



使用组件

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，并且可能已在美国或其它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得到了 Nellymoser, Inc. (www.nellymoser.com) 的许可。



Sorenson™ Spark™ 视频压缩和解压缩技术由 Sorenson Media, Inc. 授权。

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

Macromedia Flash 8 视频受 On2 TrueMotion 视频技术的支持。© 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

执行编辑：Rosana Francescato

主编：Lisa Stanziano

编辑：Mary Ferguson、Mary Kraemer、Lisa Stanziano

制作管理：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 QA 小组全体成员的帮助。

第一版：2005 年 9 月

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

目 录

简介	5
目标读者	6
系统要求	6
关于本文档	6
印刷惯例	7
本手册中使用的术语	7
其它资源	7
 第 1 章：关于组件	9
安装组件	10
组件文件的存储位置	12
修改组件文件	12
使用组件的优点	13
组件类别	14
关于第 2 版组件体系结构	15
第 2 版组件的功能	16
关于编译剪辑和 SWC 文件	17
辅助功能和组件	18
 第 2 章：使用组件创建应用程序（仅限 Flash Professional）	19
关于“Fix Your Mistake”教程	19
构建主页	21
绑定数据组件以显示礼品方案	26
显示礼品详细信息	30
创建结帐屏幕	34
测试应用程序	41
查看完成后的应用程序	42
 第 3 章：使用组件	43
“组件”面板	44
向 Flash 文档中添加组件	44
“库”面板中的组件	47
设置组件参数	48
调整组件大小	49

从 Flash 文档删除组件.....	50
使用代码提示.....	50
创建自定义焦点导航.....	51
管理文档中的组件深度.....	52
“实时预览”中的组件.....	52
配合使用预加载器和组件.....	53
关于加载组件.....	54
将第 1 版组件升级到第 2 版的体系结构.....	54
第 4 章：处理组件事件.....	55
使用侦听器处理事件.....	56
委托事件.....	63
关于事件对象.....	66
使用 on() 事件处理函数.....	67
第 5 章：自定义组件.....	69
使用样式自定义组件的颜色和文本.....	69
关于设置组件外观.....	81
关于主题.....	91
通过组合外观与样式设置来自定义组件.....	100
第 6 章：创建组件.....	105
组件源文件.....	106
组件结构概述.....	106
构建第一个组件.....	107
选择父类.....	116
创建组件影片剪辑.....	118
创建 ActionScript 类文件.....	123
在组件内组合现有组件.....	148
导出和分发组件.....	157
组件开发的最后一些步骤.....	160
第 7 章：集合属性.....	163
定义集合属性.....	164
简单集合示例.....	165
为集合项目定义类.....	167
以编程方式访问集合信息.....	167
将具有集合的组件导出为 SWC 文件.....	169
使用具有集合属性的组件.....	170
索引.....	171

简介

Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 是标准的创作工具，使用它们制作出的 Web 内容极富感染力。组件是提供此类感受的“丰富 Internet 应用程序”的构建块。组件就是影片剪辑，其中的参数由您在 Macromedia Flash 中创作时进行设置，其中的 **ActionScript** 方法、属性和事件可供您在运行时自定义组件。设计这些组件的目的是为了让开发人员重复使用和共享代码，封装复杂功能，使设计人员无需编写 **ActionScript** 就能够使用和自定义这些功能。

组件是基于 **Macromedia Component Architecture** 第 2 版设计的，因此，您可以方便而快速地构建功能强大且具有一致外观和行为的应用程序。本书介绍如何使用第 2 版组件构建应用程序。相关的《组件语言参考》中介绍了每种组件的应用程序编程接口 (API)。它包括使用 **Flash** 第 2 版组件的用法方案和过程范例，以及按字母排序的组件 API 说明。

您可以使用 **Macromedia** 创建的组件，下载其它开发人员创建的组件，还可以创建自己的组件。

本章包含以下各节：

目标读者	6
系统要求	6
关于本文档	6
印刷惯例	7
本手册中使用的术语	7
其它资源	7

目标读者

本书的目标读者是要构建 **Flash** 应用程序并希望使用组件加快开发速度的开发人员。您应当已经熟悉如何开发 **Flash** 应用程序和编写 **ActionScript**。

如果对编写 **ActionScript** 还不够熟练，您可以向文档添加组件，在“属性”检查器或“组件”检查器中设置其参数，然后使用“行为”面板处理其事件。例如，您无需编写任何 **ActionScript** 代码，就可以将“转到 Web 页”行为附加到一个 **Button** 组件，用户点击此按钮时会在 Web 浏览器中打开一个 URL。

如果希望创建功能更加强大的应用程序，则可通过动态方式创建组件，使用 **ActionScript** 在运行时设置属性和调用方法，还可使用侦听器事件模型来处理事件。

有关详细信息，请参阅第 43 页的第 3 章“使用组件”。

系统要求

除 **Flash** 外，**Macromedia** 组件没有任何其它系统要求。

使用第 2 版组件的 **SWF** 文件必须用 **Flash Player 6 (6.0.79.0)** 或更高版本查看，而且必须以 **ActionScript 2.0** 发布（您可以通过“文件”>“发布设置”下的“**Flash**”选项卡进行设置）。

关于本文档

本文档详细说明了如何使用组件开发 **Flash** 应用程序。本文档的目标读者需已掌握 **Macromedia Flash** 和 **ActionScript** 的一般性知识。**Flash** 和相关产品的专用文档将单独提供。

本文档以 PDF 格式和在线帮助的形式提供。要查看在线帮助，请启动 **Flash**，然后选择“帮助”>“使用组件”。

有关 **Macromedia Flash** 的信息，请参见以下文档：

- 《使用 **Flash**》
- 《学习 **Flash** 中的 **ActionScript 2.0**》
- 《**ActionScript 2.0** 语言参考》
- 《组件语言参考》

印刷惯例

本书使用以下印刷惯例：

- **代码字体 (Code font)** 表示 **ActionScript** 代码，其中包括方法和属性名称。
- *斜体代码字体* 表示应被替换的代码项目（例如，一个 **ActionScript** 参数）。
- **粗体字体** 表示您输入的值。

本手册中使用的术语

在本手册中使用以下术语：

在运行时 代码在 **Flash Player** 中运行时。

在创作时 在 **Flash** 创作环境中工作时。

其它资源

有关 **Flash** 的最新信息，以及专家级用户的建议、高级主题、示例、提示和其它更新，请访问定期更新的 **Macromedia DevNet Web** 站点，网址为 www.macromedia.com/cn/devnet。请经常查看这一 **Web** 站点，了解有关 **Flash** 以及如何充分利用这一程序的最新消息。

有关 **TechNote**、文档更新以及指向“**Flash** 社区”中其它资源的链接，请访问 **Macromedia Flash** 支持中心，网址为：www.macromedia.com/go/flash_support_cn。

有关 **ActionScript** 术语、语法和用法的详细信息，请参见《学习 **Flash** 中的 **ActionScript 2.0**》和《**ActionScript 2.0** 语言参考》。

有关使用组件的简介，请参见 **Macromedia On Demand Seminar**（**Macromedia** 按需研讨会）的 **Using UI Components**（使用 **UI** 组件），网址：www.macromedia.com/macromedia/events/online/ondemand/index.html。

Macromedia Flash 组件是带参数的影片剪辑，您可以修改它们的外观和行为。组件既可以是简单的用户界面控件（例如，单选按钮或复选框），也可以包含内容（例如，滚动窗格）；组件还可以是不可视的（例如，**FocusManager**，它允许您控制应用程序中接收焦点的对象）。

使用组件，您可以构建复杂的 **Macromedia Flash** 应用程序，即使您对 **ActionScript** 没有深入的理解。您不必创建自定义按钮、组合框和列表，将这些组件从“组件”面板拖到应用程序中即可为应用程序添加功能。您还可以方便地自定义组件的外观和直观感受，从而满足您自己的设计需求。

组件是基于 **Macromedia Component Architecture** 第 2 版构建的，这使您可以轻松而快速地构建功能强大且具有一致外观和行为的应用程序。第 2 版的体系结构包括所有组件基于的类、使您可以自定义组件外观的样式和外观机制、广播器 / 侦听器事件模式、深度和焦点管理、辅助功能实施等等。

提醒

发布第 2 版组件时，您必须将发布设置设为以 **ActionScript 2.0** 发布（位于“文件”>“发布设置”下的“Flash”选项卡）。如果用 **ActionScript 1.0** 发布第 2 版组件，这些组件将无法正常运行。

每个组件都有预定义参数，您可以在 **Flash** 中创作时来设置这些参数。每个组件还有一组独特的 **ActionScript** 方法、属性和事件，它们也称为 **API**（应用程序编程接口），使您可以在运行时设置参数和其它选项。

有关 **Flash Basic 8** 和 **Flash Professional 8** 包括的组件完整列表，请参见第 10 页的“安装组件”。您还可以从 **Macromedia Exchange**（网址为 www.macromedia.com/cn/exchange/）下载由 **Flash** 社区成员构建的组件。

本章包含以下各节：

安装组件	10
组件文件的存储位置	12
修改组件文件	12
使用组件的优点	13
组件类别	14
关于第 2 版组件体系结构	15

第 2 版组件的功能.....	16
关于编译剪辑和 SWC 文件.....	17
辅助功能和组件	18

安装组件

首次启动 Flash 时，系统中就已安装了一组 Macromedia 组件。您可以在“组件”面板上查看它们。

Flash Basic 8 包含以下组件：

- Button 组件
- CheckBox 组件
- ComboBox 组件
- Label 组件
- List 组件
- Loader 组件
- NumericStepper 组件
- ProgressBar 组件
- RadioButton 组件
- ScrollPane 组件
- TextArea 组件
- TextInput 组件
- Window 组件

Flash Professional 8 包括 Flash Basic 8 组件及以下其它组件和类：

- Accordion 组件（仅限 Flash Professional）
- Alert 组件（仅限 Flash Professional）
- 数据绑定类（仅限 Flash Professional）
- DateField 组件（仅限 Flash Professional）
- DataGrid 组件（仅限 Flash Professional）
- DataHolder 组件（仅限 Flash Professional）
- DataSet 组件（仅限 Flash Professional）
- DateChooser 组件（仅限 Flash Professional）
- FLVPlayback 组件（仅限 Flash Professional）
- Form 类（仅限 Flash Professional）
- 媒体组件（仅限 Flash Professional）

- Menu 组件（仅限 Flash Professional）
- MenuBar 组件（仅限 Flash Professional）
- RDBMSResolver 组件（仅限 Flash Professional）
- Screen 类（仅限 Flash Professional）
- Slide 类（仅限 Flash Professional）
- Tree 组件（仅限 Flash Professional）
- WebServiceConnector 组件（仅限 Flash Professional）
- XMLConnector 组件（仅限 Flash Professional）
- XUpdateResolver 组件（仅限 Flash Professional）

若要查看 Flash Basic 8 或 Flash Professional 8 组件，请执行以下操作：

1. 启动 Flash。
2. 如果“组件”面板没有打开，请选择“窗口”>“组件”打开它。
3. 选择“用户界面”展开树，然后查看已安装的组件。

您也可以从 Macromedia Exchange 下载组件，网址为 www.macromedia.com/cn/exchange。要安装从 Exchange 下载的组件，请下载并安装 Macromedia Extension Manager，网址为 http://www.macromedia.com/cn/exchange/em_download/

所有组件都可以在 Flash 的“组件”面板上显示。要在 Windows 或 Macintosh 计算机上安装组件，请遵循以下步骤。

在基于 Windows 的计算机上或 Macintosh 计算机上安装组件：

1. 退出 Flash。
2. 将包含组件的 SWC 或 FLA 文件放在硬盘上的以下文件夹中：
 - 在 Windows 中：C:\Program Files\Macromedia\Flex 8\语言\Configuration\Components
 - 在 Macintosh 上：Macintosh HD/Applications/Macromedia Flex 8/Configuration/Components (Macintosh)
3. 启动 Flash。
4. 如果“组件”面板尚未打开，请选择“窗口”>“组件”，以在“组件”面板中查看组件。

组件文件的存储位置

Flash 组件存储在应用程序级别的 **Configuration** 文件夹中。



有关这些文件夹的信息，请参阅《Flash 入门》中的“随 Flash 安装的配置文件夹”。

组件安装在以下位置：

- Windows 2000 或 Windows XP: C:\Program Files\Macromedia\Flash 8\ 语言 \Configuration\Components
- Mac OS X: Macintosh HD/Applications/Macromedia Flash 8/Configuration/Components

修改组件文件

组件的源 ActionScript 文件位于以下位置中：

- Windows 2000 或 Windows XP: C:\Program Files\Macromedia\Flash 8\ 语言 \First Run\Classes\mx
- Mac OS X: Macintosh HD/Applications/Macromedia Flash 8/First Run/Classes/mx

Flash 第一次启动时，**First Run** 目录中的文件会复制到您的 **Documents and Settings** 路径。
Documents and Settings 路径为：

- Windows 2000 或 Windows XP: C:\Documents and Settings\用户名\Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\Classes\mx
- Mac OS X: 用户名 /Library/Application Support/Macromedia/Flash 8/ 语言 / Configuration/Classes/mx

Flash 启动时，如果 **Document and Settings** 路径中缺少某个文件，Flash 会将该文件从 **First Run** 目录中复制到 **Documents and Settings** 路径。

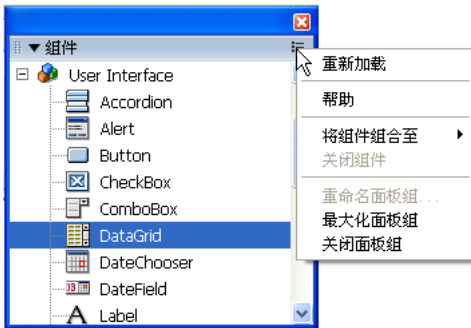


如果要修改源 ActionScript 文件，请修改 **Documents and Settings** 路径中的源 ActionScript 文件。如果您所做的任一项修改破坏了某个组件，Flash 会在您关闭并重新启动它时恢复该组件的原始功能，方法是通过复制 **First Run** 目录中的可正常运行的文件。但是，如果修改了 **First Run** 目录中的文件，并且该修改破坏了组件，您可能需要重新安装 Flash 才能将源文件恢复为可正常运行的文件。

如果您已经添加了组件，需要刷新“组件”面板。

刷新“组件”面板的内容：

- 从“组件”面板菜单中选择“重新加载”。



从“组件”面板中删除组件：

- 从配置文件夹中删除 MXP 或 FLA 文件。

使用组件的优点

组件使您可以将应用程序的设计过程和编码过程分开。通过组件，您还可以重复利用代码，您可以重复利用自己创建的组件中的代码，也可以通过下载并安装其他开发人员创建的组件来重复利用别人的代码。

通过使用组件，代码编写者可以创建设计人员在应用程序中能用到的功能。开发人员可以将常用功能封装在组件中，设计人员可以自定义组件的外观和行为，方法是在“属性”检查器或“组件”检查器中更改参数。

Flash 开发人员可以使用 www.macromedia.com/go/exchange 上的 Macromedia Exchange 来交换组件。通过使用组件，您不再需要从头构建复杂的 Web 应用程序中的每个元素。您可以查找需要的组件，然后将它们一起放入 Flash 文档来创建新应用程序。

基于第 2 版体系结构的组件共享诸如样式、事件处理、外观、焦点管理和深度管理等核心功能。将首个第 2 版组件添加到应用程序中时，提供某项核心功能的组件会令文档大小增加约 25K。在添加其它组件时，系统会再次利用这个 25K 的空间以部署新增组件，因此，文档增加的大小会比您预期的小。有关升级组件的信息，请参阅第 54 页的“将第 1 版组件升级到第 2 版的体系结构”。

组件类别

Flash 提供的组件分为以下 5 个类别（其 ActionScript 源文件的位置与其类别大致对应，也列在括号中）：

- 数据组件 (mx.data.*)

利用数据组件可加载和处理数据源的信息；WebServiceConnector 和 XMLConnector 组件都是数据组件。



数据组件的源文件不随 Flash 一起安装。但是，安装了一些支持 ActionScript 文件。

- FLVPlayback 组件 (mx.video.FLVPlayback)

FLVPlayback 组件使您可以轻松将视频播放器包括在 Flash 应用程序中，以便通过 HTTP 从 Flash 视频流服务 (FVSS) 或从 Flash Communication Server (FCS) 播放渐进式流视频。

- 媒体组件 (mx.controls.*)

利用媒体组件可播放和控制媒体流；MediaController、MediaPlayback 和 MediaDisplay 都是媒体组件。

- 用户界面组件 (mx.controls.*)

利用用户界面组件（通常称为“UI 组件”）可与应用程序进行交互；例如，RadioButton、CheckBox 和 TextInput 组件都是用户界面控件。

- 管理器 (mx.managers.*)

管理器是不可见的组件，使用此类组件可以在应用程序中管理诸如焦点或深度等功能；FocusManager、DepthManager、PopUpManager、StyleManager 和 SystemManager 都是管理器组件。

- 屏幕组件 (mx.screens.*)

屏幕类组件包括 ActionScript 类，使用此类组件可以控制 Flash 中的表单和滑块。

有关完整的组件列表，请参阅《组件语言参考》。

关于第 2 版组件体系结构

您可以使用“属性”检查器或“组件”检查器来更改组件参数，以使用组件的基本功能。然而，如果要在更大程度上控制组件，您需要使用组件的 API，并且要了解组件的一些构建方式。

Flash 组件是使用 Macromedia Component Architecture 第 2 版构建的。Flash Player 6 (6.0.79.0) 及更高版本和 ActionScript 2.0 支持第 2 版组件。这些组件不是总与用第 1 版体系结构构建的组件（所有在 Flash MX 2004 以前发布的组件）兼容。而且，Flash Player 7 不支持原始的第 1 版组件。有关详细信息，请参阅第 54 页的“将第 1 版组件升级到第 2 版的体系结构”。

提醒

Flash MX UI 组件已更新为能与 Flash Player 7 或更高版本协同使用。这些更新后的组件仍基于第 1 版的体系结构。您可以从 www.macromedia.com/go/v1_components 上的 Macromedia Flash Exchange 下载它们。

第 2 版组件作为编译剪辑 (SWC) 元件包含在“组件”面板中。编译剪辑是其代码已经过编译的组件影片剪辑。编译剪辑无法编辑，但您可以在“属性”检查器和“组件”检查器中更改它们的参数，就像更改其它任何组件的参数一样。有关详细信息，请参阅第 17 页的“关于编译剪辑和 SWC 文件”。

第 2 版组件是用 ActionScript 2.0 编写的。每个组件都是一个类，而每个类都属于一个 ActionScript 包。例如，一个单选按钮组件是 RadioButton 类的一个实例，该类的包名称为 mx.controls。有关包的详细信息，请参阅《学习 Flash 中的 ActionScript 2.0》中的“关于包”。

用 Macromedia Component Architecture 的第 2 版构建的大多数 UI 组件都是 UIObject 和 UIComponent 类的子类，并且继承了这些类的所有属性、方法和事件。许多组件也是其它组件的子类。每个组件的继承路径都在《组件语言参考》中该组件的条目里指明。

提醒

类层次结构的 FlashPaper 文件版本还可在以下安装位置找到：Flash 8\Samples and Tutorials\Samples\Components\arch_diagram.swf。

所有组件也使用相同的事件模型、基于 CSS 的样式及内置的主题和外观机制。有关样式和外观设置的详细信息，请参阅第 69 页的第 5 章“自定义组件”。有关事件处理的详细信息，请参阅第 43 页的第 3 章“使用组件”。

有关第 2 版组件体系结构的详细说明，请参阅第 105 页的第 6 章“创建组件”。

第 2 版组件的功能

本部分从使用组件构建 Flash 应用程序的开发人员的角度出发，概括地介绍了第 2 版组件的功能（与第 1 版组件比照）。有关第 1 版和第 2 版体系结构在构建组件方面的差异的详细信息，请参阅第 105 页的第 6 章“创建组件”。

“组件”检查器：在使用 Macromedia Flash 和 Macromedia Dreamweaver 进行创作时可用 come 更改组件参数。（请参阅第 48 页的“设置组件参数”。）

侦听器事件模型：允许侦听器处理事件。（请参阅第 55 页的第 4 章“处理组件事件”。）Flash 在“属性”检查器中没有 clickHandler 参数，而 Flash MX 则有；您必须编写 ActionScript 代码来处理事件。

外观属性：可以用于在运行时加载各种外观（例如，向上和向下箭头，或复选框的选中标记）。（请参阅第 81 页的“关于设置组件外观”。）

基于 CSS 的样式：可用于创建具有一致外观和直观感受的应用程序。（请参阅第 69 页的“使用样式自定义组件的颜色和文本”。）

主题使您可以将预先设计的外观从库中拖到一组组件上。（请参阅第 91 页的“关于主题”。）

“光晕”主题是第 2 版组件默认使用的主题。（请参阅第 91 页的“关于主题”。）

管理器类：提供了一种在应用程序中处理焦点和深度的简便方法。（请参阅第 51 页的“创建自定义焦点导航”和第 52 页的“管理文档中的组件深度”。）

基类 UIObject 和 UIComponent：向其扩展组件提供核心方法、属性和事件。（请参阅《组件语言参考》中的“UIComponent 类”和“UIObject 类”。）

打包为 SWC 文件：可以用来编写便于分发和可隐藏的代码。请参阅第 105 页的第 6 章“创建组件”。

内置数据绑定：可通过“组件”检查器获取。有关详细信息，请参阅《使用 Flash》中的“数据集成（仅限 Flash Professional）”。

便于扩展的类层次体系结构：使用 ActionScript 2.0，可以创建唯一的命名空间，按需要导入类，并且可以方便地创建子类来扩展组件。请参阅第 105 页的第 6 章“创建组件”和《ActionScript 2.0 语言参考》。

提醒

Flash 8 有一些不受第 2 版组件支持的功能，其中包括 9 切片（有时称为“缩放 -9”）、FlashType 和位图缓存。

关于编译剪辑和 SWC 文件

编译剪辑 是由预编译的 **Flash** 元件和 **ActionScript** 代码组成的包。使用它可避免重新编译那些不再更改的元件和代码。影片剪辑也可以在 **Flash** 中被“编译”，然后转换为编译剪辑。例如，具有大量不经常更改的 **ActionScript** 代码的影片剪辑可以转换为编译剪辑。编译剪辑的行为方式与它所编译自的影片剪辑相似，但与常规影片剪辑相比，编译剪辑的显示速度和发布速度要快得多。编译剪辑不能被编辑，但其属性可在“属性”检查器和“组件”检查器内显示。

Flash 包含的组件不是 **FLA** 文件，它们是已打包到编译剪辑 (**SWC**) 文件中的编译剪辑。**SWC** 是用于分发组件的 **Macromedia** 文件格式：它包含编译剪辑、组件的 **ActionScript** 类文件以及描述该组件的其它文件。有关 **SWC** 文件的详细信息，请参阅第 157 页的“[导出和分发组件](#)”。

将 **SWC** 文件放在 **First Run/Components** 文件夹中后，该组件会出现在“组件”面板中。在从“组件”面板将组件添加到舞台上时，就会将编译剪辑元件添加到库中。

编译影片剪辑：

- 右击 (Windows) 或按住 **Control** 键单击 (Macintosh) “库”面板中的影片剪辑，然后选择“转换为编译剪辑”。

导出 SWC 文件：

- 选择“库”面板中的影片剪辑并右击 (Windows) 或按住 **Control** 键单击 (Macintosh)，然后选择“导出 **SWC** 文件”。



Flash Basic 8 和 Flash Professional 8 继续支持 **FLA** 组件。

辅助功能和组件

对 Web 内容要具有辅助功能（即让残障人士也可以使用 Web 内容）的要求在不断增长。使用屏幕阅读程序软件可以使视觉障碍者能够比较容易地感知 Flash 应用程序中的可视内容。这种软件提供了将屏幕中元素的描述朗读出来的语音效果。

创作者在创建组件时，能够编写可实现在组件与屏幕阅读程序间通讯的 **ActionScript**。当开发人员使用该组件在 **Flash** 中构建应用程序时，该开发人员可以使用“辅助功能”面板来配置每个组件实例。

由 **Macromedia** 构建的大多数组件都针对辅助功能进行了设计。要查明某个组件是否具备辅助功能，请参阅《组件语言参考》中该组件的条目。在 **Flash** 中构建应用程序时，您需要为每个组件添加一行代码 (`mx.accessibility.ComponentNameAccImpl.enableAccessibility();`)，并在“辅助功能”面板中设置辅助功能参数。组件辅助功能的运行方式与所有 **Flash** 影片剪辑辅助功能的运行方式相同。

由 **Macromedia** 构建的大多数组件也能通过键盘来导航。《组件语言参考》中每个组件的条目都指明您是否可用键盘来控制该组件。

使用组件创建应用程序（仅限 Flash Professional）

组件是指创建 **Macromedia Flash** 应用程序时您可以使用的预先构建的 **Flash** 元素。组件包括用户界面控件、数据访问和连接性机制以及与介质相关的元素。在您构建 **Flash** 应用程序时，组件可以为您提供您所需要的元素和行为，使您不必从零开始创建应用程序。

本章中包含的教程可向您展示如何利用 **Macromedia Flash Professional 8** 中提供的组件构建 **Flash** 应用程序。您将学习如何在 **Flash** 创作环境中使用这些组件，以及如何使这些组件与 **ActionScript** 代码交互。

关于 “Fix Your Mistake” 教程

本教程可指导您完成为 “Fix Your Mistake” 礼品服务创建一个基本的在线购物应用程序的全过程。该服务可帮助用户为该用户所冒犯的人选择一种适当的礼品以弥补过失。应用程序需要过滤出符合用户过失严重程度的备选礼品列表。在这个列表中，用户可以向购物车中添加项目，然后进入结帐页面以提供开单、发货和信用卡信息。

本章包含下列各节：

关于 “Fix Your Mistake” 教程	19
构建主页	21
绑定数据组件以显示礼品方案	26
显示礼品详细信息	30
创建结帐屏幕	34
测试应用程序	41
查看完成后的应用程序	42

应用程序要使用 **ComboBox**、**DataGrid**、**TextArea** 和 **Button** 组件以及其它组件来创建应用程序的界面。界面主页的外观如下所示：



应用程序要使用 **ActionScript WebService** 类动态连接到 **Web** 服务以检索显示在组合框中的过失列表 (**problems.xml**)。还要使用 **ActionScript** 来处理用户与应用程序的交互。

应用程序将使用数据组件将界面连接到其它数据源。对于礼品列表，使用 **XMLConnector** 组件连接到 **XML** 数据文件 (**products.xml**)，并使用 **DataSet** 组件来过滤数据，然后将数据呈现到数据网格。

教程要求对 **Flash** 创作环境具有一定的知识，并具有使用 **ActionScript** 的经验。在创作环境方面，您应该具有一些面板、工具、时间轴和库的使用经验。本教程提供了创建范例应用程序所需的所有 **ActionScript**。不过，为了理解脚本概念并创建您自己的应用程序，您还需要有编写 **ActionScript** 的经验。

若要查看已完成应用程序的工作版本，请参阅第 42 页的“查看完成后的应用程序”。

不要忘了该范例应用程序仅用于示范目的，因此不会像实际的应用程序那样尽善尽美。

构建主页

执行以下步骤向框架页面添加组件来创建应用程序的主页。然后添加 **ActionScript** 代码以自定义这些组件，导入允许您操作应用程序组件的 **ActionScript** 类，并访问 **Web** 服务以便用过失列表填充组合框。通过将组合框的 `dataProvider` 属性设置为接收来自 **Web** 服务的结果，代码即会填充组合框。

1. 打开 `first_app_start.fla` 文件，您可以在以下位置之一找到该文件：
 - 在 Windows 中：安装驱动器：\Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\Components\ComponentsApplication
 - 在 Macintosh 上：Macintosh HD/Applications/Macromedia Flex 8/Samples and Tutorials/Samples/Components/ComponentsApplication

该文件中包含一个起始页，其外观如下所示：



`start_app.fla` 文件包含三个图层：具有黑色背景图像和文本标题的 **Background** 图层，具有应用程序各部分文本标签的 **Text** 图层，以及具有第一帧（**Home**）和第十帧（**Checkout**）标签的 **Labels** 图层。

2. 选择“文件” > “另存为”。重命名该文件并将其保存到硬盘上。
3. 在时间轴上选择 **Labels** 图层，然后单击“添加图层”按钮，在该图层的上面添加一个新图层。将新图层命名为 **Form**。此图层上将放置组件实例。

4. 确保选择了 Form 图层。在“组件”面板（“窗口” > “组件”）中，找到用户界面树中的 ComboBox 组件。将 ComboBox 的一个实例拖动到舞台上。将其放置在 What Did You Do? 文本下面。在“属性”检查器（“窗口” > “属性” > “属性”）中，输入实例名称 **problems_cb**。输入宽度 **400**（单位为像素）。再输入 x 坐标的位置 **76.0** 和 y 坐标的位置 **82.0**。

提醒

ComboBox 组件元件添加到了库（“窗口” > “库”）中。将组件的实例拖动到舞台上时，该组件的编译剪辑元件即被添加到了库中。和 Flash 中的所有元件一样，通过将库元件拖动到舞台上可以创建组件的更多实例。

5. 从“组件”面板的用户界面树中，将 DataGrid 组件的一个实例拖动到舞台上。将其放置在 Gift Ideas 文本的下面。输入实例名称 **products_dg**。输入宽度 **400**（单位为像素）和高度 **130**。再输入 x 坐标的位置 **76.0** 和 y 坐标的位置 **128.0**。
6. 从“组件”面板的数据树中，将 DataSet 组件的一个实例拖动到舞台的一侧。（DataSet 组件在运行时不会出现在应用程序中。DataSet 图标只是在 Flash 创作环境中使用的占位符。）输入实例名称 **products_ds**。

从“组件”面板的数据树中，将 XMLConnector 组件的一个实例拖动到舞台的一侧。（与 DataSet 组件一样，XMLConnector 组件在运行时也不会出现在应用程序中。）输入实例名称 **products_xmlcon**。单击“属性”检查器中的“参数”选项卡，然后为 URL 属性输入 www.flash-mx.com/mm/firstapp/products.xml。单击 direction 属性的值以激活组合框，单击向下箭头，然后从列表中选择 receive。

提醒

也可以使用“组件”检查器（“窗口” > “组件检查器”）设置组件参数。“属性”检查器的“参数”选项卡与“组件”检查器的“参数”选项卡工作方式相同。

URL 指定一个外部 XML 文件，其中包含显示在应用程序 Gift Ideas 部分中的产品的有关数据。在本教程的后面部分，将使用数据绑定将 XMLConnector、DataSet 和 DataGrid 组件绑定在一起；DataSet 组件将过滤来自外部 XML 文件的数据，而 DataGrid 组件则显示数据。

7. 从“组件”面板的用户界面树中，将 Button 组件的一个实例拖动到舞台上。将其放置在舞台的右下角。输入实例名称 **checkout_button**。单击“参数”选项卡，然后为 label 属性输入 **Checkout**。对于 x 和 y 坐标，分别输入 **560.3** 和 **386.0**。

导入组件类

每一个组件都与一个定义其方法和属性的 **ActionScript** 类文件关联。在教程的这一部分中，将添加 **ActionScript** 代码以导入与应用程序的组件相关联的类。对于其中一些组件，已在舞台中添加了实例。在本教程的后面部分，将为其它组件添加 **ActionScript**，以便动态创建实例。

import 语句创建对类名称的引用，并使编写组件的 **ActionScript** 更易于进行。**import** 语句引用类时只需使用类名称，而不需使用包含包名称的完整名称。例如，在用 **import** 语句创建了对 **ComboBox** 类文件的引用后，可以使用语法 `instanceName:ComboBox` 而不是 `instanceName:mx.controls.ComboBox` 来引用组合框实例。

包是驻留在指定的类路径目录下、包含类文件的目录。可使用通配符来创建对包中所有类的引用：例如，语法 `mx.controls.*` 创建对控件包中所有类的引用。（使用通配符创建对包的引用时，未使用的类将在编译应用程序时从应用程序中删除，因此不会增加任何额外的大小。）

对于此教程中的应用程序，需要使用以下包和类：

UI 组件控件包 此包中包含用户界面控件组件类，包括 **ComboBox**、**DataGrid**、**Loader**、**TextInput**、**Label**、**NumericStepper**、**Button** 和 **CheckBox**。

UI 组件容器包 此包中包含用户界面容器组件类，包括 **Accordion**、**ScrollPane** 和 **Window**。与控件包一样，您也可以通过使用通配符来创建对此包的引用。

DataGridColumn 类 使用这个类可向 **DataGrid** 实例添加列，并控制其外观。

WebService 类 这个类使用一个问题或过失的列表来填充 **ComboBox** 实例。对于这个类，还需要从“类”公用库中导入 **WebServiceClasses** 项目。该项目包含编译剪辑 (SWC) 文件，您需要这些编译剪辑来编译和生成应用程序的 SWF 文件。

Cart 类 本教程提供的自定义类，**Cart** 类定义稍后将创建的购物车的功能。（要检查 **Cart** 类文件中的代码，请打开 `cart.as` 文件，该文件与应用程序的 FLA 和 SWF 文件一起位于 `component_application` 文件夹中）。

为了导入这些类，您需要创建一个 **Actions** 图层，并将 **ActionScript** 代码添加到主时间轴的第一帧中。您要在本教程的其余步骤中添加到应用程序的所有代码都应放置在 **Actions** 图层中。

1. 若要从“类”库导入 **WebServiceClasses** 项目，请选择“窗口” > “公用库” > “类”。
2. 将 **WebServiceClasses** 项目从“类”库拖动到应用程序的库中。
从“类”库导入项目类似于将组件添加到库中：此操作将类的 SWC 文件添加到库中。要在应用程序中使用类，SWC 文件需要位于库中。
3. 在时间轴中，选择 **Form** 图层，然后单击“添加新图层”按钮。将新图层命名为 **Actions**。
4. 选择 **Actions** 图层，然后选择第 1 帧并按下 F9 打开“动作”面板。

5. 在“动作”面板中，输入以下代码以创建一个 `stop()` 函数，该函数可阻止应用程序在回放期间循环：

```
stop();
```

6. 保持选中 **Actions** 图层的第 1 帧，在“动作”面板中添加以下代码导入这些类：

```
// 导入需要的类。
import mx.services.Webservice;
import mx.controls.*;
import mx.containers.*;
import mx.controls.gridclasses.DataGridColumn;
// 导入自定义 Cart 类。
import Cart;
```

设置组件实例的数据类型

接下来，您将为本教程此前拖动到舞台上的每个组件实例指定数据类型。

ActionScript 2.0 使用严格数据类型指定，这意味着在创建变量时就要指定数据类型。严格数据类型指定使“动作”面板中的变量可以使用代码提示。

- 在“动作”面板中，添加以下代码为创建的四个组件实例指定数据类型。

```
/* 舞台上的数据类型实例；其它实例可在
   运行时从 Cart 类添加。*/
var problems_cb:ComboBox;
var products_dg:DataGrid;
var cart_dg:DataGrid;
var products_xmlcon:mx.data.components.XMLConnector;
```



您在此处指定的实例名称必须与您在向舞台上拖动组件时指定的实例名称相符。

自定义组件的外观

每个组件都有样式属性和方法，使用这些属性和方法可自定义组件的外观（包括加亮颜色、字体和字体大小）。您可以为单个组件实例设置样式，也可以在应用程序中以全局方式设置样式，以便应用于所有组件实例。在本教程中，将以全局方式设置样式。

- 添加以下代码设置样式：

```
// 为 problems_cb ComboBox 定义全局样式和缓动等式。
_global.style.setStyle("themeColor", "haloBlue");
_global.style.setStyle("fontFamily", "Verdana");
_global.style.setStyle("fontSize", 10);
_global.style.setStyle("openEasing",
    mx.transitions.easing.Bounce.easeOut);
```

此代码设置组件的主题颜色（所选项目上的加亮颜色）、字体和字体大小，并且还设置 **ComboBox** 的缓动方式，即单击 **ComboBox** 标题栏时，下拉列表出现和消失的方式。

在组合框中显示过失

在这一部分中，将要添加代码以便连接包含过失列表（Forgot to Water Your Plants 等等）的 Web 服务。Web 服务描述语言 (WSDL) 文件位于 www.flash-mx.com/mm/firstapp/problems.cfc?WSDL。若要查看 WSDL 的架构，请浏览到 WSDL 位置。

ActionScript 代码将 Web 服务结果传递到 ComboBox 实例以进行显示。函数会根据严重程度对过失进行排序。如果 Web 服务没有返回任何结果（例如，当服务关闭或未找到函数时），则在“输出”面板中将显示一条错误信息。

- 在“动作”面板中添加以下代码：

```
/* 定义用来检索问题数组的 Web 服务。  
此服务将绑定到 problems_cb ComboBox 实例。 */  
var problemService:WebService = new WebService("http://www.flash-mx.com/  
mm/firstapp/problems.cfc?WSDL");  
var myProblems:Object = problemService.getProblems();  
  
/* 如果从 Web 服务获得了结果，则设置将用作列标签的字段。  
将数据提供程序设置为从 Web 服务返回的结果。 */  
myProblems.onResult = function(wsdResults:Array) {  
    problems_cb.labelField = "name";  
    problems_cb.dataProvider = wsdResults.sortOn("severity",  
        Array.NUMERIC);  
};  
  
/* 如果无法连接到远程 Web 服务，则在“输出”面板中  
显示错误消息。 */  
myProblems.onFault = function(error:Object) {  
    trace("error:");  
    for (var prop in error) {  
        trace("  "+prop+" -> "+error[prop]);  
    }  
};
```

提示

按 Control+S 键以保存您的工作，然后按 Control+Enter 键（或选择“控制”>“测试影片”）来测试应用程序。组合框此时应当用过失列表进行填充，您应当能够看到为 Gift Ideas 创建的空的网格，以及 Checkout 按钮。

绑定数据组件以显示礼品方案

在本教程的开始部分，您已在舞台上添加了 **DataGrid**、**DataSet** 和 **XMLConnector** 组件的实例。您已将名为 **products_xmlcon** 的 **XMLConnector** 实例的 **URL** 属性设置为了某个 XML 文件的位置，而该文件包含了应用程序 **Gift Ideas** 部分的产品信息。

现在您需要在 **Flash** 创作环境中使用数据绑定功能将 **XMLConnector**、**DataSet** 和 **DataGrid** 组件绑定在一起，以便在应用程序中使用 XML 数据。有关使用数据绑定以及 **Flash** 数据集成体系结构其它功能的一般信息，请参阅《使用 **Flash**》中的第 16 章“数据集成（仅限 **Flash Professional**）”。

绑定组件时，**DataSet** 组件将根据用户在 **What Did You Do?** 部分中所选过失的严重性过滤 XML 文件中的产品列表。**DataGrid** 组件将显示该列表。

使用架构描述 XML 数据源

使用 **XMLConnector** 组件连接外部 XML 数据源时，需要指定一个架构，所谓架构就是描述 XML 文档结构的示意图。架构告诉 **XMLConnector** 组件如何读取 XML 数据源。指定架构最简单的方法是导入要连接的 XML 文件的副本，并将该副本用作架构。

1. 打开 Web 浏览器，并转到 www.flash-mx.com/mm/firstapp/problems.xml（为 **XMLConnector** 的 **URL** 参数设置的位置）。
2. 选择“文件”>“另存为”。
3. 将 **products.xml** 文件保存在与正在使用的 **FLA** 文件所在的同一位置。
4. 在主时间轴中选择第 1 帧。
5. 选择舞台旁边的 **products_xmlcon (XMLConnector)** 实例。
6. 在“组件”检查器中，单击“架构”选项卡。单击“导入”按钮（在“架构”选项卡的右侧，滚动窗格的上方）。在“打开”对话框中，找到在第 3 步中导入的 **products.xml** 文件，然后单击“打开”。**products.xml** 文件的架构即出现在“架构”选项卡的滚动窗格中。

在“架构”选项卡的顶部窗格中，选择 **image** 元素。在底部窗格中，选择 **data type**，并将值从 **<empty>** 更改为 **String**。对 **description** 元素重复此步骤。

过滤礼品方案以符合过失

您将使用“组件”检查器中的“绑定”选项卡将 XMLConnector、DataSet 和 DataGrid 组件实例进行相互绑定。

有关使用数据绑定的信息，请参阅《使用 Flash》中的“数据集成（仅限 Flash Professional）”。

1. 在舞台上选择 products_xmlcon (XMLConnector) 实例，单击“组件”检查器中的“绑定”选项卡。
2. 单击“添加绑定”按钮。
3. 在“添加绑定”对话框中，选择 results.products.product array 项目，然后单击“确定”。
4. 在“绑定”选项卡中，单击“绑定属性”窗格（底部窗格，其中显示属性名称 / 值）中的“绑定到”项目。
5. 在“绑定到”项目的“值”列中，单击放大镜图标打开“绑定到”对话框。
6. 在“绑定到”对话框中，选择“组件路径”窗格中的 DataSet <products_ds> 实例。在“架构位置”窗格中选择 dataProvider:array。单击“确定”。
7. 在“绑定”选项卡中，单击“绑定属性”窗格中的“Direction”项目。从“值”列的弹出菜单中，选择“Out”。

此选项意味着数据将从 products_xmlcon 实例传递到 products_ds 实例（而不是双向传递，或从 DataSet 实例传递到 XMLConnector 实例）。

8. 在舞台上，选择 products_ds 实例。在“组件”检查器的“绑定”选项卡中，组件的数据提供程序出现在“绑定列表”（“绑定”选项卡的顶部窗格）中。在“绑定属性”窗格中，“绑定到”参数指示 products_ds 实例绑定到 products_xmlcom 实例，且绑定方向为“In”。

在接下来的几个步骤中，将 DataSet 实例绑定到 DataGrid 实例，以便将由数据集过滤的数据显示在数据网格中。

9. 仍选中 products_ds 实例的情况下，单击“绑定”选项卡中的“添加绑定”按钮。
10. 在“添加绑定”对话框中，选择 dataProvider:array 项，然后单击“确定”。
11. 在“绑定”选项卡中，确保选择了“绑定列表”中的 dataProvider:array 项目。
12. 单击“绑定属性”窗格中的“绑定到”项目。
13. 在“绑定到”项目的“值”列中，单击放大镜图标打开“绑定到”对话框。
14. 在“绑定到”对话框中，选择“组件路径”窗格中的 products_dg (DataGrid) 实例。在“架构位置”窗格中选择 dataProvider:array。单击“确定”。

将列添加到 Gift Ideas 部分

现在您已准备就绪，可以向应用程序的 **Gift Ideas** 部分的数据网格中添加列，以显示产品信息和价格。

- 选择 **Actions** 图层。在“动作”面板中，添加以下代码创建和配置 **Name** 列和 **Price** 列，并将这些列添加到 **DataGrid** 实例：

```
// 定义数据网格列及其在 products_dg DataGrid 实例
// 中的默认宽度。
var name_dgc:DataGridColumn = new DataGridColumn("name");
name_dgc.headerText = "Name";
name_dgc.width = 280;

// 将列添加到 DataGrid。
products_dg.addColumn(name_dgc);
var price_dgc:DataGridColumn = new DataGridColumn("price");
price_dgc.headerText = "Price";
price_dgc.width = 100;

// 定义在运行时用来设置列标签
// 的函数。
price_dgc.labelFunction = function(item:Object) {
    if (item != undefined) {
        return "$"+item.price+" "+item.priceQualifier;
    }
};
products_dg.addColumn(price_dgc);
```

触发 XML Connector

接下来，将添加一行代码，使 **XMLConnector** 实例加载、解析并绑定远程 **products.xml** 文件中的内容。该文件位于您为先前创建的 **XMLConnector** 实例的 **URL** 属性所输入的 **URL** 上。它包含将出现在应用程序的 **Gift Ideas** 部分中的产品的有关信息。

- 在“动作”面板中添加以下代码：

```
products_xmlcon.trigger();
```

添加事件侦听器以过滤礼品方案

在这一部分中，将添加事件侦听器以便在用户选择 **What Did You Do?** 部分 (**problems_cb ComboBox** 实例) 中的过时进行检测。该侦听器包含一个函数，该函数根据用户选择的过时过滤 **Gift Ideas** 列表。选择轻微的过时将显示普通礼品的列表（如 **CD** 和 **鲜花**）；选择较严重的过时则显示较贵重的礼品。

有关使用事件侦听器的详细信息，请参阅《学习 Flash 中的 **ActionScript 2.0**》中的“使用事件侦听器”。

- 在“动作”面板中添加以下代码：

```
/* 为 problems_cb ComboBox 实例定义一个侦听器。  
该侦听器将过滤 DataSet (和 DataGrid) 中的产品。  
过滤是基于 ComboBox 中当前所选项目的严重性进行的。 */  
var cbListener:Object = new Object();  
cbListener.change = function(evt:Object) {  
    products_ds.filtered = false;  
    products_ds.filtered = true;  
    products_ds.filterFunc = function(item:Object) {  
        // 如果当前项目的严重性大于等于  
        // ComboBox 中所选的项目，则返回 true。  
        return (item.severity>=evt.target.selectedItem.severity);  
    };  
};  
  
// 将侦听器添加到 ComboBox。  
problems_cb.addEventListener("change", cbListener);
```

在 change() 函数的开头重置 filtered 属性（将其设置为 false，然后再设置为 true），可确保在用户重复更改 What Did You Do? 部分所选内容的情况下，该函数仍能正常工作。

filterFunc 函数检查礼品数组中的给定项目是否适用于用户在组合框中所选的严重性。如果礼品属于所选的严重性范围，则它将显示在 DataGrid 实例（已绑定到 DataSet 实例）中。

最后一行代码将侦听器注册到 problems_cb ComboBox 实例。

添加购物车

接下来将要添加的代码将会创建自定义 Cart 类的实例，并初始化该实例。

- 在“动作”面板中添加以下代码：

```
var myCart:Cart = new Cart(this);  
myCart.init();
```

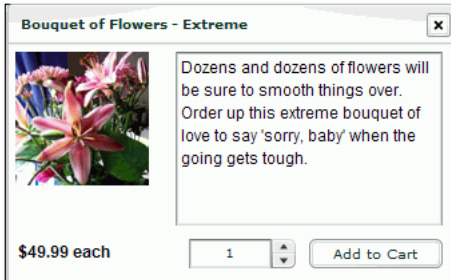
此代码使用 Cart 类的 init() 方法将一个 DataGrid 实例添加到舞台上，定义各列，并将该 DataGrid 实例定位在舞台上。它还添加一个 Button 组件实例并确定其位置，然后为该按钮添加一个 Alert 处理函数。（若要查看 Cart 类 init() 方法的代码，请打开 Cart.as 文件。）

提示

按 Control+S 键以保存您的工作，然后按 Control+Enter 键（或选择“控制”->“测试影片”）以测试应用程序。当您在组合框中选择某个过失时，您为 Gift Ideas 创建的数据网格应该显示礼品的子集，以符合所选的过失。

显示礼品详细信息

在应用程序中，用户在 **Gift Ideas** 部分中单击产品时会出现一个弹出窗口。该弹出窗口包含显示产品信息（包括文字说明、图像和价格）的组件实例。要制作此弹出窗口，需要创建一个影片剪辑元件，然后添加 **Loader**、**TextArea**、**Label**、**NumericStepper** 和 **Button** 组件的实例。**Bouquet of Flowers Extreme** 的产品详细信息窗口的外观如下所示：



稍后将添加 **ActionScript**，它可为每种产品动态创建该影片剪辑的实例。这些影片剪辑实例将显示在此前添加到库中的 **Window** 组件中。来自外部 **XML** 文件的元素将填充这些组件实例。

1. 从“组件”面板的用户界面树中，将 **Window** 组件的一个实例拖到库中。

Window 组件元件现在被添加到了库中。在本教程的后面部分，将使用 **ActionScript** 创建 **Window** 组件的实例。

2. 在“库”面板（“窗口”>“库”）中，单击标题栏右侧的选项菜单，然后选择“新建元件”。
3. 在“创建新元件”对话框中，为“名称”输入 **ProductForm**，然后为“类型”选择“影片剪辑”。
4. 单击“高级”按钮。在“链接”下面，选择“为 **ActionScript** 导出”，选中“在第一帧导出”，然后单击“确定”。新元件的文档窗口即会以元件编辑模式打开。

对于库中没有出现在舞台上的影片剪辑元件，必须选择“为 **ActionScript** 导出”，这样才能使用 **ActionScript** 进行处理。（在“第一帧导出”意味着只要加载了第一帧，即可使用影片剪辑。）在本教程的后面部分，您将添加 **ActionScript**，使用户每次在 **Gift Ideas** 部分中单击产品时，即动态生成该影片剪辑的实例。

5. 在新建元件的时间轴中，选择“图层 1”，然后将其重命名为 **Components**。
6. 从“组件”面板的用户界面树中，将 **Loader** 组件的一个实例拖动到舞台上。分别为 **x**、**y** 坐标输入 **5** 和 **5**。输入实例名称 **image_1dr**。在“属性”检查器中单击“参数”选项卡。为 **autoLoad** 选择 **false**，为 **scaleContent** 选择 **false**。

Loader 组件实例将用来显示产品的图像。将 **autoLoad** 设置为 **false** 则指定图像不自动加载。将 **scaleContent** 设置为 **false** 则指定图像不调整大小。在本教程的后面部分，将添加一些代码，这些代码根据用户在 **Gift Ideas** 部分选择的产品动态加载图像。

7. 从“组件”面板的用户界面树中，将 `TextArea` 组件的一个实例拖动到舞台上。将其放置在 `Loader` 组件的旁边。分别为 `x`、`y` 坐标输入 **125** 和 **5**。输入实例名称 **description_ta**。将“宽度”设置为 **200**，并将“高度”设置为 **130**。在“属性”检查器中单击“参数”选项卡。为 `editable` 选择 `false`。为 `html` 选择 `true`。为 `wordWrap` 选择 `true`。

`TextArea` 组件实例将用来显示所选产品的文本说明。所选设置指定该文本不能由用户编辑，可以使用 `HTML` 标记格式化，会换行以适合文本区域大小。

8. 从“组件”面板的用户界面树中，将 `Label` 组件的一个实例拖动到舞台上。将其放置在 `Loader` 组件的下面。将 `x` 和 `y` 坐标分别设置为 **5** 和 **145**。输入实例名称 **price_lbl**。在“属性”检查器中单击“参数”选项卡。为 `autoSize` 选择 `left`。为 `html` 选择 `true`。

`Label` 组件实例将显示产品价格和单位（由指定价格确定的产品的数量，如“每件”或“一打”）。

9. 从“组件”面板的用户界面树中，将 `NumericStepper` 组件的一个实例拖动到舞台上。将其放置在 `TextArea` 组件的下面。将 `x` 和 `y` 坐标分别设置为 **135** 和 **145**。输入实例名称 **quantity_ns**。在“属性”检查器中单击“参数”选项卡。为 `minimum` 输入 **1**。

将 `minimum` 设置为 **1**，即指定用户必须至少选择一件产品，才能将项目添加到购物车。

10. 从“组件”面板的用户界面树中，将 `Button` 组件的一个实例拖动到舞台上。将其放置在 `NumericStepper` 组件的旁边。将 `x` 和 `y` 坐标分别设置为 **225** 和 **145**。输入实例名称 **addToCart_button**。在“属性”检查器中单击“参数”选项卡。为 `label` 输入 **Add To Cart**。

添加事件侦听器以触发礼品详细信息的显示

接下来，您将向 `products_dg` `DataGrid` 实例中添加事件侦听器以显示每件产品的有关信息。如果用户在 `Gift Ideas` 部分中单击某件产品，则出现一个弹出窗口，其中包含该产品的有关信息。

- 在主时间轴的“动作”面板中，添加以下代码：

```
// 为 DataGrid 创建一个侦听器，以便在 DataGrid 中的行
// 发生改变时进行检测
var dgListener:Object = new Object();
dgListener.change = function(evt:Object) {
    // 如果 DataGrid 中的当前行发生改变，则启动新的弹出
    // 窗口，显示产品的详细信息。
    myWindow = mx.managers.PopUpManager.createPopUp(_root,
    mx.containers.Window, true, {title:evt.target.selectedItem.name,
    contentPath:"ProductForm", closeButton:true});
    // 设置弹出窗口的尺寸。
    myWindow.setSize(340, 210);
    // 定义用户单击关闭按钮时关闭该弹出窗口
    // 的侦听器。
```

```

var closeListener:Object = new Object();
closeListener.click = function(evt) {
    evt.target.deletePopUp();
};
myWindow.addEventListener("click", closeListener);
};
products_dg.addEventListener("change", dgListener);

```

此代码创建一个名为 **dgListener** 的新的事件侦听器，并创建先前已添加到库中的 **Window** 组件的实例。新窗口的标题设置为产品的名称。窗口的内容路径设置为 **ProductForm** 影片剪辑。窗口的大小设置为 **340 x 210** 像素。

此代码还添加了一个关闭按钮，使用户在查看信息之后可以关闭窗口。

向 ProductForm 影片剪辑添加代码

接下来，将向刚刚创建的 **ProductForm** 影片剪辑中添加 **ActionScript**。**ActionScript** 使用所选产品的有关信息填充影片剪辑内的组件，然后将一个事件侦听器添加到 **Add to Cart** 按钮，使用该按钮可将所选产品添加到购物车。

有关使用事件侦听器的详细信息，请参阅 **Using ActionScript in Flash**（《在 Flash 中使用 ActionScript》）中的“使用事件侦听器”。

1. 在 **ProductForm** 影片剪辑的时间轴中，创建一个新图层，并命名为 **Actions**。在 **Actions** 层中选择第一帧。
2. 在“动作”面板中添加以下代码：

```

// 创建一个对象以引用 DataGrid 中所选的产品项目。
var thisProduct:Object = this._parent._parent.products_dg.selectedItem;
// 使用所选产品的数据
// 填充 description_ta TextArea 实例和 price_lbl Label 实例。
description_ta.text = thisProduct.description;
price_lbl.text = "<b>$"+thisProduct.price+"
    "+thisProduct.priceQualifier+"</b>";
// 从应用程序目录加载产品的图像。
image_ldr.load(thisProduct.image);

```

提醒

代码中包含说明其用途的注释。在编写的所有 **ActionScript** 代码中，最好都包含这类注释，以便于您或任何其它人在以后查看这些代码时能够轻松地理解其用途。

首先，代码定义一个变量，用来在后续代码中引用所选产品。使用 **thisProduct** 变量意味着无需使用路径 **this._parent._parent.products_dg.selectedItem** 来引用指定的产品。

接下来，代码使用 `thisProduct` 对象的 `description`、`price` 和 `priceQualifier` 属性来填充 `TextArea` 实例和 `Label` 实例。这些属性对应于 `products.xml` 文件中的元素，`products.xml` 文件在本教程开始部分已链接到 `products.xmlcon XMLConnector` 实例。在本教程的后面部分，将把 `XMLConnector`、`DataSet` 和 `DataGrid` 组件实例绑定在一起，并使用 `XML` 文件中的元素填充其它两个组件实例。

最后，代码使用 `thisProduct` 对象实例的 `image` 属性将产品的图像加载到 `Loader` 组件中。

3. 接下来，将添加事件侦听器，以便在用户单击 **Add to Cart** 按钮时将产品添加到购物车。（在教程的后面部分，将向应用程序的主时间轴添加 `ActionScript`，以创建 `Cart` 类的实例。）添加以下代码：

```
var cartListener:Object = new Object();
cartListener.click = function(evt:Object) {
    var tempObj:Object = new Object();
    tempObj.quantity = evt.target._parent.quantity_ns.value;
    tempObj.id = thisProduct.id;
    tempObj.productObj = thisProduct;
    var theCart = evt.target._parent._parent._parent.myCart;
    theCart.addProduct(tempObj.quantity, thisProduct);
};
addToCart_button.addEventListener("click", cartListener);
```

4. 单击“语法检查”按钮（“脚本”窗格上方的蓝色复选标记），以确保代码中没有语法错误。

向应用程序添加代码时应经常检查语法。代码中发现的所有错误都在“输出”面板中列出。（在检查语法时，只检查当前的脚本；不检查可能位于 `FLA` 文件中的其它脚本。）有关详细信息，请参阅《学习 Flash 中的 `ActionScript 2.0`》中的“调试脚本”。

5. 单击文档窗口左上方的箭头按钮，或者选择“视图” > “编辑文档”以退出元件编辑模式，并返回到主时间轴。

提示

按 `Control+S` 键以保存您的工作，然后按 `Control+Enter` 键（或选择“控制” > “测试影片”）以测试应用程序。如果现在单击某个礼品选项，应该显示一个窗口，并显示该礼品的图像，同时显示说明、价格和允许您选择所需要数量的数值调节扭。

创建结帐屏幕

用户单击主屏幕上的 Checkout 按钮时，即出现 Checkout 屏幕。用户可在 Checkout 屏幕提供的表单中输入开单、发货和信用卡的信息。结帐屏幕的外观如下所示：



此结帐界面由放置在应用程序第 10 帧的关键帧上的组件组成。您将使用 **Accordion** 组件来创建结帐界面。**Accordion** 组件是包含一系列子项（一次显示一个）的浏览器。您还将添加一个 **Button** 组件实例来创建 **Back** 按钮，以便用户返回到主屏幕。

在本教程的后面部分，您将创建用作 **Accordion** 实例中的子项的影片剪辑，以显示 **Billing Information**、**Shipping Information** 和 **Credit Card Information** 窗格。

1. 在应用程序的主时间轴中，将播放头移动到第 10 帧（标记为 **Checkout**）。确保选择了 **Form** 图层。
2. 在 **Form** 图层中的第 10 帧插入一个空白关键帧（选择第 10 帧，然后选择“插入”>“时间轴”>“空白关键帧”）。
3. 选择了新的关键帧之后，从“组件”面板的用户界面树中，将 **Accordion** 组件的一个实例拖动到舞台上。在“属性”检查器中，输入实例名称 **checkout_acc**。将宽度设置为 **300** 像素，将高度设置为 **200** 像素。
4. 从“组件”面板的用户界面树中，将 **Button** 组件的一个实例拖动到舞台上的右下角处。在“属性”检查器中，输入实例名称 **back_button**。单击“参数”选项卡，然后在 **label** 属性输入 **Back**。

关于 Billing、Shipping 和 Credit Card 窗格

Billing Information、Shipping Information 和 Credit Card Information 窗格是使用显示在 Accordion 组件实例中的影片剪辑实例构建的。每个窗格都由两个嵌套的影片剪辑组成。

父影片剪辑包含一个 ScrollPane 组件，用来在可滚动区域中显示内容。子影片剪辑包含 Label 和 TextInput 组件，用户可在其中输入个人数据，如姓名、地址等等。您将使用 ScrollPane 组件来显示子影片剪辑，以便用户可以在信息字段之间滚动。

创建 Billing Information 窗格

首先您将创建两个显示 Billing Information 表单字段的影片剪辑：具有 ScrollPane 组件实例的父影片剪辑和具有 Label 和 TextArea 组件实例的子影片剪辑。

1. 在“库”面板（“窗口”>“库”）中，单击标题栏右侧的选项菜单，然后选择“新建元件”。
2. 在“创建新元件”对话框中，为“名称”输入 **checkout1_mc**，然后为“类型”选择“影片剪辑”。
3. 单击“高级”按钮。在“链接”下面，选择“为 ActionScript 导出”，选中“在第一帧导出”，然后单击“确定”。

新元件的文档窗口即会以元件编辑模式打开。

4. 将 ScrollPane 组件的一个实例拖动到舞台上。
5. 在“属性”检查器中，输入实例名称 **checkout1_sp**。将 W 和 H 的值分别设置为 **300** 和 **135**。将 x 和 y 坐标分别设置为 **0** 和 **0**。
6. 单击“参数”选项卡。将 contentPath 属性设置为 **checkout1_sub_mc**。

checkout1_sub_mc 影片剪辑将出现在滚动窗格内，并包含 Label 和 TextInput 组件。接下来，您将创建该影片剪辑。

7. 从“库”选项菜单中选择“新建元件”。
8. 在“创建新元件”对话框中，为“名称”输入 **checkout1_sub_mc**，然后为“类型”选择“影片剪辑”。
9. 单击“高级”按钮。在“链接”下面，选择“为 ActionScript 导出”，选中“在第一帧导出”，然后单击“确定”。

新元件的文档窗口即会以元件编辑模式打开。

10. 将 Label 组件的六个实例拖动到舞台上。或者，先将一个实例拖动到舞台上，然后在舞台上按住 Ctrl 键（在 Windows 中）或 Option 键（在 Macintosh 中）并单击拖动该实例以制作副本。命名并放置实例，如下所示：

- 对于第一个实例，输入实例名称 **firstname_lbl**，并将 x 和 y 坐标分别设置为 5 和 5。单击“参数”选项卡，为 text 输入 **First Name**。
- 对于第二个实例，输入实例名称 **lastname_lbl**，并将 x 和 y 坐标分别设置为 5 和 35。单击“参数”选项卡，为 text 输入 **Last Name**。
- 对于第三个实例，输入实例名称 **country_lbl**，并将 x 和 y 坐标分别设置为 5 和 65。单击“参数”选项卡，为 text 输入 **Country**。
- 对于第四个实例，输入实例名称 **province_lbl**，并将 x 和 y 坐标分别设置为 5 和 95。单击“参数”选项卡，为 text 输入 **Province/State**。
- 对于第五个实例，输入实例名称 **city_lbl**，并将 x 和 y 坐标分别设置为 5 和 125。单击“参数”选项卡，为 text 输入 **City**。
- 对于第六个实例，输入实例名称 **postal_lbl**，并将 x 和 y 坐标分别设置为 5 和 155。单击“参数”选项卡，为 text 输入 **Postal/Zip Code**。

11. 将 TextInput 组件的六个实例拖动到舞台上。紧靠每个 Label 实例的右侧放置一个 TextInput 实例。例如，第一个 TextInput 实例的 x 和 y 坐标分别应为 105 和 5。命名 TextInput 实例，如下所示：

- 将第一个实例命名为 **billingFirstName_ti**。
- 将第二个实例命名为 **billingLastName_ti**。
- 将第三个实例命名为 **billingCountry_ti**。
- 将第四个实例命名为 **billingProvince_ti**。
- 将第五个实例命名为 **billingCity_ti**。
- 将第六个实例命名为 **billingPostal_ti**。

如果滚动窗格中的内容太靠近窗格的边框，有时候内容可能会被裁切。在接下来的几个步骤中，您将向 checkout1_sub_mc 影片剪辑中添加一个白色矩形，以便正确显示 Label 和 TextInput 实例。

12. 在时间轴中，单击“添加新图层”按钮。将新图层拖动到现有图层的下面。（矩形所在的图层应位于底部，这样矩形不会干扰组件的显示。）

13. 选择新图层的第 1 帧。

14. 在“工具”面板中，选择“矩形”工具。将笔触颜色设置为“无”，将填充颜色设置为白色。

单击“工具”面板中的“笔触颜色”控件，然后单击“无”按钮，即贯穿红线的白色样本。单击“填充颜色”控件，然后单击白色样本。

15. 通过拖动创建一个超出 Label 和 TextInput 实例底边缘和右边缘的矩形。

创建 Shipping Information 窗格

Shipping Information 窗格的影片剪辑与 Billing Information 窗格的影片剪辑非常相似。您还将添加一个 CheckBox 组件，使用户能够使用他们在 Billing Information 窗格中输入的数据填充 Shipping Information 表单字段。

1. 按照前述说明（请参见第 35 页的“创建 Billing Information 窗格”）创建 Credit Card Information 窗格的影片剪辑。请注意以下这些命名差异：
 - 对于第一个影片剪辑，输入元件名称 **checkout2_mc**，实例名称 **checkout2_sp**。在“属性”检查器的“参数”选项卡中，将 `contentPath` 属性设置为 **checkout2_sub_mc**。
 - 对于第二个影片剪辑，输入元件名称 **checkout2_sub_mc**。
 - 对于 TextInput 实例，将实例名称“billing”更改为“shipping”。
2. 在元件编辑模式下打开 checkout2_sub_mc 影片剪辑之后，将 CheckBox 组件的一个实例拖动到舞台上，并将它定位在第一个 Label 实例的正上方。
确保将此实例与其它组件实例一起放置在“图层 1”中。
3. 在“属性”检查器中，输入实例名称 **sameAsBilling_ch**。
4. 单击“参数”选项卡。将 `label` 属性设置为 **Same As Billing Info**。

创建 Credit Card Information 窗格

Credit Card Information 窗格的影片剪辑与 Billing Information 和 Shipping Information 窗格的影片剪辑非常相似。但是，Credit Card Information 窗格的嵌套影片剪辑具有其它两个窗格没有的字段，这些字段用于信用卡号和其它信用卡数据。

1. 按照 Billing Information 中第 1 至第 9 步的说明（请参阅第 35 页的“创建 Billing Information 窗格”），创建 Credit Card Information 窗格的影片剪辑。请注意以下这些命名差异：
 - 对于第一个影片剪辑，输入元件名称 **checkout3_mc**，实例名称 **checkout3_sp**。在“属性”检查器的“参数”选项卡中，将 `contentPath` 属性设置为 **checkout3_sub_mc**。
 - 对于第二个影片剪辑，输入元件名称 **checkout3_sub_mc**。
2. 将 Label 组件的四个实例拖动到舞台上。命名并放置实例，如下所示：
 - 对于第一个实例，输入实例名称 **ccName_lbl**，并将 `x` 和 `y` 坐标分别设置为 **5** 和 **5**。单击“参数”选项卡，为 `text` 输入 **Name On Card**。
 - 对于第二个实例，输入实例名称 **ccType_lbl**，并将 `x` 和 `y` 坐标分别设置为 **5** 和 **35**。单击“参数”选项卡，为 `text` 输入 **Card Type**。

- 对于第三个实例，输入实例名称 **ccNumber_lbl**，并将 **x** 和 **y** 坐标分别设置为 **5** 和 **65**。单击“参数”选项卡，为 **text** 输入 **Card Number**。
 - 对于第四个实例，输入实例名称 **ccExp_lbl**，并将 **x** 和 **y** 坐标分别设置为 **5** 和 **95**。单击“参数”选项卡，为 **text** 输入 **Expiration**。
3. 将 **TextInput** 组件的一个实例拖动到舞台上，将其定位在 **ccName_lbl** 实例的右边。将此新建实例命名为 **ccName_cb**。将 **x** 和 **y** 坐标分别设置为 **105** 和 **5**。将宽度设置为 **140**。
 4. 将 **ComboBox** 组件的一个实例拖动到舞台上，将其定位在 **ccType_lbl** 实例的右边。将此新建实例命名为 **ccType_cb**。将 **x** 和 **y** 坐标分别设置为 **105** 和 **35**。将宽度设置为 **140**。
 5. 将 **TextInput** 组件的另一个实例拖动到舞台上，将其定位在 **ccNumber_lbl** 实例的右边。将此新建实例命名为 **ccNumber_cb**。将 **x** 和 **y** 坐标分别设置为 **105** 和 **65**。将宽度设置为 **140**。
 6. 将 **ComboBox** 组件的两个实例拖动到舞台上。将一个实例定位在 **ccExp_lbl** 实例的右边，然后将另一个实例定位在前一个实例的右边。将第一个新建实例命名为 **ccMonth_cb**。将宽度设置为 **60**，并将 **x** 和 **y** 坐标分别设置为 **105** 和 **95**。将第二个实例命名为 **ccYear_cb**。将宽度设置为 **70**，并将 **x** 和 **y** 坐标分别设置为 **175** 和 **95**。
 7. 将 **Button** 组件的一个实例拖动到舞台上，并将其放置在表单的底部，**ccMonth_cb** 实例的下面。将此新建实例命名为 **checkout_button**。将 **x** 和 **y** 坐标分别设置为 **125** 和 **135**。在“属性”检查器的“参数”选项卡中，将 **label** 属性设置为 **Checkout**。
 8. 按照 **Billing Information** 中第 14 至第 15 步的说明（请参阅第 35 页的“创建 **Billing Information** 窗格”），在表单的底部添加一个矩形。

向 Checkout 按钮添加事件侦听器

现在将添加代码以便在用户单击 **Checkout** 按钮时显示 **Checkout** 屏幕。

- 在主页的“动作”面板中，添加以下代码：

```
// 单击 Checkout 按钮时，转到“checkout”帧标签。
var checkoutBtnListener:Object = new Object();
checkoutBtnListener.click = function(evt:Object) {
    evt.target._parent.gotoAndStop("checkout");
};
checkout_button.addEventListener("click", checkoutBtnListener);
```

此代码指定：用户单击 **Checkout** 按钮时，播放头移动到时间轴中的 **Checkout** 标签处。

为 Checkout 屏幕添加代码

现在您已准备就绪，可以向应用程序的 **Checkout** 屏幕（主时间轴上的第 10 帧）添加代码了。此代码处理用户在 **Billing Information**、**Shipping Information** 和 **Credit Card Information** 窗格（先前使用 **Accordion** 组件和其它组件创建）中输入的数据。

1. 在时间轴中，选择 **Actions** 图层中的第 10 帧，然后插入一个空白关键帧（选择“插入”>“时间轴”>“空白关键帧”）
2. 打开“动作”面板 (F9)。
3. 在“动作”面板中添加以下代码：

```
stop();
import mx.containers.*;

// 定义舞台上的 Accordion 组件。
var checkout_acc:Accordion;
```

4. 接下来，将向 **Accordion** 组件实例添加第一个子项，以接受来自用户的开单信息。添加以下代码：

```
// 定义 Accordion 组件的子项。
var child1 = checkout_acc.createChild("checkout1_mc", "child1_mc",
    {label:"1. Billing Information"});
var thisChild1 = child1.checkout1_sp.spContentHolder;
```

第一行调用 **Accordion** 组件的 `createChild()` 方法，并创建 `checkout1_mc` 影片剪辑元件（先前已创建）的一个实例，其实例名称为 `child1_mc`，标签为“1. **Billing Information**”。第二行代码创建指向嵌入的 **ScrollPane** 组件实例的快捷方式。

5. 创建 **Accordion** 实例的第二个子项，以接受发运信息：

```
/* 将第二个子项添加到 Accordion。
为 sameAsBilling_ch CheckBox 添加一个事件侦听器。
此操作将第一个子项的表单值复制到第二个子项。 */
var child2 = checkout_acc.createChild("checkout2_mc", "child2_mc",
    {label:"2. Shipping Information"});
var thisChild2 = child2.checkout2_sp.spContentHolder;
var checkboxListener:Object = new Object();
checkboxListener.click = function(evt:Object) {
    if (evt.target.selected) {
        thisChild2.shippingFirstName_ti.text =
            thisChild1.billingFirstName_ti.text;
        thisChild2.shippingLastName_ti.text =
            thisChild1.billingLastName_ti.text;
        thisChild2.shippingCountry_ti.text =
            thisChild1.billingCountry_ti.text;
        thisChild2.shippingProvince_ti.text =
            thisChild1.billingProvince_ti.text;
        thisChild2.shippingCity_ti.text = thisChild1.billingCity_ti.text;
```

```

        thisChild2.shippingPostal_ti.text =
            thisChild1.billingPostal_ti.text;
    }
};
thisChild2.sameAsBilling_ch.addEventListener("click", checkboxListener);

```

最前面的两行代码类似于创建 **Billing Information** 子项的代码：创建 checkout2_mc 影片剪辑元件的一个实例，实例名称为 child2_mc，标签为 “2.Shipping Information”。第二行代码创建指向嵌入的 **ScrollPane** 组件实例的快捷方式。

从第三行代码开始，向 **CheckBox** 实例添加一个事件侦听器。如果用户单击该复选框，发运信息则使用用户在 **Billing Information** 窗格中输入的数据。

6. 然后，为 **Accordion** 实例创建第三个子项，用于信用卡信息：

```

// 定义第三个 Accordion 子项。
var child3 = checkout_acc.createChild("checkout3_mc", "child3_mc",
    {label:"3. Credit Card Information"});
var thisChild3 = child3.checkout3_sp.spContentHolder;

```

7. 添加此代码以创建用于信用卡月份、年份和类型的 **ComboBox** 实例，并使用一个静态定义的数组填充所有实例：

```

/* 设置舞台上三个 ComboBox 实例的值：
ccMonth_cb, ccYear_cb and ccType_cb */
thisChild3.ccMonth_cb.labels = ["01", "02", "03", "04", "05", "06", "07",
    "08", "09", "10", "11", "12"];
thisChild3.ccYear_cb.labels = [2004, 2005, 2006, 2007, 2008, 2009, 2010];
thisChild3.ccType_cb.labels = ["VISA", "MasterCard", "American Express",
    "Diners Club"];

```

8. 最后，添加以下代码将事件侦听器添加到 **Checkout** 按钮和 **Back** 按钮。用户单击 **Checkout** 按钮时，侦听器对象将 **Billing Information**、**Shipping Information** 和 **Credit Card Information** 窗格的表单字段复制到发送至服务器的 **LoadVars** 对象中。（使用 **LoadVars** 类可将对象中的所有变量发送到指定的 URL。）用户单击 **Back** 按钮时，应用程序返回到主屏幕。

```

/* 为 checkout_button Button 实例创建一个侦听器。
用户单击 Checkout 按钮时，此侦听器将所有表单变量发送到服务器。 */
var checkoutListener:Object = new Object();
checkoutListener.click = function(evt:Object){
    evt.target.enabled = false;
    /* 创建两个 LoadVars 对象实例，它们将变量发送到远程服务器，
    并接收来自远程服务器的结果。 */
    var response_lv:LoadVars = new LoadVars();
    var checkout_lv:LoadVars = new LoadVars();
    checkout_lv.billingFirstName = thisChild1.billingFirstName_ti.text;
    checkout_lv.billingLastName = thisChild1.billingLastName_ti.text;
    checkout_lv.billingCountry = thisChild1.billingCountry_ti.text;
    checkout_lv.billingProvince = thisChild1.billingProvince_ti.text;
    checkout_lv.billingCity = thisChild1.billingCity_ti.text;

```



```

checkout_lv.billingPostal = thisChild1.billingPostal_ti.text;
checkout_lv.shippingFirstName = thisChild2.shippingFirstName_ti.text;
checkout_lv.shippingLastName = thisChild2.shippingLastName_ti.text;
checkout_lv.shippingCountry = thisChild2.shippingCountry_ti.text;
checkout_lv.shippingProvince = thisChild2.shippingProvince_ti.text;
checkout_lv.shippingCity = thisChild2.shippingCity_ti.text;
checkout_lv.shippingPostal = thisChild2.shippingPostal_ti.text;
checkout_lv.ccName = thisChild3.ccName_ti.text;
checkout_lv.ccType = thisChild3.ccType_cb.selectedItem;
checkout_lv.ccNumber = thisChild3.ccNumber_ti.text;
checkout_lv.ccMonth = thisChild3.ccMonth_cb.selectedItem;
checkout_lv.ccYear = thisChild3.ccYear_cb.selectedItem;

/* 将变量从 checkout_lv LoadVars 发送到服务器上的远程脚本。
保存 response_lv 实例中的结果。*/
checkout_lv.sendAndLoad("http://www.flash-mx.com/mm/firstapp/
cart.cfm", response_lv, "POST");
response_lv.onLoad = function(success:Boolean) {
    evt.target.enabled = true;
};
};
thisChild3.checkout_button.addEventListener("click", checkoutListener);
cart_mc._visible = false;
var backListener:Object = new Object();
backListener.click = function(evt:Object) {
    evt.target._parent.gotoAndStop("home");
}
back_button.addEventListener("click", backListener);

```

测试应用程序

恭喜！您已完成应用程序的构建。现在按 **Control+S** 键以保存您的工作，然后按 **Control+Enter** 键（或选择“控制”>“测试影片”）以测试应用程序。

查看完成后的应用程序

如果您未能成功完成教程，您可以查看已完成应用程序的工作版本。您可以在硬盘上的 **Samples** 文件夹中找到开始时的 Flash (FLA) 文件 `first_app_start fla` 和完成后的文件 `first_app fla`：

- 在 Windows 中：启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\ComponentsApplication。
- 在 Macintosh 上：Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/ComponentsApplication。

若要查看应用程序的 FLA 文件，请打开 `components_application` 文件夹中的 `first_app fla` 文件。

您可以将这些文件与您自己的文件进行比较，以帮助查找错误。

应用程序中用到的所有组件（以及图形文件和用来创建应用程序的其它资源）都显示在库中。某些组件以实例的形式显示在舞台上。某些组件由 **ActionScript** 代码引用，这些组件直到运行时才会显示。

使用组件

在本章中，您将使用几个 Macromedia Flash (FLA) 文件和 ActionScript 类文件来学习如何将组件添加到文档，并设置这些组件的属性。本章还将介绍一些高级话题，例如使用代码提示、创建自定义焦点导航、管理组件深度，以及将第 1 版组件升级为第 2 版 Macromedia 组件体系结构。

本章使用的文件是 TipCalculator.fla 和 TipCalculator.swf。这些文件安装在硬盘上的如下位置：

- (Windows) Program Files\Macromedia\Flex 8\Samples and Tutorials\Samples\Components\TipCalculator
- (Macintosh) Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/TipCalculator

本章介绍以下主题：

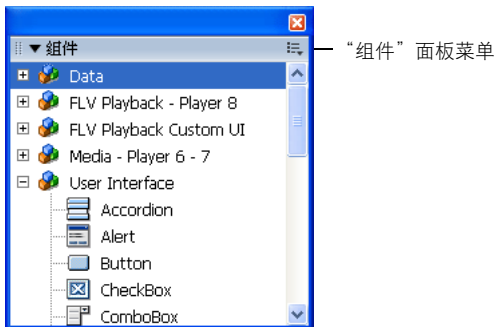
“组件”面板.....	44
向 Flash 文档中添加组件.....	44
“库”面板中的组件	47
设置组件参数	48
调整组件大小	49
从 Flash 文档删除组件.....	50
使用代码提示	50
创建自定义焦点导航	51
管理文档中的组件深度.....	52
“实时预览”中的组件	52
配合使用预加载器和组件.....	53
关于加载组件	54
将第 1 版组件升级到第 2 版的体系结构.....	54

“组件” 面板

用户级别 `configuration/Components` 目录中的所有组件都显示在“组件”面板中。（有关此目录的详细信息，请参阅第 12 页的“组件文件的存储位置”。）

要显示“组件”面板，请执行以下操作：

- 选择“窗口” > “组件”。



要显示 Flash 启动后安装的组件，请执行以下操作：

1. 选择“窗口” > “组件”。
2. 从“组件”面板弹出菜单中选择“重新加载”。

向 Flash 文档中添加组件

从“组件”面板将组件拖到舞台上时，会将一个编译剪辑 (SWC) 元件添加到“库”面板中。将 SWC 元件添加到库中后，就可以将多个实例拖动到舞台上。您也可以在运行时通过使用 `UIObject.createClassObject()` `ActionScript` 方法将该组件添加到文档中。



Menu 和 Alert 这两个组件例外，并且无法使用 `UIObject.createClassObject()` 来实例化。它们使用的是 `show()` 方法。

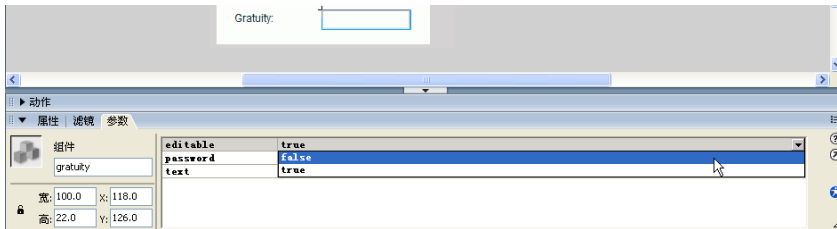
在创作时添加组件

您可以使用“组件”面板向文档添加组件，然后将组件从“库”面板中拖到舞台，向文档添加该组件的更多实例。在“属性”检查器的“参数”选项卡或“组件”检查器的“参数”选项卡中可以设置其它实例的属性。

使用“组件”面板向 Flash 文档添加组件：

1. 选择“窗口”>“组件”。
2. 请执行以下操作之一：
 - 将组件从“组件”面板拖到舞台上。
 - 双击“组件”面板中的一个组件。
3. 如果该组件为 FLA 文件（安装的所有第 2 版组件都是 SWC 文件），并且您已经编辑了同一组件的另一个实例的外观，或者编辑了与要添加的组件共享外观的组件的外观，则请执行以下操作之一：
 - 选择“不要替换现有项目”，保留已编辑的外观并将经过编辑的外观应用于新组件。
 - 选择“替换现有项目”，以默认外观替换所有外观。新组件和该组件所有以前的版本，或者与它共享相同外观的组件的以前版本都将使用默认外观。
4. 在舞台上选择该组件。
5. 选择“窗口”>“属性”>“属性”。
6. 在“属性”检查器中，输入组件实例的实例名称。
7. 单击“参数”选项卡，然后为实例指定参数。

下图显示了位于 TipCalculator.fla 范例文件（安装在 Flash 8/Samples and Tutorials/Samples/Components/TipCalculator）中的 TextInput 组件的“属性”检查器。



有关详细信息，请参阅第 48 页的“设置组件参数”。

8. 通过编辑宽度和高度的值，按需更改组件的大小。

有关调整特定组件类型大小的详细信息，请参阅《组件语言参考》中的各个组件条目。
9. 如果要更改组件的颜色和文本格式，请执行以下一项或多项操作：
 - 通过使用可用于所有组件的 `setStyle()` 方法设置或更改组件实例的特定样式属性值。有关详细信息，请参阅第 1273 页的 `UIObject.setStyle()`。
 - 在分配给所有第 2 版组件的全局样式声明中编辑多个属性。
 - 为特定组件实例创建自定义样式声明。

有关详细信息，请参阅第 69 页的“使用样式自定义组件的颜色和文本”。
10. 如果要自定义组件的外观，请执行以下操作之一：
 - 应用主题（请参阅第 91 页的“关于主题”）。

- 编辑组件的外观（请参阅第 81 页的“关于设置组件外观”）。

使用 ActionScript 在运行时添加组件

本节的说明假定您具备 ActionScript 的中级或高级知识。

使用 `createClassObject()` 方法（大多数组件都从 `UIObject` 类继承该方法）向 Flash 应用程序动态添加组件。例如，您可以添加基于用户设置的首选项（如网站入口首页上的首选项）来创建页面布局的组件。

随 Flash 安装的第 2 版组件位于包目录中。（有关详细信息，请参阅《学习 Flash 中的 ActionScript 2.0》中的“关于包”。）如果在创作时向舞台添加组件，使用其实例名称（例如，`myButton`）即可引用该组件。但是，如果使用 ActionScript（在运行时）向应用程序添加组件，则必须指定其全限定类名称（例如，`mx.controls.Button`），或者使用 `import` 语句来导入包。

例如，要编写引用 `Alert` 组件的 ActionScript 代码，可以使用 `import` 语句来引用类，如下所示：

```
import mx.controls.Alert;
Alert.show("The connection has failed", "Error");
```

或者，也可以使用完整的包路径，如下所示：

```
mx.controls.Alert.show("The connection has failed", "Error");
```

有关详细信息，请参阅《学习 Flash 中的 ActionScript 2.0》中的“关于导入类文件”。

您可以使用 ActionScript 方法为动态添加的组件设置其它参数。有关详细信息，请参阅《组件语言参考》。

提醒

要在运行时向文档添加某个组件，则编译此 SWF 文件时该组件必须在库中。若要将组件添加到库中，请将组件图标从“组件”面板拖到库中。此外，如果您要将包含动态实例化（使用 ActionScript 实例化）的组件的影片剪辑加载到另一个影片剪辑中，则在编译 SWF 文件时父影片剪辑库中必须有该组件。

使用 ActionScript 向 Flash 文档中添加组件：

1. 将组件从“组件”面板拖入当前文档的库中。

提醒

默认情况下，组件被设置为“在第一帧导出”（在 Windows 中右键单击，或在 Macintosh 中按住 `control` 键单击，然后选择“链接”菜单选项，可看到“在第一帧导出”设置）。如果要对包含组件的应用程序适用预加载器，则需要更改导出帧，有关说明，请参阅第 53 页的“配合使用预加载器和组件”。

2. 在时间轴中选择一帧放置要添加的组件。
3. 如果“动作”面板尚未打开，请将其打开。

4. 调用 `createClassObject()`，以便在运行时创建组件实例。

此方法可以单独调用，也可以从任何组件实例调用。`createClassObject()` 方法具有以下参数：组件类名称、新实例的实例名称、深度和可在运行时用来设置属性的可选初始化对象。

在类名称参数中可以指定类包，如下例所示：

```
createClassObject(mx.controls.CheckBox, "cb", 5, {label:"Check Me"});
```

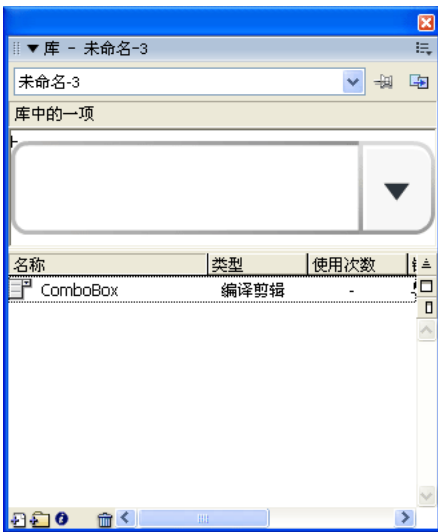
或者，也可以导入类包，如下例所示：

```
import mx.controls.CheckBox;  
createClassObject(CheckBox, "cb", 5, {label:"Check Me"});
```

有关详细信息，请参阅第 1254 页的 `UIObject.createClassObject()` 和第 55 页的第 4 章“处理组件事件”。

“库”面板中的组件

当您把组件添加到文档中后，它将在“库”面板中显示为编译剪辑（SWC 文件）元件。



“库”面板中的 ComboBox 组件

通过将组件图标从库拖到舞台上，您可以添加该组件的多个实例。

有关编译剪辑的详细信息，请参阅第 17 页的“关于编译剪辑和 SWC 文件”。

设置组件参数

每个组件都带有参数，通过设置这些参数可以更改组件的外观和行为。参数是在“属性”检查器和“组件”检查器中显示的属性。最常用的属性显示为创作参数；其它参数必须使用 **ActionScript** 来设置。可以在创作时设置的所有参数都可以使用 **ActionScript** 来设置。使用 **ActionScript** 设置参数将覆盖在创作时设置的任何值。

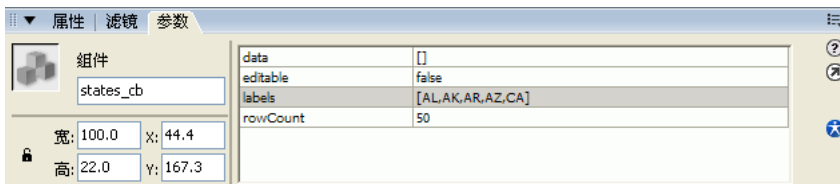
所有第 2 版用户界面 (UI) 组件都从 **UIObject** 类和 **UIComponent** 类继承属性和方法；这些是所有组件都使用的属性和方法，如 **UIObject.setSize()**、**UIObject.setStyle()**、**UIObject.x** 和 **UIObject.y** 等。每个组件还具有特有的属性和方法，其中的某些属性和方法可用作创作参数。例如，**ProgressBar** 组件具有 **percentComplete** 属性 (**ProgressBar.percentComplete**)，**NumericStepper** 组件具有 **nextValue** 和 **previousValue** 属性 (**NumericStepper.nextValue**、**NumericStepper.previousValue**)。

使用“组件”检查器或“属性”检查器（使用两个面板的作用是一样的）可以设置组件实例的参数。

在“属性”检查器中输入组件的实例名称：

1. 选择“窗口” > “属性” > “属性”。
2. 在舞台上选择组件的一个实例。
3. 在字“组件”下面的文本框中输入实例名称。

最好向实例名称添加用于指示组件类型的后缀；这可使 **ActionScript** 代码更加易读。在本例中，实例名称为 **states_cb**，这是因为该组件是用来列出美国各个州的组合框。



要在“组件”检查器中输入组件实例的参数：

1. 选择“窗口” > “组件检查器”。
2. 在舞台上选择组件的一个实例。

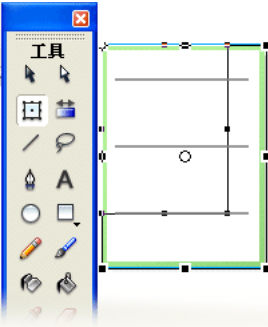
3. 要输入参数，请单击“参数”选项卡。



4. 要输入或查看组件的绑定或架构，请单击其相应选项卡。有关详细信息，请参阅《使用 Flash》中的“数据集成（仅限 Flash Professional）”。

调整组件大小

可以使用“任意变形”工具或 `setSize()` 方法来调整组件实例的大小。

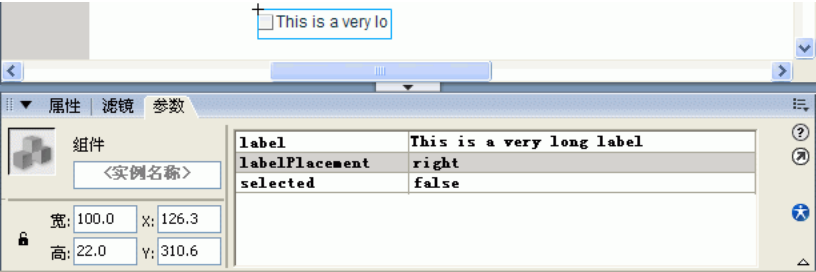


使用“任意变形”工具调整舞台上的 Menu 组件的大小

从任何组件实例中都可以调用 `setSize()` 方法（请参阅第 1271 页的 [UIObject.setSize\(\)](#)）来调整组件大小。下面的代码将 `TextArea` 组件的大小调整为 200 像素宽和 300 像素高：
`myTextArea.setSize(200, 300);`

提醒 如果使用 `ActionScript_width` 和 `_height` 属性来调整组件的宽度和高度，则可调整该组件的大小，但组件中内容的布局保持不变。这可能会导致在影片回放时组件发生扭曲。

组件不会自动调整大小以适合其标签。如果添加到文档中的组件实例不够大，而无法显示其标签，就会将标签文本剪切掉。您必须调整组件大小以适合其标签。



CheckBox 组件经剪切的标签

有关调整组件大小的详细信息，请参阅《组件语言参考》中各个组件条目。

从 Flash 文档删除组件

若要从 Flash 文档中删除组件的实例，不仅需要从舞台上将之删除，并且需要从库中删除编译剪辑的图标。

从文档中删除组件：

1. 在“库”面板中，选择编译过的剪辑 (SWC) 元件。
2. 单击“库”面板底部的“删除”按钮，或从“库”选项菜单中选择“删除”。
3. 在“删除”对话框中，单击“删除”以确认删除操作。

使用代码提示

使用 **ActionScript 2.0** 时，可以使用基于内置类（包括组件类）的变量的严格类型指定。如果您这样做，**ActionScript** 编辑器将显示该变量的代码提示。例如，假设您键入以下内容：

```
import mx.controls.CheckBox;
var myCheckBox:CheckBox;
myCheckBox.
```

在 `myCheckBox` 之后键入句点后，**Flash** 即显示一系列 **CheckBox** 组件可用的方法和属性，这是因为您已将该变量指定为 **CheckBox**。有关详细信息，请参阅《学习 Flash 中的 ActionScript 2.0》中的“关于指定数据类型和严格数据类型指定”和“使用代码提示”。

创建自定义焦点导航

当用户按 **Tab** 键在 **Flash** 应用程序中导航，或在应用程序中单击时，**FocusManager** 类确定接收输入焦点的组件（有关详细信息，请参阅《组件语言参考》中的 [FocusManager](#) 类。您不必在应用程序中添加 **FocusManager** 实例，也不必编写任何代码来激活 **Focus Manager**。

如果 **RadioButton** 对象接收焦点，焦点管理器将检查该对象和具有相同 `groupName` 值的所有对象，然后将焦点设置在 `selected` 属性设置为 `true` 的对象上。

每个模式窗口组件都包含焦点管理器的一个实例，因此，该窗口上的控件也就成为它们自己的 **Tab** 集。这样可以防止用户按 **Tab** 键切换到其它窗口中的组件。

要在应用程序中创建焦点导航，请在应该接收焦点的所有组件（包括按钮）上设置 `tabIndex` 属性。当用户按 **Tab** 键时，**FocusManager** 类便查找 `tabIndex` 值大于 `tabIndex` 当前值的已启用对象。**FocusManager** 类一旦达到最高的 `tabIndex` 属性值，则返回到 **0**。例如，在下面的代码中，`comment` 对象（可能是一个 **TextArea** 组件）首先接收焦点，然后 `okButton` 实例接收焦点：

```
var comment:mx.controls.TextArea;  
var okButton:mx.controls.Button;  
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。

如果舞台上没有任何对象具有 **Tab** 键索引值，则焦点管理器使用深度级别（**z**- 顺序）。深度级别主要按组件拖到舞台上的顺序设置；但是，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 **z**- 顺序。

要将焦点指定给应用程序中的组件，请调用 `FocusManager.setFocus()`。

要创建在用户按下 **Enter** 键 (Windows) 或 **Return** 键 (Macintosh) 时接收焦点的按钮，请将 `FocusManager.defaultPushButton` 属性设置为所需按钮的实例，如下面的代码所示：

```
focusManager.defaultPushButton = okButton;
```

[FocusManager](#) 类 (API) 会覆盖默认的 **Flash Player** 聚焦框，并绘制一个圆角的自定义聚焦框。

有关在 **Flash** 应用程序中创建焦点切换方案的详细信息，请参阅《组件语言参考》中的 [FocusManager](#) 类。

管理文档中的组件深度

如果要在应用程序中将组件放在另一个对象的前面或后面，则必须使用 [DepthManager](#) 类，请参阅《组件语言参考》中的相关章节。使用 [DepthManager](#) 类的方法，可以按照适当的相对顺序来放置用户界面组件，例如，组合框在其它组件的前面下拉、插入点显示在最前面、对话框浮于内容上方等。

深度管理器有两个主要用途：管理任何文档内的相对深度分配，管理根时间轴上为系统级别服务（例如，光标和工具提示）保留的深度。

要使用深度管理器，请调用它的方法。

下面的代码将组件实例 `loader` 放在比 `button` 组件深的位置（在发布的 SWF 文件中，如果这两个组件重叠，则 `loader` 将出现在按钮的“下方”）：

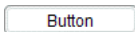
```
loader.setDepthBelow(button);
```



在文档中还可以使用“图层”和“修改”>“排列”菜单选项来管理相对深度。在使用图层和排列进行运行时深度管理时，组件与影片剪辑遵守相同的规则。

“实时预览”中的组件

使用默认启用的“实时预览”功能，可以在舞台上查看组件将在发布的 Flash 内容中出现的近似大小和外观。实时预览反映了不同组件的不同参数。有关实时预览中反映的组件参数的信息，请参阅《组件语言参考》中的各个组件条目。



启用“实时预览”的 Button 组件



禁用“实时预览”的 Button 组件

“实时预览”中的组件不可操作。要测试组件功能，可以使用“控制”>“测试影片”命令。

打开或关闭“实时预览”：

- 选择“控制”>“启用实时预览”。如果该选项旁边出现一个复选标记，表明已经启用了这项功能。

配合使用预加载器和组件

预加载过程会在用户开始与 SWF 文件交互前加载该文件中的某些数据。默认情况下，组件和类被设置为导出到包含组件的文档的第一帧。由于组件和类是首选加载的数据，因此在实现进度栏或加载动画时可能会出现问题。具体来说，组件和类可能会在进度栏出现之前开始加载，但您可能希望进度栏能反映所有数据（包括类）的加载进度。因此，您应在加载 SWF 文件的其它部分之后，但在使用组件之前加载类。

为此，当您为包含组件的应用程序创建自定义预加载器时，应对文件的发布设置进行设置，将所有类导出到包含组件的帧。若要查看“光晕”和“范例”主题（它们的资源设置为“在第一帧导出”）中所有组件的列表，请参阅第 97 页的“更改导出设置”。

若要更改所有类的导出帧，请执行以下操作：

1. 选择“文件” > “发布设置”。
2. 在“发布设置”对话框的“Flash”选项卡中，确保 ActionScript 版本设置为“ActionScript 2.0”。
3. 单击 ActionScript 版本右侧的“设置”按钮。
4. 在“ActionScript 2.0 设置”中，将“导出用于类的帧”文本框中的编号更改为组件最先出现的帧。

这样，在播放头到达所选的要加载类的帧之前，将无法使用任何类。由于组件需要类来实现其功能，因此只能在指定的、用于加载类的帧之后加载组件。如果将类导出到第 3 帧，则在播放头到达第 3 帧并加载数据之前，将无法使用这些类。

如果要预加载使用组件的文件，则还必须在 SWF 文件中预加载这些组件。要完成此操作，必须对这些组件进行设置，以便为 SWF 文件中的其它帧导出这些组件。

若要更改将组件导出到的帧，请执行以下操作：

1. 选择“窗口” > “库”以打开“库”面板。
2. 右键单击 (Windows) 或按住 Control 键单击 (Macintosh) 库中的组件。
3. 从上下文菜单中选择“链接”。
4. 取消选择“在第一帧导出”。
5. 单击“确定”。
6. 选择“文件” > “发布设置”。
7. 选择“Flash”选项卡，然后单击“设置”按钮。
8. 在“导出用于类的帧”中输入编号，然后单击“确定”。类将加载到此帧中。

9. 单击“确定”以关闭“发布设置”对话框。

如果组件不在第一帧加载，则可以为 SWF 文件的第一帧创建一个自定义进度栏。在为第 7 步中指定的帧加载类之前，不要在 **ActionScript** 中引用任何组件，也不要舞台上包含任何组件。



组件必须在其所使用的 **ActionScript** 类之后导出。

关于加载组件

如果将第 2 版组件加载到 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;
```

如果要通过调用 `loadMovie()` 来使用影片剪辑，请添加以下代码：

```
myMovieClip._lockroot = true;
```

如果在加载器影片剪辑中没有将 `_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。有关此属性的信息，请参阅《**ActionScript 2.0** 语言参考》中的“`_lockroot` (`MovieClip._lockroot` 属性)”。

将第 1 版组件升级到第 2 版的体系结构

第 2 版组件的编写符合多种 Web 标准（与事件 [www.w3.org/TR/DOM-Level-3-Events/events.html]、样式、`getter/setter` 策略等有关的标准），它们与 **Macromedia Flash MX** 以及在 **Macromedia Flash MX 2004** 之前发布的 **DRK** 中的第 1 版组件相比，有很大不同。第 2 版组件具有不同的 API，并且是用 **ActionScript 2.0** 编写的。因此，在应用程序中同时使用第 1 版组件和第 2 版组件可能会导致不可预料的情况。有关升级第 1 版组件以使用第 2 版的事件处理、样式和 `getter/setter` 来访问属性（而不是方法）的信息，请参阅第 105 页的第 6 章“创建组件”。

为 **Flash Player 6** 或 **Flash Player 6 (6.0.65.0)** 发布时，包含第 1 版组件的 **Flash** 应用程序在 **Flash Player 6** 和 **Flash Player 7** 中能正常工作。如果要升级应用程序，以便为 **Flash Player 7** 发布时能正常工作，必须将代码转换为使用严格数据类型指定。有关详细信息，请参阅《学习 **Flash** 中的 **ActionScript 2.0**》中的“编写自定义类文件”。

处理组件事件

每个组件都具有当用户与之交互时广播的事件（例如，click 和 change 事件）或当组件发生重要事情时广播的事件（例如，load 事件）。若要处理事件，您需要编写在该事件被触发时需要执行的 **ActionScript** 代码。

每个组件广播各自的一组事件。这些事件包括该组件继承的所有类的所有事件。这意味着除媒体组件外，所有其它组件均从 **UIObject** 和 **UIComponent** 类继承事件，因为它们是该第 2 版结构的基类。要查看组件广播的事件的列表，请参阅《组件语言参考》中该组件的条目及其祖先类的条目。

本章将使用一个简单的 **Macromedia Flash** 应用程序 **TipCalculator** 的几个版本来讲解如何处理组件事件。**Flash** 将 **FLA** 和 **SWF** 文件安装到以下位置：

- 在 **Windows** 中：**C:\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\TipCalculator** 文件夹。
- 在 **Macintosh** 上：**HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/TipCalculator** 文件夹。

本章包含以下各节：

使用侦听器处理事件	56
委托事件	63
关于事件对象	66
使用 on() 事件处理函数	67

使用侦听器处理事件

第 2 版组件结构具有广播器 / 侦听器事件模型。（“广播器”有时也称为“发送程序”。）了解以下有关此模型的要点非常重要：

- 所有事件均由组件类的实例广播。（组件实例是“广播器”。）
- “侦听器”可以是一个函数或一个对象。如果侦听器是一个对象，则必须在该对象上定义一个回调函数。侦听器处理事件；这意味着在事件发生时执行回调函数。
- 要向广播器注册侦听器，请从广播器调用 `addEventListener()` 方法。使用以下语法：

```
componentInstance.addEventListener("eventName",  
    listenerObjectORFunction);
```
- 您可以向一个组件实例注册多个侦听器。

```
myButton.addEventListener("click", listener1);  
myButton.addEventListener("click", listener2);
```
- 也可以向多个组件实例注册一个侦听器。

```
myButton.addEventListener("click", listener1);  
myButton2.addEventListener("click", listener1);
```
- 将一个事件对象传递给处理函数。
您可以使用函数体内的事件对象检索有关事件类型及广播该事件的实例的信息。请参阅第 66 页的“关于事件对象”。
- 在使用 `EventDispatcher.removeEventListener()` 将一个侦听器对象显式删除之前，该侦听器对象一直处于活动状态。例如：

```
myComponent.removeEventListener("change", ListenerObj);
```

使用侦听器对象

要使用侦听器对象，您可以使用 `this` 关键字指定当前对象作为侦听器、使用应用程序中已有的对象或创建一个新对象。

- 大多数情况下都使用 `this`。
通常，使用当前对象 (`this`) 作为侦听器最容易，因为其范围包含广播事件时需要做出响应的组件。
- 在方便的情况下，请使用现有对象。
例如，在一个 **Flash** 表单应用程序中，如果表单包含对事件做出响应的组件，则可能需要使用该表单作为侦听器对象。请将代码置于表单的时间轴的某一帧上。
- 如果有多个组件正在广播一个事件（例如，`click` 事件）并且您仅希望某些侦听器对象做出响应，则请使用一个新的侦听器对象。

如果使用 `this` 对象，请使用与要处理的事件相同的名称定义一个函数；语法如下：

```
function eventName(evtObj:Object){  
    // 此处是您的代码  
};
```


如果要使用一个新的侦听器对象，则必须创建该对象，定义一个与事件同名的属性，然后将该属性分配给在广播事件时执行的回调函数，如下所示：

```
var listenerObject:Object = new Object();
listenerObject.eventName = function(evtObj:Object){
    // 此处是您的代码
};
```

如果要使用现有对象，请使用与新侦听器对象相同的语法，但不创建新对象，如下所示：

```
existingObject.eventName = function(evtObj:Object){
    // 此处是您的代码
};
```

提示

evtObj 参数是在事件被触发时自动生成，并传递给回调函数的对象。该事件对象的属性包含有关事件的信息。有关详细信息，请参阅第 66 页的“关于事件对象”。

最后，从广播事件的组件实例调用 `addEventListener()` 方法。`addEventListener()` 方法有两个参数：指示事件名称和对侦听器对象的引用的字符串。

```
componentInstance.addEventListener("eventName", listenerObject);
```

以下是整个代码段，您可以复制并粘贴它。请确保用实际值替换所有斜体代码；您可以使用 `listenerObject` 和 `evtObj` 或任何其它合法标识符，但必须将 `eventName` 更改为事件的名称。

```
var listenerObject:Object = new Object();
listenerObject.eventName = function(evtObj:Object){
    // 触发事件时
    // 执行此处的代码
};
componentInstance.addEventListener("eventName", listenerObject);
```

下面的代码段使用 `this` 关键字作为侦听器对象：

```
function eventName(evtObj:Object){
    // 触发事件时
    // 执行此处的代码
}
componentInstance.addEventListener("eventName", this);
```

您可以从任何组件实例调用 `addEventListener()`；该方法以“**mix-in**”的形式添加在每一个源自 `EventDispatcher` 类的组件中。（“**mix-in**”是一个类，它提供能增强另一个类的行为的特定功能。）有关详细信息，请参阅《组件语言参考》中的“`EventDispatcher.addEventListener()`”。

有关组件广播的事件的信息，请参阅《组件语言参考》中该组件的条目。例如，`Button` 组件事件列在 `Button` 组件部分中（即“帮助” > “《组件语言参考》” > “`Button` 组件” > “`Button` 类” > “`Button` 类的事件摘要”）。

若要在 Flash (FLA) 文件中注册侦听器对象，请执行以下操作：

1. 在 Flash 中，选择“文件”>“新建”，然后创建一个新的 Flash 文档。
2. 将 Button 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 **myButton**。
4. 将 TextInput 组件从“组件”面板拖到舞台上。
5. 在属性检查器中，输入实例名称 **myText**。
6. 在时间轴中选择第 1 帧。
7. 选择“窗口”>“动作”。
8. 在“动作”面板中输入以下代码：

```
var myButton:mx.controls.Button;  
var myText:mx.controls.TextInput;
```

```
function click(evt){  
    myText.text = evt.target;  
}
```

```
myButton.addEventListener("click", this);
```

事件对象 evt 的 target 属性是对广播事件的实例的引用。此代码在 TextInput 组件中显示 target 属性的值。

在类 (AS) 文件中注册侦听器对象：

1. 从第 43 页的“使用组件”中指定的位置打开文件 TipCalculator.fla。
2. 从第 43 页的“使用组件”中指定的位置打开文件 TipCalculator.as。
3. 在 FLA 文件中选择 form1，在“属性”检查器中查看类名（即 TipCalculator）。
这是表单和类文件之间的链接。此应用程序的所有代码都在 TipCalculator.as 文件中。
表单的属性和行为默认由分配给该表单的类定义。
4. 在 AS 文件中，滚动到第 25 行，public function onLoad():Void。
当表单加载到 Flash Player 时，将执行 onLoad() 函数。在函数体内，subtotal TextInput 实例和三个 RadioButton 实例（percentRadio15、percentRadio18 和 percentRadio20）调用 addEventListener() 方法向一个事件注册一个侦听器。
5. 请查看第 27 行，subtotal.addEventListener("change", this)。
调用 addEventListener() 时，必须向其传递两个参数。第一个参数是指示所广播的事件的名称的字符串 在本例中为“change”。第二个参数是对处理事件的对象或函数的引用。在本例中，该参数是引用类文件的一个实例（一个对象）的关键字 this。然后，Flash 会在对象中查找带有该事件名称的函数。
6. 请查看第 63 行，public function change(event:Object):Void。
这是在 subtotal TextInput 实例发生更改时执行的函数。
7. 选择 TipCalculator.fla，然后选择“控制”>“测试影片”对该文件进行测试。

使用 handleEvent 回调函数

您也可以使用支持 `handleEvent` 函数的侦听器对象。不论所广播的事件的名称是什么，都会调用侦听器对象的 `handleEvent` 方法。您必须使用 `if..else` 或 `switch` 语句来处理多个事件。例如，以下代码使用 `if..else` 语句处理 `click` 和 `change` 事件：

```
// 定义 handleEvent 函数
// 将 evt 作为事件对象参数传递给它

function handleEvent(evt){
    // 检查事件是否为 click
    if (evt.type == "click"){
        // 如果事件为 click，则执行某些操作
    } else if (evt.type == "change"){
        // 如果事件为 change，则执行其它操作
    }
};

// 将侦听器对象注册到
// 两个不同的组件实例
// 因为该函数定义在
// "this" 对象上，所以侦听器为 this。

instance.addEventListener("click", this);
instance2.addEventListener("change", this);
```

使用侦听器函数

与 `handleEvent` 语法不同，有几个侦听器函数可以处理不同的事件。因此，您可以不在 `myHandler` 中使用 `if` 和 `else if` 检查，而只需为 `change` 事件定义 `myChangeHandler`，为 `scroll` 事件定义 `myScrollHandler`，然后注册它们，如下所示：

```
myList.addEventListener("change", myChangeHandler);
myList.addEventListener("scroll", myScrollHandler);
```

要使用侦听器函数，必须先定义一个函数：

```
function myFunction:Function(evtObj:Object){
    // 此处是您的代码
}
```

提示

`evtObj` 参数是在事件被触发时自动生成并传递给该函数的对象。该事件对象的属性包含有关事件的信息。有关详细信息，请参阅第 66 页的“关于事件对象”。

然后，从广播事件的组件实例调用 `addEventListener()` 方法。`addEventListener()` 方法有两个参数：一个指示事件名称的字符串以及一个对该函数的引用。

```
componentInstance.addEventListener("eventName", myFunction);
```

您可以从任何组件实例调用 `addEventListener()`；它包含在每一个源自 `EventDispatcher` 类的 UI 组件中。有关详细信息，请参阅 `EventDispatcher.addEventListener()`。

有关组件广播的事件的信息，请参阅《组件语言参考》中每个组件的条目。

若要在 Flash (FLA) 文件中注册侦听器对象，请执行以下操作：

1. 在 Flash 中，选择“文件” > “新建”，然后创建一个新的 Flash 文档。
2. 将 List 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 **myList**。
4. 在时间轴中选择第 1 帧。
5. 选择“窗口” > “动作”。
6. 在“动作”面板中输入以下代码：

```
// 声明变量
var myList:mx.controls.List;
var myHandler:Function;

// 将项目添加到列表
myList.addItem("Bird");
myList.addItem("Dog");
myList.addItem("Fish");
myList.addItem("Cat");
myList.addItem("Ape");
myList.addItem("Monkey");

// 定义 myHandler 函数
function myHandler(eventObj:Object){


    // 使用 eventObj 参数
    // 捕获事件类型
    if (eventObj.type == "change"){
        trace("The list changed");
    } else if (eventObj.type == "scroll"){
        trace("The list was scrolled");
    }
}

// 向 myList 注册 myHandler 函数。
// 当选中一个项目（触发 change 事件）或
// 滚动列表时，执行 myHandler。
myList.addEventListener("change", myHandler);
myList.addEventListener("scroll", myHandler);
```



事件对象 `evt` 的 `type` 属性是对事件名称的引用。

7. 选择“控制” > “测试影片”；然后在列表中选择一项，并滚动列表以在“输出”面板中查看结果。



在侦听器函数中，关键字 `this` 引用的是调用 `addEventListener()` 的组件实例，而不是在其中定义该函数的时间轴或类。但是，您可以使用 `Delegate` 类将侦听器函数委托到不同的范围。请参阅第 63 页的“委托事件”。要查看函数范围的示例，请参阅下一部分。

关于侦听器中的范围

范围指的是一种对象，函数在该对象中执行。该函数内的任何变量引用都被识别为该对象的属性。您可以使用 `Delegate` 类指定侦听器的范围。有关详细信息，请参阅第 63 页的“委托事件”。

如上所述，可以通过调用 `addEventListener()` 向组件实例注册侦听器。此方法采用两个参数：一个是表示事件名称的字符串，另一个是对对象或函数的引用。下表列出了每个参数类型的范围：

侦听器类型	范围
对象	侦听器对象。
函数	广播事件的组件实例。

如果向 `addEventListener()` 传递一个对象，则会在该对象范围内调用指定给该对象的回调函数（或在该对象上定义的函数）。这意味着关键字 `this` 在回调函数内使用时是引用侦听器对象，如下所示：

```
var lo:Object = new Object();
lo.click = function(evt){
    // this 引用对象 lo
    trace(this);
}
myButton.addEventListener("click", lo);
```

但是，如果向 `addEventListener()` 传递一个函数，则会在调用 `addEventListener()` 的组件实例的范围内调用该函数。这意味着关键字 `this` 在该函数内使用时引用的是正在广播的组件实例。如果在类文件中定义该函数，这就会引发一个问题。您将无法通过预期路径访问该类文件的属性和方法，因为 `this` 并不指向该类的实例。要避免此问题，可使用 `Delegate` 类来将函数委托到正确的范围。请参阅第 63 页的“委托事件”。

下面的代码说明了在类文件中传递给 `addEventListener()` 时函数的范围。要使用此代码，请将它复制到名为 **Cart.as** 的 **ActionScript (AS)** 文件中。使用一个 **Button** 组件 (`myButton`) 和一个 **DataGrid** 组件 (`myGrid`) 创建一个 **Flash (FLA)** 文件。在舞台上选择这两个组件，然后按 **F8** 键将它们转换成名为 `Cart` 的新元件。在 `Cart` 元件的 **Linkage** 属性中，为该元件分配类 `Cart`。

```
class Cart extends MovieClip {

    var myButton:mx.controls.Button;
    var myGrid:mx.controls.DataGrid;

    function myHandler(eventObj:Object){

        // 使用 eventObj 参数
        // 捕获事件类型。
        if (eventObj.type == "click"){

            /* 将 this 的值发送到“输出”面板中。
            因为 myHandler 函数并非在侦听器对象上
            定义，所以 this 是对 myHandler 所
            注册到的组件实例 (myButton)
            的引用。同时，因为 this 并不引用
            Cart 类的实例，所以 myGrid 未定义。
            */
            trace("this: " + this);
            trace("myGrid: " + myGrid);
        }
    }

    // 向 myButton 注册 myHandler 函数
    // 单击该按钮时，myHandler 将执行

    function onLoad():Void{
        myButton.addEventListener("click", myHandler);
    }
}
```

委托事件

您可以将 **Delegate** 类导入到脚本或类中，以便将事件委托给特定的范围和函数（请参阅《组件语言参考》中的“**Delegate 类**”）。要导入 **Delegate** 类，请使用以下语法：

```
import mx.utils.Delegate;
compInstance.addEventListener("eventName", Delegate.create(scopeObject,
    function));
```

scopeObject 参数指定一个范围，指定的 *function* 参数在该范围内被调用。

调用 `Delegate.create()` 有两个常见用途：

- 向两个不同的函数发送同一事件。

请参阅下一部分。

- 在包含类的范围内调用函数。

将函数作为参数传递给 `addEventListener()` 时，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内调用。请参阅第 65 页的“[委托函数的范围](#)”。

将事件委托给函数

如果有两个广播同名事件的组件，则调用 `Delegate.create()` 十分有用。例如，如果您有一个复选框和一个按钮，为了确定哪个组件正在广播 `click` 事件，则您将必须对从 `eventObject.target` 属性中获取的信息使用 `switch` 语句。

要使用下面的代码，请将名为 `myCheckBox_chb` 的复选框和名为 `myButton_btn` 的按钮放置到舞台上。选择这两个实例，再按 **F8** 键创建一个新元件。如果对话框处于基本模式，请单击“高级”，然后选择“为 **ActionScript** 导出”。在“**AS 2.0 类**”文本框中输入 **Cart**。在属性检查器中，按您的意愿设置新元件的名称。现在该元件与 **Cart** 类相关联，元件的实例变成该类的实例。

```
import mx.controls.Button;
import mx.controls.CheckBox;

class Cart {
    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;

    function onLoad() {
        myCheckBox_chb.addEventListener("click", this);
        myButton_btn.addEventListener("click", this);
    }

    function click(eventObj:Object) {
        switch(eventObj.target) {
            case myButton_btn:
                // 将广播器实例名称和
```

```

        // 事件类型发送到“输出”面板
        trace(eventObj.target + ": " + eventObj.type);
        break;
    case myCheckBox_chb:
        trace(eventObj.target + ": " + eventObj.type);
        break;
    }
}
}

```

以下代码是为使用 **Delegate** 而经过修改的同一类文件 (**Cart.as**) 中的代码:

```

import mx.utils.Delegate;
import mx.controls.Button;
import mx.controls.CheckBox;

class Cart {
    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;

    function onLoad() {
        myCheckBox_chb.addEventListener("click", Delegate.create(this,
            chb_onClick));
        myButton_btn.addEventListener("click", Delegate.create(this,
            btn_onClick));
    }

    // 两个单独的函数处理事件

    function chb_onClick(eventObj:Object) {
        // 将广播器实例名称和
        // 事件类型发送到“输出”面板
        trace(eventObj.target + ": " + eventObj.type);
        // 将 FLA 文件中与 Cart 类
        // 相关联的元件的绝对路径
        // 发送到“输出”面板
        trace(this)
    }

    function btn_onClick(eventObj:Object) {
        trace(eventObj.target + ": " + eventObj.type);
    }
}

```


委托函数的范围

`addEventListener()` 方法有两个参数：一个事件的名称和一个对侦听器的引用。侦听器可以是一个对象或一个函数。如果传递一个对象，则会在该对象的范围内调用分配给该对象的回调函数。但是，如果传递一个函数，则会在调用 `addEventListener()` 的组件实例的范围内调用该函数。（有关详细信息，请参阅第 61 页的“关于侦听器中的范围”。）

因为是在广播器实例的范围内调用函数，所以函数体内的关键字 `this` 指向的是广播器实例，而不是包含该函数的类。因此，您无法访问包含该函数的类的属性和方法。请使用 **Delegate** 类将函数的范围委托到包含类，以便可以访问包含类的属性和方法。

下面的示例对 **Cart.as** 类文件的变体使用与上一部分相同的方法：

```
import mx.controls.Button;
import mx.controls.CheckBox;

class Cart {

    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;

    // 定义一个变量以从
    // chb_onClick 函数访问
    var i:Number = 10

    function onLoad() {
        myCheckBox_chb.addEventListener("click", chb_onClick);
    }

    function chb_onClick(eventObj:Object) {
        // 您期望可以访问
        // 变量 i 并输出 10。
        // 但是，这将未定义的值发送
        // 到“输出”面板，因为
        // 该函数的范围并不是
        // 定义 i 的 Cart 实例。
        trace(i);
    }
}
```

要访问 **Cart** 类的属性和方法，请调用 `Delegate.create()` 作为 `addEventListener()` 的第二个参数，如下所示：

```
import mx.utils.Delegate;
import mx.controls.Button;
import mx.controls.CheckBox;

class Cart {
    var myCheckBox_chb:CheckBox;
    var myButton_btn:Button;
```

```

// 定义一个变量以从
// chb_onClick 函数访问
var i:Number = 10

function onLoad() {
    myCheckBox_chb.addEventListener("click", Delegate.create(this,
chb_onClick));
}

function chb_onClick(eventObj:Object) {
    // 将 10 发送到“输出”面板
    // 因为函数的范围是
    // Cart 实例
    trace(i);
}
}

```

关于事件对象

事件对象是 **ActionScript Object** 类的一个实例；它具有包含有关事件的信息的下列属性。

属性	说明
type	指示事件名称的字符串。
target	对广播事件的组件实例的引用。

当事件具有附加属性时，会列在“组件字典”中该事件的条目内。

事件对象在触发事件时自动生成，并被传递到侦听器对象的回调函数或侦听器函数。

您可以在该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。您可以从实例名称访问其它组件属性。例如，下列代码使用 `evtObj` 事件对象的 `target` 属性来访问 `myButton` 实例的 `label` 属性，并将该属性的值发送到“输出”面板：

```

var myButton:mx.controls.Button;
var listener:Object;

listener = new Object();

listener.click = function(evtObj){
    trace("The " + evtObj.target.label + " button was clicked");
}

myButton.addEventListener("click", listener);

```

使用 on() 事件处理函数

您可以将 on() 事件处理函数分配给组件实例，就像将处理函数分配给按钮或影片剪辑一样。对于简单的测试来说，on() 事件处理函数可能很有用，但是对于所有应用程序，请改用事件侦听器。有关详细信息，请参阅第 56 页的“使用侦听器处理事件”。

在直接附加到组件（分配给“动作”面板中的组件实例）的 on() 处理函数中使用关键字 this 时，this 引用该组件实例。例如，以下直接附加到 **Button** 组件实例 myButton 的代码在“输出”面板中显示“_level0.myButton”：

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

使用 on() 事件处理函数：

1. 将用户界面组件拖到舞台上。
例如，将 **Button** 组件拖到舞台上。
2. 在舞台上，选择该组件，然后打开“动作”面板。
3. 以下列格式将 on() 处理函数添加到“动作”面板：

```
on(event){  
    //your statements go here  
}
```

例如：

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

当 on() 处理函数的事件（在本例中，为按钮单击）发生时，Flash 会运行 on() 处理函数中的代码。

4. 选择“控制” > “测试影片”，然后单击该按钮查看输出内容。

自定义组件

在不同的应用程序中使用组件时，您可能需要更改组件的外观。您可以单独或结合使用以下三种方法来自定义组件的外观：

样式 用户界面 (UI) 组件具有设置组件某些方面外观的样式属性。每个组件都有其自己的可修改样式属性集，设置样式并不能更改组件的所有可视外观。有关详细信息，请参阅第 69 页的“使用样式自定义组件的颜色和文本”。

外观 外观 由构成组件图形显示的元件集合组成。设置外观是通过修改或替换组件的源图形来更改组件外观的过程。外观可以是一小部分（例如，边框的边缘或角），也可以是合成的部分（例如，呈弹起（尚未被按下）状态的按钮的整幅图片）。外观还可以是不带图形的元件，它包含绘制一部分组件的代码。组件的某些不能通过样式属性来设置的方面可以通过修改外观来设置。有关详细信息，请参阅第 81 页的“关于设置组件外观”。

主题 主题是样式和外观两者的集合，您可以将主题另存为 FLA 文件并应用于其它文档。有关详细信息，请参阅第 91 页的“关于主题”。

本章包含下列各节：

使用样式自定义组件的颜色和文本.....	69
关于设置组件外观.....	81
关于主题.....	91
通过组合外观与样式设置来自定义组件.....	100

使用样式自定义组件的颜色和文本

Flash 为每个 UI 组件都提供了您可以编辑的样式属性。在每个特定组件的文档中，您将会看到一个列出该组件的可修改样式的表（例如，您可以在《组件语言参考》的“对 Accordion 组件使用样式”中看到 Accordion 组件的样式表）。此外，UI 组件继承 UIObject 类的 setStyle() 和 getStyle() 方法（请参阅 UIObject.setStyle() 和 UIObject.getStyle()）。对于一个组件实例，您可以使用 setStyle() 和 getStyle() 方法来设置和获取样式属性值，如后面的第 71 页的“在组件实例上设置样式”中所示。

⚠

您不能设置介质组件的样式。

使用样式声明和主题

从更广泛的意义上来说，样式在样式声明 中进行组织，您可以在声明中跨越多个组件实例来控制样式属性值。样式声明是由 `CSSStyleDeclaration` 类创建的对象，它的属性是您可以指定给组件的样式设置。`ActionScript` 中的样式声明模仿“层叠样式表”（CSS）影响 HTML 页面的方式进行建模。对于 HTML 页面，您可以创建一个样式表文件，用以定义一组 HTML 页面中内容的样式属性。对于组件，您可以创建一个样式声明对象，并将样式属性添加到该样式声明对象以控制组件在 Flash 文档中的外观。

而且，样式声明在主题 中组织。Flash 为组件提供了两种可视主题：“光晕”（`HaloTheme.fla`）和“范例”（`SampleTheme.fla`）。“主题”是控制文档中组件外观的一组样式和图形。每个主题都为组件提供一些样式。每个组件使用的样式部分取决于文档使用的主题。某些样式（如 `defaultIcon`）用于与之相关联的组件，与文档应用的主题无关。而其它样式（如 `themeColor` 和 `symbolBackgroundColor`）仅在使用了相应的主题时才用于组件。例如，只有在使用“光晕”主题时才使用 `themeColor`；只有在使用“范例”主题时才使用 `symbolBackgroundColor`。若要确定可以为组件设置哪些样式属性，您必须知道指定给该组件的主题。《组件语言参考》中每个组件的样式表指示每个样式属性是应用于一个还是两个提供的主题。（有关详细信息，请参阅第 91 页的“关于主题”。）

理解样式设置

在使用样式或样式声明时，您会注意到您能以不同的方式（在全局、主题、类、样式声明或样式属性级别）设置样式。而且，某些样式属性可能会继承父组件（例如，`Accordion` 子面板可能会继承 `Accordion` 组件的字体处理）。以下是关于样式行为的几个要点：

主题相关性 对于特定的组件，您可以设置的样式属性取决于当前的主题。默认情况下，Flash 组件设计为使用“光晕”主题，但 Flash 还提供了“范例”主题。因此，当您查看样式属性表（如《组件语言参考》的“对 `Button` 组件使用样式”中 `Button` 组件的样式属性表）时，请注意哪个主题支持您想要的样式。该表将指示“光晕”、“范例”或“两者都是”（意思是两个主题都支持该样式属性）。要更改当前主题，请参阅第 91 页的“切换主题”。

继承 您不能在 `ActionScript` 中设置继承。组件子级设计为可以继承或不继承父组件的样式。

全局样式表 Flash 中的样式声明不支持 Flash 文档的“层叠”，这一点与 CSS 支持 HTML 文档的层叠不同。所有样式表声明对象都在应用程序级别（全局）定义。

优先级 如果以多种方式设置一个组件样式（例如，如果在全局级别和在组件实例级别设置 `textColor`），则 Flash 会根据第 77 页的“在同一个文档中使用全局、自定义和类样式”中列出的顺序使用它遇到的第一个样式。

设置样式

样式属性的存在、在样式声明中组织样式属性以及在主题中更广泛地组织样式声明和图形，使您能够以下列方式自定义一个组件：

- 在组件实例上设置样式。
您可以更改一个组件实例的颜色和文本属性。这种方式在有些情况下是有效的，但是，如果您需要在文档中设置所有组件上的各项属性，这种方式就会很耗时。
有关详细信息，请参阅第 71 页的“在组件实例上设置样式”。
- 调整为文档中的所有组件设置样式的全局样式声明。
如果要对整个文档应用一致的外观，可以在全局样式声明上创建样式。
有关详细信息，请参阅第 73 页的“设置全局样式”。
- 创建自定义样式声明，并将它们应用到几个组件实例。
您可能要让文档中成组的组件共享样式。为此，可创建应用到指定组件的自定义样式声明。
有关详细信息，请参阅第 73 页的“为成组的组件设置自定义样式”。
- 创建默认的类型样式声明。
您可以定义默认的类型样式声明，以使类的每个实例共享一个默认外观。
有关详细信息，请参阅第 75 页的“为组件类设置样式”。
- 使用继承样式来设置文档的某一部分中的组件的样式。
容器组件将继承在容器上设置的样式属性的值。
有关详细信息，请参阅第 75 页的“设置容器上的继承样式”。

在使用“实时预览”功能查看舞台上的组件时，Flash 并不会显示对样式属性所做的更改。
有关详细信息，请参阅第 52 页的““实时预览”中的组件”。

在组件实例上设置样式

您可以编写 **ActionScript** 代码，在任何组件实例上设置和获取样式属性。

`UIObject.setStyle()` 和 `UIObject.getStyle()` 方法均可从任何 **UI** 组件直接调用。以下语法为组件实例指定属性和值：

```
instanceName.setStyle("propertyName", value);
```

例如，以下代码为使用“光晕”主题的名为 `myButton` 的 **Button** 实例设置强调颜色：

```
myButton.setStyle("themeColor", "haloBlue");
```



如果该值是字符串，它必须括在引号中。

您还可以属性的形式直接访问样式（例如 `myButton.color = 0xFF00FF`）。

在组件实例上通过 `setStyle()` 设置的样式属性具有最高优先级，可以覆盖基于样式声明或主题的所有其它样式设置。然而，您对一个组件实例使用 `setStyle()` 设置的属性越多，组件在运行时呈现得越慢。利用 **ActionScript** 可以加速呈现自定义组件，方法是在使用《组件语言参考》中的 `UIObject.createClassObject()` 创建组件实例过程中定义样式属性，并将样式设置放置在 `initObject` 参数中。例如，对于当前文档库中的 **ComboBox** 组件，以下代码创建一个名为 `my_cb` 的 combo 框实例，并将该 combo 框中的文本设置为斜体且右对齐：

```
createClassObject(mx.controls.ComboBox, "my_cb", 1, {fontStyle:"italic",
    textAlign:"right"});
my_cb.addItem({data:1, label:"One"});
```

提醒

如果想更改多个属性，或更改多个组件实例的属性，您可以创建一个自定义样式。将一个自定义样式用于多个属性的组件实例要比多次调用 `setStyle()` 的组件实例呈现得更快。有关详细信息，请参阅第 73 页的“[为成组的组件设置自定义样式](#)”。

设置或更改使用“光晕”主题的单个组件实例的属性：

1. 在舞台上选择组件实例。
2. 在“属性”检查器中，为其指定实例名称 **myComponent**。
3. 打开“动作”面板并选择“场景 1”，然后选择“第 1 层：第 1 帧”。
4. 输入以下代码，将实例更改为橙色：

```
myComponent.setStyle("themeColor", "haloOrange");
```

5. 选择“控制” > “测试影片”以查看所做的更改。

有关某个特定组件所支持的样式的列表，请参阅《组件语言参考》中该组件的条目。

使用 ActionScript 创建一个组件实例并同时设置多个属性：

1. 将一个组件拖到库中。
2. 打开“动作”面板并选择“场景 1”，然后选择“第 1 层：第 1 帧”。
3. 输入以下语法以创建一个组件实例并设置其属性：

```
createClassObject(className, "instance_name", depth, {style:"setting",
    style:"setting"});
```

例如，对于库中的 **Button** 组件，以下 **ActionScript** 在深度 1 处创建一个按钮实例 `my_button`，其文本样式设置为紫色、斜体：

```
createClassObject(mx.controls.Button, "my_button", 1, {label:"Hello",
    color:"0x9900CC", fontStyle:"italic"});
```

有关详细信息，请参阅 `UIObject.createClassObject()`。

4. 选择“控制” > “测试影片”以查看所做的更改。

有关某个特定组件所支持的样式的列表，请参阅《组件语言参考》中该组件的条目。

设置全局样式

默认情况下，在将其它样式声明附加到组件前，所有组件都遵从全局样式声明（如第 73 页的“[为成组的组件设置自定义样式](#)”中所述）。全局样式声明分配给用第 2 版 Macromedia 组件结构构建的所有 Flash 组件。`_global` 对象的样式属性（`_global.style`）是 `CSSStyleDeclaration` 的一个实例，并且用作全局样式声明。如果在全局样式声明中更改样式属性的值，该更改将应用到 Flash 文档中的所有组件。

注意

一些样式是在组件类的 `CSSStyleDeclaration` 实例上设置的（例如，`TextArea` 和 `TextInput` 组件的 `backgroundColor` 样式）。因为在确定样式值时类样式声明优先于全局样式声明，所以在全局样式声明上设置 `backgroundColor` 对 `TextArea` 和 `TextInput` 没有影响。有关样式优先级的详细信息，请参阅第 77 页的“[在同一个文档中使用全局、自定义和类样式](#)”。有关编辑组件类的 `CSSStyleDeclaration` 的详细信息，请参阅第 75 页的“[为组件类设置样式](#)”。

更改全局样式声明中的一个或多个属性：

1. 确保该文档包含至少一个组件实例。
有关详细信息，请参阅第 44 页的“[向 Flash 文档中添加组件](#)”。
2. 在出现组件（或出现组件之后）的时间轴上选择帧。
3. 在“动作”面板中，使用如下代码来更改全局样式声明上的属性。只需列出要更改其值的属性，如下所示：

```
_global.style.setStyle("color", 0xCC6699);  
_global.style.setStyle("themeColor", "haloBlue")  
_global.style.setStyle("fontSize", 16);  
_global.style.setStyle("fontFamily", "_serif");
```

4. 选择“控制” > “测试影片”以查看所做的更改。

为成组的组件设置自定义样式

您可以创建自定义样式声明，从而为 Flash 文档中成组的组件指定一组独特的属性。除了 `_global` 对象的 `style` 属性（在第 73 页的“[设置全局样式](#)”中讨论，该属性确定整个 Flash 文档的默认样式声明）外，`_global` 对象还有一个 `styles` 属性，该属性是可用的自定义样式声明的列表。这样，您可以创建一个样式声明，作为 `CSSStyleDeclaration` 对象的一个新实例，为其指定一个自定义样式名称，然后将它放置在 `_global.styles` 列表中。然后，指定该样式的属性和值，并将该样式名称指定应具有相同外观的组件实例。

注意，在将样式名称指定给组件实例时，该组件只对它所支持的样式属性有反应。有关每个组件所支持的样式属性的列表，请参阅《组件语言参考》中的各个组件条目。

要更改自定义样式格式，请使用以下语法：

```
_global.styles.CustomStyleName.setStyle(propertyName, propertyValue);
```

自定义样式设置具有高于类样式设置、继承的样式设置和全局样式设置的优先级。有关样式优先级的列表，请参阅第 77 页的“[在同一个文档中使用全局、自定义和类样式](#)”。

为成组的组件创建自定义样式声明：

1. 在舞台上添加至少一个组件。

有关详细信息，请参阅第 44 页的“向 Flash 文档中添加组件”。

此示例使用三个按钮组件，实例名称分别为 a、b 和 c。如果使用不同组件，请在“属性”检查器中为它们指定实例名称，并在第 8 步使用这些实例名称。

2. 在出现组件（或出现组件之后）的时间轴上选择帧。
3. 打开“动作”面板。
4. 添加下面的 `import` 语句，以便在 `CSSStyleDeclaration` 类中访问构造函数来创建新的样式声明：

```
import mx.styles.CSSStyleDeclaration;
```

5. 使用以下语法创建 `CSSStyleDeclaration` 对象的一个实例，以便定义新的自定义样式格式：

```
var new_style:Object = new CSSStyleDeclaration();
```

6. 在自定义样式声明的 `_global.styles` 列表中，命名您的样式声明，如“`myStyle`”，并标识包含新样式声明的所有属性的对象。

```
_global.styles.myStyle = new_style;
```

7. 使用 `setStyle()` 方法（该方法继承自 `UIObject` 类）将属性添加到 `new_style` 对象，这些属性随即与自定义样式声明 `myStyle` 相关联：

```
new_style.setStyle("fontFamily", "_serif");
new_style.setStyle("fontSize", 14);
new_style.setStyle("fontWeight", "bold");
new_style.setStyle("textDecoration", "underline");
new_style.setStyle("color", 0x666699);
```

8. 在同一“脚本”窗格中，使用以下语法将三个特定组件的 `styleName` 属性设置为自定义样式声明名称：

```
a.setStyle("styleName", "myStyle");
b.setStyle("styleName", "myStyle");
c.setStyle("styleName", "myStyle");
```

您还可以通过声明的全局 `styleName` 属性使用 `setStyle()` 和 `getStyle()` 方法访问自定义样式声明上的样式。例如，以下代码对 `myStyle` 样式声明设置 `backgroundColor` 样式：

```
_global.styles.myStyle.setStyle("themeColor", "haloOrange");
```

但是，由于步骤 5 和 6 将 `new_style` 实例与样式声明相关联，因此您可以使用更短的语法，如 `new_style.setStyle("themeColor", "haloOrange")`。

有关 `setStyle()` 和 `getStyle()` 方法的详细信息，请参阅 `UIObject.setStyle()` 和 `UIObject.getStyle()`。

为组件类设置样式

您可以为组件（**Button**、**CheckBox** 等等）的任何类定义类样式声明，该样式声明设置该类每个实例的默认样式。创建实例之前，必须先创建样式声明。诸如 **TextArea** 和 **TextInput** 之类的一些组件默认情况下预定义了类样式声明，因为必须自定义它们的 `borderStyle` 和 `backgroundColor` 属性。



如果您要替换类样式表，请确保新样式表囊括旧样式表中所有您需要的样式，否则那些样式将会被覆盖。

以下代码首先检查当前主题是否已经有 **CheckBox** 的样式声明，如果没有，则创建一个新的样式声明。然后，代码使用 `setStyle()` 方法以为 **CheckBox** 样式声明定义一个样式属性（这种情况下，“**color**”将所有复选框标签文本的颜色设置为蓝色）：

```
if (_global.styles.CheckBox == undefined) {  
    _global.styles.CheckBox = new mx.styles.CSSStyleDeclaration();  
}  
_global.styles.CheckBox.setStyle("color", 0x0000FF);
```

有关可对 **CheckBox** 组件设置的样式属性表，请参阅《组件语言参考》中的“对 **CheckBox** 组件使用样式”。

自定义样式设置具有高于继承和全局样式设置的优先级。有关样式优先级的列表，请参阅第 77 页的“在同一个文档中使用全局、自定义和类样式”。

设置容器上的继承样式

继承样式 是从文档的 **MovieClip** 层次结构中的父组件继承其值的样式。如果未在实例、自定义或类级别设置文本或颜色样式，则 **Flash** 会在 **MovieClip** 层次结构中搜索该样式值。因此，如果在容器组件上设置样式，容器组件将继承这些样式设置。

以下样式是继承样式：

- `fontFamily`
- `fontSize`
- `fontStyle`
- `fontWeight`
- `textAlign`
- `textIndent`
- 所有单值颜色样式（例如，`themeColor` 是继承样式，而 `alternatingRowColors` 不是）

样式管理器向 **Flash** 告知样式是否继承了它的值。一些附加样式也可以在运行时作为继承样式被添加进去。有关详细信息，请参阅《组件语言参考》中的 [StyleManager](#) 类。



Flash 组件和 HTML 页面在实现层叠样式表时的一个主要不同之处是 Flash 组件不支持 CSS `inherit` 值。组件设计或者继承样式，或者不继承样式。

继承的样式具有高于全局样式的优先级。有关样式优先级的列表，请参阅第 77 页的“[在同一个文档中使用全局、自定义和类样式](#)”。

以下示例演示继承样式如何与 **Accordion** 组件一起使用，该组件在 **Flash Professional 8** 中可用（**Flash Basic 8** 和 **Flash Professional 8** 都支持继承样式功能）。

创建具有由单个 Accordion 窗格中的组件继承的样式的 Accordion 组件：

1. 打开一个新的 FLA 文件。
2. 将 **Accordion** 组件从“组件”面板拖到舞台上。
3. 使用“属性”检查器命名 **Accordion** 组件并设置其大小。对于本示例，为组件指定实例名称 **accordion**。

4. 将 **TextInput** 组件和 **Button** 组件从“组件”面板拖到库中。

通过将组件拖到库中，可使脚本在运行时使用这些组件。

5. 在时间轴的第一帧中添加以下 **ActionScript**：

```
var section1 = accordion.createChild(mx.core.View, "section1", {label:
    "First Section"});
var section2 = accordion.createChild(mx.core.View, "section2", {label:
    "Second Section"});

var input1 = section1.createChild(mx.controls.TextInput, "input1");
var button1 = section1.createChild(mx.controls.Button, "button1");

input1.text = "Text Input";
button1.label = "Button";
button1.move(0, input1.height + 10);

var input2 = section2.createChild(mx.controls.TextInput, "input2");
var button2 = section2.createChild(mx.controls.Button, "button2");

input2.text = "Text Input";
button2.label = "Button";
button2.move(0, input2.height + 10);
```

以上代码将两个子项添加到 **Accordion** 组件，然后为每个子项加载一个 **TextInput** 控件和一个 **Button** 控件，本示例使用这些控件来演示样式继承。

6. 选择“控制”>“测试影片”，在添加样式继承前查看该文档。

7. 在第一帧的脚本结尾处添加以下 **ActionScript**:

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

8. 选择“控制” > “测试影片”以查看所做的更改。

请注意，**Accordion** 组件上的 `fontStyle` 设置不仅影响 **Accordion** 文本本身，还影响 **Accordion** 组件内与 **TextInput** 和 **Button** 组件关联的文本。

在同一个文档中使用全局、自定义和类样式

如果只在文档中的一个位置定义样式，**Flash** 将在需要知道属性值时使用该定义。然而，一个 **Flash** 文档可能有多种样式设置，即直接在组件实例上设置的样式属性、自定义样式声明、默认类样式声明、继承样式以及全局样式声明。在此情况下，**Flash** 按照特定顺序查找所有这些位置的属性定义，以此来确定属性的值。

Flash 按如下顺序查找样式，直到找到一个值：

1. **Flash** 查找组件实例上的样式属性。
2. **Flash** 检查实例的 `styleName` 属性，查看是否为其指定了自定义样式声明。
3. **Flash** 查找默认类样式声明上的属性。
4. 如果样式是继承样式的一种，**Flash** 将彻底检查父层次结构来查找继承的值。
5. **Flash** 查找全局样式声明中的样式。
6. 如果尚未定义该属性，则属性值为 `undefined`。

关于颜色样式属性

颜色样式属性与非颜色样式属性的行为方式不同。所有颜色属性的名称都以“**Color**”结尾，例如 `backgroundColor`、`disabledColor` 和 `color`。更改颜色样式属性时，实例和所有相应子实例中的颜色会立即更改。所有其它样式属性更改只将对象标记为需要重绘，而实际的更改会在下一帧中生效。

颜色样式属性的值可以是数字、字符串或对象。如果该值是数字，它以十六进制数字 (0xRRGGBB) 的形式表示颜色的 RGB 值。如果该值是字符串，它必须为颜色名称。

颜色名称是映射到常用颜色的字符串。可以使用 **Style Manager** 添加新颜色名称（请参阅《组件语言参考》中的 [StyleManager 类](#)）。下表列出默认颜色名称：

颜色名称	值
black（黑色）	0x000000
white（白色）	0xFFFFFFFF
red（红色）	0xFF0000
green（绿色）	0x00FF00
blue（蓝色）	0x0000FF
magenta（洋红色）	0xFF00FF
yellow（黄色）	0xFFFF00
cyan（青色）	0x00FFFF
haloGreen	0x80FF4D
haloBlue	0x2BF5F5
haloOrange	0xFFC200

提醒

如果不定义颜色名称，可能无法正确绘制组件。

您可以使用任何有效的 **ActionScript** 标识符来创建自己的颜色名称（例如 "WindowText" 或 "ButtonText"）。使用样式管理器定义新颜色，如下所示：

```
mx.styles.StyleManager.registerColorName("special_blue", 0x0066ff);
```

大多数组件都不能将对象作为颜色样式属性值进行处理。然而，某些组件可以处理表示渐变或其它颜色组合的颜色对象。有关详细信息，请参阅《组件语言参考》中每个组件条目的“使用样式”部分。

您可以使用类样式声明和颜色名称来方便地控制屏幕上的文本和元件的颜色。例如，如果要提供看起来像 **Microsoft Windows** 的显示配置屏幕，应定义像 ButtonText 和 WindowText 的颜色名称，并定义像 Button、CheckBox 和 Window 的类样式声明。

提醒

某些组件提供的样式属性是一个颜色数组，如 alternatingRowColors。必须将这些样式设置为 RGB 数值（而不是颜色名称）的数组。

自定义组件动画

某些组件（如 **Accordion**、**ComboBox** 和 **Tree** 组件）提供动画来演示组件状态之间的过渡（例如，在 **Accordion** 子项之间切换、扩展组合框下拉列表和扩展或折叠树形文件夹时）。另外，组件提供与项目（如列表中的行）的选择和取消选择有关的动画。

可以通过以下样式来控制这些动画的各个方面：

动画样式	说明
<code>openDuration</code>	在 Accordion 、 ComboBox 和 Tree 组件中打开缓动的过渡时间（以毫秒为单位）。默认值为 250。
<code>openEasing</code>	对控制 Accordion 、 ComboBox 和 Tree 组件中的状态动画的补间函数的引用。默认等式使用正弦输入 / 输出公式。
<code>popupDuration</code>	当打开 Menu 组件中的菜单时，过渡的持续时间（以毫秒为单位）。默认值为 150。但是请注意，动画总是使用默认的正弦输入 / 输出等式。
<code>selectionDuration</code>	在 ComboBox 、 DataGrid 、 List 和 Tree 组件从正常状态到所选状态或从所选状态回到正常状态的过渡的持续时间（以毫秒为单位）。默认值为 200。
<code>selectionEasing</code>	对控制 ComboBox 、 DataGrid 、 List 和 Tree 组件中的所选动画的补间函数的引用。此样式仅适用于从正常状态到所选状态的过渡。默认等式使用正弦输入 / 输出公式。

`mx.transitions.easing` 包提供了六个类来控制缓动：


缓动类	说明
<code>Back</code>	一次在一端或两端扩展到过渡范围之外以提供轻微的溢出效果。
<code>Bounce</code>	完全在过渡范围的一端或两端内提供弹跳效果。弹跳次数与持续时间有关：持续时间越长，弹跳次数越多。
<code>Elastic</code>	提供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。
<code>None</code>	提供从开始到结尾的无任何减速或加速效果的相同的运动。该过渡通常也称为线性过渡。
<code>Regular</code>	在加速效果、减速效果或这两种效果的一端或两端提供更慢的运动。
<code>Strong</code>	在一端或两端提供很慢的运动。此效果类似于 <code>Regular</code> ，但更为显著。

`mx.transitions.easing` 包中的每个类都提供以下三个缓动方法：

缓动方法	说明
<code>easeIn</code>	在过渡的开始提供缓动效果。
<code>easeOut</code>	在过渡的结尾提供缓动效果。
<code>easeInOut</code>	在过渡的开始和结尾提供缓动效果。

因为缓动方法是缓动类的静态方法，所以您根本不必实例化缓动类。在对 `setStyle()` 的调用中使用这些方法，如以下示例所示。

```
import mx.transitions.easing.*;
trace("_global.styles.Accordion = " + _global.styles.Accordion);
_global.styles.Accordion.setStyle("openDuration", 1500);
_global.styles.Accordion.setStyle("openEasing", Bounce.easeOut);
```

 所有过渡都使用的默认等式在上面列出的缓动类中不可用。若要在指定了另一个缓动方法后指定组件应使用默认的缓动方法，请调用 `setStyle("openEasing", null)`。

有关详细信息，请参阅《组件语言参考》中的 [“将缓动方法应用于组件”](#)。

获取样式属性值

若要检索一个样式属性值，请使用 `UIObject.getStyle()`。每一个作为 `UIObject` 子类的组件（包括除 **Media** 组件外的所有第 2 版组件）都继承 `getStyle()` 方法。这意味着可以从任何组件实例调用 `getStyle()`，就像从任何组件实例调用 `setStyle()` 一样。

以下代码获取 `themeColor` 样式的值并将它赋予变量 `oldStyle`：

```
var myCheckBox:mx.controls.CheckBox;
var oldFontSize:Number

oldFontSize = myCheckBox.getStyle("fontSize");
trace(oldFontSize);
```


关于设置组件外观

外观是组件用来显示其外表的影片剪辑元件。大多数外观都包含表示组件外表的形状。某些外观只包含在文档中绘制组件的 **ActionScript** 代码。

第 2 版组件都是经过编译的剪辑，您在库中看不到它们的资源。然而，**Flash** 安装包括那些包含所有组件外观的 **FLA** 文件。这些 **FLA** 文件称为主题。每个主题具有不同的外观和行为，但其中所含的外观具有相同的元件名称和链接标识符。这允许您将主题拖到文档中的舞台上，以此来更改它的外观。您还可以使用主题 **FLA** 文件来编辑组件外观。外观位于每个主题 **FLA** 文件的“库”面板中的 **Themes** 文件夹中。（有关主题的详细信息，请参阅第 91 页的“关于主题”。）

每个组件都由大量外观组成。例如，**ScrollBar** 子组件的向下箭头由四个外观组成：**ScrollDownArrowDisabled**、**ScrollDownArrowDown**、**ScrollDownArrowOver** 和 **ScrollDownArrowUp**。整个 **ScrollBar** 使用 13 种不同的外观元件。

某些组件共享外观：例如，使用滚动条的组件（如 **ComboBox**、**List** 和 **ScrollPane**）共享 **ScrollBar Skins** 文件夹中的外观。您可以通过编辑现有外观和创建新外观来更改组件的外表。

定义每个组件类的 **AS** 文件包含了加载组件的特定外观的代码。每个组件外观都对应于一个外观属性，系统为这些外观属性指定了外观元件的链接标识符。例如，**ScrollBar** 组件向下箭头的按（下）状态具有外观属性名称 **downArrowDownName**。**downArrowDownName** 属性的默认值为“**ScrollDownArrowDown**”，它是主题 **FLA** 文件中外观元件的链接标识符。可以编辑现有外观并将它们应用到所有通过编辑外观元件和保留现有链接标识符来使用外观的组件。可以创建新外观并通过设置组件实例的外观属性将它们应用到特定组件实例。不必编辑组件的 **AS** 文件来更改其外观属性，在文档中创建组件时，您可以将外观属性值传递给组件的构造函数。

“组件字典”中的每个组件条目中都列出了每个组件的外观属性。例如，**Button** 组件的外观属性位于下面位置：《组件语言参考》>“**Button** 组件”>“自定义 **Button** 组件”>“对 **Button** 组件使用外观”。

根据您要执行的操作，选择以下一种方法来为组件设置外观。这些方法按照从易到难的顺序列出。

- 要更改与单个文档中某个特定组件的所有实例关联的外观，可复制并修改单个外观元素。（请参阅第 82 页的“编辑文档中的组件外观”。）
建议初学者使用这种外观设置方法，因为它不需要撰写任何脚本。
- 要用一组新外观替换文档中的所有外观（每一类组件共享相同的外观），可应用主题。（请参阅第 91 页的“关于主题”。）
建议在对所有组件或多个文档应用一致的外观和行为时使用这种外观设置方法。
- 要把外观元素的颜色和样式属性链接起来，可向外观添加 **ActionScript** 代码来将其注册为着色的外观元素。（请参阅第 84 页的“将外观颜色链接到样式”。）

- 要为同一组件的多个实例使用不同的外观，可创建新外观，然后设置外观属性。（请参阅第 83 页的“创建新组件外观”和第 85 页的“将新外观应用到组件”。）
- 要更改子组件中的外观（例如，List 组件中的滚动条），可将该组件分成子类。（请参阅第 86 页的“将新外观应用到子组件”。）
- 要更改无法从主组件直接访问的子组件（例如 ComboBox 组件中的 List 组件）的外观，请替换原型中的外观属性。（请参阅第 89 页的“更改子组件中的外观属性”。）

编辑文档中的组件外观

要编辑与单个文档中某个特定组件的所有实例关联的外观，可将外观元件从主题复制到文档，然后根据需要编辑图形。

下面描述的过程与创建和应用新主题非常相似（请参阅第 91 页的“关于主题”）。主要区别在于此过程描述将元件直接从已在使用的主题复制到单个文档，然后仅编辑所有可用外观的一小部分。如果所有修改都在单个文档中而且只修改几个组件的外观时，适合使用此过程。如果编辑的外观将在多个文档间共享，或者涉及多个组件中的更改，您会发现创建新主题来编辑外观将较为容易。

您可以在 Macromedia 开发人员中心找到有关高级外观设置的文章，网址为 www.macromedia.com/devnet/mx/flash/articles/skinning_2004.html。

编辑文档中的组件外观：

1. 如果已对文档应用了“范例”主题，请跳到第 5 步。
2. 选择“文件” > “导入” > “打开外部库”，然后选择 SampleTheme.fla 文件。
此文件位于应用程序级别的配置文件夹中。要了解此文件在您的操作系统上的确切位置，请参阅第 91 页的“关于主题”。
3. 在主题的“库”面板中，选择 Flash UI Components 2/Themes/MMDefault，然后将文档中任意组件的 Assets 文件夹拖到文档的库中。
例如，把 RadioButton Assets 文件夹拖到 ThemeApply.fla 库中。
4. 如果将单个组件的 Assets 文件夹拖到库中，请确保将每个组件的 Assets 元件都设置为“在第一帧导出”。
例如，RadioButton 组件的 Assets 文件夹名为 RadioButton Assets；它具有一个名为 RadioButtonAssets 的元件，该元件包含所有单个资源元件。如果将 RadioButtonAssets 元件设置为“在第一帧导出”，所有单个资源元件也将在第一帧导出。
5. 双击要修改的任何外观元件，在元件编辑模式下打开元件。
例如，打开 States/RadioButtonFalseDisabled 元件。

6. 修改元件或删除图形，然后创建新图形。

您可能需要选择“视图” > “放大”来增加缩放比例。您在编辑外观时，必须维护注册点，以便正确显示外观。所有编辑过的元件的左上角都必须位于 (0,0)。

例如，将圆的内侧更改为浅灰色。

7. 当编辑完该外观元件之后，单击舞台顶部信息栏左侧的“返回”按钮，返回到文档编辑模式。
8. 重复执行第 5 步至第 7 步，直到编辑完所有要更改的外观。



舞台上的组件“实时预览”不会反映编辑过的主题。

9. 选择“控制” > “测试影片”。

在此示例中，为了看到禁用的 **RadioButton** 的新外观，请确保舞台上有一个 **RadioButton** 实例，并在“动作”面板中将该实例的 `enabled` 属性设置为 `false`。

创建新组件外观

如果要为组件的某个实例使用某个特定外观，而为该组件的另一个实例使用另一种外观，必须打开主题 **FLA** 文件并创建新主题元件。组件的设计便于您为不同实例使用不同外观。

创建新外观：

1. 选择“文件” > “打开”，打开要用作模板的主题 **FLA** 文件。
2. 选择“文件” > “另存为”，然后选择唯一的名称，例如 **MyTheme.fla**。
3. 选择要编辑的外观（在本例中，为 **RadioTrueUp**）。
外观位于 **Themes/MMDefault/Component Assets** 文件夹中（在本例中，为 **Themes/MMDefault/RadioButton Assets/States**）。
4. 从“库”选项菜单中（或右键单击元件）选择“直接复制”，然后为元件指定唯一的名称，如 **MyRadioTrueUp**。
5. 单击“元件属性”对话框中的“高级”，然后选择“为 **ActionScript** 导出”。
自动输入与元件名称匹配的“链接标识符”。
6. 双击库中的新外观，在元件编辑模式下打开该外观。
7. 修改该影片剪辑，或删除它然后创建一个新影片剪辑。

您可能需要选择“视图” > “放大”来增加缩放比例。您在编辑外观时，必须维护注册点，以便正确显示外观。所有编辑过的元件的左上角都必须位于 (0,0)。

8. 当编辑完该外观元件之后，单击舞台顶部信息栏左侧的“返回”按钮，返回到文档编辑模式。
9. 选择“文件”>“保存”，但不关闭 **MyTheme fla**。现在必须创建一个新文档，以在其中将经过编辑的外观应用到组件。

有关详细信息，请参阅第 85 页的“将新外观应用到组件”、第 86 页的“将新外观应用到子组件”或第 89 页的“更改子组件中的外观属性”。



在使用“实时预览”查看舞台上的组件时，Flash 并不会显示对组件外观所做的更改。

将外观颜色链接到样式

使用第 2 版组件框架可轻松地将外观元素中的可视资源链接到在组件上使用外观设置的样式。要将影片剪辑实例注册到样式或将整个外观元素注册到样式，请在外观的时间轴中添加 **ActionScript** 代码来调用

```
mx.skins.ColoredSkinElement.setColorStyle(targetMovieClip, styleName)。
```

将外观链接到样式属性：

1. 如果已对文档应用了“范例”主题，请跳到第 5 步。
2. 选择“文件”>“导入”>“打开外部库”，然后选择 **SampleTheme fla** 文件。
此文件位于应用程序级别的配置文件夹中。要了解此文件在您的操作系统上的确切位置，请参阅第 91 页的“关于主题”。
3. 在主题的“库”面板中，选择 **Flash UI Components 2/Themes/MMDefault**，然后将文档中任意组件的 **Assets** 文件夹拖到文档的库中。
例如，把 **RadioButton Assets** 文件夹拖到目标库中。
4. 如果把单个组件的 **Assets** 文件夹拖到该库中，请确保将每个组件的 **Assets** 元件设置为“在第一帧导出”。
例如，**RadioButton** 组件的 **Assets** 文件夹名为 **RadioButton Assets**；它具有一个名为 **RadioButtonAssets** 的元件，该元件包含所有单个资源元件。如果将 **RadioButtonAssets** 元件设置为“在第一帧导出”，所有单个资源元件也将在第一帧导出。
5. 双击要修改的任何外观元件，在元件编辑模式下打开元件。
例如，打开 **States/RadioButtonDisabled** 元件。
6. 如果要着色的元素是图形元件而不是影片剪辑实例，请使用“修改”>“转换为元件”来将其转换为影片剪辑实例。

对于本示例，将中心图形（图形元件 **RadioShape1** 的实例）更改为影片剪辑元件；然后将其命名为 **Inner Circle**。无需选择“为 ActionScript 导出”。

最好（但不是必须）将新创建的影片剪辑元件移动到正在编辑的组件资源的 **Elements** 文件夹中。

7. 如果在上一步中将图形元件转换为影片剪辑实例，则为该实例指定一个名称，以便在 **ActionScript** 中可以将其作为目标。

对于本示例，将实例命名为 **innerCircle**。

8. 添加 **ActionScript** 代码来将外观元素或其包含的影片剪辑实例注册为着色的外观元素。

例如，将下面的代码添加到外观元素的时间轴。

```
mx.skins.ColoredSkinElement.setColorStyle(innerCircle,  
    "symbolBackgroundDisabledColor");
```

在本示例中，您使用已经与“范例”样式中现有的样式名称相对应的颜色。如果可能，最好使用与正式的层叠样式表标准或由“光晕”和“范例”主题提供的样式相对应的样式名称。

9. 重复执行第 5 步至第 8 步，直到编辑完所有要更改的外观。

对于本实例，对 **RadioTrueDisabled** 外观重复这些步骤，然后删除该图形并将现有的 **Inner Circle** 元件拖到 **RadioTrueDisabled** 外观元素中，而不是将现有图形转换为影片剪辑。

10. 当编辑完该外观元件之后，单击舞台顶部信息栏左侧的“返回”按钮，返回到文档编辑模式。

11. 将该组件的一个实例拖到舞台上。

对于本示例，将两个 **RadioButton** 组件拖到舞台上，并将其中一个设置为选中，为了查看所做的更改，使用 **ActionScript** 将两个组件都设置为禁用。

12. 向文档添加 **ActionScript** 代码，以在组件实例上或全局级别上设置新的样式属性。

对于本示例，在全局级别上设置属性，如下所示：

```
_global.style.setStyle("symbolBackgroundDisabledColor", 0xD9D9D9);
```

13. 选择“控制” > “测试影片”。

将新外观应用到组件

创建完新外观后，必须将它应用到文档中的组件。您可以使用 `createClassObject()` 方法动态创建组件实例，也可以将组件实例手动放置在舞台上。根据您给文档添加组件的方式不同，将外观应用到组件实例的方式有两种。

动态创建组件并应用新外观：

1. 选择“文件” > “新建”，创建新的 **Flash** 文档。
2. 选择“文件” > “保存”，并为该文件指定唯一的名称，例如 **DynamicSkinning fla**。
3. 将组件从“组件”面板拖到库中（包括您编辑了其外观的组件，此例中为 **RadioButton**）。该操作会将元件添加到文档的库中，但并不会在文档中显示它们。

4. 将 `MyRadioTrueUp` 和自定义的任何其它元件从 `MyTheme fla` 拖到 `DynamicSkinning fla` 的库中。

该操作会将元件添加到文档的库中，但并不会在文档中显示它们。

5. 打开“动作”面板，然后在第 1 帧中输入以下代码：

```
import mx.controls.RadioButton;
createClassObject(RadioButton, "myRadio", 0, {trueUpIcon:"MyRadioTrueUp",
    label: "My Radio Button"});
```

6. 选择“控制” > “测试影片”。

手动将组件添加到舞台并应用新外观：

1. 选择“文件” > “新建”，创建新的 Flash 文档。
2. 选择“文件” > “保存”，并为其指定唯一的名称，例如 **ManualSkinning fla**。
3. 将组件从“组件”面板拖到舞台上（包括您编辑了其外观的组件，此例中为 `RadioButton`）。
4. 将 `MyRadioTrueUp` 和自定义的任何其它元件从 `MyTheme fla` 拖到 `ManualSkinning fla` 的库中。

该操作会将元件添加到文档的库中，但并不会在文档中显示它们。

5. 选择舞台上的 `RadioButton` 组件，然后打开“动作”面板。

6. 将以下代码附加到 `RadioButton` 实例：

```
onClipEvent(initialize){
    trueUpIcon = "MyRadioTrueUp";
}
```

7. 选择“控制” > “测试影片”。

将新外观应用到子组件

某些情况下，您可能要修改组件中某个子组件的外观，但无法直接访问外观属性（例如，无法直接修改 `List` 组件中的滚动条外观）。使用以下代码，您可以访问滚动条外观。在运行此代码之后创建的所有滚动条也都会具有新外观。

如果一个组件由子组件构成，则《组件语言参考》的组件条目中会标识这些子组件。

将新外观应用到子组件：

1. 按照第 83 页的“创建新组件外观”中的步骤操作，但编辑滚动条外观。在对于本例，编辑 `ScrollDownArrowDown` 外观并为其指定新名称 **MyScrollDownArrowDown**。
2. 选择“文件” > “新建”，创建新的 Flash 文档。
3. 选择“文件” > “保存”，并为其指定唯一的名称，例如 **SubcomponentProject fla**。
4. 将 `List` 组件从“组件”面板拖到库中。

此操作会将该组件添加到“库”面板，但不会在文档中显示该组件。

5. 将 `MyScrollDownArrowDown` 和任何编辑过的其它元件从 `MyTheme.fla` 拖到 `SubcomponentProject.fla` 的库中。

此操作会将该元件添加到“库”面板，但不会在文档中显示该元件。

6. 请执行以下操作之一：

- 如果要更改文档中的所有滚动条，请在时间轴第 1 帧的“动作”面板上输入以下代码：

```
import mx.controls.List;
import mx.controls.scrollClasses.ScrollBar;
ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
```

然后可以在第 1 帧上输入以下代码，以动态创建列表：

```
createClassObject(List, "myListBox", 0, {dataProvider:
    ["AL","AR","AZ", "CA","HI","ID", "KA","LA","MA"]});
```

或者，也可以将 `List` 组件从库拖到舞台。

- 如果要更改文档中的特定滚动条，请在时间轴第 1 帧的“动作”面板上输入以下代码：

```
import mx.controls.List
import mx.controls.scrollClasses.ScrollBar
var oldName = ScrollBar.prototype.downArrowDownName;
ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
createClassObject(List, "myList1", 0, {dataProvider: ["AL","AR","AZ",
    "CA","HI","ID", "KA","LA","MA"]});
myList1.redraw(true);
ScrollBar.prototype.downArrowDownName = oldName;
```



设置足够多的数据，以便显示滚动条，或者将 `vScrollPolicy` 属性设置为 `true`。

7. 选择“控制” > “测试影片”。

您还可以通过以下方法为文档中的所有组件设置子组件外观：在外观元件的 `#initclip` 部分中对子组件的 `prototype` 对象设置外观属性。

使用 `#initclip` 将经过编辑的外观应用到文档中的所有组件：

1. 按照第 83 页的“创建新组件外观”中的步骤操作，但编辑滚动条外观。对于本示例，编辑 `ScrollDownArrowDown` 外观并为它指定新名称 **`MyScrollDownArrowDown`**。
2. 选择“文件” > “新建”，然后创建新的 Flash 文档。使用唯一名称（如 **`SkinsInitExample.fla`**）保存该文件。
3. 从编辑过的主题库范例的库中选择 `MyScrollDownArrowDown` 元件，将其拖到 `SkinsInitExample.fla` 的库中。

此操作将该元件添加到库中，但不在舞台上显示它。

4. 在 `SkinsInitExample.fla` 库中选择 `MyScrollDownArrowDown`，然后从“库”选项菜单中选择“链接”。

5. 选中“为 ActionScript 导出”复选框。单击“确定”。

“在第一帧导出”应自动选中，如果没有，请选中它。

6. 双击库中的 **MyScrollDownArrowDown**，在元件编辑模式下将其打开。

7. 在 **MyScrollDownArrowDown** 元件的第 1 帧上输入以下代码：

```
#initclip 10
import mx.controls.scrollClasses.ScrollBar;
ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
#endinitclip
```

8. 执行下列操作之一将 **List** 组件添加到文档中：

- 将 **List** 组件从“组件”面板拖到舞台。输入足够多的标签参数，以便垂直滚动条能够显示出来。
- 将 **List** 组件从“组件”面板拖到库中。在 **SkinsInitExample.fla** 主时间轴第 1 帧上输入下列代码：

```
createClassObject(mx.controls.List, "myListBox1", 0, {dataProvider:
    ["AL", "AR", "AZ", "CA", "HI", "ID", "KA", "LA", "MA"]});
```



添加足够多的数据，以便垂直滚动条可以显示出来，或者将 `vScrollPolicy` 设置为 `true`。

下面的示例解释如何为舞台上已有的对象设置外观。本范例仅设置 **List** 滚动条的外观，而不设置任何 **TextArea** 或 **ScrollPane** 滚动条的外观。

使用 #initclip 将经过编辑的外观应用到文档中的特定组件：

1. 按照第 82 页的“编辑文档中的组件外观”中的步骤操作，但编辑滚动条外观。对于本示例，编辑 **ScrollDownArrowDown** 外观并为它指定新名称 **MyScrollDownArrowDown**。
2. 选择“文件” > “新建”，创建 Flash 文档。
3. 选择“文件” > “保存”，并为文件指定唯一的名称，例如 **MyVScrollTest.fla**。
4. 将 **MyScrollDownArrowDown** 从主题库拖到 **MyVScrollTest.fla** 库。
5. 选择“插入” > “新建元件”，并为其指定唯一的名称，例如 **MyVScrollBar**。
6. 选中“为 ActionScript 导出”复选框。单击“确定”。

“在第一帧导出”应自动选中，如果没有，请选中它。

7. 在 **MyVScrollBar** 元件的第 1 帧上输入以下代码：

```
#initclip 10
import MyVScrollBar
Object.registerClass("VScrollBar", MyVScrollBar);
#endinitclip
```

8. 将 **List** 组件从“组件”面板拖到舞台。

9. 在“属性”检查器中，输入足够多的 **Label** 参数，以便显示垂直滚动条。

10. 选择“文件” > “保存”。
11. 选择“文件” > “新建”，然后创建新的 **ActionScript** 文件。
12. 输入以下代码：

```
import mx.controls.VScrollBar
import mx.controls.List
class MyVScrollBar extends VScrollBar{
    function init():Void{
        if (_parent instanceof List){
            downArrowDownName = "MyScrollDownArrowDown";
        }
        super.init();
    }
}
```

13. 选择“文件” > “保存”，然后将此文件保存为 **MyVScrollBar.as**。
14. 单击舞台上的空白区域，然后在“属性”检查器中，单击“发布设置”按钮。
15. 单击“**ActionScript** 版本设置”按钮。
16. 单击“添加新路径” (+) 按钮添加新的类路径，然后选择“目标”按钮浏览到 **MyVScrollBar.as** 文件在硬盘上的位置。
17. 选择“控制” > “测试影片”。

更改子组件中的外观属性

如果组件不直接支持外观变量，您可以创建该组件的子类并替换其外观。例如，**ComboBox** 组件不直接支持设置其下拉列表的外观，因为 **ComboBox** 使用 **List** 组件作为其下拉列表。

如果一个组件由子组件构成，则《组件语言参考》的组件条目中会标识这些子组件。

为子组件设置外观：

1. 按照第 82 页的“[编辑文档中的组件外观](#)”中的步骤操作，但编辑滚动条外观。对于本示例，编辑 **ScrollDownArrowDown** 外观并为它指定新名称 **MyScrollDownArrowDown**。
2. 选择“文件” > “新建”，创建 **Flash** 文档。
3. 选择“文件” > “保存”，并为其指定唯一的名称，例如 **MyComboTest.fla**。
4. 将 **MyScrollDownArrowDown** 从上面的主题库拖到 **MyVScrollTest.fla** 的库中。
此操作会将元件添加到库中，但不会在舞台上显示该零件。
5. 选择“插入” > “新建元件”，并为其指定唯一的名称，例如 **MyComboBox**。
6. 选中“为 **ActionScript** 导出”复选框并单击“确定”。
“在第一帧导出”应自动选中，如果没有，请选中它。

7. 在“动作”面板中，在 MyComboBox 元件的第 1 帧上输入以下代码：

```
#initclip 10
import MyComboBox
Object.registerClass("ComboBox", MyComboBox);
#endinitclip
```

8. 编辑完该元件之后，单击舞台顶部的信息栏左侧的“返回”按钮，返回到文档编辑模式。

9. 将 ComboBox 组件拖动到舞台上。

10. 在“属性”检查器中，输入足够多的 Label 参数，以便显示垂直滚动条。

11. 选择“文件” > “保存”。

12. 选择“文件” > “新建”，然后创建新的 ActionScript 文件。

13. 输入以下代码：

```
import mx.controls.ComboBox
import mx.controls.scrollClasses.ScrollBar
class MyComboBox extends ComboBox{
    function getDropdown():Object{
        var oldName = ScrollBar.prototype.downArrowDownName;
        ScrollBar.prototype.downArrowDownName = "MyScrollDownArrowDown";
        var r = super.getDropdown();
        ScrollBar.prototype.downArrowDownName = oldName;
        return r;
    }
}
```

14. 选择“文件” > “保存”，然后将此文件保存为 **MyComboBox.as**。

15. 返回到文件 MyComboTest fla。

16. 单击舞台上的空白区域，然后在“属性”检查器中，单击“发布设置”按钮。

17. 单击“ActionScript 版本设置”按钮。

18. 单击“添加新路径” (+) 按钮添加新的类路径，然后选择“目标”按钮浏览到 MyComboBox.as 文件在硬盘上的位置。

19. 选择“控制” > “测试影片”。

关于主题

主题是样式和外观的集合。Flash 的默认主题称为“光晕”(HaloTheme fla)。“光晕”主题能让您为用户提供响应积极的、印象深刻的体验。Flash 还包括其它主题，如“范例”(SampleTheme fla)。“范例”主题提供关于如何使用更多样式进行自定义的示例。（“光晕”主题不使用包含在“范例”主题中的所有样式。）在默认安装中，主题文件位于以下文件夹中：

- 在 Windows 中：C:\Program Files\Macromedia\Flash 8\语言\Configuration\ComponentFLA\
- 在 Macintosh 上：HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/

您可以创建新主题并将它们应用到应用程序，以便更改所有组件的外观和行为。例如，您可以创建模拟本机操作系统外观的主题。

组件使用外观（图形或影片剪辑元件）来显示它们的外观。定义每个组件的 AS 文件包含为该组件加载特定外观的代码。通过复制“光晕”或“范例”主题，以及修改外观中的图形，您可以很方便地创建新主题。

主题中也可以包含一组新的样式默认值。必须通过编写 **ActionScript** 代码来创建全局样式声明和任何其它样式声明。有关详细信息，请参阅第 94 页的“[修改主题中的默认样式属性值](#)”。

切换主题

Macromedia Flash 安装了两个主题：“光晕”和“范例”。您会注意到每个组件的组件参考信息都包含一个样式属性表，您可以为每个（或两个）主题设置表中的样式属性。因此，当您查看样式属性表（如《组件语言参考》的“对 **Button** 组件使用样式”中 **Button** 组件的样式属性表）时，请注意哪个主题支持您想要的样式。该表将指示“光晕”、“范例”或“两者都是”（意思是两个主题都支持该样式属性）。

“光晕”主题是组件的默认主题。因此，如果要使用“范例”主题，您需要将当前主题从“光晕”切换到“范例”。

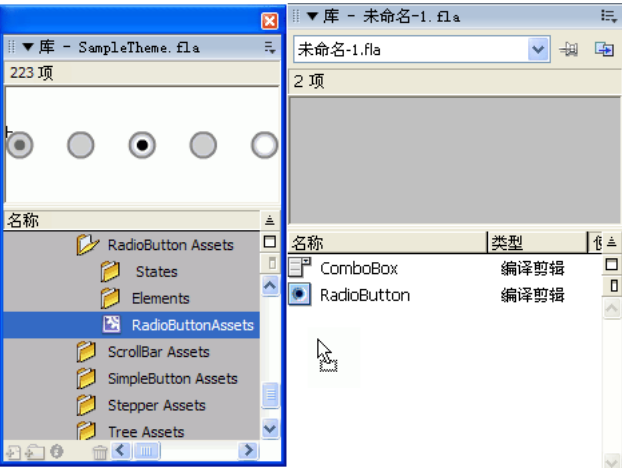
切换到“范例”主题：

1. 选择“文件”>“打开”，在 Flash 中打开使用第 2 版组件的文档，或选择“文件”>“新建”，创建使用第 2 版组件的新文档。

2. 选择“文件”>“导入”>“打开外部库”，然后选择 SampleTheme.fla 以应用于文档。
此文件位于应用程序级别的配置文件夹中。要了解此文件在您的操作系统上的确切位置，请参阅第 91 页的“关于主题”。

3. 在 SampleTheme.fla 主题的“库”面板中，选择 Flash UI Components 2/Themes/MMDefault，然后将文档中任意组件的 Assets 文件夹拖到 Flash 文档的“库”面板中。

例如，将 RadioButton Assets 文件夹拖到库中。



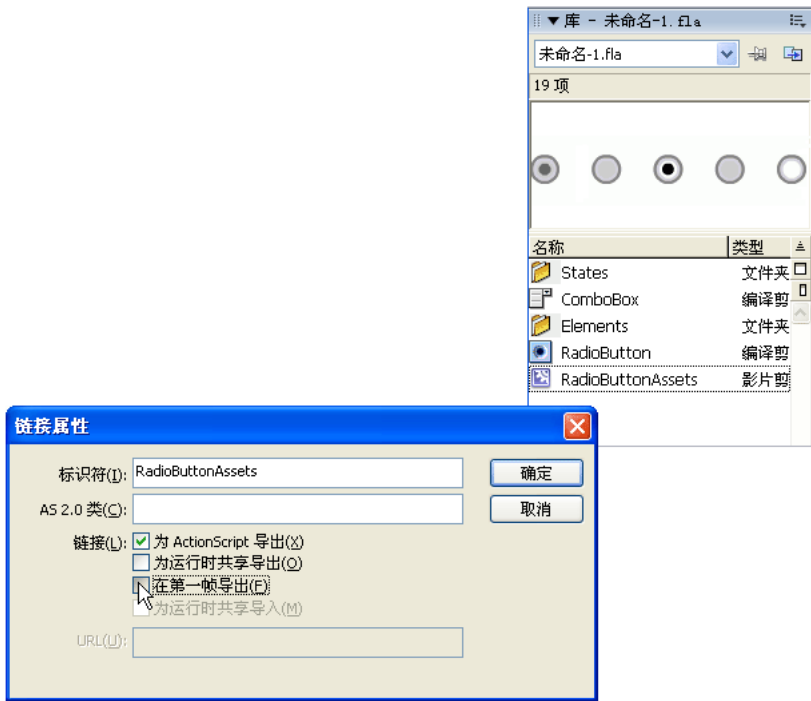
如果不确定文档中有哪些组件，可以将整个“范例主题”影片剪辑拖到舞台上。该外观即会自动指定给文档中的组件。

提醒 舞台上组件的“实时预览”不会反映新主题。

4. 如果将单个组件的 Assets 文件夹拖到文档的“库”面板中，请确保将每个组件的 Assets 元件都设置为“在第一帧导出”。

例如，RadioButton 组件的 Assets 文件夹名为 RadioButton Assets。打开 RadioButtonAssets 文件夹，您将看到一个名为 RadioButtonAssets 的影片剪辑元件。RadioButtonAssets 元件包含其中的所有单个资源元件。

右键单击（在 Windows 中）或按住 **Control** 键并单击（在 Macintosh 中）文档库中的 **RadioButtonAssets** 元件，然后选择“链接”菜单选项。选中“在第一帧导出”，使所有单个资源元件也都将在第一帧导出。然后，单击“确定”以保存设置。



5. 选择“控制” > “测试影片”，查看应用新主题的文

创建新主题

如果您不想使用“光晕”主题或“范例”主题，可以对它们中的一个进行修改，以创建新主题。

主题中的某些外观具有固定大小。您可以增加或减小它们的大小，组件会自动调整大小来与它们匹配。其它外观由多个片段组成，一些为静态，另一些可以伸展。

某些外观（例如 **RectBorder** 和 **ButtonSkin**）使用 **ActionScript** 绘制 API 来绘制它们的图形，因为就大小和性能而言，这种方式更有效率。您可以将这些外观中的 **ActionScript** 代码用作模板，按需要调整外观。

有关每个组件所支持的外观的列表以及这些外观的属性，请参阅《组件语言参考》。

创建新主题：

1. 选择要用作模板的主题 FLA 文件，然后复制该文件。
为副本指定唯一的名称，例如 **MyTheme.fl**。
2. 在 Flash 中选择“文件”>“打开 MyTheme.fl”。
3. 如果尚未打开库，请选择“窗口”>“库”将其打开。
4. 双击要修改的任何外观元件，在元件编辑模式下打开元件。
外观位于 Flash UI Components 2/Themes/MMDefault/Component Assets 文件夹（本示例使用 RadioButton Assets）。
5. 修改元件或删除图形，然后创建新图形。
您可能需要选择“视图”>“放大”来增加缩放比例。您在编辑外观时，必须维护注册点，以便正确显示外观。所有编辑过的元件的左上角都必须位于 (0,0)。
例如，打开 States/RadioFalseDisabled 资源，然后将圆的内侧更改为浅灰色。
6. 当编辑完该外观元件之后，单击舞台顶部信息栏左侧的“返回”按钮，返回到文档编辑模式。
7. 重复执行第 4 步至第 6 步，直到编辑完所有要更改的外观。
8. 执行本章稍后所示的步骤，将 MyTheme.fl 应用到文档。（请参阅第 96 页的“对文档应用新主题”。）

修改主题中的默认样式属性值

默认样式属性值是由每个主题在名为 Default 的类中提供的。要更改自定义主题的默认值，请在适合该主题的包中创建一个名为 Default 的新的 ActionScript 类，然后根据需要更改默认设置。

修改主题中的默认样式值：

1. 在 First Run/Classes/mx/skins 中为主题创建一个新文件夹。
例如，创建一个名为 **myTheme** 的文件夹。
2. 将现有的 Defaults 类复制到新的主题文件夹中。
例如，将 mx/skins/halo/Defaults.as 复制到 mx/skins/myTheme/Defaults.as 中。
3. 在 ActionScript 编辑器中打开新的 Defaults 类。
Flash Professional 8 用户可以在 Flash 中打开该文件。或者，您可以用 Windows 中的“记事本”或 Macintosh 中的 SimpleText 打开该文件。
4. 修改类声明来反映新包。
例如，新的类声明是 `class mx.skins.myTheme.Defaults`。

5. 根据需要修改样式设置。

例如，将默认的禁用颜色更改为深红色。

```
o.disabledColor = 0x663333;
```

6. 保存更改后的 Defaults 类文件。
7. 将现有的 FocusRect 类从源主题复制到自定义主题中。
例如，将 mx/skins/halo/FocusRect.as 复制到 mx/skins/myTheme/FocusRect.as。
8. 在 ActionScript 编辑器中打开新的 FocusRect 类。
9. 将所有对源主题包的引用修改为对新主题包的引用。
例如，将所有出现的 “halo” 更改为 “myTheme”。
10. 保存更改后的 FocusRect 类文件。
11. 打开自定义主题的 FLA 文件。
此示例使用 MyTheme.fla。
12. 打开库（“窗口” > “库”）并查找 Defaults 元件。
在此示例中，它位于 Flash UI Components 2/Themes/MMDefault/Defaults 中。
13. 编辑 Default 元件的元件属性。
14. 更改 “AS 2.0 类” 设置来反映新包。
示例类为 mx.skins.myTheme.Defaults。
15. 单击 “确定”。
16. 查找 FocusRect 元件。
在此示例中，它位于 Flash UI Components 2/Themes/MMDefault/FocusRect 中。
17. 编辑 FocusRect 元件的元件属性。
18. 更改 “AS 2.0 类” 设置来反映新包。
示例类为 mx.skins.myTheme.FocusRect。
19. 单击 “确定”。

20. 执行下一节中的步骤，将自定义主题应用到文档。

将资源从自定义主题拖到目标文档时，请记住要包含 Defaults 和 FocusRect 元件。

在此示例中，您使用了新主题来自定义禁用组件的文本颜色。通过如[第 69 页的“使用样式自定义组件的颜色和文本”](#)中说明的样式设置可以更容易地完成这个特殊的自定义操作，即更改单个默认样式属性值。在自定义很多样式属性时或在已经创建新主题来自定义组件图形时，适合于使用新主题自定义默认值。

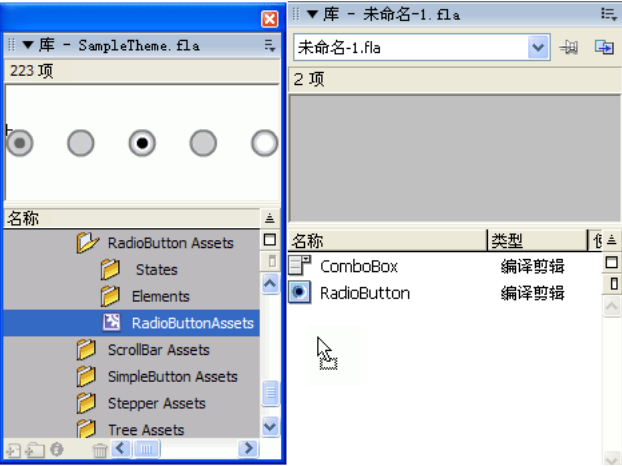
对文档应用新主题

若要将新主题应用到文档，请打开某主题 FLA 文件作为外部库，并将主题文件夹从外部库拖到文档库中。以下步骤详细说明该过程，并假定您已具有一个新主题（有关详细信息，请参阅第 93 页的“创建新主题”）。


将主题应用到文档：

- 1. 选择“文件”>“打开”，在 Flash 中打开使用第 2 版组件的文档，或选择“文件”>“新建”，创建使用第 2 版组件的新文档。
- 2. 选择“文件”>“导入”>“打开外部库”，然后选择要应用到文档的主题的 FLA 文件。
- 3. 在主题的“库”面板中，选择 Flash UI Components 2/Themes/MMDefault，并将要使用的所有组件的 Assets 文件夹拖到文档的库中。

例如，将 RadioButton Assets 文件夹拖到库中。

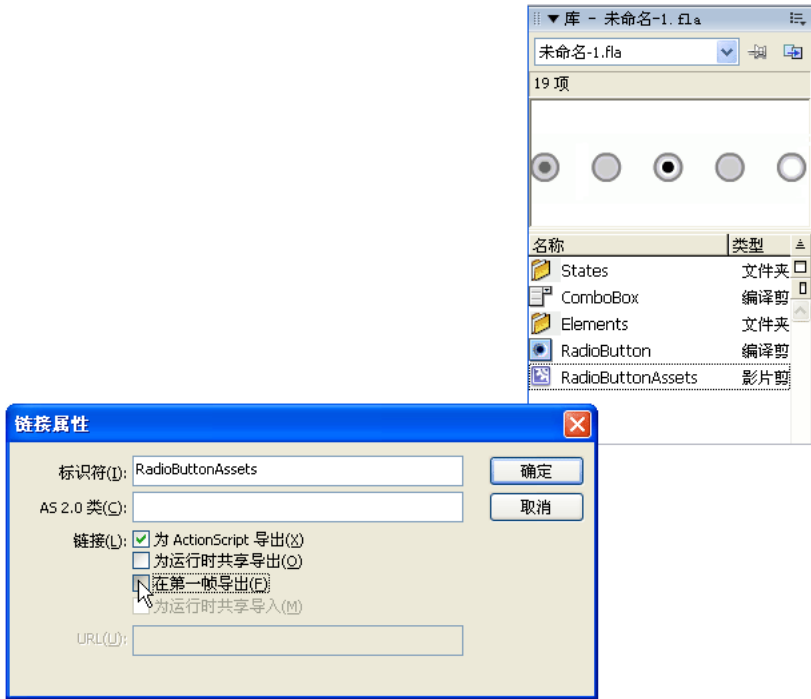


如果不能确定哪些组件在文档中，请将整个主题影片剪辑（例如，对于 SampleTheme.fla，主要的主题影片剪辑为“Flash UI Components 2”>“SampleTheme”）拖到舞台中。该外观即会自动指定给文档中的组件。

舞台上组件的“实时预览”不会反映新主题。

- 4. 如果把单个组件的 Assets 文件夹拖到 ThemeApply.fla 库中，请确保将每个组件的 Assets 元件设置为“在第一帧导出”。

例如，RadioButton 组件的 Assets 文件夹名为 RadioButton Assets；它具有一个名为 RadioButtonAssets 的元件，该元件包含所有单个资源元件。如果将 RadioButtonAssets 元件设置为“在第一帧导出”，所有单个资源元件也将在第一帧导出。



5. 选择“控制”>“测试影片”，查看应用的新主题。

更改导出设置

将“范例”或“光晕”主题应用于文档时，许多外观资源都设置为在第一帧导出，以便它们在回放时可立即用于组件。但是，如果将 FLA 文件的发布导出设置（“文件”>“发布设置”>“Flash”选项卡>“ActionScript 版本设置”按钮>“导出用于类的帧”）更改为第一帧以后的帧，您还必须更改“范例”和“光晕”主题中的资源的导出设置。要执行此操作，您必须打开文档库中的以下组件资源，然后取消选择“在第一帧导出”复选框（方法是，右键单击>“链接”>“在第一帧导出”）：

范例主题

- Flash UI Components 2/Base Classes/UIObject
- Flash UI Components 2/Themes/MMDefault/Defaults
- Flash UI Components 2/Base Classes/UIObjectExtensions

- Flash UI Components 2/Border Classes/BoundingBox
- Flash UI Components 2/SampleTheme
- Flash UI Components 2/Themes/MMDefault/Button Assets/Elements/ButtonIcon
- Flash UI Components 2/Themes/MMDefault/DateChooser Assets/ Elements/ Arrows/cal_disabledArrow
- Flash UI Components 2/Themes/MMDefault/FocusRect
- Flash UI Components 2/Themes/MMDefault/Window Assets/ States/ CloseButtonOver
- Flash UI Components 2/Themes/MMDefault/Accordion Assets/ AccordionHeaderSkin
- Flash UI Components 2/Themes/MMDefault/Alert Assets/AlertAssets
- Flash UI Components 2/Themes/MMDefault/Border Classes/Border
- Flash UI Components 2/Themes/MMDefault/Border Classes/CustomBorder
- Flash UI Components 2/Themes/MMDefault/Border Classes/RectBorder
- Flash UI Components 2/Themes/MMDefault/Button Assets/ActivatorSkin
- Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin

光晕主题

- Flash UI Components 2/Base Classes/UIObject
- Flash UI Components 2/Themes/MMDefault/Defaults
- Flash UI Components 2/Base Classes/UIObjectExtensions
- Flash UI Components 2/Component Assets/BoundingBox
- Flash UI Components 2/HaloTheme
- Flash UI Components 2/Themes/MMDefault/Accordion Assets/ AccordionHeaderSkin
- Flash UI Components 2/Themes/MMDefault/Alert Assets/AlertAssets
- Flash UI Components 2/Themes/MMDefault/Border Classes/Border
- Flash UI Components 2/Themes/MMDefault/Border Classes/CustomBorder
- Flash UI Components 2/Themes/MMDefault/Border Classes/RectBorder
- Flash UI Components 2/Themes/MMDefault/Button Assets/ActivatorSkin
- Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin
- Flash UI Components 2/Themes/MMDefault/Button Assets/Elements/ ButtonIcon
- Flash UI Components 2/Themes/MMDefault/CheckBox Assets/Elements/ CheckThemeColor1

- Flash UI Components 2/Themes/MMDefault/CheckBox Assets/CheckBoxAssets
- Flash UI Components 2/Themes/MMDefault/ComboBox Assets/ComboBoxAssets
- Flash UI Components 2/Themes/MMDefault/DataGrid Assets/DataGridAssets
- Flash UI Components 2/Themes/MMDefault/DateChooser Assets/DateChooserAssets
- Flash UI Components 2/Themes/MMDefault/FocusRect
- Flash UI Components 2/Themes/MMDefault/Menu Assets/MenuAssets
- Flash UI Components 2/Themes/MMDefault/MenuBar Assets/MenuBarAssets
- Flash UI Components 2/Themes/MMDefault/ProgressBar Assets/ProgressBarAssets
- Flash UI Components 2/Themes/MMDefault/RadioButton Assets/Elements/RadioThemeColor1
- Flash UI Components 2/Themes/MMDefault/RadioButton Assets/Elements/RadioThemeColor2
- Flash UI Components 2/Themes/MMDefault/RadioButton Assets/RadioButtonAssets
- Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/HScrollBarAssets
- Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/ScrollBarAssets
- Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/VScrollBarAssets
- Flash UI Components 2/Themes/MMDefault/Stepper Assets/Elements/StepThemeColor1
- Flash UI Components 2/Themes/MMDefault/Stepper Assets/NumericStepperAssets
- Flash UI Components 2/Themes/MMDefault/Tree Assets/TreeAssets
- Flash UI Components 2/Themes/MMDefault/Window Assets/Window Assets

通过组合外观与样式设置来自定义组件

在此部分中，您将使用样式、主题和外观设置来自定义一个组合框组件实例。该过程演示如何将外观设置与样式设置组合在一起，以创建独特的组件表现形式。

在舞台上创建组件实例

此练习的第一部分要求您创建一个用于自定义的 `ComboBox` 实例。

创建 `ComboBox` 实例：

1. 将 `ComboBox` 组件拖动到舞台上。
2. 在“属性”面板中，将该实例命名为 `my_cb`。
3. 在主时间轴的第一帧中，添加以下 `ActionScript`（请确保将其添加到该帧，而不是组件本身；“动作”面板应在标题栏中显示“动作 - 帧”）：

```
my_cb.addItem({data:1, label:"One"});  
my_cb.addItem({data:2, label:"Two"});
```
4. 选择“控制”>“测试影片”来查看使用“光晕”主题中的默认样式和外观设置的组合框。

创建新的样式声明

现在，您需要创建新的样式声明，然后为其指定各个样式。在样式声明中指定了所有想要的样式后，您就可以将新样式名称指定给组合框实例。

创建新的样式声明，并命名它：

1. 在主时间轴的第一帧中，在您的 `ActionScript` 的开头添加以下一行代码（将所有 `import` 语句放置在 `ActionScript` 的开头是一种编码规范，您应遵守它）：

```
import mx.styles.CSSStyleDeclaration;
```
2. 在下一行中，命名该新样式声明，然后将其添加到全局样式声明：

```
var new_style:Object = new CSSStyleDeclaration();  
_global.styles.myStyle = new_style;
```

将新样式声明指定给 `_global` 样式表后，您可以将各个样式设置附加到 `new_style` 样式声明。有关创建组件组的样式表（而不是单个实例的样式定义）的详细信息，请参阅第 73 页的“[为成组的组件设置自定义样式](#)”。

3. 将一些样式设置附加到 `new_style` 样式声明。以下样式设置包括可用于 `ComboBox` 组件的样式定义（请参阅《组件语言参考》中的“对 `ComboBox` 组件使用样式”以获得更完整的列表），以及 `RectBorder` 类中的样式，因为 `ComboBox` 组件使用 `RectBorder` 类：

```
new_style.setStyle("textAlign", "right");
new_style.setStyle("selectionColor", "white");
new_style.setStyle("useRollOver", false);
// RectBorder 类中的 borderStyle
new_style.setStyle("borderStyle", "none");
```

将样式定义指定给组合框

此时，您有了一个包含多种样式的样式声明，但是您需要将样式名称显式指定给组件实例。您可以用以下方法将此新样式声明指定给文档中的任何组件实例。在 `my_cb` 的 `addItem()` 语句后添加下行代码（将组合框构造语句放在一起是一种编码规范，您应遵守它）：

```
my_cb.setStyle("styleName", "myStyle");
```

附加到主时间轴第一帧的 `ActionScript` 代码应如下：

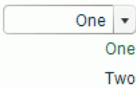
```
import mx.styles.CSSStyleDeclaration;

var new_style:Object = new CSSStyleDeclaration();
_global.styles.myStyle = new_style;

new_style.setStyle("textAlign", "right");
new_style.setStyle("selectionColor", "white");
new_style.setStyle("useRollOver", false);
// RectBorder 类中的 borderStyle
new_style.setStyle("borderStyle", "none");

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

选择“控制” > “测试影片”查看应用样式后的组合框：



更改组合框主题

每个用户界面组件都列出了您可为其设置的样式属性（例如，可为 **ComboBox** 组件设置的所有样式属性都在《组件语言参考》的“自定义 **ComboBox** 组件”中列出）。在样式属性表中，标题为“主题”的列显示哪个已安装的主题支持每个样式属性。不是所有的已安装主题都支持所有的样式属性。所有用户界面组件的默认主题都为“光晕”主题。将主题更改为“范例”主题时，您可以使用其它样式属性集（列为“仅光晕”的属性将不再可用）。

更改已应用样式的组件的主题：

1. 选择“文件”>“导入”>“打开外部库”，然后选择 **SampleTheme.fla** 以在 Flash 中打开“范例”主题库。

此文件位于应用程序级别的配置文件夹中：

- 在 Windows 中：C:\Program Files\Macromedia\Flash 8\语言\Configuration\ComponentFLA\
- 在 Macintosh 上：HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/

2. 将主 **SampleTheme**（“Flash UI Components 2”>“**SampleTheme**”）影片剪辑从 **SampleTheme** 库中拖到文档库中。

ComboBox 组件是一些组件和类的组合，因此需要其它此类组件和资源中的资源，其中包括 **Border** 和 **ScrollBar** 资源。确保拥有所需主题中的所有资源的最简单方法是将该主题的所有资源都拖到库中。

3. 选择“控制”>“测试影片”查看应用样式后的组合框：



编辑组合框外观资源

要编辑组件的外观，请以绘图方式编辑构成该组件的外观。要编辑外观，请打开当前主题中组件的图形资源，然后编辑该组件的元件。**Macromedia** 推荐使用此方法，原因是该方法不删除或添加其它组件可能需要的元件；此方法可以编辑现有组件外观元件的外观。

提醒

可以但不推荐编辑组件的源类文件，它将使该组件将具有不同名称的元件用作外观，您可以以编程方式更改外观元件中的 **ActionScript**（有关自定义的 **ActionScript** 和外观元件的示例，请参阅《组件语言参考》中的“自定义 **Accordion** 组件（仅限 **Flash Professional**）”）。但是，由于一些组件（包括 **ComboBox** 组件）与其它组件共享资源，因此编辑源文件或更改外观元件名称可产生意外的结果。

编辑组件外观元件时：

- 该组件的所有实例都将使用新的外观（但不是自定义的样式，除非将样式显式附加到实例），并且一些依赖于该组件的组件也将使用新的外观。
- 如果在编辑组件外观后指定新主题，请确保不要覆盖现有已编辑的外观（有一个对话框会问您是否要覆盖外观，它使您可以阻止 Flash 覆盖外观）。

在此部分中，您将继续使用前一部分中的组合框（请参阅第 102 页的“更改组合框主题”）。以下步骤将打开组合框菜单的向下箭头的外观从箭头更改为圆。

编辑组合框的向下箭头元件：

1. 在文档库中，打开 **ComboBox** 资源，查看在运行时打开和关闭组合框实例的按钮的外观影片剪辑。具体讲，在文档库中打开“主题” > “MMDefault” > “ComboBox 资源” > “状态”文件夹。

“状态”文件夹包含四个影片剪辑：**ComboDownArrowDisabled**、**ComboDownArrowDown**、**ComboDownArrowOver** 和 **ComboDownArrowUp**。这四个元件都是由其它元件组成的。并且这四个元件都将称为 **SymDownArrow** 的同一元件用于向下箭头（三角形）。

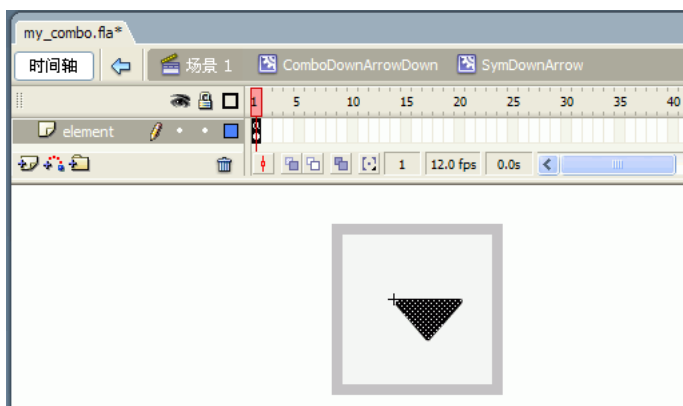
2. 双击 **ComboDownArrowDown** 元件编辑它。

您可能需要进行放大（最大至 800%）以便查看该按钮的细节。

3. 双击向下箭头（黑色三角形）编辑它。



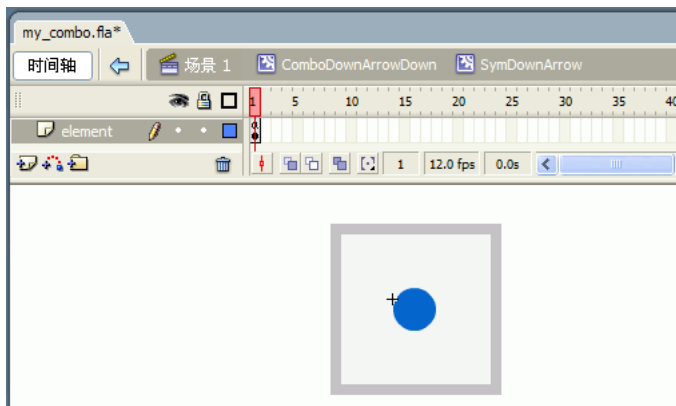
请确保选中元件 **SymDownArrow**，以便只删除影片剪辑中的该形状，而不是影片剪辑元件本身。



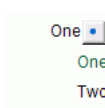
4. 删除舞台上的所选向下箭头（黑色三角形形状，而不是整个影片剪辑）。

5. 继续编辑 `SymDownArrow`，在向下箭头曾经所在的位置绘制一个圆。

为了使更改更明显一些，考虑绘制亮色（如蓝色）的圆，大小大约为 4 x 4 像素，x 坐标为 0，y 坐标为 -1，这样它就居中了。



6. 选择“控制” > “测试影片”查看应用外观后的复选框：



在文档库中，如果选择 `ComboDownArrowOver` 和 `ComboDownArrowUp`，您将会看到它们用蓝色的圆代替了黑色三角形，原因是它们也将 `SymDownArrow` 用于向下箭头元件。

本章介绍如何创建自己的组件并将其打包以进行分发。

本章包含以下各节：

组件源文件	106
组件结构概述	106
构建第一个组件	107
选择父类	116
创建组件影片剪辑	118
创建 <code>ActionScript</code> 类文件	123
在组件内组合现有组件	148
导出和分发组件	157
组件开发的最后一些步骤	160

组件源文件

“组件”面板中提供的组件都是预编译过的 SWC 剪辑。同时还提供了源 Flash 文档 (FLA) (其中包括的是这些组件的图形) 和源 **ActionScript** 类文件 (AS) (其中包含的是这些组件的代码), 以便于您在创建自己的自定义组件时使用。第 2 版组件的源文件随 **Macromedia Flash** 一起安装。在构建自己的组件之前, 打开并查看其中部分文件, 尝试去了解这些文件的结构, 会对您有所帮助。**RadioButton** 组件就是较为简单的组件的示例, 您可能应首先研究该组件。**StandardComponents.fla** 的库中的所有组件都是元件。每个元件都链接到一个 **ActionScript** 类。它们的位置如下:

- **FLA 文件源代码**
 - 在 Windows 中: C:\Program Files\Macromedia\Flash 8\语言\Configuration\ComponentFLA\StandardComponents.fla。
 - 在 Macintosh 上: HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/StandardComponents.fla
- **ActionScript 类文件**
 - 在 Windows 中: C:\Program Files\Macromedia\Flash 8\语言\First Run\Classes\mx
 - 在 Macintosh 上: HD/Applications/Macromedia Flash 8/First Run/Classes/mx

组件结构概述

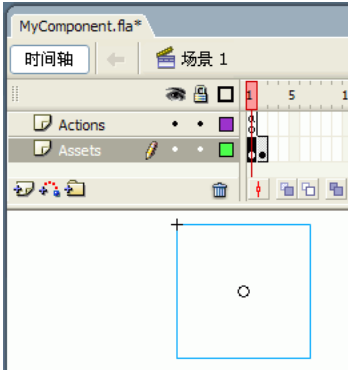
组件由 **Flash (FLA)** 文件和 **ActionScript (AS)** 文件组成。您可以选择创建其它文件 (例如, 图标和 .swd 调试文件), 并将其与组件一起打包, 但所有组件都需要一个 **FLA** 文件和一个 **ActionScript** 文件。完成组件开发后, 需要将它导出为 **SWC** 文件。



Flash (FLA) 文件、ActionScript (AS) 文件和 SWC 文件

FLA 文件包含一个影片剪辑元件, 该元件必须在 “链接属性” 和 “组件定义” 对话框中链接到 AS 文件。

影片剪辑元件有两个帧和两个图层。第一个图层是 **Actions** 图层，它的第一帧上有一个 `stop()` 全局函数。第二个图层是 **Assets** 图层，它有两个关键帧：第一帧包含一个边框，第二帧包含所有其它资源，其中包括组件使用的图形和基类。



指定组件的属性和方法的 **ActionScript** 代码位于单独的 **ActionScript** 类文件中。此类文件还声明组件的扩展的类（如果有）。AS 类文件的名称为组件的名称加上 “.as” 扩展名。例如，**MyComponent.as** 包含 **MyComponent** 组件的源代码。

最好将组件的 **FLA** 和 **AS** 文件保存在同一文件夹并将这两个文件指定为相同的名称。如果将 **AS** 文件保存在其它文件夹中，则必须确认该文件夹在类路径中，以便 **FLA** 文件能够找到它。有关类路径的详细信息，请参阅《学习 Flash 中的 **ActionScript 2.0**》中的“类”。

构建第一个组件

在本节中，将构建一个 **Dial** 组件。已完成的组件文件 **Dial.fla**、**Dial.as** 和 **DialAssets.fla** 位于计算机上的以下示例文件夹中：

- 在 Windows 中：C:\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\DialComponent 文件夹。
- 在 Macintosh 上：HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/DialComponent 文件夹。

Dial 组件是一个电位计，类似于测量电压差电位计。用户可以单击并拖动指针来更改指针位置。**Dial** 组件的 API 具有一个 `value` 属性，可用来获取和设置指针的位置。

本节将指导您完成创建组件的步骤。后面的章节（包括第 116 页的“选择父类”、第 118 页的“创建组件影片剪辑”、第 123 页的“创建 **ActionScript** 类文件”和第 157 页的“导出和分发组件”）中将对这些过程进行更详细的讨论。本节包含以下主题：

- 第 108 页的“创建 **Dial Flash (FLA)** 文件”
- 第 110 页的“创建 **Dial** 类文件”
- 第 113 页的“测试和导出 **Dial** 组件”

创建 Dial Flash (FLA) 文件

在创建组件的前几个步骤中，将在 FLA 文档文件中创建组件影片剪辑。

若要创建 Dial FLA 文件，请执行以下操作：

1. 在 Flash 中，选择“文件”>“新建”，然后创建一个新文档。
2. 选择“文件”>“另存为”，将文件另存为 **Dial.fla**。
该文件可以使用任何名称，但习惯上为其指定与组件相同的名称。
3. 选择“插入”>“新建元件”。组件本身做为一个新的 MovieClip 元件进行创建，因此可以通过库来使用该组件。
将组件命名为 **Dial**，并为其指定行为“影片剪辑”。
4. 如果“创建新元件”对话框中的“链接”部分未打开，则单击“高级”按钮以显示该部分。
5. 在“链接”区域中，选择“为 ActionScript 导出”，并取消选择“在第一帧导出”。
6. 在“标识符”文本框中，输入链接标识符，例如 **Dial_ID**。
7. 在“AS 2.0 类”文本框中，输入 **Dial**。该值是组件类名称。如果该类是一个包（例如，**mx.controls.Button**），则输入完整的包名称。
8. 单击“确定”。

Flash 会切换到元件编辑模式。

9. 插入一个新图层。将顶部图层命名为 **Actions**，将底部图层命名为 **Assets**。
10. 选择 Assets 图层的第二帧，并插入一个关键帧 (F6)。
该组件影片剪辑的结构为：一个 Actions 图层和一个 Assets 图层。Actions 图层有一个关键帧，Assets 图层有两个关键帧。
11. 选择 Actions 图层的第一帧，然后打开“动作”面板 (F9)。输入 `stop();` 全局函数。
这样可防止影片剪辑进入第二帧。
12. 选择“文件”>“导入”>“打开外部库”，然后从 **Configuration/ComponentFLA** 文件夹中选择 **StandardComponents.fla** 文件。例如：
 - 在 Windows 中：C:\Program Files\Macromedia\Flash 8\语言\Configuration\ComponentFLA\StandardComponents.fla。
 - 在 Macintosh 上：HD/Applications/Macromedia Flash 8/Configuration/ComponentFLA/StandardComponents.fla



有关文件夹位置的信息，请参阅《Flash 入门》中的“随 Flash 安装的配置文件夹”。

13. Dial 扩展 UIComponent 基类；因此，必须将 UIComponent 类的实例拖到 Dial 文档中。在 StandardComponents.fla 库中，浏览到以下文件夹中的 UIComponent 影片剪辑：Flash UI Components 2 > Base Classes > FUIObject Subclasses，然后将该影片剪辑拖到 Dial.fla 库中。

资源依赖项和 UIComponent 一起自动复制到 Dial 库中。

提醒

将 UIComponent 拖到 Dial 库时，会更改 Dial 库中的文件夹层次结构。如果打算再次使用库，或将它用于其它组件组（例如第 2 版组件），应重新组织文件夹层次结构，以便与 StandardComponents.fla 库匹配，从而使库组织有序，避免元件重复。

14. 在 Assets 图层中，选择第二帧，将 UIComponent 的一个实例拖到舞台上。
15. 关闭 StandardComponents.fla 库。
16. 选择“文件”>“导入”>“打开外部库”，然后选择 DialAssets.fla 文件。
- 在 Windows 中：C:\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\DialComponent\DialAssets.fla。
 - 在 Macintosh 上：HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/DialComponent/DialAssets.fla
17. 在 Assets 图层中，选择第二帧，然后将 DialFinal 影片剪辑的一个实例从 DialAssets 库拖到舞台上。
- 所有组件资源都已添加到 Assets 图层的第二帧中。因为 Actions 图层的第一帧上有 stop() 全局函数，所以第二帧中的资源排列在舞台时用户看不到这些资源。
- 之所以将资源添加到第二帧，有两个原因：
- 这样做可使所有资源和子资源自动复制到库中，并可用于动态实例化（如果是 DialFinal）或访问它们的方法、属性和事件（如果是 UIComponent）。
 - 将资源放入帧中可以确保在传送影片流时能够更顺畅地加载这些资源，所以，无需将库资源设置为在第一帧中导出。此方法可避免下载过程中出现数据传输的初始峰值。
18. 关闭 DialAssets.fla 库。

19. 在 Assets 图层中，选择第一帧，将 BoundingBox 影片剪辑从库（Flash UI Components 2 > Component Assets 文件夹）拖到舞台上。将 BoundingBox 实例命名为 **boundingBox_mc**。使用“信息”面板将 DialFinal 影片剪辑的高和宽都设置为 250 像素，且 x、y 坐标为 0、0。

BoundingBox 实例用于创建组件的实时预览并在创作时处理调整大小操作。必须设置边框大小以使边框能够包含组件中的所有图形元素。

提醒

如果要扩展一个组件（包括任何第 2 版组件），则必须保留任何已被该组件使用的实例名称，因为其代码会引用这些实例名称。例如，如果包括已使用实例名称 boundingBox_mc 的第 2 版组件，则不应重命名该实例。对于您自己的实例名称，您可以使用和同一范围内现有的名称不冲突的任何唯一名称。

20. 在库中选择 **Dial** 影片剪辑，然后从“库”上下文菜单中选择“组件定义”（Windows: 右键单击; Mac: 按住 **Control** 键单击）。

21. 在“AS 2.0 类”文本框中，输入 **Dial**。

这个值是 **ActionScript** 类的名称。如果该类在包中，则值为整个包，例如 **mx.controls.CheckBox**。

22. 单击“确定”。

23. 保存该文件。

创建 Dial 类文件

现在需要将 **Dial** 类文件来创建为新的 **ActionScript** 文件。

要创建 **Dial** 类文件：

1. 在 **Flash** 中，选择“文件”>“新建”，然后选择“**ActionScript** 文件”。

2. 选择“文件”>“另存为”，将文件保存到 **Dial.fla** 所在的文件夹中，文件名为 **Dial.as**。



可以使用任何文本编辑器保存 **Dial.as** 文件。

3. 您可以在新建的 **Dial.as** 文件中复制或键入以下 **Dial** 组件 **ActionScript** 类代码。键入代码要比复制代码更有助于熟悉组件代码的每个元素。

请阅读代码中的注释，了解每一部分的说明。（有关组件类文件的元素的详细信息，请参阅第 124 页的“[组件类文件概述](#)”。

```
// 导入包以便
// 直接引用类。
import mx.core.UIComponent;

// 事件元数据标记
[Event("change")]
class Dial extends UIComponent
{
    // 组件必须将它们声明为
    // 组件框架中的正确组件。
    static var symbolName:String = "Dial";
    static var symbolOwner:Object = Dial;
    var className:String = "Dial";

    // 作为组件的图形表示形式的
    // 指针和 Dial 影片剪辑
    private var needle:MovieClip;
    private var dial:MovieClip;
    private var boundingBox_mc:MovieClip;

    // 通过隐式 getter/setter 方法
```

```

// 可公共地访问私有成员变量 “__value”,
// 当该值已设置时, 更新此属性
// 就可以更新指针的位置。
private var __value:Number = 0;

// 此标记在用户用鼠标拖动
// 指针时设置, 随后即被清除。
private var dragging:Boolean = false;

// 构造函数;
// 为所有类所需要的同时, 第 2 版组件还需要
// 构造函数为空, 有零个参数。
// 构造类实例后, 所有初始化都发生
// 在所需的 init() 方法中。
function Dial() {
}

// 初始化代码:
// 第 2 版组件需要 init() 方法。它还必须
// 用 super.init() 调用它的父类 init() 方法。
// 组件扩展 UIComponent 需要 init() 方法。
function init():Void {
    super.init();
    useHandCursor = false;
    boundingBox_mc._visible = false;
    boundingBox_mc._width = 0;
    boundingBox_mc._height = 0;
}

// 创建启动时所需的子对象:
// 组件扩展 UIComponent 需要
// createChildren() 方法。
public function createChildren():Void {
    dial = createObject("DialFinal", "dial", 10);
    size();
}

// 第 2 版组件需要 draw() 方法。
// 通过调用 invalidate() 使组件
// 无效后, 即调用该方法。
// 这种方法比在该值的 set() 函数内
// 重绘更好, 因为如果存在其它属性,
// 最好在一次重绘中成批更改,
// 而不是逐个执行这些更改。此方法
// 能使效率更高, 代码更集中。
function draw():Void {
    super.draw();
    dial.needle._rotation = value;
}

// 组件的大小更改时,

```

```

// 即调用 size() 方法。这一特性可用来调整子级大小，
// 拨盘和指针图形的大小。
// 组件扩展 UIComponent 需要 size() 方法。
function size():Void {
    super.size();
    dial._width = width;
    dial._height = height;
    // 导致指针重绘（如果需要的话）。
    invalidate();
}

// 这是 value 属性的 getter/setter。
// [Inspectable] 元数据使属性显示在
// “属性”检查器中。此为 getter/setter，
// 以便在值更改时调用 invalidate 并强制组件
// 重绘。
[Bindable]
[ChangeEvent("change")]
[Inspectable(defaultValue=0)]
function set value (val:Number)
{
    __value = val;
    invalidate();
}

function get value ():Number
{
    return twoDigits(__value);
}

function twoDigits(x:Number):Number
{
    return (Math.round(x * 100) / 100);
}

// 指示组件预期鼠标按下
function onPress()
{
    beginDrag();
}

// 按下拨盘时，即设置拖动标记。
// 向鼠标事件分配回调函数。
function beginDrag()
{
    dragging = true;
    onMouseMove = mouseMoveHandler;
    onMouseUp = mouseUpHandler;
}

```



```

// 拖动完成时删除鼠标事件，
// 并清除标记。
function mouseUpHandler()
{
    dragging = false;
    delete onMouseMove;
    delete onMouseUp;
}

function mouseMoveHandler()
{
    // 计算角度
    if (dragging) {
        var x:Number = _xmouse - width/2;
        var y:Number = _ymouse - height/2;

        var oldValue:Number = value;
        var newValue:Number = 90+180/Math.PI*Math.atan2(y, x);
        if (newValue<0) {
            newValue += 360;
        }
        if (oldValue != newValue) {
            value = newValue;
            dispatchEvent( {type:"change"} );
        }
    }
}
}

```

测试和导出 Dial 组件

您已创建了一个包含图形元素、基类和类文件（包含 **Dial** 组件的所有功能）的 **Flash** 文件。现在即可测试组件。

测试组件最好在工作过程中，尤其是在编写类文件时进行。开发过程中最快速的测试方法，是将组件转换为编译剪辑，然后在组件的 **FLA** 文件中使用。

组件彻底完成之后，将它导出为 **SWC** 文件。有关详细信息，请参阅第 157 页的“导出和分发组件”。

测试 Dial 组件：

1. 在 Dial.fla 文件中，选择库中的 Dial 组件，打开“库”上下文菜单（Windows：右键单击；Mac：按住 **Control** 键单击），然后选择“转换为编译剪辑”。

一个名为 Dial SWF 的编译剪辑即添加到库中。

提醒

如果已创建了一个编译剪辑（例如，这是第二次或第三次测试），则会出现一个“解决库冲突”对话框。选择“替换现有项目”，将新版本添加到文档中。

2. 将 Dial SWF 拖到主时间轴的舞台上。
3. 您可以调整它的大小，也可以在“属性”检查器或“组件”检查器中设置它的 **value** 属性。设置它的 **value** 属性时，指针的位置应随之变化。
4. 要在运行时测试 **value** 属性，请将拨盘命名为实例名称 **dial**，然后将下面的代码添加到主时间轴的第一帧：

```
// 文本字段的位置
var textXPos:Number = dial.width/2 + dial.x
var textYPos:Number = dial.height/2 + dial.y;

// 创建一个文本字段，以便在其中查看 dial.value
createTextField("dialValue", 10, textXPos, textYPos, 100, 20);

// 创建一个用于处理更改事件的侦听器
function change(evt){
    // 只要指针移动，
    // 就将 value 属性放在文本字段中
    dialValue.text = dial.value;
}
dial.addEventListener("change", this);
```

5. 选择“控制” > “测试影片”，在 Flash Player 中测试此组件。

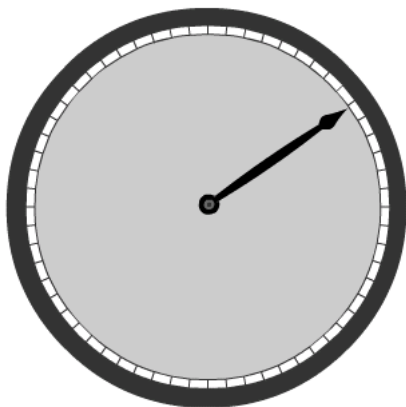
导出 Dial 组件：

1. 在 Dial.fla 文件中，选择库中的 Dial 组件，打开“库”上下文菜单（Windows：右键单击；Mac：按住 Control 键单击），然后选择“导出 SWC 文件”。
2. 选择一个位置来保存 SWC 文件。

如果将文件保存在用户级别 Configuration 文件夹中的 Components 文件夹下，则无需重新启动 Flash 即可重新加载“组件”面板，组件将显示在面板中。



有关文件夹位置的信息，请参阅《使用 Flash》中的“随 Flash 安装的配置文件夹”。



完成的 Dial 组件

选择父类

创建组件时，首先要确定是否扩展某个第 2 版类。如果选择扩展某个第 2 版类，则可以扩展某个组件类（例如，**Button**、**CheckBox**、**ComboBox**、**List** 等）或某个基类（**UIObject** 或 **UIComponent**）。除 **Media** 组件之外，所有其它组件类都扩展基类；如果扩展组件类，该类也会自动从基类继承。

两个基类为组件提供了常见功能。通过扩展这些类，组件一开始即具备一组基本的方法、属性和事件。

在第 2 版框架中，您无需创建 **UIObject** 子类、**UIComponent** 子类或任何其它类。即使组件类直接继承自 **MovieClip** 类，也可以使用许多强大的组件功能：导出到 SWC 文件或编译剪辑、使用内置实时预览、查看可检查属性等等。但是，如果要将组件用于 **Macromedia** 第 2 版组件，并要使用管理器类，就需要扩展 **UIObject** 或 **UIComponent**。

下表简要说明第 2 版基类：

基类	扩展	说明
<code>mx.core.UIObject</code>	MovieClip	UIObject 是所有图形对象的基类。它可以有形状、可以自己进行绘制，还可以是不可见的。 UIObject 提供以下功能： <ul style="list-style-type: none">• 编辑样式• 事件处理• 按缩放比例调整大小
<code>mx.core.UIComponent</code>	UIObject	UIComponent 是所有组件的基类。 UIComponent 提供以下功能： <ul style="list-style-type: none">• 创建焦点导航• 创建 Tab 键切换方案• 启用和禁用组件• 调整组件大小• 处理低级鼠标和键盘事件

了解 UIObject 类

基于 Macromedia Component Architecture 第 2 版的组件源自 **UIObject** 类，该类是 **MovieClip** 类的子类。**MovieClip** 类是 Flash 中可以在屏幕上表示可视对象的所有类的基类。

UIObject 添加用于处理样式和事件的方法。它在加载和卸载时（load 和 unload）、布局更改时（move、resize）以及隐藏或显示时（hide 和 reveal），都会在绘制（draw 事件等效于 **MovieClip.onEnterFrame** 事件）之前将事件发送到它的侦听器。

UIObject 另外提供只读变量来确定组件的位置和大小（width、height、x、y），并提供 **move()** 和 **setSize()** 方法来改变对象的位置和大小。

UIObject 类实现以下功能：

- 样式
- 事件
- 按缩放比例调整大小

了解 UIComponent 类

UIComponent 类是 **UIObject** 的子类（请参阅《组件语言参考》中的 **UIComponent** 类）。它是处理用户交互（鼠标和键盘输入）的所有组件的基类。**UIComponent** 类允许组件执行以下操作：

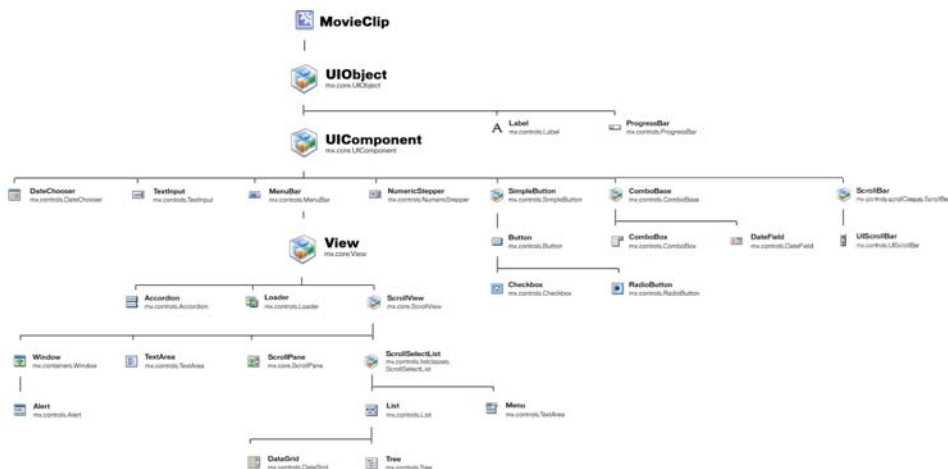
- 接收焦点和键盘输入
- 启用和禁用组件
- 按布局调整大小

关于扩展其它第 2 版类

为了能够更方便地构造组件，可以扩展任何类的子类，这样也就不需要直接扩展 **UIObject** 或 **UIComponent** 类。如果扩展任何其它第 2 版组件类（**Media** 组件除外），默认情况下也会扩展 **UIObject** 和 **UIComponent**。您可以通过扩展“组件”字典中列出的任何组件类来创建新组件类。

例如，如果要创建一个组件，其行为与 **Button** 组件的行为几乎相同，就可以扩展 **Button** 类，而不必从基类重新创建 **Button** 类的所有功能。

下图显示了第 2 版组件的层次结构：



第 2 版组件层次结构

此文件的 FlashPaper 版本可在 Flash 安装目录的以下位置找到：Flash 8\Samples and Tutorials\Samples\Components\arch_diagram.swf。

关于扩展 MovieClip 类

您可以选择不扩展第 2 版类，而让组件直接继承 **ActionScript MovieClip** 类。但是，如果需要 **UIObject** 和 **UIComponent** 的任何功能，则必须亲自构建。您可以打开 **UIObject** 和 **UIComponent** 类 (First Run/Classes/mx/core) 来检查它们的构造方式。

创建组件影片剪辑

若要创建一个组件，必须创建一个影片剪辑元件并将它链接到该组件的类文件。

影片剪辑有两个帧和两个图层。第一个图层是 **Actions** 图层，它的第一帧上有一个 **stop()** 全局函数。第二个图层是 **Assets** 图层，它有两个关键帧。第一帧包含一个边框或充当最终图片的占位符的任意图形。第二帧包含所有其它资源，包括组件使用的图形和基类。

插入新的影片剪辑元件

所有组件都是 MovieClip 对象。要创建新组件，首先必须将新元件插入新的 FLA 文件中。

添加新组件元件：

1. 在 Flash 中，创建空白的 Flash 文档。
2. 选择 “插入” > “新建元件”。
显示 “创建新元件” 对话框。
3. 输入元件名称。为组件命名，方法是将组件中每个单词的第一个字母更改为大写字母（例如 MyComponent）。
4. 选择 “影片剪辑” 行为。
5. 单击 “高级” 按钮显示高级设置。
6. 选择 “为 ActionScript 导出”，取消选择 “在第一帧导出”。
7. 输入链接标识符。
8. 在 “AS 2.0 类” 文本框中，输入 ActionScript 2.0 类的完全限定路径。

类名称应与显示在 “组件” 面板中的组件名称相同。例如，Button 组件的类为 mx.controls.Button。



不要包含文件扩展名；“AS 2.0 类” 文本框指向类的打包位置，而不是该文件的文件系统名称。

如果 ActionScript 文件位于包内，必须包含该包的名称。此值可以是类路径的相对路径，也可以是包的绝对路径（例如 mypackage.MyComponent）。

9. 大多数情况下，应取消选中 “在第一帧导出”（默认选中该选项）。有关详细信息，请参阅第 161 页的 “组件开发检查列表”。
10. 单击 “确定”。

Flash 将元件添加到库中，然后切换到元件编辑模式。在此模式下，元件的名称显示于舞台左上角的上方，并且有一个十字线表明该元件的注册点。

编辑影片剪辑

在创建新元件并为其定义链接后，即可在该元件的时间轴中定义组件的资源。

组件的元件应有两个图层。本节说明应该插入哪些图层，应该在这些图层上添加哪些内容。

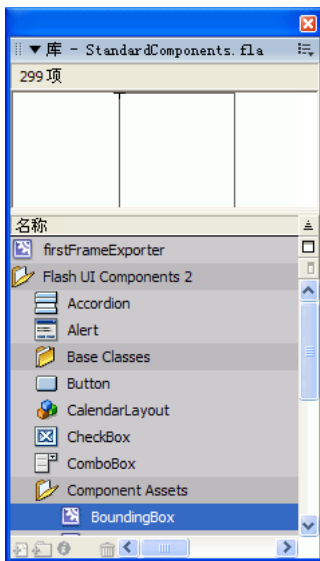
若要编辑影片剪辑，请执行以下操作：

1. 将图层 1 重命名为**动作**，然后选择第一帧。
2. 打开“动作”面板，然后添加 `stop()` 函数，如下所示：

```
stop();
```

不要向此帧添加任何图形资源。

3. 添加一个名为 **Assets** 的图层。
4. 在 **Assets** 图层上，选择第二帧并插入一个空白关键帧。
现在该图层上有两个空白关键帧。
5. 请执行以下操作之一：
 - 如果组件具有定义边界区域的可视资源，则将这些元件拖到第一帧中并进行适当安排。
 - 如果组件在运行时创建其所有可视资源，请将一个 **BoundingBox** 元件拖到舞台上第一帧中，适当调整其大小，然后将实例命名为 **boundingBox_mc**。该元件位于 **Configuration/ComponentFLA** 文件夹中的 **StandardComponents.fla** 的库中。

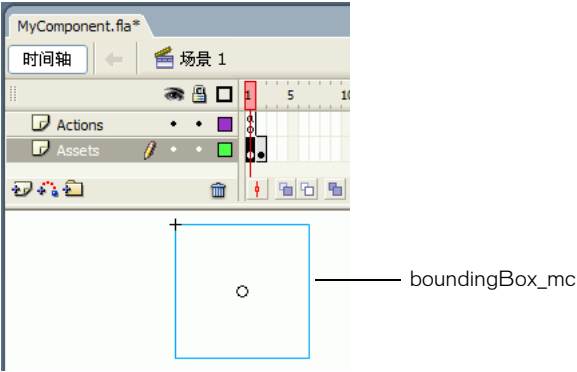


6. 如果要扩展现有组件，请将该组件的一个实例和任何其它基类放在 **Assets** 图层的第二帧中。

为此，请从“组件”面板中选择该元件并将它拖到舞台上。如果要扩展一个基类，请打开 **Configuration/ComponentFLA** 文件夹中的 **StandardComponents.flc**，然后将该类从库中拖到舞台上。

提醒 将 **UIComponent** 拖到组件库时，会更改库中的文件夹层次结构。如果打算再次使用库，或将它用于其它组件组（如第 2 版组件），应重新组织文件夹层次结构，以便与 **StandardComponents.flc** 库匹配，从而使库组织有序，避免元件重复。

7. 在组件的 **Assets** 图层的第二帧添加由该组件使用的所有图形资源。
- 组件所使用的任何资源（无论是其它组件还是位图之类的媒体）都应具有一个放置在 **Assets** 图层的第二帧中的实例。
8. 完成的元件看起来应与下图相似：



将影片剪辑定义为组件

影片剪辑元件必须在“组件定义”对话框中链接到 **ActionScript** 类文件。这是 **Flash** 了解在何处查找组件元标记的方式。（有关元标记的详细信息，请参阅第 128 页的“添加组件元数据”。）您也可以在“组件定义”对话框中选择其它选项。

将影片剪辑定义为组件：

1. 在库中选择影片剪辑，然后从“库”上下文菜单中选择“组件定义”（Windows：右键单击；Mac：按住 **Control** 键单击）。

2. 必须输入一个 **AS 2.0** 类。

如果该类在包中，则输入完整的包名称。

3. 如果需要，可以在“组件定义”对话框中指定其它选项：

- 单击加号 (+) 按钮可定义参数。

此为可操作。最好的做法是在组件的类文件中使用元数据 **Inspectable** 标记来指定参数。如果未指定 **ActionScript 2.0** 类，则在此处定义组件的参数。

- 指定自定义 UI。

这是一个在“组件”检查器中播放的 **SWF** 文件。您可以将它嵌入组件 **FLA** 文件或浏览到一个外部 **SWF** 文件。

- 指定实时预览。

这是一个外部或嵌入的 **SWF** 文件。此处无需指定实时预览；将一个边框添加到组件影片剪辑，**Flash** 即会为您创建实时预览。请参阅第 118 页的“创建组件影片剪辑”。

- 输入描述。

在 **Flash MX 2004** 中，由于“参考”面板已删除，因此不建议使用“描述”字段。提供此字段目的在于以 **Flash MX** 格式保存 **FLA** 文件时的向后兼容性。

- 选择图标。

此选项指定一个 **PNG** 文件来用作组件的图标。如果在 **ActionScript 2.0** 类文件中指定 **IconFile** 元数据标记（最佳做法），则会忽略该字段。

- 选择或取消选择“实例中参数已锁定”。

如果取消选择此选项，用户则可以向每个组件实例添加不同于组件参数的参数。通常应选择此设置。此选项提供与 **Flash MX** 的向后兼容性。

- 指定显示在“组件”面板中的工具提示。

创建 ActionScript 类文件

所有组件元件都链接到 ActionScript 2.0 类文件。（有关链接的信息，请参阅第 118 页的“创建组件影片剪辑”。）

要编辑 ActionScript 类文件，您可以使用 Flash、任何文本编辑器，也可以使用任何“集成开发环境”（IDE）。

外部 ActionScript 类扩展其它类（无论该类是第 2 版组件、第 2 版基类还是 ActionScript MovieClip 类）。您应扩展所创建的功能与要创建的组件最类似的类。只能从一个类继承（扩展）。ActionScript 2.0 不允许多继承。

组件类文件的简单示例

下面是类文件的一个简单示例，该类文件名为 MyComponent.as。如果要创建此组件，应将该类文件链接到 Flash 中的组件影片剪辑。

此示例包含从 UIComponent 类继承的组件 MyComponent 至少应有的一组导入、方法和声明。MyComponents.as 文件保存在 myPackage 文件夹中。

```
[Event("eventName")]
```

```
// 导入包。
```

```
import mx.core.UIObject;
```

```
// 声明类并从父类扩展。
```

```
class mypackage.MyComponent extends UIObject {
```

```
    // 标识此类所绑定到的元件名称。
```

```
    static var symbolName:String = "mypackage.MyComponent";
```

```
    // 标识元件所有者的完全限定包名称。
```

```
    static var symbolOwner:Object = Object(mypackage.MyComponent);
```

```
    // 提供 className 变量。
```

```
    var className:String = "MyComponent";
```

```
    // 定义一个空构造函数。
```

```
    function MyComponent() {  
    }
```

```
    // 调用父级的 init() 方法。
```

```
    // 隐藏边框，该边框
```

```
    // 仅在创作时使用。
```

```
    function init():Void {  
        super.init();
```

```
        boundingBox_mc.width = 0;
```

```

        boundingBox_mc.height = 0;
        boundingBox_mc.visible = false;
    }

    function createChildren():Void{
        // 调用 createClassObject 来创建子对象。
        size();
        invalidate();
    }

    function size(){
        // 编写处理大小的代码。
        super.size();
        invalidate();
    }

    function draw(){
        // 编写处理可视化表示形式的代码。
        super.draw();
    }
}

```

组件类文件概述

以下是介绍如何为组件类创建 **ActionScript** 文件的常规过程。根据所创建组件类型的不同，某些步骤是可选步骤。

编写组件类文件：

1. （可选）导入类。（请参阅[第 125 页的“导入类”](#)）。
执行这一步骤，则无需写出包（例如，使用 **Button** 而不是 **mx.controls.Button**）即可引用类。
2. 使用 **class** 关键字定义类；使用 **extend** 关键字扩展父类。（请参阅[第 126 页的“定义类及其超类”](#)）。
3. 定义 **symbolName**、**symbolOwner** 和 **className** 变量。（请参阅[第 126 页的“标识类、元件和所有者名称”](#)）。
只有第 2 版组件才需要这些变量。
4. 定义成员变量。（请参阅[第 127 页的“定义变量”](#)）。
这些变量可用于 **getter/setter** 方法。
5. 定义一个构造函数。（请参阅[第 140 页的“关于构造函数”](#)）。

6. 定义 `init()` 方法。(请参阅第 138 页的“定义 `init()` 方法”)。
如果类扩展的是 `UIComponent`，则创建类时会调用此方法。如果类扩展的是 `MovieClip`，则从构造函数调用此方法。
7. 定义 `createChildren()` 方法。(请参阅第 139 页的“定义 `createChildren()` 方法”)。
如果类扩展的是 `UIComponent`，则创建类时会调用此方法。如果类扩展的是 `MovieClip`，则从构造函数调用此方法。
8. 定义 `size()` 方法。(请参阅第 142 页的“定义 `size()` 方法”)。
如果类扩展的是 `UIComponent`，则调整组件大小时调用此方法。此外，在创作时调整组件的实时预览大小时也会调用此方法。
9. 定义 `draw()` 方法。(请参阅第 142 页的“关于无效”)。
如果类扩展的是 `UIComponent`，则组件无效时调用此方法。
10. 添加一个元数据标记和声明。(请参阅第 128 页的“添加组件元数据”)。
添加标记和声明可使 `getter/setter` 属性显示在 Flash 的“属性”检查器和“组件”检查器中。
11. 定义 `getter/setter` 方法。(请参阅第 127 页的“使用 `getter/setter` 方法定义参数”)。
12. (可选) 为组件中使用的每个外观元素 / 链接创建变量。(请参阅第 144 页的“关于指定外观”)。
执行这一步骤，用户则可通过更改组件中的参数来设置不同的外观元素。

导入类

您可以导入类文件，这样就无需在整个代码中使用完全限定类名称。从而使代码更为简洁易读。要导入类，请在类文件的顶部使用 `import` 语句，如下所示：

```
import mx.core.UIObject;  
import mx.core.ScrollView;  
import mx.core.ext.UIObjectExtensions;
```

```
class MyComponent extends UIComponent{
```

您也可以使用通配符 (*) 导入给定包中的所有类。例如，以下语句导入 `mx.core` 包中的所有类：

```
import mx.core.*;
```

如果未在脚本中使用导入的类，那么不会在生成的 `SWF` 文件的字节代码中包含该类。因此，使用通配符导入整个包不会创建不必要的大型 `SWF` 文件。

定义类及其超类

组件类文件的定义与任何其它类文件的定义一样。使用 `class` 关键字指示类名称。类名称也必须与类文件的名称相同。使用 `extends` 关键字指示超类。有关详细信息，请参阅《学习 Flash 中的 **ActionScript 2.0**》中的第 6 章“编写自定义类文件”。

```
class MyComponentName extends UIComponent{  
  
}
```

标识类、元件和所有者名称

若要帮助 **Flash** 查找适当的 **ActionScript** 类和包，并且保留组件的命名，必须在组件的 **ActionScript** 类文件中设置 `symbolName`、`symbolOwner` 和 `className` 变量。

`symbolOwner` 变量是引用元件的 **Object** 引用。如果组件是其自己的 `symbolOwner` 或是已导入的 `symbolOwner`，则无需完全限定。

下表对这些变量进行说明：

变量	类型	说明
<code>symbolName</code>	<code>String</code>	<code>ActionScript</code> 类的名称（例如， <code>ComboBox</code> ）。此名称必须与元件的链接标识符匹配。此变量必须是静态的。
<code>symbolOwner</code>	<code>Object</code>	完全限定类名称（例如， <code>mypackage.MyComponent</code> ）。不要在 <code>symbolOwner</code> 值两边加引号，因为它是 <code>Object</code> 数据类型。此名称必须与“链接属性”对话框中的 AS 2.0 类匹配。此变量用在对 <code>createClassObject()</code> 方法的内部调用中。此变量必须是静态的。
<code>className</code>	<code>String</code>	组件类的名称。它不包括包名称，在 Flash 开发环境中也没有对应设置。设置样式属性时可以使用此变量的值。

下面的示例将 `symbolName`、`symbolOwner` 和 `className` 变量添加到 **MyButton** 类中：

```
class MyButton extends mx.controls.Button {  
    static var symbolName:String = "MyButton";  
    static var symbolOwner = myPackage.MyButton;  
    var className:String = "MyButton";  
}
```

定义变量

下面的代码范例摘自 **Button.as** 文件 (**mx.controls.Button**)。它定义一个要在类文件中使用的变量 **btnOffset**。它还定义 **__label** 和 **__labelPlacement** 变量。后两个变量带有双下划线前缀,以防止在 **getter/setter** 方法中使用,以及最后用作组件中的属性和参数时发生名称冲突。有关详细信息,请参阅《学习 Flash 中的 ActionScript 2.0》中的第 127 页的“使用 **getter/setter** 方法定义参数”。

```
/**
 * 在按下按钮时用于偏移标签和 / 或图标数字。
 */
var btnOffset:Number = 0;

/**
 * @private
 * 未指定值时出现在标签中的文本。
 */
var __label:String = "default value";

/**
 * @private
 * 默认标签位置
 */
var __labelPlacement:String = "right";
```

使用 getter/setter 方法定义参数

定义组件参数最简单的方法是添加一个使该参数成为可检查参数的公共成员变量。这可以通过使用“组件”检查器中的 **Inspectable** 标记来实现,也可以按如下所示添加 **Inspectable** 变量:

```
[Inspectable(defaultValue="strawberry")]
public var flavorStr:String;
```

但是,如果使用某一组件的代码修改了 **flavorStr** 属性,则该组件通常必须执行一个动作对其自身进行更新以响应该属性更改。例如,如果将 **flavorStr** 设置为“cherry”,则组件可以使用一个樱桃图像重绘自身,而不是使用默认的草莓图像。

对于常规成员变量,则不自动向组件发送成员变量的值已更改的通知。

getter/setter 方法是一种检测组件属性更改的简单方法。声明 **getter/setter** 方法，而不使用 **var** 声明常规变量，如下所示：

```
private var __flavorStr:String = "strawberry";

Inspectable(defaultValue="strawberry")]

public function get flavorStr():String{
    return __flavorStr;
}
public function set flavorStr(newFlavor:String) {
    __flavorStr = newFlavor;
    invalidate();
}
```

invalidate() 调用使组件使用新风格重绘自身。这就是对 **flavorStr** 属性使用 **getter/setter** 方法，而不是使用常规成员变量的优点。请参阅第 141 页的“定义 **draw()** 方法”。

要定义 **getter/setter** 方法，请记住以下要点：

- 在方法名称前面加 **get** 或 **set**，并且后跟一个空格和属性名称。
- 存储属性值的变量不能与 **getter** 或 **setter** 同名。按照惯例，**getter** 和 **setter** 变量名前面应加两条下划线。

Getter 和 **setter** 通常结合标记，用于对“属性”检查器和“组件”检查器中可见的属性进行定义。

有关 **getter/setter** 方法的更多信息，请参阅《学习 Flash 中的 **ActionScript 2.0**》中的“关于 **getter** 和 **setter** 方法”。

添加组件元数据

在外部 **ActionScript** 类文件中可以添加组件元数据标记，以便将组件参数、数据绑定属性和事件通知给编译程序。在 **Flash** 创作环境中，元数据有多种用途。

元数据标记只可以在外部 **ActionScript** 类文件中使用。**FLA** 文件中不能使用元数据标记。

元数据与类声明或单个的数据字段相关。如果属性的值是字符串，必须在该属性两边加引号。

元数据语句绑定到 **ActionScript** 文件的下一行。在定义组件属性时，请在属性声明的前一行添加元数据标记。唯一的例外是 **Event** 元数据标记。在定义组件事件时，请在类定义之外添加元数据标记，以便将事件绑定到整个类。

在下面的示例中，**Inspectable** 标记定义 flavorStr、colorStr 和 shapeStr 参数：

```
[Inspectable(defaultValue="strawberry")]
public var flavorStr:String;
[Inspectable(defaultValue="blue")]
public var colorStr:String;
[Inspectable(defaultValue="circular")]
public var shapeStr:String;
```

在“属性”检查器和“组件”检查器的“参数”选项卡中，Flash 将所有这些参数显示为 **String** 类型。

元数据标记

下表说明可以在 **ActionScript** 类文件中使用的元数据标记：

标记	说明
Inspectable	公开“组件”检查器和“属性”检查器中的属性。请参阅第 130 页的“关于 Inspectable 标记 ”。
InspectableList	标识可检查属性的哪个子集应在“属性”检查器和“组件”检查器中列出。如果不向组件的类添加 InspectableList 属性，“属性”检查器中会显示所有可检查参数。请参阅第 132 页的“关于 InspectableList 标记 ”。
Event	定义组件事件。请参阅第 132 页的“关于 Event 标记 ”。
Bindable	在“组件”检查器的“绑定”选项卡中显示属性。请参阅第 133 页的“关于 Bindable 标记 ”。
ChangeEvent	标识导致数据绑定发生的事件。请参阅第 134 页的“关于 ChangeEvent 标记 ”。
Collection	标识在“组件”检查器中公开的 collection 属性。请参阅第 134 页的“关于 Collection 标记 ”。
IconFile	指定图标文件名，该图标在“组件”面板中表示这一组件。请参阅第 135 页的“关于 IconFile 标记 ”。
ComponentTask	指定在创作环境中执行任务的一个或多个关联的 JSFL 文件的文件名。请参阅第 135 页的“关于 ComponentTask 标记 ”。

下面各节更详细地说明组件元数据标记。

关于Inspectable 标记

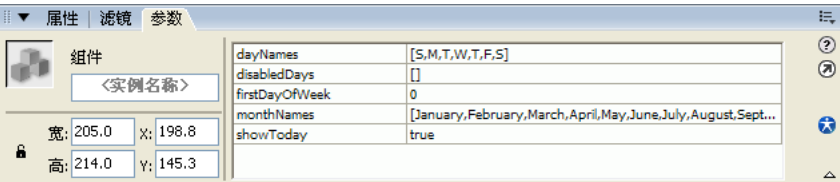
使用 **Inspectable** 标记可以指定显示在“组件”检查器和“属性”检查器中的用户可编辑（可检查）的参数。这样，您就可以在同一个位置维护可检查属性和基本的 **ActionScript** 代码。要查看组件属性，请将组件的实例拖到舞台上，然后在“组件”检查器中选择“参数”选项卡。

Collection 参数也是可检查参数。有关详细信息，请参阅第 134 页的“关于 **Collection** 标记”。

下图显示了 **DateChooser** 组件的“组件”检查器中的“参数”选项卡：



或者，您也可以在“属性”检查器的“参数”选项卡中查看组件属性的子集。



Flash 使用 **Inspectable** 标记确定应在创作环境中显示哪些参数。此标记的语法如下：

```
[Inspectable(value_type=value[,attribute=value,...])]  
property_declaration name:type;
```

下面的示例将 **enabled** 参数定义为可检查参数：

```
[Inspectable(defaultValue=true, verbose=1, category="Other")]  
var enabled:Boolean;
```

Inspectable 标记也支持宽松类型的属性，如下所示：

```
[Inspectable("danger", 1, true, maybe)]
```

元数据声明必须紧挨着属性的变量声明且在它之前，这样才能绑定到该属性。

下表介绍 **Inspectable** 标记的属性：

属性	类型	说明
defaultValue	String 或 Number	（可选）可检查属性的默认值。
enumeration	String	（可选）指定以逗号分隔的属性合法值列表。
listOffset	Number	（可选）其作用是向后兼容 Flash MX 组件。它用作 List 值的默认索引。
name	String	（可选）属性的显示名称。例如，Font Width。如果未指定，则使用属性的名称，例如 _fontWidth。
type	String	（可选）类型指定。如果省略，则使用属性的类型。下面是可接受的值： <ul style="list-style-type: none">• Array• Boolean• Color• Font Name• List• Number• Object• String
variable	String	（可选）其作用是向后兼容 Flash MX 组件。指定此参数所绑定的变量。
verbose	Number	（可选）将 verbose 属性设置为 1 的可检查属性，它不显示在“属性”检查器中，但显示在“组件”检查器中。通常用于不经常修改的属性。

不需要任何这些属性：可以将 **Inspectable** 用作元数据标记。

标记为 **Inspectable** 的所有超类属性在当前类中都是自动可检查的。如果要为当前类隐藏其中某些属性，请使用 **InspectableList** 标记。

关于 InspectableList 标记

InspectableList 标记用于指定“属性”检查器中应显示可检查属性的哪个子集。请将 **InspectableList** 与 **Inspectable** 组合使用，这样可以隐藏子类组件的继承属性。如果不给组件的类添加 **InspectableList** 标记，所有可检查的参数（包括组件父类的可检查参数）都会显示在“属性”检查器中。

InspectableList 的语法如下：

```
[InspectableList("attribute1"[,...])]  
// 类定义
```

InspectableList 标记必须紧挨着类定义且在它之前，因为它应用于整个类。

下面的示例允许 `flavorStr` 和 `colorStr` 属性在“属性”检查器中显示，但排除 **Parent** 类的其它可检查属性：

```
[InspectableList("flavorStr","colorStr")]  
class BlackDot extends DotParent {  
    [Inspectable(defaultValue="strawberry")]  
    public var flavorStr:String;  
    [Inspectable(defaultValue="blue")]  
    public var colorStr:String;  
    ...  
}
```

关于 Event 标记

Event 标记用于定义组件发出的事件。

此标记的语法如下：

```
[Event("event_name")]
```

例如，下面的代码定义 `click` 事件：

```
[Event("click")]
```

在 **ActionScript** 文件中将 **Event** 语句添加到类定义之外，以便将这些事件绑定到类，而不是绑定到类的特定成员。

下面的范例显示 **UIObject** 类的 **Event** 元数据，它处理 `resize`、`move` 和 `draw` 事件：

```
...  
import mx.events.UIEvent;  
[Event("resize")]  
[Event("move")]  
[Event("draw")]  
class mx.core.UIObject extends MovieClip {  
    ...  
}
```

若要广播特定的实例，请调用 `dispatchEvent()` 方法。请参阅[第 143 页的“使用 dispatchEvent\(\) 方法”](#)。

关于 Bindable 标记

数据绑定使各组件之间相互连接在一起。通过“组件”检查器的“绑定”选项卡可实现可视数据绑定。您可以从该选项卡添加、查看和删除组件的绑定。

虽然数据绑定适用于任何组件，但它的主要用途是将用户界面组件连接到外部数据源，例如 Web 服务和 XML 文档。这些数据源是带有属性的组件，它们可被绑定到其它组件属性。

在 `ActionScript` 类的某个属性前面使用 `Bindable` 标记可以使该属性显示在“组件”检查器的“绑定”选项卡中。可以使用 `var` 或 `getter/setter` 方法声明一个属性。如果属性同时具有 `getter` 和 `setter` 方法，只需将 `Bindable` 标记应用于其中一个方法即可。

`Bindable` 标记的语法如下：

```
[Bindable "readonly"|"writeonly",type="datatype"]
```

两个属性都是可选的。

下面的示例将变量 `flavorStr` 定义为一个可从“组件”检查器的“绑定”选项卡访问的属性：

```
[Bindable]
public var flavorStr:String = "strawberry";
```

`Bindable` 标记采用三个选项来指定属性的访问类型以及属性的数据类型。下表对这些选项进行说明：

选项	说明
<code>readonly</code>	指示在“组件”检查器中创建绑定时，只能创建将此属性用作源的绑定。但是，如果使用 <code>ActionScript</code> 创建绑定，则没有此类限制。 [Bindable("readonly")]
<code>writeonly</code>	指示在“组件”检查器中创建绑定时，此属性只能用作绑定目标。但是，如果使用 <code>ActionScript</code> 创建绑定，则没有此类限制。 [Bindable("writeonly")]
<code>type="datatype"</code>	指示数据绑定所使用的属性类型。进行类型声明之后，Flash 则使用此处声明的类型。 如果不指定此选项，数据绑定将使用 <code>ActionScript</code> 代码中声明的属性数据类型。在下面的示例中，即使 <code>x</code> 实际上是 <code>Object</code> 类型，数据绑定也将它视为 <code>DataProvider</code> 类型： [Bindable(type="DataProvider")] var x:Object;

所有组件的所有属性都可用于数据绑定。`Bindable` 标记只控制“组件”检查器中哪些属性可用于绑定。如果某个属性前面没有 `Bindable` 标记，它仍然可用于数据绑定，但必须使用 `ActionScript` 创建绑定。

使用 `ChangeEvent` 标记时，必须使用 `Bindable` 标记。

有关在 Flash 创作环境中创建数据绑定的信息，请参阅《使用 Flash》中的“数据绑定（仅限 Flash Professional）”。

关于 ChangeEvent 标记

ChangeEvent 标记会通知数据绑定：每当特定属性更改时组件都将生成一个事件。为响应该事件，数据绑定执行将该属性用作源的所有绑定。如果在组件中编写适当的 **ActionScript** 代码，则组件只生成事件。该事件应包含在由类声明的 **Event** 元数据列表中。

可以使用 **var** 或 **getter/setter** 方法声明一个属性。如果属性同时具有 **getter** 和 **setter** 方法，只需将 **ChangeEvent** 标记应用于其中一个方法即可。

ChangeEvent 标记的语法如下：

```
[Bindable]
[ChangeEvent("event")]
property_declaration or getter/setter function
```

在下面的示例中，组件在可绑定属性 **flavorStr** 的值更改时生成 **change** 事件：

```
[Bindable]
[ChangeEvent("change")]
public var flavorStr:String;
```

元数据中指定的事件发生时，**Flash** 会通知绑定该属性已更改。

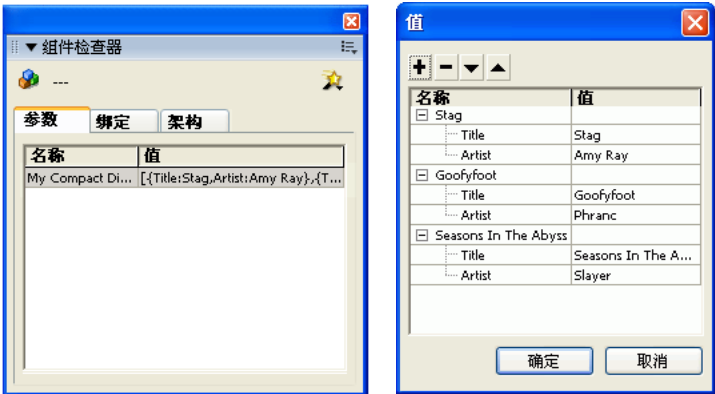
在标记中可以注册多个事件，如下例所示：

```
[ChangeEvent("change1", "change2", "change3")]
```

其中的任一事件都指示属性发生更改。不一定要发生所有事件才表示变量有更改。

关于 Collection 标记

Collection 标记用于描述在创作时可在“值”对话框中修改为项目集合的对象数组。这些对象的类型由 **collectionItem** 属性标识。**Collection** 属性包含一系列在单独类中定义的集合项目。单独类可以是 **mx.utils.CollectionImpl** 或其子类。使用 **collectionClass** 属性标识的类的方法可以访问这些单个对象。



“组件”检查器中的集合属性和单击放大镜时显示的“值”对话框。

Collection 标记的语法如下：

```
[Collection (name="name", variable="varname",
    collectionClass="mx.utils.CollectionImpl",
    collectionItem="coll-item-classname", identifier="string")]
public var varname:mx.utils.Collection;
```

下表对 Collection 标记的属性进行说明：

属性	类型	说明
name	String	（必需）显示在“组件”检查器中的集合名称。
variable	String	（必需）指向基础 Collection 对象的 ActionScript 变量（例如，可以将 Collection 参数命名为 Columns，但基础 variable 属性可以是 __columns）。
collectionClass	String	（必需）指定要将集合属性实例化的类的类型。它通常为 mx.utils.CollectionImpl，但也可以是扩展 mx.utils.CollectionImpl 的类。
collectionItem	String	（必需）指定要存储在集合中的一类集合项目。该类包含它自己的通过元数据公开的可检查属性。
identifier	String	（必需）指定在用户通过“值”对话框添加一个新集合项目时，Flash MX 用作默认标识符的可检查属性（在集合项目类中）的名称。每当用户创建一个新集合项目时，Flash MX 都会将该项目的名称设置为 identifier，外加一个唯一索引（例如，如果 identifier=name，则“值”对话框显示 name0、name1、name2，依此类推）。

有关详细信息，请参阅第 163 页的“集合属性”。

关于 IconFile 标记

您可以添加在 Flash 创作环境的“组件”面板中表示组件的图标。有关详细信息，请参阅第 160 页的“添加图标”。

关于 ComponentTask 标记

可以指定一个或者多个 Flash JavaScript (JSFL) 文件，在 Flash 创作环境中执行组件的任务。使用 ComponentTask 标记定义组件及其 JSFL 文件之间的关联，以及关联任何需要 JSFL 文件的其它文件。JSFL 文件在 Macromedia Flash 创作环境中与 JavaScript API 进行交互。



在将组件导出为 SWC 文件时，任何使用 ComponentTask 标记声明的 JSFL 任务文件和所需的依赖项文件必须与组件 FLA 文件位于同一文件夹中。

ComponentTask 标记的语法如下：

```
[ComponentTask [taskName,taskFile [,otherFile[,  ]]
```

taskName 和 taskFile 属性是必需的。otherFile 属性是可选的

下面的示例将 `SetUp.jsfl` 和 `AddNewSymbol.jsfl` 与名为 **myComponent** 的组件类相关联。`AddNewSymbol.jsfl` 需要 `testXML.xml` 文件，这将在 `otherFile` 属性中进行指定。

```
[ComponentTask("Do Some Setup","SetUp.jsfl")]
[ComponentTask("Add a new Symbol","AddNewSymbol.jsfl","testXML.xml")]
class myComponent{
    //...
}
```

下表介绍 **ComponentTask** 标记的属性：

属性	类型	说明
taskName	String	（必需）字符串形式的任务名称。此名称出现在“组件”检查器的“架构”选项卡的“任务”弹出菜单中。
taskFile	String	（必需）在创作环境中实现任务的 JSFL 文件的名称。在将组件导出为 SWC 文件时，该文件必须与组件 FLA 位于同一文件夹中。
otherFile	String	（可选）JSFL 文件所需的一个或多个文件（如 XML 文件）的名称。在将组件导出为 SWC 文件时，这些文件必须与组件 FLA 位于同一文件夹中。

定义组件参数

在构建组件时，可以添加定义组件外观和行为的参数。最常用的参数在“组件”检查器和“属性”检查器中显示为创作参数。您也可以使用 **ActionScript** 设置所有可检查参数和集合参数。在组件类文件中可以使用 **Inspectable** 标记定义这些属性（请参阅第 130 页的“关于 **Inspectable** 标记”）。

下面的示例在 **JellyBean** 类文件中设置多个组件参数，然后使用 **Inspectable** 标记将它们显示在“组件”检查器中：

```
class JellyBean{
    // 一个字符串参数
    [Inspectable(defaultValue="strawberry")]
    public var flavorStr:String;

    // 一个字符串列表参数
    [Inspectable(enumeration="sour,sweet,juicy,rotten",defaultValue="sweet")]
    public var flavorType:String;
```



```

// 一个数组参数
[Inspectable(name="Flavors", defaultValue="strawberry,grape,orange",
verbose=1, category="Fruits")]
var flavorList:Array;

// 一个对象参数
[Inspectable(defaultValue="belly:flop,jelly:drop")]
public var jellyObject:Object;

// 一个颜色参数
[Inspectable(defaultValue="#ffffff")]
public var jellyColor:Color;

// 一个 setter
[Inspectable(defaultValue="default text")]
function set text(t:String)

}

```

可以使用以下任何参数类型：

- Array
- Object
- List
- String
- Number
- Boolean
- Font Name
- Color

提醒

JellyBean 类是一个理论示例。若要查看实际示例，请查看 语言 /First Run/Classes/mx/controls 目录中随 Flash 安装的 Button.as 类文件。

关于核心函数

在组件类文件中，必须定义五个函数：`init()`、`createChildren()`、构造函数、`draw()` 和 `size()`。组件扩展 **UIComponent** 时，类文件中的这五个函数按以下顺序调用：

- `init()`
所有初始化都在 **init()** 函数调用期间进行。例如，实例成员变量可以在此时设置，组件边框可以在此时隐藏。
调用 `init()` 之后，会自动设置 `width` 和 `height` 属性。请参阅第 138 页的“定义 `init()` 方法”。
- `createChildren()`
当在时间轴中播放帧时调用。在这段时间中，组件用户可以调用方法和属性来设置组件。组件需要创建的所有子对象都在 `createChildren()` 函数中创建。请参阅第 139 页的“定义 `createChildren()` 方法”。
- 构造函数
调用以创建组件的实例。组件构造函数通常为 `空`，以避免发生初始化冲突。请参阅第 140 页的“关于构造函数”。
- `draw()`
以编程方式创建或修改的组件的任何可视元素都应在 **draw** 函数中出现。请参阅第 141 页的“定义 `draw()` 方法”。
- `size()`
每当在运行时调整组件的大小以及将组件的更新后的 `width` 和 `height` 属性传递给组件时，都会调用此函数。可以根据 **size()** 函数中组件的更新后的 `width` 和 `height` 属性来调整组件子对象的大小或对其进行移动。请参阅第 142 页的“定义 `size()` 方法”。

后面的章节中将详细介绍这些核心组件函数。

定义 `init()` 方法

Flash 在创建类时调用 `init()` 方法。此方法只在实例化组件时被调用一次。

应使用 `init()` 方法执行以下操作：

- 调用 `super.init()`。
此调用是必需的。
- 使 `boundingBox_mc` 不可见。

```
boundingBox_mc.width = 0;  
boundingBox_mc.height = 0;  
boundingBox_mc.visible = false;
```
- 创建实例成员变量。

只有在调用此方法之后，才能正确设置 width、height 和 clip 参数。

init() 方法是从 UIObject 的构造函数调用的，因此控制流攀升构造函数链，直至到达 UIObject。UIObject 的构造函数调用在最低子类上定义的 init() 方法。init() 的每个实现都应调用 super.init()，以便它的基类可以完成初始化。如果实现一个 init() 方法但没有调用 super.init()，则不会在任何基类上调用 ()init 方法，因此这些基类可能永远无法处于可用状态。

定义 createChildren() 方法

组件实现 createChildren() 方法，以便在组件中创建子对象（如其它组件）。调用 createClassObject() 或 createObject()，而不在 createChildren() 方法中调用子对象的构造函数，来实例化组件的子对象。

最好在 createChildren() 方法中调用 size()，以确保所有子级最初都设置为合适的大小。此外，在 createChildren() 方法中调用 invalidate() 可以刷新屏幕。（有关详细信息，请参阅第 142 页的“关于无效”。）

createClassObject() 方法的语法如下：

```
createClassObject(className, instanceName, depth, initObject)
```

下表对参数进行说明：

参数	类型	说明
<i>className</i>	Object	类的名称。
<i>instanceName</i>	String	实例的名称。
<i>depth</i>	Number	实例的深度。
<i>initObject</i>	Object	包含初始化属性的对象。

要调用 createClassObject()，必须知道有哪些子级，这是因为必须在 createClassObject() 调用中指定对象的名称和类型以及所有初始化参数。

下面的示例调用 createClassObject() 创建一个在组件内使用的新 **Button** 对象：

```
up_mc.createClassObject(mx.controls.Button, "submit_btn", 1);
```

在 createClassObject() 调用中设置属性，方法是将这些属性作为 *initObject* 参数的一部分进行添加。下面的示例设置 label 属性的值：

```
form.createClassObject(mx.controls.CheckBox, "cb", 0, {label:"Check this"});
```

下面的示例创建 **TextInput** 和 **SimpleButton** 组件:

```
function createChildren():Void {
    if (text_mc == undefined)
        createClassObject(TextInput, "text_mc", 0, { preferredWidth: 80,
            editable:false });
        text_mc.addEventListener("change", this);
        text_mc.addEventListener("focusOut", this);

    if (mode_mc == undefined)
        createClassObject(SimpleButton, "mode_mc", 1, { falseUpSkin:
            modeUpSkinName, falseOverSkin: modeOverSkinName, falseDownSkin:
            modeDownSkinName });
        mode_mc.addEventListener("click", this);
        size()
        invalidate()
}
```

关于构造函数

您可以识别构造函数，因为它与组件类同名。例如，下面的代码显示 **ScrollBar** 组件的构造函数:

```
function ScrollBar() {
}
```

在本例中，实例化新的滚动条时，即会调用 `ScrollBar()` 构造函数。

通常，组件构造函数应为空。有时在构造函数中设置属性会导致覆盖默认值，这与初始化调用的顺序有关。

如果组件扩展的是 **UIComponent** 或 **UIObject**，则 **Flash** 会自动调用 `init()`、`createChildren()` 和 `size()` 方法，您可以使构造函数保留为空，如下所示:

```
class MyComponent extends UIComponent{
    ...
    // 此为构造函数
    function MyComponent(){
    }
}
```

所有第 2 版组件都应定义一个 `init()` 函数，该函数在调用构造函数之后调用。初始化代码应放在组件的 `init()` 函数中。有关详细信息，请参阅下一节。

如果组件扩展的是 **MovieClip**，则可能需要调用 `init()` 方法、`createChildren()` 方法以及一个通过构造函数布置组件的方法，如下面的代码示例所示：

```
class MyComponent extends MovieClip{
    ...
    function MyComponent(){
        init()
    }

    function init():Void{
        createChildren();
        layout();
    }
    ...
}
```

有关构造函数的详细信息，请参阅《学习 Flash 中的 ActionScript 2.0》中的“编写构造函数”。

定义 draw() 方法

在 `draw()` 方法中编写代码可以创建或修改组件的可视元素。换言之，在 `draw()` 方法中，组件绘制本身以匹配其状态变量。自上次调用 `draw()` 方法以来，可能已调用了多个属性或方法，因此，应在 `draw()` 体中尽量考虑所有这些属性或方法。

但是，不应直接调用 `draw()` 方法。而应调用 `invalidate()` 方法，以便将 `draw()` 调用排队并进行批处理。这一方法可提高效率并使代码集中。（有关详细信息，请参阅第 142 页的“关于无效”。）

在 `draw()` 方法内，可以使用对 **Flash** 绘制 API 的调用来绘制边框、标尺和其它图形元素。也可以设置属性值和调用方法。此外，还可以调用 `clear()` 方法，该方法将删除可见对象。

在 **Dial** 组件的以下示例（请参阅第 107 页的“构建第一个组件”）中，`draw()` 方法将指针的旋转设置为 `value` 属性：

```
function draw():Void {
    super.draw();
    dial.needle._rotation = value;
}
```

定义 size() 方法

如果在运行时使用 `componentInstance.setSize()` 方法调整组件大小，则会调用 `size()` 函数并向该函数传递 `width` 和 `height` 属性。在组件的类文件中可以使用 `size()` 方法布置组件的内容。

至少 `size()` 方法应调用超类的 `size()` 方法 (`super.size()`)。

在 **Dial** 组件的以下示例（请参阅第 107 页的“构建第一个组件”）中，`size()` 方法使用 `width` 和 `height` 参数调整拨盘影片剪辑的大小：

```
function size():Void {  
    super.size();  
    dial._width = width;  
    dial._height = height;  
    invalidate();  
}
```

在 `size()` 方法内调用 `invalidate()` 方法，将组件标记为重绘，而不是直接调用 `draw()` 方法。有关详细信息，请参阅下一节。

关于无效

大多数情况下，**Macromedia** 不建议组件立即更新自身，而建议组件保存新属性值的副本，设置一个标记指示发生的更改，然后调用 `invalidate()` 方法。（此方法指示只有对象的视觉内容发生了更改，但子对象的大小和位置尚未更改。此方法调用 `draw()` 方法。）

在实例化组件期间，至少应调用一次无效方法。执行此调用的最常见位置是在 `createChildren()` 或 `layoutChildren()` 方法中。

发送事件

如果希望组件广播除可从父类继承的事件之外的事件，则必须在组件的类文件中调用 `dispatchEvent()` 方法。

`dispatchEvent()` 方法在 `mx.events.EventDispatcher` 类中定义，扩展 `UIObject` 的所有组件都会继承此方法。（请参阅《组件语言参考》中的“`EventDispatcher` 类”。）

在类文件的顶部还应为每个新事件添加一个 `Event` 元数据标记。有关详细信息，请参阅第 132 页的“关于 `Event` 标记”。



有关在 Flash 应用程序中处理组件事件的信息，请参阅第 55 页的第 4 章“处理组件事件”。

使用 dispatchEvent() 方法

在组件 **ActionScript** 类文件的正文中，您可以使用 `dispatchEvent()` 方法来广播事件。
`dispatchEvent()` 方法的语法如下：

```
dispatchEvent(eventObj)
```

`eventObj` 参数是一个描述事件的 **ActionScript** 对象（请参阅本节后面部分中的示例）。

在调用 `dispatchEvent()` 方法之前，必须在代码中对其进行声明，如下所示：

```
private var dispatchEvent:Function;
```

此外，还必须创建一个事件对象以传递给 `dispatchEvent()`。该事件对象包含侦听器可用来处理该事件的有关信息。

在发送事件之前可以显式地构建一个事件对象，如下面示例所示：

```
var eventObj = new Object();  
eventObj.type = "myEvent";  
eventObj.target = this;  
dispatchEvent(eventObj);
```

您也可以使用快捷语法在一行代码中设置 `type` 属性和 `target` 属性的值，并发送事件：

```
ancestorSlide.dispatchEvent({type:"revealChild", target:this});
```

在上面的示例中，由于 `target` 属性是隐式的，因此，设置该属性是可选操作。

Flash 8 文档中对每个事件的说明列出了可选和必需的事件属性。例如，`ScrollBar.scroll` 事件除了使用 `type` 和 `target` 属性外，还使用 `detail` 属性。有关详细信息，请参阅《组件语言参考》中的事件说明。

公共事件

下表列出了由各种类广播的公共事件。如果事件适用于组件，每个组件都应广播这些事件。这里没有列出所有组件的全部事件，只列出了可能会由其它组件重新使用的事件。虽然某些事件未指定任何参数，但所有事件都有隐式参数：对广播事件的对象的引用。

事件	使用
click	由 Button 组件使用，或在鼠标单击没有其它含义时使用。
change	由 List 、 ComboBox 和其它文本输入组件使用。
scroll	由 ScrollBar 和其它导致滚动（在滚动弹出菜单上滚动“缓冲器”）的控件使用。

此外，由于是从基类继承而来，所有组件都广播以下事件：

UIComponent 事件	说明
load	组件正在创建或加载其子对象。
unload	组件正在卸载其子对象。
focusIn	组件现在有输入焦点。某些 HTML 等效组件（ListBox、ComboBox、Button、Text）还可以广播 focus，但所有组件都广播 DOMFocusIn。
focusOut	组件已失去输入焦点。
move	组件已被移至新位置。
resize	组件大小已更改。

下表对一些常见的键事件进行说明：

键事件	说明
keyDown	按下某个键。code 属性包含被按下键的键控代码，ascii 属性包含它的 ASCII 代码。不要使用低级 Key 对象检查，因为 Key 对象可能尚未生成该事件。
keyUp	释放某个键。

关于指定外观

用户界面 (UI) 组件完全由附加的影片剪辑组成。这意味着 UI 组件的所有资源都可以是 UI 组件影片剪辑的外部资源，因此可由其它组件使用。例如，如果组件需要复选框功能，则可重用现有的 CheckBox 组件资源。

CheckBox 组件使用单独的影片剪辑来表示它的每个状态（FalseUp、FalseDown、Disabled、Selected 等等）。不过，您可以使自定义影片剪辑（称为外观）与这些状态相关联。在运行时，新旧影片剪辑都会导出到 SWF 文件中。旧状态会成为不可见状态，以让位于新影片剪辑。在创作时和运行时更改外观的功能称为设置外观。

若要设置组件外观，请为组件中使用的每个外观元素（影片剪辑元件）创建一个变量，并将该变量设置为该元件的链接 ID。这样，开发人员只需更改组件中的一个参数即可设置另一个外观元素，如下所示：

```
var falseUpIcon = "mySkin";
```


下面的示例显示 **CheckBox** 组件各种状态的外观变量：

```
var falseUpSkin:String = "";
var falseDownSkin:String = "";
var falseOverSkin:String = "";
var falseDisabledSkin:String = "";
var trueUpSkin:String = "";
var trueDownSkin:String = "";
var trueOverSkin:String = "";
var trueDisabledSkin:String = "";
var falseUpIcon:String = "CheckFalseUp";
var falseDownIcon:String = "CheckFalseDown";
var falseOverIcon:String = "CheckFalseOver";
var falseDisabledIcon:String = "CheckFalseDisabled";
var trueUpIcon:String = "CheckTrueUp";
var trueDownIcon:String = "CheckTrueDown";
var trueOverIcon:String = "CheckTrueOver";
var trueDisabledIcon:String = "CheckTrueDisabled";
```

关于样式

使用样式可以向某个类注册组件中的所有图形，并使该类在运行时控制图形的配色方案。组件实现中无需任何特殊代码即可支持样式。样式完全在基类和外观中实现。

要向组件添加新样式，请在组件类中调用 `getStyle("styleName")`。如果该样式已在实例、自定义样式表或全局样式表上进行了设置，则会检索到该值。如果不进行调用，则需要在全局样式表上设置一个默认样式值。

有关样式的详细信息，请参阅第 69 页的[“使用样式自定义组件的颜色和文本”](#)。

将外观注册到样式

下面的示例创建一个名为 **shape** 的组件。该组件显示一个形状，该形状为以下两个外观之一：圆形或方形。这些外观注册到了 `themeColor` 样式。

将外观注册到样式：

1. 创建一个新的 **ActionScript** 文件，然后将下面的代码复制到该文件中：

```
import mx.core.UIComponent;

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();
}
}

```

2. 将该文件另存为 **Shape.as**。
3. 创建一个新的 **Flash** 文档，然后将其另存到 **Shape.as** 所在的文件夹，且将其命名为 **Shape.fla**。
4. 在舞台上绘制一个圆形，将其选中，然后按 **F8** 转换为影片剪辑。

为该圆形指定名称和链接标识符 **circle_skin**。

5. 打开 **circle_skin** 影片剪辑，然后将下面的 **ActionScript** 放置在第一帧上，以便用样式名称 **themeColor** 注册元件：

```
mx.skins.ColoredSkinElement.setColorStyle(this, "themeColor");
```

6. 为组件创建一个新的影片剪辑。
将影片剪辑和链接标识符命名为 **Shape**。
7. 创建两个图层。将一个 **stop()** 函数放置在第一个图层的第一帧中。将元件 **circle_skin** 放置在第二帧中。

这就是组件影片剪辑。有关详细信息，请参阅第 118 页的“创建组件影片剪辑”。

8. 打开 **StandardComponents.fla** 作为外部库，然后将 **UIComponent** 影片剪辑拖到 **Shape** 影片剪辑第二帧的舞台上（与 **circle_skin** 一起）。
9. 关闭 **StandardComponents.fla**。
10. 选择库中的 **Shape** 影片剪辑，然后从“库”上下文菜单选择“组件定义”（Windows：右键单击；Mac：按住 **Control** 键单击），并输入 AS 2.0 类名 **Shape**。
11. 在舞台上使用 **Shape** 组件测试影片剪辑。

要更改主题颜色，请在实例上设置样式。下面的代码将实例名称为 **shape** 的 **Shape** 组件的颜色更改为红色：

```
shape.setStyle("themeColor",0xff0000);
```

12. 在舞台上绘制一个方形并将其转换为影片剪辑。

输入链接名称 **square_skin**，并确保选中“在第一帧导出”复选框。



由于影片剪辑未放置在组件中，因此必须选中“在第一帧导出”才能在初始化之前使用该外观。

13. 打开 square_skin 影片剪辑，然后将下面的 **ActionScript** 放置在第一帧上，以便用样式名称 themeColor 注册元件：

```
mx.skins.ColoredSkinElement.setColorStyle(this, "themeColor");
```

14. 将下面的代码放置在舞台主时间轴中的 **Shape** 组件的实例上：

```
onClipEvent(initialize){  
    themeShape = "square_skin";  
}
```

15. 在舞台上测试包含 **Shape** 的影片剪辑。结果应显示一个红色方形。

注册新样式名称

如果创建了一个新样式名称并且该样式为颜色样式，将该新名称添加到 **StyleManager.as** 文件 (First Run\Classes\mx\styles\StyleManager.as) 中的 colorStyles 对象中。下面的示例添加 shapeColor 样式：

```
// 初始化继承颜色样式集  
static var colorStyles:Object =  
{  
    barColor: true,  
    trackColor: true,  
    borderColor: true,  
    buttonColor: true,  
    color: true,  
    dateHeaderColor: true,  
    dateRollOverColor: true,  
    disabledColor: true,  
    fillColor: true,  
    highlightColor: true,  
    scrollTrackColor: true,  
    selectedDateColor: true,  
    shadowColor: true,  
    strokeColor: true,  
    symbolBackgroundColor: true,  
    symbolBackgroundDisabledColor: true,  
    symbolBackgroundPressedColor: true,  
    symbolColor: true,  
    symbolDisabledColor: true,  
    themeColor: true,  
    todayIndicatorColor: true,  
    shadowCapColor: true,  
    borderCapColor: true,  
    focusColor: true,  
    shapeColor: true  
};
```

将新样式名称分别注册到每个外观影片剪辑的第一帧上的圆形和方形外观，如下所示：

```
mx.skins.ColoredSkinElement.setColorStyle(this, "shapeColor");
```

通过设置实例的样式，颜色可以随新样式名称更改，如下所示：

```
shape.setStyle("shapeColor",0x00ff00);
```

在组件内组合现有组件

在本节中，您将构建一个可以组合 **Label**、**TextInput** 和 **Button** 组件的简单的 **LogIn** 组件。本教程演示如何在新组件中通过添加未经编译的 **Flash (FLA)** 库元件来组合现有组件。已完成的组件文件 **LogIn.fla**、**LogIn.as** 和 **LogIn.swf** 位于硬盘上的以下示例文件夹中：

- 在 Windows 中：the C:\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\Login folder。
- 在 Macintosh 上：HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/Login 文件夹。

LogIn 组件提供输入名称和密码的界面。**LogIn** 的 API 有两个属性 **name** 和 **password**，用来设置和获取名称和密码 **TextInput** 字段中的字符串值。**LogIn** 组件还将在用户单击标记为“**LogIn**”的按钮时发送一个“**click**”事件。

- [第 148 页的“创建 LogIn Flash \(FLA\) 文件”](#)
- [第 151 页的“LogIn 类文件”](#)
- [第 155 页的“测试和导出 LogIn 组件”](#)

创建 LogIn Flash (FLA) 文件

首先创建一个将容纳组件元件的 **Flash (FLA)** 文件。

若要创建 LogIn FLA 文件：

1. 在 **Flash** 中，选择“文件”>“新建”，然后创建一个新文档。
2. 选择“文件”>“另存为”，将文件另存为 **LogIn.fla**。
3. 选择“插入”>“新建元件”。将其命名为 **LogIn**，然后选择影片剪辑类型单选按钮。
如果“创建新元件”对话框中的“链接”部分未打开，则单击“高级”按钮以显示该部分。
4. 选择“为 **ActionScript** 导出”并取消选择“在第一帧导出”。
5. 输入链接标识符。

默认的链接标识符为 **LogIn**。这些步骤的其余部分均假定您使用的是默认值。

6. 在“AS 2.0 类”文本框中输入 **LogIn**。此值是组件类名称。
如果将类放入了包中，请输入完整的包名。例如，`mx.controls.CheckBox` 表示 `mx.controls` 包中的 `CheckBox` 类。
7. 单击“确定”。
Flash 打开元件编辑模式。
8. 插入一个新图层。将顶部图层命名为**动作**，将底部图层命名为**资源**。
9. 选择 **Assets** 图层的第二帧，并插入一个关键帧 (F6)。
该组件影片剪辑的结构为：一个 **Actions** 图层和一个 **Assets** 图层。**Actions** 图层有一个关键帧，**Assets** 图层有两个关键帧。
10. 选择 **Actions** 图层的第一帧，然后打开“动作”面板 (F9)。输入 `stop()` 全局函数。
这样可防止影片剪辑进入第二帧。
11. 选择“文件”>“导入”>“打开外部库”，然后从 **Configuration/ComponentFLA** 文件夹中选择 **StandardComponents.fla** 文件。
 - 在 Windows 中：\Program Files\Macromedia\Flex 8\语言\Configuration\ComponentFLA\StandardComponents.fla。
 - 在 Macintosh 上：HD/Applications/Macromedia Flex 8/Configuration/ComponentFLA/StandardComponents.fla



有关文件夹位置的信息，请参阅“使用 Flex”中的“与 Flex 一起安装的 Configuration 文件夹”。

12. 选择 **Assets** 图层的第二帧。在 **StandardComponents.fla** 库中，浏览到 **Flash UI Components 2** 文件夹。将 **Button**、**Label** 和 **TextInput** 组件元件拖到 **Assets** 图层的第二帧。

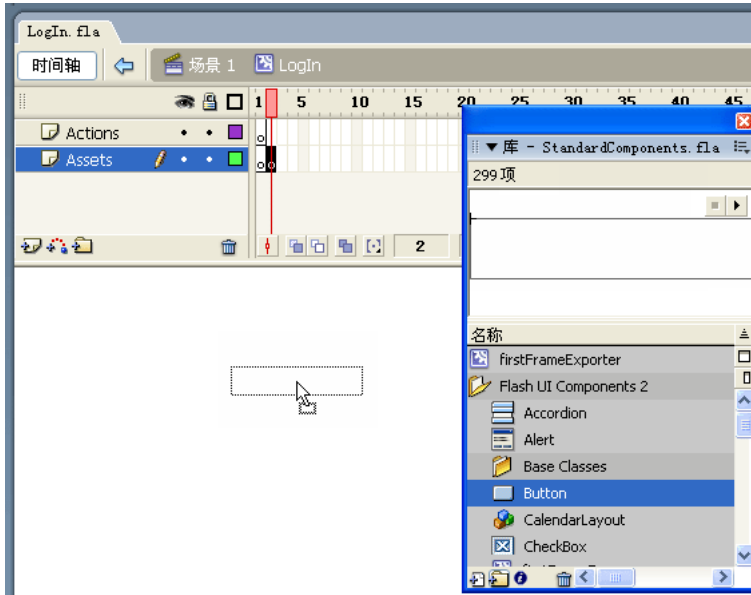
这些组件的资源依赖项会自动复制到 **LogIn.fla** 库中。

所有组件资源都已添加到 **Assets** 图层的第二帧中。由于 **Actions** 图层的第一帧上有一个 `stop()` 全局函数，所以第二帧中的资源排列在舞台时用户看不到这些资源。

之所以将资源添加到第二帧，有两个原因：

- 这样做可使所有资源自动复制到库中，并可用于动态实例化和访问它们的方法、属性和事件。

- 将资源放入帧中可以确保在传送影片流时能够更流畅地加载这些资源，所以，无需将库中的资源设置为在第一帧之前导出。此方法可防止出现数据传输的初始峰值，这种情况可能会导致下载延迟或长时间暂停。



将 Button 组件元件从 StandardComponents.fla 库拖到 LogIn.fla 的 Assets 图层的第二帧中

13. 关闭 StandardComponents.fla 库。
14. 在 Assets 图层中，选择第一帧，将 BoundingBox 影片剪辑从 LogIn.fla 库（位于 Component Assets 文件夹中）拖到舞台上。
15. 将 BoundingBox 实例命名为 **boundingBox_mc**。
16. 使用“信息”面板将 BoundingBox 的大小调整为 LogInFinal 影片剪辑的大小 (340, 150)，并将其放置在 0, 0。

BoundingBox 实例用于创建组件的实时预览，并允许用户在创作期间调整组件的大小。必须设置边框大小以使边框能够包含组件中的所有图形元素。

提醒

如果要扩展一个组件（包括任何第 2 版组件），则必须保留已被该组件使用的实例名称，因为其代码会引用这些实例名称。例如，如果包括已使用实例名称 boundingBox_mc 的第 2 版组件，则不要重命名该实例。对于您自己的组件，您可以选择和同一范围内现有的名称不冲突的任何唯一实例名称。

17. 在库中选择 **LogIn** 影片剪辑，然后从“库”上下文菜单中选择“组件定义”（Windows: 右键单击；Mac: 按住 **Control** 键单击）。
18. 在“AS 2.0 类”文本框中，输入 **LogIn**。
这个值是 **ActionScript** 类的名称。如果该类在包中，则值为整个包。例如，**mx.controls.CheckBox** 表示 **mx.controls** 包中的 **CheckBox** 类。
19. 单击“确定”。
20. 保存该文件。

LogIn 类文件

下面的代码为 **LogIn** 组件的 **ActionScript** 类。请阅读代码中的注释，了解每一部分的说明。（有关组件类文件的元素的详细信息，请参阅第 124 页的“[组件类文件概述](#)”）。

要创建此文件，可以在 **Flash** 中创建一个新的 **ActionScript** 文件，也可以使用其它任何文本编辑器创建。将文件另存为 **LogIn.as** 并放在 **LogIn.fla** 文件所在的文件夹中。

可以在新的 **LogIn.as** 文件中复制或键入以下 **LogIn** 组件 **ActionScript** 类代码。键入代码要比复制代码更有助于快速熟悉组件代码的每个元素。

```
/* 导入包，以便能够从此类
   直接引用这些包。 */
import mx.core.UIComponent;
import mx.controls.Label;
import mx.controls.TextInput;
import mx.controls.Button;

// 事件元数据标记
[Event("change")]
[Event("click")]
class LogIn extends UIComponent
{
    /* 组件必须将这些成员变量声明为
       组件框架中的相应组件。 */
    static var symbolName:String = "LogIn";
    static var symbolOwner:Object = LogIn;
    var className:String = "LogIn";

    // 组件的图形表示形式。
    private var name_label:MovieClip;
    private var password_label:MovieClip;
    private var name_ti:MovieClip;
    private var password_ti:MovieClip;
    private var login_btn:MovieClip;
    private var boundingBox_mc:MovieClip;
    private var startDepth:Number = 10;
```

```

/* 可通过 getter/setter 公开使用的私有成员变量。
   这些表示名称和密码 InputText 字符串值。 */
private var __name:String;
private var __password:String;

/* 构造函数:
   为所有类所需要的同时, 第 2 版组件还需要
   构造函数为空, 有零个参数。
   构造类实例后, 所有初始化都发生
   在所需的 init 方法中。 */
function LogIn() {
}

/* 初始化代码:
   第 2 版组件需要 init 方法。它还必须
   用 super.init() 调用它的父类 init() 方法。
   组件扩展 UIComponent 需要 init 方法。 */
function init():Void {
    super.init();
    boundingBox_mc._visible = false;
    boundingBox_mc._width = 0;
    boundingBox_mc._height = 0;
}

/* 创建启动时所需的子对象:
   组件扩展 UIComponent 需要
   createChildren 方法。 */
public function createChildren():Void {
    name_label = createObject("Label", "name_label", this.startDepth++);
    name_label.text = "Name:";
    name_label._width = 200;
    name_label._x = 20;
    name_label._y = 10;

    name_ti = createObject("TextInput", "name_ti",
this.startDepth++, {_width:200,_heigh:22,_x:20,_y:30});
    name_ti.html = false;
    name_ti.text = __name;
    name_ti.tabIndex = 1;
    /* 将此文本输入字段设置为具有焦点。
       注意: 确保在 Flash Debugger 中选择 “控制” > “禁用快捷键”
       (如果尚未选择), 否则
       测试时可能不设置焦点。 */
    name_ti.setFocus();

    name_label = createObject("Label", "password_label",
this.startDepth++, {_width:200,_heigh:22,_x:20,_y:60});
    name_label.text = "Password:";

```



```

        password_ti = createObject("TextInput", "password_ti",
this.startDepth++, {_width:200,_height:22,_x:20,_y:80});
        password_ti.html = false;
        password_ti.text = __password;
        password_ti.password = true;
        password_ti.tabIndex = 2;

        login_btn = createObject("Button", "login_btn",
this.startDepth++, {_width:80,_height:22,_x:240,_y:80});
        login_btn.label = "LogIn";
        login_btn.tabIndex = 3;
        login_btn.addEventListener("click", this);

        size();

    }

    /* 第 2 版组件需要 draw 方法。
       通过调用 invalidate() 使组件
       无效后，即调用该方法。
       这将在一次重绘中成批进行更改，
       而不是逐个执行这些更改。此方法
       可以使效率更高，代码更集中。 */
    function draw():Void {
        super.draw();
    }

    /* 组件的大小更改时，
       即调用 size 方法。这一特性可用来调整子级大小，
       组件扩展 UIComponent 需要 size 方法。 */
    function size():Void {
        super.size();
        // 导致需要时进行重绘。
        invalidate();
    }

    /* 事件处理函数：
       在 LogIn 按钮接收到鼠标单击时即由 LogIn 按钮调用。
       由于希望此事件可以在此组件范围外
       访问，因此使用 dispatchEvent 发送 click 事件。 */
    public function click(evt){
        // 用输入字段内容更新成员变量。
        __name = name_ti.text;
        __password = password_ti.text;
        // 当按钮激发一个 click 事件时即发送该事件。
        dispatchEvent({type:"click"});
    }

```

```

/* 这是 name 属性的 getter/setter。
   [Inspectable] 元数据使属性显示在
   “属性”检查器中，并允许设置
   默认值。在值更改时，通过使用 getter/setter，您可以
   调用 invalidate 并强制组件重绘。 */
[Bindable]
[ChangeEvent("change")]
[Inspectable(defaultValue="")]
function set name(val:String){
    __name = val;
    invalidate();
}

function get name():String{
    return(__name);
}

[Bindable]
[ChangeEvent("change")]
[Inspectable(defaultValue="")]
function set password(val:String){
    __password=val;
    invalidate();
}

function get password():String{
    return(__password);
}
}

```

测试和导出 LogIn 组件

您已创建了一个包含图形元素、基类和类文件（包含 LogIn 组件的所有功能）的 Flash 文件。现在即可测试组件。

测试组件最好在工作过程中，尤其是在编写类文件时进行。开发过程中最快速的测试方法，是将组件转换为编译剪辑，然后在组件的 FLA 文件中使用。

组件彻底创建完成之后，将它导出为 SWC 文件。有关详细信息，请参阅第 157 页的“导出和分发组件”。

测试 LogIn 组件：

1. 在 LogIn.fla 文件中，选择库中的 LogIn 影片剪辑，并从“库”上下文菜单中选择“转换为编译剪辑”（Windows：右键单击；Mac：按住 Control 键单击）。

一个名为 LogIn SWF 的编译剪辑即添加到库中。现在所做的只是为进行测试而编译影片剪辑。否则，请按照本节后面部分中的说明来导出 LogIn 影片剪辑。



如果已创建了一个编译剪辑（例如，这是第二次或第三次测试），则会出现一个“解决库冲突”对话框。选择“替换现有项目”，将新版本添加到文档中。

2. 将 LogIn SWF 拖到舞台主时间轴的第一帧上（确保您处在第 1 个场景的主时间轴，而不是影片剪辑的时间轴中）。

您可以在“参数”选项卡或“组件”检查器中设置 name 和 password 属性。如果希望在用户输入内容之前显示默认文本（例如“在此处输入您的姓名”），上述方法将非常有用。在设置 name 和 / 或 password 属性后，名称和密码 InputText 子组件的默认文本将会在运行时相应地更改。

要在运行时测试 value 属性，请将舞台上的 LogIn 实例命名为 myLogin，然后将下面的代码添加到主时间轴的第一帧：

```
// 创建一个可在其中查看登录值的文本字段。
createTextField("myLoginValues",10,10,10,340,40)
myLoginValues.border = true;
// 用来响应登录组件实例中被发送的 click 事件的事件处理函数。
function click(evt){
/* 此处会进行身份验证。
   例如名称和密码会被传送到 Web 服务，该服务对名称和密码进行身份验证，然后返回会话 ID
   和 / 或授予用户的权限角色。*/
  myLoginValues.text = "Processing...\r";
  myLoginValues.text += "Name: " + myLogin.name + " Password: " +
  myLogin.password;
}

myLogin.addEventListener("click",this);
```

3. 选择“控制” > “测试影片”，在 Flash Player 中测试此组件。

提醒

由于是在原始文档中测试此组件，您可能会看到一条警告消息，指出两个元件有相同的链接标识符。此组件仍将运行。在实际运用中，您将使用另一文档中的新组件，这样链接标识符应该是唯一的。

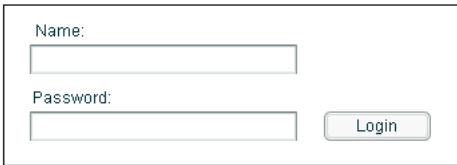
导出 LogIn 组件：

1. 在 LogIn.fla 文件中，在库中选择 LogIn 影片剪辑，并从“库”上下文菜单中选择“组件定义”（Windows：右键单击；Mac：按住 Control 键单击）。
2. 选中“显示在‘组件’面板中”。
3. 单击“确定”。
4. 在 LogIn.fla 文件中，再次选择库中的 LogIn 影片剪辑，并从“库”上下文菜单中选择“导出 SWC 文件”（Windows：右键单击；Mac：按住 Control 键单击）。
5. 选择一个位置来保存 SWC 文件。

如果将文件保存在用户级别 Configuration 文件夹中的 Components 文件夹下，则无需重新启动 Flash 即可重新加载“组件”面板，组件将显示在该面板中。

提醒

有关文件夹位置的信息，请参阅《Flash 入门》中的“随 Flash 安装的配置文件夹”。



完成的 LogIn 组件

导出和分发组件

Flash 将组件导出为组件包（SWC 文件）。组件可以分发给 SWC 文件或 FLA 文件。（有关将组件作为 FLA 分发的信息，请参阅 Macromedia DevNet 上的文章，网址为 www.macromedia.com/support/flash/applications/creating_comps/creating_comps12.html。）

分发组件的最好方法是将它导出为 SWC 文件，这是因为 SWC 文件包含全部 ActionScript、SWF 文件以及使用组件所需的其它可选文件。如果要同时开发某个组件和使用该组件的应用程序，则 SWC 文件也十分有用。

SWC 文件可用于分发给 Macromedia Flash 8、Macromedia Dreamweaver MX 2004 和 Macromedia Director MX 2004 中使用的组件。

无论开发组件的目的是为了供他人使用还是自己使用，对作为组件开发的正在进行部分的 SWC 文件进行测试都非常重要。例如，组件的 FLA 文件中不会出现的问题可能会在 SWC 文件中出现。

本节对 SWC 文件进行说明，并解释如何在 Flash 中导入和导出 SWC 文件。

了解 SWC 文件

SWC 文件是类似 zip 的文件（通过 PKZIP 归档格式打包和展开），它由 Flash 创作工具生成。

下表对 SWC 文件的内容进行说明：

文件	说明
catalog.xml	（必需）列出组件包及其各个组件的内容，并用作 SWC 文件内其它文件的目录。
ActionScript (AS) 文件	如果使用 Flash Professional 8 创建组件，源代码就是包含组件类声明的一个或多个 ActionScript 文件。 在扩展组件时，编译程序使用源代码进行类型检查。AS 文件不由创作工具编译，这是因为编译后的字节代码已在实现 SWF 文件中。 源代码可能包含内部类定义，这些定义中不包含任何函数主体，其目的仅仅是进行类型检查。
SWF 文件	（必需）实现组件的 SWF 文件。在单个 SWF 文件内可定义一个或多个组件。如果组件是使用 Flash 8 创建的，则每个 SWF 文件只导出一个组件。
实时预览 SWF 文件	（可选）如果指定，这些 SWF 文件即可在创作工具中用于实时预览。如果省略，则实现组件的 SWF 文件用于实时预览。几乎所有情况下均可省略“实时预览”SWF 文件，只有在组件外观取决于动态数据（例如，显示 Web 服务调用结果的文本字段）时，才应包含此文件。
SWD 文件	（可选）与实现 SWF 文件对应的 SWD 文件，可用于调试 SWF 文件。此文件名始终与 SWF 文件的文件名相同，但扩展名为 .swd。

文件	说明
PNG 文件	(可选) 包含 18 x 18、每像素 8 位图标的 PNG 文件，可用在创作工具用户界面中显示组件图标。如果未提供图标，则显示默认图标。(请参阅第 160 页的“添加图标”。)
“属性”检查器 SWF 文件	(可选) 一个 SWF 文件，可在创作工具中用作自定义“属性”检查器。如果省略此文件，则向用户显示默认“属性”检查器。

从 Flash 环境生成 SWC 文件后，可以选择在 SWC 文件中包含其它文件。例如，您可能需要包含 Read Me 文件，如果需要用户访问组件的源代码，可能还需要包含 FLA 文件。若要添加其它文件，请使用 Macromedia Extension Manager (请访问 www.macromedia.com/exchange/em_download/)。

SWC 文件展开到单个目录中，因此每个组件必须具有唯一的文件名，以免发生冲突。

导出 SWC 文件

Flash 提供了通过将影片剪辑导出为 SWC 文件来导出 SWC 文件的功能。在导出 SWC 文件时，Flash 会报告编译时错误，就像在测试 Flash 应用程序一样。

导出 SWC 文件有两个原因：

- 分发完成的组件
- 在开发期间进行测试

为完成的组件导出 SWC 文件

组件可以导出为 SWC 文件，该文件包含全部 ActionScript、SWF 文件以及使用组件所需的其它可选文件。

为完成的组件导出 SWC 文件：

1. 在 Flash 库中选择组件影片剪辑。
2. 右键单击 (Windows) 或按住 Control 键单击 (Mac)，打开“库”上下文菜单。
3. 从“库”上下文菜单中选择“导出 SWC 文件”。
4. 保存 SWC 文件。

在开发期间测试 SWC 文件

在开发的不同阶段，最好将组件导出为 SWC 文件并在应用程序中进行测试。如果将 SWC 文件导出到用户级别 Configuration 文件夹中的 Components 文件夹，无需退出再重新启动 Flash 即可重新加载“组件”面板。

若要在开发期间测试 SWC 文件，请执行以下操作：

1. 在 Flash 库中选择组件影片剪辑。
2. 右键单击 (Windows) 或按住 Control 键单击 (Mac)，打开“库”上下文菜单。
3. 从“库”上下文菜单中选择“导出 SWC 文件”。
4. 浏览到用户级别 Configuration 文件夹中的 Components 文件夹。

Configuration/Components



有关文件夹位置的信息，请参阅《Flash 入门》中的“随 Flash 安装的配置文件夹”。

5. 保存 SWC 文件。
6. 从“组件”面板的选项菜单中选择“重新加载”。
组件即出现在“组件”面板中。
7. 将组件从“组件”面板拖到文档中。

将组件 SWC 文件导入 Flash

在向其它开发人员分发组件时，您可以包含以下说明，以便他们能够立即安装和使用组件。

导入 SWC 文件：

1. 将 SWC 文件复制到 Configuration/Components 目录中。
2. 重新启动 Flash。

组件的图标应显示在“组件”面板中。

组件开发的最后一些步骤

组件创建完成并准备好打包之后，即可添加图标和工具提示。要确保已完成所有必需的步骤，请参阅第 161 页的“组件开发检查列表”。

添加图标

您可以添加在 Flash 创作环境的“组件”面板中表示组件的图标。

添加组件的图标：

1. 创建新图像。

图像必须为 18 x 18 像素，并且必须保存为 PNG 格式。它的 Alpha 透明度必须是 8 位，左上角的像素必须是透明的，以支持遮罩。

2. 在组件 `ActionScript` 类文件中的类定义之前添加以下定义：

```
[IconFile("component_name.png")]
```

3. 将图像添加到 FLA 文件所在的目录。在导出 SWC 文件时，Flash 将在归档的根级包含该图像。

添加工具提示

当用户将鼠标滚到 Flash 创作环境中“组件”面板上的组件名称或图标上时，工具提示就会显示出来。

在“组件定义”对话框中可以定义工具提示。可以从组件 FLA 文件的“库”选项菜单访问此对话框（Windows：右键单击；Mac：按住 **Control** 键单击）。

在“组件定义”对话框中添加工具提示：

1. 在 Flash 中打开组件的 FLA 文件时，确保“库”可见（“窗口” > “库”菜单）。

2. 单击“库”选项菜单（Windows：右键单击；Mac：按住 **Control** 键单击）。

“库”选项菜单位于“库”标题栏的右侧，显示为一个三条线和向下三角形的图标。

3. 选择“组件定义”选项。

4. 在“组件定义”对话框的“选项”下，选择“显示在‘组件’面板中”。

“工具”提示文本框即会变为可编辑文本框。

5. 在“工具”提示文本框中输入组件的工具提示文本。

6. 单击“确定”以保存更改。

组件开发检查列表

在设计组件时，请遵循以下做法：

- 尽量使文件保持最小。
- 通过使功能通用来让组件尽量保持可重用。
- 不使用图形元素，而使用 **RectBorder** 类 (`mx.skins.halo.RectBorder`) 来绘制对象周围的边框。（请参阅《组件语言参考》中的“**RectBorder** 类”。）
- 使用基于标记的外观设置。
- 定义 `symbolName`、`symbolOwner` 和 `className` 变量。
- 假设初始状态。由于样式属性现在位于对象上，因此可以设置样式和属性的初始设置，这样，在构造对象时，初始化代码就不必设置这些样式和属性，除非用户覆盖默认状态。
- 在定义元件时，除非绝对必要，否则不要选择“在第一帧导出”选项。**Flash** 只是在您的 **Flash** 应用程序使用组件前才加载组件，因此，如果选择此选项，**Flash** 会在其父组件的第一帧中预加载该组件。通常不在第一帧中预加载组件的原因是出于 **Web** 上的一些考虑：组件在预加载器开始之前加载，从而使预加载器无效。
- 避免使用多帧影片剪辑（两帧 **Assets** 图层除外）。
- 总是实现 `init()` 和 `size()` 方法并分别调用 `Super.init()` 和 `Super.size()`，但不要放置不必要的代码。
- 避免使用绝对引用，如 `_root.myVariable`。
- 使用 `createClassObject()`，而不要使用 `attachMovie()`。
- 使用 `invalidate()` 和 `invalidateStyle()` 调用 `draw()` 方法，而不要显式调用 `draw()`。
- 在将 **Flash** 组件组合到您的组件中时，使用它们位于 **Configuration/ComponentFLA** 文件夹的 **StandardComponents.fla** 文件的库中的未经编译的影片元件。

集合属性

在 Macromedia Flash 中创建新的自定义组件时，可以使用户能够编辑属性值。这些属性称为集合属性。用户可以在“值”对话框中编辑这些属性的值。“值”对话框可从组件的“参数”选项卡内的某个文本框打开。

组件通常包括用于特定任务的功能，同时还保留了使组件用户能更改一系列要求的灵活性。为使组件具有灵活性，组件内的属性需要具有灵活性，换句话说，对于某些组件，组件用户和属性值都可以更改属性。

利用集合属性可以在对象模型中创建数量不确定的可编辑属性。Flash 提供的 **Collection** 类可以帮助您通过“组件”检查器管理这些属性。

具体地讲，**Collection** 类是用来管理相关对象组的帮助器类，组中的每个对象称为一个集合项目。如果将组件的某个属性定义为集合项目，并通过“组件”检查器将其提供给用户，用户便可在创作过程中使用“值”对话框对集合项目进行添加、删除和修改。



请按下述方法定义集合和集合项目：

- 在组件的 **ActionScript** 文件中使用 **Collection** 元数据标记定义集合属性。有关详细信息，请参阅第 134 页的“关于 **Collection** 标记”。
- 在自身包含可检查属性的单独 **ActionScript** 文件中，将某个集合项目定义为类。

在 Flash 中，集合使您可以更轻松地以编程方式管理相关项目组。（在 Flash 的早期版本中，组件作者是通过多个以编程方式实现同步的数组管理相关项目组的）。

除了“值”对话框以外，Flash 还提供了 **Collection** 和 **Iterator** 接口，用于以编程方式管理 **Collection** 的实例和值。请参阅《组件语言参考》中的“**Collection** 接口（仅限 Flash Professional）”和“**Iterator** 接口（仅限 Flash Professional）”。

本章包含下列各节：

定义集合属性	164
简单集合示例	165
为集合项目定义类	167
以编程方式访问集合信息	167
将具有集合的组件导出为 SWC 文件	169
使用具有集合属性的组件	170

定义集合属性

在组件的 **ActionScript** 文件中使用 **Collection** 标记定义集合属性。有关详细信息，请参阅第 134 页的“关于 **Collection** 标记”。

提醒

本节假定您已了解如何创建组件和可检查的组件属性。

定义集合属性：

1. 为组件创建一个 **FLA** 文件。请参阅第 118 页的“创建组件影片剪辑”。
2. 创建一个 **ActionScript** 类。请参阅第 123 页的“创建 **ActionScript** 类文件”。
3. 在 **ActionScript** 类中，插入一个 **Collection** 元数据标记。有关详细信息，请参阅第 134 页的“关于 **Collection** 标记”。
4. 在组件的 **ActionScript** 文件中，为集合定义 **get** 和 **set** 方法。
5. 向 **FLA** 文件添加工具类，方法是选择“窗口” > “其它面板” > “公用库” > “类”，然后将 **UtilsClasses** 拖动到组件的库中。

UtilsClasses 包含用于 **Collection** 接口的 **mx.utils.*** 包。

提醒

由于与 **UtilsClasses** 关联的是 **FLA** 文件，而不是 **ActionScript** 类，因此若在查看组件的 **ActionScript** 类时进行语法检查，Flash 会抛出一个编译器错误。

6. 编写一个包含集合项目属性的类。
请参阅第 167 页的“为集合项目定义类”。

简单集合示例

下面是组件类文件的一个简单示例，该文件名为 **MyShelf.as**。此示例包含从 **UIObject** 类继承的组件的一个集合属性以及至少应有的一组导入、方法和声明。

如果在此示例中导入 **mx.utils.***，则不再要求来自 **mx.utils** 的类名称是全限定类名。例如，**mx.utils.Collection** 可以写为 **Collection**。

```
import mx.utils.*;
// 标准类声明
class MyShelf extends mx.core.UIObject
{
    // 所有类必需的变量
    static var symbolName:String = "MyShelf";
    static var symbolOwner:Object = Object(MyShelf);
    var className:String = "MyShelf";

    // Collection 元数据标记和属性
    [Collection(variable="myCompactDiscs",name="My Compact Discs",collectionClass="mx.utils.CollectionImpl",collectionItem="CompactDisc", identifier="Title")]

    // 集合的 get 和 set 方法
    public function get MyCompactDiscs():mx.utils.Collection
    {
        return myCompactDiscs;
    }
    public function set MyCompactDiscs(myCDs:mx.utils.Collection):Void
    {
        myCompactDiscs = myCDs;
    }

    // 私类成员
    private var myCompactDiscs:mx.utils.Collection;

    // 必须编写一个对集合项目类的引用，
    // 才能强制编译器在 SWC 中将其作为
    // 依赖项提供
    private var collItem:CompactDisc;

    // 必须编写一个对 mx.utils.CollectionImpl 类的引用，
    // 才能强制编译器在 SWC 中将其作为
    // 依赖项提供
    private var coll:mx.utils.CollectionImpl;

    // 所有类必需的方法
    function init(Void):Void {
        super.init();
    }
    function size(Void):Void {
        super.size();
    }
}
```

创建伴随此类的 FLA 文件以便进行测试：

1. 在 Flash 中，选择“文件”>“新建”，然后创建一个 Flash 文档。
2. 选择“插入”>“新建元件”。为其指定名称、链接标识符和 AS 2.0 类名称 **MyShelf**。
3. 取消选择“在第一帧导出”，然后单击“确定”。
4. 在库中选择 MyShelf 元件，然后从“库”选项菜单中选择“组件定义”。输入 ActionScript 2.0 类名称 **MyShelf**。
5. 选择“窗口”>“公用库”>“类”，然后将 UtilClasses 拖动到 MyShelf.fla 的库中。
6. 在 MyShelf 元件的时间轴中，将一个图层命名为资源。创建另一个图层，并将其命名为动作。
7. 在“动作”图层的第 1 帧上放置一个 stop() 函数。
8. 选择“资源”图层的第 2 帧，然后选择“插入”>“时间轴”>“关键帧”。
9. 从 Configuration/ComponentFLA 文件夹打开 StandardComponents.fla 文件，然后将一个 UIObject 实例拖动到“资源”图层第 2 帧内的 MyShelf 舞台上。
您必须在组件的 FLA 文件中提供 UIObject，因为如上面的类文件所示，MyShelf 会扩展 UIObject。
10. 在“资源”图层的第 1 帧中绘制一个工具架。
这可以是一个简单的矩形；它是 MyShelf 组件的一种可视表示形式，仅用于学习的目的。
11. 在库中选择 MyShelf 影片剪辑，然后选择“转换为编译剪辑”。



此时您应该已经创建了 CompactDisc 类；否则在转换为编译剪辑时会遇到编译器错误。

为集合项目定义类

在单独的 `ActionScript` 类中为集合项目编写属性，定义方法如下所示：

- 对类进行定义，使其不扩展 `UIObject` 或 `UIComponent`。
- 使用 `Inspectable` 标记定义所有属性。
- 将所有属性定义为变量。不要使用 `get` 和 `set` (`getter/setter`) 方法。

以下是一个集合项目类文件（名为 `CompactDisc.as`）的简单示例。

```
class CompactDisc{
    [Inspectable(type="String", defaultValue="Title")]
    var title:String;
    [Inspectable(type="String", defaultValue="Artist")]
    var artist:String;
}
```

要查看 `CompactDisc.as` 类文件，请参阅第 165 页的“简单集合示例”。

以编程方式访问集合信息

Flash 使您可以通过 `Collection` 和 `Iterator` 接口以编程方式访问集合数据。`Collection` 接口允许您在集合中添加、修改和删除项目。`Iterator` 接口允许您在集合中遍历项目。

有两种使用 `Collection` 和 `Iterator` 接口的方案：

- 第 167 页的“访问组件类 (AS) 文件中的集合信息”
- 第 168 页的“使用 Flash 应用程序在运行时访问集合项目”

高级开发人员还能以编程方式创建、填充、访问和删除集合；有关更多信息，请参阅《组件语言参考》中的“`Collection` 接口（仅限 Flash Professional）”。

访问组件类 (AS) 文件中的集合信息

在组件类文件中，您可以编写可与在创作过程中或运行时定义的集合项目进行交互的代码。

要访问组件类文件中的集合项目信息，可以使用以下方法之一：

- `Collection` 标记包含一个 `variable` 属性，可为其指定类型为 `mx.utils.Collection` 的变量。使用此变量访问集合，如本例所示：

```
[Collection(name="LinkButtons", variable="__linkButtons",
    collectionClass="mx.utils.CollectionImpl", collectionItem="ButtonC",
    identifier="ButtonLabel")]
public var __linkButtons:mx.utils.Collection;
```

- 通过调用 `Collection.getIterator()` 方法访问集合项目的 **Iterator** 接口，如本例所示：

```
var itr:mx.utils.Iterator = __linkButtons.getIterator();
```

- 使用 **Iterator** 接口跟踪集合中的项目。`Iterator.next()` 方法的返回类型为 **Object**，因此必须定义集合项目的类型，如本例所示：

```
while (itr.hasNext()) {
    var button:ButtonC = ButtonC(itr.next());
    ...
}
```

- 根据特定的应用程序访问集合项目属性，如本例所示：

```
item.label = button.ButtonLabel;

if (button.ButtonLink != undefined) {
    item.data = button.ButtonLink;
}
else {
    item.enabled = false;
}
```

使用 Flash 应用程序在运行时访问集合项目

如果某个 **Flash** 应用程序使用具有集合属性的组件，您就可以在运行时访问该集合属性。本示例使用“值”对话框将若干项目添加到集合属性中，然后使用 **Collection** 和 **Iterator** API 在运行时显示这些项目。

在运行时访问集合项目：

1. 打开先前创建的 `MyShelf.fla` 文件。

请参阅第 165 页的“简单集合示例”。

本示例是在 **MyShelf** 组件和 **CompactDisc** 集合的基础上构建的。

2. 打开“库”面板，将组件拖动到舞台上，然后为其指定实例名称。

本示例使用的实例名称为 **myShelf**。

3. 选择该组件，打开“组件”检查器，然后显示“参数”选项卡。单击包含集合属性的行，然后单击该行右侧的放大镜。此时 **Flash** 会显示“值”对话框。

4. 使用“值”对话框为集合属性输入值。

5. 在舞台上选择该组件，打开“动作”面板并输入以下代码（必须附加到该组件）：

```
onClipEvent (mouseDown) {
    import mx.utils.Collection;
    import mx.utils.Iterator;
    var myColl:mx.utils.Collection;
    myColl = _parent.myShelf.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;
    trace("Title: " + title + " Artist: " + artist);
}
}

```

要访问集合，请使用语法 `componentName.collectionVariable`；要访问迭代器并遍历集合项目，请使用 `componentName.collectionVariable.getIterator()`。

6. 选择“控制” > “测试影片”，然后单击工具架，在“输出”面板中查看集合数据。

将具有集合的组件导出为 SWC 文件

分发具有集合的组件时，SWC 文件必须包含以下依赖文件：

- Collection 接口
- 集合实现类
- 集合项目类
- Iterator 接口

在这些文件中，代码通常会使用 **Collection** 和 **Iterator** 接口，将其标记为依赖类。Flash 会自动在 SWC 文件和输出 SWF 文件中包含依赖文件。

但是，集合实现类 (`mx.utils.CollectionImpl`) 和特定于组件的集合项目类不会自动包含在 SWC 文件中。

要在 SWC 文件中包含集合实现类和集合项目类，请在组件的 **ActionScript** 文件中定义私有变量，如下例所示：

```

// 集合项目类
private var collItem:CompactDisc;
// 集合实现类
private var coll:mx.utils.CollectionImpl;

```

有关 SWC 文件的更多信息，请参阅第 157 页的“了解 SWC 文件”。

使用具有集合属性的组件

使用包含集合属性的组件时，通常要使用“值”对话框在集合中建立项目。

使用包含集合属性的组件：

1. 向舞台添加组件。
2. 使用“属性”检查器为组件实例命名。
3. 打开“组件”检查器，然后显示“参数”选项卡。
4. 单击包含集合属性的行，然后单击该行右侧的放大镜。

此时 Flash 会显示“值”对话框。

5. 单击“增加” (+) 按钮，然后定义一个集合项目。
6. 单击“增加” (+)、“删除” (-) 和箭头按钮可以添加、修改、移动和删除集合项目。
7. 单击“确定”。

有关以编程方式访问集合的信息，请参阅第 168 页的[“使用 Flash 应用程序在运行时访问集合项目”](#)。

索引

数字

9 切片不受支持 16

英文

ActionScript 类文件 123

class 关键字 126

className 变量 126

Collection 标记 134

components
 creating subobjects 139

ComponentTask 标记
 JavaScript (JSFL) 135

createClassObject() method 139

CSSStyleDeclaration 73

DataGrid 组件
 绑定到 DataSet (教程) 26
 添加列 (教程) 28

DataSet 组件, 绑定到 XMLConnector 和 DataGrid
 (教程) 26

defaultPushButton 属性 51

Delegate 类 (教程) 63

DepthManager 类, 概述 52

Dial 组件 107, 148

draw() 方法, 定义 141

Event 元数据标记 132

Flash JavaScript (JSFL), ComponentTask 标记 135

Flash MX 版本和可用组件 10

Flash Player
 和组件 15
 支持 54

FlashType 不受支持 16

FocusManager 类, 创建焦点导航 51

getter/setter 方法 127

handleEvent 回调函数 59

init() 方法, 定义 138

invalidate() 方法 142

Label 组件
 教程 35

MovieClip 类, 扩展 118

on() 事件处理函数 67

ScrollPane 组件
 教程 35

size() 方法, 定义 142

StandardComponents 库 120

subobjects, creating 139

superclass 关键字 126

SWC 文件
 测试 159
 导出 158
 导出集合属性 169
 导入 159
 关于 17
 和编译剪辑 17
 文件格式 157

symbolName 变量 126

symbolOwner 变量 126

Tab 键切换 51

TextInput 组件 (教程) 35

UIComponent 类
 概述 117
 和组件继承性 15

UIObject 类
 关于 117

Web service, 连接 (教程) 25

WebService 类 (教程) 25

XMLConnector 组件
 绑定到 DataSet 组件 (教程) 26
 加载外部 XML 文件 (教程) 28
 指定架构 (教程) 26

A

安装组件 10

B

“绑定”选项卡, 范例应用程序中 (教程) 27
包 15

本文档的目标读者 6

编译剪辑

 关于 17

 在“库”面板中 47

变量, 定义 127

标记。参阅元数据

C

参见 DataGrid 组件 28

参数。参见组件参数

测试 SWC 文件 159

处理函数 55

D

代码提示, 触发 50

导出组件 157

导入语句 125

第 1 版组件, 升级 54

第 2 版组件

 和 Flash Player 15

 继承 15

 优点 13

F

发送程序 (事件广播器) 56

发送事件 143

范例主题 91

辅助功能 18

父类, 选择 116

G

光晕主题 91

广播器 56

J

集合属性

 导出组件 169

 定义 164

 定义类 167

 使用 170

 示例 165

 以编程方式访问 167

集合项目 167

继承, 在第 2 版组件中 15

加载, 组件 54

K

可检查参数 130

库

 StandardComponents 120

 编译剪辑 47

 “库”面板 47

扩展类 117

L

类

 UIComponent 117

 UIObject 117

 创建。参阅创建组件

 创建引用 (教程) 23

 导入 125

 定义 126

 和组件继承性 15

 扩展 117

 选择父类 116

类文件

 关于 124

 示例 123, 165

类样式表 69

列

 添加 (教程) 28

P

屏幕阅读程序, 辅助功能 18

Q

全局样式声明 73

S

- 删除组件 50
- 升级第 1 版组件 54
- 实例
 - 设置样式 71
 - 样式声明 69
- 实时预览功能 52
- 事件
 - 另请参见各个组件名称 55
 - 处理函数 55
 - 发送 143
 - 公共 143
 - 关于 55
 - 事件对象 66
 - 委托范围 63
 - 元数据 132
- 事件侦听器。参见侦听器
- “属性”检查器 48
- 数据绑定，使用 XML 文件（教程） 26
- 数据类型，实例的设置（教程） 24
- 数据网格。

T

- 提示文本，添加 160
- 图标，用于组件 160

W

- 外观
 - 另请参见各个组件名称
 - 编辑 83
 - 创建变量 144
 - 链接标识符 81
 - 应用到子组件 86
 - 应用于组件 85
- 外观的链接标识符 81
- 外观属性
 - 设置 81
 - 在原型中更改 89
- 外观组件 81
- 网格。参见 **DataGrid** 组件
- 位图缓存不受支持 16
- 文本，自定义 69
- 文档
 - 概述 6
 - 术语指南 7
- 文档中的术语 7

Y

- 颜色
 - 设置样式属性 77
 - 自定义 69
- 样式
 - 另请参见各个组件名称
 - 创建组件 145
 - 关于 69
 - 确定优先顺序 77
 - 设置 69
 - 设置全局 73
 - 设置自定义 73
 - 使用（教程） 24
 - 在实例上设置 71
- 样式表
 - 类 69
 - 自定义 69
- 样式声明
 - 默认类 75
 - 全局 73
 - 设置类 75
 - 自定义 73
- 样式属性，颜色 77
- 印刷惯例 7
- 影片剪辑
 - 创建 118
 - 定义为组件 122
- 预加载组件 53
- 预览组件 52
- 元数据
 - Collection** 标记 134
 - ComponentTask** 标记 135
 - Event** 标记 132
 - Inspectable** 标记 130
 - 标记，列表 129
 - 关于 128
- 原型 89

Z

- 侦听器
 - 对象 56
 - 范围 61
 - 关于 56
 - 函数 59
 - 使用组件（教程） 28
 - 用于组件实例（教程） 32
- “值”对话框 163
- 主题

- 创建 93
- 关于 91
- 应用 96
- 资源, 附加 Macromedia 7
- 子类, 用于替换外观 89
- 子组件, 应用外观 86
- 自定义
 - 文本 69
- 自定义颜色和文本, 使用样式表 69
- 组件 123
 - ActionScript 类 123
 - className 变量 126
 - Flash MX 各版本中提供 10
 - Flash Player 支持 15
 - getter/setter 方法 127
 - symbolOwner 变量 126
 - 另请参见各个组件名称
 - 安装 10
 - 编辑影片剪辑 120
 - 测试 SWC 文件 159
 - 创建影片剪辑 118
 - 导出 SWC 文件 158
 - 导出和分发 157
 - 导入 SWC 文件 159
 - 定义 draw() 方法 141
 - 定义 init() 方法 138
 - 定义 size() 方法 142
 - 定义变量 127
 - 定义参数 136
 - 发送事件 143
 - 公共事件 143
 - 构建组件示例 107, 148
 - 继承 15
 - 加载 54
 - 将外观注册到样式 145
 - 将组件导出为 SWC 158
 - 结构 106
 - 具有集合的类文件的示例 165
 - 开发检查列表 161
 - 扩展类 117
 - 类别, 已介绍 14
 - 类概述 124
 - 类文件示例 123
 - 删除 50
 - 事件 55
 - 体系结构 15
 - 添加到 Flash 文档 44
 - 添加工具提示 160
 - 添加图标 160
 - 无效, 关于 142

- 选择父类 116
- 选择元件名称 126
- 样式 145
- 预加载 53
- 预览 52
- 元数据, ComponentTask 标记 135
- 元数据标记 128
- 源文件 106
- 在应用程序中使用 (教程) 19
- 在运行时添加 46
- 指定外观 144
- “组件”检查器
 - “绑定”选项卡 27
 - 设置参数 48
- “组件”面板 44
- 组件参数
 - inspectable 130
 - 另请参见各个组件名称
 - 查看 48
 - 定义 136
 - 关于 48
 - 设置 48
- 组件的系统要求 6
- 组件开发的最佳做法 161
- 组件类文件。参阅类成员