果 `TextInput` 组件太小而无法显示文本，则该文本将会被裁剪。

## 对 TextInput 组件使用样式

`TextInput` 组件在类样式声明中定义了它的 `backgroundColor` 和 `borderStyle` 样式属性。类样式覆盖全局样式，因此，如果要设置 `backgroundColor` 和 `borderStyle` 样式属性，必须在实例上创建或定义一个不同的自定义样式声明。

`TextInput` 组件支持以下样式：

样式	主题	说明
<code>backgroundColor</code>	光晕和范例	背景色。默认颜色为白色。
<code>borderStyle</code>	光晕和范例	<code>TextInput</code> 组件使用 <code>RectBorder</code> 实例作为其边框并对在该类上定义的样式做出响应。请参见第 985 页的“ <a href="#">RectBorder 类</a> ”。  默认边框样式为 <code>"inset"</code> 。
<code>marginLeft</code>	光晕和范例	表示文本左边距的数字。默认值为 0。
<code>marginRight</code>	光晕和范例	表示文本右边距的数字。默认值为 0。
<code>color</code>	光晕和范例	文本颜色。“光晕”主题的默认值为 <code>0x0B333C</code> ，“范例”主题的默认值为空白。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>0x848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	文本的字体名称。默认值为 <code>"_sans"</code> 。
<code>fontSize</code>	光晕和范例	字体的磅值。默认值为 10。
<code>fontStyle</code>	光晕和范例	字体样式： <code>"normal"</code> 或 <code>"italic"</code> 。默认值为 <code>"normal"</code> 。

样式	主题	说明
fontWeight	光晕和范例	字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textAlign	光晕和范例	文本对齐方式: "left"、"right" 或 "center"。默认值为 "left"。
textIndent	光晕和范例	表示文本缩进的数字。默认值为 0。
textDecoration	光晕和范例	文本修饰: "none" 或 "underline"。默认值为 "none"。

`TextArea` 和 `TextInput` 组件使用相同的样式, 并且通常使用方式也相同。因此, 默认情况下它们共享同一个类级别样式声明。例如, 下面的代码在 `TextArea` 声明时设置样式, 但它同时影响 `TextArea` 和 `TextInput` 组件。

```
_global.styles.TextArea.setStyle("disabledColor", 0xBBBBFF);
```

要分离这两个组件并且只为其中一个提供类级别样式, 请创建新的样式声明。

```
import mx.styles.CSSStyleDeclaration;
_global.styles.TextInput = new CSSStyleDeclaration();
_global.styles.TextInput.setStyle("disabledColor", 0xFFBBBB);
```

请注意, 此示例在覆盖 `_global.styles.TextInput` 之前, 不检查它是否已经存在; 在此示例中, 您知道它的存在并要将其覆盖。

## 使用具有 `TextInput` 组件的外观

`TextArea` 组件使用 `RectBorder` 的实例作为边框。有关设置这些可视元素的外观的更多信息, 请参见第 985 页的“[RectBorder 类](#)”。

# TextInput 类

继承 `MovieClip` > [UIObject 类](#) > [UIComponent 类](#) > `TextInput`

ActionScript 类名称 `mx.controls.TextInput`

使用 `TextInput` 类的属性, 您可以在运行时设置文本内容、格式和水平位置。您也可以指明该字段是否可编辑, 以及该字段是否为“密码”字段。您还可以限制用户可以输入的字符。

使用 ActionScript 设置 `TextInput` 类的属性将会覆盖在“属性”检查器或“组件”检查器中设置的同名参数。

`TextInput` 组件使用焦点管理器覆盖默认的 Flash Player 焦点矩形, 并绘制一个带圆角的自定义焦点矩形。有关更多信息, 请参见第 663 页的“[FocusManager 类](#)”。

`TextInput` 组件支持 CSS 样式和 Flash Player 支持的任何其它 HTML 样式。有关 CSS 支持的信息, 请参见 W3C 规范, 网址为 [www.w3.org/TR/REC-CSS2/](http://www.w3.org/TR/REC-CSS2/)。

您可以通过使用由文本对象返回的字符串来操作文本字符串。

每个组件类都有一个 `version` 属性，该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.TextInput.version);
```

提醒

代码 `trace(myTextInputInstance.version)`；返回 `undefined`。

## TextInput 类的方法摘要

没有 `TextInput` 类专用的方法。

### 从 `UIObject` 类继承的方法

下表列出了 `TextInput` 类从 `UIObject` 类继承的方法。从 `TextInput` 对象调用这些方法时，请使用 `TextInputInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 `TextInput` 类从 `UIComponent` 类继承的方法。从 `TextInput` 对象调用这些方法时，请使用 `TextInputInstance.methodName` 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## TextInput 类的属性摘要

下表列出了 `TextInput` 类的属性。

属性	说明
<code>TextInput.editable</code>	一个布尔值，它指示该字段是 (true) 否 (false) 可编辑。
<code>TextInput.hPosition</code>	文本在字段中的水平滚动位置。
<code>TextInput.length</code>	只读； <code>TextInput</code> 组件中的字符数。
<code>TextInput.maxChars</code>	用户可以在文本字段中输入的最大字符数。
<code>TextInput.maxHPosition</code>	只读； <code>TextField.hPosition</code> 可能具有的最大值。
<code>TextInput.password</code>	一个布尔值，它指示该文本字段是否为隐藏所输入字符的密码字段。
<code>TextInput.restrict</code>	指明用户可以在文本字段输入哪些字符。
<code>TextInput.text</code>	设置 <code>TextInput</code> 组件的文本内容。

## 从 UIObject 类继承的属性

下表列出了 `TextInput` 类从 `UIObject` 类继承的属性。从 `TextInput` 对象访问这些属性时，请使用 `TextInputInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	只读；对象的底边缘位置（相对于其父对象的底边缘）。
<code>UIObject.height</code>	只读；对象的高度，以像素为单位。
<code>UIObject.left</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.right</code>	只读；对象的右边缘位置（相对于其父对象的右边缘）。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	只读；对象上边缘的位置（相对于其父对象）。

属性	说明
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

## 从 UIComponent 类继承的属性

下表列出了 `TextInput` 类从 `UIComponent` 类继承的属性。从 `TextInput` 对象访问这些属性时，请使用 `TextInputInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指示组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指示文档中组件的 Tab 键顺序。

## TextInput 类的事件摘要

下表列出了 `TextInput` 类的事件。

事件	说明
<code>TextInput.change</code>	当 <code>TextInput</code> 字段更改时广播。
<code>TextInput.enter</code>	当按下 Enter 键时广播。

## 从 UIObject 类继承的事件

下表列出了 `TextInput` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

# 从 UIComponent 类继承的事件

下表列出了 `TextInput` 类从 `UIComponent` 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

# TextInput.change

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.change = function(eventObject:Object) {
 //...
};
textInputInstance.addEventListener("change", listenerObject)
```

用法 2:

```
on (change){
 //...
}
```

## 说明

事件：通知侦听器文本已更改。在文本更改后广播该事件。不能使用该事件防止某些字符添加到组件的文本字段；为实现该目的，应使用 `TextInput.restrict`。该事件是通过用户输入触发的，不能通过编程方式的更改来触发。

第一个用法示例使用一个 `on()` 处理函数，并且必须直接附加到一个 `TextInput` 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，如果将以下代码附加到 `myTextInput` 实例，它会将 “\_level0.myTextInput” 发送到 “输出” 面板：

```
on (change){
 trace(this);
}
```

第二个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*textInputInstance*) 调度一个事件 (在本例中为 *change*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作 “处理函数”) 处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的 “[EventDispatcher 类](#)”。

### 示例

此示例为 `my_ti TextInput` 实例上的 `change` 事件创建侦听器。当发生 `change` 事件时，该示例显示 “Input has changed”。

首先必须将一个 `TextInput` 组件拖到舞台上，并为其指定实例名称 `my_ti`，然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有一个 TextInput 实例（实例名称：my_ti）
 */

var my_ti:mx.controls.TextInput;

// 创建侦听器对象。
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
 trace("Input has changed");
};
// 添加侦听器。
my_ti.addEventListener("change", tiListener);
```

### 另请参见

[EventDispatcher.addEventListener\(\)](#)



# TextInput.editable

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textInputInstance*.editable

## 说明

属性；一个布尔值，指明组件是 (true) 否 (false) 可编辑。默认值为 true。

## 示例

此示例针对 **my\_ti TextInput** 实例将 `editable` 属性的值设置为 `false`。这会阻止用户在实例中输入文本。您可以将该属性设置为 `true`，以便使 **TextInput** 实例可编辑。

首先必须将一个 **TextInput** 组件拖到舞台上，并为其指定实例名称 `my_ti`，然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有 TextInput 实例（实例名称: my_ti）
 */

var my_ti:mx.controls.TextInput;

my_ti.editable = false;
```

# TextInput.enter

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.enter = function(eventObject:Object) {
 //...
```

```
};
textInputInstance.addEventListener("enter", listenerObject);
```

用法 2:

```
on (enter) {
 //...
}
```

## 说明

事件：通知侦听器 **Enter** 键已被按下。

第一个用法示例使用一个 `on()` 处理函数，并且必须直接附加到一个 **TextInput** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，如果将以下代码附加到 `myTextInput` 实例，它会将 “\_level0.myTextInput” 发送到 “输出” 面板：

```
on (enter){
 trace(this);
}
```

第二个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (`textInputInstance`) 调度一个事件（在本例中为 `enter`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的 “[EventDispatcher 类](#)”。

## 示例

此示例为名为 `my_ti` 的 **TextInput** 实例上的输入事件创建侦听器。当发生该输入事件时，如果用户输入的字符数少于 8 个，该示例显示：您必须至少输入 8 个字符。否则显示 Thanks！

首先必须将一个 **TextInput** 组件拖到舞台上，并为其指定实例名称 `my_ti`，然后将以下代码添加到第一帧中。

```
/**
 要求：
 - 舞台上有 TextInput 实例（实例名称：my_ti）
*/

var my_ti:mx.controls.TextInput;

// 创建侦听器对象。
var tiListener:Object = new Object();
tiListener.handleEvent = function (evt_obj:Object){
 if (evt_obj.type == "enter"){
```

```

 if (my_ti.length < 8) {
 trace("You must enter at least 8 characters");
 } else {
 trace("Thanks");
 }
 }
}
// 添加侦听器。
my_ti.addEventListener("enter", tiListener);

```

另请参见

[EventDispatcher.addEventListener\(\)](#)

## TextInput.hPosition

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

*textInputInstance.hPosition*

说明

属性；指定已滚动过多少像素以容纳用户在 **TextInput** 框中的输入。默认值为 0。



由于监视器类型、屏幕大小和字体特征不同，对于同一文本该值也会发生变化。

示例

以下示例为名为 **my\_ti** 的 **TextInput** 实例上的 **change** 事件创建侦听器。该侦听器访问 **hPosition** 属性以显示用户输入的每个字符的当前位置。

首先必须将一个 **TextInput** 组件拖到舞台上，并为其指定实例名称 **my\_ti**，然后将以下代码添加到第一帧中。

```

/**
 * 要求：
 * - 舞台上有一个 TextInput 实例（实例名称：my_ti）
 */

var my_ti:mx.controls.TextInput;

```

```
// 创建侦听器对象。
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
 trace("hPosition = " + my_ti.hPosition);
};
// 添加侦听器。
my_ti.addEventListener("change", tiListener);
```

# TextInput.length

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textInputInstance.length*

## 说明

只读属性；一个数字，指明 **TextInput** 组件中的字符数。制表符 ("\t") 这样的字符会被算作一个字符。默认值为 0。

## 示例

以下示例为 **TextInput** 实例 `my_ti` 上的 **change** 事件创建侦听器。该侦听器访问 `length` 属性以便在用户输入文本的过程中显示 `my_ti` 中的文本的长度。

首先必须将一个 **TextInput** 组件拖到舞台上，并为其指定实例名称 `my_ti`，然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有 TextInput 实例（实例名称: my_ti）
 */

var my_ti:mx.controls.TextInput;

// 创建侦听器对象。
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
 trace("Length of text: " + my_ti.length);
};
// 添加侦听器。
my_ti.addEventListener("change", tiListener);
```

# TextInput.maxChars

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textInputInstance*.maxChars

## 说明

属性：该文本字段最多可容纳的字符数。脚本插入的文本可能会比 maxChars 属性允许的字符数多；该属性只是指示用户可以输入多少文本。如果此属性为 null，则对用户可以输入的文本量没有限制。默认值为 null。

## 示例

以下示例将用户在名为 my\_ti 的 **TextInput** 实例中输入的字符数限制为 8 个以内。它还设置 password 属性，该属性通过显示星号来取代输入的字符，将输入的字符隐藏起来。

首先必须将一个 **TextInput** 组件拖到舞台上，并为其指定实例名称 my\_ti，然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有一个 TextInput 实例（实例名称：my_ti）
 */

var my_ti:mx.controls.TextInput;

my_ti.maxChars = 8;
my_ti.password = true;
```

# TextInput.maxHPosition

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textInputInstance*.maxHPosition

## 说明

只读属性： `maxHPosition` 的值为指针移至文本最右侧时所能看见的字符的像素位置。它不是最后一个字符的像素位置，而是 `TextInput` 字段中最后一个字符右侧的像素位置。默认值为 `0`。

首先必须将一个 `TextInput` 组件拖到舞台上，并为其指定实例名称 `my_ti`，然后将以下代码添加到第一帧中。

## 示例

以下代码为 `my_ti` `TextInput` 实例上的 `change` 事件创建一个侦听器。当 `change` 事件发生时，该侦听器将显示用户输入的每个字符的当前 `hPosition` 值和 `maxHPosition` 值：

```
/**
 * 要求：
 * - 舞台上有 TextInput 实例（实例名称：my_ti）
 */

var my_ti:mx.controls.TextInput;

// 创建侦听器对象。
var tiListener:Object = new Object();
tiListener.change = function(evt_obj:Object) {
 trace("hPosition: " + my_ti.hPosition + " of " + my_ti.maxHPosition);
};
// 添加侦听器。
my_ti.addEventListener("change", tiListener);
```

# TextInput.password

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`textInputInstance.password`

## 说明

属性：一个布尔值，指明文本字段是 (`true`) 否 (`false`) 为密码字段。如果此属性为 `true`，则此文本字段为密码文本字段并且会隐藏输入字符。如果此属性为 `false`，则此文本字段不是密码文本字段。默认值为 `false`。

## 示例

以下示例设置 `password` 属性, 以便在用户在名为 `my_ti` 的 **TextInput** 实例中输入时会显示星号来代替输入的字符。它还设置 `maxChars` , 以便将用户可以输入的字符数限制在 8 个以内。

首先必须将一个 **TextInput** 组件拖到舞台上, 并为其指定实例名称 `my_ti`, 然后将以下代码添加到第一帧中。

```
/**
 * 要求:
 * - 舞台上有一个 TextInput 实例 (实例名称: my_ti)
 */

var my_ti:mx.controls.TextInput;

my_ti.maxChars = 8;
my_ti.password = true;
```

# TextInput.restrict

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textInputInstance.restrict*

## 说明

属性; 指示用户可输入到文本字段中的字符集。默认值为 `undefined`。如果此属性为 `null` 或空字符串 (`"`), 则用户可以输入任何字符。如果此属性为一个字符串, 则用户只能输入该字符串中的字符; 该字符串是从左向右扫描的。可以使用短划线 (`-`) 指定一个范围。

如果字符串以 `^` 开头, 则 `^` 后跟随的所有字符都视为不可接受的字符。如果字符串不以 `^` 开头, 则该字符串中的字符都将视为可接受。 `^` 也可以用作可接受和不可接受字符之间的切换标记。

例如, 下面的代码允许除 **X** 和 **Q** 之外的 **A-Z** 字符:

```
Ta.restrict = "A-Z^XQ";
```

可以使用反斜杠 (`\`) 来输入连字符 (`-`)、插入符号 (`^`) 或反斜杠 (`\`) 字符, 如下所示:

```
\^
\ -
\\
```

在“动作”面板中，当在双引号中输入 \ 字符时，对于“动作”面板的双引号解释器，该字符有特殊的含义。它表示 \ 之后的字符应被视为其本身的含义。例如，可以使用下面的代码来输入一个单引号：

```
var leftQuote = "\'";
```

“动作”面板的 **restrict** 解释器也将 \ 用作转义符。因此，您可能会认为下列代码应该起作用：

```
myText.restrict = "0-9\-\^\\";
```

但是，因为此表达式包含在双引号内，所以会将下面的值发送到 **restrict** 解释器：0-9-^\，**restrict** 解释器将不能识别此值。

因为必须在双引号中输入此表达式，所以不仅要为 **restrict** 解释器提供表达式，而且还必须转义“动作”面板中双引号的内置解释器。若要将值 0-9\-\^\ 发送到 **restrict** 解释器，您必须输入下面的代码：

```
myText.restrict = "0-9\\-\\^\\\\\";
```

**restrict** 属性只限制用户交互；可使用脚本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。

## 示例

以下示例提供了 **restrict** 属性的 3 种不同用法。第一种用法将输入内容限制为大写字母 A 至 Z、空格和数字。第二种用法允许除小写字母 a 至 z 以外的任何字符。第三种用法只允许数字、-、^ 和 \。

您必须先将 **TextInput** 组件拖到舞台上，并赋予它实例名 **my\_ti**，然后将代码添加到 **Frame 1**，一次只使用以下一种 **restrict** 语句。

```
/**
 要求：
 - 舞台上有 TextInput 实例（实例名称：my_ti）
*/
```

```
var my_ti:mx.controls.TextInput;
```

```
// 示例 1：只允许大写 A-Z、空格和数字 0-9。
my_ti.restrict = "A-Z 0-9";
```

```
// 示例 2：除小写 a-z 外都允许。
my_ti.restrict = "^a-z";
```

```
// 示例 3：只允许数字 0-9、连字符、^ 和 \
my_ti.restrict = "0-9\\-\\^\\\\\";
```



# TextInput.text

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*textInputInstance.text*

## 说明

属性； **TextInput** 组件的文本内容。默认值为 ""（空字符串）。

## 示例

以下代码在名为 **my\_ti** 的 **TextInput** 实例中放置一个字符串，然后用 **trace** 命令将该字符串发送到“输出”面板。

首先必须将一个 **TextInput** 组件拖到舞台上，并为其指定实例名称 **my\_ti**，然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有一个 TextInput 实例（实例名称：my_ti）
 */

var my_ti:mx.controls.TextInput;

my_ti.text = "The Royal Nonesuch";
trace(my_ti.text); // "The Royal Nonesuch"
```



# TransferObject 接口

ActionScript 类名称 `mx.data.to.TransferObject`

`TransferObject` 接口定义了 `DataSet` 组件所管理的项目必须实现的一组方法。

`DataSet.itemClassName` 属性指定在每次需要新项时将实例化的传送对象类的名称。也可以使用“属性”检查器为所选的 `DataSet` 组件指定此属性。

## TransferObject 接口的方法摘要

下表列出了 `TransferObject` 接口的方法。

方法	说明
<code>TransferObject.clone()</code>	创建传送对象的新实例。
<code>TransferObject.getPropertyData()</code>	返回此传送对象的数据。
<code>TransferObject.setPropertyData()</code>	设置此传送对象的数据。

## TransferObject.clone()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004.

### 用法

```
class itemClass implements mx.data.to.TransferObject {
 function clone() {
 // 此处是您的代码。
 }
}
```

### 参数

无。

### 返回

传送对象的副本。

### 说明

方法；创建传送对象的实例。此方法的实现会创建现有传送对象及其属性的副本，然后返回该对象。

### 示例

下面的函数返回传送对象的副本，并将其所有属性设置为与原始值相同：

```
class itemClass implements mx.data.to.TransferObject {
 function clone():Object {
 var copy:itemClass = new itemClass();
 for (var p in this) {
 copy[p]= this[p];
 }
 return(copy);
 }
}
```

## TransferObject.getPropertyData()

### 可用性

Flash Player 7。

### 版本

Flash MX Professional 2004.

### 用法

```
class itemClass implements mx.data.to.TransferObject {
 function getPropertyData() {
 // 此处是您的代码。
 }
}
```

### 参数

无。

### 返回

一个对象。

## 说明

方法：返回此传送对象的数据。此方法的实现可能会返回具有属性和相应值的匿名 **ActionScript** 对象。

## 示例

以下函数返回一个名为 `internalData` 的对象，该对象包含 **Contact** 对象中的属性及属性值：

```
class Contact implements mx.data.to.TransferObject {
function getPropertyData():Object {
 var internalData:Object = {name:name, readOnly:_readOnly, phone:phone,
 zip:zip.zipPlus4};
 return(internalData);
}
```

# TransferObject.setPropertyData()

## 可用性

Flash Player 7。

## 版本

Flash MX 2004.

## 用法

```
class yourClass implements TransferObject {
 function setPropertyData(propData) {
 // 此处是您的代码。
 }
}
```

## 参数

*propData* 一个包含分配给此传送对象的数据的对象。

## 返回

无。

## 说明

方法：设置此传送对象的数据。*propData* 参数是一个对象，其字段包含 **DataSet** 组件分配给此传送对象的数据。

## 示例

以下函数接收一个 *propData* 参数，并将其属性值应用到 **Contact** 对象的属性：

```
class Contact implements mx.data.to.TransferObject {

 function setPropertyData(propData: Object):Void {
 _readOnly = propData.readOnly;
 phone = propData.phone;
 zip = new mx.data.types.ZipCode(data.zip);
 }

 public var name:String;
 public var phone:String;
 public var zip:ZipCode;
 private var _readOnly:Boolean; // 指示是否不可改变
}
```

# TransitionManager 类

**ActionScript 类名称** mx.transitions.TransitionManager

通过 **TransitionManager** 类和基于效果来定义过渡的类，您可以迅速将令人印象深刻的过渡动画效果应用于幻灯片和影片剪辑。

顾名思义，**TransitionManager** 类用于管理过渡。它允许您将十种动画效果中的一种应用于幻灯片和影片剪辑。在创建 **Macromedia Component Architecture** 第 2 版的自定义组件时，您可以使用 **TransitionManager** 将动画效果应用于组件可视界面中的影片剪辑。过渡效果在一组过渡类中定义，这些过渡类全都是对基类 `mx.transitions.Transition` 的扩展。您只是通过 **TransitionManager** 的实例应用过渡；而不直接实例化它们。由 **TransitionManager** 类实现动画事件。

## 使用 TransitionManager 类

若要使用 **TransitionManager** 类的方法和属性，您可以在两种方法中选择一种来创建新实例。最简单的方法是调用 `TransitionManager.start()` 方法，只须调用一次，就可以创建新的 **TransitionManager** 实例、指定目标对象、用一种缓动方法应用过渡并启动过渡效果。以下代码使用 `TransitionManager.start()` 方法：

```
mx.transitions.TransitionManager.start(myMovieClip_mc,
 {type:mx.transitions.Zoom, direction:mx.transitions.Transition.IN,
 duration:1, easing:mx.transitions.easing.Bounce.easeOut});
```

有关 `TransitionManager.start()` 方法及其用法和参数的更多信息，请参见第 1145 页的 `TransitionManager.start()`。

您还可以通过使用 `new` 运算符来创建 **TransitionManager** 类的新实例。然后可以指定过渡属性，并通过调用 `TransitionManager.startTransition()` 方法在另一步中启动过渡效果。以下代码使用 `TransitionManager.startTransition()` 方法：

```
var myTransitionManager:mx.transitions.TransitionManager = new
 mx.transitions.TransitionManager(myMovieClip_mc);
myTransitionManager.startTransition({type:mx.transitions.Zoom,
 direction:Transition.IN, duration:1,
 easing:mx.transitions.easing.Bounce.easeOut});
```

# TransitionManager 类参数

通过使用 `new` 运算符创建 `TransitionManager` 类的新实例时，您必须在其构造函数的 `content` 参数中指定一个目标影片剪辑。`mx.transitions.TransitionManager` 类的构造函数具有以下参数名称和类型：

`TransitionManager(content:MovieClip)`

`content` 是 `TransitionManager` 实例要向其应用过渡的影片剪辑对象。

碎  
碎

如果您通过使用 `new` 运算符创建一个 `TransitionManager` 实例，则必须指定要应用的过渡的属性，接着使用 `TransitionManager.startTransition()` 方法进行调用以启动过渡；否则，过渡将不应用于影片剪辑或不启动。有关 `TransitionManager.startTransition()` 方法及其用法和参数的详细信息，请参见第 1147 页的 `TransitionManager.startTransition()`。对于上述用于创建 `TransitionManager` 实例的由两个步骤组成的过程而言，有一个更快捷的替代方法，就是只调用 `TransitionManager.start()` 方法；有关更多信息，请参见第 1145 页的 `TransitionManager.start()`。通过 `TransitionManager.start()` 方法，只须调用一次，就可以创建 `TransitionManager` 实例、提供目标影片剪辑并指定过渡属性。

## 在过渡中指定缓动类和方法

在您通过使用 `TransitionManager.start()` 方法创建 `TransitionManager` 类的实例时，可使用 `transParam` 参数的缓动属性指定提供缓动计算的函数或方法。有关可用的缓动类和方法的完整说明，请参见第 1140 页的“在过渡中指定缓动类和方法”。

# TransitionManager 类摘要

以下各节列出了 `TransitionManager` 类的方法、属性和事件。

## TransitionManager 类的方法摘要

下表列出了 `TransitionManager` 类的方法。

方法	说明
<code>TransitionManager.start()</code>	创建新的 <code>TransitionManager</code> 实例，指定目标对象，应用过渡并启动该过渡。
<code>TransitionManager.startTransition()</code>	创建过渡实例，然后启动该实例。
<code>TransitionManager.toString()</code>	以字符串形式返回 <code>TransitionManager</code> 的类型。



# TransitionManager 类的属性摘要

下表列出了 TransitionManager 类的属性。

属性	说明
<code>TransitionManager.content</code>	TransitionManager 要将过渡应用到的影片剪辑实例。
<code>TransitionManager.contentAppearance</code>	一个对象，它包含内容（目标影片剪辑，过渡将应用到该内容）的已保存的可视属性。该属性为只读。

# TransitionManager 类的事件摘要

下表列出了 TransitionManager 类的事件。

事件	说明
<code>TransitionManager.allTransitionsInDone</code>	当 TransitionManager 实例用 <code>mx.transitions.Transition.IN</code> 的 <code>direction</code> 属性完成过渡时，由该实例广播。
<code>TransitionManager.allTransitionsOutDone</code>	当 TransitionManager 实例用 <code>mx.transitions.Transition.OUT</code> 的 <code>direction</code> 属性完成过渡时，由该实例广播。

# TransitionManager.allTransitionsInDone

可用性  
Flash Player 6 (6.0.79.0)。

版本  
Flash MX 2004。

用法

```
var listenerObject:Object = new Object();
listenerObject.allTransitionsInDone = function(eventObj:Object) {
 // ...
};
transitionManagerInstance.addEventListener("allTransitionsInDone",
 listenerObject);
```

## 说明

事件；通知侦听器，**TransitionManager** 实例已完成 **direction** 属性为 `mx.transitions.Transition.IN` 的所有过渡，并已从要应用的过渡列表中删除了这些过渡。

该用法示例使用一个调度程序或侦听器事件模型。**TransitionManager** 实例 (*transitionManagerInstance*) 调度一个事件（在本例中为 `allTransitionsInDone`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。在本例中，`allTransitionsInDone` 事件提供一个目标属性，该属性包含触发了该事件的 **TransitionManager** 实例，从而允许您在用于接收 `allTransitionsInDone` 事件的代码内使用该实例及其所有属性和方法。有关更多信息，请参见第 461 页的第 21 章“[EventDispatcher 类](#)”。

## 示例

以下代码指定一个对象来侦听 `allTransitionsInDone` 事件，并指定用作该事件的处理函数的方法。在调用该方法以处理该事件时，已完成了 **direction** 属性为 `mx.transitions.Transition.IN` 的过渡。

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Iris, direction:Transition.IN,
 duration:1, easing:None.easeNone, startPoint:5, shape:Iris.CIRCLE});

var myListener:Object = new Object();
myListener.allTransitionsInDone = function(eventObj:Object) {
 trace("allTransitionsInDone event occurred.");
};
myTransitionManager.addEventListener("allTransitionsInDone", myListener);
```

## 另请参见

[第 461 页的第 21 章“EventDispatcher 类”](#)

# TransitionManager.allTransitionsOutDone

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.allTransitionsOutDone = function(eventObj:Object) {
 // ...
};
transitionManagerInstance.addEventListener("allTransitionsOutDone",
 listenerObject);
```

## 说明

事件：通知侦听器，**TransitionManager** 实例已完成 **direction** 属性为“out”的所有过渡，并已从要应用的过渡列表中删除了这些过渡。

该用法示例使用一个调度程序或侦听器事件模型。**TransitionManager** 实例 (transitionManagerInstance) 调度一个事件（在本例中为 allTransitionsOutDone），而该事件由您创建的侦听器对象 (listenerObject) 上的函数（也称作“处理函数”）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (eventObject) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。在本例中，allTransitionsInDone 事件提供一个目标属性，该属性包含触发了该事件的 **TransitionManager** 实例，这样，您就可以在用于接收 allTransitionsInDone 事件的代码内使用该实例及其所有属性和方法。有关更多信息，请参见第 461 页的第 21 章“[EventDispatcher 类](#)”。

## 示例

以下代码指定一个对象来侦听 allTransitionsOutDone 事件，并指定用作该事件的处理函数的方法。在调用该方法以处理该事件时，已完成了 **direction** 属性为

mx.transitions.Transition.IN 的过渡。

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Iris, direction:Transition.OUT,
 duration:1, easing:None.easeNone,startPoint:5, shape:Iris.CIRCLE});

var myListener:Object = new Object();
myListener.allTransitionsOutDone = function(eventObj:Object) {
 trace("allTransitionsOutDone event occurred.");
};
```

```
};
myTransitionManager.addEventListener("allTransitionsOutDone", myListener);
```

另请参见

[第 461 页的第 21 章 “EventDispatcher 类”](#)

## TransitionManager.content

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

```
transitionManagerInstance.content
```

说明

属性： **TransitionManager** 要将某个过渡应用到的影片剪辑实例。

示例

以下示例会返回当前作为 **TransitionManager** 实例目标的影片剪辑对象：

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
var myMovieClip:MovieClip = myTransitionManager.content;
trace(myMovieClip._name);
```

## TransitionManager.contentAppearance

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX 2004。

用法

```
transitionManagerInstance.contentAppearance
```

## 说明

属性（只读）：一个对象，它包含应用过渡前 **TransitionManager** 实例的目标影片剪辑的属性的快照。此对象有助于获取有关过渡完成后预期影片剪辑返回到的属性值的信息。从 `TransitionManager.contentAppearance` 返回的对象包含目标影片剪辑的下列属性的相应原始设置记录：`_x`、`_y`、`_xscale`、`_yscale`、`_alpha`、`_rotation`、`_innerBounds`、`_outerBounds`、`_width`、`_height` 和 `colorTransform`。将保存这些属性，并且调用 `TransitionManager.start()` 或 `TransitionManager.startTransition()` 方法。

## 示例

以下示例调用 `TransitionManager.contentAppearance()` 以获取在应用过渡前 **TransitionManager** 对象的目标影片剪辑的原始属性设置：

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Zoom, direction:Transition.OUT,
 duration:3, easing:Bounce.easeOut});

var myMovieClip:MovieClip = myTransitionManager.content;
var myOriginalMovieClipProps:Object = myTransitionManager.contentAppearance;

for (var prop in myOriginalMovieClipProps) {
 trace(myMovieClip._name + "." +prop+" = "+myOriginalMovieClipProps[prop]);
}
```

# TransitionManager.start()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*transitionManagerInstance.start(content, transParams)*

## 参数

*content*，要应用过渡效果的 **MovieClip** 对象。

*transParams*，在对象内传递的参数集合。

*transParams* 对象应包含 *type* 参数，该参数（后面跟有 *direction*、*duration* 和 *easing* 参数）指示要应用的过渡效果类。此外，还必须包括该过渡效果类所必需的任何参数。例如，`mx.transitions.Iris` 过渡效果类需要附加的 *startPoint* 和 *shape* 参数。因此，除了每个过渡都需要的 *type*、*duration* 和 *easing* 参数外，您还应该添加 `mx.transitions.Iris` 效果所需要的 *startPoint* 和 *shape* 参数（添加到 *transParams* 对象）。以下代码将 *startPoint* 和 *shape* 参数添加到 `mx.transitions.Iris` 效果：

```
{type:mx.transitions.Iris, direction:mx.transitions.Transition.IN,
 duration:5, easing:mx.transitions.easing.Bounce.easeOut, startPoint:5,
 shape:mx.transitions.Iris.CIRCLE}
```

若要确认您在 *transParam* 对象的 *type* 参数中指定的过渡类效果的其它必需参数，请参见该过渡类的 **API**。例如，有关 **Blinds** 过渡类的更多信息，请参见第 1150 页的“遮帘过渡”。

### 提醒

*transParams* 对象的 *type* 参数应包括为其参数指定的类的完整类包名称，除非已通过使用 `import` 语句导入了它们。为了避免不得不为所有 *transParams* 参数集合都提供完整类包名称，请预先在代码中放置以下 `import` 语句，以导入所有 `mx.transitions` 类和所有 `mx.transitions.easing` 类：

```
import mx.transitions.*;
import mx.transitions.easing.*;
```

## 返回

指定 **TransitionManager** 实例要应用的过渡对象的实例。

## 说明

方法：创建 **TransitionManager** 类的一个实例（如果尚不存在该实例），创建 *transParams.type* 参数中指定的过渡类的实例，然后启动该过渡。过渡会应用于 *content* 参数中指定的幻灯片或影片剪辑。

## 示例

以下代码使用 `TransitionManager.start()` 方法创建 **TransitionManager** 的一个实例，并将 **Iris** 过渡分配给名为 `img1_mc` 的影片剪辑。`TransitionManager.start()` 方法包含两个参数。第一个参数 *content* 是过渡效果将应用到的 **MovieClip** 对象。

`TransitionManager.start()` 方法的第二个参数是 *transParam*，它包含保存参数集合的对象。包含参数集合的此对象首先用 *type* 参数（后面跟有 *direction*、*duration* 和 *easing* 参数）指定过渡效果的类型。*type*、*direction*、*duration* 和 *easing* 参数是所有 **TransitionManager** 效果的必需信息。跟在 *easing* 参数之后的是该过渡类型专门需要的所有参数。在以下示例中，**Iris** 过渡是过渡类型，并且 **Iris** 过渡需要 *startPoint* 和 *shape* 参数（有关 **Iris** 过渡参数的更多信息，请参见第 1152 页的“光圈过渡”）：

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Iris, direction:Transition.IN,
 duration:5, easing:Bounce.easeOut, startPoint:5, shape:Iris.CIRCLE});
```

# TransitionManager.startTransition()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*transitionManagerInstance.startTransition(transParams)*

## 参数

*transParams*，在对象内传递的参数组成的集合。

*transParams* 对象应包含 *type* 参数，该参数（后面跟有 *direction*、*duration* 和 *easing* 参数）指示要应用的过渡效果类。此外，还必须包括指定的过渡效果类所必需的任何参数。例如，*mx.transitions.Iris* 过渡效果类需要附加的 *startPoint* 和 *shape* 参数。因此，除了每个过渡都需要的 *type*、*duration* 和 *easing* 参数外，您还应该添加 *mx.transitions.Iris* 效果需要的 *startPoint* 和 *shape* 参数（添加到 *transParams* 对象）。以下代码将 *startPoint* 和 *shape* 参数添加到 *mx.transitions.Iris* 效果：

```
{type:mx.transitions.Iris, direction:mx.transitions.Transition.IN,
 duration:5, easing:mx.transitions.easing.Bounce.easeOut, startPoint:5,
 shape:mx.transitions.Iris.CIRCLE}
```

若要确认您在 *transParam* 对象的 *type* 参数中指定的过渡类效果所必需的其它参数，请参见该过渡类的 API。例如，有关 *Blinds* 过渡类的更多信息，请参见第 1150 页的“遮帘过渡”。



*transParams* 对象的 *type* 参数应包括为其参数指定的类的完整类包名称，除非已通过使用 *import* 语句导入了它们。为了避免不得不为所有 *transParams* 参数集合都提供完整类包名称，请预先在代码中放置以下 *import* 语句，以导入所有 *mx.transitions* 类和所有 *mx.transitions.easing* 类：

```
import mx.transitions.*;
import mx.transitions.easing.*;
```

## 返回

指定 *TransitionManager* 实例要应用的过渡对象的实例。

## 说明

方法：创建和启动指定的过渡类的实例，该实例将应用于指定 **TransitionManager** 实例要影响的幻灯片或影片剪辑。如果已经存在匹配的过渡，则会删除该过渡并创建和启动一个新过渡。

## 示例

以下代码导入 **TransitionManager** 类，并创建一个新的 **TransitionManager** 实例。接下来，**TransitionManager.startTransition()** 方法在其 **type** 参数中指定一个 **mx.transitions.Zoom** 过渡。**direction** 参数通过指定 **mx.transitions.Transition.OUT**，指示过渡的方向为“输出”。过渡的持续时间是 3 秒。可通过使用 **Bounce** 类的 **mx.transitions.Bounce.easeOut()** 方法计算缓动。此效果使 **img1\_mc** 影片剪辑显示为一边跳动一边缩小，直到消失，整个效果持续 3 秒钟。

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
myTransitionManager.startTransition({type:Zoom, direction:Transition.OUT,
 duration:3, easing:Bounce.easeOut});
```

# TransitionManager.toString()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*transitionManagerInstance.toString()*

## 返回

返回以下字符串: "[TransitionManager]"。

## 说明

方法：以字符串的形式返回 **TransitionManager** 对象的类型。



## 示例

以下代码指示 **TransitionManager** 实例返回一个指示其类型的字符串：

```
import mx.transitions.*;
import mx.transitions.easing.*;
var myTransitionManager:TransitionManager = new TransitionManager(img1_mc);
var myType:String = myTransitionManager.toString();
trace(myType);
```

# 基于过渡的类

继承 （根类）

**ActionScript 类名称** mx.transitions.Transition

**Transition** 类是所有过渡类的基类。您不必直接使用或访问此类。它允许基于过渡的类共享某些可通过 **TransitionManager** 类的实例访问的公共行为和属性。基于过渡的类所定义的效果可多次应用于影片剪辑或幻灯片。

**Flash** 包含十种过渡效果，您可以将这些效果应用于影片剪辑对象。所有过渡均可以通过包括可选的缓动方法进行自定义，并且大多数过渡接受使用多个可选参数以控制其效果的特定方面。“缓动”是指动画过程中的渐进加速或减速，它会使您的动画看起来更逼真。例如，一个球在刚开始运动阶段是以加速形式运动的，在接近停止到完全停止阶段是以减速形式运动的。关于此加速和减速有许多公式，它们可以对缓动动画进行更改。

过渡与 **TransitionManager** 类组合使用。请参见第 1139 页的“**TransitionManager** 类”。您可以使用 **TransitionManager** 来指定过渡并将其应用于影片剪辑对象，而不是直接调用过渡。例如，若要将 **Zoom** 过渡应用于名为 **img1\_mc** 的影片剪辑，需要指定 **Zoom** 过渡类作为 **TransitionManager.start()** 中的 **type** 参数：

```
mx.transitions.TransitionManager.start(myMovieClip_mc,
 {type:mx.transitions.Zoom, direction:mx.transitions.Transition.IN,
 duration:1, easing:mx.transitions.easing.Bounce.easeOut});
```

Flash 中包括以下过渡：

过渡	说明
遮帘过渡	使用逐渐消失或逐渐出现的矩形来显示影片剪辑对象。
淡化过渡	淡入或淡出影片剪辑对象。
飞行过渡	从某一指定方向滑入影片剪辑对象。
光圈过渡	使用可以缩放的方形或圆形动画遮罩来显示或隐藏影片剪辑对象。
照片过渡	使影片剪辑对象像放映照片一样出现或消失。
像素溶解过渡	使用随机出现或消失的棋盘图案矩形来显示或隐藏影片剪辑对象。
旋转过渡	旋转影片剪辑对象。
挤压过渡	水平或垂直缩放影片剪辑对象。
划入 / 划出过渡	使用水平移动的动画遮罩形状来显示或隐藏影片剪辑对象。
缩放过渡	通过按比例缩放来放大或缩小影片剪辑对象。

 这些过渡仅在 ActionScript 2.0 中可用。

## 遮帘过渡

ActionScript 类名称 `mx.transitions.Blinds`

### 参数

*numStrips*，“遮帘”效果中的遮罩条纹数。建议的范围是 1 到 50。

*dimension*，一个整数，指示遮帘条纹是垂直的 (0) 还是水平的 (1)。

### 说明

一种过渡效果：使用逐渐消失或逐渐出现的矩形来显示影片剪辑对象。

该类的用法为将 `mx.transitions.Blinds` 指定为 `TransitionManager` 类的 `transObject.type` 参数。

## 示例

以下代码创建一个 **TransitionManager** 实例，该实例应用 numStrips 为 10、dimension 整数指定为垂直 (0) 的“遮帘”过渡。过渡的内容目标为影片剪辑 **img1\_mc**。该 **TransitionManager** 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 2 秒并且没有缓动。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Blinds, direction:Transition.IN,
 duration:2, easing:None.easeNone, numStrips:10, dimension:0});
```

## 淡化过渡

**ActionScript 类名称** `mx.transitions.Fade`

### 说明

一种过渡效果：淡入或淡出影片剪辑对象。

该类的用法为将 `mx.transitions.Fade` 指定为 **TransitionManager** 类的 `transObject.type` 参数。

### 示例

以下代码创建一个应用“淡化”过渡的 **TransitionManager** 实例。过渡的内容目标为影片剪辑 **img1\_mc**。该 **TransitionManager** 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 3 秒并且没有缓动。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Fade, direction:Transition.IN,
 duration:3, easing:None.easeNone});
```

## 飞行过渡

**ActionScript 类名称** `mx.transitions.Fly`

### 参数

*startPoint*，一个指示起始位置的整数；范围是 1 到 9：

左上, 1；上中, 2；右上, 3；左中, 4；中心, 5；右中, 6；左下, 7；下中, 8；右下, 9。

## 说明

一种过渡效果：从某一指定方向滑入影片剪辑对象。

该类的用法为将 `mx.transitions.Fly` 指定为 `TransitionManager` 类的 `transObject.type` 参数。

## 示例

以下代码创建一个 `TransitionManager` 实例，该实例应用 `startPoint` 设置为右下角 (9) 的“飞行”过渡。过渡的内容目标为影片剪辑 `img1_mc`。该 `TransitionManager` 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 3 秒并且具有 `Elastic.easeOut` 缓动效果。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Fly, direction:Transition.IN,
 duration:3, easing:Elastic.easeOut, startPoint:9});
```

# 光圈过渡

**ActionScript 类名称** `mx.transitions.Iris`

## 参数

*startPoint*，一个指示起始位置的整数；范围是 1 到 9：

左上, 1；上中, 2；右上, 3；左中, 4；中心, 5；右中, 6；左下, 7；下中, 8；右下, 9。

*shape*，值为 `mx.transitions.Iris.SQUARE`（方形）或 `mx.transitions.Iris.CIRCLE`（圆形）的遮罩形状。

## 说明

一种过渡效果：使用可以缩放的方形或圆形动画遮罩来显示影片剪辑对象。

该类的用法为将 `mx.transitions.Iris` 指定为 `TransitionManager` 类的 `transObject.type` 参数。

## 示例

以下代码创建一个 `TransitionManager` 实例，该实例应用 `startPoint` 为中心 (5) 且遮罩形状值为 `mx.transitions.Iris.CIRCLE` 的“光圈”过渡。过渡的 `content` 目标为影片剪辑 `img1_mc`。该 `TransitionManager` 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 2 秒并且具有“强制”缓动（通过指定 `mx.transitions.easing.Strong.easeOut` 缓动计算方法来强调 **easeOut**）。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Iris, direction:Transition.IN,
 duration:2, easing:Strong.easeOut, startPoint:5, shape:Iris.CIRCLE});
```

## 照片过渡

**ActionScript 类名称** mx.transitions.Photo

### 说明

一种过渡效果：使影片剪辑对象像放映照片一样出现或消失。

该类的用法为将 `mx.transitions.Photo` 指定为 **TransitionManager** 类的 `transObject.type` 参数。

### 示例

以下代码创建一个 **TransitionManager** 实例，该实例向内容目标影片剪辑 `img1_mc` 应用“照片”过渡。该 **TransitionManager** 类将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 1 秒而且不具有缓动。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start (img1_mc, {type:Photo, direction:Transition.IN,
 duration:1, easing:None.easeNone});
```

## 像素溶解过渡

**ActionScript 类名称** mx.transitions.PixelDissolve

### 参数

`xSections`，一个整数，指示沿水平轴的遮罩矩形部分的数目。建议的范围是 1 到 50。

`ySections`，一个整数，指示沿垂直轴的遮罩矩形部分的数目。建议的范围是 1 到 50。

### 说明

一种过渡效果：使用随机出现或消失的棋盘图案矩形来显示影片剪辑对象。

该类的用法为将 `mx.transitions.PixelDissolve` 指定为 **TransitionManager** 类的 `transObject.type` 参数。

### 示例

以下代码创建一个 **TransitionManager** 实例，该实例应用 `xSections` 为 10 且 `ySections` 为 10 的“像素溶解”过渡。过渡的内容目标为影片剪辑 `img1_mc`。该 **TransitionManager** 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 2 秒并且没有缓动。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:PixelDissolve,
 direction:Transition.IN, duration:2, easing:None.easeNone, xSections:10,
 ySections:10});
```

## 旋转过渡

ActionScript 类名称 `mx.transitions.Rotate`

### 参数

*ccw*, 一个布尔值：对于顺时针旋转为 *false*；对于逆时针旋转为 *true*。

*degrees*, 一个整数，指示对象要旋转的度数。建议是范围是 1 到 9999。例如，*degrees* 设置为 1080 时，会将对象完全旋转三次。

### 说明

一种过渡效果：旋转影片剪辑对象。

该类的用法为将 `mx.transitions.Rotate` 指定为 `TransitionManager` 类的 `transObject.type` 参数。

### 示例

以下代码创建一个 `TransitionManager` 实例，该实例应用顺时针 720 度（两整周）的“旋转”过渡。过渡的内容目标为影片剪辑 `img1_mc`。该 `TransitionManager` 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 3 秒、缓动设置为 `Strong.easeInOut`。这样，过渡开始时缓慢，然后加速，最后缓慢结束。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Rotate, direction:Transition.IN,
 duration:3, easing:Strong.easeInOut, ccw:false, degrees:720});
```

## 挤压过渡

ActionScript 类名称 `mx.transitions.Squeeze`

### 参数

*dimension*, 一个整数，指示“挤压”效果应是水平的 (0) 还是垂直的 (1)。

### 说明

一种过渡效果：水平或垂直缩放影片剪辑对象。

该类的用法为将 `mx.transitions.Squeeze` 指定为 `TransitionManager` 类的 `transObject.type` 参数。

## 示例

以下代码创建一个 **TransitionManager** 实例，该实例应用 `dimension` 整数指定为垂直 (1) 的“挤压”过渡。过渡的内容目标为影片剪辑 `img1_mc`。该 **TransitionManager** 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 2 秒并且在 `easeOut` 的方向上有“弹性”缓动效果。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Squeeze, direction:Transition.IN,
 duration:2, easing:Elastic.easeOut, dimension:1});
```

## 划入 / 划出过渡

**ActionScript 类名称** `mx.transitions.Wipe`

### 参数

*startPoint*，一个整数，指示开始位置。范围是 1 到 4 和 6 到 9：

左上，1；上中，2；右上，3；左中，4；右中，6；左下，7；下中，8；右下，9。

### 说明

一种过渡效果：使用水平移动的动画遮罩形状来显示或隐藏影片剪辑对象。

该类的用法为将 `mx.transitions.Wipe` 指定为 **TransitionManager** 类的 `transObject.type` 参数。

## 示例

以下代码创建一个 **TransitionManager** 实例，该实例应用 `startPoint` 为左上角 (1) 的“划入 / 划出”过渡。过渡的内容目标为影片剪辑 `img1_mc`。该 **TransitionManager** 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 2 秒并且没有缓动。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Wipe, direction:Transition.IN,
 duration:2, easing:None.easeNone, startPoint:1});
```

## 缩放过渡

ActionScript 类名称 `mx.transitions.Zoom`

### 说明

一种过渡效果：通过按比例缩放来放大或缩小影片剪辑对象。

该类的用法为将 `mx.transitions.Zoom` 指定为 `TransitionManager` 类的 `transObject.type` 参数。

### 示例

以下代码创建一个 `TransitionManager` 实例，该实例向内容目标影片剪辑 `img1_mc` 应用“缩放”过渡。该 `TransitionManager` 实例将应用的效果：方向为 `mx.transitions.Transition.IN`、持续时间为 2 秒、具有“弹性”类型的缓动（开始时速度快，结束时速度递减）。

```
import mx.transitions.*;
import mx.transitions.easing.*;
TransitionManager.start(img1_mc, {type:Zoom, direction:Transition.IN,
 duration:2, easing:Elastic.easeOut});
```



# TreeDataProvider 接口（仅限 Flash Professional）

# 49

**TreeDataProvider** 接口是一组属性和方法，无需对其实例化即可使用。如果在 SWF 中打包了 **Tree** 类，则该 SWF 文件中的所有 XML 实例都包含 **TreeDataProvider** 接口。树中的所有节点都是包含 **TreeDataProvider** 接口的 XML 对象。

使用 **TreeDataProvider** 方法为 `Tree.dataProvider` 属性创建 XML 是最佳方式，因为只有 **TreeDataProvider** 广播用于刷新树显示的事件。这些事件由 **Tree** 类处理；您无需编写函数来处理它们。（内置 XML 类方法不广播这样的事件。）

可以使用 **TreeDataProvider** 的方法来控制数据模型和数据显示。可以将内置的 XML 类方法用于只读的任务，如遍历树的层次结构。

通过指定 `labelField` 或 `labelFunction` 属性可选择保存有待显示文本的属性。例如，代码 `myTree.labelField = "firstName"`；会计算出对显示的文本进行查询的 `myTreeDP.attributes.fred` 属性的值。

## TreeDataProvider 接口的方法摘要

下表列出了 **TreeDataProvider** 接口的方法。

方法	说明
<code>TreeDataProvider.addTreeNode()</code>	在树的根添加一个子节点。
<code>TreeDataProvider.addTreeNodeAt()</code>	在父节点上的指定位置添加一个子节点。
<code>TreeDataProvider.getTreeNodeAt()</code>	返回节点的指定子项。
<code>TreeDataProvider.removeTreeNode()</code>	从节点的父项中删除节点和节点的所有后代。
<code>TreeDataProvider.removeTreeNodeAt()</code>	从子节点的索引位置处删除节点和节点的所有后代。

# TreeDataProvider 接口的属性摘要

下表列出了 TreeDataProvider 接口的属性。

属性	说明
<code>TreeDataProvider.attributes.data</code>	指定与节点关联的数据。
<code>TreeDataProvider.attributes.label</code>	指定将在节点旁边显示的文本。

## TreeDataProvider.addTreeNode()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
someNode.addTreeNode(label, data)
```

用法 2:

```
someNode.addTreeNode(child)
```

### 参数

*label* 一个显示该节点的字符串。

*data* 一个与该节点关联的任何类型的对象。

*child* 任何 XMLNode 对象。

### 返回

添加的 XML 节点。

### 说明

方法；在树的根添加子节点。该节点或者是根据 *label* 和 *data* 参数中提供的信息构建的（用法 1），或者是根据作为 XMLNode 对象的预先生成的子节点构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

调用此方法将刷新树的显示。

## 示例

在以下示例中，第一行代码确定将向其添加子项的节点的位置。第二行将一个新节点添加到指定的节点。

```
var myTreeNode = myTreeDP.firstChild.firstChild;
myTreeNode.addTreeNode("Inbox", 3);
```

以下代码将一棵树中的某个节点移到另一棵树的根：

```
myTreeNode.addTreeNode(mySecondTree.getTreeNodeAt(3));
```

# TreeDataProvider.addTreeNodeAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

用法 1:

```
someNode.addTreeNodeAt(index, label, data)
```

用法 2:

```
someNode.addTreeNodeAt(index, child)
```

## 参数

*index* 一个整数，它表示应将节点添加到的索引位置（位于子节点中）。

*label* 一个显示该节点的字符串。

*data* 一个与该节点关联的任何类型的对象。

*child* 任何 **XMLNode** 对象。

## 返回

添加的 XML 节点。

## 说明

方法：在父节点中的指定位置添加子节点。该节点或者是根据 *label* 和 *data* 参数中提供的信息构建的（用法 1），或者是根据作为 **XMLNode** 对象的预先生成的子节点构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

调用此方法将刷新树的显示。

### 示例

以下代码确定将向其添加节点的节点的位置，并添加作为根的第二个子项的新节点：

```
var myTreeNode = myTreeDP.firstChild.firstChild;
myTreeNode.addTreeNodeAt(1, "Inbox", 3);
```

以下代码移动一棵树中的某个节点，使其成为另一棵树的根的第四个子项：

```
myTreeNode.addTreeNodeAt(3, mySecondTree.getTreeNodeAt(3));
```

## TreeDataProvider.attributes.data

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
someNode.attributes.data
```

### 说明

属性；指定与节点关联的数据。这会将值作为 **XMLNode** 对象中的属性添加。设置此属性不会刷新任何树显示。此属性可以是任何数据类型。

### 示例

以下代码确定要调整的节点的位置，并设置其 data 属性：

```
var myTreeNode = myTreeDP.firstChild.firstChild;
myTreeNode.attributes.data = "hi"; // 设置结果为 <node data = "hi">;
```

### 另请参见

[TreeDataProvider.attributes.label](#)

# TreeDataProvider.attributes.label

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
someNode.attributes.label
```

## 说明

属性：一个指定为节点显示的文本的字符串。此字符串会被写入 XMLNode 对象的一个属性。设置此属性不会刷新任何树显示。

## 示例

以下代码确定要调整的节点的位置，并设置其 `label` 属性。以下代码的结果是 `<node label="Mail">`。

```
var myTreeNode = myTreeDP.firstChild.firstChild;
myTreeNode.attributes.label = "Mail";
```

## 另请参见

[TreeDataProvider.attributes.data](#)

# TreeDataProvider.getTreeNodeAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
someNode.getTreeNodeAt(index)
```

## 参数

*index* 一个表示子节点在当前节点中的位置的整数。

## 返回

指定的节点。

## 说明

方法：返回节点的指定子节点。

## 示例

以下代码确定某个节点的位置，然后检索 myTreeNode 的第二个子项：

```
var myTreeNode = myTreeDP.firstChild.firstChild;
myTreeNode.getTreeNodeAt(1);
```

# TreeDataProvider.removeTreeNode()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
someNode.removeTreeNode()
```

## 参数

无。

## 返回

删除的 XML 节点或 undefined（如果发生错误）。

## 说明

方法：从指定节点的父级删除该节点及其所有后代。

## 示例

以下代码删除某个节点：

```
myTreeDP.firstChild.removeTreeNode();
```

# TreeDataProvider.removeTreeNodeAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
someNode.removeTreeNodeAt(index)
```

## 参数

*index* 一个整数，表示将被删除的节点的位置。

## 返回

删除的 XML 节点或 undefined（如果发生错误）。

## 说明

方法：删除由当前节点和子节点的索引位置指定的节点（及其所有后代）。调用此方法将刷新视图。

## 示例

以下代码删除 myTreeDP.firstChild 节点的第四个子项：

```
myTreeDP.firstChild.removeTreeNodeAt(3);
```





# Tree 组件（仅限 Flash Professional）

**Tree** 组件允许用户查看分层数据。树显示在类似 **List** 组件的框中，但树中的每一项称为节点，并且可以是叶或分支。默认情况下，用旁边带有文件图标的文本标签表示叶，用旁边带有文件夹图标的文本标签表示分支，并且文件夹图标带有展开箭头（展示三角形），用户可以打开它以显示子节点。分支的子项可以是叶或分支。

必须通过 XML 数据源提供树组件的数据。有关更多信息，请参见第 1166 页的“使用 [Tree 组件（仅限 Flash Professional）](#)”。

当树实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

键	说明
向下箭头	将选区向下移动一项。
向上箭头	将选区向上移动一项。
向右箭头	打开所选的分支节点。如果分支已打开，则会移到第一个子节点。
向左箭头	关闭所选的分支节点。如果是在已关闭分支节点的叶节点上，则会移到父节点。
空格键	打开或关闭所选的分支节点。
End	将选区移到列表底端。
Home	将选区移到列表顶端。
Page Down	将选区向下移动一页。
Page Up	将选区向上移动一页。
Ctrl	允许选择多个不相邻的项。
Shift	允许选择多个相邻的项。

无法使用屏幕阅读器来访问 **Tree** 组件。

## 使用 Tree 组件（仅限 Flash Professional）

**Tree** 组件可用于表示分层数据，如电子邮件客户端文件夹、文件浏览器窗格或库存的类别浏览系统。绝大多数情况下，是通过服务器以 XML 的形式来获得树的数据，但此数据也可以是在 **Flash** 中创建的结构合理的 XML。为树创建 XML 的最好方法是使用 **TreeDataProvider** 接口。也可以使用 **ActionScript XML** 类或建立 XML 字符串。创建 XML 数据源（或通过外部源加载一个 XML 数据源）后，需将其指定给 `Tree.dataProvider`。

**Tree** 组件由两组 API 组成：**Tree** 类和 **TreeDataProvider** 接口。**Tree** 类包含可视的配置方法和属性。**TreeDataProvider** 接口允许您构建 XML 并将其添加到多个树实例中。**TreeDataProvider** 对象会向任何使用它的树广播更改。另外，存在于与树或菜单相同的帧上的任何 XML 或 **XMLNode** 对象会自动获得 **TreeDataProvider** 方法和属性。有关更多信息，请参见第 1157 页的“**TreeDataProvider** 接口（仅限 **Flash Professional**）”。

### 为 Tree 组件设置 XML 格式

**Tree** 组件设计成利用 XML 作为数据模型来显示分层数据结构。了解 XML 数据源与 **Tree** 组件的关系很重要。

请考虑以下的 XML 数据源范例：

```
<node>
 <node label="Mail">
 <node label="INBOX"/>
 <node label="Personal Folder">
 <node label="Business" isBranch="true" />
 <node label="Demo" isBranch="true" />
 <node label="Personal" isBranch="true" />
 <node label="Saved Mail" isBranch="true" />
 <node label="bar" isBranch="true" />
 </node>
 <node label="Sent" isBranch="true" />
 <node label="Trash"/>
 </node>
</node>
```



`isBranch` 属性是只读属性；无法直接设置它。若要设置它，请调用 `Tree.setIsBranch()`。

XML 数据源中的节点可以有任何名称。请注意，上例中的每个节点都以一般性名称 `node` 进行命名。树会通读 XML，并根据各节点的嵌套关系建立显示层次结构。

在树中，每个 XML 节点可以显示为两种类型之一：分支或叶。分支节点可以包含多个子节点，并显示为带有展开箭头的文件夹图标，用户可以利用展开箭头打开和关闭文件夹。叶节点显示为文件图标，并且不能包含子节点。叶和分支都可以是根；根节点显示在树的最顶层，并且没有父节点。图标是可自定义的；有关更多信息，请参见第 1177 页的“使用具有 Tree 组件的外观”。

构建 XML 有很多种方式，但是 Tree 组件无法使用所有类型的 XML 结构。请勿在子节点中嵌套节点属性；每个节点应包含所有必需的属性。另外，每个节点的属性应一致，这样才实用。例如，要利用 Tree 组件描述信箱结构，请在每个节点上都使用相同的属性（消息、数据、时间、附件等）。这可让树知道它预期要呈现的内容，并可让您以循环方式遍历层次结构，以便比较数据。

当树显示节点时，默认情况下，它使用节点的 label 属性作为文本标签。如果存在任何其它属性，它们会成为树中节点属性的附加属性。

实际的根节点被解释为 Tree 组件本身。这表示第一个子节点（上例中的 <node label="Mail">）在树视图中显示为根节点。这表示树可以有多个根文件夹。在上例中，树中只显示一个根文件夹：“Mail”。但是，如果您将在 XML 中向该层添加同级节点，则树中会显示多个根节点。

树的数据提供程序总是需要拥有能够显示的子节点的节点。它显示 XMLNode 对象的第一个子节点。当 XML 包装在一个 XML 对象中时，结构看起来就像下面这样：

```
<XMLDocumentObject>
 <node>
 <node label="Mail">
 <node label="INBOX"/>
 <node label="Personal Folder">
 <node label="Business" isBranch="true" />
 <node label="Demo" isBranch="true" />
 <node label="Personal" isBranch="true" />
 <node label="Saved Mail" isBranch="true" />
 <node label="bar" isBranch="true" />
 </node>
 <node label="Sent" isBranch="true" />
 <node label="Trash"/>
 </node>
 </node>
</XMLDocumentObject>
```

Flash Player 将 XML 节点包装在传递给树的另外一个文档节点中。当树要显示 XML 的时候，它会试着显示 <node>，但是该节点没有标签，所以无法正确显示。

为了避免这个问题，Tree 组件的数据提供程序应该指向 XMLDocumentObject 的第一个子节点，如下所示：

```
myTree.dataProvider = myXML.firstChild;
```

## Tree 参数

可以在“属性”检查器或“组件”检查器中为每个 **Tree** 组件实例设置以下创作参数：

**multipleSelection** 是一个布尔值，它指示用户是 (true) 否 (false) 可以选择多个项。默认值为 **false**。

**rowHeight** 指示每行的高度（以像素为单位）。默认值为 20。

您可以编写 **ActionScript**，以便使用 **Tree** 组件的属性、方法和事件来控制该组件的这些和其它选项。有关更多信息，请参见第 1177 页的“**Tree 类（仅限 Flash Professional）**”。

对于 **Tree** 组件，您不能像操作其它组件那样在“属性”检查器或“组件”检查器中为其输入数据参数。有关更多信息，请参见第 1166 页的“使用 **Tree** 组件（仅限 **Flash Professional**）”和第 1168 页的“创建具有 **Tree** 组件的应用程序”。

## 创建具有 Tree 组件的应用程序

下面的步骤展示了如何使用 **Tree** 组件在一个电子邮件应用程序中显示信箱。

**Tree** 组件不允许您在“属性”检查器或“组件”检查器中输入数据参数。由于 **Tree** 组件的复杂数据结构，您必须在运行时导入 XML 对象，或者在创作时在 **Flash** 中建立一个 XML 对象。要在 **Flash** 中创建 XML，可以使用 **TreeDataProvider** 接口、使用 **ActionScript** XML 对象或建立 XML 字符串。这些选择会在以下过程中逐一说明。

将 **Tree** 组件添加到应用程序并加载 XML：

1. 在 **Flash** 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 将文档另存为 **treeMenu.fla**。
3. 在“组件”面板中，双击 **Tree** 组件以将其添加到舞台上。
4. 选择该 **Tree** 实例。在“属性”检查器中，输入实例名称 **menuTree**。
5. 选择该 **Tree** 实例并按 F8。选择“影片剪辑”，然后输入名称 **TreeNavMenu**。
6. 单击“高级”按钮，并选择“为 **ActionScript** 导出”。
7. 在“AS 2.0 类”文本框中输入 **TreeNavMenu**，然后单击“确定”。
8. 选择“文件”>“新建”并选择“**ActionScript** 文件”。
9. 将文件另存为 **TreeNavMenu.as**，与 **treeMenu.fla** 文件放在同一目录下。
10. 在“脚本”窗口中，输入以下代码：

```
import mx.controls.Tree;

class TreeNavMenu extends MovieClip {
 var menuXML:XML;
 var menuTree:Tree;
 function TreeNavMenu() {
```

```

// 设置树的外观和事件处理函数。
menuTree.setStyle("fontFamily", "_sans");
menuTree.setStyle("fontSize", 12);
// 加载 XML 菜单。
var treeNavMenu = this;
menuXML = new XML();
menuXML.ignoreWhite = true;
menuXML.load("TreeNavMenu.xml");
menuXML.onLoad = function() {
 treeNavMenu.onMenuLoaded();
};
}
function change(event:Object) {
 if (menuTree == event.target) {
 var node = menuTree.selectedItem;
 // 如果遇到分支, 则展开 / 折叠它。
 if (menuTree.getIsBranch(node)) {
 menuTree.setIsOpen(node, !menuTree.getIsOpen(node), true);
 }
 // 如果是超链接, 则跳过。
 var url = node.attributes.url;
 if (url) {
 getURL(url, "_top");
 }
 // 清除所有选择。
 menuTree.selectedNode = null;
 }
}
function onMenuLoaded() {
 menuTree.dataProvider = menuXML.firstChild;
 menuTree.addEventListener("change", this);
}
}

```

这段 **ActionScript** 代码设置树的样式。它创建了一个 **XML** 对象, 用于加载创建树节点的 **XML** 文件。然后定义 **onLoad** 事件处理函数, 以将数据提供程序设置为 **XML** 文件的内容。

**11.** 在文本编辑器中新建一个名为 **TreeNavMenu.xml** 的文件。

**12.** 在文件中输入以下代码:

```

<node>
 <node label="My Bookmarks">
 <node label="Macromedia Web site" url="http://www.macromedia.com" />
 <node label="MXNA blog aggregator" url="http://www.markme.com/mxna" />
 >
</node>
<node label="Google" url="http://www.google.com" />
</node>

```

**13.** 保存文档并返回到 **treeMenu.fla**。选择 “控制” > “测试影片” 来测试应用程序。

### 从外部文件加载 XML：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 将 Tree 组件的一个实例拖到舞台上。
3. 选择该 Tree 实例。在“属性”检查器中，输入实例名称 **myTree**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```
var myTreeDP:XML = new XML();
myTreeDP.ignoreWhite = true;
myTreeDP.load("treeXML.xml");
myTreeDP.onLoad = function() {
 myTree.dataProvider = this.firstChild;
};
var treeListener:Object = new Object();
treeListener.change = function(evt:Object) {
 trace("selected node is: "+evt.target.selectedNode);
 trace("");
};
myTree.addEventListener("change", treeListener);
```

这段代码创建一个名为 myTreeDP 的 XML 对象，并调用 XML.load() 方法来加载 XML 数据源。然后，代码定义了 onLoad 事件处理函数，该函数在 XML 加载时将 myTree 实例的 dataProvider 属性设置为新的 XML 对象。有关该 XML 对象的更多信息，请参见它在《ActionScript 2.0 语言参考》中的条目。

5. 在文本编辑器中创建一个名为 **treeXML.xml** 的新文件。
6. 在文件中输入以下代码：

```
<node>
 <node label="Mail">
 <node label="INBOX"/>
 <node label="Personal Folder">
 <node label="Business" isBranch="true" />
 <node label="Demo" isBranch="true" />
 <node label="Personal" isBranch="true" />
 <node label="Saved Mail" isBranch="true" />
 <node label="bar" isBranch="true" />
 </node>
 <node label="Sent" isBranch="true" />
 <node label="Trash"/>
 </node>
</node>
```

7. 选择“控制”>“测试影片”。

在 SWF 文件中，您可以看到树中显示的 XML 结构。单击树中的各项可以看到 change 事件处理函数中将数据值发送到“输出”面板的 trace() 语句。

在创作时使用 **TreeDataProvider** 类在 Flash 中创建 XML：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 将 **Tree** 组件的一个实例拖到舞台上。
3. 选择该 **Tree** 实例，并在“属性”检查器中输入实例名称 **myTree**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```
var myTreeDP:XML = new XML();
myTreeDP.addTreeNode("Local Folders", 0);
// 使用 XML.firstChild 嵌套 Local Folders 下的子节点。
var myTreeNode:XMLNode = myTreeDP.firstChild;
myTreeNode.addTreeNode("Inbox", 1);
myTreeNode.addTreeNode("Outbox", 2);
myTreeNode.addTreeNode("Sent Items", 3);
myTreeNode.addTreeNode("Deleted Items", 4);
// 将 myTreeDP 数据源指定给 myTree 组件。
myTree.dataProvider = myTreeDP;
// 将四个子节点全部设置为分支。
for (var i = 0; i<myTreeNode.childNodes.length; i++) {
 var node:XMLNode = myTreeNode.getTreeNodeAt(i);
 myTree.setIsBranch(node, true);
}
```

这段代码创建了一个名为 **myTreeDP** 的 **XML** 对象。与 **Tree** 组件位于同一帧上的任何 **XML** 对象都会自动接收 **TreeDataProvider** 接口的所有属性和方法。第二行代码创建了一个名为 **Local Folders** 的根节点。有关其余代码的详细信息，请参见代码中的注释（以 `//` 开头的行）。

5. 选择“控制”>“测试影片”。

在 **SWF** 文件中，您可以看到树中显示的 **XML** 结构。单击树中的各项可以看到 `change` 事件处理函数中将数据值发送到“输出”面板的 `trace()` 语句。

使用 **ActionScript XML** 类创建 **XML**：

1. 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
2. 将 **Tree** 组件的一个实例拖到舞台上。
3. 选择该 **Tree** 实例。在“属性”检查器中，输入实例名称 **myTree**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```
// 创建一个 XML 对象。
var myTreeDP:XML = new XML();
// 创建节点值。
var myNode0:XMLNode = myTreeDP.createElement("node");
myNode0.attributes.label = "Local Folders";
myNode0.attributes.data = 0;
var myNode1:XMLNode = myTreeDP.createElement("node");
myNode1.attributes.label = "Inbox";
myNode1.attributes.data = 1;
```

```

var myNode2:XMLNode = myTreeDP.createElement("node");
myNode2.attributes.label = "Outbox";
myNode2.attributes.data = 2;
var myNode3:XMLNode = myTreeDP.createElement("node");
myNode3.attributes.label = "Sent Items";
myNode3.attributes.data = 3;
var myNode4:XMLNode = myTreeDP.createElement("node");
myNode4.attributes.label = "Deleted Items";
myNode4.attributes.data = 4;
// 将节点指定到 XML 树中的层次结构。
myTreeDP.appendChild(myNode0);
myTreeDP.firstChild.appendChild(myNode1);
myTreeDP.firstChild.appendChild(myNode2);
myTreeDP.firstChild.appendChild(myNode3);
myTreeDP.firstChild.appendChild(myNode4);
// 将 myTreeDP 数据源指定给 Tree 组件。
myTree.dataProvider = myTreeDP;

```

有关该 XML 对象的更多信息，请参见它在《ActionScript 2.0 语言参考》中的条目。

#### 5. 选择“控制” > “测试影片”。

在 SWF 文件中，您可以看到树中显示的 XML 结构。单击树中的各项可以看到 change 事件处理函数中将数据值发送到“输出”面板的 trace() 语句。

#### 在创作时使用构造良好的字符串在 Flash 中创建 XML：

1. 在 Flash 中，选择“文件” > “新建”，然后选择“Flash 文档”。
2. 将 Tree 组件的一个实例拖到舞台上。
3. 选择该 Tree 实例。在“属性”检查器中，输入实例名称 **myTree**。
4. 在“动作”面板中的第 1 帧上，输入以下代码：

```

var myTreeDP:XML = new XML("<node label='Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node>");
myTree.dataProvider = myTreeDP;

```

这段代码创建了 XML 对象 myTreeDP，并将其指定给 myTree 的 dataProvider 属性。

#### 5. 选择“控制” > “测试影片”。

在 SWF 文件中，您可以看到树中显示的 XML 结构。单击树中的各项可以看到 change 事件处理函数中将数据值发送到“输出”面板的 trace() 语句。



# 自定义 Tree 组件（仅限 Flash Professional）

在创作过程中和运行时，可以在水平方向和垂直方向上将 **Tree** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参见 `UIObject.setSize()`）。当树的宽度不足以显示节点的文本时，文本会被裁剪。

## 对 Tree 组件使用样式

**Tree** 组件使用下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
<code>backgroundColor</code>	光晕和范例	列表的背景颜色。默认的颜色为白色，在类样式声明中定义。如果指定了 <code>alternatingRowColors</code> ，则此样式被忽略。
<code>backgroundDisabledColor</code>	光晕和范例	当组件的 <code>enabled</code> 属性设置为 "false" 时的背景颜色。默认值为 <code>0xDDDDDD</code> （中度灰）。
<code>depthColors</code>	光晕和范例	根据各节点的深度设置各个行的背景色。值是一个颜色数组，其中第一个元素是根节点的背景色，第二个元素是它的子节点的背景色，以此类推，直到穷举完数组中的所有颜色。默认情况下不设置此样式属性。
<code>borderStyle</code>	光晕和范例	<b>Tree</b> 组件使用 <code>RectBorder</code> 实例作为其边框，并对该类定义的样式做出响应。请参见第 985 页的“ <a href="#">RectBorder 类</a> ”。  默认边框样式为 "inset"。
<code>color</code>	光晕和范例	文本颜色。
<code>disabledColor</code>	光晕和范例	组件禁用时的文本颜色。默认值为 <code>0x848384</code> （深灰）。
<code>embedFonts</code>	光晕和范例	一个布尔值，它指示在 <code>fontFamily</code> 中指定的字体是否为嵌入字体。如果 <code>fontFamily</code> 引用了嵌入字体，则此样式必须设置为 <code>true</code> 。否则，将不使用该嵌入字体。如果此样式设置为 <code>true</code> ，并且 <code>fontFamily</code> 不引用嵌入字体，则不会显示任何文本。默认值为 <code>false</code> 。
<code>fontFamily</code>	光晕和范例	文本的字体名称。默认值为 "_sans"。

样式	主题	说明
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式: "normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细: "none" 或 "bold"。默认值为 "none"。在调用 <code>setStyle()</code> 期间, 所有组件还可以接受值 "normal" 来代替 "none", 但随后对 <code>getStyle()</code> 的调用将返回 "none"。
textAlign	光晕和范例	文本对齐方式: "left"、"right" 或 "center"。默认值为 "left"。
textDecoration	光晕和范例	文本修饰: "none" 或 "underline"。默认值为 "none"。
textIndent	光晕和范例	表示文本缩进的数字。默认值为 0。
defaultLeafIcon	光晕和范例	当没有为特定节点指定图标时, <code>Tree</code> 控件中为叶节点显示的图标。默认值是 "TreeNodeIcon", 代表一个纸片状的图像。
disclosureClosedIcon	光晕和范例	<code>Tree</code> 组件中显示在关闭的文件夹节点旁的图标。默认值是 "TreeDisclosureClosed", 代表一个灰色向右箭头。
disclosureOpenIcon	光晕和范例	<code>Tree</code> 组件中显示在打开的文件夹节点旁的图标。默认值是 "TreeDisclosureOpen", 代表一个灰色向下箭头。
folderClosedIcon	光晕和范例	在未设置特定于节点的图标时, <code>Tree</code> 组件中为关闭的文件夹节点显示的图标。默认值是 "TreeFolderClosed", 代表一个关闭的黄色文件夹图像。
folderOpenIcon	光晕和范例	在未设置特定于节点的图标时, <code>Tree</code> 组件中为打开的文件夹节点显示的图标。默认值是 "TreeFolderOpen", 代表一个打开的黄色文件夹图像。
indentation	光晕和范例	<code>Tree</code> 组件中每行缩进的像素数。默认值为 17。
openDuration	光晕和范例	展开和折叠动画的持续时间 (以毫秒为单位)。默认值为 250。
openEasing	光晕和范例	对控制展开和折叠动画的补间函数的引用。默认为正弦输入 / 输出。有关更多信息, 请参见《使用组件》中的“自定义组件动画”。
repeatDelay	光晕和范例	用户在滚动条中第一次按下某个按钮与开始重复此动作之间间隔的毫秒数。默认值为 500 (半秒)。
repeatInterval	光晕和范例	用户在滚动条的某个按钮上使鼠标按键保持按下状态时两次自动单击之间所间隔的毫秒数。默认值为 35。

样式	主题	说明
rollOverColor	光晕和范例	<p>滑过的行的背景颜色。“光晕”主题的默认值为 <code>OxE3FFD6</code>（亮绿），“范例”主题的默认值为 <code>OxAAAAAA</code>（浅灰）。</p> <p>当通过调用 <code>setStyle()</code> 更改 <code>themeColor</code> 时，框架会将 <code>rollOverColor</code> 设置为一个与所选 <code>themeColor</code> 相关的值。</p>
selectionColor	光晕和范例	<p>所选行的背景颜色。“光晕”主题的默认值为 <code>OxCDFFC1</code>（浅绿），“范例”主题的默认值为 <code>OxEEEEEE</code>（极浅灰）。</p> <p>当通过调用 <code>setStyle()</code> 更改 <code>themeColor</code> 时，框架会将 <code>selectionColor</code> 设置为一个与所选 <code>themeColor</code> 相关的值。</p>
selectionDuration	光晕和范例	在正常状态和所选状态之间过渡所耗费的时间，以毫秒为单位。默认值为 <code>200</code> 。
selectionDisabledColor	光晕和范例	<p>所选行的背景颜色。默认值为 <code>OxDDDDDD</code>（中度灰）。因为此属性的默认值与 <code>backgroundDisabledColor</code> 的默认值相同，所以除非更改了其中的某个样式属性，否则当组件被禁用时该选区会不可见。</p>
selectionEasing	光晕和范例	对用于控制选择状态间过渡的扩大等式的引用。只适用于从正常状态到所选择状态的过渡。默认等式使用正弦输入 / 输出公式。有关更多信息，请参见《使用组件》中的“自定义组件动画”。
textRollOverColor	光晕和范例	<p>指针在文本上滑过时，该文本的颜色。默认值为 <code>Ox2B333C</code>（深灰）。设置 <code>rollOverColor</code> 时此样式非常重要，因为这两个设置必须相辅相成，才能使文本在指针滑过期间易于查看。</p>
textSelectedColor	光晕和范例	<p>所选行的文本颜色。默认值为 <code>Ox005F33</code>（深灰）。设置 <code>selectionColor</code> 时此样式非常重要，因为这两个设置必须相辅相成，才能使选中的文本易于查看。</p>
useRollOver	光晕和范例	确定滑过一行时是否激活亮显示该行。默认值为 <code>true</code> 。

例如，以下代码使用 `UIObject.createClassObject()` 创建一个 **Tree** 实例 `first_tr`，并使用一个数据提供程序填充树，然后使用 `UIObject.setStyle()` 将树节点的缩进更改为 8 像素。将一个 **Tree** 组件拖到当前文档的库中，然后将以下代码添加到主时间轴的第一帧中：

```
import mx.controls.Tree;

this.createClassObject(Tree, "first_tr", 20);
first_tr.setSize(200, 100);
first_tr.move(0, 120);

var trDP_xml:XML = new XML("<node label='1st Local Folder'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /><node label='2nd
 Local Folder'><node label='Inbox' data='0' /><node label='Outbox' data='1' /
 ></node></node>");
first_tr.dataProvider = trDP_xml;

first_tr.setStyle("indentation", 8);
```

## 为文档中的所有 Tree 组件设置样式

**Tree** 类继承自 **List** 类，而 **List** 类又继承自 **ScrollSelectList** 类。默认的分类样式属性在 **ScrollSelectList** 类上定义，而 **Menu** 组件和所有基于列表的组件均扩展了 **ScrollSelectList** 类。您可以直接为此类设置新的默认样式值，新的设置将反映在所有受影响的类中。

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

若要只对 **Tree** 组件设置样式属性，可以创建新的 `CSSStyleDeclaration` 实例，并将其存储在 `_global.styles.DataGrid` 中。

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.Tree == undefined) {
 _global.styles.Tree = new CSSStyleDeclaration();
}
_global.styles.Tree.setStyle("backgroundColor", 0xFF00AA);
```

当创建新的分类样式声明时，由 `ScrollSelectList` 声明提供的所有默认值都将丢失。这包括支持鼠标事件所需的 `backgroundColor`。若要创建分类样式声明并保留默认值，请使用下面所示的 `for` 循环，将旧的设置复制到新的声明中。

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.Tree;
for (var style in source) {
 target.setStyle(style, source.getStyle(style));
}
```

## 使用具有 Tree 组件的外观

Tree 组件使用一个 `RectBorder` 实例作为其边框，并使用滚动条滚动图像。有关设置这些可视元素的外观的更多信息，请参见第 985 页的“`RectBorder` 类”和第 1283 页的“对 `UIScrollBar` 组件使用外观”。

## Tree 类（仅限 Flash Professional）

继承 `MovieClip` > `UIObject` 类 > `UIComponent` 类 > `View` > `ScrollView` > `ScrollSelectList` > `List` 组件 > `Tree`

ActionScript 类名称 `mx.controls.Tree`

Tree 类的方法、属性和事件使您可以管理和处理 Tree 对象。

### Tree 类的方法摘要

下表列出了 Tree 类的方法。

方法	说明
<code>Tree.addNode()</code>	向 Tree 实例添加节点。
<code>Tree.addNodeAt()</code>	在 Tree 实例中的特定位置添加节点。
<code>Tree.getDisplayIndex()</code>	返回给定节点的显示索引。
<code>Tree.getIsBranch()</code>	指定文件夹是否为分支（具有文件夹图标和展开箭头）。
<code>Tree.getIsOpen()</code>	指示节点是打开还是关闭。
<code>Tree.getNodeDisplayedAt()</code>	将树的某个显示索引映射到在该索引处显示的节点。
<code>Tree.getTreeNodeAt()</code>	返回在树的根上的节点。
<code>Tree.refresh()</code>	更新树。
<code>Tree.removeAll()</code>	从 Tree 实例中删除所有节点并刷新树。
<code>Tree.removeTreeNodeAt()</code>	删除在指定位置的节点并刷新树。
<code>Tree.setIcon()</code>	为指定的节点指定图标。
<code>Tree.setIsBranch()</code>	指定节点是否为分支（具有文件夹图标和展开箭头）。
<code>Tree.setIsOpen()</code>	打开或关闭节点。

## 从 UIObject 类继承的方法

下表列出了 **Tree** 类从 **UIObject** 类继承的方法。从 **Tree** 对象调用这些方法时，请使用 *TreeInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **Tree** 类从 **UIComponent** 类继承的方法。从 **Tree** 对象调用这些方法时，请使用 *TreeInstance.methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## 从 List 类继承的方法

下表列出了 **Tree** 类从 **List** 类继承的方法。从 **Tree** 对象调用这些方法时，请使用 *TreeInstance.methodName* 的形式。

方法	说明
<code>List.addItem()</code>	向列表的结尾添加项目。
<code>List.addItemAt()</code>	将项目添加到指定索引处的列表。对于 <b>Tree</b> 组件，最好使用 <code>Tree.addTreeNodeAt()</code> 。
<code>List.getItemAt()</code>	返回指定索引处的项目。
<code>List.removeAll()</code>	删除列表中的所有项目。
<code>List.removeItemAt()</code>	删除指定索引处的项目。
<code>List.replaceItemAt()</code>	用其它项目替换指定索引处的项目。
<code>List.setPropertiesAt()</code>	将指定的属性应用到指定的项目。
<code>List.sortItems()</code>	按照指定的比较函数对列表中的项目进行排序。
<code>List.sortItemsBy()</code>	按照指定的属性对列表中的项目进行排序。

## Tree 类的属性摘要

下表列出了 **Tree** 类的属性。

属性	说明
<code>Tree.dataProvider</code>	指定 XML 数据源。
<code>Tree.firstVisibleNode</code>	指定显示在最顶层的第一个节点。
<code>Tree.selectedNode</code>	指定 <b>Tree</b> 实例中的一个选定节点。
<code>Tree.selectedNodes</code>	指定 <b>Tree</b> 实例中的多个选定节点。

## 从 UIObject 类继承的属性

下表列出了 **Tree** 类从 **UIObject** 类继承的属性。从 **Tree** 对象访问这些属性时，请使用 *TreeInstance.propertyName* 的形式。

属性	说明
<code>UIObject.bottom</code>	只读；对象的底边缘位置（相对于其父对象的底边缘）。
<code>UIObject.height</code>	只读；对象的高度，以像素为单位。
<code>UIObject.left</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.right</code>	只读；对象的右边缘位置（相对于其父对象的右边缘）。

属性	说明
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	只读；对象上边缘的位置（相对于其父对象）。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

## 从 UIComponent 类继承的属性

下表列出了 **Tree** 类从 **UIComponent** 类继承的属性。从 **Tree** 对象访问这些属性时，请使用 *TreeInstance.propertyName* 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## 从 List 类继承的属性

下表列出了 **Tree** 类从 **List** 类继承的属性。从 **Tree** 对象访问这些属性时，请使用 *TreeInstance.propertyName* 的形式。

属性	说明
<code>List.cellRenderer</code>	指定要使用的类或元件以显示列表的每一行。
<code>List.dataProvider</code>	列表项目的来源。
<code>List.hPosition</code>	列表的水平位置。
<code>List.hScrollPolicy</code>	指示是 ("on") 否 ("off") 显示水平滚动条。
<code>List.iconField</code>	各项目中用于指定图标的数据项。
<code>List.iconFunction</code>	一个函数，它确定要使用的图标。
<code>List.labelField</code>	指定各项目中用作标签文本的数据项。
<code>List.labelFunction</code>	一个函数，它确定各个项目的哪些字段要用作标签文本。
<code>List.length</code>	列表中的项目数。该属性为只读。
<code>List.maxHPosition</code>	当将 <code>List.hScrollPolicy</code> 设置为 "on" 时，列表可以向右滚动的像素数目。



属性	说明
<code>List.multipleSelection</code>	指示列表中是 (true) 否 (false) 允许多选。
<code>List.rowCount</code>	列表中至少可以看到一部分的行数。
<code>List.rowHeight</code>	列表中每行的像素高度。
<code>List.selectable</code>	指示列表是 (true) 否 (false) 为可选择列表。
<code>List.selectedIndex</code>	单选列表中的选择索引。
<code>List.selectedIndices</code>	多选列表中的已选择项目的数组。
<code>List.selectedItem</code>	单选列表中的已选择项目。该属性为只读。
<code>List.selectedItems</code>	多选列表中的已选择的项目对象。该属性为只读。
<code>List.vPosition</code>	滚动列表，以便使最顶部可见的项目为指定的数。
<code>List.vScrollPolicy</code>	指示是显示 ("on")、不显示 ("off") 还是在需要时显示 ("auto") 垂直滚动条。

## Tree 类的事件摘要

下表列出了 `Tree` 类的事件。

事件	说明
<code>Tree.nodeClose</code>	在用户关闭节点时广播。
<code>Tree.nodeOpen</code>	在用户打开节点时广播。

## 从 `UIObject` 类继承的事件

下表列出了 `Tree` 类从 `UIObject` 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UIComponent 类继承的事件

下表列出了 Tree 类从 UIComponent 类继承的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## 从 List 类继承的事件

下表列出了 Tree 类从 List 类继承的事件。

事件	说明
<code>List.change</code>	只要用户交互造成选择更改就广播。
<code>List.itemRollOut</code>	当指针在列表项上滑过然后又滑离时广播。
<code>List.itemRollOver</code>	当指针滑过列表项时进行广播。
<code>List.scroll</code>	滚动列表时，进行广播。

# Tree.addTreeNode()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
treeInstance.addTreeNode(label [, data])
```

用法 2:

```
treeInstance.addTreeNode(child)
```

## 参数

*label* 一个显示节点或带有标签字段的对象的字符串（或由 *labelField* 属性指定的任何标签字段名称）。

*data* 一个与该节点关联的任何类型的对象。此参数是可选的。

*child* 任何 **XMLNode** 对象。

## 返回

添加的 XML 节点。

## 说明

方法；将子节点添加到树。该节点或者是根据 *label* 和 *data* 参数中提供的信息构建的（用法 1），或者是根据作为一个 **XMLNode** 对象的预先生成的子节点构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

调用此方法将刷新视图。

## 示例

以下示例创建各自分别具有一个节点的两个 **Tree** 组件：1st Local Folders 和 2nd Local Folders。然后，该示例将树节点 (2nd Local Folders) 从第二个 **Tree** 添加到第一个 **Tree**，并且添加一个新节点 Inbox。

首先必须将组件添加到文档库中（通过将 **Tree** 组件拖到舞台上，然后删除该组件）；然后将以下代码添加到第一帧中。



对于本示例，首先尝试在去掉末尾的两条 `addTreeNode()` 语句的情况下运行；然后尝试运行完整的示例。

```
/**
 * 要求:
 * - 库中有 Tree 组件
 */

import mx.controls.Tree;

this.createClassObject(Tree, "first_tr", 10);
first_tr.setSize(200, 100);

this.createClassObject(Tree, "second_tr", 20);
second_tr.setSize(200, 100);
second_tr.move(0, 120);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node>");
first_tr.dataProvider = trDP_xml;
```

```
var trDP2_xml:XML = new XML("<node label='2nd Local Folders'><node
 label='Outbox' data='0' /><node label='Outbox' data='1' /></node>");
second_tr.dataProvider = trDP2_xml;

// 从 second_tr 向 first_tr 添加节点。
first_tr.addTreeNode(second_tr.getTreeNodeAt(0));

// 向 first_tr 中添加节点。
first_tr.addTreeNode("Inbox", "data");
```

## Tree.addTreeNodeAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

用法 1:

```
treeInstance.addTreeNodeAt(index, label [, data])
```

用法 2:

```
treeInstance.addTreeNodeAt(index, child)
```

### 参数

*index* 从零开始的索引位置（在多个子节点之间），应在该处添加节点。

*label* 一个字符串，它包含要添加的节点的名称。

*data* 一个与该节点关联的任何类型的对象。此参数是可选的。

*child* 任何 **XMLNode** 对象。

### 返回

添加的 XML 节点。

### 说明

方法：在树的指定位置添加节点。该节点或者是通过 *label* 和 *data* 参数中提供的信息构建的（用法 1），或者通过预先生成的 **XMLNode** 对象构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

调用此方法将刷新视图。

## 示例

以下示例创建分别具有一个节点的两个 **Tree** 组件: 1st Local Folders 和 2nd Local Folders。它使用 `addTreeNodeAt()` 的第一种用法向第一个 **Tree** 中添加一个新节点 `Inbox`。然后使用 `addTreeNodeAt()` 的第二种用法将第一个节点 (`getTreeNodeAt(0)`) 从第二个 **Tree** 添加到第一个 **Tree** 中。

首先必须将组件添加到文档库中 (通过将 **Tree** 组件拖到舞台上, 然后删除该组件); 然后将以下代码添加到第一帧中。

**提示**

对于本示例, 首先尝试在去掉末尾的两条 `addTreeNodeAt()` 语句的情况下运行; 然后尝试运行完整的示例。

```
/**
 * 要求:
 * - 库中有 Tree 组件
 */

import mx.controls.Tree;

this.createClassObject(Tree, "first_tr", 10);
first_tr.setSize(200, 100);

this.createClassObject(Tree, "second_tr", 20);
second_tr.setSize(200, 100);
second_tr.move(0, 120);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0'><node label='Outbox' data='1'></node></node>");
first_tr.dataProvider = trDP_xml;

var trDP2_xml:XML = new XML("<node label='2nd Local Folders'><node
 label='Outbox' data='0'><node label='Outbox' data='1'></node></node>");
second_tr.dataProvider = trDP2_xml;

// 向 first_tr 中添加节点。
first_tr.addTreeNodeAt(1, "Inbox", "data");
// 从 second_tr 向 first_tr 添加节点。
first_tr.addTreeNodeAt(2, second_tr.getTreeNodeAt(0));
```

# Tree.dataProvider

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`treeInstance.dataProvider`

## 说明

属性： XML 或字符串。如果 `dataProvider` 是 XML 对象，则直接将它添加到树。如果 `dataProvider` 是字符串，则它必须包含由树读取并被转换为 XML 对象的有效 XML。

可以在运行时从外部源加载 XML，或者在创作时在 Flash 中创建它。要创建 XML，可以使用 `TreeDataProvider` 的方法或内置的 `ActionScript XML` 类的方法和属性。也可以创建包含 XML 的字符串。

在与 `Tree` 组件相同的帧上的 XML 对象会自动包含 `TreeDataProvider` 的方法和属性。可以使用 `ActionScript XML` 或 `XMLNode` 对象。

## 示例

以下示例使用 `dataProvider` 属性将 XML 文件的内容添加到 `Tree` 组件的 `my_tr` 实例中：

首先必须将一个 `Tree` 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有 Tree 组件（实例名称: my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML();
trDP_xml.ignoreWhite = true;
trDP_xml.onLoad = function(success:Boolean){
 my_tr.dataProvider = trDP_xml.firstChild;
}
trDP_xml.load("http://www.flash-mx.com/mm/xml/tree.xml");
```



大多数 XML 文件都包含空白。如果想让 Flash 忽略空白，必须将 `XML.ignoreWhite` 属性设置为 `true`。

# Tree.firstVisibleNode

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.firstVisibleNode = someNode
```

## 说明

属性；表示显示在树顶层的第一个节点。利用该属性可将树的显示区滚动到所需位置。如果指定的节点 *someNode* 位于某个未展开的节点下，则对 *firstVisibleNode* 进行的设置将无效。默认值是第一个可见节点，或者如果没有可见的节点，则默认值是 *undefined*。此属性的值是 **XMLNode** 对象。

此属性是 *List.vPosition* 属性的一个相似属性。

## 示例

以下示例填充具有六个节点（通过 XML 文本字符串创建）的名为 *my\_tr* 的 **Tree** 组件。通过将最后一个节点（相对位置为 5）分配给 *firstVisibleNode* 属性，使 **Tree** 中的最后一个节点可见。通过将 5 更改为从 0 到 4 的其它值，可以使其它节点可见。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 *my\_tr*；然后将以下代码添加到第一帧中。

```
/**
```

```
 要求:
```

```
 - 舞台上 有 Tree 组件（实例名称: my_tr）
*/
```

```
var my_tr:mx.controls.Tree;
```

```
my_tr.setSize(200, 100);
```

```
var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='0' /><node
 label='Outbox' data='1' /></node><node label='3rd Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='4th Local Folders'><node label='Inbox' data='0' /><node
 label='Outbox' data='1' /></node><node label='5th Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='6th Local Folders'><node label='Inbox' data='0' /><node
 label='Outbox' data='1' /></node></node>");
```

```
my_tr.dataProvider = trDP_xml;

// 将最后一个节点设置为可见节点。
my_tr.firstVisibleNode = my_tr.getTreeNodeAt(5);
```

# Tree.getDisplayIndex()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.getDisplayIndex(node)
```

## 参数

*node* 一个 XMLNode 对象。

## 返回

所指定节点的索引，或者如果当前未显示该节点，则为 undefined。

## 说明

方法：返回在 *node* 参数中指定的节点的显示索引。

显示索引指示项目在树窗口中可见项目的列表中的位置。例如，任何关闭的节点的子项都不在显示索引中。显示索引列表从 **0** 开始并继续到所有可见项，而与父项无关。换言之，显示索引是显示的行从 **0** 开始的行编号。

## 示例

以下示例向 **Tree** 中添加六个节点，并调用 `getDisplayIndex()` 以显示用户所选节点的位置。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有 Tree 组件（实例名称：my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 140);
```



```

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='0' /><node
 label='Outbox' data='1' /></node><node label='3rd Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='4th Local Folders'><node label='Inbox' data='0' /><node
 label='Outbox' data='1' /></node><node label='5th Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='6th Local Folders'><node label='Inbox' data='0' /><node
 label='Outbox' data='1' /></node>");
my_tr.dataProvider = trDP_xml;
my_tr.firstVisibleNode = my_tr.getTreeNodeAt(0);

my_tr.addEventListener("change", listChanged);
function listChanged(evt_obj:Object) {
 trace(my_tr.getDisplayIndex(evt_obj.target.selectedNode));
}

```

## Tree.getIsBranch()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
treeInstance.getIsBranch(node)
```

### 参数

*node* 一个 XMLNode 对象。

### 返回

一个布尔值，指示节点是 (true) 否 (false) 为分支。

### 说明

方法；指示指定的节点是否具有文件夹图标和展开箭头（是否为分支）。在向该节点添加子项时会自动设置此方法。只需调用 `Tree.setIsBranch()` 即可创建空的文件夹。

## 示例

以下示例在树中创建两个节点，并调用 `isBranch()` 以确定第二个节点是否为分支。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求:
 * - 舞台上有一个 Tree 组件（实例名称: my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

var isBranch:Boolean = my_tr.getIsBranch(my_tr.getTreeNodeAt(1));
trace("2nd node is a branch: " + isBranch);
```

## 另请参见

[Tree.setIsBranch\(\)](#)

# Tree.getIsOpen()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.getIsOpen(node)
```

## 参数

*node* 一个 `XMLNode` 对象。

## 返回

一个布尔值，指示树是打开 (`true`) 还是关闭 (`false`)。

## 说明

方法：指示指定的节点是打开还是关闭。



在每次节点打开和关闭时会显示索引的更改。

## 示例

以下示例向名为 `my_tr` 的 **Tree** 中添加两个节点，并打开第二个节点，然后调用 `getIsOpen()` 以显示第二个节点的状态。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有 Tree 组件（实例名称：my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;
my_tr.setIsOpen(my_tr.getTreeNodeAt(1), true);
var isOpen:Boolean = my_tr.getIsOpen(my_tr.getTreeNodeAt(1));
trace("2nd node is a open: " + isOpen);
```

# Tree.getNodeDisplayedAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`treeInstance.getNodeDisplayedAt(index)`

## 参数

*index* 一个整数，表示树的可查看区域中的显示位置。此数字从零开始；在第一个位置的节点为 0，第二个位置为 1，依此类推。

## 返回

指定的 `XMLNode` 对象。

## 说明

方法：将树的某个显示索引映射到在该索引处显示的节点上。例如，如果树的第五行显示了在层次结构中深居第八层的节点，则通过调用 `getNodeDisplayedAt(4)` 会返回该节点。

显示索引是可以在树窗口中查看的项的数组。例如，任何关闭的节点的子项都不在显示索引中。显示索引从 0 开始并继续到可见项，而与父项无关。换言之，显示索引是显示的行从 0 开始的行编号。



在每次节点打开和关闭时会显示索引的更改。

## 示例

以下示例向名为 `my_tr` 的 `Tree` 实例中添加一个节点，然后调用 `getNodeDisplayedAt()` 方法以检索显示位置为 0（零）的节点。该示例调用 `trace()` 函数以显示该节点。

首先必须将一个 `Tree` 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有一个 Tree 组件（实例名称：my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label=\"1st Local Folders\"><node
 label=\"Inbox\" data=\"0\" /></>");
my_tr.dataProvider = trDP_xml;

trace(my_tr.getNodeDisplayedAt(0));
```

# Tree.getTreeNodeAt()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.getTreeNodeAt(index)
```

## 参数

*index* 节点的索引号。

## 返回

XMLNode 对象。

## 说明

方法；返回在 myTree 的根上的指定节点。

## 示例

以下示例向名为 my\_tr 的 **Tree** 实例中添加一个节点，然后调用 getTreeNodeAt() 方法以返回树中的第一个节点。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 my\_tr ；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上 有 Tree 组件（实例名称: my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label=\"1st Local Folders\"><node
 label=\"Inbox\" data=\"0\" /><node label=\"Outbox\" data=\"1\" /></
 node>");
my_tr.dataProvider = trDP_xml;

trace(my_tr.getTreeNodeAt(0));
```

# Tree.nodeClose

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
var listenerObject:Object = new Object();
listenerObject.nodeClose = function(eventObject:Object) {
 // 在此处插入您的代码。
};
treeInstance.addEventListener("nodeClose", listenerObject);
```

## 说明

事件：在用户关闭 **Tree** 组件的节点时，向所有注册的侦听器广播。

第 2 版组件使用调度程序 / 侦听器事件模型。**Tree** 组件会在其一个节点被单击关闭时广播 `nodeClose` 事件，而该事件由附加到您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）进行处理。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。`Tree.nodeClose` 事件的事件对象还有一个附加属性：`node`（已关闭的 XML 节点）。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例向 **Tree** 实例 `my_tr` 中添加两个节点，然后创建两个侦听器对象，一个用于 `nodeOpen` 事件，一个用于 `nodeClose` 事件。当这两个事件发生时，侦听器函数会使用 `trace` 语句在“输出”面板中显示该事件和受影响的节点。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上要有 Tree 组件（实例名称: my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);
```

```

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0'/><node label='Outbox' data='1'/></node><node
 label='2nd Local Folders'><node label='Inbox' data='2'/><node
 label='Outbox' data='3'/></node>");
my_tr.dataProvider = trDP_xml;

// 创建侦听器对象。
var trListener:Object = new Object();
trListener.nodeOpen = function(evt_obj:Object){
 trace("Node opened\n" + evt_obj.node);
 trace("\n");
}
trListener.nodeClose = function(evt_obj:Object){
 trace("Node closed\n" + evt_obj.node);
 trace("\n");
}
// 添加侦听器。
my_tr.addEventListener("nodeOpen", trListener);
my_tr.addEventListener("nodeClose", trListener);

```

## Tree.nodeOpen

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```

var listenerObject:Object = new Object();
listenerObject.nodeOpen = function(eventObject:Object) {
 // 在此处插入您的代码。
};
treeInstance.addEventListener("nodeOpen", listenerObject);

```

### 说明

事件：在用户打开 **Tree** 组件上的节点时，向所有注册的侦听器广播。

第 2 版组件使用调度程序 / 侦听器事件模型。**Tree** 组件会在用户单击打开一个节点时调度 nodeOpen 事件，该事件由附加到您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）进行处理。您需要调用 addEventListener() 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。Tree.nodeOpen 事件的事件对象还有一个附加属性：node（已打开的 XML 节点）。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例向 **Tree** 实例 `my_tr` 中添加两个节点，然后创建两个侦听器对象，一个用于 `nodeOpen` 事件，一个用于 `nodeClose` 事件。当这两个事件发生时，侦听器函数会调用 **trace** 语句在“输出”面板中显示该事件和受影响的节点。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 要求:
 - 舞台上 有 Tree 组件 (实例名称: my_tr)
*/
var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// 创建侦听器对象。
var trListener:Object = new Object();
trListener.nodeOpen = function(evt_obj:Object){
 trace("Node opened\n" + evt_obj.node);
 trace("\n");
}
trListener.nodeClose = function(evt_obj:Object){
 trace("Node closed\n" + evt_obj.node);
 trace("\n");
}
// 添加侦听器。
my_tr.addEventListener("nodeOpen", trListener);
my_tr.addEventListener("nodeClose", trListener);
```



# Tree.refresh()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.refresh()
```

## 参数

无。

## 返回

无。

## 说明

方法；更新树。

## 示例

以下示例向名为 **my\_tr** 的 **Tree** 实例中添加一个节点，并为“刷新”按钮和“全部删除”按钮创建侦听器。如果数据提供程序的 **XML** 源发生更改，用户可以单击“刷新”按钮，然后代码将调用 `refresh()` 方法以更新树内容。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`。之后添加一个名为 `refresh_button` 的按钮，然后将以下代码添加到主时间轴的第一帧中：

```
/**
 要求：
 - 舞台上有 Tree 组件（实例名称: my_tr）
 - 舞台上有 Button 组件（实例名称: refresh_button）
*/

var my_tr:mx.controls.Tree;
var refresh_button:mx.controls.Button;
var removeAll_button:mx.controls.Button;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML();
trDP_xml.ignoreWhite = true;
trDP_xml.onLoad = function() {
 my_tr.dataProvider = this.firstChild;
};
```

```
trDP_xml.load("http://yourXMLsourcehere");

function refreshListener(evt_obj:Object):Void {
 my_tr.refresh();
}
refresh_button.addEventListener("click", refreshListener);
```

## Tree.removeAll()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
treeInstance.removeAll()
```

### 参数

无。

### 返回

无。

### 说明

方法：删除所有节点并刷新树。

### 示例

以下示例向名为 **my\_tr** 的 **Tree** 实例中添加一个节点，并为“全部删除”按钮创建一个侦听器。用户单击“全部删除”按钮时，代码会调用 `removeAll()` 方法删除树中的所有节点。

首先将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`，之后添加一个名为 `removeAll_button` 的按钮；然后将以下代码添加到第一帧中。

```
/**
 要求：
 - 舞台上有 Tree 组件（实例名称: my_tr）
 - 舞台上有 Button 组件（实例名称: refresh_button）
 - 舞台上有 Button 组件（实例名称: removeAll_button）
*/

var my_tr:mx.controls.Tree;
var removeAll_button:mx.controls.Button;

my_tr.setSize(200, 100);
```

```

var trDP_xml:XML = new XML();
trDP_xml.ignoreWhite = true;
trDP_xml.onLoad = function() {
 my_tr.dataProvider = this.firstChild;
};
trDP_xml.load("http://www.flash-mx.com/mm/xml/tree.xml");

function removeAllListener(evt_obj:Object):Void {
 my_tr.removeAll();
}
removeAll_button.addEventListener("click", removeAllListener);

```

## Tree.removeTreeNodeAt()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
treeInstance.removeTreeNodeAt(index)
```

### 参数

*index* 树的子项的索引号。树的每个子项均按其创建顺序获得了一个从零开始的索引。

### 返回

XMLNode 对象；或者如果出错，则返回 undefined。

### 说明

方法；删除树的根上的节点（由其索引位置指定）并刷新树。

### 示例

以下示例向 Tree 实例中添加两个节点，并为树上的 change 事件创建一个侦听器。发生 change 事件时，侦听器函数会调用 removeTreeNodeAt() 方法删除树中的所选节点。

首先将一个 Tree 组件实例添加到舞台上，并将其命名为 my\_tr，然后将以下代码添加到第一帧中。

```

/**
 要求：
 - 舞台上有一个 Tree 组件（实例名称：my_tr）
*/

```

```

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

var treeListener:Object = new Object();
treeListener.change = function (evt_obj:Object) {
 my_tr.removeTreeNodeAt(my_tr.selectedIndex);
}
my_tr.addEventListener("change", treeListener);

```

## Tree.selectedNode

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

treeInstance.selectedNode

### 说明

属性；指定树实例中的一个选定节点。

### 示例

以下示例向 **Tree** 实例中添加两个节点，并将第二个节点设置为选中状态。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```

/**
 * 要求：
 * - 舞台上 有 Tree 组件（实例名称: my_tr）
 */
var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

```

```
var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// 选择第二个节点。
my_tr.selectedNode = my_tr.getTreeNodeAt(1);
```

另请参见

[Tree.selectedNodes](#)

## Tree.selectedNodes

可用性

Flash Player 6 (6.0.79.0)。

版本

Flash MX Professional 2004。

用法

```
treeInstance.selectedNodes
```

说明

属性；指定树实例中的多个选定节点。

示例

以下示例向 **Tree** 实例中添加三个节点，并将前两个节点设置为选中状态。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 要求：
 - 舞台上 有 Tree 组件（实例名称: my_tr）
*/

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node><node label='3rd Local Folders'><node
 label='Inbox' data='2' /><node label='Outbox' data='3' /></node>");
```

```
my_tr.dataProvider = trDP_xml;

// 允许多个选择。
my_tr.multipleSelection = true;

// 选择第一个和第二个节点。
my_tr.selectedNodes = [my_tr.getTreeNodeAt(0), my_tr.getTreeNodeAt(1)];
```

**另请参见**

[Tree.selectedNode](#)

## Tree.setIcon()

**可用性**

Flash Player 6 (6.0.79.0)。

**版本**

Flash MX Professional 2004。

**用法**

```
treeInstance.setIcon(node, linkID [, linkID2])
```

**参数**

*node* 一个 XML 节点。

*linkID* 元件的链接标识符，该元件将被用作节点旁的图标。此参数用于叶节点和分支节点的关闭状态。

*linkID2* 对于分支节点，指示被用作图标的某元件的链接标识符，它表示节点的打开状态。此参数是可选的。

**返回**

无。

**说明**

方法：为指定的节点指定图标。对于叶节点，此方法采用一个 **ID** 参数 (*linkID*)；对于分支节点，此方法采用两个 **ID** 参数 (*linkID* 和 *linkID2*，指示关闭和打开图标)。对于叶节点，第二个参数被忽略。对于分支节点，如果省略 *linkID2*，则图标同时用作关闭和打开状态。

## 示例

以下示例向名为 `my_tr` 的 **Tree** 实例中添加两个节点，并调用 `setIcon()` 函数为第二个节点在库中指定链接 **ID** 为 `imageIcon` 的图标。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`，接着将链接 **ID** 为 `imageIcon` 的图标添加到库中；然后将以下代码添加到第一帧中。

```
/**
 要求:
 - 舞台上有 Tree 组件 (实例名称: my_tr)
 - 库项目具有链接 ID ImageIcon
*/

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// 将 movieclip 设置为第二个节点的图标。
my_tr.setIcon(my_tr.getTreeNodeAt(1), "imageIcon");
```

# Tree.setIsBranch()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.setIsBranch(node, isBranch)
```

## 参数

*node* 一个 XML 节点。

*isBranch* 一个布尔值，它指示该节点是 (true) 否 (false) 为分支。

## 返回

无。

## 说明

方法：指定节点是否具有文件夹图标和展开箭头，以及是否具有子项或者可能具有子项。如果节点具有子项，它会自动设置为分支；当您想创建空文件夹时，只需调用 `setIsBranch()` 即可。如果在用户打开文件夹时只希望加载子节点，则您可能要创建尚未拥有子项的分枝。

调用 `setIsBranch()` 会刷新所有视图。

## 示例

以下示例向名为 `my_tr` 的 **Tree** 实例中添加单个节点，并调用 `setIsBranch()` 使其成为不具有子节点的分支。

首先必须将一个 **Tree** 组件实例添加到舞台上，并将其命名为 `my_tr`；然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 舞台上有 Tree 组件（实例名称：my_tr）
 */

var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='Inbox' data='0'/>");
my_tr.dataProvider = trDP_xml;

// 将第一个节点设置为分支。
my_tr.setIsBranch(my_tr.getTreeNodeAt(0), true);
```

# Tree.setIsOpen()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
treeInstance.setIsOpen(node, open [, animate [, fireEvent]])
```



## 参数

*node* 一个 XML 节点。

*open* 一个布尔值，用于打开一个节点 (true) 或者关闭一个节点 (false)。

*animate* 一个布尔值，它确定是 (true) 否 (false) 以动画显示打开过渡过程。此参数是可选的。

*fireEvent* 一个布尔值，它确定打开或关闭树节点时是 (true) 否 (false) 调度 *nodeOpen* 和 *nodeClose* 事件。此参数是可选的。默认值为 false。

## 返回

无。

## 说明

方法：打开或关闭节点。

## 示例

以下示例在名为 *my\_tr* 的 **Tree** 实例中创建两个节点，并调用 *setIsOpen()* 方法打开第二个节点。

```
/**
 要求：
 - 舞台上有 Tree 组件（实例名称: my_tr）
*/
var my_tr:mx.controls.Tree;

my_tr.setSize(200, 100);

var trDP_xml:XML = new XML("<node label='1st Local Folders'><node
 label='Inbox' data='0' /><node label='Outbox' data='1' /></node><node
 label='2nd Local Folders'><node label='Inbox' data='2' /><node
 label='Outbox' data='3' /></node>");
my_tr.dataProvider = trDP_xml;

// 将第二个节点设置为打开状态。
my_tr.setIsOpen(my_tr.getTreeNodeAt(1), true);
```



继承 （根类）

ActionScript 类名称 `mx.transitions.Tween`

**Tween** 类使您能够使用 **ActionScript** 指定目标影片剪辑的属性在若干帧数或秒数中为补间动画，从而在舞台上轻松地对影片剪辑进行移动、调整大小和淡入淡出操作。**Tween** 类还使您能够指定各种缓动方法。缓动是指动画过程中的渐进加速或减速，它会使您的动画看起来更逼真。例如，您创建的下拉列表组件的选项可能在接近动画的开始处时随着选项的出现而逐渐地增加速度，而在列表展开时速度减慢，直到结束动画时完全停止。**Flash** 提供了很多缓动方法（包括这种加速和减速的等式），它们会相应地更改缓动动画。

此外，**Tween** 类还调用事件处理函数，以便您的代码在动画开始、结束、恢复或增加其补间属性值时能够进行响应。例如，在第一个补间调用其 `Tween.onMotionStopped` 事件处理函数（指示第一个补间已停止）时，可以开始另一个补间动画。

## Tween 类的方法摘要

下表列出了 **Tween** 类的方法：

方法	说明
<code>Tween.yoyo()</code>	指示补间动画从当前值继续到一个新值。
<code>Tween.ffforward()</code>	将补间动画直接快进到动画结尾。
<code>Tween.nextFrame()</code>	将补间动画快进到下一帧。
<code>Tween.prevFrame()</code>	指示补间动画进到当前帧的前一帧。
<code>Tween.resume()</code>	从动画中的停止点继续运行补间动画。
<code>Tween.rewind()</code>	将补间动画后退到补间动画的开头。
<code>Tween.start()</code>	从头开始补间动画。
<code>Tween.stop()</code>	在当前位置停止补间动画。
<code>Tween.toString()</code>	返回类名称，“[Tween]”。
<code>Tween.yoyo()</code>	指示补间动画按与其补间属性最后一次增加的方向相反的方向播放。

# Tween 类的属性摘要

下表列出了 Tween 类的属性。

属性	说明
<code>Tween.duration</code>	以帧或秒为单位的补间动画的持续时间。只读。
<code>Tween.finish</code>	补间动画结尾的最后一个补间值。只读。
<code>Tween.FPS</code>	补间动画的每秒的帧数。只读。
<code>Tween.position</code>	目标影片剪辑补间属性的当前值。只读。
<code>Tween.time</code>	动画持续时间内的当前时间。只读。

# Tween 类的事件处理函数摘要

下表列出了 Tween 类的事件处理函数。

事件	说明
<code>Tween.onMotionChanged</code>	事件处理函数；在正执行动画的补间对象的属性每次发生更改时调用。
<code>Tween.onMotionFinished</code>	事件处理函数；在 Tween 对象结束其动画时调用。
<code>Tween.onMotionResumed</code>	事件处理函数；在调用 Tween.resume() 方法时调用，从而使补间动画继续。
<code>Tween.onMotionStarted</code>	事件处理函数；在调用 Tween.start() 方法时调用，从而使补间动画开始。
<code>Tween.onMotionStopped</code>	事件处理函数；在调用 Tween.stop() 方法时调用，从而使补间动画停止。

# 使用 Tween 类

若要使用 Tween 类的方法和属性，需要使用 new 运算符创建该类的新实例。例如，若要将一个 tween 实例应用于某个影片剪辑对象（名为 myMovieClip\_mc），可使用以下代码创建 mx.transitions.Tween 的新实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(myMovieClip_mc, "_x",
 mx.transitions.easing.Elastic.easeOut, 0, 300, 3, true);
```

## Tween 类参数

当创建 Tween 类的新实例时，需要传递一些参数。您必须指明目标影片剪辑对象、补间将影响的影片剪辑的属性、补间的范围和用于计算补间的属性的缓动方法。

**mx.transitions.Tween** 类的构造函数有以下参数名称和类型：

```
Tween(obj:Object, prop:String, func:Function, begin:Number, finish:Number,
 duration:Number, useSeconds:Boolean)
```

**obj**， Tween 实例所面对的影片剪辑对象。

**prop**， 要用值补间的 obj 中属性的字符串名称。

**func**， 计算补间对象属性值的缓动效果的缓动方法。请参见第 1209 页的“关于缓动类和方法”。

**begin**， 一个指示 prop（要补间的目标对象属性）的开始值的数字。

**finish**， 一个指示 prop（要补间的目标对象属性）的结束值的数字。

**duration**， 一个数字，指示补间动画的时间长度。如果省略，**duration** 会默认设置成无穷大。

**useSeconds**， 一个布尔值，如果相对于 duration 参数中指定的值为 true，则表示使用秒；如果为 false，则表示使用帧。

## 关于缓动类和方法

当创建 Tween 类的实例时，使用 func 参数指定提供缓动计算的函数或方法。Flash 提供了五个缓动类，每个缓动类都有三个方法，这些方法指示过渡动画的以下哪一部分应用缓动效果：在动画的开始、结尾或开始和结尾。此外，带有 easeNone 方法的 None 缓动类可用于指示未使用缓动。

以下类和组件使用缓动类和方法：

- 对补间动画产生缓动效果的 **mx.transitions.Tween** 类
- 对过渡产生缓动效果的 **mx.transitions.TransitionManager** 类。请参见第 1139 页的第 48 章“**TransitionManager** 类”。
- 第 2 版 Macromedia Component Architecture 中的一些组件。请参见第 1210 页的“将缓动方法应用于组件”。

下表描述了这六个缓动计算类。

缓动类	说明
Back	在过渡范围外的一端或两端扩展动画一次，以产生从其范围外回拉的效果。
Bounce	在过渡范围的一端或两端内添加弹跳效果。弹跳数与持续时间相关，持续时间越长，弹跳数越多。
Elastic	添加一端或两端超出过渡范围的弹性效果。弹性量不受持续时间影响。
Regular	在一端或两端添加较慢的运动。此功能使您能够添加加速效果、减速效果或这两种效果。
Strong	在一端或两端添加较慢的运动。此效果类似于 Regular 缓动类，但它更明显。
None	添加从开始到结尾无任何减速或加速效果的相同的运动。此过渡也称为线性过渡。

这六种缓动计算类的每一种都有三个缓动方法，它们指明缓动效果应用于动画的哪个部分。此外，None 缓动类还有第四个缓动方法：**easeNone**。下表中描述了这些缓动方法：

方法	说明
easeIn	在过渡的开始提供缓动效果。
easeOut	在过渡的结尾提供缓动效果。
easeInOut	在过渡的开始和结尾提供缓动效果。
easeNone	指明不使用缓动计算。只在 None 缓动类中提供。

## 将缓动方法应用于组件

各种缓动方法的另一种用途是将它们应用到 Macromedia Component Architecture 第 2 版组件。您只能将缓动方法应用到以下第 2 版组件: Accordion、ComboBox、DataGrid、List、Menu 和 Tree。每种组件都使用缓动方法进行不同的自定义。例如，Accordion、ComboBox 和 Tree 组件使您能够选择缓动类来用于它们各自的打开和关闭动画。相比之下，Menu 组件只能使您定义动画持续的毫秒数。

## 将缓动方法应用于 Accordion 组件

本节描述如何将 **Accordion** 组件添加到 **Flash** 文档中、添加一些子幻灯片以及更改默认的缓动方法和持续时间。如果打算在项目中使用这段代码，需要减小 `openDuration` 属性值，以避免用户因打开和关闭 **Accordion** 组件的子组件时动画太慢而感到厌烦。

将不同的缓动方法应用到 **Accordion** 组件：

1. 创建一个新的 **Flash** 文件并将其另存为 `accordion.fla`。
2. 将 **Accordion** 组件的副本拖到舞台上。
3. 打开“属性”检查器，并在“实例名称”文本框中键入 `my_acc`。
4. 在图层 1 上插入一个新图层，将其命名为 `actions`。
5. 在 `actions` 图层的第 1 帧中添加以下 **ActionScript** 代码：

```
import mx.core.View;
import mx.transitions.easing.*;
my_acc.createChild(View, "studio_view", {label:"Studio"});
my_acc.createChild(View, "dreamweaver_view", {label:"Dreamweaver"});
my_acc.createChild(View, "flash_view", {label:"Flash"});
my_acc.createChild(View, "coldfusion_view", {label:"ColdFusion"});
my_acc.createChild(View, "contribute_view", {label:"Contribute"});
my_acc.setStyle("openEasing", Bounce.easeOut);
my_acc.setStyle("openDuration", 3500);
```

这段代码导入缓动类，因此您可以键入 `Bounce.easeOut` 而无需使用如 `mx.transitions.easing.Bounce.easeOut` 之类的完全限定名称来引用各类。然后，代码将五个新子窗格添加到 **Accordion** 组件中（**Studio**、**Dreamweaver**、**Flash**、**ColdFusion** 和 **Contribute**）。最后两行代码将缓动样式从默认缓动方法设置为 `Bounce.easeOut`，然后将动画长度设置为 **3500 毫秒（3.5 秒）**。

6. 保存文档，然后选择“控制”>“测试影片”以在测试环境中预览此文件。

单击不同的标题栏以查看修改的动画以及在每个窗格间切换。

如果想增加动画的速度，请将 `openDuration` 从 **3500 毫秒**减少到一个较小的数字。动画的默认持续时间是 **250 毫秒（四分之一秒）**。

## 将缓动方法应用于 ComboBox 组件

更改 ComboBox 组件上默认缓动方法的过程类似于第 1211 页的“将缓动方法应用于 Accordion 组件”中修改 Accordion 组件动画的示例。在以下示例中，使用 `ActionScript` 在运行时将组件动态地添加到舞台上。

要将缓动方法应用到 ComboBox 组件：

1. 创建一个新的 Flash 文档，并将其另存为 `combobox fla`。
2. 将 ComboBox 组件的副本从“组件”面板拖到当前文档的库中。



该组件将出现在库中（不是舞台上），在运行时可供 SWF 文件使用。

3. 插入一个新的图层，并将其重命名为 `actions`。

确定 `actions` 图层是在层 1 上面。

4. 在 `actions` 图层的第 1 帧中添加以下 `ActionScript` 代码：

```
import mx.transitions.easing.*;
this.createClassObject(mx.controls.ComboBox, "my_cb", 20);
var product_array:Array = new Array("Studio", "Dreamweaver", "Flash",
 "ColdFusion", "Contribute", "Breeze", "Director", "Flex");
my_cb.dataProvider = product_array;
my_cb.move(10, 10);
my_cb.setSize(140, 22);
my_cb.setStyle("openDuration", 2000);
my_cb.setStyle("openEasing", Elastic.easeOut);
```

导入每个缓动方法后（它们出现在代码的第一行），`createClassObject()` 方法将创建 `ComboBox` 组件的实例。代码第二行中的 `this` 关键字是指 SWF 文件的主时间轴。此行代码在运行时将组件放到舞台上，并为其指定实例名称 `my_cb`。

然后，创建一个名为 `product_array` 的数组，其中包括 **Macromedia** 软件列表。在下一行代码中使用该数组将 `dataProvider` 属性设置为产品名称组成的数组。然后使用 `setSize()` 方法来调整该组件实例的大小，将 `openDuration` 设置为 2000 毫秒（2 秒），并将缓动方法更改为 `Elastic.easeOut`。



如前几个示例中那样导入缓动类，它们使您能够使用简化形式的类名称，而不是使用完全限定的类名称 `mx.transitions.easing.Elastic.easeOut`。

5. 保存当前文档，然后选择“控制” > “测试影片”以在测试环境中查看此文档。



- 单击舞台上的 **ComboBox** 组件，以使用指定的缓动类使产品名称下拉列表产生动画效果。



对 **ComboBox** 或 **Accordion** 组件使用缓动方法（如 **Elastic** 或 **Bounce**）时要小心。有些用户可能会感觉，如果选项需要很长时间才能停止移动，以便可以从菜单中读取和选择，这会很混乱。请测试各个应用程序和设置，然后确定缓动方法是增强了 **Flash** 文档的功能，还是减小了。

## 对 **DataGrid** 组件进行动画处理

**Flash 8** 还使您能够在选择组件（如 **DataGrid**、**Tree**、**ComboBox** 或 **List** 组件）中的项时，对所使用的动画进行调整。尽管动画是模糊不清的，但是有些情况下能控制动画的微小细节或者增加动画速度仍然是好的。

将缓动添加到 **DataGrid** 组件中：

- 创建一个新的 **Flash** 文档并将其另存为 **datagrid.fla**。
- 将 **DataGrid** 组件的实例拖到舞台上，并为其指定名称 **my\_dg**。
- 插入一个新的图层，并将其重命名为 **actions**。

确定 **actions** 图层在层 1 上面。

- 在该 **actions** 图层中添加以下 **ActionScript**：

```
import mx.transitions.easing.*;
my_dg.setSize(320, 240);
my_dg.addColumn("product");
my_dg.getColumnAt(0).width = 304;
my_dg.rowHeight = 60;
my_dg.addItem({product:"Studio"});
my_dg.addItem({product:"Dreamweaver"});
my_dg.addItem({product:"Flash"});
my_dg.setStyle("selectionEasing", Elastic.easeInOut);
my_dg.setStyle("selectionDuration", 1000);
```

这段 **ActionScript** 代码将导入缓动类，并将舞台上的组件实例大小更改为 320 像素宽、240 像素高。然后，创建一个名为 **product**（产品）的新列，并将该列的大小更改为 304 像素宽。数据网格本身是 320 像素宽，而滚动条是 16 像素宽，有 304 像素的差值。然后将行高设置为 60 像素，这样会更易于看到缓动动画。

**ActionScript** 的下三行将项目添加到数据网格，这样您就可以单击并看到动画。最后，使用 **setStyle()** 方法设置 **selectionEasing** 和 **selectionDuration** 属性。缓动方法设置为 **Elastic.easeInOut**，**duration** 设置为 1000 毫秒（一秒，是默认值 200 毫秒的五倍）。

5. 保存文档，然后选择“控制”>“测试影片”以在测试环境中查看结果。

当您单击 **DataGrid** 实例中的某个项目时，可以看到该项目使用弹性效果进行渐进和渐出。因为显著地增加了持续时间，应该很容易看到该动画。



您还可以对 **ComboBox**、**List** 和 **Tree** 组件使用同样的属性（`selectionEasing` 和 `selectionDuration`）。

## Tween.continueTo()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

`tweenInstance.continueTo(finish, duration)`

### 参数

*finish*，一个数字，指示要补间的目标对象属性的结束值。

*duration*，如果 `Tween.start()` `useSeconds` 参数设置为 **true**，则为一个指示补间动画的时间长度或帧数的数字；如果设置为 **false**，则不用帧进行度量。有关 `useSeconds` 参数的更多信息，请参见第 1227 页的 `Tween.start()`。

### 返回

无。

### 说明

方法；指示补间动画从当前动画点继续补间到一个新的结束和持续时间点。

### 示例

在此示例中，`onMotionFinished` 事件触发一个处理函数，该函数通过调用 `Tween.continueTo()` 方法通知 **Tween** 实例使用新的 `finish` 和 `duration` 值继续执行动画。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_y",
 mx.transitions.easing.Elastic.easeOut,0, 200, 3, true);
myTween.onMotionFinished = function() {
 var myFinish:Number = 100;
 var myDuration:Number = 5;
 myTween.continueTo(myFinish, myDuration);
};
```

# Tween.duration

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.duration

## 说明

属性（只读）；一个数字，指示以帧或秒为单位的补间动画持续时间。当创建一个新的 **Tween** 实例或者调用 `Tween.yoyo()` 方法时，将该属性设置为参数。

## 示例

以下示例通过获取 `Tween.duration` 属性跟踪 **Tween** 对象的 `duration` 设置。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_y",
 mx.transitions.easing.Strong.easeOut,0, Stage.height, 50, false);
var theDuration:Number = myTween.duration;
trace(theDuration);
```

# Tween.fforward()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.fforward()

## 返回

无。

## 说明

方法；将补间动画直接快进到补间动画的最终值。

## 示例

在此示例中，通过调用 `Tween.fforward()` 将补间动画快进到它的最后值，从而立即触发 `onMotionFinished` 事件。`Tween.onMotionFinished` 事件的处理函数将调用 `Tween.yoyo()` 方法。由于初始动画跳到结尾，因此补间动画显然以 `Tween.yoyo()` 方法的反转效果开始。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.Elastic.easeOut,0, Stage.width - img1_mc._width, 8,
 true);

myTween.fforward();

myTween.onMotionFinished = function() {
 myTween.yoyo();
};
```

# Tween.finish

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance.finish*

## 说明

属性（只读）：一个数字，指示要补间的目标对象属性的结束值。当创建一个新的 **Tween** 实例或者调用 `Tween.yoyo()` 方法时，将该属性设置为参数。

## 示例

以下示例返回一个 **Tween** 实例的 `finish` 当前设置。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_y",
 mx.transitions.easing.Strong.easeOut,0, Stage.height - img1_mc._height,
 50, false);
var myFinish:Number = myTween.finish;
trace(myFinish);
```

# Tween.FPS

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance.FPS*

## 说明

属性：计入补间动画的每秒的帧数。默认情况下，使用当前舞台帧频计算补间动画。设置此属性会重新计算每秒向 Tween.FPS 属性显示的补间属性中的增加数量，而不是当前舞台帧频。设置 Tween.FPS 属性不会更改舞台的实际帧频。



除非首先显式设置 Tween.FPS 属性，否则该属性返回 undefined。

## 示例

以下示例创建两个用两个不同的 FPS 设置所设置的补间动画。并显示这两个 Tween 实例的 FPS 当前设置。此示例需要在舞台上有一个名为 img1\_mc 的影片剪辑实例和一个名为 img2\_mc 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween1:Tween = new Tween(img1_mc, "_y",
 mx.transitions.easing.Strong.easeOut,0, Stage.height - img1_mc._height,
 400, false);
myTween1.FPS = 1;
var myFPS1:Number = myTween1.FPS;
trace("myTween1.FPS:" + myFPS1);

var myTween2:Tween = new Tween(img2_mc, "_y",
 mx.transitions.easing.Strong.easeOut,0, Stage.height - img2_mc._height,
 400, false);
myTween2.FPS = 12;
var myFPS2:Number = myTween2.FPS;
trace("myTween2.FPS:" + myFPS2);
```

# Tween.nextFrame()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.nextFrame()

## 返回

无。

## 说明

方法：将补间动画快进到已停止的动画的下一帧。使用 `Tween.stop()` 方法停止补间动画后，使用该方法可以将补间动画一次快进一帧。



该方法只能用于基于帧的补间。通过将 `useSeconds` 参数设置为 `false`，可以在创建时将补间设置为基于帧。

## 示例

此示例将补间动画应用于影片剪辑 `img1_mc`。通过在 `Tween.onMotionFinished` 事件所触发的处理函数中调用 `Tween.start()` 方法，将从开始点起重复地循环播放动画。单击名为 `forwardByFrame_btn` 的按钮会调用 `Tween.stop()` 方法以停止动画，然后会调用 `Tween.nextFrame()` 方法。在补间动画过程中单击该按钮时，会产生停止动画然后只前进一帧的效果。当创建 **Tween** 实例时，将 `useSeconds` 参数声明为 `false` 以使补间基于帧。此过程是使用 `Tween.nextFrame()` 方法所必需的。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone,0, Stage.width, 60, false);

myTween.onMotionFinished = function() {
 myTween.start();
};

forwardByFrame_btn.onRelease = function() {
 myTween.stop();
 myTween.nextFrame();
};
```

# Tween.onMotionChanged

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
tweenInstance.onMotionChanged = function() {
 // ...
};
```

## 说明

事件处理函数；在正执行动画的补间对象属性每次发生更改时调用。处理该事件允许您的代码在正在补间的目标影片剪辑属性增加到下一个值时做出响应。

## 示例

在此示例中，补间应用于影片剪辑 `img1_mc`。在影片剪辑的 `_x` 属性的补间每次增加时，都将调用 `onMotionChanged` 事件处理函数，并显示一条指示补间影片剪辑新位置的跟踪消息。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.Elastic.easeOut,0, Stage.width-img1_mc._width, 3,
 true);

myTween.onMotionChanged = function() {
 trace(this.position);
};
```

# Tween.onMotionFinished

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
tweenInstance.onMotionFinished = function() {
 // ...
};
```

## 说明

事件处理函数；当动画到达持续时间的结束点时调用。处理该事件允许您的代码在补间动画结束点做出响应。

## 示例

在以下示例中，将补间应用于影片剪辑 `img1_mc`。当补间到达其动画结尾时，它会调用 `onMotionFinished` 事件处理函数，后者调用 `Tween.yoyo()` 方法。因此补间能够在调用 `Tween.yoyo()` 方法倒退动画前完成动画。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.Elastic.easeOut,0, Stage.width-img1_mc._width, 3,
 true);
myTween.FPS = 30;
myTween.onMotionFinished = function() {
 myTween.yoyo();
};
```

# Tween.onMotionResume

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
tweenInstance.onMotionResume = function() {
 // ...
};
```

## 说明

事件处理函数；在调用 `Tween.resume()` 方法时调用。处理该事件允许您的代码在补间动画结束点恢复时做出响应。



## 示例

以下示例将补间动画应用于影片剪辑 `img1_mc`。通过在 `onMotionFinished` 事件处理函数中调用 `Tween.start()` 方法，将从开始点起重复地循环播放动画。单击名为 `stopTween_btn` 的按钮上单击会调用 `Tween.stop()` 方法以在当前点停止补间动画。单击名为 `resumeTween_btn` 的按钮会调用 `Tween.resume()` 方法，以从它的停止点继续执行补间动画。在调用 `Tween.resume()` 方法时，`Tween` 实例会调用 `onMotionResumed` 处理函数。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例、一个名为 `stopTween_btn` 的影片剪辑实例和一个名为 `resumeTween_btn` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone,0, Stage.width, 3, true);

myTween.onMotionFinished = function() {
 myTween.start();
};

myTween.onMotionResumed = function() {
 trace("onMotionResumed");
};

stopTween_btn.onRelease = function() {
 myTween.stop();
};

resumeTween_btn.onRelease = function() {
 myTween.resume();
};
```

# Tween.onMotionStarted

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
tweenInstance.onMotionStarted = function() {
 // ...
};
```

## 说明

事件处理函数；在动画执行过程中或完成后动画再次开始时调用。在补间动画初次开始时，不调用此事件处理函数。调用 `Tween.start()`、`Tween.yoyo()` 或 `Tween.yoyo()` 方法以重新启动结束的动画，或者在动画调用 `onMotionStarted` 事件处理函数时重新启动动画。处理该事件允许您的代码在补间动画初次开始后的某刻再次开始时做出响应。

## 示例

以下示例将补间动画应用于影片剪辑 `img1_mc`。通过在 `Tween.onMotionFinished` 事件处理函数中调用 `Tween.start()` 方法，将从开始点起重复地循环播放动画。在调用 `Tween.start()` 方法时，`Tween` 实例会调用 `Tween.onMotionStarted` 处理函数。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone,0, Stage.width, 4, true);

myTween.onMotionFinished = function() {
 myTween.start();
};

myTween.onMotionStarted = function() {
 trace("onMotionStarted");
};
```

# Tween.onMotionStopped

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
tweenInstance.onMotionStopped = function() {
 // ...
};
```

## 说明

事件处理函数；当补间动画在动画的结束点完成时或调用 `Tween.stop()` 方法时调用。处理该事件允许您的代码在补间动画停止点做出响应。

## 示例

以下示例将补间动画应用于影片剪辑 `img1_mc`。当 **Tween** 实例结束其动画时，该 **Tween** 实例会调用 `Tween.onMotionStopped` 事件处理函数。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone,0, Stage.width-img1_mc._width, 3,
 true);

myTween.onMotionStopped = function() {
 trace("onMotionStopped");
};
```

# Tween.position

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance.position*

## 说明

属性（只读）；正在补间的目标对象属性的当前值。用补间动画的每一个绘制好的帧更新此值。

## 示例

以下示例跟踪 **Tween** 对象的当前 `Tween.position` 和补间动画最后一帧结束的补间位置的 `Tween.position` 值。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new mx.transitions.Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone, 0, Stage.width-img1_mc._width, 3,
 true);
myTween.onMotionChanged = function() {
 var myPosition:Number = myTween.position;
 var myFinish:Number = myTween.finish;
 trace(myPosition + " : " + myFinish);
};
```

# Tween.prevFrame()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.prevFrame()

## 返回

无。

## 说明

方法：从已停止动画的当前停止点播放补间动画的前一帧。在使用 `Tween.stop()` 方法停止补间动画后，使用此方法可将补间动画一次向后播放一帧。

### 提醒

该方法只能用于基于帧的补间。通过将 `Tween.start()` `useSeconds` 参数设置为 `false`，可以在创建时将补间设置为基于帧。有关 `useSeconds` 参数的更多信息，请参见第 1227 页的 `Tween.start()`。

## 示例

此示例将补间动画应用于影片剪辑 `img1_mc`。通过在 `onMotionFinished` 事件所触发的处理函数中调用 `Tween.start()` 方法，将从开始点起重复地循环播放动画。单击名为 `forwardByFrame_btn` 的按钮会调用 `Tween.stop()` 方法来停止动画，随后调用 `Tween.prevFrame()` 方法。在补间动画过程中单击该按钮会停止动画，然后只将动画退回一帧。单击 `resumeTween_btn` 按钮会调用 `Tween.resume()` 方法，并继续执行补间动画。在创建 `Tween` 实例时，将 `useSeconds` 参数声明为 `false` 会使补间基于帧。此过程是使用 `Tween.nextFrame()` 方法所必需的。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例、一个名为 `resumeTween_btn` 的影片剪辑实例和一个名为 `reverseByFrame_btn` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone, -img1_mc._width, Stage.width, 50,
 false);

myTween.onMotionFinished = function() {
 myTween.start();
};

reverseByFrame_btn.onRelease = function() {
 myTween.stop();
```

```

 myTween.prevFrame();
 };
 resumeTween_btn.onRelease = function() {
 myTween.resume();
 };

```

## Tween.resume()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*tweenInstance*.resume()

### 返回

无。

### 说明

方法；继续播放已停止的补间动画。通过使用 [Tween.stop\(\)](#) 方法停止补间动画后，使用该方法可以继续执行补间动画。



该方法只能用于基于帧的补间。通过将 `useSeconds` 参数设置为 `false`，可以在创建时将补间设置为基于帧。

### 示例

此示例将补间动画应用于影片剪辑 `img1_mc`。通过在 `onMotionFinished` 事件所触发的处理函数中调用 [Tween.start\(\)](#) 方法，将从开始点起重复地循环播放动画。单击

`stopTween_btn` 按钮会调用 [Tween.stop\(\)](#) 方法，以在补间动画的当前值停止补间动画。

单击 `resumeTween_btn` 按钮会调用 `Tween.resume()` 方法，以从补间动画的停止点继续补间动画。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例、一个名为 `stopTween_btn` 的影片剪辑实例和一个名为 `resumeTween_btn` 的影片剪辑实例：

```

import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone, -img1_mc._width, Stage.width, 3,
 true);

myTween.onMotionFinished = function() {
 myTween.start();
};

```

```
stopTween_btn.onRelease = function() {
 myTween.stop();
};
resumeTween_btn.onRelease = function() {
 myTween.resume();
};
```

## Tween.rewind()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*tweenInstance*.rewind()

### 参数

无。

### 返回

无。

### 说明

方法：使补间动画的播放后退到其开始值。如果在补间动画仍在播放期间调用 Tween.rewind()，则补间动画将后退到其开始值，然后继续播放。如果在补间动画已停止或已结束其动画期间调用 Tween.rewind()，则补间动画将后退到其开始值，并保持停止状态。通过使用 Tween.stop() 方法停止补间动画后，使用该方法可以将补间动画后退到它的开始点，或者在播放补间动画时使它后退。

### 示例

以下示例将补间动画应用于影片剪辑 img1\_mc。通过在 Tween.onMotionFinished 事件所触发的处理函数中调用 Tween.start() 方法，将从开始点起重复地循环播放动画。单击 **rewindTween\_btn** 按钮会调用 Tween.rewind() 方法，以将补间动画后退到其开始点。此示例需要在舞台上有一个名为 img1\_mc 的影片剪辑实例、一个名为 stopTween\_btn 的影片剪辑实例、一个名为 rewindTween\_btn 的影片剪辑实例和一个名为 resumeTween\_btn 的影片剪辑实例：

```
import mx.transitions.Tween;
```

```
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone, img1_mc._width, Stage.width, 8,
 true);

myTween.onMotionFinished = function() {
 myTween.start();
};

stopTween_btn.onRelease = function() {
 myTween.stop();
};

rewindTween_btn.onRelease = function() {
 myTween.rewind();
};

resumeTween_btn.onRelease = function() {
 myTween.resume();
};
```

## Tween.start()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*tweenInstance*.start()

### 返回

无。

### 说明

方法：从开始点开始播放补间动画。此方法用于在补间动画停止或已结束其动画后，从其动画的开始处重新开始补间。

## 示例

此示例创建一个新的 **Tween** 对象，该对象对影片剪辑 `img1_mc` 的 `_x` 属性进行动画处理。在补间动画完成并调用 `Tween.onMotionFinished` 事件处理函数后，将再次通过调用 `Tween.start()` 方法重新播放补间动画。结果是影片剪辑从舞台左侧移至右侧，然后在到达舞台尽头时再次开始播放补间动画。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone,0, Stage.width, 50, false);

myTween.onMotionFinished = function() {
 myTween.start();
};
```

# Tween.stop()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.stop()

## 返回

无。

## 说明

方法：在当前值停止播放补间动画。

## 示例

以下示例将补间动画应用于影片剪辑 `img1_mc` 的 `_x` 属性。`stopTween_btn` 影片剪辑的 `onRelease()` 处理函数调用 `Tween.stop()` 方法以停止补间动画，而 `resumeTween_btn` 影片剪辑的 `onRelease()` 处理函数调用 `Tween.resume()` 方法以从停止位置继续执行补间动画。此示例需要在舞台上具有一个名为 `img1_mc` 的影片剪辑实例、一个名为 `stopTween_btn` 的影片剪辑实例和一个名为 `resumeTween_btn` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone, img1_mc._width, Stage.width, 8,
 true);

myTween.onMotionFinished = function() {
```



```
 myTween.start();
 };

 stopTween_btn.onRelease = function() {
 myTween.stop();
 };

 resumeTween_btn.onRelease = function() {
 myTween.resume();
 };
};
```

## Tween.time

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

`tweenInstance.time`

### 说明

属性（只读）；如果在创建 **Tween** 实例时，将 `useSeconds` 参数设置为 `true`，则表示在补间动画持续过程中所经过的当前秒数。如果补间动画的 `useSeconds` 参数设置为 `false`，则 `Tween.time` 返回在 **Tween** 对象动画过程中所经过的当前帧数。

### 示例

以下示例返回一个 **Tween** 实例的 **time** 值。此示例需要在舞台上有一个名为 `img1_mc` 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new mx.transitions.Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone, 0, Stage.width-img1_mc._width, 10,
 false);
myTween.onMotionChanged = function() {
 var myCurrentTime:Number = myTween.time;
 var myCurrentDuration:Number = myTween.duration;
 trace(myCurrentTime + " of " + myCurrentDuration);
};
```

# Tween.toString()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.toString()

## 返回

返回以下字符串：“[Tween]”。

## 说明

方法；返回类名称 “[Tween]”。

## 示例

在以下示例中，通过调用 Tween.toString() 方法以返回 “[Tween]”，然后标识 Tween 对象的类名来标识该对象。此示例需要在舞台上有一个名为 img1\_mc 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_alpha",
 mx.transitions.easing.None.easeNone,0, 100, 50, false);
var theClassName:String = myTween.toString();
trace(theClassName);
```

# Tween.yoyo()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*tweenInstance*.yoyo()

## 返回

无。

## 说明

方法；指示补间动画以与补间属性最后一次增加的方向相反的方向播放。如果在 **Tween** 对象的动画完成之前调用此方法，则该动画将立即跳至其播放末尾，然后从该点以相反方向播放。通过在 **Tween.onMotionFinished** 事件处理函数中调用 **Tween.yoyo()** 方法，可以获得动画完成其整个播放后反转其整个播放的效果。此过程可确保 **Tween.yoyo** 方法的反转效果直到当前补间动画完成后才会开始。请参见第 1219 页的 **Tween.onMotionFinished**。

## 示例

在以下示例中，**Tween.onMotionFinished** 事件触发一个处理函数，而该处理函数通过调用 **Tween.yoyo()** 方法通知 **Tween** 实例将影片剪辑 **img1\_mc** 反向动画。结果是在动画循环过程中，从舞台左侧移至右侧的影片剪辑反转方向，从右侧移至了左侧。此示例需要在舞台上具有一个名为 **img1\_mc** 的影片剪辑实例：

```
import mx.transitions.Tween;
var myTween:Tween = new Tween(img1_mc, "_x",
 mx.transitions.easing.None.easeNone,0, Stage.width, 4, true);
myTween.onMotionFinished = function() {
 myTween.yoyo();
};
```



# UIComponent 类

**UIComponent** 类不是可视组件；它包含可让 **Macromedia** 组件共享某些常用行为的方法、属性和事件。所有 **Macromedia Component Architecture** 第 2 版组件都扩展 **UIComponent**。**UIComponent** 类允许您执行以下操作：

- 接收焦点和键盘输入
- 启用和禁用组件
- 按布局调整大小

要使用 **UIComponent** 的方法和属性，您可以直接从您正在使用的任意一个组件中调用它们。例如，若要从 **RadioButton** 组件调用 **UIComponent.setFocus()**，需要编写以下代码：

```
myRadioButton.setFocus();
```

如果您要使用 **Macromedia Component Architecture** 第 2 版创建新组件，只需创建 **UIComponent** 的实例即可。即使在这种情况下，其它子类（如 **Button**）通常仍会隐式创建 **UIComponent**。如果您确实需要创建 **UIComponent** 的实例，请使用以下代码：

```
class MyComponent extends mx.core.UIComponent;
```

## UIComponent 类 (API)

继承 **MovieClip** > **UIObject** 类 > **UIComponent**

**ActionScript** 类名称 **mx.core.UIComponent**

使用 **UIComponent** 类的方法、属性和事件可以控制 **Flash** 可视组件的常用行为。

# UIComponent 类的方法摘要

下表列出了 UIComponent 类的方法。

方法	说明
<code>UIComponent.setFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

## 从 UIObject 类继承的方法

下表列出了 UIComponent 类从 UIObject 类继承的方法。从 UIComponent 对象调用这些方法时，请使用 `UIComponentInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

# UIComponent 类的属性摘要

下表列出了 UIComponent 类的属性。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## 从 UIObject 类继承的属性

下表列出了 UIComponent 类从 UIObject 类继承的属性。从 UIComponent 对象访问这些属性时，请使用 `UIComponentInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

# UIComponent 类的事件摘要

下表列出了 UIComponent 类的事件。

事件	说明
<code>UIComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UIComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UIComponent.keyDown</code>	当按下按键时进行广播。
<code>UIComponent.keyUp</code>	当松开按键时进行广播。

## 从 UIObject 类继承的事件

下表列出了 UICOMPONENT 类从 UIObject 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

# UICOMPONENT.enabled

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

`componentInstance.enabled`

### 说明

属性；指示组件是 (true) 否 (false) 可以接受焦点和鼠标输入。默认值为 true。

### 示例

以下示例将 **CheckBox** 组件的 `enabled` 属性设置为 false：

```
checkBoxInstance.enabled = false;
```



# UIComponent.focusIn

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.focusIn = function(eventObj:Object) {
 //...
};
componentInstance.addEventListener("focusIn", listenerObject);
```

用法 2:

```
on (focusIn) {
 // ...
}
```

## 说明

事件：通知侦听器对象已接收到键盘焦点。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 *focusIn*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法：当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 [EventDispatcher.addEventListener\(\)](#) 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 *on()* 处理函数，并且必须直接附加到一个组件实例。

## 示例

以下代码在用户在 **TextInput** 组件 `txt` 中键入内容时禁用 **Button** 组件 `btn`，而在用户单击该按钮时启用它：

```
var txt:mx.controls.TextInput;
var btn:mx.controls.Button;

var txtListener:Object = new Object();
txtListener.focusOut = function() {
 _root.btn.enabled = true;
}
txt.addEventListener("focusOut", txtListener);

var txtListener2:Object = new Object();
txtListener2.focusIn = function() {
 _root.btn.enabled = false;
}
txt.addEventListener("focusIn", txtListener2);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)、[UIComponent.focusOut](#)

# UIComponent.focusOut

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
on(focusOut){
 ...
}
listenerObject = new Object();
listenerObject.focusOut = function(eventObject){
 ...
}
componentInstance.addEventListener("focusOut", listenerObject)
```

## 说明

事件：通知侦听器对象已丢失键盘焦点。

第一个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 `focusOut`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下代码在用户在 `TextInput` 组件 `txt` 中键入内容时禁用 `Button` 组件 `btn`，而在用户单击该按钮时启用它：

```
var txt:mx.controls.TextInput;
var btn:mx.controls.Button;

var txtListener:Object = new Object();
txtListener.focusOut = function() {
 _root.btn.enabled = true;
}
txt.addEventListener("focusOut", txtListener);

var txtListener2:Object = new Object();
txtListener2.focusIn = function() {
 _root.btn.enabled = false;
}
txt.addEventListener("focusIn", txtListener2);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#), [UIComponent.focusIn](#)

# UIComponent.getFocus()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
componentInstance.getFocus();
```

## 参数

无。

## 返回

对当前具有焦点的对象的引用。

## 说明

方法；返回对具有键盘焦点的对象的引用。

## 示例

以下代码返回对具有焦点的对象的引用，并将它分配给 tmp 变量：

```
var tmp = checkbox.getFocus();
```

# UIComponent.keyDown

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
on(keyDown){
 ...
}
listenerObject = new Object();
listenerObject.keyDown = function(eventObject){
 ...
}
componentInstance.addEventListener("keyDown", listenerObject)
```

## 说明

事件：当某个按键被按下时通知侦听器。这是一个级别很低的事件，除非必要，否则不应使用，因为它可能会影响系统性能。

第一个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 `keyDown`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下代码会在某个按键被按下时使图标闪烁：

```
formListener.handleEvent = function(eventObj)
{
 form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyDown", formListener);
```

# UIComponent.keyUp

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
on(keyUp){
 ...
}
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
 ...
}
componentInstance.addEventListener("keyUp", listenerObject)
```

## 说明

事件：当某个按键被松开时通知侦听器。这是一个级别很低的事件，除非必要，否则不应使用，因为它可能会影响系统性能。

第一个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 `keyUp`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下代码在某个按键被松开时使图标闪烁：

```
formListener.handleEvent = function(eventObj)
{
 form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyUp", formListener);
```

# UIComponent.setFocus()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
componentInstance.setFocus();
```

## 参数

无。

## 返回

无。

### 说明

方法：设置此组件实例的焦点。具有焦点的实例接收所有的键盘输入。

### 示例

以下代码将焦点设置给 `checkbox` 实例：

```
checkbox.setFocus();
```

## UIComponent.tabIndex

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
instance.tabIndex
```

### 说明

属性：指明文档中组件的 **Tab** 键顺序的值。

### 示例

以下代码将 `tmp` 的值设置为 `checkbox` 实例的 `tabIndex` 属性：

```
var tmp = checkbox.tabIndex;
```





# UIEventDispatcher 类

ActionScript 类名称 `mx.events.UIEventDispatcher`

继承 [EventDispatcher](#) 类 > `UIEventDispatcher`

`UIEventDispatcher` 类混合在 `UIComponent` 类中，允许组件发出某些事件。

要使不是继承自 `UIComponent` 的对象发送某些事件，可以使用 `UIEventDispatcher`。

## UIEventDispatcher 类的方法摘要

下表列出了 `UIEventDispatcher` 类的方法。

方法	说明
<code>UIEventDispatcher.removeEventListener()</code>	从组件实例删除已注册侦听器。此方法覆盖 <code>eventDispatcher.removeEventListener()</code> 方法。

## 从 EventDispatcher 类继承的方法

下表列出了 `UIEventDispatcher` 类从 `EventDispatcher` 类继承的方法。从

`UIEventDispatcher` 对象调用这些方法时，请使用

`UIEventDispatcherInstance.methodName` 的形式。

方法	说明
<code>EventDispatcher.addEventListener()</code>	向组件实例注册侦听器。
<code>EventDispatcher.dispatchEvent()</code>	向所有注册的侦听器发送事件。

# UIEventDispatcher 类的事件摘要

下表列出了 UIEventDispatcher 类的事件。

方法	说明
<code>UIEventDispatcher.keyDown</code>	当按下按键时进行广播。
<code>UIEventDispatcher.keyUp</code>	当松开按键时进行广播。
<code>UIEventDispatcher.load</code>	当组件加载到 Flash Player 时进行广播。
<code>UIEventDispatcher.mouseDown</code>	当按下鼠标时进行广播。
<code>UIEventDispatcher.mouseOut</code>	当鼠标移离组件实例时进行广播。
<code>UIEventDispatcher.mouseOver</code>	当鼠标在移到组件实例上进行广播。
<code>UIEventDispatcher.mouseUp</code>	当按下鼠标然后松开时进行广播。
<code>UIEventDispatcher.unload</code>	从 Flash Player 中卸载组件时进行广播。

## UIEventDispatcher.keyDown

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
listenerObject = new Object();
listenerObject.keyDown = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("keyDown", listenerObject)
```

### 说明

事件：当按下按键且 **Flash** 应用程序具有焦点时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIEventDispatcher.keyUp

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("keyUp", listenerObject)
```

## 说明

事件：当松开已按下的按键且 **Flash** 应用程序具有焦点时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIEventDispatcher.load

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.load = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("load", listenerObject)
```

## 说明

事件：当组件加载到 **Flash Player** 时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIEventDispatcher.mouseDown

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.mouseDown = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("mouseDown", listenerObject)
```

## 说明

事件：当 **Flash** 应用程序具有焦点且按下鼠标时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIEventDispatcher.mouseOut

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.mouseOut = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("mouseOut", listenerObject)
```

## 说明

事件：当 **Flash** 应用程序具有焦点且鼠标移离组件实例时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIEventDispatcher.mouseOver

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.mouseOver = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("mouseover", listenerObject)
```

## 说明

事件：当 **Flash** 应用程序具有焦点且鼠标移到组件实例上对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIEventDispatcher.mouseUp

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.mouseUp = function(eventObject){
 // 在此处插入您的代码。
}
componentInstance.addEventListener("mouseup", listenerObject)
```

## 说明

事件：当 **Flash** 应用程序具有焦点且按下并松开鼠标时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

# UIEventDispatcher.removeEventListener()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004 和 Flash MX Professional 2004。

## 用法

```
componentInstance.removeEventListener(event, listener)
```

## 参数

*event* 表示事件名称的字符串。

*listener* 对侦听器对象或函数的引用。

## 返回

无。

## 说明

方法：从播放事件的组件实例注销侦听器对象。此方法覆盖在 `EventDispatcher` 基类中找到的 `EventDispatcher.removeEventListener()` 事件。

# UIEventDispatcher.unload

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
listenerObject = new Object();
listenerObject.unload = function(eventObject) {
 // 在此处插入您的代码。
}
componentInstance.addEventListener("unload", listenerObject)
```

## 说明

事件：从 **Flash Player** 中卸载组件时对所有注册的侦听器广播。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。

# UIObject 类

继承 MovieClip > UIObject

ActionScript 类名称 mx.core.UIObject

UIObject 是所有 Macromedia Component Architecture 第 2 版组件的基类；它不是可视组件。UIObject 类包装了 ActionScript MovieClip 对象，并包含允许第 2 版组件共享某些常用行为的函数和属性。环绕 MovieClip 类允许 Macromedia 在将来无需中断内容即可添加新事件和扩展功能。同时，对于不熟悉传统的 Flash “影片”和“帧”概念的用户，环绕 MovieClip 类还允许他们无需了解这些概念即可使用属性、方法和事件来创建基于组件的应用程序。

UIObject 类实现了以下内容：

- 样式
- 事件
- 按缩放比例调整大小

若要使用 UIObject 类的方法和属性，可直接从所用任何组件中调用它们。例如，若要从 RadioButton 组件调用 `UIObject.setSize()` 方法，需要编写以下代码：

```
myRadioButton.setSize(30, 30);
```

如果您在用 Macromedia Component Architecture 第 2 版创建新组件，则只需创建一个 UIObject 实例即可。即使在这种情况下，UIObject 也经常由其它子类（例如 Button）隐式创建。如果您确实需要创建一个 UIObject 实例，请使用下列代码：

```
class MyComponent extends UIObject;
```

# UIObject 类的方法摘要

下表列出了 UIObject 类的方法。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createLabel()</code>	创建一个 TextField 子对象，供创建组件时使用。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

# UIObject 类的属性摘要

下表列出了 UIObject 类的属性。

属性	说明
<code>UIObject.bottom</code>	对象的底边缘位置（相对于其父对象的底边缘）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右边缘位置（相对于其父对象的右边缘）。只读。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。



属性	说明
<code>UIObject.x</code>	对象的左边缘（以像素为单位）。只读。
<code>UIObject.y</code>	对象的上边缘（以像素为单位）。只读。

## UIObject 类的事件摘要

下表列出了 UIObject 类的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## UIObject.bottom

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
componentInstance.bottom
```

### 说明

属性（只读）；一个数字，指明对象底边相对于其父对象底边的位置（以像素为单位）。若要设置此属性，请调用 `UIObject.move()`。

### 示例

此示例移动复选框，使其对齐到列表框底边的下方：

```
myCheckbox.move(myCheckbox.x, form.height - listbox.bottom);
```

# UIObject.createClassObject()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
componentInstance.createClassObject(className, instanceName, depth, initObject)
```

## 参数

*className* 一个对象，指示新实例的类。

*instanceName* 一个字符串，指示新实例的实例名称。

*depth* 一个数字，指示新实例的深度。

*initObject* 一个对象，它包含新实例的初始化属性。

## 返回

一个 UIObject 对象，它是所指定类的一个实例。

## 说明

方法：在运行时创建组件的一个实例。在调用此方法之前，使用 `import` 语句并指定类包名称。此外，该组件必须在 **FLA** 文件的库中。

## 示例

以下代码导入 **Button** 组件的资源，然后创建 **Button** 组件的一个子对象。

```
import mx.controls.Button;
createClassObject(Button,"button2",5,{label:"Test Button"});
```

以下示例创建了一个 **CheckBox** 对象：

```
import mx.controls.CheckBox;
form.createClassObject(CheckBox, "cb", 0, {label:"Check this"});
```

也可以使用以下语法指定类包名称：

```
createClassObject(mx.controls.Button, "button2", 5, {label:"Test Button"});
```

# UIObject.createLabel()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
createLabel(name, depth, text)
```

## 参数

*name* 一个字符串，表示实例名称。

*depth* 一个数字，指示新实例的深度。

*text* 标签文本。

## 返回

一个 TextField 对象。

## 说明

方法；创建一个 TextField 子对象。大多数组件使用该方法获取轻型文本对象，以在继承组件的调整大小和样式方法及属性时显示组件中的文本。创建新组件时使用该方法。该方法所创建的 TextField 与使用 MovieClip.createTextField() 创建的 TextField 对象相同，但增加了从父 UIObject 继承一些有用的属性和方法的优点。

使用 UIObject.createLabel() 在组件中创建的 TextField 可以利用以下继承的 UIObject 方法在父 UIObject 的上下文中设置调整大小和样式：

- TextField.getPreferredHeight() :Number
- TextField.getPreferredWidth() :Number
- TextField.setStyle( styleName :String, value )
- TextField.setSize( width :Number, height :Number)
- TextField.setValue( text :String)

### 提醒

用 UIObject.createLabel() 创建的 TextField 最初具有一个值为 false 的 TextField.\_visible 属性。此属性用于避免在父组件调用 UIObject.setSize() 时可能出现的闪烁现象。在调整父组件的子对象大小后调用 UIObject.draw() 时，会将 TextField.\_visible 属性设置为 true。

有关更多信息，请参见第 103 页的“单元格渲染器简单示例”中的 MultilineCell.as 文件示例。

## 示例

以下示例在组件的 `UIComponent.createChildren()` 方法中创建一个名为 `multiLineLabel` 的 `TextField` 实例：

```
public function createChildren():Void {
 var myTextField_txt:TextField = this.createLabel("multiLineLabel", 900,
 "Hello World");
 // 设置 TextField 的 fontSize 样式属性。
 myTextField_txt.setStyle("fontSize", 18);
 // 设置 TextField 的初始大小。
 myTextField_txt.setSize(myTextField_txt.getPreferredWidth(),
 myTextField_txt.getPreferredHeight());
 // 设置 TextField 在舞台中央的初始位置。
 myTextField_txt._x = (Stage.width/2) - (myTextField_txt._width/2);
 myTextField_txt._y = (Stage.height/2) - (myTextField_txt._height/2);
}
```

# UIObject.createObject()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
componentInstance.createObject(linkageName, instanceName, depth, initObject)
```

## 参数

*linkageName* 一个字符串，指示库中某元件的链接标识符。

*instanceName* 一个字符串，指示新实例的实例名称。

*depth* 一个数字，指示新实例的深度。

*initObject* 一个对象，它包含新实例的初始化属性。

## 返回

一个 `UIObject` 对象，它是元件的一个实例。

## 说明

方法：创建对象的子对象。通常只有组件开发人员或高级开发人员使用此方法。

## 示例

以下示例在 `form` 对象上创建了一个 `CheckBox` 实例：

```
form.createObject("CheckBox", "sym1", 0);
```

# UIObject.destroyObject()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`destroyObject(instanceName)`

## 参数

*instanceName* 一个字符串，指示要破坏的对象的实例名称。

## 返回

无。

## 说明

方法；破坏组件实例。

## 示例

以下示例在单击按钮时删除 **TextInput** 实例 `my_ti`。在当前文档库中具有一个 **Button** 组件和一个 **TextInput** 组件的情况下，将以下代码添加到主时间轴的第一帧中：

```
// 创建 TextInput 和 Button 实例
this.createClassObject(mx.controls.TextInput, "my_ti", 1, {text:"Hello
 World"});
this.createClassObject(mx.controls.Button, "my_button", 2, {label:"My
 Button"});
// 将 Button 移至 TextInput 的下方
my_button.move(my_ti.left, Stage.height - my_ti.bottom);

// 为按钮单击事件创建侦听器对象
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object){
 destroyObject("my_ti");
}
// 添加侦听器
my_button.addEventListener("click", buttonListener);
```

# UIObject.doLater()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.doLater(target, "function")*

## 参数

*target* 对包含指定的函数的时间轴的引用。

*function* 一个字符串，指示经过组件影片剪辑中的某帧后（以便“属性”或“组件”检查器中设置的组件属性均可用）要调用的函数的名称。

## 返回

无。

## 说明

方法：调用一个用户定义函数，但仅在“属性”检查器或“组件”检查器中的所有组件属性设置均已完成后才调用。所有继承自 **UIObject** 的第 2 版组件都有 `doLater()` 方法。

时间轴中的 **ActionScript** 可能无法立即使用“属性”检查器或“组件”检查器中设置的组件属性。例如，尝试使用 SWF 文件的第一帧上的 **ActionScript** 来跟踪 **CheckBox** 组件的 `label` 属性时，会在无提示的情况下失败，即使组件仍按预期的那样出现在舞台上时也是如此。

尽管在类或帧脚本中设置的属性会立即可用，但在“属性”检查器或“组件”检查器中指定的大部分属性都要等到到达组件内的下一帧时才予设置。

虽然所有延迟访问属性的途径都能解决这个问题，但最简单直接的解决方式是使用 `doLater()` 方法。

## 示例

以下示例显示如何使用 `doLater()` 方法：

```
// 从组件实例调用 doLater()
myCheckBox.doLater(this, "delay");

// 从 doLater() 中调用的函数或方法。
function delay() {
 trace(myCheckBox.label); // 现在可以跟踪属性
 // 在此处执行其它任何语句
}
```

# UIObject.draw

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.draw = function(eventObject:Object) {
 // ...
};
componentInstance.addEventListener("draw", listenerObject);
```

用法 2:

```
on (draw) {
 // ...
}
```

## 说明

事件；通知侦听器对象将要绘制它的图形。这是一个级别很低的事件，除非必要，否则不应使用，因为它可能会影响系统性能。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 *draw*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

## 示例

以下代码在 form 对象被绘制时，重绘对象 form2：

```
formListener.draw = function(eventObj:Object) {
 form2.redraw(true);
}
form.addEventListener("draw", formListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# UIObject.getStyle()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.getStyle(propertyName)*

## 参数

*propertyName* 一个字符串，指示样式属性的名称（如 "fontWeight"、"borderStyle" 等）。

## 返回

样式属性的值。这些值可以是任何数据类型：

## 说明

方法：从样式声明或对象中获取样式属性。如果样式属性是继承性样式，那么该对象的始祖就可能是样式值的源。

有关各组件所支持的样式的列表，请参见各组件条目。另请参见《使用组件》中的“在同一个文档中使用全局、自定义和类样式”。

## 示例

如果 cb 实例的 fontWeight 样式属性是粗体，以下代码会将 ib 实例的 fontWeight 样式属性设置为粗体：

```
if (cb.getStyle("fontWeight") == "bold") {
 ib.setStyle("fontWeight", "bold");
};
```



# UIObject.height

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.height*

## 说明

属性（只读）：指明对象高度的数字（以像素为单位）。若要更改 height 属性，请调用 [UIObject.setSize\(\)](#)。

## 示例

以下示例增加了复选框的高度：

```
myCheckbox.setSize(myCheckbox.width, myCheckbox.height + 10);
```

# UIObject.hide

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.hide = function(eventObject:Object) {
 // ...
};
componentInstance.addEventListener("hide", listenerObject);
```

用法 2：

```
on (hide) {
 // ...
}
```

## 说明

事件：在将对象的 `visible` 属性从 `true` 改为 `false` 时广播。

## 示例

以下处理函数在它所附加到的对象变为不可见时，在“输出”面板中显示一条消息。

```
on (hide) {
 trace("I e become invisible.");
}
```

## 另请参见

[UIObject.reveal](#)

# UIObject.invalidate()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.invalidate()*

## 返回

无。

## 说明

方法：标记对象以便在下一帧时间间隔时进行重绘。

此方法对于开发新的自定义组件的人员尤为有用。自定义组件很可能会支持许多更改组件外观的操作。

通常，构建组件的最佳方式是将用于更新组件外观的逻辑集中在 `draw()` 方法内。如果组件有 `draw()` 方法，您便可以对该组件调用 `invalidate()` 来重绘组件。（有关定义 `draw()` 方法的信息，请参见《使用组件》中的“定义 **draw()** 方法”。）

所有更改组件外观的操作都能调用 `invalidate()`，而不必自己重绘组件本身。这样做的好处是：代码不会无谓地重复，可以在一次重绘中成批处理多项更改操作，而不会导致多次重复的重绘。

## 示例

以下示例将标记 `ProgressBar` 实例 `pBar` 以便进行重绘：

```
pBar.invalidate();
```

# UIObject.left

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance*.left

## 说明

属性（只读）：一个数字，指明对象的左边相对于其父对象的位置（以像素为单位）。若要设置此属性，请调用 [UIObject.move\(\)](#)。

# UIObject.load

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.load = function(eventObject:Object) {
 // ...
};
componentInstance.addEventListener("load", listenerObject);
```

用法 2:

```
on (load) {
 //...
}
```

## 说明

事件：通知侦听器正在创建该对象的子对象。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 `load`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

## 示例

以下示例将在加载 `form` 实例后创建一个 `MySymbol` 实例：

```
var formListener:Object = new Object();
formListener.load = function(eventObj:Object) {
 form.createObject("MySymbol", "sym1", 0);
};
form.addEventListener("load", formListener);
```

# UIObject.move

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1：

```
var listenerObject:Object = new Object();
listenerObject.move = function(eventObject:Object):Void {
 // ...
};
componentInstance.addEventListener("move", listenerObject);
```

用法 2:

```
on (move) {
 // ...
}
```

### 说明

事件：通知侦听器对象已移动。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 *move*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

### 示例

以下示例调用 `move()` 方法以重新定位 **Button** 组件 `my_button`，从当前位置重新定位到舞台的左上角 (10,10)：

```
var my_button:mx.controls.Button;
my_button.addEventListener("move", doMove);
function doMove(evt_obj:Object):Void {
 trace(evt_obj.target + " moved from {oldX:" + evt_obj.oldX + ", oldY:" +
 evt_obj.oldY + "} to {x:" + evt_obj.target.x + ", y:" + evt_obj.target.y +
 "}");
}
my_button.move(10, 10);
```

## UIObject.move()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
componentInstance.move(x, y, noEvent)
```

### 参数

*x* 一个数字，指示对象的左上角相对于其父对象的位置。

*y* 一个数字，指示对象的左上角相对于其父对象的位置。

*noEvent* 一个布尔值，指示是否应该调度移动事件。

### 返回

无。

### 说明

方法：将对象移动到要求的位置。您应该只将整数值传递给 `UIObject.move()`，否则，组件可能会模糊不清。

### 示例

以下示例将复选框向右移 10 个像素：

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

以下示例调用 `move()` 方法以重新定位 **Button** 组件 `my_button`，从当前位置重新定位到舞台的左上角 (10,10)：

```
var my_button:mx.controls.Button;
my_button.addEventListener("move", doMove);
function doMove(evt_obj:Object):Void {
 trace(evt_obj.target + " moved from {oldX:" + evt_obj.oldX + ", oldY:" +
 evt_obj.oldY + "} to {x:" + evt_obj.target.x + ", y:" + evt_obj.target.y +
 "}");
}
my_button.move(10, 10);
```

## UIObject.redraw()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
componentInstance.redraw(always)
```

### 参数

*always* 一个布尔值。如果为 `true`，则即使未调用 `invalidate()`，该方法也会绘制对象。如果为 `false`，则只有在调用了 `invalidate()` 的情况下，该方法才绘制对象。

## 返回

无。

## 说明

方法：迫使对象有效以便在当前帧中绘制它。

## 示例

以下示例创建一个复选框和一个按钮并绘制它们，原因是未期望其它脚本修改表单：

```
form.createClassObject(mx.controls.CheckBox, "cb", 0);
form.createClassObject(mx.controls.Button, "b", 1);
form.redraw(true)
```

# UIObject.resize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.resize = function(eventObject:Object) {
 // ...
};
componentInstance.addEventListener("resize", listenerObject);
```

用法 2:

```
on (resize) {
 // ...
}
```

## 说明

事件：通知侦听器对象的大小已被调整。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 `resize`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

## 示例

在以下示例中，当移动 `form` 时，将调用 `setSize()` 方法，以使 `sym1` 的宽度变为原来的二分之一，高度变为原来的四分之一：

```
var formListener:Object = new Object();
formListener.resize = function(eventObj:Object):Void {
 form.sym1.setSize(sym1.width / 2, sym1.height / 4);
};
form.addEventListener("resize", formListener);
```

以下示例调用 `setSize()` 方法以将 **Button** 组件 `my_button` 的大小调整为 200 像素宽、100 像素高：

```
var my_button:mx.controls.Button;

my_button.addEventListener("resize", doSize);
function doSize(evt_obj:Object):Void {
 trace(evt_obj.target + " resized from {oldWidth:" + evt_obj.oldWidth + ",
 oldHeight:" + evt_obj.oldHeight + "} to {width:" + evt_obj.target.width +
 ", height:" + evt_obj.target.height + "}");
}
my_button.setSize(200, 100);
```



# UIObject.reveal

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

### 用法 1:

```
var listenerObject:Object = new Object();
listenerObject.reveal = function(eventObject:Object) {
 // ...
};
componentInstance.addEventListener("reveal", listenerObject);
```

### 用法 2:

```
on (reveal) {
 // ...
}
```

## 说明

事件：在将对象的 `visible` 属性从 `false` 更改为 `true` 时广播。

## 示例

以下处理函数在它所附加到的对象变为可见时，在“输出”面板中显示一条消息。

```
on (reveal) {
 trace("I e become visible.");
}
```

## 另请参见

[UIObject.hide](#)

# UIObject.right

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance*.right

## 说明

属性（只读）：一个数字，指明对象的右边相对于其父对象右边的位置（以像素为单位）。若要设置此属性，请调用 `UIObject.move()`。

## 示例

以下示例移动复选框，使其对齐到列表框右边的下方：

```
myCheckbox.move(form.width - listBox.right, myCheckbox.y);
```

# UIObject.scaleX

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance*.scaleX

## 说明

属性：一个数字，指示对象相对于其父对象在 **x** 方向上的缩放因子。

## 示例

以下示例将复选框的宽度放大两倍，并将 tmp 变量设置为水平缩放因子：

```
checkboxbox.scaleX = 200;
var tmp:Number = checkbox.scaleX;
```

# UIObject.scaleY

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.scaleY*

## 说明

属性；一个数字，指示对象相对于其父对象在 y 方向上的缩放因子。

## 示例

以下示例将复选框的高度放大两倍，并将 tmp 变量设置为垂直缩放因子：

```
checkbox.scaleY = 200;
var tmp:Number = checkbox.scaleY;
```

# UIObject.setSize()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.setSize(width, height, noEvent)*

## 参数

*width* 一个数字，指示对象的宽度（以像素为单位）。

*height* 一个数字，指示对象的高度（以像素为单位）。

*noEvent* 一个布尔值，指示是否应该调度调整大小事件。

## 返回

无。

## 说明

方法：将对象调整到要求的大小。您应该只将整数值传递给 `UIObject.setSize()`，否则，组件可能会模糊不清。该方法（与 `UIObject` 的所有方法和属性相似）可以从任何组件实例中使用。

对 **ComboBox** 实例调用此方法时，将会调整组合框的大小，而且所包含列表的 `rowHeight` 属性也会更改。



一些组件只允许您修改高度或宽度尺寸。例如，`CheckBox` 和 `RadioButton` 组件不允许您修改高度。

## 示例

此示例将 `pBar` 组件实例的大小调整为 100 像素宽、100 像素高：

```
pBar.setSize(100, 100);
```

以下示例调用 `setSize()` 方法以将 **Button** 组件 `my_button` 的大小调整为 200 像素宽、100 像素高：

```
var my_button:mx.controls.Button;

my_button.addEventListener("resize", doSize);
function doSize(evt_obj:Object):Void {
 trace(evt_obj.target + " resized from {oldWidth:" + evt_obj.oldWidth + ",
 oldHeight:" + evt_obj.oldHeight + "} to {width:" + evt_obj.target.width +
 ", height:" + evt_obj.target.height + "}");
}
my_button.setSize(200, 100);
```

# UIObject.setSkin()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
componentInstance.setSkin(id, linkageName)
```

## 参数

*id* 一个数字，指示组件中外观的深度。

*linkageName* 一个字符串，指示库中的一个资源。

## 返回

对附加的影片剪辑（外观）的引用。

## 说明

方法：设置组件实例中的外观。可在创建组件时在组件的类文件中使用此方法。有关更多信息，请参见《使用组件》中的“关于指定外观”。

不能用此方法设置组件外观（例如，像在运行时设置组件样式那样）。

## 示例

此示例是从一个名为 **Shape** 的新组件的类文件中摘取的代码片断。它创建一个 `themeShape` 变量，并将其设置为外观的链接标识符。`createChildren()` 方法调用了 `setSkin()` 方法，并为其传递 `id 1` 和存有外观链接标识符的变量：

```
class Shape extends UIComponent {

 static var symbolName:String = "Shape";
 static var symbolOwner:Object = Shape;
 var className:String = "Shape";

 var themeShape:String = "circle_skin"

 function Shape() {
 }

 function init(Void):Void {
 super.init();
 }

 function createChildren():Void {
 setSkin(1, themeShape);
 super.createChildren();
 }
}
```

# UIObject.setStyle()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.setStyle(propertyName, value)*

## 参数

*propertyName* 一个字符串，指示样式属性的名称。不同的组件支持不同的样式。每个组件都有一个您可以设置的不同的样式集。例如，第 1088 页的“自定义 **TextArea** 组件”显示了一个包括 `fontWeight` 的样式表。这样，对于 **TextArea** 组件，您就可以将 `fontWeight` 用作 *propertyName* 参数。

*value* 属性的值。如果该值是字符串，它必须括在引号中。

## 返回

无。

## 说明

方法：设置样式声明或对象的样式属性。如果样式属性是继承的样式，则会将新值通知给该对象的子对象。

若要查看各组件所支持的样式的列表，请参见各组件条目。例如，在第 87 页的“对 **Button** 组件使用样式”中列出了 **Button** 组件样式。

若要增强性能，在 **SWF** 文件中对样式进行加载、计算并应用于对象之前，可以更改这些样式。如果可以在加载和计算样式之前更改这些样式，则不必调用 `setStyle`。

**Macromedia** 建议您对每个对象均设置属性，因为在使用样式时，实例化对象可提高性能。在将实例动态附加到舞台时，在调用 `UIObject.createClassObject()` 的情况下设置 `initObj` 参数中的属性，如以下 **ActionScript** 所示：

```
createClassObject(ComponentClass, "myInstance", 0, {styleName:"myStyle",
 color:0x99CCFF});
```

### 提醒

此示例使用 `myStyle` 自定义样式声明。若要更改多个属性，或者更改多个组件实例的属性，请创建自定义样式声明。对于使用自定义样式声明的组件，Flash 的呈现速度要比对为多个属性使用 `UIObject.setStyle()` 的组件的呈现速度快。有关更多信息，请参见《使用组件》中的“为成组的组件设置自定义样式”。

## 示例

以下代码将复选框实例 `cb` 的样式属性 `fontWeight` 设置为粗体：

```
cb.setStyle("fontWeight", "bold");
```

# UIObject.top

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.top*

## 说明

属性（只读）：一个数字，指明对象的上边缘相对于其父对象的位置（以像素为单位）。若要设置此属性，请调用 [UIObject.move\(\)](#)。

# UIObject.unload

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.unload = function(eventObject:Object):Void {
 // ...
};
componentInstance.addEventListener("unload", listenerObject);
```

用法 2:

```
on (unload) {
 // ...
}
```

## 说明

事件：通知侦听器正在卸载该对象的子对象。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 `unload`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法：当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象的属性都包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

## 示例

以下示例会在 `unload` 事件被触发时删除 `sym1`：

```
function doUnload():Void {
 form.destroyObject(sym1);
}
form.addEventListener("unload", doUnload);
```

# UIObject.visible

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance*.visible

## 说明

属性：一个布尔值，指示对象是可见的 (`true`) 还是不可见的 (`false`)。

## 示例

以下示例使 `myLoader` 加载器实例可见：

```
myLoader.visible = true;
```



# UIObject.width

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.width*

## 说明

属性（只读）；一个数字，指明对象的宽度（以像素为单位）。若要更改宽度，请调用 [UIObject.setSize\(\)](#)。

## 示例

以下示例增加了复选框的宽度：

```
myCheckbox.setSize(myCheckbox.width + 10, myCheckbox.height);
```

# UIObject.x

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.x*

## 说明

属性（只读）；一个数字，指明对象的左边缘（以像素为单位）。若要设置此属性，请调用 [UIObject.move\(\)](#)。

## 示例

以下示例将复选框向右移 10 个像素：

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

# UIObject.y

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*componentInstance.y*

## 说明

属性（只读）：一个数字，指明对象的上边缘（以像素为单位）。若要设置此属性，请调用 [UIObject.move\(\)](#)。

## 示例

以下示例将复选框向下移动 10 个像素：

```
myCheckbox.move(myCheckbox.x, myCheckbox.y + 10);
```

# UIScrollBar 组件

UIScrollBar 组件允许将滚动条添加至文本字段。您可以在创作时将滚动条添加至文本字段，或使用 **ActionScript** 在运行时添加。

UIScrollBar 组件的功能与其它所有滚动条类似。它两端各有一个箭头按钮，按钮之间有一个滚动轨道和滚动框（滑块）。它可以附加至文本字段的任何一边，既可以垂直使用也可以水平使用。

## 使用 UIScrollBar 组件

要使用 UIScrollBar 组件，请验证对象贴紧功能是否已打开（“视图” > “贴紧” > “贴紧至对象”）。然后在舞台上创建一个文本输入字段，并从“组件”面板中将 UIScrollBar 组件拖动到文本字段的边框的任何一个象限。

如果滚动条的长度小于其滚动箭头相加的尺寸，则滚动条将无法正确显示。一个箭头按钮将隐藏在另一个的后面。**Flash** 对此不提供错误检查。在此情况下最好使用 **ActionScript** 将滚动条隐藏。如果调整滚动条的尺寸以至没有足够的空间留给滚动框（滑块），则 **Flash** 会将滚动框变为不可见。

与许多其它组件不同，UIScrollBar 组件能够接收连续的鼠标输入（例如当用户按住鼠标按钮时的输入），而不需要重复的单击。

UIScrollBar 组件和键盘不存在交互。

## UIScrollBar 参数

您可以在“属性”检查器或“组件”检查器（“窗口” > “组件检查器”菜单选项）中为每个 UIScrollBar 实例设置以下创作参数：

**\_targetInstanceName** 指示 UIScrollBar 组件所附加到的文本字段实例的名称。

**horizontal** 指示滚动条是水平方向 (true) 还是垂直方向 (false)。默认值为 false。

您可以在“组件”检查器（“窗口” > “组件检查器”）中为每个 UIScrollBar 组件实例设置以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 true。

**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。

提醒

minHeight 和 minWidth 属性由内部的大小调整例程使用。这两个属性在 UIObject 中定义，并可以根据需要被不同的组件覆盖。在为应用程序创建自定义布局管理器时，可以使用这两个属性。否则，在“组件”检查器中设置这两个属性会不具有可视效果。

您可以编写 **ActionScript**，以便使用 UIScrollBar 组件的属性、方法和事件来控制该组件的这些和其它选项。有关更多信息，请参见第 1284 页的“UIScrollBar 类”。

## 创建具有 UIScrollBar 组件的应用程序

以下过程解释了如何在创作时将 UIScrollBar 组件添加到应用程序。

### 创建具有 UIScrollBar 组件的应用程序：

1. 创建一个动态文本字段，并在“属性”检查器中为其指定实例名称 **myText**。
2. 在“属性”检查器中，将该文本输入字段的“线条类型”设置为“多行”或“多行不换行”（如果您计划在水平方向使用滚动条）。
3. 在第一帧中，使用 **ActionScript** 将足够多的文本添加到字段，以使用户必须滚动字段才能查看所有文本。您可以编写以下代码：

```
myText.text="When the moon is in the seventh house and Jupiter aligns with Mars, then peace will guide the planet and love will rule the stars."
```

提醒

确保舞台上的文本字段足够小，需要滚动字段才能查看所有文本。否则，滚动条将不会显示，或者可能只显示为不带滑块抓手（您拖动以滚动内容的部分）的两行。

4. 验证对象对齐功能是否已打开（“视图” > “对齐” > “对齐对象”）。
5. 从“组件”面板中将一个 UIScrollBar 实例拖到文本输入字段上靠近要附加该实例的一边。在松开鼠标时，该组件必须与文本字段重叠，以便能正确地绑定到该字段。  
该组件的 \_targetInstanceName 属性自动填充为“属性”检查器和“组件”检查器中该文本字段的实例名称。如果它未在“参数”选项卡中显示，则可能是重叠 UIScrollBar 实例的程度还不够。

## 6. 选择 “控制” > “测试影片”。

应用程序运行，滚动条滚动文本字段的内容。

您也可以使用 **ActionScript** 在运行时创建 **UIScrollBar** 组件实例，并将它与文本字段相关联。

以下代码创建垂直方向的 **UIScrollBar** 实例，将它附加到名为 **my\_txt** 的文本字段实例的右侧，并设置滚动条大小以匹配文本字段的大小：

```
/**
 * 要求：
 * - 库中有 UIScrollBar 组件
 */
// 创建文本字段。
this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// 为滚动条设置目标文本字段。
my_sb.setScrollTarget(my_txt);


// 调整其尺寸以匹配文本字段。
my_sb.setSize(16, my_txt._height);

// 将其移动到文本字段的旁边。
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 } else {
 my_txt.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

# 自定义 UIScrollView 组件

在创作过程中和在运行时，您都可以在水平和垂直方向上改变 `UIScrollView` 组件的形状。但垂直 `UIScrollView` 不允许您修改宽度，而水平 `UIScrollView` 不允许您修改高度。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参见 `UIObject.setSize()`）或者 `UIScrollView` 类的任何适用的属性和方法。



如果使用 `UIObject.setSize()` 方法，您只能更改实例的高度或宽度，具体取决于实例是水平滚动条还是垂直滚动条。因此，`setSize()` 方法忽略 `height` 或 `width` 参数。

但是请注意，在使用“光晕”主题时，垂直方向的滚动条的宽度必须为 16 像素，水平方向的滚动条的高度也必须为 16 像素。这些尺寸是由滚动条使用的当前主题所严格决定的。只有对应滚动条长度的尺寸能够改变。

您可以使用样式和外观来自定义 `UIScrollView` 实例的外表。

## 对 UIScrollView 组件使用样式

`UIScrollView` 组件支持下列样式：

样式	主题	说明
<code>themeColor</code>	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
<code>scrollTrackColor</code>	范例	滚动轨道的背景颜色。默认值为 <code>0xCCCCCC</code> （浅灰）。
<code>symbolColor</code>	范例	向上和向下滚动箭头的颜色。默认值为 <code>0x000000</code> （黑色）。
<code>symbolDisabledColor</code>	范例	禁用的滚动条中向上和向下滚动箭头的颜色。默认值为 <code>0x848384</code> （深灰）。

# 对 UIScrollBar 组件使用外观

UIScrollBar 组件可对轨道、滚动框（滑块）和按钮使用 13 种外观。要自定义这些外观元素，请编辑 `Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/States` 文件夹中的元件。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。

水平和垂直滚动条都使用相同的垂直外观，在显示水平滚动条时，UIScrollBar 组件会适当旋转外观。

UIScrollBar 组件支持下面的外观属性。

属性	说明
<code>upArrowUpName</code>	向上和向左按钮的弹起（正常）状态。默认值为 <code>ScrollUpArrowUp</code> 。
<code>upArrowOverName</code>	向上和向左按钮的滑过状态。默认值为 <code>ScrollUpArrowOver</code> 。
<code>upArrowDownName</code>	向上和向左按钮的按下状态。默认值为 <code>ScrollUpArrowDown</code> 。
<code>downArrowUpName</code>	向下和向右按钮的弹起（正常）状态。默认值为 <code>ScrollDownArrowUp</code> 。
<code>downArrowOverName</code>	向下和向右按钮的滑过状态。默认值为 <code>ScrollDownArrowOver</code> 。
<code>downArrowDownName</code>	向下和向右按钮的按下状态。默认值为 <code>ScrollDownArrowDown</code> 。
<code>scrollTrackName</code>	用于滚动条的轨道（背景）的元件。默认值为 <code>ScrollTrack</code> 。
<code>scrollTrackOverName</code>	用于滑过滚动轨道时的轨道（背景）的元件。默认值为 <code>undefined</code> 。
<code>scrollTrackDownName</code>	用于按下滚动轨道时的轨道（背景）的元件。默认值为 <code>undefined</code> 。
<code>thumbTopName</code>	滚动框（滑块）的顶端和左端。默认值为 <code>ScrollThumbTopUp</code> 。
<code>thumbMiddleName</code>	滑块的中间（可扩展）部分。默认值为 <code>ScrollThumbMiddleUp</code> 。
<code>thumbBottomName</code>	滑块的底端和右端。默认值为 <code>ScrollThumbBottomUp</code> 。
<code>thumbGripName</code>	显示在滑块之上的抓手。默认值为 <code>ScrollThumbGripUp</code> 。

以下示例演示如何将一条空白细线放在滚动轨道的中间。

## 创建影片剪辑元件以用于 UIScrollBar 外观：

1. 创建一个新的 FLA 文件。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 `HaloTheme.fla` 文件。  
此文件位于应用程序级配置文件夹中。若要了解此文件在您的操作系统中的确切位置，请参见《使用组件》中的“关于主题”。
3. 在主题的“库”面板中，展开 `Flash UI Components 2/Themes/MMDefault` 文件夹并将 `ScrollBar Assets` 文件夹拖动到文档的库中。
4. 在文档的库中展开 `ScrollBar Assets/States` 文件夹。

5. 打开要自定义的元件以进行编辑。  
例如，打开 `ScrollTrack` 元件。
6. 按需要自定义元件。  
例如，从 (8,0) 位置开始在轨道的中间绘制一个 1 x 4 的黑色矩形。
7. 对所有要自定义的元件重复步骤 5-6。  
例如，在 `ScrollTrackDisabled` 元件上绘制相同的线条。
8. 单击“返回”按钮返回主时间轴。
9. 在舞台上创建一个 `TextField` 输入类型实例。
10. 将一个 `UIScrollBar` 组件拖到 `TextField` 实例上。
11. 选择“控制” > “测试影片”。

## UIScrollBar 类

继承 `MovieClip` > `UIObject` 类 > `UIComponent` 类 > `ScrollBar` > `UIScrollBar`

ActionScript 类名称 `mx.controls.UIScrollBar`

使用 `UIScrollBar` 类的属性可以调整滚动位置，以及在用户单击滚动箭头或滚动轨道时发生的滚动量。

与其它大多数组件不同的是，当按下鼠标按键时，事件开始广播，一直到松开按键才结束。

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.controls.UIScrollBar.version);
```



代码 `trace(myUIScrollBarInstance.version);` 返回 `undefined`。



# UIScrollBar 类的方法摘要

下表列出了 UIScrollBar 类的方法。

方法	说明
<code>UIScrollBar.setScrollProperties()</code>	设置滚动条的滚动范围以及滚动条所附加到的文本字段的尺寸。
<code>UIScrollBar.setScrollTarget()</code>	将滚动条分配给文本字段。

## 从 UIObject 类继承的方法

下表列出了 UIScrollBar 类从 UIObject 类继承的方法。从 UIScrollBar 对象调用这些方法时，请使用 `UIScrollBarInstance.methodName` 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 UIScrollBar 类从 UIComponent 类继承的方法。从 UIScrollBar 对象调用这些方法时，请使用 `UIScrollBarInstance.methodName` 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# UISearchBar 类的属性摘要

下表列出了 UISearchBar 类的属性。

属性	说明
<code>UISearchBar.lineScrollSize</code>	当用户单击滚动条的箭头按钮时滚动的行数或像素数。
<code>UISearchBar.pageScrollSize</code>	当用户单击滚动条的轨道时滚动的行数或像素数。
<code>UISearchBar.scrollPosition</code>	滚动条当前的滚动位置。
<code>UISearchBar._targetInstanceName</code>	与 UISearchBar 实例相关联的文本字段的实例名称。
<code>UISearchBar.horizontal</code>	一个布尔值，指示滚动条是垂直方向 (false)（默认）还是水平方向 (true)。

## 从 UIObject 类继承的属性

下表列出了 UISearchBar 类从 UIObject 类继承的属性。从 UISearchBar 对象访问这些属性时，请使用 `UISearchBarInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	只读；对象的底边缘位置（相对于其父对象的底边缘）。
<code>UIObject.height</code>	只读；对象的高度，以像素为单位。
<code>UIObject.left</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.right</code>	只读；对象的右边缘位置（相对于其父对象的右边缘）。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	只读；对象上边缘的位置（相对于其父对象）。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

## 从 UICollection 类继承的属性

下表列出了 UIScrollView 类从 UICollection 类继承的属性。从 UIScrollView 对象访问这些属性时，请使用 `UIScrollViewInstance.propertyName` 的形式。

属性	说明
<code>UIScrollView.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIScrollView.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

## UIScrollView 类的事件摘要

下表列出了 UIScrollView 类的事件。

事件	说明
<code>UIScrollView.scroll</code>	在单击滚动条的任何部分时广播。

## 从 UIObject 类继承的事件

下表列出了 UIScrollView 类从 UIObject 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UICollection 类继承的事件

下表列出了 UIScrollView 类从 UICollection 类继承的事件。

事件	说明
<code>UICollection.focusIn</code>	当对象收到焦点时进行广播。
<code>UICollection.focusOut</code>	当对象失去焦点时进行广播。
<code>UICollection.keyDown</code>	当按下按键时进行广播。
<code>UICollection.keyUp</code>	当松开按键时进行广播。

# UIScrollBar.horizontal

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*scrollBarInstance*.horizontal

## 说明

属性；指示滚动条是垂直方向 (false) 还是水平方向 (true)。

可以测试和设置此属性。默认值为 false。

## 示例

以下示例使用 horizontal 属性将名为 my\_sb 的滚动条设置为水平方向，并在 TextField 组件 my\_txt 中显示文本：

```
/**
 * 要求：
 * - 库中有 UIScrollBar 组件
 */
// 创建文本字段。
this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = false;

my_txt.text = "Mary had a little lamb whose fleece " +
"was white as snow and everywhere that Mary went the " +
"lamb was sure to go."

// 创建滚动条。
this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);
my_sb.horizontal = true;

// 为滚动条设置目标文本字段。
my_sb.setScrollTarget(my_txt);
// 调整其尺寸以匹配文本字段。
my_sb.setSize(my_txt._width, 16);

// 将其移动到文本字段的底部。
my_sb.move(my_txt._x, my_txt._y + my_txt._height);
```

# UIScrollBar.lineScrollSize

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*scrollBarInstance*.lineScrollSize

## 说明

属性；获取或设置当用户单击 **UIScrollBar** 组件的箭头按钮时滚动的行数或像素数。如果滚动条为垂直方向，则值为行数。如果滚动条为水平方向，则值为像素数。

默认值为 1。

## 示例

以下示例创建一个滚动条以便滚动文本字段中的文本，而该字段从网页中加载内容。该示例将 lineScrollSize 属性设置为每次单击箭头按钮时滚动两行：

```
/**
 * 要求：
 * - 库中有 UIScrollBar 组件
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// 设置目标文本字段。
my_sb.setScrollTarget(my_txt);

// 调整其尺寸以匹配文本字段。
my_sb.setSize(16, my_txt._height);

// 将其移动到文本字段的旁边。
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 每次单击滚动箭头时滚动 2 行。
my_sb.lineScrollSize = 2;

// 每次单击滚动轨道时滚动 5 行。
my_sb.pageScrollSize = 5;

// 加载要显示的文本，并定义 onData 处理函数。
```

```

var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 } else {
 my_txt.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

```

## UIScrollBar.pageScrollSize

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*scrollBarInstance.pageScrollSize*

### 说明

属性；获取或设置当用户单击 **UIScrollBar** 组件的轨道时滚动的行数或像素数。如果滚动条为垂直方向，则值为行数。如果滚动条为水平方向，则值为像素数。

您也可以通过使用 **UIScrollBar.setScrollTarget()** 方法传递 *pageSize* 参数来设置此值。

### 示例

以下示例创建一个滚动条以便滚动文本字段中的文本，该字段从网页中加载内容。该示例将 *pageScrollSize* 属性设置为用户每次单击滚动轨道时滚动五行文本：

```

/**
 * 要求：
 * - 库中有 UIScrollBar 组件
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// 设置目标文本字段。
my_sb.setScrollTarget(my_txt);

// 调整其尺寸以匹配文本字段。

```

```

my_sb.setSize(16, my_txt._height);

// 将其移动到文本字段的旁边。
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 每次单击滚动箭头时滚动 2 行。
my_sb.lineScrollSize = 2;

// 每次单击滚动轨道时滚动 5 行。
my_sb.pageScrollSize = 5;

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 } else {
 my_txt.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

```

## UIScrollBar.scroll

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

#### 用法 1:

```

var listenerObject:Object = new Object();
listenerObject.scroll = function(eventObject:Object) {
 // ...
};
scrollBarInstance.addEventListener("scroll", listenerObject)

```

#### 用法 2:

```

on (scroll) {
 // ...
}

```

## 说明

事件：在滚动条上单击（松开）鼠标时向所有已注册的侦听器广播。在广播此事件前，会更新 `UIScrollBar.scrollPosition` 属性和滚动条在屏幕上的图像。

第一个用法示例使用了一个调度程序 / 侦听器事件模型，在其中，脚本放置在时间轴中包含组件实例的帧上。组件实例 (`scrollBarInstance`) 调度一个事件（在本示例中为 `scroll`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件发生时，就会调用该方法。该事件发生时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()`（请参见 `EventDispatcher.addEventListener()`），以便将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

除了事件对象的常规属性（`type` 和 `target`）外，`scroll` 事件的事件对象还包含一个名为 `direction` 的另一个属性。`direction` 属性包含一个字符串，用于描述滚动条的滚动方向。`direction` 属性的可能值为 `vertical`（默认）和 `horizontal`。

有关 `type` 和 `target` 事件对象属性的更多信息，请参见第 461 页的“事件对象”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到 `UIScrollBar` 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 `UIScrollBar` 组件实例 `myUIScrollBarComponent`，它将“\_level0.myUIScrollBarComponent”发送到“输出”面板：

```
on (scroll) {
 trace(this);
}
```

## 示例

以下示例实现的是用法 1，并创建一个具有 `scroll` 事件处理函数的名为 `sbListener` 的侦听器对象：

```
/**
 要求：
 - 库中有 UIScrollBar 组件
*/

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// 设置目标文本字段。
my_sb.setScrollTarget(my_txt);

// 调整其尺寸以匹配文本字段。
```



```

my_sb.setSize(16, my_txt._height);

// 将其移动到文本字段的旁边。
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 创建侦听器对象。
var sbListener:Object = new Object();
sbListener.scroll = function(evt_obj:Object){
 // 插入代码以便处理 “scroll” 事件。
 trace("text is scrolling");
}
// 添加侦听器。
my_sb.addEventListener("scroll", sbListener);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 } else {
 my_txt.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

```

以下代码实现的是用法 2。代码附加到 **UIScrollBar** 组件实例中，并在用户单击滚动条时将一条消息发送到“输出”面板。on() 处理函数必须直接附加到 **UIScrollBar** 实例。

```

on (scroll) {
 trace("UIScrollBar component was clicked");
}

```

## UIScrollBar.scrollPosition

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*scrollBarInstance.scrollPosition*

## 说明

属性：当设置了新的 `scrollTopPosition` 值时，获取或设置滚动框（缩略图）的当前滚动位置。`scrollTopPosition` 的值取决于 **UIScrollBar** 实例是用于垂直滚动还是水平滚动。

使用以下语法单独设置滚动条目标实例的滚动：

```
my_scrollbar._targetInstanceName.scroll = 20;
```

如果 **UIScrollBar** 实例用于垂直滚动（最常见的用法），则 `scrollTopPosition` 的值为一定范围内的某个整数，其最小值为 **0**，最大值为文本字段中的总行数与文本字段中可同时显示的行数之商。如果 `scrollTopPosition` 设置为一个超出此范围的数字，则文本字段只滚动到文本的末尾。

若要将滚动框（缩略图）滚动到第一行，可将 `scrollTopPosition` 设置为 **0**。

若要将滚动框（缩略图）滚动到末尾，可将 `scrollTopPosition` 设置为文本字段中文本的行数减 **1**。通过检索文本字段的 `maxscroll` 属性值可以确定行数。

如果 **UIScrollBar** 实例用于水平滚动，则 `scrollTopPosition` 的值为介于 **0** 到文本字段宽度之间的一个整数值（以像素为单位）。通过获取文本字段的 `maxhscroll` 属性值可以确定文本字段的宽度（以像素为单位）。

`scrollTopPosition` 的默认值为 **0**。

## 示例

以下示例将文本设置为位置 **20**：

```
/**
 * 要求：
 * - 库中有 UIScrollBar 和 Button 组件
 */
this.createTextField("my_txt", 10, 10, 20, 200, 100);
this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);
this.createClassObject(mx.controls.Button, "my_bt", 30, {label: "Scroll"});

my_txt.wordWrap = true;
my_bt.move(300, 100);

// 设置目标文本字段。
my_sb.setScrollTarget(my_txt);

// 将其移动到文本字段的旁边。
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();

my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 }
}
```

```
 } else {
 my_txt.text = "Error loading text.";
 }
};

my_lv.load("http://www.helpexamples.com/flash/lorem.txt");

var scroll_listener = new Object();
scroll_listener.click = function() {
 my_sb.scrollTop = 20;
 my_txt.scroll = 20;
};
my_bt.addEventListener("click", scroll_listener);
```

## UIScrollBar.setScrollProperties()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

```
scrollBarInstance.setScrollProperties(pageSize, minPos, maxPos)
```

### 参数

*pageSize* 显示区域中可查看的项目数。此参数设置文本字段的边框尺寸。如果滚动条为垂直的，则此值为文本的行数；如果滚动条为水平的，则此值为像素数。

*minPos* 此参数指垂直使用滚动条时文本的最低编号行；或水平使用滚动条时文本字段边框中的最低编号像素。该值通常为 0。

*maxPos* 此值指垂直使用滚动条时文本的最高编号行；或水平使用滚动条时文本字段边框中的最高编号像素。

### 说明

方法；设置滚动条的滚动范围以及滚动条所附加到的文本字段的尺寸。此方法主要在运行时（而不是在创作时）使用 [UIScrollBar.setScrollTarget\(\)](#) 将 **UIScrollBar** 组件附加到文本字段时使用，并且赋值不会导致文本字段广播 **change** 事件。如果使用 `replaceText` 方法设置文本字段的文本，则必须使用 `setScrollProperties()` 以使滚动条更新。

**UIScrollBar** 组件结合使用 `minPos` 和 `maxPos` 值来确定滚动条以及相关文本字段的滚动范围。

## 示例

以下示例将 `UIScrollBar` 组件设置为在文本字段中一次显示 10 行文本（文本字段包含的行数范围为 0 - 99）：

```
my_sb.setScrollProperties(10, 0, 99);
```

# UIScrollBar.setScrollTarget()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
scrollBarInstance.setScrollTarget(textInstance)
```

## 参数

*textInstance* 分配给滚动条的文本字段。

## 说明

方法：将 `UIScrollBar` 组件分配给一个文本字段实例。如果需要在运行时将文本字段和 `UIScrollBar` 组件关联起来，请使用此方法。

## 示例

以下示例创建一个滚动条以便滚动文本字段中的文本，而该字段从网页中加载内容。该示例调用 `setScrollTarget()` 方法，以使滚动条 `my_sb` 与文本字段 `my_txt` 关联起来。

```
/**
 * 要求：
 * - 库中有 UIScrollBar 组件
 */

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// 设置目标文本字段。
my_sb.setScrollTarget(my_txt);

// 调整其尺寸以匹配文本字段。
my_sb.setSize(16, my_txt._height);

// 将其移动到文本字段的旁边。
```

```
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 每次单击滚动箭头时滚动 2 行。
my_sb.lineScrollSize = 2;

// 每次单击滚动轨道时滚动 5 行。
my_sb.pageScrollSize = 5;

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 } else {
 my_txt.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

## UIScrollBar.\_targetInstanceName

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX 2004。

### 用法

*scrollBarInstance*.\_targetInstanceName

### 说明

属性；指明与 **UIScrollBar** 组件关联的文本字段的实例名称。可以测试和设置此属性。但是，不应使用此属性创建文本字段和滚动条之间的关联。创建关联时请改用

[UIScrollBar.setScrollTarget\(\)](#)。

## 示例

以下示例创建一个滚动条以便滚动文本字段中的文本，而该字段从网页中加载内容。该示例调用 `trace()` 函数 以显示 `targetInstanceName` 属性的值。

```
/**
 要求:
 - 库中有 UIScrollBar 组件
*/

this.createTextField("my_txt", 10, 10, 20, 200, 100);
my_txt.wordWrap = true;

this.createClassObject(mx.controls.UIScrollBar, "my_sb", 20);

// 设置目标文本字段。
my_sb.setScrollTarget(my_txt);

trace(my_sb._targetInstanceName);

// 调整其尺寸以匹配文本字段。
my_sb.setSize(16, my_txt._height);

// 将其移动到文本字段的旁边。
my_sb.move(my_txt._x + my_txt._width, my_txt._y);

// 设置滚动属性。
my_sb.setScrollProperties(10, 0, 99);

// 加载要显示的文本，并定义 onData 处理函数。
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
 if (src != undefined) {
 my_txt.text = src;
 my_txt.condenseWhite = true;
 } else {
 my_txt.text = "Error loading text.";
 }
};
my_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

# Web 服务类（仅限 Flash Professional）


使用 `mx.services` 包中的 Web 服务类能够访问使用简单对象访问协议 (SOAP) 的 Web 服务。此 API 不同于 `WebServiceConnector` 组件 API。Web 服务 API 是只能在 ActionScript 代码中使用的一组类，在各种 Macromedia 产品中很常见。而 `WebServiceConnector` 组件是 Flash 所独有的 API，它对可视 `WebServiceConnector` 组件提供了一个 ActionScript 接口。

下表列出了 `mx.services` 包中的类。这些类紧密地集成在一起，因此当您初次了解这个包时，可能要按照下表所列顺序来阅读信息。

类	说明
<a href="#">WebService 类（仅限 Flash Professional）</a>	通过使用定义了 Web 服务的 Web 服务定义语言 (WSDL) 文件，为调用 Web 服务的方法和处理 Web 服务的回调构建新的 <code>WebService</code> 对象。
<a href="#">PendingCall 类（仅限 Flash Professional）</a>	从 Web 服务的方法调用返回的对象，您实现该对象以处理调用的结果和错误。
<a href="#">Log 类（仅限 Flash Professional）</a>	用于记录与 <code>WebService</code> 对象相关的活动的可选对象。
<a href="#">SOAPCall 类（仅限 Flash Professional）</a>	高级的类，它包含有关 Web 服务操作的信息并提供对某些行为的控制。

# 使 Web 服务类在运行时可用（仅限 Flash Professional）

为了能够在运行时使用 Web 服务类，WebServiceConnector 组件必须位于 FLA 文件的库中。此组件包含运行时类，这些类可让您使用 Web 服务。有关向 FLA 文件添加这些类的详细信息，请参见《使用 Flash》中的第 16 章“数据集成（仅限 Flash Professional）”。

 当您向 FLA 文件添加 WebServiceConnector 组件时，会自动使这些类对于 Flash 文档可用。

## Log 类（仅限 Flash Professional）

ActionScript 类名称 mx.services.Log

Log 类是 mx.services 包的一部分，与 WebService 类一起使用（请参见第 1320 页的“WebService 类（仅限 Flash Professional）”）。有关 mx.data.services 包中类的概述，请参见第 1299 页的“Web 服务类（仅限 Flash Professional）”。

可以创建新的 Log 对象，以记录与 WebService 对象相关的活动。若要在向 Log 对象发送消息时执行代码，请使用 Log.onLog() 回调函数。并不存在日志文件；记录机制是您已在 onLog() 回调函数中使用的任何举措，例如向 trace() 语句发送日志消息。

此类的构造函数创建一个 Log 对象，该对象可作为可选参数传递给 WebService 构造函数（请参见第 1320 页的“WebService 类（仅限 Flash Professional）”）。

### Log 类的方法摘要

下表列出了 PendingCall 类的方法。

方法	说明
<code>Log.getDateString()</code>	按 Log 消息所使用的以下格式以字符串的形式返回当前的日期和时间：mm/dd hh:mm:ss。
<code>Log.logInfo()</code>	用指定的日志级别和指定的消息生成 Log.onLog 事件。
<code>Log.logDebug()</code>	用 Log.DEBUG 日志级别和指定的消息生成 Log.onLog 事件。



## Log 对象的属性摘要

下表列出了 PendingCall 类的属性。

属性	说明
<code>Log.level</code>	要在日志中记录的信息的类别。
<code>Log.name</code>	一个字符串名称，它用于标识 Log 对象，并包括在每条 Log.onLog 事件消息中。

## Log 对象的回调摘要

下表列出了 Log 对象的回调。

回调	说明
<code>Log.onLog()</code>	在向日志文件发送日志消息时由 Flash Player 调用。

## Log 类的构造函数

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
myWebSrvLog = new Log([logLevel] [, logName]);
```

### 参数

*logLevel* 用于指示您想在日志中记录的信息类别的级别。可以使用四个日志级别：

- `Log.BRIEF` 日志将记录主要的生命周期事件和错误通知。这是默认值。
- `Log.VERBOSE` 日志将记录所有的生命周期事件和错误通知。
- `Log.DEBUG` 日志将记录度量标准和精细的事件和错误。
- `Log.NONE` 日志不记录任何内容。可用于暂时关闭 Log.onLog 事件。

*logName* 随附于每条日志消息的可选名称。如果您使用了多个 **Log** 对象，则可以使用日志名称来确定哪个日志记录了给定的消息。

### 返回

无。

## 说明

构造函数：创建一个 **Log** 对象。创建了 **Log** 对象后，可以将它传递给 **Web** 服务以获取消息。

## 示例

可以调用 `new Log` 构造函数，以返回一个 **Log** 对象传递给 **Web** 服务：

```
// 创建新的 Log 对象。
import mx.services.*;
myWebSvcLog = new Log();
myWebSvcLog.onLog = function(msg : String) : Void
{
 myTrace(txt)
}
```

然后，将此 **Log** 对象作为参数传递给 **WebService** 构造函数：

```
myWebSvc = new WebService("http://www.myco.com/info.wsdl", myWebSvcLog);
```

在执行 **Web** 服务代码以及向此 **Log** 对象发送消息时，会调用 **Log** 对象的 `onLog()` 函数。

如果您想实时看到 **Log** 消息，这是唯一可以放置显示这些消息的代码的地方。

以下是 **Log** 消息的示例：

```
7/30 15:22:43 [INFO] SOAP: Decoding PendingCall response
7/30 15:22:43 [DEBUG] SOAP: Decoding SOAP response envelope
7/30 15:22:43 [DEBUG] SOAP: Decoding SOAP response body
7/30 15:22:44 [INFO] SOAP: Decoded SOAP response into result [16 millis]
7/30 15:22:46 [INFO] SOAP: Received SOAP response from network [6469 millis]
7/30 15:22:46 [INFO] SOAP: Parsed SOAP response XML [15 millis]
7/30 15:22:46 [INFO] SOAP: Decoding PendingCall response
7/30 15:22:46 [DEBUG] SOAP: Decoding SOAP response envelope
7/30 15:22:46 [DEBUG] SOAP: Decoding SOAP response body
7/30 15:22:46 [INFO] SOAP: Decoded SOAP response into result [16 millis]
```

# Log.getDateString()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myWebSvcLog.getDateString()
```

## 参数

无。

## 返回

按以下格式的字符串形式的当前日期和时间：mm/dd hh:mm:ss。

## 说明

函数：按以下格式以字符串形式返回当前日期和时间：mm/dd hh:mm:ss。可以使用 `Log.getDateString()` 来获取日期，日期的格式与日志消息中提供的格式相同，也可以只在 `log.onLog` 事件处理函数中记录日期字符串以便用于自定义日志处理。

## 示例

以下示例创建一个新的 **Log** 对象，将其传递给一个新的 **WebService** 对象，然后使用 `Log.getDateString()` 处理日志信息以获取日志时间。

```
import mx.services.*;
// 创建新的 Log 对象。
myWebSvcLog = new Log(Log.BRIEF, "myLog");
// 将 Log 对象传递给 Web 服务。
myWebService = new WebService(wsdlURI, myWebSvcLog);
// 处理传入的日志消息。
myWebSvcLog.onLog = function(message : String) : Void
{
 trace("A Log Event Occurred At This Time: "+ this.getDateString());
}
```

# Log.logInfo()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myWebSvcLog.logInfo(myMessageString)
```

## 参数

*msg* 希望显示在结果日志事件消息中的字符串类型的消息。

*level* 用于指示您想在日志中记录的信息类别的级别。可以使用四个日志级别：

- `Log.BRIEF` 日志将记录主要的生命周期事件和错误通知。这是默认值。
- `Log.VERBOSE` 日志将记录所有的生命周期事件和错误通知。
- `Log.DEBUG` 日志将记录度量标准和精细的事件和错误。
- `Log.NONE` 日志不记录任何内容。可用于暂时关闭 `Log.onLog` 事件。

## 返回

无。

## 说明

函数：在 `level` 参数设置的某个日志级别生成由 `msg` 参数设置的日志消息。此方法提供了以任何日志级别创建自己的日志事件的一种方式。

## 示例

以下示例创建一个新的 **Log** 对象，即包含消息的 `onLog` 事件，该消息指示已通过调用 `Log.logDebug()` 开始生成一个新日志。

```
import mx.services.*;
// 创建新的 Log 对象。
myWebSvcLog = new Log(Log.VERBOSE, "myLog");

// 处理传入的日志消息。
myWebSvcLog.onLog = function(message : String) : Void
{
 trace(message);
}
myWebSvcLog.logInfo("New Log Started");
// 将 Log 对象传递给 Web 服务。
myWebService = new WebService(wsdlURI, myWebSvcLog);
```

# Log.logDebug()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myWebSvcLog.logDebug(msg)
```

## 参数

*msg* 日志消息字符串。您在此参数中提供的字符串在结果日志事件中显示为日志消息。

## 返回

无。

## 说明

函数：生成一条包含 msg 和消息类型指示器 [debug] 的消息。此方法为创建日志消息中具有 [debug] 的自定义日志事件提供了一种途径，只有当日志级别设置为 Log.DEBUG 时才能查看这些日志消息。

以下字符串是由 Log.logDebug() 生成的调试级别日志消息的示例：

```
12/18 23:20:17 [DEBUG] myLog: My log message
```

## 示例

以下示例创建一个新的 **Log** 对象，即包含消息的 onLog 事件，该消息指示已通过调用 Log.logDebug() 开始生成一个新日志。

```
import mx.services.*;
// 创建新的 Log 对象。
myWebSrvLog = new Log(Log.DEBUG, "myLog");

// 处理传入的日志消息。
myWebSrvLog.onLog = function(message : String) : Void
{
 trace(message);
}
// 生成日志级别为 Log.DEBUG 的日志消息。
myWebSrvLog.logDebug("New Log Started");
// 将 Log 对象传递给 Web 服务。
myWebService = new WebService(wsdlURI, myWebSrvLog);
```

# Log.level

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myLevel_Number = myWebSrvLog.level
```

## 说明

属性：指示要在日志中记录的信息的类别。可以使用四个日志级别：

- `Log.BRIEF` 日志将记录主要的生命周期事件和错误通知。这是默认值。设置为 `Log.BRIEF` 的 `Log.level` 属性返回数字 `0`。
- `Log.VERBOSE` 日志将记录所有的生命周期事件和错误通知。设置为 `Log.VERBOSE` 的 `Log.level` 属性返回数字 `1`。
- `Log.DEBUG` 日志将记录度量标准和精细的事件和错误。设置为 `Log.DEBUG` 的 `Log.level` 属性返回数字 `2`。
- `Log.NONE` 日志不记录任何内容。可用于暂时关闭 `Log.onLog` 事件。设置为 `Log.NONE` 的 `Log.level` 属性返回数字 `-1`。

尽管可以直接设置此属性，但在创建一个新 **Log** 对象时通常以参数的形式设置 `Log.level`。请参见第 1300 页的“**Log** 类（仅限 Flash Professional）”。

## 示例

以下示例创建一个 `Log.level` 属性为 `Log.DEBUG` 的新 **Log** 对象，并跟踪当前的 `Log.level` 属性。然后将 **Log** 对象的 `Log.level` 属性设置为 `Log.VERBOSE`。

```
import mx.services.*;
// 创建新的 Log 对象。
myWebSvcLog = new Log(Log.DEBUG, "myLog");
trace("myWebSvcLog.level: "+ myWebSvcLog.level);

// 现在更改 Log 对象的级别。
myWebSvcLog.level = Log.VERBOSE;
trace("myWebSvcLog.level: "+ myWebSvcLog.level);
```

# Log.name

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myWebServiceName = myWebSvcLog.name
```

## 说明

属性：一个字符串，它用于标识 **Log** 实例，并包括在每条 `Log.onLog` 事件消息中。可以获取和设置此属性。在创建新的 **Log** 对象时通常设置该属性。请参见第 1300 页的“**Log** 类（仅限 Flash Professional）”。

## 示例

以下示例创建一个 `Log.level` 属性为 `Log.VERBOSE`、名称为 “**myLog**” 的新 **Log** 对象，并跟踪当前的 `Log.name` 属性。然后将 **Log** 对象的 `Log.name` 属性设置为 “myNewLogName”。

```
import mx.services.*;
// 创建新的 Log 对象。
myWebSvcLog = new Log(Log.VERBOSE, "myLog");
trace("myWebSvcLog.level: " + myWebSvcLog.level);

// 为 Log 对象设置一个新名称。
myWebSvcLog.name = "myNewLogName";
trace("myWebSvcLog.name: " + myWebSvcLog.name);
```

# Log.onLog()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myWebSvcLog.onLog = function(message)
```

## 参数

*message* 传递给处理函数的日志消息。例如：

```
“7/30 15:22:43 [INFO] SOAP: Decoding PendingCall response”
```

## 返回

无。

## 说明

回调函数：在向日志文件发送日志消息时由 **Flash Player** 调用。此函数适合于放置用于记录或显示日志消息的代码（如 `trace` 命令）。（有关日志的结构的信息，请参见第 1300 页的“**Log** 类（仅限 **Flash Professional**）”。）

## 示例

以下示例创建一个新的 **Log** 对象，将其传递给新的 **WebService** 对象，并处理日志消息：

```
import mx.services.*;
// 创建新的 Log 对象。
myWebSvcLog = new Log();
// 将 Log 对象传递给 Web 服务。
myWebService = new WebService(wsdlURI, myWebSvcLog);
// 处理传入的日志消息。
myWebSvcLog.onLog = function(message : String) : Void
{
 mytrace("myWebSvcLog.message: " + message);
}
```

# PendingCall 类（仅限 Flash Professional）

**ActionScript 类名称** mx.services.PendingCall

**PendingCall** 类是 **mx.services** 包的一部分，与 **WebService** 类一起使用。有关 **mx.services** 包中类的概述，请参见第 1299 页的“**Web 服务类（仅限 Flash Professional）**”。

您不必创建 **PendingCall** 对象或使用构造函数；当对 **WebService** 对象调用方法时，**WebService** 方法会返回一个 **PendingCall** 对象。请使用 **PendingCall.onResult** 和 **PendingCall.onFault** 回调函数来处理来自 **Web** 服务方法的异步响应。如果 **Web** 服务方法返回错误，则 **Flash Player** 会调用 **PendingCall.onFault**，并传递 **SOAPFault** 对象，该对象代表由服务器或 **Web** 服务返回的 XML SOAP 错误。**SOAPFault** 对象不是由您直接构建的，但作为失败的结果返回。该对象是 **SOAPFault XML** 类型的 **ActionScript** 映射。

如果成功调用 **Web** 服务，则 **Flash Player** 会调用 **PendingCall.onResult** 并传递结果对象。结果对象是来自在 **ActionScript** 中解码或反序列化的 **Web** 服务的 XML 响应。有关 **WebService** 对象的更多信息，请参见第 1320 页的“**WebService 类（仅限 Flash Professional）**”。

当 **Web** 服务方法返回不止一个结果时，**PendingCall** 对象还可以访问多个输出参数。在此 API 中，“返回值”只是指第一个（或唯一一个）结果；若要访问所有的结果，您可以使用“获取输出”函数。例如，当在 **onResult** 回调的参数中传递给您的返回值不是您要访问的唯一值时，可以使用 **getOutputValues()**（返回一个数组）或 **getOutputValue()**（返回单个值）来获取经过 **ActionScript** 解码的值。

也可以直接访问 **SOAPParameter** 对象。**SOAPParameter** 对象是具有两个属性的 **ActionScript** 对象：**value**（输出参数的 **ActionScript** 值）和 **element**（输出参数的 XML 值）。以下函数返回一个 **SOAPParameter** 对象或 **SOAPParameter** 对象的一个数组：**getOutputParameters()**、**getOutputParameterByName()** 和 **getOutputParameter()**。



## PendingCall 类的方法摘要

下表列出了 PendingCall 类的方法。

方法	说明
<code>PendingCall.getOutputParameter()</code>	按索引检索一个 SOAPParameter 对象。
<code>PendingCall.getOutputParameterByName()</code>	按名称检索一个 SOAPParameter 对象。
<code>PendingCall.getOutputParameters()</code>	检索 SOAPParameter 对象的一个数组。
<code>PendingCall.getOutputValue()</code>	按照指定的索引检索输出值。
<code>PendingCall.getOutputValues()</code>	检索所有输出值的一个数组。

## PendingCall 对象的属性摘要

下表列出了 PendingCall 类的属性。

属性	说明
<code>PendingCall.myCall</code>	用于 PendingCall 操作的 SOAPCall 操作描述符。
<code>PendingCall.request</code>	XML 原始格式的 SOAP 请求。
<code>PendingCall.response</code>	XML 原始格式的 SOAP 响应。

## PendingCall 对象的回调摘要

下表列出了 PendingCall 类的回调。

回调	说明
<code>PendingCall.onFault</code>	在 Web 服务方法已失败且返回错误时由 Flash Player 调用。
<code>PendingCall.onResult</code>	在方法已成功且返回结果时调用。

# PendingCall.getOutputParameter()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCall.getOutputParameter(index)
```

## 参数

*index* 参数的从零开始的索引。

## 返回

一个 **SOAPParameter** 对象，它具有两个属性：value（输出参数的 **ActionScript** 值）和 element（输出参数的 **XML** 值）。

## 说明

函数：获取 **SOAPParameter** 对象的另一个输出参数，包含值和 XML 元素。SOAP RPC 调用可以返回多个输出参数。第一个（或唯一一个）返回值总是在 `onResult` 回调函数的 *result* 参数中传送，但为了访问其它返回值，您必须使用如 `getOutputParameter()` 和 `getOutputValue()` 等函数。`getOutputParameter()` 函数返回第 **n** 个输出参数作为 **SOAPParameter** 对象。

## 示例

在给定以下 **SOAP** 描述符文件的情况下，`getOutputParameter(1)` 会返回一个满足 `value="Hi there!"` 和 `element=the <outParam2> XMLNode` 的 **SOAPParameter** 对象。

```
...
<SOAP:Body>
 <rpcResponse>
 <outParam1 xsi:type="xsd:int">54</outParam1>
 <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
 <outParam3 xsi:type="xsd:boolean">true</outParam3>
 </rpcResponse>
</SOAP:Body>
...
```

## 另请参见

[PendingCall.getOutputParameterByName\(\)](#)、[PendingCall.getOutputParameters\(\)](#)、[PendingCall.getOutputValue\(\)](#)、[PendingCall.getOutputValues\(\)](#)

# PendingCall.getOutputParameterByName()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCall.getOutputParameterByName(var localName)
```

## 参数

*localName* 参数的本地名称。也就是去掉了所有命名空间信息的 XML 元素的名称。例如，以下两个元素的本地名称都是 bob：

```
<bob abc="123">
<xsd:bob def="ghi">
```

## 返回

一个 **SOAPParameter** 对象，它具有两个属性：value（输出参数的 **ActionScript** 值）和 element（输出参数的 XML 值）。

## 说明

函数：获取作为 **SOAPParameter** 对象的任何输出参数，包含值和 XML 元素。SOAP RPC 调用可以返回多个输出参数。第一个（或唯一一个）返回值总是在 `onResult` 回调函数的 *result* 参数中传送，但为了访问其它返回值，您必须使用 `getOutputParameterByName()` 之类的函数。此函数返回带有名称 *localName* 的输出参数。

## 示例

在给定以下 SOAP 描述符文件的情况下，`getOutputParameterByName("outParam2")` 会返回一个满足 `value="Hi there!"` 和 `element=the <outParam2> XMLNode` 的 **SOAPParameter** 对象。

```
...
<SOAP:Body>
 <rpcResponse>
 <outParam1 xsi:type="xsd:int">54</outParam1>
 <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
 <outParam3 xsi:type="xsd:boolean">true</outParam3>
 </rpcResponse>
</SOAP:Body>
...
```

## 另请参见

[PendingCall.getOutputParameter\(\)](#)、[PendingCall.getOutputParameters\(\)](#)、  
[PendingCall.getOutputValue\(\)](#)、[PendingCall.getOutputValues\(\)](#)

# PendingCall.getOutputParameters()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCall.getOutputParameters()
```

## 参数

无。

## 返回

一个 **SOAPParameter** 对象，它具有两个属性：**value**（输出参数的 **ActionScript** 值）和 **element**（输出参数的 **XML** 值）。

## 说明

函数；获取 **SOAPParameter** 对象的其它输出参数，包含值和 **XML** 元素。**SOAP RPC** 调用可以返回多个输出参数。第一个（或唯一一个）返回值总是在 **onResult** 回调函数的 *result* 参数中传送，但为了访问其它返回值，您必须使用如 **getOutputParameters()** 和 **getOutputValues()** 等函数。

## 另请参见

[PendingCall.getOutputParameterByName\(\)](#)、[PendingCall.getOutputParameter\(\)](#)、[PendingCall.getOutputValue\(\)](#)、[PendingCall.getOutputValues\(\)](#)

# PendingCall.getOutputValue()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCall.getOutputValue(var index)
```

## 参数

*index* 输出参数的索引。第一个参数的索引是 0。

## 返回

第 *n* 个输出参数。

## 说明

函数：获取某个输出参数的已解码 **ActionScript** 值。SOAP RPC 调用可以返回多个输出参数。第一个（或唯一一个）返回值总是在 *onResult* 回调函数的 *result* 参数中传送，但为了访问其它返回值，您必须使用如 `getOutputValue()` 和 `getOutputParameter()` 等函数。`getOutputValue()` 函数返回第 *n* 个输出参数。

## 示例

在给定以下 SOAP 描述符文件的情况下，`getOutputValue(2)` 会返回 `true`。

```
...
<SOAP:Body>
 <rpcResponse>
 <outParam1 xsi:type="xsd:int">54</outParam1>
 <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
 <outParam3 xsi:type="xsd:boolean">true</outParam3>
 </rpcResponse>
</SOAP:Body>
...
```

## 另请参见

[PendingCall.getOutputParameterByName\(\)](#)、[PendingCall.getOutputParameter\(\)](#)、[PendingCall.getOutputParameters\(\)](#)、[PendingCall.getOutputValues\(\)](#)

# PendingCall.getOutputValues()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCall.getOutputValues()
```

## 参数

无。

## 返回

所有输出参数的已解码值的数组。

## 说明

函数：获取所有输出参数的已解码 **ActionScript** 值。SOAP RPC 调用可以返回多个输出参数。第一个（或唯一一个）返回值总是在 `onResult` 回调函数的 *result* 参数中传送，但为了访问其它返回值，您必须使用如 `getOutputValues()` 和 `getOutputParameters()` 等函数。

## 另请参见

[PendingCall.getOutputParameterByName\(\)](#)、[PendingCall.getOutputParameter\(\)](#)、[PendingCall.getOutputParameters\(\)](#)、[PendingCall.getOutputValue\(\)](#)

# PendingCall.myCall

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*PendingCall*.myCall

## 说明

属性：与 **PendingCall** 操作对应的 **SOAPCall** 对象。此 **SOAPCall** 对象包含有关 Web 服务操作的信息并提供对某些行为的控制。有关更多信息，请参见第 1318 页的“**SOAPCall** 类（仅限 **Flash Professional**）”。

## 示例

下面的 `onResult` 回调跟踪 **SOAPCall** 操作的名称。

```
callback.onResult = function(result)
{
 // 检查我的操作名称。
 trace("My operation name is " + this.myCall.name);
}
```

# PendingCall.onFault

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCallObj.onFault = function(fault)
{
 // 此处是您的代码。
}
```

## 参数

*fault* 具有属性的 **SOAPFault** 对象的已解码 **ActionScript** 对象版本。如果 XML 形式的错误信息来自某一服务器，则 **SOAPFault** 对象将是该 XML 的已解码 **ActionScript** 版本。返回到 **PendingCall.onFault** 的错误对象的类型是 **SOAPFault** 对象。该对象不是由您直接构建的，但作为失败的结果返回。该对象是 **SOAPFault** XML 类型的 **ActionScript** 映射。

SOAPFault 属性	说明
<i>faultcode</i>	字符串；一个描述错误的短字符串。
<i>faultstring</i>	字符串；可由人直接读取的关于错误的说明。
<i>detail</i>	字符串；与错误相关的特定于应用程序的信息，例如由 Web 服务引擎返回的堆栈跟踪或其它信息。
<i>element</i>	XML；表示错误的 XML 版本的 XML 对象。
<i>faultactor</i>	字符串；错误的来源（如果未涉及中间项，则为可选）。

## 返回

无。

## 说明

回调函数；此函数由您提供，由 Flash Player 在 Web 服务方法失败并返回错误时调用。*fault* 参数是一个 **ActionScript** **SOAPFault** 对象。

此位置适合于放置任何错误处理代码，例如，在适当的时候通知用户服务器不可用，或请用户与技术支持部门联系。

## 示例

以下范例处理从 Web 服务方法返回的错误。

```
// 处理在使用 Web 服务方法时返回的所有错误。
myPendingCallObj = myWebService.methodName(params)
myPendingCallObj.onFault = function(fault)
{
 // 捕获 SOAP 错误。
 DebugOutputField.text = fault.faultstring;

 // 添加代码以处理任何错误，例如，通过
 // 通知用户服务器不可用或者与技术支持部门
 // 联系。
}
```

# PendingCall.onResult

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myPendingCallObj.onResult = function(result)
{
 // 此处是您的代码。
}
```

## 参数

*result* 由通过 `myPendingCallObj = myWebService.methodName(params)` 方式调用的 Web 服务方法返回的 XML 结果（已解码的 **ActionScript** 对象版本）。

## 返回

无。

## 说明

回调函数：此函数由您提供，由 Flash Player 在 Web 服务方法成功并返回结果时调用。结果是由操作返回的 XML 的已解码 **ActionScript** 对象版本。在此函数中放入根据结果执行适当操作的代码。要返回原始 XML 而不是已解码的结果，请访问 `PendingCall.response` 属性。



## 示例

以下范例处理从 Web 服务方法返回的结果。

```
// 处理在使用 Web 服务方法时返回的结果。
myPendingCallObj = myWebService.methodName(params)
myPendingCallObj.onResult = function(result)
{
 // 捕获结果并为此应用程序对其进行处理。
 ResultOutputField.text = result;
}
```

## 另请参见

[PendingCall.response](#)

# PendingCall.request

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
rawXML = myPendingCallback.request;
```

## 说明

属性：包含利用 `myPendingCallback = myWebService.methodName()` 发送的当前请求的原始 XML 格式。通常，您不会用到 `PendingCall.request`，但如果对通过网络发送的 SOAP 通信感兴趣，则可以使用该属性。若要获取请求结果的 **ActionScript** 版本，请使用 `myPendingCallback.onResult`。

# PendingCall.response

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
rawXML = myPendingCallback.response;
```

说明

属性：包含对利用 `myPendingCallback = myWebService.methodName()` 发送的最新 Web 服务方法调用的响应的原始 XML 格式。通常，您不会用到 `PendingCall.response`，但如果对通过网络发送的 SOAP 通信感兴趣，则可以使用该属性。若要获取请求结果的相应 `ActionScript` 版本，请使用 `myPendingCallback.onResult`。

# SOAPCall 类（仅限 Flash Professional）

**ActionScript 类名称** `mx.services.SOAPCall`

`SOAPCall` 类是 `mx.services` 包的一部分，是一种与 `WebService` 类一起使用的高级类（请参见第 1320 页的“[WebService 类（仅限 Flash Professional）](#)”）。有关 `mx.data.services` 包中类的概述，请参见第 1299 页的“[Web 服务类（仅限 Flash Professional）](#)”。

`SOAPCall` 对象不是由您构建的。相反，在对 `WebService` 对象调用方法时，`WebService` 对象会返回一个 `PendingCall` 对象。若要访问关联的 `SOAPCall` 对象，请使用 `myPendingCall.myCall`。

创建新的 `WebService` 对象时，它包含与传入的 WSDL URL 中的操作对应的方法。另外，也会在后台为 WSDL 中的每个操作创建一个 `SOAPCall` 对象。`SOAPCall` 对象是该操作的描述符，因此包含与该操作有关的所有信息（该 XML 在网络上的外观和操作样式等）。它还提供对某些行为的控制。使用 `WebService.getCall()` 函数可以获取给定操作的 `SOAPCall` 对象。对于每一操作只有一个 `SOAPCall`，由对该操作的所有活动调用共享。一旦获取 `SOAPCall` 对象，即可通过执行以下操作自定义描述符：

- 打开 / 关闭 XML 响应的解码
- 打开 / 关闭将 SOAP 数组转换为 `ActionScript` 对象的延迟
- 更改给定操作的并发性配置

## SOAPCall 对象的属性摘要

下表列出了 `SOAPCall` 对象的属性。

属性	说明
<code>SOAPCall.concurrency</code>	并发请求的数量。
<code>SOAPCall.doDecoding</code>	打开或关闭 XML 响应的解码。
<code>SOAPCall.doLazyDecoding</code>	打开或关闭“迟滞解码”（将 SOAP 数组转换为 <code>ActionScript</code> 对象的延迟）。

# SOAPCall.concurrency

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*SOAPCall.concurrency*

## 说明

属性；并发请求的数量。下表列出了可能的值：

值	说明
<code>SOAPCall.Multiple_Concurrency</code>	允许多个活动的调用。
<code>SOAPCall.Single_Concurrency</code>	一次只允许一个调用，在已有活动调用时激活另一个调用将导致错误。
<code>SOAPCall.Last_Concurrency</code>	只允许一个调用，激活一个调用将取消以前的活动调用。

# SOAPCall.doDecoding

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*SOAPCall.doDecoding*

## 说明

属性；打开 (true) 或关闭 (false) XML 响应的解码。默认情况下，XML 响应被转换（解码）为 **ActionScript** 对象。如果只需要 XML，请将 `SOAPCall.doDecoding` 设置为 false。

# SOAPCall.doLazyDecoding

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`SOAPCall.doLazyDecoding`

## 说明

属性；打开 (true) 或关闭 (false) 数组的“迟滞解码”。默认情况下将使用“迟滞解码”算法，用以将 SOAP 数组到 `ActionScript` 对象的转换延迟到最后一刻进行，从而大大加快函数在返回大型数据集时的执行速度。这意味着您从远端接收的所有数组都是 `ArrayProxy` 对象。然后，当您访问特定的索引 (`foo[5]`) 时，该元素在必要时会自动解码。通过将 `SOAPCall.doLazyDecoding` 设置为 `false` 可以关闭此行为（这会导致所有数组全部解码）。

# WebService 类（仅限 Flash Professional）

ActionScript 类名称 `mx.services.WebService`

`WebService` 类是 `mx.services` 包的一部分，与 `Log`、`PendingCall` 和 `SOAPCall` 类一起使用。有关 `mx.services` 包中类的概述，请参见第 1299 页的“[Web 服务类（仅限 Flash Professional）](#)”。



此 `WebService` 类不同于 `WebServiceConnector` 类。`WebServiceConnector` 类对可视 `WebServiceConnector` 组件提供了一个 ActionScript 接口。

`WebService` 对象充当远程 Web 服务的本地引用。创建新的 `WebService` 对象时，会下载和分析定义 Web 服务的 WSDL 文件，并将其放在该对象中。然后，可以直接对该 `WebService` 对象调用 Web 服务的方法，并处理该 Web 服务的任何回调。当成功处理了 WSDL 且 `WebService` 对象准备就绪时，会调用 `WebService.onLoad` 回调。如果在加载 WSDL 时遇到问题，会调用 `WebService.onFault` 回调。

对 `WebService` 对象调用方法时，返回值是一个回调对象。从所有 `Web` 服务方法返回的回调的对象类型都是 `PendingCall`。这些对象通常不是由您构建的，而是作为调用的 `webServiceObject.webServiceMethodName()` 方法的结果自动构建的。这些对象并不是稍后进行的 `WebService` 调用的结果。而 `PendingCall` 对象表示正在进行的调用。当 `WebService` 操作完成执行时（通常是在调用方法后的几秒钟），会填写各种 `PendingCall` 数据字段，而且会调用您提供的 `PendingCall.onResult` 或 `PendingCall.onFault` 回调。有关 `PendingCall` 对象的更多信息，请参见第 1308 页的“[PendingCall 类（仅限 Flash Professional）](#)”。

`Flash Player` 会在分析 `WSDL` 之前将您进行的所有调用进行排队，并尝试在分析 `WSDL` 之后执行这些调用。这是因为，`WSDL` 包含正确编码和发送 `SOAP` 请求所需的信息。在分析 `WSDL` 后进行的函数调用无需排队；它们会立即执行。如果某个排队的调用与 `WSDL` 中定义的任何操作的名称都不匹配，`Flash Player` 会向您在初次进行该调用时被指定的回调对象返回一个错误。

随附在 `mx.services` 包中的 `WebServices` API 由 `WebService` 类、`Log` 类和 `PendingCall` 类组成。

为了能够在运行时使用 `Web` 服务类，`FLA` 文件的库中必须有 `WebServiceConnector` 组件。如果您仅使用 `ActionScript` 在运行时访问 `Web` 服务，必须手动将此组件添加到文档的库中。有关如何将此组件添加到文档中的信息，请参见《使用 `Flash`》中的第 16 章“数据集成（仅限 `Flash Professional`）”。

# WebService 对象的方法摘要

下表列出了 `WebService` 对象的方法。

方法	说明
<code>WebService.getCall()</code>	获取给定操作的 <code>SOAPCall</code> 。
<code>WebService.myMethodName</code> <code>()</code>	调用由 <code>WSDL</code> 定义的特定 <code>Web</code> 服务操作。

# WebService 对象的回调摘要

下表列出了 `WebService` 对象的回调。

回调	说明
<code>WebService.onFault</code>	在 <code>WSDL</code> 分析的过程中出错时调用。
<code>WebService.onLoad</code>	当 <code>Web</code> 服务已成功加载并分析了其 <code>WSDL</code> 文件时调用。

# 支持的类型（仅限 Flash Professional）

Web 服务类支持下表定义的 XML 架构类型（数据类型）的子集。

也支持复杂的数据类型和 SOAP 编码的数组类型，而这些类型可以由其它复杂的类型、数组或内置的 XML 架构类型组成：

- [第 1322 页](#)的“数字简单类型”
- [第 1323 页](#)的“日期和时间简单类型”
- [第 1323 页](#)的“名称和字符串简单类型”
- [第 1323 页](#)的“布尔型”
- [第 1323 页](#)的“对象类型”
- [第 1324 页](#)的“支持的 XML 架构对象元素”

## 数字简单类型

XML 架构类型	ActionScript 绑定
decimal	Number
integer	Number
negativeInteger	Number
nonNegativeInteger	Number
positiveInteger	Number
long	Number
int	Number
short	Number
byte	Number
unsignedLong	Number
unsignedShort	Number
unsignedInt	Number
unsignedByte	Number
float	Number
double	Number

## 日期和时间简单类型

XML 架构类型	ActionScript 绑定
date	Date 对象
datetime	Date 对象
duration	Date 对象
gDay	Date 对象
gMonth	Date 对象
gMonthDay	Date 对象
gYear	Date 对象
gYearMonth	Date 对象
time	Date 对象

## 名称和字符串简单类型

XML 架构类型	ActionScript 绑定
string	ActionScript 字符串
normalizedString	ActionScript 字符串
QName	mx.services.Qname 对象

## 布尔型

XML 架构类型	ActionScript 绑定
Boolean	布尔型

## 对象类型

XML 架构类型	ActionScript 绑定
Any	XML 对象
Complex Type	由任何支持的类型的属性组成的 ActionScript 对象
Array	由任何支持的对象或类型组成的 ActionScript 数组

## 支持的 XML 架构对象元素

下面的架构描述说明了支持的 XML 架构对象元素：

```
schema
 complexType
 complexContent
 restriction
 sequence | simpleContent
 restriction
 element
 complexType | simpleType
```

## WebService 安全性（仅限 Flash Professional）

**WebService** 类的方法和回调符合 **Flash Player** 安全模型。有关 **Flash Player** 安全模型的更多信息，请参见《学习 Flash 中的 ActionScript 2.0》中的“了解安全性”。

**用户身份验证和授权** 对于 **WebService** API，身份验证和授权规则与对于 **Flash** 的任何 XML 网络操作的规则相同。**SOAP** 本身并未指定任何身份验证和授权的方法。例如，当下层的 **HTTP** 传送在 **HTTP** 标头中返回 **HTTP BASIC** 响应时，浏览器会通过以下方式响应：为用户显示一个对话框，随后将用户的输入附加到后续消息中的 **HTTP** 标头。此机制存在于比 **SOAP** 低的级别上，并且是 **Flash HTTP** 身份验证设计的一部分。

**消息完整性** 消息级安全性的机制是：在网络包之上传送 **SOAP** 消息的概念层对 **SOAP** 消息本身进行加密。

**传送安全性** **Flash Player SOAP Web** 服务的下层网络传送始终是 **HTTP POST**。因此，可以应用在 **Flash HTTP** 传送层上的任何安全性方式（例如 **SSL**）通过 **Flash** 中的 **Web** 服务调用获得支持。**SSL/HTTPS** 为 **SOAP** 消息传递提供最常用的传送安全性方式，而在传送层与 **SSL** 结合使用 **HTTP BASIC** 身份验证是当今最常用的网站安全性方式。

## WebService 类的构造函数

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
myWebServiceObject = new WebService(wsdlURI [, logObject]);
```



## 参数

*wsdlURI* Web 服务 WSDL 文件的 URI。

*logObject* 为此 Web 服务指定 Log 对象名称的可选参数。如果使用此参数，必须首先创建 Log 对象。有关更多信息，请参见第 1300 页的“Log 类（仅限 Flash Professional）”。

## 返回

无。

## 说明

构造函数：创建一个 **WebService** 对象。调用 `new WebService()` 时，请提供一个 WSDL URL，而 Flash Player 会返回一个 **WebService** 对象。构造函数可以选择是否接受一个代理 URI 和一个 Log 对象。

如果需要，可以对 **WebService** 对象使用两个回调。在完成 WSDL 文件的分析且对象完成时，Flash Player 会调用 `webServiceObject.onLoad`。对于您只想在完全分析 WSDL 文件后才执行的代码，这是一个放置此代码的好地方。例如，您可能会选择将第一个 Web 服务方法调用放在此函数中。

如果在查找或分析 WSDL 文件时出错，Flash Player 会调用 `webServiceObject.onFault`。此函数中适合放置调试代码和那些告知用户“服务器不可用，请稍后重试”之类信息的代码。有关详细信息，请参见这些函数的各个条目。

**调用 Web 服务操作** 将 Web 服务操作作为可以直接在 Web 服务上使用的方法进行调用。例如，如果 Web 服务具有方法 `getCompanyInfo(tickerSymbol)`，则按以下方式调用该方法：

```
myPendingCallObject = myWebServiceObject.getCompanyInfo(tickerSymbol);
```

在本例中，回调对象被命名为 `myPendingCallObject`。所有的方法调用都是异步的，并返回 `PendingCall` 类型的回调对象。（异步表示不能立即获得 Web 服务调用的结果。）

考虑以下调用：

```
x = stockService.getQuote("macr");
```

进行此调用时，`x` 对象不是 `getQuote` 的结果，而是一个 `PendingCall` 对象。只能在稍后 Web 服务操作完成时获得实际结果。ActionScript 代码由对 `onResult` 回调函数的调用进行通知。

**处理 PendingCall 对象** 此回调对象是一个 `PendingCall` 对象，用于处理已调用的 Web 服务方法的结果和错误（请参见第 1308 页的“`PendingCall` 类（仅限 Flash Professional）”）。下面是一个示例：

```
MyPendingCallObject = myWebServiceObject.myMethodName(param1, ..., paramN);
MyPendingCallObject.onResult = function(result)
{
 OutputField.text = result
}
MyPendingCallObject.onFault = function(fault)
```

```
{
 DebugField.text = fault.faultCode + "," + fault.faultstring;

 // 添加代码以处理任何错误，例如，通过
 // 通知用户服务器不可用或者与技术支持部门
 // 联系。
}
```

## WebService.getCall()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
getCall(var operationName)
```

### 参数

*operationName* 您想检索的相应 SOAPCall 对象的 Web 服务操作。

### 返回

SOAPCall 对象。

### 说明

函数：创建新的 **WebService** 对象时，它包含与传入的 WSDL URL 中的操作对应的方法。另外，也会在后台为 WSDL 中的每个操作创建一个 **SOAPCall** 对象。**SOAPCall** 对象是该操作的描述符，因此包含与该操作有关的所有信息（该 XML 在网络上的外观和操作样式等）。它还提供对某些行为的控制。使用 `getCall()` 方法可以获取给定操作的 **SOAPCall** 对象。对于每一操作只有一个 **SOAPCall** 对象，由对该操作的所有活动调用共享。一旦获取了 **SOAPCall** 对象，即可使用 **SOAPCall** 类更改操作符的描述符；请参见 [第 1318 页的“SOAPCall 类（仅限 Flash Professional）”](#)。

# WebService.myMethodName()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
callbackObj = myWebServiceObject.myMethodName(param1, ... paramN);
```

## 参数

*param1*, ... *paramN* 各种参数，具体取决于调用的 Web 服务方法。

## 返回

一个 **PendingCall** 对象，您可以将用于处理调用结果和错误的函数附加到该对象。有关更多信息，请参见第 1308 页的“**PendingCall 类（仅限 Flash Professional）**”。

在从 **WebService** 方法返回的响应是 **PendingCall.onResult** 或 **PendingCall.onFault** 的情况下调用的回调。通过唯一地标识回调对象，可以管理多个 **onResult** 回调，如下例所示：

```
myWebService = new WebService("http://www.myCompany.com/myService.wsdl");
callback1 = myWebService.getWeather("02451");
callback1.onResult = function(result)
{
 // 完成一些操作
}
callback2 = myWebService.getDetailedWeather("02451");
callback2.onResult = function(result)
{
 // 完成另一些操作
}
```

## 说明

方法：调用 Web 服务操作。可对 Web 服务直接调用此方法。例如，如果 Web 服务具有方法 **getCompanyInfo(tickerSymbol)**，则按以下方式调用该方法：

```
myCallbackObject.myservice.getCompanyInfo(tickerSymbol);
```

所有调用都是异步的，并返回 **PendingCall** 类型的回调对象。

# WebService.onFault

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## Usage

```
MyWebServiceObject.onFault = function(fault)
{
 // 此处是您的代码。
}
```

## 参数

*fault* 具有属性的错误的已解码 **ActionScript** 对象版本。如果 XML 形式的错误信息来自某一服务器，则 **SOAPFault** 对象将是该 XML 的已解码 **ActionScript** 版本。

返回给 **WebService.onFault** 方法的错误对象类型是 **SOAPFault** 对象。该对象不是由您直接构建的，但作为失败的结果返回。该对象是 **SOAPFault** XML 类型的 **ActionScript** 映射。

SOAPFault 属性	说明
<i>faultcode</i>	字符串；描述错误的标准短 QName。
<i>faultstring</i>	字符串；可由人直接读取的关于错误的说明。
<i>detail</i>	字符串；与错误相关的特定于应用程序的信息，例如由 Web 服务引擎返回的堆栈跟踪或其它信息。
<i>element</i>	XML；表示错误的 XML 版本的 XML 对象。
<i>faultactor</i>	字符串；错误的来源。如果未涉及中间项，则为可选。

## 返回

无。

## 说明

回调函数；当 **new WebService()** 构造函数已失败且返回错误时，由 **Flash Player** 调用。这可能发生在无法分析或找到 **WSDL** 文件的时候。*fault* 参数是一个 **ActionScript SOAPFault** 对象。

## 示例

以下范例处理在 **WebService** 对象的创建中返回的任何错误。

```
MyWebServiceObject.onFault = function(fault)
{
 // 捕获错误。
 DebugOutputField.text = fault.faultstring;

 // 添加代码以处理任何错误，例如，通过
 // 通知用户服务器不可用或者与技术支持部门
 // 联系。
}
```

# WebService.onLoad

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
myService.onLoad = function(wsd1Document)
{
 // 此处是您的代码。
}
```

## 参数

*wsd1Document* WSDL XML 文档。

## 返回

无。

## 说明

回调函数：当 **WebService** 对象已成功加载并分析了其 WSDL 文件时，由 Flash Player 调用。如果在调用此回调函数之前应用程序调用了操作，则这些操作会在内部排队，直到 WSDL 加载后才会被实际传送。

## 示例

以下范例指定 WSDL URL，创建新的 Web 服务对象，并在加载后接收 WSDL 文档。

```
// 指定 WSDL URL。
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";

// 创建新的 Web 服务对象。
stockService = new WebService(wsdlURI);

// 加载后接收 WSDL 文档。
stockService.onLoad = function(wsdlDocument);
{
 // 当 WSDL 加载完成且已创建了对象
 // 时执行的代码。
}
```

# WebServiceConnector 组件 (仅限 Flash Professional)

**WebServiceConnector** 组件使您可以使用业界标准的 **SOAP**（简单对象访问协议）来访问服务器公开的远程方法。**Web** 服务方法可以接受参数和返回结果。通过使用 **Flash Professional 8** 创作工具和 **WebServiceConnector** 组件，您可以检查、访问和绑定远程 **Web** 服务和 **Flash** 应用程序之间的数据。

**WebServiceConnector** 组件的单个实例可用于对同一操作进行多个调用。您需要为想调用的每个不同操作使用不同的 **WebServiceConnector** 实例。



**WebServiceConnector** 组件在应用程序创作的过程中显示在舞台上，但在运行时应用程序中不可见。

有关处理此组件的结果的介绍性信息，请参见《使用 **Flash**》中的“在“架构”选项卡中处理架构（仅限 **Flash Professional**）”。

## 使用 WebServiceConnector 组件 (仅限 Flash Professional)

可以使用 **WebServiceConnector** 组件连接到 **Web** 服务，并使 **Web** 服务的属性可用于绑定到应用程序中的 **UI** 组件的属性。要连接到 **Web** 服务，必须首先输入表示该 **Web** 服务的 **WSDL** 文件的 **URL**。可以在“组件”检查器或“**Web** 服务”面板中输入此 **URL**。请参见《使用 **Flash**》中的“用 **WebService** 连接器组件连接 **Web** 服务（仅限 **Flash Professional**）”。

有关连接到 **Web** 服务的更多信息，请参见《使用 **Flash**》中的“数据绑定（仅限 **Flash Professional**）”。

## WebServiceConnector 参数

您可以使用“组件”检查器的“参数”选项卡为每个 **WebServiceConnector** 组件实例设置以下创作参数：

**multipleSimultaneousAllowed** 是一个布尔值，它指示是否可以同时执行多个调用；默认值为 **false**。如果此参数为 **false**，则当正在执行一个调用时，**trigger()** 方法不执行调用。并使用代码 **CallAlreadyInProgress** 发出 **status** 事件。如果此参数为 **true**，则执行调用。

**operation** 是一个字符串，指示 WSDL 文件中的 SOAP 端口内显示的操作名称。

**suppressInvalidCalls** 是一个布尔值，指示在参数无效时是否禁止调用；默认值为 **false**。如果此参数为 **true**，则 **trigger()** 方法在数据绑定参数未通过验证的情况下将不执行调用。并使用代码 **InvalidParams** 发出 **status** 事件。如果此参数为 **false**，则使用需要的无效数据执行该调用。

**WSDLURL**（字符串类型）是定义 Web 服务操作的 WSDL 文件的 URL。在创作期间设置此 URL 时，立即获取和分析该 WSDL 文件。可以在“组件”检查器的“架构”选项卡上看到产生的参数和结果信息。还会将服务描述添加到“Web 服务”面板。若要查看示例，请参见 [www.flash-mx.com/mm/tips/tips.cfc?wsdl](http://www.flash-mx.com/mm/tips/tips.cfc?wsdl)。

## WebServiceConnector 组件的通用工作流程

以下过程介绍了 **WebServiceConnector** 组件的典型工作流程。

**要使用 WebServiceConnector 组件：**

1. 使用“Web 服务”面板输入 Web 服务 WSDL 文件的 URL。
2. 通过选择方法，右击 (Windows) 或按住 Ctrl 键单击 (Macintosh)，然后从上下文菜单中选择“添加方法调用”，从而添加对 Web 服务的方法的调用。

这将在应用程序中创建 **WebServiceConnector** 组件实例。可在“组件”检查器的“架构”选项卡上找到组件的架构。可以根据需要随意编辑此架构，例如，提供附加格式或验证设置。



每次更改 WSDLURL 或 operation 参数时，params 和 results 组件属性的架构都会随之更新。这将覆盖您所编辑的任何设置。

3. 使用“组件”检查器中的“绑定”选项卡，将架构中现在定义的 Web 服务参数和结果绑定到应用程序中的组件。



4. 使用以下方法之一添加触发器以启动数据绑定操作：

- 将“触发器数据源”行为附加到某个按钮上。
- 添加自己的 `ActionScript` 来调用 `WebServiceConnector` 组件上的 `trigger()` 方法。
- 创建 web 服务参数和 UI 控件之间的绑定，并将其 `Kind` 属性设置为 `AutoTrigger`。  
有关更多信息，请参见《使用 Flash》中的“架构种类”。

若要查看使用 `WebServiceConnector` 组件连接和显示 Web 服务的分步示例，请参见“Web 服务教程：Macromedia 提示”。

# WebServiceConnector 类 (仅限 Flash Professional)

继承 `RPC > WebServiceConnector`

`ActionScript` 类名称 `mx.data.components.webServiceConnector`

该类允许使用 `ActionScript` 代码（而不是舞台上的组件实例）连接到远程 Web 服务。要使用 `WebServiceConnector` 类，需要将 `WebServiceConnector` 组件的一个实例添加到库中。组件不必立即放置在舞台上。在脚本的开始部分必须导入 `ActionScript` 类 `mx.data.components.WebServiceConnector`，也可以在整个代码中使用完全限定类名。

## WebServiceConnector 类的方法摘要

下表列出了 `WebServiceConnector` 类的方法。

方法	说明
<code>WebServiceConnector.trigger()</code>	启动对 Web 服务的调用。

## WebServiceConnector 类的属性摘要

下表列出了 WebServiceConnector 类的属性。

属性	说明
<a href="#">WebServiceConnector.multipleSimultaneousAllowed</a>	指明是否可以同时进行多个调用。
<a href="#">WebServiceConnector.operation</a>	指示显示在 WSDL 文件中的 SOAP 端口内的操作名称。
<a href="#">WebServiceConnector.params</a>	指定在执行下一个 trigger() 操作时要发送到服务器的数据。
<a href="#">WebServiceConnector.results</a>	标识作为 trigger() 操作的结果从服务器接收的数据。
<a href="#">WebServiceConnector.suppressInvalidCalls</a>	指明在参数无效的情况下是否禁止调用。
<a href="#">WebServiceConnector.WSDLURL</a>	指定定义 Web 服务操作的 WSDL 文件的 URL。

## WebServiceConnector 类的事件摘要

下表列出了 WebServiceConnector 类的事件。

事件	说明
<a href="#">WebServiceConnector.result</a>	在对 Web 服务进行的调用成功完成时广播。
<a href="#">WebServiceConnector.send</a>	当 trigger() 方法正在执行时（在收集了参数数据之后，但在验证这些数据 and 启动对 Web 服务的调用之前）进行广播。
<a href="#">WebServiceConnector.status</a>	在启动对 Web 服务的调用时广播，以通知用户操作状态。

# WebServiceConnector.multipleSimultaneousAllowed

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.multipleSimultaneousAllowed*

## 说明

属性：指示能 (true) 否 (false) 同时进行多个调用。如果此属性为 true，则执行调用。如果此属性为 false 且另一个调用正在进行中，则 `WebServiceConnector.trigger()` 方法会使用代码 `CallAlreadyInProgress` 发出 status 事件。

当同时进行多个调用时，不保证这些调用的完成顺序与触发它们的顺序相同。此外，浏览器和 / 或操作系统可以对同时执行的网络操作的数目加以限制。最可能遇到的限制是，浏览器实施可同时下载的 URL 的最大数目。此限制通常可在浏览器配置。但是，即使遇到此限制，浏览器也会对“流”进行排队，不会影响 Flash 应用程序的预期行为。

## 示例

以下示例允许同时对 myXmlUrl 进行多个调用：

```
myXmlUrl.multipleSimultaneousAllowed = true;
```

# WebServiceConnector.operation

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.operation;*

## 说明

属性：显示在 WSDL 文件中的 SOAP 端口内的操作名称。

## 示例

此示例返回来自远程 **Web** 服务的数据，并使用 **trace** 命令发送一个提示以及数据返回到 **SWF** 文件服务所用的时间。将一个 **WebServiceConnector** 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.WebServiceConnector;

var startTime:Number;
var wscListener:Object = new Object();
wscListener.result = function(evt:Object) {
 var resultTimeMS:Number = getTimer()-startTime;
 trace("result loaded in "+resultTimeMS+" ms.");
 trace(evt.target.results);
};
wscListener.send = function(evt:Object) {
 startTime = getTimer();
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", wscListener);
wsConn.addEventListener("send", wscListener);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.multipleSimultaneousAllowed = false;
wsConn.trigger();
```

# WebServiceConnector.params

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.params*

## 说明

属性：指定在执行下一个 `trigger()` 操作时要发送到服务器的数据。数据类型由 **Web** 服务的 WSDL 描述确定。

调用 **Web** 服务方法时，`params` 属性的数据类型必须是一个 **ActionScript** 对象或数组，如下所示：

- 如果 **Web** 服务为文档格式，则 `params` 的数据类型是 XML 文档。
- 如果在创作时使用“属性”检查器或“组件”检查器来设置 WSDL URL 和操作，则可以按照 **Web** 服务方法所要求的同一顺序提供作为参数数组的 `params`，如 `[1, "hello", 2432]`。

## 示例

以下示例为名为 `wsc` 的 **Web** 服务组件设置 `params` 属性：

```
wsc.params = [param_txt.text];
```

# WebServiceConnector.result

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
componentInstance.addEventListener("result", myListenerObject)
```

## 说明

事件：在对 **Web** 服务进行的调用成功完成时广播。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 `"result"`
- `target`：对发出该事件的对象的引用（如 **WebServiceConnector** 组件）

您可以使用 `results` 属性获取实际结果值。

## 示例

以下示例为 `result` 事件定义 `res` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var res = function (ev) {
 trace(ev.target.results);
};
wsc.addEventListener("result", res);
```

此示例返回来自远程 **Web** 服务的数据，并使用 `trace` 命令发送一个提示。将一个 **WebServiceConnector** 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.WebServiceConnector;
var wscListener:Object = new Object();
wscListener.result = function(evt:Object) {
 trace(evt.target.results);
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", wscListener);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.trigger();
```

# WebServiceConnector.results

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.results*

## 说明

属性：标识作为 `trigger()` 操作的结果从服务器接收的数据。每个 **WebServiceConnector** 组件均定义获取此数据的方式以及有效的类型。这些数据在 **RPC** 操作成功完成后出现，并由 `result` 事件通知。它在该组件卸载前或执行下一个 **RPC** 操作前可用。

返回的数据可能很大。您可以通过以下两种方式管理该大小：

- 选择适当的影片剪辑、时间轴或屏幕作为 **WebServiceConnector** 组件的父级。在父级销毁后，就可从该组件的存储内存中将其当作垃圾回收。
- 在 **ActionScript** 中，可以随时将 `null` 分配给该属性。

# WebServiceConnector.send

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
componentInstance.addEventListener("send", myListenerObject)
```

## 说明

事件：在处理 `trigger()` 操作期间（在收集了参数数据后，但在验证这些数据 and 启动对 Web 服务的调用前）广播。此位置适合于存放将在调用前修改参数数据的代码。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 "send"
- `target`：对发出该事件的对象的引用（如 `WebServiceConnector` 组件）

您可以使用 `params` 属性获取或者修改实际参数值。

## 示例

以下示例为 `send` 事件定义 `sendFunction` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var sendFunction = function (sendEnv) {
 sendEnv.target.params = [newParam_txt.text];
};
wsc.addEventListener("send", sendFunction);
```

# WebServiceConnector.status

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
componentInstance.addEventListener("status", myListenerObject)
```

说明

事件：在启动对 Web 服务的调用时广播，以通知用户操作状态。

事件处理函数的参数是具有以下字段的对象：

- type: 字符串 "status"
- target: 对发出该事件的对象的引用（如 **WebServiceConnector** 组件）
- code: 一个字符串，提供已发生的情况的名称
- data: 一个对象，其内容取决于代码

下面是可用于 status 事件的代码和关联数据：

代码	数据	说明
StatusChange	{callsInProgress:nnn}	只要 Web 服务调用开始或结束就发出该事件。 nnn 项提供当前正发生的调用的数目。
CallAlreadyInProgress	无数据	如果调用了 trigger(), multipleSimultaneousAllowed 为 false, 并且某个调用已在执行, 则发出此事件。在发生此事件后, 则认为尝试的调用已完成, 并且不会发生 result 或 send 事件。
InvalidParams	无数据	如果 trigger() 方法发现 params 属性不包含有效数据, 则发出此事件。如果 suppressInvalidCalls 属性为 true, 则认为尝试的调用已完成, 并且不会发生 result 或 send 事件。
WebServiceFault	{faultcode:code, faultstring:string, detail:detail}	如果在处理调用的过程中发生其它问题, 则会发出此事件。数据是 SOAPFault 对象。在发生此事件后, 则认为尝试的调用已完成, 并且不会发生 "result" 或 "send" 事件。参见下表以了解可能发生的错误的列表。



以下是可能的 Web 服务值：

faultcode	faultstring	detail
Timeout	调用方法 xxx 时的超时	
MustUnderstand	对于标题 xxx 无回调	
Server.Connection	无法连接到端点：xxx	
VersionMismatch	请求实现版本：xxx 响应实现版本 yyy	
Client.Disconnected	无法加载 WSDL	无法加载 WSDL，如果当前在线，请验证 WSDL xxx 的 URI 和 / 或格式。
Server	出错的 WSDL 格式	定义必须是 WSDL 文档中的第一个元素
Server.NoServicesInWSDL	无法加载 WSDL	在 xxx 处找不到 WSDL 中的元素
WSDL.UnrecognizedNamespace	WSDL 分析器没有用于命名空间 xxxx 的注册文档	
WSDL.UnrecognizedBindingName	WSDL 分析器在命名空间 yyy 中无法找到命名为 xxx 的绑定	
WSDL.UnrecognizedPortTypeName	WSDL 分析器在命名空间 yyy 中无法找到命名为 xxx 的 portType	
WSDL.UnrecognizedMessageName	WSDL 分析器在命名空间 yyy 中无法找到命名为 xxx 的消息	
WSDL.BadElement	元素 xxx 不可解析	
WSDL.BadType	类型 xxx 不可解析	
Client.NoSuchMethod	无法在服务中找到 “xxx” 方法	
yyy	yyy - 从服务器报告的错误，这取决于您与哪一台服务器通信	
No.WSDLURL.Defined	WebServiceConnector 组件没有已定义的 WSDL URL	
Unknown.Call.Failure	WebService 调用因未知原因而失败	
Client.Disconnected	无法加载导入的架构	无法加载架构；如果当前在线，请在 (XXXX) 处验证架构的 URI 和 / 或格式

## 示例

以下示例为 `status` 事件定义 `fault` 函数，并且将该函数分配给 `addEventListener` 事件处理函数。本示例故意拼错服务的 **URI**，以便返回有关 **Web** 服务错误（URL 应该为 "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl"）和让用户验证 **URI** 的消息。在库中有 **WebServiceConnector** 组件的情况下，将以下内容添加到时间轴的第一帧中：

```
import mx.data.components.WebServiceConnector;

var fault = function (stat) {
 if (stat.code == "WebServiceFault"){
 trace(stat.data.faultcode);
 trace(stat.data.faultstring);
 trace(stat.data.detail);
 }
};

var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("status", fault);
wsConn.WSDLURL = "http://www.flasht-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.trigger();
```

# WebServiceConnector.suppressInvalidCalls

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance*.suppressInvalidCalls

## 说明

属性；指明在参数无效的情况下是否禁止调用。如果此属性为 `true`，则 `trigger()` 方法在绑定参数未通过验证的情况下将不执行调用。并使用代码 `InvalidParams` 发出 `status` 事件。如果此属性为 `false`，则使用需要的无效数据执行该调用。

## 示例

由于没有传递必需的参数，此示例会显示一个错误。将一个 `WebServiceConnector` 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.WebServiceConnector;
var res:Function = function (evt:Object) {
 trace(evt.target.results);
};
var stat:Function = function (error:Object) {
 switch (error.code) {
 case 'InvalidParams' :
 trace("Unable to connect to remote Web Service: "+error.code);
 break;
 case 'StatusChange' :
 break;
 default :
 trace("Error: "+error.code);
 break;
 }
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", res);
wsConn.addEventListener("status", stat);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
// wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.trigger();
```

若要显示提示而不是错误，请注释掉行 `wsConn.params = ["Flash"];`。

# WebServiceConnector.trigger()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance*.trigger();

## 说明

方法：启动对 **Web** 服务的调用。每个 **Web** 服务都准确地定义了此调用所涉及的内容。如果操作成功，则操作的结果将显示在 **Web** 服务的 `results` 属性中。

`trigger()` 方法执行以下步骤：

1. 如果任何数据绑定到 `params` 属性，则该方法执行所有绑定以确保提供最新数据。这样做还会导致发生数据验证。
2. 如果数据无效且 `suppressInvalidCalls` 设置为 `true`，则操作将停止。
3. 如果操作继续，则发出 `send` 事件。
4. 使用指示的连接方法（例如 **HTTP**）启动实际远程调用。

## 示例

此示例返回来自远程 **Web** 服务的数据，并使用 `trace` 命令发送一个提示。将一个 **WebServiceConnector** 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.WebServiceConnector;
var res:Function = function (evt:Object) {
 trace(evt.target.results);
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", res);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.trigger();
```

# WebServiceConnector.WSDLURL

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

`componentInstance.WSDLURL`

## 说明

属性：定义 Web 服务操作的 WSDL 文件的 URL。在创作时设置此 URL 时，立即获取和分析该 WSDL 文件。产生的参数和结果显示在“组件”检查器的“架构”选项卡中。此外，服务描述显示在“Web 服务”面板中。

## 示例

此示例返回来自远程 Web 服务的数据，并使用 `trace` 命令发送一个提示。将一个 `WebServiceConnector` 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.WebServiceConnector;
var res:Function = function (evt:Object) {
 trace(evt.target.results);
};
var wsConn:WebServiceConnector = new WebServiceConnector();
wsConn.addEventListener("result", res);
wsConn.WSDLURL = "http://www.flash-mx.com/mm/tips/tips.cfc?wsdl";
wsConn.operation = "getTipByProduct";
wsConn.params = ["Flash"];
wsConn.suppressInvalidCalls = true;
wsConn.trigger();
```



# Window 组件

**Window** 组件在一个具有标题栏、边框和关闭按钮（可选）的窗口内显示影片剪辑的内容。

**Window** 组件可以是模式的，也可以是非模式的。模式窗口会防止鼠标和键盘输入转至该窗口之外的其它组件。**Window** 组件还支持拖动操作；用户可以单击标题栏并将窗口及其内容拖动到另一个位置。拖动边框不会更改窗口的大小。

每个 **Window** 实例的实时预览反映在创作过程中对“属性”检查器或“组件”检查器中的所有参数所做的更改，但 `contentPath` 除外。

在将 **Window** 组件添加到应用程序时，您可以使用“辅助功能”面板，以使其可由屏幕阅读器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.WindowAccImpl.enableAccessibility();
```

不管组件有多少实例，都只对组件启用一次辅助功能。有关更多信息，请参见《使用 Flash》中的第 19 章“创建辅助内容”。

## 使用 Window 组件

无论何时您需要向用户提供信息或最优先的选择时，您都可以在应用程序中使用一个窗口。例如，您可能会需要用户填写登录窗口或者发生了更改并需要确认新密码的窗口。

将窗口添加到应用程序有几种方式。您可以将窗口从“组件”面板拖动到舞台。您也可以调用 `createClassObject()`（参见 `UIObject.createClassObject()`）将窗口添加到应用程序。将窗口添加到应用程序的第三种方法是使用 **PopUpManager** 类。使用弹出管理器可以创建与舞台上其它对象重叠的模式窗口。有关更多信息，请参见第 1352 页的“**Window** 类”。

如果使用弹出管理器向文档添加 **Window** 组件，则该 **Window** 实例将具有自己的焦点管理器，这与文档的其它组件不同。如果您不使用弹出管理器，则窗口的内容会与文档中的其它组件一起参加焦点排序。有关控制焦点的更多信息，请参见《使用组件》中的第 663 页的“**FocusManager** 类”或“创建自定义焦点导航”。

如 **Loader**、**ScrollPane** 和 **Window** 的组件具有用于确定内容何时完成加载的事件。若要设置 **Loader**、**ScrollPane** 或 **Window** 组件内容上的属性，请在 “complete” 事件处理函数中添加属性语句，如下例所示：


```
loadtest = new Object();
loadtest.complete = function(eventObject){
 content_mc._width= 100;
}
my_window.addEventListener("complete", loadtest)
```

有关更多信息，请参见第 1359 页的 “[Window.complete](#)”。

## Window 参数


您可以在 “属性” 检查器或 “组件” 检查器（“窗口” > “组件检查器” 菜单选项）中为每个 **Window** 组件实例设置以下创作参数：

**closeButton** 指示是 (true) 否 (false) 显示关闭按钮。单击关闭按钮会广播一个 click 事件，但不关闭窗口。您必须编写调用 [Window.deletePopUp\(\)](#) 的处理函数，以显式关闭窗口。有关 click 事件的更多信息，请参见 [Window.click](#)。

 如果窗口是通过弹出管理器以外的方式创建的，则无法关闭该窗口。

**contentPath** 指定窗口的内容。这可以是电影剪辑的链接标识符，或者是屏幕、表单或包含窗口内容的幻灯片的元件的名称。也可以是要加载到窗口的 **SWF** 或 **JPEG** 文件的绝对或相对 **URL**。默认值为 “”。加载的内容会被裁剪，以适合窗口大小。

**title** 指示窗口的标题。

 **minHeight** 和 **minWidth** 属性由内部的大小调整例程使用。这两个属性在 **UIObject** 中定义，并可以根据需要被不同的组件覆盖。在为应用程序创建自定义布局管理器时，可以使用这两个属性。否则，在 “组件” 检查器中设置这两个属性不具有可视效果。

您可以在 “组件” 检查器（“窗口” > “组件检查器”）中为每个 **Window** 组件实例设置以下附加参数：

**enabled** 是一个布尔值，它指示组件是否可以接收焦点和输入。默认值为 true。

**visible** 是一个布尔值，它指示对象是 (true) 否 (false) 可见。默认值为 true。

 有关以下五种外观参数的更多信息，请参见第 1351 页的 “[对 Window 组件使用外观](#)”。

**skinCloseDisabled** 确定关闭按钮是否处于其禁用状态。默认值为 **CloseButtonDisabled**。

**skinCloseDown** 确定关闭按钮是否处于其按下状态。默认值为 **CloseButtonDown**。

**skinCloseOver** 确定关闭按钮是否处于其指针经过状态。默认值为 **CloseButtonOver**。

**skinCloseUp** 确定关闭按钮是否处于其弹起（默认）状态。默认值为 **CloseButtonUp**。



`skinTitleBackground` 确定是否显示标题栏。默认值为 `TitleBackground`。

`titleStyleDeclaration` 指定标题文本的样式声明的名称。默认值为 `undefined`，它可使标题栏具有白色粗体文本。请参见《使用组件》中的“为成组的组件设置自定义样式”。

您可以编写 `ActionScript`，以便使用 `Window` 组件的属性、方法和事件来控制该组件的这些和其它选项。有关更多信息，请参见第 1352 页的“[Window 类](#)”。

## 创建具有 Window 组件的应用程序

以下过程解释了如何将 `Window` 组件添加到应用程序。在本示例中，当用户单击某个按钮时，窗口会显示一个图像。

### 创建具有 Window 组件的应用程序：

1. 将一个 `Window` 组件从“组件”面板拖到当前文档的库中。此操作会将组件添加到库中（而不是舞台上）。
2. 将一个按钮组件从“组件”面板中拖动到舞台上，并在“属性”检查器中为该组件指定实例名称 `my_button`。
3. 打开“动作”面板，然后在第一帧上输入下列 `click` 处理函数：

```
/**
 要求：
 - 舞台上有 Button 组件（实例名称: my_button）
 - 库中有 Window 组件
*/
import mx.containers.Window;

var my_button:mx.controls.Button;

System.security.allowDomain("http://www.helpexamples.com");

// 创建侦听器对象。
var buttonListener:Object = new Object();
buttonListener.click = function(evt_obj:Object) {
 // 实例化窗口。
 var my_win:MovieClip =
 mx.managers.PopUpManager.createPopUp(evt_obj.target, Window, true,
 {title:"Sample Image", contentPath:"http://www.helpexamples.com/flash/
 images/image1.jpg"});
 my_win.setSize(320, 240);
};
// 添加侦听器。
my_button.addEventListener("click", buttonListener);
```

用户单击 `my_button` 按钮时，`buttonListener` 事件侦听器调用本示例创建的 `click()` 函数。`click` 事件处理函数 `buttonListener.click()` 调用 `PopUpManager.createPopUp()` 以对显示图像的窗口进行实例化。若要在单击“确定”或“取消”按钮时关闭窗口，您将需要编写另一个处理函数。

# 自定义 Window 组件

在创作过程中和运行时，可以在水平方向和垂直方向上将 **Window** 组件变形。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `UIObject.setSize()`。

调整窗口大小不会更改关闭按钮或标题题注的大小。标题题注为左对齐，关闭栏为右对齐。

## 对 Window 组件使用样式

**Window** 组件支持以下样式：

样式	主题	说明
themeColor	光晕	组件的基本配色方案。可能的值包括 "haloGreen"、"haloBlue" 和 "haloOrange"。默认值为 "haloGreen"。
backgroundColor	光晕和范例	背景色。“光晕”主题的默认值为白色，“范例”主题的默认值为 OxEFEBEF（浅灰）。
borderStyle	光晕和范例	<b>Window</b> 组件使用 <b>RectBorder</b> 实例作为其边框，并响应在该类中定义的样式。请参见第 985 页的“ <b>RectBorder</b> 类”。 <b>Window</b> 组件具有特定于组件的边框样式，“光晕”主题的边框样式为“default”，“范例”主题的边框样式为“outset”。
color	光晕和范例	文本颜色。“光晕”主题的默认值为 0x0B333C，“范例”主题的默认值为空白。
disabledColor	光晕和范例	组件禁用时的文本颜色。默认值为 0x848384（深灰）。
embedFonts	光晕和范例	一个布尔值，它指示在 fontFamily 中指定的字体是否为嵌入字体。如果 fontFamily 引用了嵌入字体，则此样式必须设置为 true。否则，将不使用该嵌入字体。如果此样式设置为 true，并且 fontFamily 不引用嵌入字体，则不会显示任何文本。默认值为 false。
fontFamily	光晕和范例	文本的字体名称。默认值为 "_sans"。
fontSize	光晕和范例	字体的磅值。默认值为 10。
fontStyle	光晕和范例	字体样式："normal" 或 "italic"。默认值为 "normal"。
fontWeight	光晕和范例	字体粗细："none" 或 "bold"。默认值为 "none"。在调用 setStyle() 期间，所有组件还可以接受值 "normal" 来代替 "none"，但随后对 getStyle() 的调用将返回 "none"。
textAlign	光晕和范例	文本对齐方式："left"、"right" 或 "center"。默认值为 "left"。
textDecoration	光晕和范例	文本修饰："none" 或 "underline"。默认值为 "none"。
textIndent	光晕和范例	表示文本缩进的数字。默认值为 0。

文本样式可以在 **Window** 组件本身上设置，也可以在 `_global.styles.windowStyles` 类样式声明上设置（仅限文本样式，而不是来自 `_global.styles.Window` 类样式声明的、类似于 **themeColor** 或 **backgroundColor** 的其它样式）。这具有不会导致样式设置通过样式继承传播到子组件的优点。

下面的示例演示如何在不使此设置传播到子组件的情况下将 **Window** 组件的标题设置为斜体。

```
import mx.containers.Window;
_global.styles.windowStyles.setStyle("fontStyle", "italic");
createClassObject(Window, "window", 1, {title: "A Window"});
```

注意，此示例在通过 `createClassObject()` 实例化窗口前设置属性。要使样式生效，必须在创建窗口前设置样式。

## 对 Window 组件使用外观

**Window** 组件对其标题背景和关闭按钮使用外观，并将一个 **RectBorder** 实例用作边框。**Window** 外观位于每个主题文件的 **Flash UI Components 2/Themes/ MMDefault/ Window Assets** 文件夹中。有关更多信息，请参见《使用组件》中的“关于设置组件外观”。有关 **RectBorder** 类和使用该类自定义边框的更多信息，请参见第 985 页的“**RectBorder** 类”。

标题背景外观始终会显示。背景的高度由外观图形确定。外观的宽度由 **Window** 组件根据 **Window** 实例的大小设置。当 `closeButton` 属性设置为 `true` 以及用户交互产生更改状态时，会显示关闭外观。

**Window** 组件使用下列外观属性：

属性	说明
<code>skinTitleBackground</code>	标题栏。默认值为 <code>TitleBackground</code> 。
<code>skinCloseUp</code>	关闭按钮。默认值为 <code>CloseButtonUp</code> 。
<code>skinCloseDown</code>	处于按下状态的关闭按钮。默认值为 <code>CloseButtonDown</code> 。
<code>skinCloseDisabled</code>	处于禁用状态的关闭按钮。默认值为 <code>CloseButtonDisabled</code> 。
<code>skinCloseOver</code>	处于悬停状态的关闭按钮。默认值为 <code>CloseButtonOver</code> 。

以下示例演示如何创建新的影片剪辑元件以用作标题背景。

将 Window 组件的标题设置为自定义影片剪辑元件：

1. 创建一个新的 FLA 文件。
2. 通过选择 “插入” > “新建元件” 创建一个新元件。
3. 将名称设置为 TitleBackground。
4. 如果高级视图未显示出来，则单击 “高级” 按钮。
5. 选择 “为 ActionScript 导出”。
6. 标识符将自动填写为 TitleBackground。
7. 将 AS 2.0 类设置为 mx.skins.SkinElement。

SkinElement 是一个简单类，可用于所有自身不提供 ActionScript 实现方法的外观元素。它提供第 2 版 Macromedia Component Architecture 组件框架所需的移动和调整大小功能。

8. 确保 “在第一帧导出” 处于选中状态，然后单击 “确定”。
9. 打开新元件以进行编辑。
10. 使用绘画工具创建一个黑色线条和红色填充的方框。
11. 将边框样式设置为 hairline。
12. 设置此方框（包括边框），使其位于 (0,0) 并且宽度为 100，高度为 22。

Window 组件将根据需要设置适当的外观宽度，但会使用现有的高度作为标题的高度。

13. 单击 “返回” 按钮返回主时间轴。
14. 将 Window 组件拖动到舞台上。
15. 选择 “控制” > “测试影片”。

## Window 类

继承 MovieClip > UIObject 类 > UIComponent 类 > View > ScrollView > Window

ActionScript 类名称 mx.containers.Window

使用 Window 类的属性可以在运行时执行以下操作：设置标题题注、添加关闭按钮以及设置显示内容。使用 ActionScript 设置 Window 类的属性会覆盖在 “属性” 检查器或 “组件” 检查器面板中设置的同名参数。

实例化窗口的最佳方法是调用 `PopUpManager.createPopUp()`。此方法既可创建模式窗口（重叠并禁用应用程序中的现有对象），也可创建非模式窗口。例如，下面的代码创建模式 Window 实例（最后一个参数指明是模式窗口）：

```
var newWindow = PopUpManager.createPopUp(this, Window, true);
```

Flash 通过在 **Window** 组件下方创建一个大的透明窗口来模拟形态。由于透明窗口的呈现方式，您可能会注意到透明窗口下的对象略显暗淡。有效透明度可以通过下面的方法进行设置：更改 `_global.style.modalTransparency` 值，范围为从 **0**（完全透明）到 **100**（不透明）。如果使窗口部分透明，还可以设置窗口的颜色，方法是：在默认主题中更改“模式”外观。


如果使用 `PopUpManager.createPopUp()` 创建模式窗口，则在删除时必须调用 `Window.deletePopUp()`，以便也可以删除透明窗口。例如，如果在窗口上使用关闭按钮，则要编写下列代码：

```
var win = PopUpManager.createPopUp(_root, Window, true, {closeButton:true});
function click(evt){
 evt.target.deletePopUp();
}
win.addEventListener("click", this);
```

创建模式窗口时代码不会停止执行。在其它环境（例如 **Microsoft Windows**）中，如果创建一个模式窗口，则创建窗口之后的代码行在窗口关闭之前不会运行。在 **Flash** 中，这些代码行在创建窗口之后、关闭窗口之前运行。

每个组件类都有一个 `version` 属性，而该属性是一个类属性。类属性只能用于该类本身。`version` 属性会返回一个字符串，该字符串指示组件的版本。若要访问此属性，请使用以下代码：

```
trace(mx.containers.Window.version);
```



代码 `trace(myWindowInstance.version);` 返回 `undefined`。

## Window 类的方法摘要

下表列出了 **Window** 类的方法。

方法	说明
<code>Window.deletePopUp()</code>	删除由 <code>PopUpManager.createPopUp()</code> 创建的 <b>Window</b> 实例。

## 从 UIObject 类继承的方法

下表列出了 **Window** 类从 **UIObject** 类继承的方法。从 **Window** 对象调用这些方法时，请使用 *WindowInstance.methodName* 的形式。

方法	说明
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.doLater()</code>	在“属性”检查器和“组件”检查器中设置了参数之后，调用一个函数。
<code>UIObject.getStyle()</code>	从样式声明或对象获取样式属性。
<code>UIObject.invalidate()</code>	标记对象使其在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。
<code>UIObject.setStyle()</code>	设置样式声明或对象的样式属性。

## 从 UIComponent 类继承的方法

下表列出了 **Window** 类从 **UIComponent** 类继承的方法。从 **Window** 对象调用这些方法时，请使用 *WindowInstance.methodName* 的形式。

方法	说明
<code>UIComponent.getFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

# Window 类的属性摘要

下表列出了 Window 类的属性。

属性	说明
<code>Window.closeButton</code>	指示标题栏上是 (true) 否 (false) 包含关闭按钮。
<code>Window.content</code>	对窗口的内容（根影片剪辑）的引用（只读）。
<code>Window.contentPath</code>	设置要在窗口中显示的内容的名称。
<code>Window.title</code>	标题栏中显示的文本。
<code>Window.titleStyleDeclaration</code>	设置标题栏中文本格式的样式声明。

## 从 UIObject 类继承的属性

下表列出了 Window 类从 UIObject 类继承的属性。从 Window 对象访问这些属性时，请使用 `WindowInstance.propertyName` 的形式。

属性	说明
<code>UIObject.bottom</code>	只读；对象的底边缘位置（相对于其父对象的底边缘）。
<code>UIObject.height</code>	只读；对象的高度，以像素为单位。
<code>UIObject.left</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.right</code>	只读；对象的右边缘位置（相对于其父对象的右边缘）。
<code>UIObject.scaleX</code>	一个数字，它指示对象相对于其父对象在 x 方向上的缩放因子。
<code>UIObject.scaleY</code>	一个数字，它指示对象相对于其父对象在 y 方向上的缩放因子。
<code>UIObject.top</code>	只读；对象上边缘的位置（相对于其父对象）。
<code>UIObject.visible</code>	一个布尔值，它指示对象是可见的 (true) 还是不可见的 (false)。
<code>UIObject.width</code>	只读；对象的宽度，以像素为单位。
<code>UIObject.x</code>	只读；对象的左边缘（以像素为单位）。
<code>UIObject.y</code>	只读；对象的上边缘（以像素为单位）。

## 从 UIComponent 类继承的属性

下表列出了 Window 类从 UIComponent 类继承的属性。从 Window 对象访问这些属性时，请使用 `WindowInstance.propertyName` 的形式。

属性	说明
<code>UIComponent.enabled</code>	指明组件是否可以接收焦点和输入。
<code>UIComponent.tabIndex</code>	一个数字，指明文档中组件的 Tab 键顺序。

# Window 类的事件摘要

下表列出了 Window 类的事件。

事件	说明
<code>Window.click</code>	单击（松开）关闭按钮时广播。
<code>Window.complete</code>	创建窗口时广播。
<code>Window.mouseDownOutside</code>	在模式窗口外单击（松开）鼠标时广播。

## 从 UIObject 类继承的事件

下表列出了 Window 类从 UIObject 类继承的事件。

事件	说明
<code>UIObject.draw</code>	当对象将要绘制它的图形时进行广播。
<code>UIObject.hide</code>	在对象的状态从可见变为不可见时广播。
<code>UIObject.load</code>	创建子对象时广播。
<code>UIObject.move</code>	移动了对象时广播。
<code>UIObject.resize</code>	在调整对象大小后广播。
<code>UIObject.reveal</code>	在对象的状态从不可见变为可见时广播。
<code>UIObject.unload</code>	卸载子对象时广播。

## 从 UICComponent 类继承的事件

下表列出了 Window 类从 UICComponent 类继承的事件。

事件	说明
<code>UICComponent.focusIn</code>	当对象收到焦点时进行广播。
<code>UICComponent.focusOut</code>	当对象失去焦点时进行广播。
<code>UICComponent.keyDown</code>	当按下按键时进行广播。
<code>UICComponent.keyUp</code>	当松开按键时进行广播。



# Window.click

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.click = function(eventObject:Object) {
 // ...
};
windowInstance.addEventListener("click", listenerObject);
```

用法 2:

```
on (click) {
 // ...
}
```

## 说明

事件：在关闭按钮上单击（松开）鼠标时向所有已注册的侦听器广播。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*windowInstance*) 调度一个事件（在本例中为 `click`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用

`EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **Window** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **Window** 实例 `myWindow`，它将 “\_level0.myWindow” 发送到“输出”面板：

```
on(click){
 trace(this);
}
```

## 示例

以下示例创建一个带有关闭按钮的模式窗口。它定义一个 `click` 处理函数，可在用户单击按钮时调用 `click()` 方法来删除窗口。将一个 **Window** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 {closeButton:true, contentPath:"http://www.flash-mx.com/images/
 image1.jpg"});
var winListener:Object = new Object();
winListener.click = function() {
 my_win.deletePopUp();
};
my_win.addEventListener("click", winListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)、[Window.closeButton](#)

# Window.closeButton

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*windowInstance*.closeButton

## 说明

属性：一个布尔值，指示标题栏是 (`true`) 否 (`false`) 应包含关闭按钮。此属性必须在 [PopUpManager.createPopUp\(\)](#) 方法的 *initObject* 参数中设置。默认值为 `false`。

单击关闭按钮会广播一个 `click` 事件，但不关闭窗口。您必须编写调用 [Window.deletePopUp\(\)](#) 的处理函数，以显式关闭窗口。有关 `click` 事件的更多信息，请参见 [Window.click](#)。

## 示例

以下示例创建一个弹出窗口并设置 `closeButton` 属性，以便向窗口中添加一个关闭按钮。将一个 **Window** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 {closeButton:true, contentPath:"http://www.flash-mx.com/images/
 image1.jpg"});
```

## 另请参见

[PopUpManager.createPopUp\(\)](#)、[Window.click](#)

# Window.complete

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
 ...
}
windowInstance.addEventListener("complete", listenerObject)
```

## 说明

事件：在创建窗口时向所有已注册的侦听器广播。使用此事件调整窗口的大小以适合其内容。

组件实例 (*windowInstance*) 调度一个事件（在本例中为 *complete*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

## 示例

以下示例创建一个窗口，然后定义一个 *complete* 处理函数，该处理函数调整窗口的大小以适合其内容。将一个 **Window** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 {closeButton:true, contentPath:"http://www.flash-mx.com/images/
 imagel.jpg"});
var winListener:Object = new Object();
winListener.click = function(evt_obj:Object) {
 my_win.deletePopUp();
};
winListener.complete = function(evt_obj:Object) {
 my_win.setSize(my_win.content._width, my_win.content._height + 25);
}
my_win.addEventListener("click", winListener);
my_win.addEventListener("complete", winListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# Window.content

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*windowInstance.content*

## 说明

只读属性；对窗口的内容（根影片剪辑）的引用。此属性返回一个 **MovieClip** 对象。从库中附加一个元件时，默认值为所附加元件的一个实例。从 URL 加载内容时，在开始加载操作之前默认值为 `undefined`。

## 示例

以下示例创建一个窗口，然后定义一个 `complete` 处理函数，该处理函数调整窗口的大小以适合其内容。它使用 `content` 属性来引用窗口的影片剪辑内容的宽度。将一个 **Window** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
```

```
 要求:
```

- 库中有 Window 组件

```
*/
```

```
import mx.managers.PopUpManager;
import mx.containers.Window;
```

```
System.security.allowDomain("http://www.flash-mx.com");
```

```
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 {closeButton:true, contentPath:"http://www.flash-mx.com/images/
 image1.jpg"});
var winListener:Object = new Object();
winListener.click = function(evt_obj:Object) {
 my_win.deletePopUp();
};
winListener.complete = function(evt_obj:Object) {
 my_win.setSize(my_win.content._width, my_win.content._height + 25);
}
my_win.addEventListener("click", winListener);
my_win.addEventListener("complete", winListener);
```

# Window.contentPath

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*windowInstance*.contentPath

## 说明

属性：设置要在窗口中显示的内容的名称。此值可以是库中的一个影片剪辑的链接标识符，也可以是要加载的 SWF 或 JPEG 文件的绝对或相对 URL。默认值为 ""（空字符串）。

## 示例

以下示例创建一个窗口，并使用 contentPath 属性指定要在窗口中显示的图像的位置。将一个 Window 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

// 创建窗口。
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true, {
 contentPath:"http://www.flash-mx.com/images/image2.jpg"});
```

# Window.deletePopUp()

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*windowInstance*.deletePopUp()

## 参数

无。

## 返回

无。

## 说明

方法：删除窗口实例和移除模式状态。只能对由 `PopUpManager.createPopUp()` 创建的 **Window** 实例调用此方法。

## 示例

以下示例创建一个模式窗口，然后定义一个 **click** 处理函数，该处理函数调用 `deletePopUp()` 函数来删除此窗口。将一个 **Window** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 {closeButton:true, contentPath:"http://www.flash-mx.com/images/
 imagel.jpg"});
var winListener:Object = new Object();
winListener.click = function() {
 my_win.deletePopUp();
};
my_win.addEventListener("click", winListener);
```

# Window.mouseDownOutside

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

用法 1:

```
var listenerObject:Object = new Object();
listenerObject.mouseDownOutside = function(eventObject:Object) {
 // ...
};
windowInstance.addEventListener("mouseDownOutside", listenerObject);
```

用法 2:

```
on (mouseDownOutside) {
 // ...
}
```

## 说明

事件：在模式窗口外部单击（松开）鼠标时向所有已注册的侦听器广播。此事件很少使用，但如果用户尝试与窗口之外的内容进行交互时，则可以用它来退出窗口。

第一个用法示例使用一个调度程序 / 侦听器事件模型。组件实例 (*windowInstance*) 调度一个事件（在本例中为 *mouseDownOutside*），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您需要定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象的属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `EventDispatcher.addEventListener()` 方法，以将侦听器注册到该实例。当该实例调度该事件时，就会调用该侦听器。

有关更多信息，请参见第 461 页的“[EventDispatcher 类](#)”。

第二个用法示例使用 `on()` 处理函数，并且必须直接附加到一个 **Window** 实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 **Window** 实例 `myWindowComponent`，它将“\_level0.myWindowComponent”发送到“输出”面板：

```
on (mouseDownOutside) {
 trace(this);
}
```



## 示例

以下示例创建一个 **Window** 实例，并定义一个 `mouseDownOutside` 处理函数。当用户单击窗口外部时，该处理函数显示一条消息。将一个 **Window** 组件从“组件”面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager;
import mx.containers.Window;

System.security.allowDomain("http://www.flash-mx.com");

// 创建窗口。
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 undefined, true);

// 创建侦听器对象。
var winListener:Object = new Object();
winListener.mouseDownOutside = function(evt_obj:Object)
{
 trace("mouseDownOutside event triggered.");
}
// 添加侦听器。
my_win.addEventListener("mouseDownOutside", winListener);
```

## 另请参见

[EventDispatcher.addEventListener\(\)](#)

# Window.title

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

`windowInstance.title`

## 说明

属性；一个字符串，它指示标题栏的文本。默认值为 ""（空字符串）。

## 示例

以下示例创建一个弹出窗口，并使用 `title` 属性将标题设置为 “Hello World”。将一个 **Window** 组件从 “组件” 面板拖到当前文档的库中，然后将以下代码添加到第一帧中：

```
/**
 * 要求：
 * - 库中有 Window 组件
 */

import mx.managers.PopUpManager
import mx.containers.Window

// 创建窗口。
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true);

// 设置窗口属性。
my_win.title = "Hello World!";
my_win.setSize(200, 100);
my_win.move(20, 20);
```

# Window.titleStyleDeclaration

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX 2004。

## 用法

*windowInstance*.titleStyleDeclaration

## 说明

属性；一个字符串，指明设置窗口标题栏格式的样式声明。默认值为 `undefined`，它指示粗体、白色文本。

## 示例

以下示例创建一个 **CSS** 样式声明，使文本为 14 磅大小、斜体并具有下划线。它使用 `titleStyleDeclaration` 属性将该样式应用于所创建的弹出窗口的标题。将一个 **Window** 组件从 “组件” 面板拖到当前文档的库中，然后将以下代码添加到第一帧中。

```
/**
 * 要求：
 * - 库中有 Window 组件
 */
```

```

import mx.styles.CSSStyleDeclaration
import mx.managers.PopUpManager
import mx.containers.Window

// 创建一个名为 TitleStyles 的新 CSSStyleDeclaration,
// 并在全局样式列表中列出。
_global.styles.TitleStyles = new CSSStyleDeclaration();

// 自定义样式。
_global.styles.TitleStyles.fontStyle = "italic";
_global.styles.TitleStyles.textDecoration = "underline";
_global.styles.TitleStyles.color = 0xff0000;
_global.styles.TitleStyles.fontSize = 14;

// 创建窗口。
var my_win:MovieClip = PopUpManager.createPopUp(this, Window, true,
 {closeButton:true, titleStyleDeclaration:"TitleStyles"});

// 设置窗口属性。
my_win.title = "Testing Styles";
my_win.setSize(200, 100);
my_win.move(20, 20);

// 创建侦听器对象。
var winListener:Object = new Object();
winListener.click = function(evt_obj:Object) {
 trace("closing window");
 evt_obj.target.deletePopUp();
};
// 添加侦听器。
my_win.addEventListener("click", winListener);

```

有关样式的更多信息，请参见《使用组件》中的“使用样式自定义组件的颜色和文本”。



# XMLConnector 组件（仅限 Flash Professional）

**XMLConnector** 组件允许使用 HTTP GET 和 POST 操作读写 XML 文档。它充当其它组件和外部 XML 数据源之间的连接器。**XMLConnector** 组件使用 Flash 创作环境中的 **ActionScript** 代码或数据绑定功能与应用程序中的其它组件进行通信。**XMLConnector** 组件具有属性、方法和事件，但它没有运行时可视外观。

## 使用 XMLConnector 组件（仅限 Flash Professional）

**XMLConnector** 组件使应用程序可以访问通过 HTTP 返回或接收 XML 的任何外部数据源。连接外部 XML 数据源以及将该数据源的参数和结果用于您的应用程序的最简易方法是指定一个架构 XML 文档的结构，它标识可绑定到的文档中的数据元素。

有关使用 **XMLConnector** 组件的更多信息，请参见《使用 Flash》中的“用 **XMLConnector** 组件连接 XML 数据（仅限 Flash Professional）”。

### XMLConnector 的参数

您可以在“组件”检查器的“参数”选项卡中为每个 **XMLConnector** 组件实例设置以下创作参数：

**URL**，一个指向外部 XML 数据源的字符串。

**direction**，一个字符串，它定义调用 **XMLConnector.trigger()** 方法时要执行的 HTTP 操作。此参数可以使用的值为 "send"、"receive" 或 "send/receive"。

值 "send" 意味着 XML 数据（通过 HTTP POST）被发送到 URL，但 Flash 忽略返回的所有数据。对 **XMLConnector.results** 属性不做任何设置，而且没有任何 **result** 事件发生。

值 "receive" 意味着不发送任何数据到 XML URL。Flash 通过 HTTP GET 访问 URL，并预期返回有效的 XML 数据。

值 "send/receive" 意味着 Flash 通过 HTTP POST 发送 XML 数据，并预期返回有效的 XML 数据。

如果 direction 参数为 null 或无法识别，则默认值为 "send/receive"。

**ignoreWhite**，一个布尔值，默认设置为 false。如果将此参数设置为 true，则仅包含空格的文本节点在分析过程中会被丢弃。带有前导或尾随空格的文本节点不受影响。

**multipleSimultaneousAllowed**，一个布尔值；如果设置为 true，则允许 trigger() 操作在另一个 trigger() 操作正在进行时启动。多个同时进行的 trigger() 操作可能不会按调用它们时的相同顺序完成。此外，Flash Player 可能会对同时发生的网络操作的数目加以限制。该限制将因版本和平台而异。如果将该参数设置为 false，则 trigger() 操作不能在另一个操作正在进行时启动。

**suppressInvalidCall**，一个布尔值；如果设置为 true，它会在数据参数无效时禁止 trigger() 操作。如果 suppressInvalidCall 设置为 false，则 trigger() 操作将在需要时执行并使用无效数据。

## XMLConnector 组件的通用工作流程

以下过程概括了 XMLConnector 组件的典型工作流程。

**要使用 XMLConnector 组件：**

1. 将 XMLConnector 组件的一个实例添加到应用程序，并为其指定一个实例名称。
2. 使用“组件”检查器的“参数”选项卡，输入要访问的外部 XML 数据源的 URL。
3. 使用“组件”检查器的“架构”选项卡指定 XML 文档的架构。



可以使用“导入示例架构”按钮使此过程自动化。

4. 使用“组件”检查器的“绑定”选项卡将数据元素（params 和 results）从 XML 文档绑定到应用程序中可视组件的属性。

例如，您可以连接到提供天气数据的 XML 文档，并将“地点”和“温度”数据元绑定到应用程序中的标签组件。指定城市的名称和温度在运行时即会出现在应用程序中。

5. 使用以下方法之一添加触发器以启动数据绑定操作：
  - 将“触发器数据源”行为附加到某个按钮上。
  - 添加自己的 ActionScript 来调用 XMLConnector 组件上的 trigger() 方法。
  - 创建 XML 参数与 UI 控件之间的绑定，并将其 Kind 属性设置为 AutoTrigger。有关更多信息，请参见《使用 Flash》中的“架构种类”。

若要查看使用 XMLConnector 组件连接和显示 XML 的分步示例，请参见“数据集成”教程中的“XML 教程: Timesheet”，网址为 [www.macromedia.com/go/data\\_integration](http://www.macromedia.com/go/data_integration)。

# XMLConnector 类（仅限 Flash Professional）

继承 RPCCall > XMLConnector

ActionScript 类名称 mx.data.components.XMLConnector

XMLConnector 类允许使用 HTTP 发送或接收 XML 文件。您可以使用 ActionScript 将其它组件绑定到返回 XML 数据的数据源，实现组件之间的通信。

## XMLConnector 类的方法摘要

下表列出了 XMLConnector 类的方法。

方法	说明
<code>XMLConnector.trigger()</code>	启动远程过程调用。

## XMLConnector 类的属性摘要

下表列出了 XMLConnector 类的属性。

属性	说明
<code>XMLConnector.direction</code>	指示是发送、接收数据还是两者同时进行。
<code>XMLConnector.ignoreWhite</code>	指示在分析过程中是否放弃仅包含空格的文本节点。
<code>XMLConnector.multipleSimultaneousAllowed</code>	指明是否可以同时进行多个调用。
<code>XMLConnector.params</code>	指定在执行下一个 <code>trigger()</code> 操作时要发送到服务器的数据。
<code>XMLConnector.results</code>	标识作为 <code>trigger()</code> 操作的结果从服务器接收的数据。
<code>XMLConnector.suppressInvalidCalls</code>	指明在参数无效的情况下是否禁止调用。
<code>XMLConnector.URL</code>	组件在 HTTP 操作中使用的 URL。

# XMLConnector 类的事件摘要

下表列出了 XMLConnector 类的事件。

事件	说明
<code>XMLConnector.result</code>	远程过程调用成功完成后广播。
<code>XMLConnector.send</code>	当 <code>trigger()</code> 方法正在执行时（在收集了参数数据之后，但在验证这些数据 and 启动远程调用之前）进行广播。
<code>XMLConnector.status</code>	在远程过程调用启动时广播，以通知用户操作状态。

## XMLConnector.direction

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

`componentInstance.direction`

### 说明

属性；指示是发送、接收数据还是两者同时进行。值如下所示：

- `send` `params` 属性的 XML 数据通过 HTTP POST 方法发送到 XML 文档的 URL。忽略任何返回的数据。对 `results` 属性不做任何设置，而且没有任何 `result` 事件。



`params` 和 `results` 属性以及 `result` 事件继承于 RPC 组件 API。

- `receive` 不向此 URL 发送 `params` 数据。通过 HTTP GET 访问 XML 文档的 URL，并预期从该 URL 收到有效的 XML 数据。
- `send/receive` `params` 数据被发送到 URL，并预期从该 URL 收到有效的 XML 数据。

### 示例

以下示例为文档 `mysettings.xml` 将 `direction` 设置为 `receive`：

```
myXMLConnector.direction = "receive";
myXMLConnector.URL = "mysettings.xml";
myXMLConnector.trigger();
```



# XMLConnector.ignoreWhite

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.ignoreWhite*

## 说明

属性；一个布尔值。如果将此参数设置为 `true`，则仅包含空格的文本节点在分析过程中会被丢弃。带有前导或尾随空格的文本节点不受影响。默认设置为 `false`。

## 示例

以下代码将 `ignoreWhite` 属性设置为 `true`：

```
myXMLConnector.ignoreWhite = true;
```

# XMLConnector.multipleSimultaneousAllowed

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.multipleSimultaneousAllowed*

## 说明

属性；指明是否可以同时进行多个调用。如果此属性为 `false`，则 `XMLConnector.trigger()` 方法将在另一个调用已在执行时执行一个调用。并使用代码 `CallAlreadyInProgress` 发出 `status` 事件。如果此属性为 `true`，则执行调用。

当同时进行多个调用时，不保证这些调用的完成顺序与触发它们的顺序相同。此外，浏览器和 / 或操作系统可以对同时执行的网络操作的数目加以限制。最可能遇到的限制是，浏览器实施可同时下载的 `URL` 的最大数目。此限制通常可在浏览器配置。但是，即使遇到此限制，浏览器也会对“流”进行排队，不会影响 **Flash** 应用程序的预期行为。

## 示例

此示例通过将 `direction` 属性设置为 `receive`，使用 **XMLConnector** 组件来检索远程 XML 文件。将一个 **XMLConnector** 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
 trace("results:");
 trace(evt.target.results);
 trace("");
};
xmlListener.status = function(evt:Object) {
 trace("status::"+evt.code);
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/tips/tips.xml";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = true;
myXMLConnector.trigger();
myXMLConnector.trigger();
myXMLConnector.trigger();
```

此示例指定 XML 文件的 URL，并将 `multipleSimultaneousAllowed` 设置为 `false`。它触发 **XMLConnector** 实例三次，这将导致事件侦听器的 `status` 方法将错误代码 `CallAlreadyInProgress` 在“输出”面板中显示两次。第一次尝试被成功地从 Flash 发送到服务器。当第一次触发成功收到结果时，将广播 `result` 事件并将收到的 XML 数据包显示在“输出”面板中。

# XMLConnector.params

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.params*

## 说明

属性：指定在执行下一个 `trigger()` 操作时要发送到服务器的数据。每个 **RPC** 组件均定义使用此数据的方式以及有效的类型。

## 示例

以下示例为 `myXMLConnector` 定义 `name` 和 `city` 参数：

```
myXMLConnector.params = new XML("<mydoc><name>Bob</name><city>Oakland</city></mydoc>");
```

# XMLConnector.result

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

```
componentInstance.addEventListener("result", myListenerObject)
```

## 说明

事件：在远程过程调用操作成功完成后广播。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 "result"
- `target`：对发出该事件的对象的引用（如 **WebServiceConnector** 组件）

您可以使用 `results` 属性获取实际结果值。

## 示例

以下示例为 `result` 事件定义 `res` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var res = function (ev) {
 trace(ev.target.results);
};
xcon.addEventListener("result", res);
```

# XMLConnector.results

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.results*

## 说明

属性；标识作为 `trigger()` 操作的结果从服务器接收的数据。每个 **RPC** 组件均定义获取此数据的方式以及有效的类型。这些数据在 **RPC** 操作成功完成后出现，并由 `result` 事件通知。它在该组件卸载前或执行下一个 **RPC** 操作前可用。

返回的数据可能会非常大。您可以通过以下两种方式管理这些数据：

- 选择适当的影片剪辑、时间轴或屏幕作为 **RPC** 组件的父级。在父级销毁后该组件的内存将可用于垃圾回收。
- 在 **ActionScript** 中，可以随时将 `null` 分配给该属性。

## 示例

以下示例跟踪 `myXMLConnector` 的 `results` 属性：

```
trace(myXMLConnector.results);
```

# XMLConnector.send

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance.addEventListener("send", myListenerObject)*

### 说明

事件：当 `trigger()` 操作执行时（在收集了参数数据之后，但在验证这些数据 and 启动远程调用之前）进行广播。此位置适合于存放将在调用前修改参数数据的代码。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 "send"
- `target`：对发出该事件的对象的引用（如 `WebServiceConnector` 组件）

您可以使用 `params` 属性获取或者修改实际参数值。

### 示例

以下示例为 `send` 事件定义 `sendFunction` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var sendFunction = function (sendEnv) {
 sendEnv.target.params = [newParam_txt.text];
};
xcon.addEventListener("send", sendFunction);
```

## XMLConnector.status

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

```
componentInstance.addEventListener("status", myListenerObject)
```

### 说明

事件：在远程过程调用启动时广播，以通知用户操作状态。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 "status"
- `target`：对发出该事件的对象的引用（如 `WebServiceConnector` 组件）
- `code`：一个字符串，提供已发生的特定情况的名称
- `data`：一个对象，其内容取决于代码

如果调用时发生问题，则 `status` 事件的代码字段设置为 `Fault`，如下所示：

代码	数据	说明
Fault	{faultcode:code, faultstring:string, detail:detail, element:element, faultactor:actor}	如果在处理调用的过程中发生其它问题，则会发出此事件。数据是 SOAPFault 对象。在发生此事件后，则认为尝试的调用已完成，并且不会发生 <code>result</code> 或 <code>send</code> 事件。

下面是可能伴随 `status` 事件发生的错误：

FaultCode	FaultString	备注
XMLConnector.Not.XML	params 不是 XML 对象	params 必须是 ActionScript XML 对象。
XMLConnector.Parse.Error	params 具有 XML 分析错误 NN。	params XML 对象的 <code>status</code> 属性具有非零值 NN。若要查看可能的错误 NN，请参见《ActionScript 2.0 语言参考》中的 <code>XML.status</code> 。
XMLConnector.No.Data.Received	没有从服务器接收到任何数据	由于不同的浏览器限制，此消息可能表示 (a) 服务器 URL 无效、无响应或返回 HTTP 错误代码；或 (b) 服务器请求成功但响应为 0 字节的数据。要解决这一限制问题，应对应用程序进行调整，使服务器在任何情况下都不会返回 0 字节数据。如果接收到错误代码 <code>XMLConnector.No.Data.Received</code> ，则表示发生了服务器错误。您可以相应地通知用户。
XMLConnector.Results.Parse.Error	收到的数据具有 XML 分析错误 NN	Flash Player 内置的 XML 分析器确定收到的 XML 无效。若要查看可能的错误 NN，请参见《ActionScript 2.0 语言参考》中的 <code>XML.status</code> 。
XMLConnector.Params.Missing	Direction 为“send”或“send/receive”，但 params 为空。	

## 示例

以下示例为 status 事件定义 statusFunction 函数，并且将该函数分配给 addEventListener 事件处理函数：

```
var statusFunction = function (stat) {
 trace(stat.code);
 trace(stat.data.faultcode);
 trace(stat.data.faultstring);
};
xcon.addEventListener("status", statusFunction);
```

# XMLConnector.suppressInvalidCalls

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance*.suppressInvalidCalls

## 说明

属性；指明在参数无效的情况下是否禁止调用。如果此属性为 true，则 trigger() 方法在绑定参数未通过验证的情况下将不执行调用。并使用代码 InvalidParams 发出 status 事件。如果此属性为 false，则使用需要的无效数据执行该调用。

## 示例

由于没有传递必需的参数，此示例会显示一个错误。将一个 XMLConnector 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
 trace("results:");
 trace(evt.target.results);
 trace("");
};
xmlListener.status = function(evt:Object) {
 switch (evt.code) {
 case 'Fault' :
 trace("ERROR! ["+evt.data.faultcode+"]");
 trace("\t"+evt.data.faultstring);
 break;
 case 'InvalidParams' :
```

```
 trace("ERROR! [" + evt.code + "]");
 break;
 }
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "send/receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/login_xml.cfm";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = false;
// myXMLConnector.params = new XML("<login username='Mort'
 password='Guacamole' />");
myXMLConnector.trigger();
```

删除第二行到最后一行的代码注释，以使代码片断正常工作。

## XMLConnector.trigger()

### 可用性

Flash Player 6 (6.0.79.0)。

### 版本

Flash MX Professional 2004。

### 用法

*componentInstance.trigger()*

### 说明

方法；通过 **XMLConnector** 组件启动远程过程调用。可以是对指定的 XML 文件的获取或发送。如果操作成功，则操作结果将出现在 **RPC** 组件的 **results** 属性中。

**trigger()** 方法执行以下步骤：

1. 如果任何数据绑定到 **params** 属性，则该方法执行所有绑定以确保提供最新数据。这样做还会导致发生数据验证。
2. 如果数据无效且 **suppressInvalidCalls** 设置为 **true**，则操作将停止。
3. 如果操作继续，则发出 **send** 事件。
4. 使用指示的连接方法（例如 **HTTP**）启动实际远程调用。



## 示例

此示例通过将 `direction` 属性设置为 `receive`，使用 **XMLConnector** 来检索远程 XML 文件。将一个 **XMLConnector** 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
 trace("results:");
 trace(evt.target.results);
 trace("");
};
xmlListener.status = function(evt:Object) {
 trace("status::"+evt.code);
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/tips/tips.xml";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = true;
myXMLConnector.trigger();
myXMLConnector.trigger();
myXMLConnector.trigger();
```

这段代码指定 XML 文件的 URL，并将 `multipleSimultaneousAllowed` 设置为 `false`。它触发 **XMLConnector** 实例三次，这将导致事件侦听器的 `status` 方法将错误代码 `CallAlreadyInProgress` 在“输出”面板中显示两次。第一次尝试被成功地从 Flash 发送到服务器。当第一次触发成功收到结果时，将广播 `result` 事件并将收到的 XML 数据包显示在“输出”面板中。

# XMLConnector.URL

## 可用性

Flash Player 6 (6.0.79.0)。

## 版本

Flash MX Professional 2004。

## 用法

*componentInstance*.URL

## 说明

属性：此组件在执行 HTTP 操作时所使用的 URL。此 URL 可以是绝对或相对的 URL。此 URL 受所有标准 Flash Player 安全保护机制的约束（有关 Flash Player 安全保护机制的更多信息，请参见《学习 Flash 中的 ActionScript 2.0》中的“了解安全性”）。

## 示例

此示例通过将 direction 属性设置为 receive，使用 XMLConnector 组件来检索远程 XML 文件。将一个 XMLConnector 组件拖到库中，然后在时间轴的第 1 帧中输入以下代码：

```
import mx.data.components.XMLConnector;
var xmlListener:Object = new Object();
xmlListener.result = function(evt:Object) {
 trace("results:");
 trace(evt.target.results);
 trace("");
};
xmlListener.status = function(evt:Object) {
 trace("status::"+evt.code);
};
var myXMLConnector:XMLConnector = new XMLConnector();
myXMLConnector.addEventListener("result", xmlListener);
myXMLConnector.addEventListener("status", xmlListener);
myXMLConnector.direction = "receive";
myXMLConnector.URL = "http://www.flash-mx.com/mm/tips/tips.xml";
myXMLConnector.multipleSimultaneousAllowed = false;
myXMLConnector.suppressInvalidCalls = true;
myXMLConnector.trigger();
myXMLConnector.trigger();
myXMLConnector.trigger();
```

这段代码指定 XML 文件的 URL，并将 multipleSimultaneousAllowed 设置为 false。它触发 XMLConnector 实例三次，这将导致事件侦听器的 status() 方法将错误代码 CallAlreadyInProgress 在“输出”面板中显示两次。第一次尝试被成功地从 Flash 发送到服务器。当第一次触发成功收到结果时，将广播 result 事件并将收到的 XML 数据包显示在“输出”面板中。

**ActionScript 类名称** mx.xpath.XPathAPI

使用 XPathAPI 类可以在 Macromedia Flash 中执行简单的 XPath 搜索。基于节点名称和属性值搜索 XML 数据包时，该类很有用。也就是说，您可以使用 XpathAPI 方法在 XML 文档中快速找到节点和属性。

若要在 Flash 中执行 XPath 搜索，首先需要通过添加 DataBindingClass（如果尚未添加它）将 XPathAPI 类包括在 Flash 库中。如果已经设置了绑定，则可能已经自动包括了该类；否则，您需要从公用库（“窗口” > “公用库” > “类”）中选择该类。只需从 Classes.fla 库面板中将 DataBindingClasses 组件的一个副本拖到当前 Flash 文档的库中。现在，您可以通过以下两种方法来导入类：键入 `import mx.xpath.XPathAPI`，或者在访问类的方法时用 `mx.xpath.XPathAPI.method_name` 作为类方法的前缀来使用完全限定的类名。

有关该类的更多信息，请查看 Flash 文档资源中心，网址为：[www.macromedia.com/go/xpathapi](http://www.macromedia.com/go/xpathapi)。



# XUpdateResolver 组件（仅限 Flash Professional）

# 61

Resolver 组件和 DataSet 组件（Flash 数据结构中的数据管理功能的一部分）一起使用，用于保存对外部数据源所做的更改。解析程序接受一个 `DataSet.deltaPacket` 对象，并将其转换为格式适合解析程序类型的更新数据包。然后，此更新包通过一个连接器组件传送到外部数据源。Resolver 组件在运行时没有可视外观。

有关如何使用 DataSet 组件管理 Flash 中数据的一般信息，请参见《使用 Flash》中的“数据管理（仅限 Flash Professional）”。

XUpdate 是一种描述对 XML 文档所进行更改的标准，各种 XML 数据库（如 Xindice 和 XHive）都支持该标准。XUpdateResolver 组件将对 DataSet 组件所作的更改转换为 XUpdate 语句。XUpdateResolver 组件的更新以 XUpdate 数据包的形式发送，并通过某个连接对象与数据库或服务器进行通信。XUpdateResolver 组件从 DataSet 组件获取变量增量包，将其自己的更新数据包发送到连接器，从连接中接收服务器错误，并且将这些错误传送回 DataSet 组件 - 全都使用可绑定的属性。

有关 XUpdate 语言规范的工作草案的信息，请参见 <http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html>。有关 Flash 数据结构的信息，请参见《使用 Flash》中的“数据解析（仅限 Flash Professional）”；有关解析 XML 数据的信息，请参见《使用 Flash》中的“用 XUpdateResolver 组件解析 XML 数据（仅限 Flash Professional）”。



您也可以使用 XUpdateResolver 组件将数据更新发送到能够解析 Xupdate 语言的任何外部数据源（例如，ASP 页、Java servlet 或 ColdFusion 组件）。

# 使用 XUpdateResolver 组件（仅限 Flash Professional）

仅当 Flash 应用程序包含 DataSet 组件且必须将更新发送回外部数据源时，才使用 XUpdateResolver 组件。

XUpdateResolver 组件使用 DataSetDeltaToXUpdateDelta 编码器与 DataSet 组件通信。该编码器创建 XPath 语句，这些语句按照 DataSet 组件的变量增量包所含的信息唯一地标识 XML 文件中的节点。XUpdateResolver 组件使用此信息生成 XUpdate 语句。有关 DataSetDeltaToXUpdateDelta 编码器的更多信息，请参见《使用 Flash》中的“架构编码器”。

有关使用 XUpdateResolver 组件的更多信息，请参见《使用 Flash》中的“数据解析（仅限 Flash Professional）”。

## XUpdateResolver 组件参数

XUpdateResolver 组件具有一个创作参数，即 includeDeltaPacketInfo 参数（布尔类型）。如果将此参数设置为 true，更新数据包将包含附加信息，外部数据源可以使用此信息生成可发送回应用程序的结果。此信息包含一个唯一的事务和操作 ID（由数据集内部使用）。

提醒

更新数据包中包含的附加信息会使 XUpdate 无效。只有在要将此信息存储在服务器对象内并使用它来生成结果数据包时，您才需要选择添加此信息。在此方案下，服务器对象将根据自己的需要从更新数据包中提取信息，然后将（现在有效的）XUpdate 传递到数据库。

下面是当 includeDeltaPacketInfo 参数设置为 false 时，XML 更新数据包的示例：

```
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
 <xupdate:remove select="/datapacket/row[@id='100']"/>
</xupdate:modifications>
```

下面是当 includeDeltaPacketInfo 参数设置为 true 时，XML 更新数据包的示例：

```
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate"
 transId="46386292065:Wed Jun 25 15:52:34 GMT-0700 2003">
 <xupdate:remove select="/datapacket/row[@id='100']" opId="0123456789"/>
</xupdate:modifications>
```

# XUpdateResolver 组件的通用工作流程

以下过程概述了 XUpdateResolver 组件的典型工作流程。

## 要使用 XUpdateResolver 组件：

1. 将 XMLConnector 组件的两个实例以及每个 DataSet 组件和 XUpdateResolver 组件的一个实例添加到应用程序，并为它们指定实例名称。
2. 选择第一个 XMLConnector 组件，然后使用“组件”检查器的“参数”选项卡输入要访问的外部 XML 数据源的 URL。
3. 在 XMLConnector 组件仍然处于选定状态的情况下，单击“组件”检查器的“架构”选项卡，并导入 XML 示例文件以生成架构。



如果要访问绑定到数据集的数组的子元素，您可能需要为 XML 文件创建一个虚拟架构。有关更多信息，请参见《使用 Flash》中的“虚拟架构”。

4. 使用“组件”检查器的“绑定”选项卡将 XMLConnector 组件内的数组绑定到 DataSet 组件的 dataProvider 属性。
5. 选择 DataSet 组件，并使用“组件”检查器的“架构”选项卡创建将绑定到数组内对象的字段的 DataSet 字段。
6. 使用“组件”检查器的“绑定”选项卡将数据元（DataSet 字段）绑定到应用程序中的可视组件。
7. 选择 XUpdateResolver 组件的“架构”选项卡。在 deltaPacket 组件属性处于选定状态的情况下，使用“架构属性”窗格将 encoder 属性设置为 DataSetDeltaToXUpdateDelta 编码器。
8. 选择 Encoder Options，并输入唯一标识 XML 文件中的行节点的 rowNodeKey 值。



rowNodeKey 值结合使用 XPath 语句和字段参数来定义应如何为增量数据包内包含的更新数据生成唯一的 XPath 语句。有关 DataSetDeltaToXUpdateDelta 编码器的信息，请参见《使用 Flash》中的“架构编码器”。

9. 单击“绑定”选项卡，并在 XUpdateResolver 组件的 deltaPacket 属性和 DataSet 组件的 deltaPacket 属性之间创建一个绑定。
10. 创建另一个从 xupdatePacket 属性到第二个 XMLConnector 组件的绑定，以便将数据发送回外部数据源。



xupdatePacket 属性包含将发送到服务器的格式化增量数据包（XUpdate 语句）。

11. 添加触发器以启动数据绑定操作：使用附加至按钮的“触发器数据源”行为，或添加 ActionScript。

除了这些步骤之外，您还可以创建绑定以应用通过 XUpdateResolver 组件从服务器重新发送到数据集的结果数据包。

若要查看使用 XUpdate 将数据解析为外部数据源的分步示例，请参见“数据集成”教程中的“更新时间表”，网址为 [www.macromedia.com/go/data\\_integration](http://www.macromedia.com/go/data_integration)。

## XUpdateResolver 类（仅限 Flash Professional）

继承 MovieClip > XUpdateResolver

ActionScript 类名称 mx.data.components.XUpdateResolver

XUpdateResolver 类的属性和事件允许使用 DataSet 组件将更改保存到外部数据源。

### XUpdateResolver 类的属性摘要

下表列出了 XUpdateResolver 类的属性。

属性	说明
<code>XUpdateResolver.deltaPacket</code>	包含对 DataSet 组件所作更改的说明。DataSet 组件的 <code>deltaPacket</code> 属性应绑定到此属性，以便在调用 <code>DataSet.applyUpdates()</code> 时，该绑定可复制此属性，并由解析程序创建 XUpdate 数据包。
<code>XUpdateResolver.includeDeltaPacketInfo</code>	在 XUpdate 节点上的属性中包含变量增量包的其它信息。
<code>XUpdateResolver.updateResults</code>	说明更新的结果。
<code>XUpdateResolver.xupdatePacket</code>	包含对 DataSet 组件所作更改的 XUpdate 翻译。

### XUpdateResolver 类的事件摘要

下表列出了 XUpdateResolver 类的事件。

事件	说明
<code>XUpdateResolver.beforeApplyUpdates</code>	在创建 XML 包后并紧接着在发送该包之前由解析程序组件调用，以立即进行自定义的修改。
<code>XUpdateResolver.reconcileResults</code>	由解析程序组件调用以比较两个包。



# XUpdateResolver.beforeApplyUpdates

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData*.beforeApplyUpdates(*eventObject*)

## 参数

*eventObject* 解析程序事件对象；描述在通过连接器将更新发送到数据库之前对 XML 数据包进行的自定义。此事件对象应包含以下属性：

属性	说明
target	对象；生成此事件的解析程序。
type	字符串；事件的名称。
updatePacket	XML 对象；将要应用的 XML 对象。

## 返回

无。

## 说明

事件：在为新的变量增量包创建 XML 包后，并紧接着在使用数据绑定发送该包之前由解析程序组件调用，以立即进行自定义的修改。在将更新的数据发送到连接器之前，可以使用此事件处理函数对 XML 进行自定义的修改。

## 示例

以下范例将用户身份验证数据添加到 XML 包：

```
on (beforeApplyUpdates) {
 // 添加用户身份验证数据。
 var userInfo = new XML(""+getUserId()+" "+getPassword()+"");
 xupdatePacket.firstChild.appendChild(userInfo);
}
```

# XUpdateResolver.deltaPacket

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.deltaPacket*

## 说明

属性；包含对 **DataSet** 组件所作更改的说明。此属性为 `deltaPacket` 类型，它接收要转换为 **XUpdate** 数据包的增量数据包，并从放置在 `updateResults` 属性中的所有服务器结果输出增量数据包。此属性提供一个方法，您可以通过此方法在将更新的数据发送到连接器前对 XML 进行自定义修改。

`updateResults` 属性中的消息视为错误。这意味着，具有消息的增量被再次添加到增量数据包中，这样，下次将增量数据包发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个增量数据包前，将这些消息提供给用户并可以修改增量。

**DataSet** 组件的 `deltaPacket` 属性应绑定到此属性，以便在调用 `DataSet.applyUpdates()` 时，该绑定可复制此属性，并由解析程序创建 **XUpdate** 数据包。

# XUpdateResolver.includeDeltaPacketInfo

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.includeDeltaPacketInfo*

## 说明

属性；布尔类型的属性，如果为 `true`，则在 **XUpdate** 节点上的属性中包含增量数据包的其它信息。此信息由事务 ID 和操作 ID 组成。

若要查看产生的 XML 的示例，请参见第 1386 页的 [“XUpdateResolver 组件参数”](#)。

# XUpdateResolver.reconcileResults

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

```
resolveData.reconcileResults(eventObject)
```

## 参数

*eventObject* **ResolverEvent** 对象；它描述用于比较两个更新数据包的事件对象。此事件对象应包含以下属性：

属性	说明
target	对象；生成此事件的解析程序。
type	字符串；事件的名称。

## 返回

无。

## 说明

事件；由解析程序组件调用以比较两个包。在从服务器收到结果后，并紧接着在通过数据绑定传送包含操作结果的变量增量包之前，使用此回调来插入任何代码。这是放置对来自服务器的消息进行处理的代码的好地方。

## 示例

下面的示例协调两个更新数据包并在成功后清除更新：

```
on (reconcileResults) {
 // 检查结果。
 if(examine(updateResults))
 myDataSet.purgeUpdates();
 else
 displayErrors(results);
}
```

# XUpdateResolver.updateResults

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.updateResults*

## 说明

属性：deltaPacket 类型的属性，包含使用连接器从服务器返回的更新的结果。使用此属性可将错误和更新的数据从服务器传送到 **DataSet** 组件；例如，当服务器为自动分配的字段分配新 ID 时。将此属性绑定到连接器的 results 属性，以便它可以接收更新的结果并将结果传送回 **DataSet** 组件。

updateResults 属性中的消息视为错误。这意味着，具有消息的增量被再次添加到增量数据包中，这样，下次将增量数据包发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个增量数据包前，将这些消息提供给用户并可以修改增量。

# XUpdateResolver.xupdatePacket

## 可用性

Flash Player 7。

## 版本

Flash MX Professional 2004。

## 用法

*resolveData.xupdatePacket*

## 说明

属性：xml 类型的属性，包含对 **DataSet** 组件所作更改的 XUpdate 转换。将此属性绑定到连接器组件的属性，后者将经过翻译的更改的更新包传送回 **DataSet** 组件。

# 索引

## 英文

### Accordion 组件

- 包 45
- 参数 36
- 创建应用程序 37
- 方法 46
- 继承 45
- 将缓动方法应用于 1211
- 使用外观 41
- 使用样式 40
- 事件 48
- 属性 47
- 自定义 40

### Alert 组件

- 包 67
- 参数 62
- 创建应用程序 62
- 方法 67
- 继承 67
- 使用外观 65
- 使用样式 63
- 事件 70
- 属性 68
- 自定义 63

### Binding 类

- 方法 193
- 关于 192

### Button 组件

- 包 94
- 参数 84
- 创建应用程序 85
- 方法 95, 1039
- 关于 83
- 继承 94
- 使用外观 89
- 使用样式 87
- 事件 97

属性 96

自定义 87

### CellRenderer API

- 方法 108, 109
- 关于 101
- 使用 102
- 示例 103
- 属性 109

### CheckBox 组件

- 包 124
- 参数 120
- 创建应用程序 120
- 方法 125
- 关于 119
- 继承 124
- 使用外观 123
- 使用样式 122
- 事件 128
- 属性 126

### Collection 接口

- 方法 136
- 关于 135

### ComboBox 组件

- 包 152
- 参数 147
- 创建应用程序 148
- 方法 153
- 关于 145
- 继承 152
- 将缓动方法应用于 1212
- 使用外观 151
- 使用样式 149
- 事件 156
- 属性 154

### ComponentMixins 类

- 方法 210
- 关于 210

## CustomFormatter 类

范例 196

方法 198

关于 196

## CustomValidator 类

方法 200

关于 200

## DataGrid 组件

DataGridColumn 类 282

包 243

参数 235

创建应用程序 236

方法 243

方法, 继承的 244, 245

关于 231

继承 243

继承的属性 246

进行动画处理 1213

进行交互 232

设计 234

使用 233

使用外观 242

使用样式 240

事件 248

事件, 继承的 248, 249

视图, 数据 233, 234

属性 245, 246

属性, 继承的 247

数据模型 233, 234

性能策略 238

自定义 240

## DataGridColumn 类

关于 280

属性 281

## DataHolder 组件

包 293

创建应用程序 292

关于 291

继承 293

属性 293

## DataProvider API

包 295

方法 296

关于 295

事件 296

属性 296

## DataSet 组件

包 313

参数 310

创建应用程序 311

方法 313

关于 309

继承 313

事件 315

属性 314

通用工作流程 310

## DataType 类

方法 217

关于 216

属性 217

## DateChooser 组件

包 386

参数 381

创建应用程序 382

方法 387

关于 381

继承 386

类 386

使用外观 385

使用样式 383

事件 389

属性 388

自定义 383

## DateField 组件

包 406

参数 401

创建应用程序 402

方法 406

关于 401

继承 406

使用外观 405

使用样式 403

事件 408

属性 407

## Delegate 类

方法 425

关于 425

## Delta 接口

方法 433

关于 433

## DeltaItem 类

关于 427

属性 427

## DeltaPacket 接口

方法 443

关于 443

## DepthManager 类 449

方法 449

## detail 属性

PendingCall.onFault 1315

- WebService.onFault 1328
- element
  - PendingCall.onFault 1315
  - WebService.onFault 1328
- EndPoint 类
  - 方法 205
  - 关于 204
- EventDispatcher 类
  - 包 462
  - 方法 462
  - 关于 461
- faultactor 属性
  - PendingCall.onFault 1315
  - WebService.onFault 1328
- faultcode 属性
  - PendingCall.onFault 1315
  - WebService.onFault 1328
- faultstring 属性
  - PendingCall.onFault 1315
  - WebService.onFault 1328
- FLV, 播放 467
- FLVPlayback 组件 467
  - VideoError 类 642
  - VideoPlayer 类 649
  - 播放多个 FLV 481
  - 创建新外观 491
  - 创建应用程序 469
  - 方法 497
  - 类 497
  - 使用 469
  - 使用 SMIL 文件 653
  - 使用提示点 474
  - 事件 503
  - 属性 499
  - 自定义 484
  - 组件参数 471
- FocusManager 类
  - 包 667
  - 创建应用程序 666
  - 方法 667
  - 关于 663
  - 继承 667
  - 事件 669
  - 属性 668
  - 自定义 666
- Form 类
  - 包 678
  - 参数 678
  - 方法 678
  - 关于 677

- 继承 678
- 事件 682
- 属性 680
- Iterator 接口
  - 包 691
  - 方法 691
  - 关于 691
- Label 组件
  - 包 696
  - 参数 694
  - 创建应用程序 695
  - 方法 697
  - 关于 693
  - 继承 696
  - 使用样式 695
  - 事件 699
  - 属性 698
  - 自定义 695
- List 组件
  - 包 711
  - 参数 705
  - 创建应用程序 706
  - 方法 712
  - 关于 703
  - 滚动行为 102
  - 继承 711
  - 设计 101
  - 使用外观 711
  - 使用样式 708
  - 事件 716
  - 属性 714
  - 自定义 708
- Loader 组件
  - 包 755
  - 参数 752
  - 创建应用程序 753
  - 方法 755
  - 关于 751
  - 继承 755
  - 使用外观 754
  - 使用样式 754
  - 事件 758
  - 属性 756
  - 自定义 754
- Log 类 1300
- MediaController 组件
  - 参数 780
  - 关于 774
- MediaDisplay 组件
  - 参数 779

- 关于 774
- MediaPlayer 组件
  - 参数 780
  - 关于 774
- Menu 组件
  - 菜单项类型 821
  - 参数 825
  - 初始化对象属性 824
  - 创建应用程序 825
  - 方法 833
  - 关于 817
  - 关于 XML 属性 820
  - 使用外观 832
  - 使用样式 829
  - 事件 836
  - 视图 819
  - 属性 835
  - 数据模型 819
  - 添加分层菜单 820
  - 通过 ActionScript 访问项目 823
  - 自定义 829
- MenuBar 组件
  - 参数 876
  - 创建应用程序 877
  - 方法 880
  - 关于 875
  - 类 880
  - 使用外观 879
  - 使用样式 878
  - 事件 883
  - 属性 882
  - 自定义 878
- MenuDataProvider 类
  - 方法 865
  - 关于 864
  - 事件 865
- multipleSimultaneousAllowed 参数 1332
- NumericStepper 组件
  - 参数 898
  - 创建应用程序 898
  - 方法 903
  - 关于 897
  - 使用外观 901
  - 使用样式 900
  - 事件 905
  - 属性 904
  - 自定义 899
- onFault callback function 1329
- PendingCall 类
  - 方法 1309
- 关于 1308
- 回调 1309
- 属性 1309
- PopUpManager 类 915
- ProgressBar 组件
  - 参数 920
  - 创建应用程序 921
  - 方法 926
  - 关于 919
  - 使用外观 925
  - 使用样式 924
  - 事件 928
  - 属性 927
  - 自定义 924
- RadioButton 组件
  - 参数 950
  - 创建应用程序 950
  - 方法 955
  - 关于 949
  - 使用外观 953
  - 使用样式 951
  - 事件 958
  - 属性 956
  - 自定义 951
- RDBMSResolver 组件
  - 参数 972
  - 方法 974
  - 关于 971
  - 事件 975
  - 属性 975
  - 通用工作流程 973
- RectBorder 类
  - 关于 985
  - 使用样式 986
- Screen 类
  - 方法 994
  - 关于 991
  - 加载外部内容 992
  - 事件 997
  - 属性 996
  - 引用屏幕 993
- ScrollPane 组件
  - 参数 1012
  - 创建应用程序 1013
  - 方法 1016
  - 关于 1011
  - 使用外观 1015
  - 使用样式 1014
  - 事件 1018
  - 属性 1017



- 自定义 1014
- SimpleButton 类 1039
  - 方法 1039
  - 关于 1039
  - 事件 1041
  - 属性 1040
- Slide 类 1047
  - 包 1049
  - 参数 1048
  - 方法 1050
  - 继承 1049
  - 示例 1048
  - 事件 1054
  - 属性 1051
- SOAPCall 类
  - 关于 1318
  - 属性 1318
- SOAPFault 对象 1328
- StyleManager 类
  - 方法 1079
  - 关于 1079
- suppressInvalidCalls 参数 1332
- SystemManager 类
  - 关于 1083
  - 属性 1083
- Tab 键顺序, 组件的 663
- TextArea 组件
  - 包 1090
  - 参数 1086
  - 创建应用程序 1087
  - 方法 1091
  - 关于 1085
  - 继承 1090
  - 使用外观 1090
  - 使用样式 1088
  - 事件 1093
  - 属性 1092
  - 自定义 1088
- TextInput 组件 1115
  - 参数 1116
  - 创建应用程序 1117
  - 方法 1120
  - 关于 1115
  - 类 1119
  - 使用 1116
  - 使用样式 1118
  - 事件 1122
  - 属性 1121
  - 自定义 1118

- TransferObject 接口
  - 方法 1135
  - 关于 1135
- TransitionManager 类
  - 参数 1140
  - 淡化过渡 1151
  - 方法 1140
  - 飞行过渡 1151
  - 光圈过渡 1152
  - 划入 / 划出过渡 1155
  - 基于过渡的类 1149
  - 挤压过渡 1154
  - 事件 1141
  - 属性 1141
  - 缩放过渡 1156
  - 像素溶解过渡 1153
  - 旋转过渡 1154
  - 照片过渡 1153
  - 遮帘过渡 1150
- Tree 组件
  - XML 格式设置 1166
  - 包 1177
  - 参数 1168
  - 创建应用程序 1168
  - 方法 1179
  - 继承 1177
  - 使用外观 1177
  - 使用样式 1173
  - 事件 1181
  - 属性 1179
  - 自定义 1173
- TreeDataProvider 接口
  - 方法 1157
  - 关于 1157
  - 属性 1158
- Tween 类
  - Accordion 组件 1211
  - ComboBox 组件 1212
  - DataGrid 组件 1213
  - 参数 1209
  - 方法 1207
  - 缓动类和方法 1209
  - 将缓动方法应用于组件 1210
  - 事件 1208
  - 属性 1208
- TypedValue 类
  - 关于 228
  - 属性 228
- UI 组件 30

## UIComponent 类

- 包 1233
- 方法 1234
- 关于 1233
- 继承 1233
- 事件 1235
- 属性 1235

## UIEventDispatcher 类

- 方法 1245
- 关于 1245
- 事件 1246

## UIObject 类 1251

- 包 1251
- 方法 1207, 1252
- 关于 1207, 1251
- 继承 1251
- 事件 1208, 1253
- 属性 1208, 1252

## UIScrollBar 组件

- 包 1284
- 参数 1280
- 创建应用程序 1280
- 方法 1285
- 关于 1279
- 继承 1284
- 使用外观 1283
- 使用样式 1282
- 事件 1287
- 属性 1286
- 自定义 1282

## VideoError 类

- 属性 642
- 已定义 642

## VideoPlayer 类 649

- 方法 650
- 事件 653
- 属性 650

## Web 服务类

- Log 类 1300
- PendingCall 类 1308
- SOAPCall 类 1318
- WebService 类 1320
- 关于 1299
- 在运行时使用 1300

## WebService 类

- 安全性 1324
- 方法 1321
- 关于 1320
- 回调 1321
- 支持的类型 1322

## WebServiceConnector 组件

- 参数 1332
- 方法 1333
- 关于 1331
- 事件 1334
- 属性 1334
- 通用工作流程 1332

## Window 组件

- 包 1352
- 参数 1348
- 创建应用程序 1349
- 方法 1353
- 关于 1347
- 继承 1352
- 使用外观 1351
- 使用样式 1350
- 事件 1356
- 属性 1355
- 自定义 1350

## WSDLURL 参数 1332

## XML

- 菜单项的属性 820
- 架构类型 1322
- 设置 Tree 组件的格式 1166

## XMLConnector 组件

- 参数 1369
- 方法 1371
- 关于 1369
- 架构和 1369
- 事件 1372
- 属性 1371
- 通用工作流程 1370

## XUpdateResolver 组件

- 参数 1386
- 关于 1385
- 事件 1388
- 属性 1388
- 通用工作流程 1387

## A

安全性, 和 WebService 类 1324

## B

边框。请参见 RectBorder 类  
表。参见 DataGrid 组件

## C

操作参数 1332

## F

分隔符菜单项 821

## G

管理器组件 32

## H

缓动类和方法, Tween 类 1209

## J

激活器, 菜单 875

加载外部内容 992

架构类型, XML 1322

接口

Collection 135

Delta 433

DeltaPacket 443

Iterator 691

TransferObject 1135

TreeDataProvider 1157

## L

类

Accordion 45

Alert 67

Binding 192

CheckBox 124

ComboBox 152

ComponentMixins 210

CustomFormatter 196

CustomValidator 200

DataGrid 243

DataGridColumn 280

DataHolder 293

DataSet 313

DataType 216

DateChooser 386

DateField 406

Delegate 425

DeltaItem 427

DepthManager 449

EndPoint 204

EventDispatcher 461

FLVPlayback 497

FocusManager 663

Form 677

Label 696

List 711

Loader 755

Log 1300

Media 782

Menu 817

MenuBar 880

MenuDataProvider 864

NumericStepper 902

PendingCall 1308

PopUpManager 915

ProgressBar 926

RadioButton 954

RDBMSResolver 974

RectBorder 985

Screen 991

ScrollPane 1015

SimpleButton 1039

Slide 1049

SOAPCall 1318

StyleManager 1079

SystemManager 1083

TextArea 1090

TextInput 1119

Tree 1177

TypedValue 228

UIEventDispatcher 1245

UIScrollBar 1284

Web 服务 1299

WebService 1320

WebServiceConnector 1333

XMLConnector 1371

XUpdateResolver 1385

按钮 94

窗口 1352

数据绑定 191

类型。请参见数据类型

列, DataGridColumn 类 280

另请参见 各个组件名称图像

## M

媒体组件

MediaController 组件 769

- MediaDisplay 组件 769
- MediaPlayer 组件 769
  - 包 782
  - 参数 779
  - 创建应用程序 781
  - 方法 783
  - 关于 32, 769
  - 继承 782
  - 设计 771
  - 使用外观 782
  - 使用样式 782
  - 事件 785
  - 属性 784
  - 行为 777
  - 自定义 782
  - “组件”检查器 776

## P

- 屏幕组件 32

## S

- 身份验证和 WebService 类 1324
- 事件
  - 事件对象 461
- 事件对象 461
- 视频回放 777
- 视图, Menu 组件 819
- 数据绑定类
  - Binding 类 192
  - ComponentMixins 类 210
  - CustomValidator 类 200
  - DataType 类 216
  - EndPoint 类 204
  - TypedValue 类 228
- 包 192
- 关于 191
- 在运行时使用 191
- 数据集。参见 DataSet 组件
- 数据类型, 受 Web 服务类的支持 1322
- 数据模型
  - DataGrid 组件 234
  - Menu 组件 819
- 数据组件 31

## T

- 提示点, 使用 474

## W

- 外观, 自定义 FLVPlatyback 484

## X

- 行为和视频回放 777

## Y

- 样式
  - 986
  - RectBorder 类 986
- 用户界面组件 30

## Z

- 组件, 将缓动方法应用于 1210
- “组件”检查器, 媒体组件 776
- 组件类别
  - UI 组件 30
  - 管理器 32
  - 媒体 32
  - 屏幕 32
  - 其它 33
  - 数据 31