MX

macromedia®
FLASH™MX
2004

使用组件

鸣谢

主管：Erick Vera

项目管理：Stephanie Gowin、 Barbara Nelson

撰稿：Jody Bleyle、Mary Burger、Kim Diezel、Stephanie Gowin、Dan Harris、Barbara Herbert、Barbara Nelson、Shirley Ong、 Tim Statler

主编：Rosana Francescato

编辑：Mary Ferguson、 Mary Kraemer、 Noreen Maher、 Antonio Padial、 Lisa Stanziano、 Anne Szabla

生产管理：Patrice O'Neill

媒体设计和制作：Adam Barnett、 Christopher Basmajian、 Aaron Begley、 John Francis、 Jeff Harmon

本地化：Tim Hussey、 Seungmin Lee、Masayo Noda、 Simone Pux、 Yuko Yagi、 Jerry Wang

第一版：2003 年 8 月

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

目 录

引 言
组件入门

Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 是专业的标准创作工具，利用它们制作出的 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。一个组件就是一段影片剪辑，其中所带的参数由您在 Macromedia Flash 中创作时进行设置，其中所带的动作脚本 API 供您在运行时自定义组件。组件旨在让开发人员重用和共享代码，封装复杂功能，使设计人员在没有 “动作脚本” 时也能使用和自定义这些功能。

组件基于 Macromedia Component Architecture 的第 2 版 (V2)，因此，您可以方便而快速地构建具有一致的外观和行为的功能强大的应用程序。本手册介绍如何使用 v2 组件构建应用程序，以及每个组件的应用程序编程接口 (API)。它包括使用 Flash MX 2004 或 Flash MX Professional 2004 V2 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，下载其他开发人员创建的组件，还可以创建自己的组件。

目标读者

本书的目标读者是要构建 Flash MX 2004 或 Flash MX Professional 2004 应用程序并希望使用组件加快开发速度的开发人员。对于在 Macromedia Flash 中开发应用程序、编写动作脚本，以及 Macromedia Flash Player，您都应该已经比较熟悉了。

本书假定您已安装了 Flash MX 2004 或 Flash MX Professional 2004，并且知道如何使用它。在使用这些组件之前，您应该先学习完课程 “使用组件创建用户界面”（选择 “帮助” > “如何” > “快速任务” > “使用组件创建用户界面”）。

如果您希望尽可能少编写动作脚本，则可将组件拖到文档中，然后在属性检查器或 “组件检查器” 面板中设置这些组件的参数，并在 “动作” 面板中将 `on()` 处理函数直接附加到某个组件，以处理组件事件。

如果您希望创建更稳定的应用程序，则可以动态创建组件，使用其 API 在运行时设置属性和调用方法，然后使用侦听器事件模型来处理事件。

有关详细信息，请参阅第 15 页的第 2 章 “使用组件”。

系统要求

除 Flash MX 2004 或 Flash MX Professional 2004 外，Macromedia 组件没有任何其他系统要求。

安装组件

首次启动 Flash MX 2004 或 Flash MX Professional 2004 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 MX 2004 包括以下组件：

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

Flash MX Professional 2004 包括 Flash MX 2004 组件及以下附加组件和类：

- Accordion 组件
- Alert 组件
- DataBinding 包
- DateField 组件
- DataGrid 组件
- DataHolder 组件
- DataSet 组件
- DateChooser 组件
- Form 类
- MediaController 组件
- MediaDisplay 组件
- MediaPlayback 组件
- Menu 组件
- RDBMSResolver 组件
- Screen 类
- Tree 组件

- WebServiceConnector 组件
- XMLConnector 组件
- XUpdateResolver 组件

检查安装的 Flash MX 2004 或 Flash MX Professional 2004 组件：

1 启动 Flash。

2 如果“组件”面板尚未打开，请选择“窗口”>“开发面板”>“组件”，打开“组件”面板。

3 选择“用户界面组件”，展开组件树并查看已安装的组件。

您也可以从 Macromedia Exchange_cn 下载组件。要安装从 Exchange 下载的组件，请下载并安装 Macromedia Extension Manager。

任何组件，无论是 SWC 文件还是 FLA 文件（请参阅第 13 页的“关于编译剪辑和 SWC 文件”），都会在 Flash 的“组件”面板中出现。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：
   - HD/Applications/Macromedia Flash MX 2004/First Run/Components (Macintosh)
   - \Program Files\Macromedia\Flash MX 2004\<language>\First Run\Components (Windows)

3 打开 Flash。

4 如果“组件”面板尚未打开，请选择“窗口”>“开发面板”>“组件”，以在“组件”面板中查看组件。

关于本文档

本文档详细说明如何使用组件开发 Flash 应用程序。它假定读者已具备 Macromedia Flash 和“动作脚本”的一般知识。有关 Flash 及相关产品的说明，另备有专门文档。

- 有关 Macromedia Flash 的信息，请参阅使用 Flash、动作脚本参考指南 和 “动作脚本字典帮助”。
- 有关使用 Flash 应用程序访问 Web 服务的信息，请参阅 Using Flash Remoting（使用 Flash Remoting）。

印刷惯例

本书使用以下印刷惯例：

- *斜体字体* 表示应被替换的值（例如，文件夹路径中的文本）。
- 代码字体 表示动作脚本代码。
- *斜体代码字体* 表示动作脚本参数。
- **粗体字体**表示完全按原样采用的条目。

注意：粗体字体与用于平行式标题的字体不同。平行式标题的字体用作项目符号的备用项。

本手册中使用的术语

本书中使用以下术语：

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。

其他资源

有关 Flash 的最新信息，以及专家的建议、高级主题、范例、提示和其他更新，请访问定期更新的 Macromedia DevNet Web 站点。请经常访问该 Web 站点，查看有关 Flash 的最新消息以及如何充分利用该程序的指导。

有关 TechNote、文档更新以及指向 “Flash 社区” 中其他资源的链接，请访问 Macromedia Flash 技术支持中心，其地址为：www.macromedia.com/go/flash_support_cn。

有关动作脚本术语、语法和用法的详细信息，请参阅 《动作脚本参考指南》和 “动作脚本字典帮助”。

第 1 章
关于组件

组件是带有参数的影片剪辑，这些参数使您可以修改组件的外观和行为。组件可以提供创建者能想到的任何功能。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，它用于控制应用程序中接收焦点的对象）。

任何人都可以使用组件构建复杂的 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 应用程序，即使他们对动作脚本并没有深入的了解。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行为来满足自己的设计需求。

组件基于 Macromedia Component Architecture 的第 2 版 (V2)，因此，您可以方便而快速地构建具有一致的外观和行为的功能强大的应用程序。第 2 版的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、实施辅助功能，等等。

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动作脚本方法、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他选项。

Flash MX 2004 和 Flash MX Professional 2004 包括许多新的 Flash 组件和若干新版本的 Flash MX 组件。要查看 Flash MX 2004 和 Flash MX Professional 2004 中所含组件的完整列表，请参阅第 8 页的“安装组件”。您还可以从 Macromedia Exchange_cn 下载由 Flash 社区成员构建的组件。

第 2 版组件的优点

使用组件，就可以做到编码与设计的分离，而且，您还可以重复利用您自己创建的组件中的代码，或者通过下载和安装其他开发人员创建的组件来重复利用代码。

使用组件，代码编写者可以创建设计人员能够在应用程序中使用的功能。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”面板中更改参数。

Flash 社区的成员可以使用 Macromedia Exchange_cn 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 版组件结构的组件共享样式、事件处理、外观、焦点管理和深度管理等核心功能。在将第一个第 2 版组件添加到应用程序中时，文档大小约增加 25K，用以提供此核心功能。如果再添加其他组件，系统会为这些组件重用相同的 25K，因此，增加的文档大小比预期的要小。有关将第 1 版组件升级到第 2 版组件的信息，请参阅第 24 页的 “将第 1 版组件升级为第 2 版结构”。

组件类别

Flash MX 2004 和 Flash MX Professional 2004 中包含的组件分为四类：用户界面 (UI) 组件、媒体组件、数据组件和管理器。使用 UI 控件，用户可以与应用程序进行交互操作；例如，RadioButton、 CheckBox 和 TextInput 组件都是 UI 控件。利用媒体组件，您可以将媒体流入到应用程序中； MediaPlayback 组件就是一个媒体组件。利用数据组件可以加载和处理数据源的信息； WebServiceConnector 和 XMLConnector 组件都是数据组件。管理器是不可见的组件，使用这些组件，您可以在应用程序中管理诸如焦点或深度之类的功能； FocusManager、DepthManager、 PopUpManager 和 StyleManager 都是 Flash MX 2004 和 Flash MX Professional 2004 包含的管理器组件。有关每个类别的完整列表，请参阅第 41 页的第 4 章 “组件字典”。

组件结构

您可以使用 “属性检查器” 或 “组件检查器” 面板来更改组件参数，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 MX 2004 和 Flash MX Professional 2004 组件是使用 Macromedia Component Architecture 第 2 版 (V2) 构建的。Flash Player 6 和 Flash Player 7 支持第 2 版组件。这些组件不一定总是与使用第 1 版 (v1) 结构构建的组件（在 Flash MX 2004 发布之前发布的所有组件）兼容。而且， Flash Player 7 不支持第 1 版组件。有关详细信息，请参阅第 24 页的 “将第 1 版组件升级为第 2 版结构”。

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。编译剪辑具有内置的实时预览，无法对它们进行编辑，但您可以在属性检查器和 “组件检查器” 面板中更改它们的参数，就像更改任何其他组件的参数一样。有关详细信息，请参阅第 13 页的 “关于编译剪辑和 SWC 文件”。

第 2 版组件是在动作脚本 2 中编写的。每个组件都是一个类，每个类都属于一个动作脚本包。例如，单选按钮组件是 RadioButton 类的实例，该类的包名称是 `mx.controls`。有关包的详细信息，请参阅 《动作脚本参考指南》帮助中的 “使用包”。

用 Macromedia Component Architecture 的第 2 版构建的所有组件都是 UIObject 和 UIComponent 类的子类，并且从这些类继承了所有属性、方法和事件。许多组件也是其他组件的子类。在第 41 页的第 4 章 “组件字典”中，分别在每个组件的条目中指明了该组件的继承路径。

所有组件也使用相同的事件模型、基于 CSS 的样式和内置的外观机制。有关样式和外观的详细信息，请参阅第 25 页的第 3 章 “自定义组件”。有关事件处理的详细信息，请参阅第 15 页的第 2 章 “使用组件”。

第 2 版组件的新功能

“组件检查器”面板：在使用 Macromedia Flash 和 Macromedia Dreamweaver 进行创作时可以利用它来更改组件参数。（请参阅第 16 页的 “‘组件检查器”面板和属性检查器中的组件”。）

侦听器事件模型：函数的侦听器对象可以使用它来处理事件。（请参阅第 21 页的 “关于组件事件”。）

外观属性：使您可以只在需要时才加载状态。（请参阅第 33 页的 “关于设置组件外观”。）

基于 CSS 的样式：您可以使用它来创建具有一致外观和行为的应用程序。（请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。）

主题：您可以使用它将新外观拖到一组组件上。（请参阅第 32 页的 “关于主题”。）

光晕主题：为应用程序提供预先创建的、互动式而且非常灵活的用户界面。

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 23 页的 “创建自定义焦点导航”和第 24 页的 “管理文档中的组件深度”。）

基类 UIObject 和 UIComponent ：为所有组件提供核心功能。（请参阅 第 210 页的 “UIComponent 类”和第 218 页的 “UIObject 类”。）

打包为 SWC 文件：您可以使用它来编写便于分发和可隐藏的代码。请参阅 “创建组件”。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

内置数据绑定通过 “组件检查器”面板提供。有关此功能的更多信息，请按 “帮助更新”按钮。

便于扩展的类层次结构：使用动作脚本 2，使您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 “创建组件”和动作脚本参考指南。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

关于编译剪辑和 SWC 文件

编译剪辑用于预编译 Flash 文档中的复杂元件。例如，包含大量动作脚本代码且不经常更改的影片剪辑可以转换为编译剪辑。这样可使 “测试影片”和 “发布”的执行时间较短。

SWC 文件是用于保存和分发组件的文件类型。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。

有关 SWC 文件的详细信息，请参阅 “创建组件”。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

辅助功能和组件

对 Web 内容要具有辅助功能（即，让各种残疾人可以使用 Web 内容）的要求在不断增长。使用屏幕读取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。这种软件可以提供对屏幕内容的口语声音描述。

创作者在创建组件时，可以编写在组件与屏幕读取程序之间通讯的动作脚本。随后，当开发人员使用组件在 Flash 中构建应用程序时，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。要了解某个组件是否具备辅助功能，请检查第 41 页的第 4 章 “组件字典”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.`*`ComponentName`*`AccImpl.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。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

由 Macromedia 构建的大多数组件也能通过键盘来导航。第 41 页的第 4 章 “组件字典”中每个组件的条目指出您是否可以用键盘控制该组件。

第 2 章
使用组件

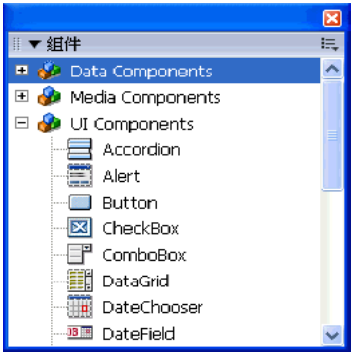
在 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中使用组件有多种方法。您可以使用 “组件” 面板来查看组件，并可以在创作过程中将组件添加到文档中。在将组件添加到文档中后，即可在属性检查器或 “组件检查器” 面板中查看组件属性。组件可以与其他组件通信，方法是侦听其他组件的事件并使用 “动作脚本” 处理这些事件。您还可以管理文档中的组件深度以及控制组件接收焦点的时间。

“组件” 面板

所有组件都存储在 “组件” 面板中。安装 Flash MX 2004 或 Flash MX Professional 2004 之后第一次启动它时，“组件” 面板中将会显示 Macromedia Flash 2004/First Run/Components (Macintosh) 或 Macromedia\Flash 2004\< 语言 >\First Run\Components (Windows) 文件夹中的组件。

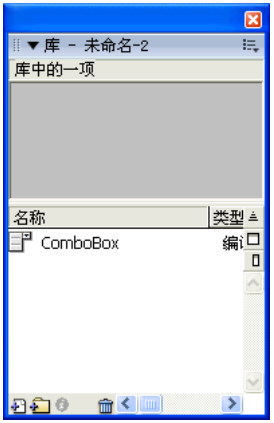
显示 “组件” 面板：

- 选择 “窗口” > “开发面板” > “组件”。



“库”面板中的组件

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



“库”面板中的 ComboBox 组件。

通过将组件图标从库拖到舞台上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 18 页的 “处理 SWC 文件和编译剪辑”。

“组件检查器”面板和属性检查器中的组件

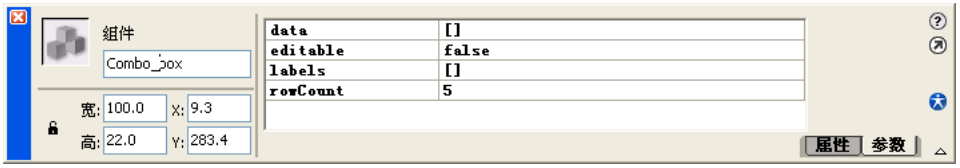
在将组件的一个实例添加到 Flash 文档后，可以使用属性检查器来设置和查看该实例的信息。通过从 “组件”面板将组件拖到舞台上创建该组件的一个实例，然后在属性检查器中给该实例命名，接着使用 “参数”选项卡上的字段来指定该实例的参数。您也可以通过使用 “组件检查器”面板来设置组件实例的参数。使用哪个面板来设置参数无关紧要，这只是您的个人喜好问题。有关设置参数的详细信息，请参阅第 20 页的 “设置组件参数”。

在属性检查器中查看组件实例的信息：

1 选择 “窗口” > “属性”。

2 在舞台上选择组件的一个实例。

3 要查看参数，请单击 “参数”选项卡。



要查看 “组件检查器”面板中组件实例的参数：

1 选择 “窗口” > “组件检查器”。

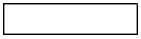
2 在舞台上选择组件的一个实例。

3 要查看参数，请单击 “参数”选项卡。

“实时预览”中的组件

“实时预览”功能在默认情况下处于启用状态，使用该功能，您可以按组件出现在所发布 Flash 内容中的实际情况在舞台上查看这些组件，包括其大概尺寸。实时预览反映了不同组件的不同参数。至于 “实时预览”中反映了哪些组件参数，有关这方面的信息，请参阅第 41 页的第 4 章 “组件字典”中的每个组件条目。“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。

启用 “实时预览”的 Button 组件。



禁用 “实时预览”的 Button 组件。

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。

有关详细信息，请参阅 “创建组件”。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

处理 SWC 文件和编译剪辑

Flash MX 2004 或 Flash MX Professional 2004 中包含的组件不是 FLA 文件，而是 SWC 文件。 SWC 是用于组件的 Macromedia 文件格式。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。 SWC 是已导出以进行分发的编译剪辑。

也可以在 Flash 中 “编译” 影片剪辑，并将其转换为编译剪辑元件。编译剪辑元件的行为方式与从中编译该元件的影片剪辑元件相似，但与常规影片剪辑元件相比，编译剪辑的显示和发布速度更快。编译剪辑不可编辑，但它们确实具有在属性检查器和 “组件检查器” 面板中出现的属性，并且它们包含实时预览。

Flash MX 2004 或 Flash MX Professional 2004 中包含的组件已转换为编译剪辑。如果创建组件，您可以选择将其导出为 SWC 以进行分发。有关详细信息，请参阅 “创建组件” 。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

编译影片剪辑元件：

- 选择库中的影片剪辑并用右键单击 (Windows) 或按住 Control 键单击 (Macintosh)，然后选择 “转换为编译剪辑”。

要导出 SWC：

- 选择库中的影片剪辑并用右键单击 (Windows) 或按住 Control 键单击 (Macintosh)，然后选择 “导出 SWC 文件”。

注意： Flash MX 2004 和 Flash MX Professional 2004 继续支持 FLA 组件。

向 Flash 文档中添加组件

在从 “组件” 面板将组件拖到舞台上时，就会将编译剪辑元件添加到 “库” 面板中。如果编译剪辑元件位于库中，您也可以通过使用 `UIObject.createClassObject()` 动作脚本方法，在运行时将该组件添加到文档中。

- 初级 Flash 用户可以使用 “组件” 面板将组件添加到 Flash 文档中，接着使用属性检查器或 “组件参数” 面板指定基本参数，然后使用 `on()` 事件处理函数来控制该组件。
- 中级 Flash 用户可以使用 “组件” 面板将组件添加到 Flash 文档中，然后使用属性检查器、“动作脚本” 方法，或两者的组合来指定参数。他们可以使用 `on()` 事件处理函数或事件侦听器来处理组件事件。
- 高级 Flash 程序员可以将 “组件” 面板和动作脚本结合在一起使用，以便添加组件并指定属性，或者选择在运行时完全使用动作脚本来实现组件实例。他们可以使用事件侦听器来控制组件。

如果在编辑了组件的外观后要添加该组件的另一个版本，或者添加共享同一外观的组件，则可以选择使用经过编辑的外观或使用一组新的默认外观来替换经过编辑的外观。如果替换了经过编辑的外观，那么所有使用这些外观的组件都会用默认的外观进行更新。有关如何编辑外观的详细信息，请参阅第 25 页的第 3 章 “自定义组件”。

使用 “组件”面板添加组件

在使用 “组件”面板将组件添加到文档后，通过将该组件从 “库”面板拖到舞台上，可以将该组件的其他实例添加到文档中。您可以在属性检查器的 “参数”选项卡或 “组件参数”面板中设置其他实例的属性。

使用 “组件”面板向 Flash 文档中添加组件：

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（所有安装的第 2 版组件都是 SWC）而且 您已经编辑了同一组件的另一个实例的外观，或者编辑了与您要添加的组件共享外观的组件的外观，则请执行下列操作：

- 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”。

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。

有关详细信息，请参阅第 20 页的 “设置组件参数”。

8 根据需要更改组件的大小。

有关调整特定组件类型大小的详细信息，请参阅第 41 页的第 4 章 “组件字典”中的各个组件条目。

9 通过执行以下一项或多项操作，根据需要更改组件的颜色和文本格式：

- 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 `UIObject.setStyle()`。
- 在分配给所有第 2 版组件的 _global 样式声明中编辑多个属性。
- 如果需要，可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 26 页的 “使用样式自定义组件的颜色和文本”。

10 根据需要，执行以下其中一项操作来自定义组件的外观：

- 应用主题（请参阅第 32 页的 “关于主题”）。
- 编辑组件的外观（请参阅第 33 页的 “关于设置组件外观”）。

使用动作脚本添加组件

要使用 “动作脚本” 向文档中添加组件，必须先将组件添加到库中。

您可以使用 “动作脚本” 方法为动态添加的组件设置其他参数。有关详细信息，请参阅第 41 页的第 4 章 “组件字典”。

注意：本节的说明假定您具备 “动作脚本” 的中级或高级知识。

使用 “动作脚本” 向 Flash 文档中添加组件：

1 将组件从 “组件” 面板拖到舞台上并将其删除。

这会将该组件添加到库中。

2 选择您想将该组件放置到的时间轴中的某一帧。

3 如果 “动作” 面板尚未打开，请将其打开。

4 调用 `createClassObject()` 方法，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。它将组件类名称、新实例的实例名称、深度和可选初始化对象作为它的参数。您可以在 *`className`* 参数中指定类包，如下所示：

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

或者您可以导入类包，如下所示：

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

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

5 使用该组件的 “动作脚本” 方法和属性在创作期间指定其他选项或覆盖设置的参数。

有关每个组件的可用 “动作脚本” 方法和属性的详细信息，请参阅第 41 页的第 4 章 “组件字典”中的各个组件条目。

关于组件标签大小以及组件的宽度和高度

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。如果组件实例比文本大，点击区域就会超出标签。

可以使用 “任意变形” 工具或 `setSize()` 方法来调整组件实例的大小。您可以从任何组件实例调用 `setSize()` 方法（请参阅 `UIObject.setSize()`）。如果使用 “动作脚本” 的 `_width` 和 `_height` 属性来调整组件的宽度和高度，则可以调整该组件的大小，但组件内容的布局依然保持不变。这可能会导致在影片回放时组件发生扭曲。有关调整组件大小的详细信息，请参阅第 41 页的第 4 章 “组件字典”中的各个组件条目。

设置组件参数

每个组件都带有参数，您可以通过设置这些参数来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” 面板中显示的属性或方法。最常用的属性和方法显示为创作参数；其他参数必须使用 “动作脚本” 来设置。在创作过程中可以设置的所有参数也可以使用 “动作脚本” 来设置。使用 “动作脚本” 设置的参数值将覆盖在创作过程中设置的任何值。

所有第 2 版组件都从 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，例如 `UIObject.setSize()`、`UIObject.setStyle()`、`UIObject.x` 和 `UIObject.y`。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`)，而 NumericStepper 组件具有 `nextValue` 和 `previousValue` 属性 (`NumericStepper.nextValue`、`NumericStepper.previousValue`)。

从 Flash 文档删除组件

要从 Flash 文档删除组件的实例，请通过删除编译剪辑图标来从库中删除组件。

从文档中删除组件：

1 在 “库”面板中，选择编译剪辑 (SWC) 元件。

2 单击 “库”面板底部的 “删除”按钮，或从 “库”面板选项菜单中选择 “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

在使用动作脚本 2 时，您可以精确键入基于内置类 （包括组件类）的变量。如果您这样做，动作脚本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：

```
var myCheckBox:CheckBox
myCheckBox.
```

您键入句点后， Flash 立即显示一列可供 CheckBox 组件使用的方法和属性，因为您是将该变量作为 CheckBox 键入的。有关数据键入的详细信息，请参阅 《动作脚本参考指南》帮助中的 “精确数据键入”。有关在出现代码提示时如何使用它们的信息，请参阅 《动作脚本参考指南》帮助中的 “使用代码提示”。

关于组件事件

所有组件都有事件，在用户与组件进行交互操作或组件中发生重要事情时，即会广播这些事件。要处理事件，请编写在触发事件时执行的 “动作脚本”代码。

您可以通过以下方式来处理组件事件：

- 使用 `on()` 组件事件处理函数。
- 使用事件侦听器。

使用组件事件处理函数

处理组件事件最简单的方式是使用 `on()` 组件事件处理函数。您可以将 `on()` 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。

附加到组件上的 `on()` 处理函数中使用的关键字 `this` 指的是组件实例。例如，以下代码附加到 Button 组件实例 `myButtonComponent`，它将 “_level0.myButtonComponent” 发送到 “输出”面板：

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

使用 on() 处理函数：

1 将 CheckBox 组件从 “组件”面板拖到舞台中。

2 选择该组件，然后选择 “窗口” > “动作”。

3 在 “动作”面板中，输入以下代码：

```
on(click){
  trace("CheckBox was clicked");
}
```

您可以在大括号 (`{}`) 之间输入所需的任何代码。

4 选择 “控制” > “测试影片”，然后在 “输出” 面板中单击要查看跟踪的复选框。

有关详细信息，请参阅第 41 页的第 4 章 “组件字典”中的每个事件条目。

使用组件事件侦听器

处理组件事件最强大的方式是使用侦听器。事件由组件进行广播，作为侦听器注册到事件广播器（组件实例）的任何对象都会收到该事件的通知。给侦听器分配一个处理事件的函数。您可以向一个组件实例注册多个侦听器。您也可以向多个组件实例注册一个侦听器。

要使用事件侦听器模型，请创建一个侦听器对象，该对象所带的属性应为事件的名称。给该属性分配一个回调函数。接着，对广播事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，并将事件的名称和侦听器对象的名称传递给它。调用 `UIEventDispatcher.addEventListener()` 方法称为 “注册” 或 “订阅” 侦听器，如下所示：

```
listenerObject.eventName = function(evtObj){
  // 此处是您的代码
};
componentInstance.addEventListener("eventName", listenerObject);
```

在上面的代码中，如果在回调函数中使用关键字 `this`，则该关键字的作用范围是 `listenerObject`。

`evtObj` 参数是在触发事件并将其传递到侦听器对象回调函数时自动生成的事件对象。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 216 页的 “UIEventDispatcher 类”。

有关组件所广播事件的信息，请参阅第 41 页的第 4 章 “组件字典”中的每个组件条目。

要注册事件侦听器，请执行以下操作：

1 将 Button 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，输入实例名称 **button**。

3 将 TextInput 组件从 “组件” 面板拖到舞台上。

4 在属性检查器中，输入实例名称 **myText**。

5 在时间轴中选择第 1 帧。

6 选择 “窗口” > “动作”。

7 在 “动作” 面板中，输入以下代码：

```
form = new Object();
form.click = function(evt){
  myText.text = evt.target;
}
button.addEventListener("click", form);
```

事件对象的目标属性是对广播事件的实例的引用。此代码在文本输入字段显示目标属性的值。

其他事件语法

除了使用侦听器对象外，您还可以将函数用作侦听器。如果侦听器不属于对象，它就是函数。例如，以下代码创建侦听器函数 `myHandler` 并将它注册到 `buttonInstance`：

```
function myHandler(eventObj){
  if (eventObj.type == "click"){
    // your code here
  }
}
buttonInstance.addEventListener("click", myHandler);
```

注意：在函数侦听器中， this 关键字是指 buttonInstance，不是指定义函数的时间轴。

您也可以使用支持 handleEvent 方法的侦听器对象。不论事件的名称是什么，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，因此，该语法不够灵活。例如，以下代码使用 if else 语句处理 click 和 enter 事件：

```
myObj.handleEvent = function(o){
  if (o.type == "click"){
    // your code here
  } else if (o.type == "enter"){
    // your code here
  }
}
target.addEventListener("click", myObj);
target2.addEventListener("enter", myObj);
```

另外还有一种事件语法样式，只有在创作组件并且知道某个特定对象是事件的唯一侦听器时，才可使用该事件语法样式。在这种情况下，您可以利用 V2 事件模型的一个特点，那就是它始终在事件名称加 “Handler” 的组件实例上调用方法。例如，如果要处理 click 事件，应编写以下代码：

```
componentInstance.clickHandler = function(o){
  // insert your code here
}
```

在上面的代码中，如果在回调函数中使用关键字 this，则该关键字的作用范围是 componentInstance。

有关详细信息，请参阅 “创建组件”。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

创建自定义焦点导航

当用户按 Tab 键在 Flash 应用程序中导航时，或在应用程序中单击时， FocusManager 类会确定接收焦点的组件。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 FocusManager。

如果 RadioButton 对象接收焦点， FocusManager 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 属性设为 true 的对象上。

每个模式 Window 组件都包含一个 FocusManager 的实例，以便该窗口上的控件形成它们自己的 Tab 键组，这样可以避免用户无意中按 Tab 键切换到其他窗口中的组件。

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 类查找 tabIndex 属性值高于当前 tabIndex 值的已启用对象。 FocusManager 类达到最高的 tabIndex 属性值后，会回到零。例如，在下面的范例中， comment 对象 （可能为 TextArea 组件）首先接收焦点，然后 okButton 对象接收焦点：

```
comment.tabIndex = 1;
okButton.tabIndex = 2;
```

要创建在用户按 Enter 键 (Windows) 或 Return 键 (Macintosh) 时接收焦点的按钮，请将 FocusManager.defaultPushButton 属性设为所需按钮的实例名称，如下所示：

```
FocusManager.defaultPushButton = okButton;
```

FocusManager 类会覆盖默认的 Flash Player 焦点矩形，并绘制带圆角的自定义焦点矩形。

管理文档中的组件深度

如果要在应用程序中将某个组件放在另一个对象的上方或下方，必须使用 DepthManager 类。DepthManager 应用程序编程接口 (API) 使您可以按照相应的 Z 顺序来放置用户界面 (UI) 组件 （例如，组合框在其他组件的前面下拉、插入点在所有内容的前面显示、对话窗口浮于内容上方，等等）。

DepthManager 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 DepthManager，请调用其方法 （请参阅第 87 页的 “DepthManager 类”）。

下面的代码将组件实例 `loader` 放在 `button` 组件下：

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

关于对组件使用预加载器

默认情况下组件设置为 “在第一帧导出”。这导致组件在应用程序的第一帧呈现之前加载。如果您需要为应用程序创建预加载器，则应为库中的任何编译剪辑元件取消选中 “在第一帧导出”。

注意：如果您使用 ProgressBar 组件来显示加载进度，则对于 ProgressBar，请保留 “在第一帧导出” 为选中状态。

将第 1 版组件升级为第 2 版结构

第 2 版组件在编写时符合多种 Web 标准（与事件、样式、getter/setter 策略等等有关的标准），它们与 Macromedia Flash MX 中以及在 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，而且是用动作脚本 2 编写的。因此，在应用程序中同时使用第 1 版和第 2 版组件可能会导致不可预测的结果。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 “创建组件”。您可能需要从 Flash 技术支持中心下载最新的 PDF 才能查看此信息。

包含第 1 版组件的 Flash 应用程序如果是为 Flash Player 6 或 Flash Player 6 版本 65 发布的，则在 Flash Player 6 和 Flash Player 7 中可以正常工作；如果您希望更新应用程序，以便在为 Flash Player 7 发布时也能工作，则必须转换代码以使用精确数据键入功能。有关详细信息，请参阅 《动作脚本字典》帮助中的 “使用动作脚本 2 创建类”。

第 3 章
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。在 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中完成这一任务有三种方法：

- 使用样式 API。
- 应用主题。
- 修改或替换组件的外观。

“样式 API”（应用程序编程接口）具有一些方法和属性，您可以使用这些方法和属性来更改组件的颜色和文本格式。

主题是组成组件外表的样式和外观的集合。

外观是用来显示组件的元件。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。

使用样式自定义组件的颜色和文本

每个组件实例都有样式属性和 `setStyle()` 和 `getStyle()`（请参阅 `UIObject.setStyle()` 和 `UIObject.getStyle()`）方法，您可以使用这些方法来修改和访问样式属性。您可以使用样式通过以下方法来自定义组件：

- 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。

- 使用 _global 样式声明，该样式声明为文档中的所有组件设置样式。

  如果要对整个文档应用一致的外观，可以在 _global 样式声明上创建样式。

- 创建自定义样式声明，并将它们应用到特定的组件实例。

  您可能还要让文档中成组的组件共享样式。为此，可创建应用到特定组件的自定义样式声明。

- 创建默认的类样式声明。

  您还可以定义默认的类样式声明，以使类的每个实例共享一个默认外观。

在使用“实时预览”功能查看舞台上的组件时，对样式属性所做的更改并不会显示出来。有关详细信息，请参阅第 17 页的“‘实时预览’中的组件”。

在组件实例上设置样式

您可以编写动作脚本代码，在任何组件实例上设置和获取样式属性。`UIObject.setStyle()` 和 `UIObject.getStyle()` 方法均可从任何组件直接调用。例如，以下代码在称为 `myButton` 的 Button 实例上设置文本颜色：

```
myButton.setStyle("color", 0xFF00FF");
```

虽然可以通过属性（例如，`myButton.Color = 0xFF00FF`）直接访问样式，但最好使用 `setStyle()` 和 `getStyle()` 方法，以便样式可以正常工作。有关详细信息，请参阅第 30 页的“设置样式属性值”。

注意：不应通过多次调用 UIObject.setStyle() 方法来设置多个属性。如果想更改多个属性，或更改多个组件实例的属性，应创建自定义样式格式。有关详细信息，请参阅第 27 页的“为特定组件设置样式”。

设置或更改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 **myComp**。

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，将实例更改为蓝色：

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

以下语法为组件实例指定属性和值：

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

5 选择“控制”>“测试影片”，查看所做的更改。

有关支持样式的列表，请参阅第 30 页的“支持的样式”。

设置全局样式

_global 样式声明分配给用第 2 版 Macromedia 组件结构（v2 组件）构建的所有 Flash 组件。_global 对象具有称为 `style` (`_global.style`) 的属性，该属性是 CSSStyleDeclaration 的实例。该 `style` 属性起着 _global 样式声明的作用。如果在 _global 样式声明中更改属性的值，该更改将应用到 Flash 文档中的所有组件。

一些样式是在组件类的 CSSStyleDeclaration 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 _global 样式声明，所以在 _global 样式声明上设置 `backgroundColor` 对 TextArea 和 TextInput 没有影响。有关详细信息，请参阅第 28 页的 “在同一个文档中使用全局、自定义和类样式”。

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 18 页的 “向 Flash 文档中添加组件”。

2 在时间轴中创建一个新层，并为该层指定一个名称。

3 在出现组件（或出现组件之前）的新层上选择帧。

4 打开 “动作” 面板。

5 使用以下语法更改 _global 样式声明上的任何属性。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");
```

有关样式列表，请参阅第 30 页的 “支持的样式”。

6 选择 “控制” > “测试影片”，查看所做的更改。

为特定组件设置样式

您可以创建自定义样式声明，从而为 Flash 文档中的特定组件指定一组独特的属性。首先，创建 CSSStyleDeclaration 对象的新实例，创建自定义样式名称并将其放置在 `_global.styles` 列表上 (`_global.styles.newStyle`)，然后为样式指定属性和值，再将该样式分配给实例。如果您在舞台上已经至少放置了一个组件实例，CSSStyleDeclaration 对象就能够被访问了。

您可以对自定义样式格式进行更改，方法和编辑 _global 样式声明中的属性的方法相同。只是使用 CSSStyleDeclaration 实例，而不是使用 _global 样式声明名称。有关 _global 样式声明的详细信息，请参阅第 27 页的 “设置全局样式”。

有关 CSSStyleDeclaration 对象属性的信息，请参阅第 30 页的 “支持的样式”。有关每个组件所支持样式的列表，请参阅第 41 页的第 4 章 “组件字典”中的各个组件条目。

为特定组件创建自定义样式声明：

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 18 页的 “向 Flash 文档中添加组件”。

此范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 9 步使用这些实例名称。

2 在时间轴中创建一个新层，并为该层指定一个名称。

3 在出现组件（或出现组件之前）的新层上选择帧。

4 在专家模式下打开 “动作” 面板。

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

```
var styleObj = new mx.styles.CSSStyleDeclaration;
```

6 设置样式声明的 styleName 属性，为该样式命名：

```
styleObj.styleName = "newStyle";
```

7 将该样式放置在全局样式表上：

```
_global.styles.newStyle = styleObj;
```

注意：您也可以使用以下语法来创建 CSSStyleDeclaration 对象并将它分配给新样式声明：

```
var styleObj = _global.styles.newStyle = new mx.styles.CSSStyleDeclaration();
```

8 使用以下语法指定要为 myStyle 样式声明定义的属性：

```
styleObj.fontFamily = "_sans";
styleObj.fontSize = 14;
styleObj.fontWeight = "bold";
styleObj.textDecoration = "underline";
styleObj.color = 0x336699;
styleObj.setStyle("themeColor", "haloBlue");
```

9 在同一个“脚本”窗格中，使用以下语法将两个特定组件的 styleName 属性设置为自定义样式声明：

```
a.setStyle("styleName", "newStyle");
b.setStyle("styleName", "newStyle");
```

您也可以使用 setStyle() 和 getStyle() 方法访问自定义样式声明上的样式。以下代码设置 newStyle 样式声明上的 backgroundColor 样式：

```
_global.styles.newStyle.setStyle("backgroundColor", "0xFFCCFF");
```

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。

以下代码创建 CheckBox 的类样式声明，并将复选框颜色设为蓝色：

```
var o = _global.styles.CheckBox = new mx.styles.CSSStyleDeclaration();
o.color = 0x0000FF;
```

您也可以使用 setStyle() 和 getStyle() 方法访问类样式声明上的样式。以下代码设置 RadioButton 样式声明上的颜色样式：

```
_global.styles.RadioButton.setStyle("color", "blue");
```

有关所支持样式的详细信息，请参阅第 30 页的 “支持的样式”。

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 文档可能同时具有 _global 样式声明、自定义样式声明、直接在组件实例上设置的样式属性，以及默认类样式声明。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。

首先，Flash 查找组件实例上的样式属性。如果实例上没有直接设置样式，Flash 将查看实例的 styleName 属性，确定是否向它分配了样式声明。

如果 styleName 属性未分配给样式声明，Flash 将查找默认类样式声明上的属性。如果没有类样式声明，并且属性没有继承它的值，则将检查 _global 样式声明。如果属性未在 _global 样式声明中定义，则该属性为 undefined。

如果没有类样式声明，但属性确实继承了它的值，Flash 将查找该实例父级上的属性。如果属性未在父实例上定义，Flash 将检查父实例的 styleName 属性；如果未定义该属性，Flash 将继续查看父实例，直到达到 _global 级别。如果属性未在 _global 样式声明中定义，则该属性为 undefined。

StyleManager 向 Flash 告知样式是否继承了它的值。有关详细信息，请参阅第 187 页的 “StyleManager 类”。

注意：不支持 CSS "inherit" 值。

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。所有颜色样式属性的名称都以 “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。所有其他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。您可以使用 StyleManager（请参阅第 187 页的 “StyleManager 类”）来添加新颜色名称。下表列出默认颜色名称：

颜色名称	值
black（黑色）	0x000000
white（白色）	0xFFFFFF
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

注意：如果不定义颜色名称，可能无法正确绘制组件。

您可以使用任何合法的动作脚本标识符来创建自己的颜色名称（例如 "WindowText" 或 "ButtonText"）。请使用 StyleManager 定义新颜色，如下所示：

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

大多数组件无法将对象处理为颜色样式属性值。然而，某些组件可以处理表示渐变或其他颜色组合的颜色对象。有关详细信息，请参阅第 41 页的第 4 章 “组件字典”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，如果要提供与 Microsoft Windows 类似的显示配置屏幕，可以定义 ButtonText 和 WindowText 等颜色名称，以及 Button、CheckBox 和 Window 等类样式声明。通过将样式声明中的颜色样式属性设置为 ButtonText 和 WindowText，并为用户提供可用于更改 ButtonText 和 WindowText 值的用户界面，由此即可提供与 Micosoft Windows、Mac OS 或任何操作系统相同的颜色方案。

设置样式属性值

请使用 `UIObject.setStyle()` 方法在组件实例、全局样式声明、自定义样式声明或类样式声明上设置样式属性。以下代码将单选按钮实例的 `color` 样式设为红色：

```
myRadioButton.setStyle("color", "red");
```

以下代码设置自定义样式声明 CheckBox 的 `color` 样式：

```
_global.styles.CheckBox.setStyle("color", "white");
```

`UIObject.setStyle()` 方法可以识别样式是否为继承样式，并在该实例的子实例样式更改时，向它们发出通知。它也会通知组件实例：必须重绘自身以反映新样式。因此，应该使用 `setStyle()` 来设置或更改样式。然而，作为创建样式声明时的一项优化功能，您可以直接在对象上设置属性。有关详细信息，请参阅第 27 页的 “设置全局样式”、第 27 页的 “为特定组件设置样式” 和第 28 页的 “为组件类设置样式”。

请使用 `UIObject.getStyle()` 方法从组件实例、全局样式声明、自定义样式声明或类样式声明检索样式。以下代码获取颜色属性的值并将它赋予变量 `o`：

```
var o = myRadioButton.getStyle("color");
```

以下代码获取 _global 样式声明中定义的样式属性的值：

```
var r = _global.style.getValue("marginRight");
```

如果未定义样式，`getStyle()` 可能返回 `undefined` 值。然而，`getStyle()` 了解样式属性的继承方式。因此，即使样式为属性，您也应使用 `UIObject.getStyle()` 来访问样式，这样就不必知道它是否为继承样式。

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

支持的样式

Flash MX 2004 和 Flash MX Professional 2004 带有两个主题：光晕 (HaloTheme.fla) 和 范例 (SampleTheme.fla)。每个主题支持一组不同的样式。“范例” 主题使用第 2 版样式机制的所有样式，旨在向您提供这些样式在文档中的范例。“光晕” 主题支持 “范例” 主题样式的一个子集。

“范例” 样式中的大多数第二版组件都支持以下样式属性。有关各组件支持哪些 “光晕” 样式的信息，请参阅第 41 页的第 4 章 “组件字典”。

如果输入值不是任何允许值，则将使用默认值。在重用 CSS 样式声明，而该样式声明使用值的 Macromedia 子集以外的其他值时，这一点很重要。

组件支持以下样式：

样式	描述
backgroundColor	组件的背景。这是唯一不继承样式值的颜色样式。默认值为透明。
borderColor	三维边框的黑色部分或二维边框的彩色部分。默认值为 0x000000（黑色）。
borderStyle	组件边框：可以是 “none”、“inset”、“outset” 或 “solid”。此样式不继承其值。默认值为 “solid”。
buttonColor	按钮的表面和三维边框的一部分。默认值为 0xEFEEEF （浅灰）。
color	组件标签的文本。默认值为 0x000000 （黑色）。
disabledColor	禁用的文本颜色。默认值为 0x848384 （深灰）。
fontFamily	文本的字体名称。默认值为 _sans。
fontSize	字体的磅值。默认值为 10。
fontStyle	字体样式：可以是 “normal” 或 “italic”。默认值为 “normal”。
fontWeight	字体粗细：可以是 “normal” 或 “bold”。默认值为 “normal”。
highlightColor	三维边框的一部分。默认值为 0xFFFFFF （白色）。
marginLeft	表示文本左边距的数字。默认值为 0。
marginRight	表示文本右边距的数字。默认值为 0。
scrollTrackColor	滚动条的滚动滑轨。默认值为 0xEFEEEF （浅灰）。
shadowColor	三维边框的一部分。默认值为 0x848384 （深灰）。
symbolBackgroundColor	复选框和单选按钮的背景色。默认值为 0xFFFFFF （白色）。
symbolBackgroundDisabledColor	复选框和单选按钮在禁用时的背景色。默认值为 0xEFEEEF（浅灰）。
symbolBackgroundPressedColor	复选框和单选按钮在按下时的背景色。默认值为 0xFFFFFF（白色）。
symbolColor	复选框的复选标记或单选按钮的点。默认值为 0x000000 （黑色）。
symbolDisabledColor	被禁用的复选标记或单选按钮点的颜色。默认值为 0x848384（深灰）。
textAlign	文本对齐方式：可以是 “left”、“right” 或 “center”。默认值为 “left”。
textDecoration	文本修饰：可以是 “none” 或 “underline”。默认值为 “none”。
textIndent	表示文本缩进的数字。默认值为 0。

关于主题

主题是样式和外观的集合。 Flash MX 2004 和 Flash MX Professional 2004 的默认主题称为“光晕” (HaloTheme.fla)。“光晕”主题旨在使您能够为用户提供极富表现力的互动式体验。Flash MX 2004 和 Flash MX Professional 2004 还带有一个称为“范例” (SampleTheme.fla) 的主题。使用“范例”主题，您可以试验可用于第 2 版组件的全套样式。（“光晕”主题只使用可用样式的一个子集。） 主题文件位于以下文件夹中：

- Windows - First Run\ComponentFLA
- Macintosh - First Run/ComponentFLA

您可以创建新主题并将它们应用到应用程序，以此来更改所有组件的外观和行为。例如，您可以创建二维主题和三维主题。

第 2 版组件使用外观（图形或影片剪辑元件）来显示它们的可视外观。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。例如， ScrollBar 经过编程后，可以将链接标识符为 `ScrollDownArrowDown` 的元件加载为 ScrollBar 的向下箭头在按下状态时的外观。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样式。必须通过编写动作脚本代码来创建全局样式声明和任何其他样式声明。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

将主题应用到文档

若要将新主题应用到文档，请打开某主题 FLA 作为外部库，并将主题文件夹从外部库拖到文档库中。以下步骤详细解释了该过程。

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” > “保存”，并为其选择诸如 **ThemeApply.fla** 的唯一名称。
3. 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。

   如果尚未创建新主题，则可以使用“范例”主题，该主题位于 Flash 2004/en/Configuration/SampleFLA 文件夹下。
4. 在主题的“库”面板中，选择“Flash UI 组件 2” > “主题” > “MMDefault”，然后将文档中所有组件的 Assets 文件夹拖到 ThemeApply.fla 库中。

   如果不确定文档中有哪些组件，可以将整个 Themes 文件夹拖到舞台上。库中 Themes 文件夹内的外观自动分配到文档中的组件。

   注意：舞台上的组件“实时预览”不会反映新主题。
5. 选择“控制” > “测试影片”，查看应用新主题的文档。

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。其他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）使用动作脚本绘制 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动作脚本代码用作模板，按需要调整外观。

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。

为副本指定唯一名称，例如 **MyTheme.fla**。

2 选择“文件”>“在 Flash 中打开 MyTheme.fla”。

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，在编辑元件模式下打开元件。

外观位于 Themes > MMDefault > 组件 Assets 文件夹下（在本例中为 Themes > MMDefault > 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，必须您可以注册点，以便使外观能够正确显示。所有被编辑元件的左上角必须位于 (0,0)。

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，返回到编辑文档模式。

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 执行前一节第 32 页的 “将主题应用到文档”中的步骤，将 MyTheme.fla 应用到文档。

关于设置组件外观

外观是组件用来显示其外表的元件。外观可以是图形元件，也可以是影片剪辑元件。大多数外观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动作脚本代码。

Macromedia 第 2 版组件是编译剪辑，您在库中看不到它们的资源。然而，Flash 中安装的 FLA 文件包含所有组件外观。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。这样，您就可以将主题拖到文档中的舞台上，以此来更改它的外观。有关主题的详细信息，请参阅第 32 页的 “关于主题”。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 的“库”面板中的 Themes 文件夹中。

每个组件由许多外观组成。例如，ScrollBar 的向下箭头由三个外观组成：ScrollDownArrowDisabled、ScrollDownArrowUp 和 ScrollDownArrowDown。一些组件彼此共享外观。使用滚动条的组件（包括 ComboBox、List、ScrollBar 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。每个组件外观都有外观属性，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 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"`，它是外观元件的“链接标识符”。您可以使用这些外观属性来编辑外观并将它们应用到组件。不必通过编辑组件的 .as 文件来更改其外观属性，您可以在文档中创建组件时，将外观属性值传递到组件的构造函数。

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：

- 若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 32 页的 “关于主题”）。

  注意：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。

- 若要为同一组件的多个实例使用不同的外观，可编辑现有的外观，然后设置外观属性（请参阅第 34 页的 “编辑组件外观”和第 35 页的 “将经过编辑的外观应用到组件”）。
- 若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 36 页的 “将经过编辑的外观应用到子组件”）。
- 若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 38 页的 “在原型中更改外观属性”）。

注意：上述方法按易用程度从上到下列出。

编辑组件外观

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。

要编辑外观，请执行以下操作：

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 > 组件 Assets 文件夹中（在本例中，为 Themes > MMDefault > RadioButton Assets > States）。

4 从 “库选项” 菜单选择 “重制”（或右键单击元件），然后为元件指定诸如 MyRadioTrueUp 的唯一名称。

5 在 “元件属性” 对话框中选择 “高级” 按钮，然后选择 “为动作脚本导出”。

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，在编辑元件模式下打开该外观。

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，必须您可以注册点，以便使外观能够正确显示。所有被编辑元件的左上角必须位于 (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，返回到编辑文档模式。

9 选择 “文件” > “保存”，但不关闭 MyTheme.fla。现在必须创建新文档，在新文档中将经过编辑的外观应用到组件。

有关详细信息，请参阅第 35 页的 “将经过编辑的外观应用到组件”、第 36 页的 “将经过编辑的外观应用到子组件”或第 38 页的 “在原型中更改外观属性”。有关如何应用新外观的信息，请参阅第 33 页的 “关于设置组件外观”。

注意：当使用 “实时预览” 功能在舞台上查看组件时，看不到对组件外观的更改。

将经过编辑的外观应用到组件

编辑完外观后，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。

要动态创建组件并应用经过编辑的外观，请执行以下操作：

1 选择 “文件” > “新建”，创建新的 Flash 文档。

2 选择 “文件” > “保存”，并为其指定诸如 **DynamicSkinning.fla** 的唯一名称。

3 将任意组件从 “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），然后删除它们。

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 和自定义的任何其他元件从 MyTheme.fla 拖到 DynamicSkinning.fla 的舞台上，然后删除它们。

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”。

要将组件手动添加到舞台并应用经过编辑的外观，请执行以下操作：

1 选择 “文件” > “新建”，创建新的 Flash 文档。

2 选择 “文件” > “保存”，并为其指定诸如 **ManualSkinning.fla** 的唯一名称。

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 和自定义的任何其他元件从 MyTheme.fla 拖到 ManualSkinning.fla 的舞台上，然后删除它们。

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”。

将经过编辑的外观应用到子组件

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。

如果组件中包含子组件，则第 41 页的第 4 章 “组件字典”的组件条目中会标明这些子组件。

若要将新外观应用到子组件，请执行下列操作：

1 执行第 34 页的 “编辑组件外观”中的步骤，但编辑滚动条外观。在此范例中，请编辑 ScrollDownArrowDown 外观并为它指定新名称 **MyScrollDownArrowDown**。

2 选择 “文件” > “新建”，创建新的 Flash 文档。

3 选择 “文件” > “保存”，并为其指定诸如 **SubcomponentProject.fla** 的唯一名称。

4 双击 “组件” 面板中的 List 组件，将它添加到舞台中，然后按 Backspace 键，将它从舞台中删除。

这样会将该组件添加到 “库” 面板，但不会在文档中显示该组件。

5 将 MyScrollDownArrowDown 和任何编辑过的其他元件从 MyTheme.fla 拖到 SubcomponentProject.fla 的舞台上，然后删除它们。

这样会将该组件添加到 “库” 面板，但不会在文档中显示该组件。

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 帧上输入下列代码，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 组件从库拖到舞台上。

- 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;
```

注意：您必须设置足够多的数据，以便使滚动条显示出来，或者将 `vScrollPolicy` 属性设置为 `true`。

7 选择 “控制” > “测试影片”。

为文档中的所有组件设置子组件外观还有另一种方法：在外观元件 `#initclip` 部分的该子组件 `prototype` 对象上设置外观属性。有关原型对象的详细信息，请参阅《动作脚本字典》帮助中的 `Function.prototype`。

要使用 #initclip 将经过编辑的外观应用到文档中的所有组件，请执行以下操作：

1 执行第 34 页的 “编辑组件外观”中的步骤，但编辑滚动条外观。在此范例中，请编辑 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，将其拖动到 SkinsInitExample.fla 的舞台上，然后删除它。

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 “选项” 菜单中选择 “链接”。

5 选中 “为动作脚本导出” 复选框。单击 “确定”。

自动选中 “在第一帧导出”。

6 双击库中的 MyScrollDownArrowDown，以编辑元件模式将其打开。

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” 面板拖到舞台上并将其删除。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"]});
```

注意：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`。

下面的范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 的外观，而不设置任何 TextArea 或 ScrollPane 滚动条的外观。

要使用 #initclip 将经过编辑的外观应用到文档中的特定组件，请执行以下操作：

1 执行第 34 页的 “编辑组件外观”中的步骤，但编辑滚动条外观。在此范例中，请编辑 ScrollDownArrowDown 外观并为它指定新名称 **MyScrollDownArrowDown**。

2 选择 “文件” > “新建”，然后创建 Flash 文档。

3 选择 “文件” > “保存”，并为文件指定诸如 **MyVScrollTest.fla** 的唯一名称。

4 将 MyScrollDownArrowDown 从主题库拖到 MyVScrollTest.fla 库。

5 选择 “插入” > “新元件”，并为其指定诸如 **MyScrollBars** 的唯一名称。

6 选中 “为动作脚本导出” 复选框。单击 “确定”。

自动选中 “在第一帧导出”。

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，以便使垂直滚动条显示出来。

10 选择 “文件” > “保存”。

11 选择 “文件” > “新建”，然后创建新的动作脚本文件。

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，选择 “发布设置” 按钮。

15 选择 “动作脚本版本设置” 按钮。

16 单击加号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。

17 选择 “控制” > “测试影片”。

在原型中更改外观属性

如果组件不直接支持外观变量，您可以将该组件分成子类并替换其外观。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。

如果组件中包含子组件，则第 41 页的第 4 章 “组件字典”的组件条目中会标明这些子组件。

要为子组件设置外观，请执行以下操作：

1 执行第 34 页的 “编辑组件外观”中的步骤，但编辑滚动条外观。在此范例中，请编辑 ScrollDownArrowDown 外观并为它指定新名称 **MyScrollDownArrowDown**。

2 选择 “文件” > “新建”，然后创建 Flash 文档。

3 选择 “文件” > “保存”，并为该文件指定诸如 **MyComboTest.fla** 的唯一名称。

4 将 MyScrollDownArrowDown 从主题库拖到 MyComboTest.fla 的舞台上并删除它。

此操作将元件添加到库中，但不会在舞台上显示它。

5 选择 “插入” > “新元件”，并为其指定诸如 **MyComboBox** 的唯一名称。

6 选中 “为动作脚本导出” 复选框并单击 “确定”。

自动选中 “在第一帧导出”。

7 在 MyComboBox 第 1 帧动作的 “动作” 面板上输入以下代码：

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

8 将 ComboBox 组件拖动到舞台上。

9 在属性检查器中，输入足够多的 Label 参数，以便使垂直滚动条显示出来。

10 选择 “文件” > “保存”。

11 选择 “文件” > “新建”，创建新的动作脚本文件 （仅限 Flash Professional）。

12 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;
  }
}
```

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

14 单击舞台上的空白区域，然后在属性检查器中，选择 “发布设置” 按钮。

15 选择 “动作脚本版本设置” 按钮。

16 单击加号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。

17 选择 “控制” > “测试影片”。

第 4 章
组件字典

本章介绍了各个组件及其应用程序编程接口 (API) 以供参考。

每个组件描述都包含了有关以下方面的信息：

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

组件是按字母顺序排列的。您也可以在下表中找到按类别排列的组件。

用户界面 (UI) 组件

组件	描述
Accordion 组件	一组垂直的互相重叠的视图，视图顶部有一些按钮，用户利用这些按钮可以在视图之间进行切换。
Alert 组件	一个窗口，用于给用户提出问题并提供按钮来捕获用户的响应。
Button 组件	一个大小可调整的按钮，可使用自定义图标来自定义。
CheckBox 组件	允许用户进行布尔型选择 （真或假）。
ComboBox 组件	允许用户从滚动的选择列表中选择一个选项。该组件可以在列表顶部有一个可编辑的文本字段，以允许用户搜索此列表。
DateChooser 组件	允许用户从日历中选择一个或多个日期。
DateField 组件	一个不可编辑的文本字段，并带有日历图标。当用户在组件边框内的任何位置单击时，就会显示一个 DateChooser 组件。
DataGrid 组件	允许用户显示和操作多列数据。
Label 组件	一个不可编辑的单行文本字段。
List 组件	允许用户从滚动列表中选择一个或多个选项。
Loader 组件	一个包含已载入的 SWF 或 JPEG 文件的区块。
Menu 组件	允许用户从列表中选择一个命令；它是一个标准的桌面应用程序菜单。
MenuBar 组件	水平的菜单栏。允许将菜单组合使用，以便您可以处理键盘和鼠标输入。
NumericStepper 组件	可单击的箭头，单击它们可以增加或减小数字的值。
ProgressBar 组件	显示一个过程 （通常是加载过程）的进度。
RadioButton 组件	允许用户在相互排斥的选项之间进行选择。
ScrollPane 组件	使用自动滚动条在有限的区域内显示影片、位图和 SWF 文件。
TextArea 组件	一个可随意编辑的多行文本字段。
TextInput 组件	一个可以随意编辑的单行文本输入字段。
Tree 组件	允许用户处理分级信息。
Window 组件	一个可拖动的窗口，带有标题栏、题注、边框和关闭按钮，用于向用户显示内容。

媒体组件

组件	描述
MediaController 组件	在应用程序中控制媒体流的播放。
MediaDisplay 组件	在应用程序中显示媒体流
MediaPlayback 组件	MediaDisplay 和 MediaController 组件的结合体。

数据组件

组件	描述
DataBinding 包	这些类实现 Flash 的运行时数据绑定功能。
DataHolder 组件	保存数据，并可用作组件之间的连接器。
DataProvider 组件	此组件是数据线性访问列表的模型。这一模型提供简单的用于广播其更改的数组操作功能。
DataSet 组件	一个构造块，用于创建数据驱动的应用程序。
RDBMSResolver 组件	用于将数据保存回任何支持的数据源。该解析程序组件对 Web 服务、JavaBean、 servlet 或 ASP 页可接收并分析的 XML 进行翻译。
WebServiceConnector 组件	提供对 Web 服务方法调用的无脚本访问。
XMLConnector 组件	使用 HTTP GET 和 POST 方法来读写 XML 文档。
XUpdateResolver 组件	用于将数据保存回任何支持的数据源。此解析程序组件将 DeltaPacket 翻译为 XUpdate。

管理器

组件	描述
DepthManager 类	管理对象的堆叠深度。
FocusManager 类	处理屏幕上各组件之间的 Tab 键导航。还处理当用户在应用程序中单击时的焦点变化。
PopUpManager 类	允许您创建和删除弹出式窗口。
StyleManager 类	允许您注册样式和管理继承的样式。

屏幕

组件	描述
Slide 类	用于在运行时操作幻灯片演示屏幕。
Form 类	用于在运行时操作窗体应用程序屏幕。

Accordion 组件

有关此功能的最新信息，请单击 “帮助”选项卡顶部的 “更新”按钮。

Alert 组件

有关此功能的最新信息，请单击 “帮助”选项卡顶部的 “更新”按钮。

Button 组件

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

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

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

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航”或第 93 页的 “FocusManager 类”。

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

在将 Button 组件添加到应用程序时，您可以使用 “辅助功能”面板，以使其可由屏幕读取器访问。首先，您必须添加下面的代码，为 Button 组件启用辅助功能：

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash”帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 Button 组件

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

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

注意：如果图标比按钮大，它将会延伸到按钮的边框外。

Button 参数

以下是您可以在属性检查器中或在 “组件检查器”面板中为每个 Button 组件实例设置的创作参数：

label 设置按钮上文本的值；默认值是 “Button”。

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

toggle 将按钮转变为切换开关。如果值为 true，则按钮在按下后保持按下状态，直到再次按下时才返回到弹起状态。如果值为 false，则按钮的行为就像一个普通按钮；默认值为 false。

selected 如果切换参数的值是 true，则该参数指定是按下 (true) 还是释放 (false) 按钮。默认值为 false。

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

您可以编写“动作脚本”，通过利用其属性、方法和事件来控制 Button 组件的这些选项以及其他选项。有关详细信息，请参阅 Button 类。

创建具有 Button 组件的应用程序

以下过程解释了如何在创作时将 Button 组件添加到应用程序。在本范例中，按钮是一个带有自定义图标的“帮助”按钮，当用户按下该按钮时，会打开一个“帮助”系统。

要创建具有 Button 组件的应用程序，请执行以下操作：

1. 将 Button 组件从“组件”面板拖到舞台上。
2. 在属性检查器中，输入实例名称 **helpBtn**。
3. 在属性检查器中，执行以下操作：
   - 为标签参数输入 **Help**。
   - 为图标参数输入 **HelpIcon**。

     要使用图标，必须将库中一个带有链接标识符的影片剪辑或图形元件用作图标参数。在本范例中，链接标识符是 HelpIcon。
   - 将切换属性设置为 true。
4. 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
clippyListener = new Object();
clippyListener.click = function (evt){
  clippyHelper.enabled = evt.target.selected;
}
helpBtn.addEventListener("click", clippyListener);
```

最后一行代码将 `click` 事件处理函数添加到 `helpBtn` 实例。该处理函数启用和禁用 `clippyHelper` 实例，此实例可能是某种“帮助”面板。

自定义 Button 组件

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

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

如果图标比按钮大，它将会延伸到按钮的边框外。

对 Button 组件使用样式

您可以设置样式属性来更改按钮实例的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

Button 组件支持下列光晕样式：

样式	描述
themeColor	组件的背景。这是唯一不继承样式值的颜色样式。可能的值包括 “haloGreen”、“haloBlue” 和 “haloOrange”。
color	组件标签的文本。
disabledColor	禁用的文本颜色。
fontFamily	文本的字体名称。
fontSize	字体的磅值。
fontStyle	字体样式：“常规” 或 “斜体”。
fontWeight	字体粗细：“常规” 或 “粗体”。

对 Button 组件使用外观

Button 组件使用 “动作脚本” 绘图 API 来绘制按钮的状态。若要在创作时设计 Button 组件的外观，请修改 ButtonSkin.as 文件 （该文件位于 First Run\Classes\mx\skins\halo 文件夹下）中的动作脚本代码。

如果您使用 `UIObject.createClassObject()` 方法 （在运行时）动态创建 Button 组件实例，就可以动态设计其外观。要在运行时设计组件的外观，请设置传递给 `createClassObject()` 方法的 `initObject` 参数的外观属性。这些外观属性设置用作按钮状态的元件的名称，元件可以带有图标，也可以没有图标。

如果您在创作过程中设置图标参数，或者在运行时设置 `icon` 动作脚本属性，则将同一链接标识符指定为三种图标状态：`falseUpIcon`、`falseDownIcon` 和 `trueUpIcon`。如果您希望为八种图标状态中的任一种指定唯一的图标 （例如，如果您希望在用户按下按钮时显示一个不同的图标），则必须设置传递到 `createClassObject()` 方法的 `initObject` 参数的属性。

下面的代码创建了一个名为 `initObject` 的对象，以用作 `initObject` 参数，并将外观属性设为新建元件的链接标识符。最后一行代码调用 `createClassObject()` 方法以创建一个新的 Button 类实例，该实例具有在 `initObject` 参数中传递的属性，如下所示：

```
var initObject = new Object();
initObject.falseUpIcon = "MyFalseUpIcon";
initObject.falseDownIcon = "MyFalseDownIcon";
initObject.trueUpIcon = "MyTrueUpIcon";
createClassObject(mx.controls.Button, "ButtonInstance", 0, initObject);
```

有关详细信息，请参阅第 33 页的 “关于设置组件外观” 和 `UIObject.createClassObject()`。

如果按钮被启用，当鼠标指针在它上方移过时，它会显示其悬停状态。在单击按钮时，按钮将接收输入焦点并显示其按下状态。当松开鼠标后，按钮又返回其悬停状态。如果在按下鼠标时指针移离按钮，按钮就会返回到其初始状态并保留输入焦点。如果切换参数设置为 true，则按钮的状态不会改变，直到鼠标在它上方松开。

如果按钮被禁用，不管用户进行什么交互操作，它都会显示其禁用状态。

Button 组件使用以下外观属性：

属性	描述
falseUpSkin	弹起状态。默认值为 RectBorder。
falseDownSkin	按下状态。默认值为 RectBorder。
falseOverSkin	悬停状态。默认值为 RectBorder。
falseDisabledSkin	禁用状态。默认值为 RectBorder。
trueUpSkin	切换状态。默认值为 RectBorder。
trueDownSkin	按下切换状态。默认值为 RectBorder。
trueOverSkin	悬停切换状态。默认值为 RectBorder。
trueDisabledSkin	禁用切换状态。默认值为 RectBorder。
falseUpIcon	图标弹起状态。默认值未定义。
falseDownIcon	图标按下状态。默认值未定义。
falseOverIcon	图标悬停状态。默认值未定义。
falseDisabledIcon	图标禁用状态。默认值未定义。
trueUpIcon	图标切换状态。默认值未定义。
trueOverIcon	图标悬停切换状态。默认值未定义。
trueDownIcon	图标按下切换装态。默认值未定义。
trueDisabledIcon	图标禁用切换状态。默认值未定义。

Button 类

继承 UIObject > UIComponent > SimpleButton > Button

动作脚本类命名空间 mx.controls.Button

Button 类的属性允许您在运行时给按钮添加图标、创建文本标签，或者指定该按钮在运行时的作用是普通按钮还是切换开关。

使用 “动作脚本”设置 Button 类的属性将会覆盖在属性检查器或 “组件检查器”面板中设置的同名参数。

Button 组件会用 FocusManager 覆盖默认的 Flash Player 焦点矩形，并画出一个带有圆角的自定义焦点矩形。有关详细信息，请参阅第 23 页的 “创建自定义焦点导航”。

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

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

注意：下面的代码返回未定义的：`trace(myButtonInstance.version);`。

Button 组件类与动作脚本内置 Button 对象不同。

Button 类的方法摘要

继承 UIObject 类和 UIComponent 类中的所有方法。

Button 类的属性摘要

方法	描述
SimpleButton.emphasized	指明按钮是否具有默认的普通按钮外观。
SimpleButton.emphasizedStyleDeclaration	当 emphasized 属性设置为 true 时的样式声明。
Button.icon	指定按钮实例的图标。
Button.label	指定在按钮上显示的文本。
Button.labelPlacement	指定标签文本相对于图标的方向。
Button.selected	当 toggle 属性为 true 时，指定按钮是处于按下状态 (true) 还是释放状态 (false)。
Button.toggle	指明按钮是否用作切换开关。

继承 UIObject 类和 UIComponent 类的所有属性。

Button 类的事件摘要

方法	描述
Button.click	在按钮实例上方按下鼠标或者按下空格键时进行广播。

继承 UIObject 类和 UIComponent 类的所有事件。

Button.click

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

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

用法 2：

```
listenerObject = new Object();
listenerObject.click = function(eventObject){
   ...
}
buttonInstance.addEventListener("click", listenerObject)
```

描述

事件；当在按钮上单击（释放）鼠标，或者当按钮具有焦点并按下空格键时，对所有已注册的侦听器进行广播。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 Button 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 Button 组件实例 `myButtonComponent`，它将 “_level0.myButtonComponent” 发送到 “输出” 面板：

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

请注意，这与附加到常规 Flash 按钮元件的 `on()` 处理函数内部使用的 `this` 的行为不同。在附加到按钮元件的 `on()` 处理函数中使用 `this` 时，它指的是包含该按钮的时间轴。例如，以下代码附加到按钮元件实例 `myButton`，它将 “_level0” 发送到 “输出” 面板：

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

注意：内置的动作脚本 Button 对象没有 `click` 事件；最接近的事件是 `release`。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*`buttonInstance`*) 调度一个事件（在本例中为 `click`），而该事件由附加到您创建的侦听器对象 (*`listenerObject`*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*`eventObject`*) 传递到侦听器对象方法。该事件对象有一组属性，这些属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()` 方法（请参阅 `UIEventDispatcher.addEventListener()`），以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

此范例是在时间轴上的某一帧上编写的，当单击名为 buttonInstance 的按钮时，它会向 “输出” 面板发送一条消息。第一行代码给该按钮设置标签。第二行代码指定该按钮用作切换开关。第三行代码创建一个名为 form 的侦听器对象。第四行代码为侦听器对象上的 click 事件定义一个函数。该函数内部有一个 trace 动作，它使用自动传递到该函数的事件对象（在本例中是 eventObj）来生成消息。事件对象的 target 属性是生成该事件的组件，在本例中是 buttonInstance。从事件对象的 target 属性中可以访问 Button.selected 属性。最后一行代码从 buttonInstance 调用 addEventListener() 方法，并将 click 事件和 form 侦听器对象作为参数传递给该方法，如下所示：

```
buttonInstance.label = "Click Test"
buttonInstance.toggle = true;
form = new Object();
form.click = function(eventObj){
  trace("The selected property has changed to " + eventObj.target.selected);
}
buttonInstance.addEventListener("click", form);
```

下列代码还会在单击 buttonInstance 时向 “输出” 面板发送一条消息。必须将 on() 处理函数直接附加到 buttonInstance 上，如下所示：

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

另请参见

UIEventDispatcher.addEventListener()

SimpleButton.emphasized

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

buttonInstance.emphasized

描述

属性；指明按钮是否处于强调状态，如果是，则为 true，否则为 false。强调状态相当于默认的普通按钮外观。一般来说，应使用 FocusManager.defaultPushButton 属性，而不是直接设置 emphasized 属性。默认值为 false。

emphasized 属性是 SimpleButton 类的静态属性。因此，只能直接从 SimpleButton 访问它，如下所示：

```
SimpleButton.emphasizedStyleDeclaration = "foo";
```

如果您没有使用 FocusManager.defaultPushButton，则您可能只想将按钮设置为强调状态，或者使用强调状态来更改文本颜色。在下面的范例中，将设置按钮实例 myButton 的 emphasized 属性：

```
_global.styles.foo = new CSSStyleDeclaration();
_global.styles.foo.color = 0xFF0000;
SimpleButton.emphasizedStyleDeclaration = "foo";
myButton.emphasized = true;
```

另请参见

SimpleButton.emphasizedStyleDeclaration

SimpleButton.emphasizedStyleDeclaration

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

buttonInstance.emphasizedStyleDeclataion

描述

属性；一个指明样式声明的字符串，该样式声明会在 emphasized 属性设置为 true 时格式化按钮。

另请参见

Window.titleStyleDeclaration, SimpleButton.emphasized

Button.icon

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

buttonInstance.icon

描述

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

要创建一个自定义图标，请创建一个影片剪辑或者图形元件。在编辑元件模式下，选择舞台上的该元件，并在属性检查器中的 X 和 Y 框内都输入 0。在“库”面板中，选择影片剪辑并从“选项”菜单中选择“链接”。选择“为动作脚本导出”，并在“标识符”文本框内输入一个标识符。

默认值是一个空字符串 ("")，指明没有图标。

使用 labelPlacement 属性来设置图标相对于按钮的位置。

范例

下列代码从 “库” 面板中将带有链接标识符 `happiness` 的影片剪辑作为图标分配给 Button 实例：

```
myButton.icon = "happiness"
```

另请参见

`Button.labelPlacement`

Button.label

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

`buttonInstance``.label`

描述

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

范例

下列代码将标签设为 “Remove from list”：

```
buttonInstance.label = "Remove from list";
```

另请参见

`Button.labelPlacement`

Button.labelPlacement

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

`buttonInstance``.labelPlacement`

描述

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

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

范例

下列代码将标签设在图标的左侧。第二行代码将 `labelPlacement` 属性的值发送到 “输出” 面板：

```
iconInstance.labelPlacement = "left";
trace(iconInstance.labelPlacement);
```

Button.selected

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

buttonInstance.selected

描述

属性；一个布尔值，指定按钮是 (true) 否 (false) 处于按下状态。要将 `selected` 属性设为 `true`，`toggle` 属性的值必须为 `true`。如果 `toggle` 属性为 `false`，则给 `selected` 属性赋予值 `true` 将不起作用。默认值为 `false`。

当用动作脚本更改 `selected` 属性的值时，不会触发 `click` 事件。当用户与该按钮交互时触发该事件。

范例

在下面的范例中，`toggle` 属性设为 `true` 并且 `selected` 属性也设为 `true`，这样就会使按钮处于按下状态。`trace` 动作将值 `true` 发送到 “输出” 面板：

```
ButtonInstance.toggle = true; // toggle 需要为 true，以便设置 selected 属性
ButtonInstance.selected = true; // 显示按钮的切换状态
trace(ButtonInstance.selected); // 跟踪 true
```

另请参见

Button.toggle

Button.toggle

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

buttonInstance.toggle

描述

属性；一个布尔值，指定将按钮用作切换开关 (true) 还是用作普通按钮 (false)。默认值是 `false`。当按下切换开关后，它将保持按下状态，直到再次单击它时为止。

范例

下列代码将 `toggle` 属性设为 `true`，这将会使 `myButton` 实例的行为与切换开关类似：

```
myButton.toggle = true;
```

CellRenderer 接口

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

CheckBox 组件

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

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

如果复选框被禁用，它会显示其禁用状态，而不管用户的交互操作。在禁用状态下，按钮不接收鼠标或键盘输入。

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

按键	描述
Shift + Tab	将焦点移到前一个元素。
空格键	选中或者取消选中组件并触发 `click` 事件。
Tab 键	将焦点移到下一个元素。

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航” 或第 93 页的 “FocusManager 类”。

每个 CheckBox 实例的实时预览反映在创作过程中对属性检查器或组件检查器面板中的参数所做的更改。

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

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 CheckBox 组件

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

CheckBox 参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 CheckBox 组件实例设置的创作参数：

label 设置复选框上文本的值；默认值是 defaultValue。

selected 将复选框的初始值设为选中 (true) 或取消选中 (false)。

labelPlacement 确定复选框上标签文本的方向。该参数可以是下列四个值之一：left、right、top 或 bottom，默认值是 right。有关详细信息，请参阅 `CheckBox.labelPlacement`。

您可以编写动作脚本，通过利用 CheckBox 组件的属性、方法和事件来控制该组件的这些选项以及其他选项。有关详细信息，请参阅 CheckBox 类。

创建具有 CheckBox 组件的应用程序

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

要创建具有 CheckBox 组件的应用程序，请执行以下操作：

1. 将两个 TextInput 组件从 “组件” 面板拖到舞台上。
2. 在属性检查器中，输入实例名称 `minimumAge` 和 `maximumAge`。
3. 将 CheckBox 组件从 “组件” 面板拖到舞台上。
4. 在属性检查器中，执行以下操作：
   - 输入 **restrictAge** 作为实例名称。
   - 输入 **Restrict Age** 作为标签参数。
5. 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

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

此代码创建一个 `click` 事件处理函数，该函数可启用和禁用已放到舞台上的 `minimumAge` 和 `maximumAge` 文本字段组件。有关 `click` 事件的详细信息，请参阅 `CheckBox.click`。有关 TextInput 组件的详细信息，请参阅第 200 页的 “TextInput 组件”。

自定义 CheckBox 组件

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

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

对 CheckBox 组件使用样式

您可以设置样式属性以更改 CheckBox 实例的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

CheckBox 组件支持下列光晕样式：

样式	描述
`themeColor`	组件的背景。这是唯一不继承样式值的颜色样式。可能的值包括 “haloGreen”、“haloBlue” 和 “haloOrange”。
`color`	组件标签的文本。
`disabledColor`	禁用的文本颜色。
`fontFamily`	文本的字体名称。
`fontSize`	字体的磅值。
`fontStyle`	字体样式：“常规” 或 “斜体”。
`fontWeight`	字体粗细：“常规” 或 “粗体”。
`textDecoration`	文本修饰：“无” 或 “下划线”。

对 CheckBox 组件使用外观

CheckBox 组件使用 “库” 面板中的元件来表示按钮的状态。若要在创作时设计 CheckBox 组件的外观，请修改 “库” 面板中的元件。 CheckBox 组件外观位于 HaloTheme.fla 文件或者 SampleTheme.fla 文件的库中的 Flash UI Components 2/Themes/MMDefault/CheckBox Assets/states 文件夹下。有关详细信息，请参阅第 33 页的 “关于设置组件外观”。

CheckBox 组件使用下列外观属性：

属性	描述
falseUpSkin	弹起状态。默认值为 RectBorder。
falseDownSkin	按下状态。默认值为 RectBorder。
falseOverSkin	悬停状态。默认值为 RectBorder。
falseDisabledSkin	禁用状态。默认值为 RectBorder。
trueUpSkin	切换状态。默认值为 RectBorder。
trueDownSkin	按下切换状态。默认值为 RectBorder。
trueOverSkin	悬停切换状态。默认值为 RectBorder。
trueDisabledSkin	禁用切换状态。默认值为 RectBorder。

CheckBox 类

继承 UIObject > UIComponent > SimpleButton > Button > CheckBox

动作脚本类命名空间 mx.controls.CheckBox

CheckBox 类的属性允许您在运行时创建一个文本标签，并将其放在复选框的左、右、上或下部。

使用“动作脚本”设置 CheckBox 类的属性将会覆盖在属性检查器或“组件检查器”面板中设置的同名参数。

CheckBox 组件用 FocusManager 覆盖默认的 Flash Player 焦点矩形，并画出一个带有圆角的自定义焦点矩形。有关详细信息，请参阅第 23 页的 “创建自定义焦点导航”。

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

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

注意：下面的代码返回未定义的：trace(myCheckBoxInstance.version);。

CheckBox 类的属性摘要

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

继承 UIObject 类和 UIComponent 类的所有属性。

CheckBox 类的方法摘要

继承 UIObject 类和 UIComponent 类中的所有方法。

CheckBox 类的事件摘要

事件	描述
CheckBox.click	在按钮实例上按下鼠标时触发。

继承 UIObject 类和 UIComponent 类的所有事件。

CheckBox.click

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

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

用法 2：

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

描述

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

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 CheckBox 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下面的代码附加到复选框 myCheckBox，它将 “_level0.myCheckBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*checkBoxInstance*) 调度一个事件（在本例中为 click），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象有一组属性，这些属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 addEventListener() 方法（请参阅 UIEventDispatcher.addEventListener()），以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

此范例是在时间轴上的某一帧上编写的，当单击名为 checkBoxInstance 的按钮时，它会向 “输出” 面板发送一条消息。代码的第一行创建一个名为 form 的侦听器对象。第二行代码为侦听器对象上的 click 事件定义一个函数。该函数内部有一个 trace 动作，它使用自动传递到该函数的事件对象 （在本例中是 eventObj）来生成消息。事件对象的 target 属性是生成该事件的组件，在本例中是 checkBoxInstance。从事件对象的 target 属性中可以访问 CheckBox.selected 属性。最后一行代码从 checkBoxInstance 调用 addEventListener() 方法，并将 click 事件和 form 侦听器对象作为参数传递给该方法，如下所示 ：

```
form = new Object();
form.click = function(eventObj){
  trace("The selected property has changed to " + eventObj.target.selected);
}
checkBoxInstance.addEventListener("click", form);
```

下面的代码也会在 checkBoxInstance 被单击时向 “输出” 面板发送消息。on() 处理函数必须直接附加到 checkBoxInstance，如下所示 ：

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

另请参见

UIEventDispatcher.addEventListener()

CheckBox.label

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

checkBoxInstance.label

描述

属性 ；显示复选框的文本标签。默认情况下，标签出现在复选框的右侧。对此属性的设置将会覆盖在剪辑参数面板中指定的标签参数。

范例

下面的代码设置显示在 CheckBox 组件旁的文本，并向 “输出” 面板发送该值 ：

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

另请参见

CheckBox.labelPlacement

CheckBox.labelPlacement

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

checkBoxInstance.labelReplacement

描述

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

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

- `"left"` 复选框被固定在边界区域的右上角。标签设置在复选框的左边。

- `"bottom"` 标签设置在复选框的下面。复选框和标签组水平、垂直居中。

- `"top"` 标签放置在复选框的上面。复选框和标签组水平、垂直居中。

在创作时可以使用 Transform 命令来更改组件的边界区域，或在运行时使用 `UIObject.setSize()` 属性来更改组件的边界区域。有关详细信息，请参阅第 56 页的 “自定义 CheckBox 组件”。

范例

下面的范例将标签的位置设置在复选框的左侧。

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

另请参见

`CheckBox.label`

CheckBox.selected

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

checkBoxInstance.selected

描述

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

范例

下面的范例选中 checkbox1 实例：

```
checkbox1.selected = true;
```

ComboBox 组件

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

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

如果单击文本框或按钮，ComboBox 组件就会获取焦点。当 ComboBox 组件拥有焦点并为可编辑时，所有键盘输入都会传递到文本框并根据 TextInput 组件（请参阅第 200 页的 “TextInput 组件”）的规则进行处理，但以下按键除外：

按键	描述
Control+ 下箭头键	打开下拉列表并给它设置焦点。
Shift + Tab	将焦点移到前一个对象。
Tab 键	将焦点移到下一个对象。

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

按键	描述
Control+ 下箭头键	打开下拉列表并给它设置焦点。
Control+ 上箭头键	关闭下拉列表 （如果下拉列表打开的话）。
向下箭头	选区会向下移动一项。
End 键	选区会移动到列表底端。
Esc 键	关闭下拉列表，并将焦点返回到组合框。
Enter 键	关闭下拉列表，并将焦点返回到组合框。
Home 键	选区会移动到列表顶端。
Page Down 键	选区会向下移动一页。
Page Up 键	选区会向上移动一页。
Shift + Tab	将焦点移到前一个对象。
Tab 键	将焦点移到下一个对象。

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

按键	描述
Control+ 上箭头键	如果下拉列表处于打开状态，焦点就会返回到文本框，并且下拉列表会关闭。
向下箭头	选区会向下移动一项。
End 键	插入点移动到文本框的末尾。
Enter 键	如果下拉列表处于打开状态，焦点就会返回到文本框，并且下拉列表会关闭。
Esc 键	如果下拉列表处于打开状态，焦点就会返回到文本框，并且下拉列表会关闭。
Home 键	插入点移动到文本框的开始位置。
Page Down 键	选区会向下移动一页。
Page Up 键	选区会向上移动一页。
Tab 键	将焦点移到下一个对象。
Shift-End	选中从插入点到末尾位置的文本。
Shift-Home	选择从插入点到开始位置的文本。
Shift-Tab	将焦点移到前一个对象。
向上箭头	选区会向上移动一项。

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

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航”或第 93 页的 “FocusManager 类”。

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

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

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 ComboBox 组件

在任何需要从列表中选择一项的表单或应用程序中，您都可以使用 ComboBox 组件。例如，您可以在客户地址表单中提供一个州 / 省的下拉列表。对于比较复杂的情况，您可以使用可编辑的组合框。例如，在一个驾驶方向应用程序中，您可以使用一个可编辑的组合框来让用户输入出发地址和目标地址。下拉列表可以包含用户以前输入过的地址。

ComboBox 参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 ComboBox 组件设置的创作参数 ：

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

labels 用一个文本值数组填充 ComboBox 组件。

data 将一个数据值与 ComboBox 组件中的每个项目相关联。该数据参数是一个数组。

rowCount 设置在不使用滚动条的情况下一次最多可以显示的项目数。默认值为 5。

您可以编写 “动作脚本”，通过利用 ComboBox 类的方法、属性和事件来设置 ComboBox 实例的其他选项。有关详细信息，请参阅 ComboBox 类。

创建具有 ComboBox 组件的应用程序

以下过程解释了如何在创作时将 ComboBox 组件添加到应用程序。在此范例中，组合框在其下拉列表呈现出一个从中选择城市的列表。

要创建具有 ComboBox 组件的应用程序，请执行以下操作：

1 将 ComboBox 组件从 “组件” 面板拖到舞台上。

2 选择 “变形” 工具，并在舞台上调整该组件的大小。

组合框只能在创作时在舞台上调整大小。通常，您只需改变组合框的宽度以适应其条目。

3 选择组合框，并在属性检查器中输入实例名称 **comboBox**。

4 在 “组件检查器” 面板或属性检查器中，执行以下操作 ：

- 输入 Minneapolis、Portland 和 Keene 作为标签参数。双击标签参数字段以打开 “值” 对话框。然后单击加号 （+）以添加项目。
- 输入 MN.swf、OR.swf 和 NH.swf 作为数据参数。

这些是假想的 SWF 文件。例如，当用户在组合框中选择了一个城市时，你就可以加载这些文件。

5 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

```
form = new Object();
form.change = function (evt){
  trace(evt.target.selectedItem.label);
}
comboBox.addEventListener("change", form);
```

最后一行代码将 `change` 事件处理函数添加到 `ComboBox` 实例。有关详细信息，请参阅 `ComboBox.change`。

自定义 ComboBox 组件

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

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

在可编辑的组合框中，只有按钮是点击区，文本框不是。对于静态组合框，按钮和文本框一起组成点击区。

对 ComboBox 组件使用样式

您可以设置样式属性来更改 ComboBox 组件的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

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

- 该按钮是一个 Button 实例，并使用它自己的样式。（请参阅第 46 页的 “对 Button 组件使用样式”。）
- 文本是一个 TextInput 实例，它使用自己的样式。（请参阅第 201 页的 “对 TextInput 组件使用样式”。）
- 下拉列表是一个 List 实例并使用它自己的样式。（请参阅第 107 页的 “对 List 组件使用样式”。）

ComboBox 组件使用下列光晕样式：

样式	描述
`themeColor`	组件的背景。这是唯一不继承样式值的颜色样式。可能的值包括 “haloGreen”、“haloBlue” 和 “haloOrange”。
`color`	组件标签的文本。
`disabledColor`	禁用的文本颜色。
`fontFamily`	文本的字体名称。
`fontSize`	字体的磅值。
`fontStyle`	字体样式：“常规” 或 “斜体”。
`fontWeight`	字体粗细：“常规” 或 “粗体”。
`textDecoration`	文本修饰：“无” 或 “下划线”。

样式	描述
openDuration	打开下拉列表的毫秒数。默认值为 250。
openEasing	对控制下拉列表动画的补间函数的引用。默认为正弦输入 / 输出。要了解更多的公式，请从 Robert Penner website （Robert Penner 的 Web 站点）下载列表。

对 ComboBox 组件使用外观

ComboBox 组件使用 “库” 面板中的元件来表示按钮的状态，ComboBox 具有向下箭头的外观变量。除此之外，它还使用滚动条外观和列表外观。要在创作过程中设计 ComboBox 组件的外观，请在 “库” 面板中修改元件并将组件重新导出为 SWC。CheckBox 组件外观位于 HaloTheme.fla 文件或者 SampleTheme.fla 文件的库中的 Flash UI Components 2/Themes/MMDefault/ComboBox Assets/states 文件夹下。有关详细信息，请参阅第 33 页的 “关于设置组件外观”。

ComboBox 组件使用下列外观属性：

属性	描述
ComboDownArrowDisabledName	向下箭头的禁用状态。默认值为 RectBorder。
ComboDownArrowDownName	向下箭头的按下状态。默认值为 RectBorder。
ComboDownArrowUpName	向下箭头的弹起状态。默认值为 RectBorder。
ComboDownArrowOverName	向下箭头的悬停状态。默认值为 RectBorder。

ComboBox 类

继承 UIObject > UIComponent > ComboBase > ComboBox

动作脚本类命名空间 mx.controls.ComboBox

ComboBox 组件结合了三个单独的子组件：Button、TextInput 和 List。从 ComboBox 组件中可以直接使用每个子组件的大多数 API，ComboBox 类的方法、属性和事件表中列出了这些 API。

所提供的组合框中的下拉列表是作为 Array 或者作为 DataProvider 对象。如果您使用 DataProvider 对象，列表会在运行时更改。通过切换到一个新的 Array 或 DataProvider 对象，可以动态改变 ComboBox 的数据源。

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

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

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

如果该项目是原始数据类型而不是字符串，则会转换为字符串。如果某个项目是一个对象，则其 label 属性必须是字符串，而 data 属性可以是任何动作脚本值。

您向其提供项目的 ComboBox 组件方法有两个参数：label 和 data，它们指的是上述属性。返回一个项目的方法会将该项目作为一个 “对象” 返回。

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

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

注意：下面的代码返回未定义的：`trace(myComboBoxInstance.version);`。

ComboBox 类的方法摘要

属性	描述
ComboBox.addItem()	向列表的结尾添加项目。
ComboBox.addItemAt()	向列表的结尾在指定的索引处添加项目。
ComboBox.close()	关闭下拉列表。
ComboBox.getItemAt()	返回指定索引处的项目。
ComboBox.open()	打开下拉列表。
ComboBox.removeAll()	删除列表中的所有项目。
ComboBox.removeItemAt()	删除位于列表中指定位置的项目。
ComboBox.replaceItemAt()	用其他指定项目替换列表中的某个项目。

继承 UIObject 类和 UIComponent 类中的所有方法。

ComboBox 类的属性摘要

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

继承 UIObject 类和 UIComponent 类的所有属性。

ComboBox 类的事件摘要

事件	描述
ComboBox.change	当组合框的值因用户交互操作而改变时广播。
ComboBox.close	当下拉列表开始关闭时广播。
ComboBox.enter	当按下 Enter 键时广播。
ComboBox.itemRollOut	当指针滑离一个下拉列表项时广播。
ComboBox.itemRollOver	当滑过下拉列表的一个项目时广播。
ComboBox.open	当下拉列表开始打开时广播。
ComboBox.scroll	当滚动下拉列表时广播。

继承 UIObject 类和 UIComponent 类的所有事件。

ComboBox.addItem()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
comboBoxInstance.addItem(label[, data])
```

用法 2：

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

用法 3：

```
comboBoxInstance.addItem(obj);
```

参数

label 一个字符串，它指明新项目的标签。

data 该项目的数据，可以是任何数据类型。此参数是可选的。

obj 具有 label 属性和可选 data 属性的对象。

返回

在其位置添加了项目的索引。

描述

方法；在列表的结尾添加新项目。

范例

下面的代码给 myComboBox 实例添加一个项目：

```
myComboBox.addItem("this is an Item");
```

ComboBox.addItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
comboBoxInstance.addItemAt(index, label[, data])
```

参数

index 一个等于或大于 0 的数字，指明要插入项目的位置 （新项目的索引）。

label 一个字符串，它指明新项目的标签。

data 该项目的数据，可以是任何数据类型。此参数是可选的。

返回

在其位置添加了项目的索引。

描述

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

范例

下面的代码在索引 3 处 （即组合框列表中的第 4 个位置 （0 是第 1 个位置））插入一个项目：

```
myBox.addItemAt(3, "this is the fourth Item");
```

ComboBox.change

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(change){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
  // 此处是您的代码
}
comboBoxInstance.addEventListener("change", listenerObject)
```

描述

事件；当组合框的值因用户交互操作而改变时向所有已注册的侦听器广播。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 `myBox` 上，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 调度一个事件 （在本例中为 `change`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `addEventListener()` 方法 （请参阅 `UIEventDispatcher.addEventListener()`），以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例将生成 `change` 事件的组件的实例名称发送到 “输出” 面板：

```
form.change = function(eventObj){
  trace("Value changed to " + eventObj.target.value);
}
myCombo.addEventListener("change", form);
```

另请参见

`UIEventDispatcher.addEventListener()`

ComboBox.close()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox`.close()`

参数

无。

返回

无。

描述

方法；关闭下拉列表。

范例

下面的范例关闭组合框 myBox 的下拉列表：

```
myBox.close();
```

另请参见

ComboBox.open()

ComboBox.close

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(close){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.close = function(eventObject){
  // 此处是您的代码
}
comboBoxInstance.addEventListener("close", listenerObject)
```

描述

事件；当组合框的列表开始回缩时，向所有已注册的侦听器广播。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 myBox 上，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 调度一个事件 （在本例中为 close），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例在下拉列表开始关闭时，向 “输出” 面板发送一条消息：

```
form.close = function(){
  trace("The combo box has closed");
}
myCombo.addEventListener("close", form);
```

另请参见

`UIEventDispatcher.addEventListener()`

ComboBox.dataProvider

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

comboBoxInstance.dataProvider

描述

属性；在列表中查看的项目的数据模型。该属性的值可以是一个数组或任何实现 DataProvider 接口的对象。默认值为 []。这是 List 组件的一个属性，但是可以从 ComboBox 组件实例中直接访问。

List 组件以及其他支持数据的组件会将方法添加到 Array 对象的原型，以便它们符合 DataProvider 接口（有关详细信息，请参阅 DataProvider.as）。因此，任何同时作为列表存在的数组都会自动具有作为列表的模型所需的所有方法（`addItem()`、`getItemAt()` 等等），并可用于向多个组件广播模型更改。

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

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

任何实现 DataProvider 接口的实例都可以作为 List 的数据提供程序。这包括 Flash Remoting RecordSets、Firefly DataSets，等等。

范例

本范例使用一个字符串数组来填充下拉列表：

```
comboBox.dataProvider = ["Ground Shipping","2nd Day Air","Next Day Air"];
```

本范例创建了一个数据提供程序数组，并将它分配给 dataProvider 属性，如下所示：

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

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

ComboBox.dropdown

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.dropdown

描述

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

另请参见

ComboBox.dropdownWidth

ComboBox.dropdownWidth

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.change

描述

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

范例

下面的代码将 dropdownWidth 设置为 150 个像素：

```
myComboBox.dropdownWidth = 150;
```

另请参见

ComboBox.dropdown

ComboBox.editable

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.editable

描述

属性；指明组合框是 (true) 否 (false) 可编辑。对于可编辑组合框，可以在文本框中输入值，这些值不显示在下拉列表中。如果组合框是不可编辑的，则只有下拉列表中列出的值才可以输入到文本框中。默认值为 false。

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

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

范例

下面的代码将 myComboBox 设置为可编辑的：

```
myComboBox.editable = true;
```

ComboBox.enter

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(enter){
   // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.enter = function(eventObject){
  // 此处是您的代码
}
comboBoxInstance.addEventListener("enter", listenerObject)
```

描述

事件；当在文本框中按下 Enter 键时，向所有已注册的侦听器广播。该事件只从可编辑的组合框广播。这是一个从组合框广播的 TextInput 事件。有关详细信息，请参阅 TextInput.enter。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 myBox 上，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 会调度一个事件 （在本例中为 enter），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例在下拉列表开始关闭时，向 “输出” 面板发送一条消息：

```
form.enter = function(){
  trace("The combo box enter event was triggered");
}
myCombo.addEventListener("enter", form);
```

另请参见

UIEventDispatcher.addEventListener()

ComboBox.getItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
comboBoxInstance.getItemAt(index)
```

参数

index 一个大于或等于 0，且小于 ComboBox.length 的数字。要检索的项目的索引。

返回

被索引的项目对象或值。如果索引超出范围，该值就会是未定义的。

描述

方法；检索所指定索引处的项目。

范例

下面的代码显示索引位置 4 处的项目：

```
trace(myBox.getItemAt(4).label);
```

ComboBox.itemRollOut

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(itemRollOut){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.itemRollOut = function(eventObject){
  // 此处是您的代码
}
comboBoxInstance.addEventListener("itemRollOut", listenerObject)
```

事件对象

除了事件对象的标准属性外，`itemRollOut` 事件还有一个附加属性：`index`。`index` 是已经滚动出的项目的数量。

描述

事件；当指针滚动出下拉列表项目时，向所有已注册的侦听器广播。这是一个从组合框广播的 List 事件。有关详细信息，请参阅 `List.itemRollOut`。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 `myBox` 上，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 调度一个事件 （在本例中为 itemRollOut），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

最后，对广播该事件的组件实例调用 addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

范例

下面的范例向 “输出” 面板发送一条消息，指明已经滚动出哪些项目索引编号：

```
form.itemRollOut = function (eventObj) {
  trace("Item #" + eventObj.index + " has been rolled out of.");
}
myCombo.addEventListener("itemRollOut", form);
```

另请参见

ComboBox.itemRollOver, UIEventDispatcher.addEventListener()

ComboBox.itemRollOver

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(itemRollOver){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.itemRollOver = function(eventObject){
  // 此处是您的代码
}
comboBoxInstance.addEventListener("itemRollOver", listenerObject)
```

事件对象

除了事件对象的标准属性外，itemRollOver 事件还有一个附加属性：index。index 是滑过的项目的数量。

描述

事件；当滑过下拉列表项目时，向所有已注册的侦听器广播。这是一个从组合框广播的 List 事件。有关详细信息，请参阅 List.itemRollOver。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 myBox 上，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 调度一个事件（在本例中为 itemRollOver），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

最后，对广播该事件的组件实例调用 addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

范例

下面的范例向 “输出” 面板发送一条消息，指明已在哪个项目索引编号上滑过：

```
form.itemRollOver = function (eventObj) {
  trace("Item #" + eventObj.index + " has been rolled over.");
}
myCombo.addEventListener("itemRollOver", form);
```

另请参见

ComboBox.itemRollOut, UIEventDispatcher.addEventListener()

ComboBox.labelField

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.labelField

描述

属性；在 dataProvider 数组对象中要用作标签字段的字段名称。这是 List 组件的一个属性，可以从 ComboBox 组件实例中使用。有关详细信息，请参阅 List.labelField。

默认值未定义。

范例

下面的范例将 dataProvider 属性设置为一个字符串数组并设置 labelField 属性，以指明应将 name 字段用作下拉列表的标签：

```
myComboBox.dataProvider  = [
  {name:"Gary", gender:"male"},
  {name:"Susan", gender:"female"} ];

myComboBox.labelField = "name";
```

另请参见

List.labelFunction

ComboBox.labelFunction

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.labelFunction

描述

属性；一个计算 dataProvider 项目标签的函数。必须定义该函数。默认值未定义。

范例

下面的范例创建了一个数据提供程序，然后定义一个函数以指定用作下拉列表标签的内容：

```
myComboBox.dataProvider  = [
  {firstName:"Nigel", lastName:"Pegg", age:"really young"},
  {firstName:"Gary", lastName:"Grossman", age:"young"},
  {firstName:"Chris", lastName:"Walcott", age:"old"},
  {firstName:"Greg", lastName:"Yachuk", age:"really old"} ];

myComboBox.labelFunction = function(itemObj){
  return (itemObj.lastName + ", " + itemObj.firstName);
}
```

另请参见

List.labelField

ComboBox.length

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.length

描述

属性（只读）；下拉列表的长度。这是 List 组件的一个属性，可以从 ComboBox 实例使用。有关详细信息，请参阅 List.length。默认值为 0。

范例

下面的范例将 length 的值存储到一个变量：

```
dropdownItemCount = myBox.length;
```

ComboBox.open()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
myComboBox.open()
```

参数

无。

返回

无。

描述

属性；打开下拉列表。

范例

下面的代码打开 `combo1` 实例的下拉列表：

```
combo1.open();
```

另请参见

`ComboBox.close()`

ComboBox.open

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(open){
   // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.open = function(eventObject){
   // 此处是您的代码
}
comboBoxInstance.addEventListener("open", listenerObject)
```

描述

事件；当下拉列表开始出现时，向所有已注册的侦听器广播。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 `myBox` 上，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 调度一个事件 （在本例中为 `open`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

范例

以下范例向 “输出” 面板发送一条消息，以指明已经滚动出的项目索引号：

```
form.open = function () {
  trace("The combo box has opened with text " + myBox.text);
}
myBox.addEventListener("open", form);
```

另请参见

`ComboBox.close`, `UIEventDispatcher.addEventListener()`

ComboBox.removeAll()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
comboBoxInstance.removeAll()
```

参数

无。

返回

无。

描述

方法；删除列表中的所有项目。这是 List 组件的一个方法，可以从 ComboBox 组件的实例中获得。

范例

下列代码会清除列表：

```
myCombo.removeAll();
```

另请参见

ComboBox.removeItemAt(), ComboBox.replaceItemAt()

ComboBox.removeItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.removeItemAt(index)
```

参数

index 一个数字，指明要删除的项目的位置。该值是从零开始的。

返回

一个对象；已删除的项目（如果项目不存在，则它是未定义的）。

描述

方法；删除指定索引位置处的项目。*index* 参数指明的索引后面的列表索引会折叠一项。这是 List 组件的一个方法，可以从 ComboBox 组件的实例中获得。

范例

下面的代码会删除在索引位置 3 的项目：

```
myCombo.removeItemAt(3);
```

另请参见

ComboBox.removeAll(), ComboBox.replaceItemAt()

ComboBox.replaceItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
comboBoxInstance.replaceItemAt(index, label[, data])
```

参数

index 一个等于或大于 0 的数字，指明要插入项目的位置 （新项目的索引）。

label 一个字符串，它指明新项目的标签。

data 项目的数据。此参数是可选的。

返回

无。

描述

方法；替换 index 参数指定的索引处的项目内容。它是 List 组件的一个方法，可以从 ComboBox 组件中获得。

范例

以下范例会更改第三个索引位置：

```
myCombo.replaceItemAt(3, "new label");
```

另请参见

ComboBox.removeAll(), ComboBox.removeItemAt()

ComboBox.rowCount

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
myComboBox.rowCount
```

描述

属性；下拉列表中可见的最大行数。默认值为 5。

如果下拉列表中的项目数大于或等于 rowCount 属性，那么它会调整大小，并根据需要显示滚动条。如果下拉列表中包含的项目数少于 rowCount 属性，那么它会调整到列表中的项目数。

这种行为与 List 组件的行为不同， List 组件会始终显示 rowCount 属性所指定的行数，即使有些地方出现空白。

如果该值是负的或小数，则该行为是未定义的。

范例

以下范例指定组合框中应该可以看到 20 行或更少的行：

```
myComboBox.rowCount = 20;
```

ComboBox.scroll

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(scroll){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.scroll = function(eventObject){
  // 此处是您的代码
}
comboBoxInstance.addEventListener("scroll", listenerObject)
```

事件对象

除了标准的事件对象属性之外，滚动事件还有另一个 direction 属性。它是一个字符串，有两个可能的值："horizontal" 或 "vertical"。对于 ComboBox scroll 事件，该值始终是 "vertical"。

描述

事件；当滚动下拉列表时，向所有已注册的侦听器广播。这是一个 List 组件事件，可以用于 ComboBox。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 ComboBox 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下面的代码附加到 ComboBox 组件实例 myBox 上，它将 "_level0.myBox" 发送到 "输出" 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*comboBoxInstance*) 调度一个事件 （在本例中为 `scroll`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

范例

以下范例向 “输出”面板发送一条消息，以指明已经滚动到的项目索引号：

```
form.scroll = function(eventObj){
  trace("The list had been scrolled to 项 # " + eventObj.target.vPosition);
}
myCombo.addEventListener("scroll", form);
```

另请参见

`UIEventDispatcher.addEventListener()`

ComboBox.selectedIndex

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.selectedIndex

描述

属性；下拉列表中所选项目的索引（数值）。默认值为 0。给该属性赋值会清除当前的选择，而选择指定项目，并在组合框的文本框中显示指定项目的标签。

如果对 `selectedIndex` 的赋值超出了范围，则会被忽略。在可编辑组合框的文本字段中输入文本会将 `selectedIndex` 设置为未定义。

范例

下列代码会选择列表中的最后一项：

```
myComboBox.selectedIndex = myComboBox.length-1;
```

另请参见

`ComboBox.selectedItem`

ComboBox.selectedItem

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.selectedItem

描述

属性；下拉列表中所选项目的值。

如果组合框是可编辑的，在文本框中输入任何文本时，selectedItem 都将返回未定义。如果您从下拉列表中选择一项，或者通过动作脚本设置该值，那么它将只有一个值。如果组合框是静态的，则 selectedItem 的值始终有效。

范例

如果数据提供程序包含了原始类型，则以下范例会显示 selectedItem：

```
var item = myComboBox.selectedItem;
trace("You selected the item " + item);
```

如果数据提供程序包含了具有 label 和 data 属性的对象，则以下范例会显示 selectedItem：

```
var obj = myComboBox.selectedItem;
trace("You have selected the color named:" + obj.label);
trace("The hex value of this color is:" + obj.data);
```

另请参见

ComboBox.dataProvider, ComboBox.selectedIndex

ComboBox.text

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.text

描述

属性；文本框中的文本。您可以为可编辑的组合框获取和设定该值。对于静态的组合框，该值是只读的。

范例

以下范例设置了可编辑组合框的当前 text 值：

```
myComboBox.text = "California";
```

ComboBox.textField

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.textField

描述

属性 （只读）；对 ComboBox 包含的 TextInput 组件的引用。

使用该属性，您可以访问下层的 TextInput 组件，这样您就可以操作该组件。例如，您可能需要更改文本框的选定内容或者限制可以输入到文本框中的字符。

范例

下列代码会限制 myComboBox 的文本框只接受数字：

```
myComboBox.textField.restrict = "0-9";
```

ComboBox.value

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

myComboBox.value

描述

属性（只读）；如果组合框是可编辑的，value 会返回文本框的值。如果组合框是静态的，value 会返回下拉列表的值。下拉列表的值为 data 字段，或者，如果 data 字段不存在，则为 label 字段。

范例

以下范例通过设置 dataProvider 属性，将数据输入组合框。然后，它会在 “输出” 面板中显示 value。最后，它选择 "California"，并将其显示在文本框中，如下所示：

```
cb.dataProvider = [
  {label:"Alaska", data:"AZ"},
  {label:"California", data:"CA"},
  {label:"Washington", data:"WA"}];

cb.editable = true;
cb.selectedIndex = 1;
trace('Editable value is "California":'+ cb.value);

cb.editable = false;
```

```
cb.selectedIndex = 1;
trace('Non-editable value is "CA":'+ cb.value);
```

DataBinding 包

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DataGrid 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DataHolder 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DataProvider 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DataSet 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DateChooser 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DateField 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

DepthManager 类

动作脚本类命名空间 mx.managers.DepthManager

DepthManager 类向动作脚本的 MovieClip 类添加功能，使您可以管理任何组件或影片剪辑的相对深度（包括 `_root`）分配。它还允许您为光标或工具提示这样的系统级服务来管理在 `_root` 上的特殊最深剪辑中保留的深度。

以下方法组成了按相对深度排序的 API：

- `DepthManager.createChildAtDepth()`
- `DepthManager.createClassChildAtDepth()`
- `DepthManager.setDepthAbove()`
- `DepthManager.setDepthBelow()`
- `DepthManager.setDepthTo()`

以下方法组成了保留深度空间 API：

- `DepthManager.createClassObjectAtDepth()`
- `DepthManager.createObjectAtDepth()`

DepthManager 类的方法摘要

方法	描述
DepthManager.createChildAtDepth()	在指定深度处创建指定元件的子级。
DepthManager.createClassChildAtDepth()	在该指定深度处创建指定类的对象。
DepthManager.createClassObjectAtDepth()	在特殊最深剪辑中的指定深度处创建指定类的实例。
DepthManager.createObjectAtDepth()	在最深剪辑中的指定深度处创建一个对象。
DepthManager.setDepthAbove()	将深度设置到指定实例之上。
DepthManager.setDepthBelow()	将深度设置到指定实例之下。
DepthManager.setDepthTo()	将深度设置为最深剪辑中的指定实例。

DepthManager.createChildAtDepth()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

movieClipInstance.createChildAtDepth(*linkageName*, *depthFlag*[, *initObj*])

参数

linkageName 链接标识符。此参数是一个字符串。

depthFlag 它是下列值之一：DepthManager.kTop、DepthManager.kBottom、DepthManager.kTopmost、DepthManager.kNotopmost。所有深度标记都是 DepthManger 类的静态属性。您必须引用 DepthManager 包（例如 mx.managers.DepthManager.kTopmost），或者使用 import 语句导入 DepthManager 包。

initObj 一个初始化对象。此参数是可选的。

返回

指向所创建的对象的引用。

描述

方法；创建一个由 *linkageName* 参数指定的元件的子实例，该子实例位于 *depthFlag* 参数指定的深度。

范例

下面的范例创建 MinuteSymbol 影片剪辑的实例 minuteHand，并将其放在 clock 之上：

```
import mx.managers.DepthManager;
minuteHand = clock.createChildAtDepth("MinuteSymbol", DepthManager.kTop);
```

DepthManager.createClassChildAtDepth()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
movieClipInstance.createClassChildAtDepth( className, depthFlag[, initObj] )
```

参数

className 一个类名称。

depthFlag 它是下列值之一：`DepthManager.kTop`、`DepthManager.kBottom`、`DepthManager.kTopmost`、`DepthManager.kNotopmost`。所有深度标记都是 DepthManger 类的静态属性。您必须引用 DepthManager 包（例如 `mx.managers.DepthManager.kTopmost`），或者使用 `import` 语句导入 DepthManager 包。

initObj 一个初始化对象。此参数是可选的。

返回

指向所创建的子级的引用。

描述

方法；在 *depthFlag* 参数指定的深度创建一个 *className* 参数指定的类的子类。

范例

下面的代码在所有 NoTopmost 对象之上绘制一个焦点矩形：

```
import mx.managers.DepthManager
this.ring = createClassChildAtDepth(mx.skins.RectBorder, DepthManager.kTop);
```

下面的代码创建一个 Button 类的实例，并将其 `label` 属性的值作为 *initObj* 参数传递给该实例：

```
import mx.managers.DepthManager
button1 = createClassChildAtDepth(mx.controls.Button, DepthManager.kTop,
  {label:"Top Button"});
```

DepthManager.createClassObjectAtDepth()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
DepthManager.createClassObjectAtDepth(className, depthSpace[, initObj])
```

参数

className 一个类名称。

depthSpace 它是下列值之一：DepthManager.kCursor、DepthManager.kTooltip。所有深度标记都是 DepthManger 类的静态属性。您必须引用 DepthManager 包（例如 mx.managers.DepthManager.kCursor），或者使用 import 语句导入 DepthManager 包。

initObj 一个初始化对象。此参数是可选的。

返回

指向所创建的对象的引用。

描述

方法；在 *depthSpace* 参数指定的深度创建一个 *className* 参数指定的类的对象。该方法用于访问特殊最高深度剪辑中保留的深度空间。

范例

下面的范例创建一个 Button 类的对象：

```
import mx.managers.DepthManager
myCursorButton = createClassObjectAtDepth(mx.controls.Button,
  DepthManager.kCursor, {label:"Cursor"});
```

DepthManager.createObjectAtDepth()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
DepthManager.createObjectAtDepth(linkageName, depthSpace[, initObj])
```

参数

linkageName 链接标识符。

depthSpace 它是下列值之一：DepthManager.kCursor、DepthManager.kTooltip。所有深度标记都是 DepthManger 类的静态属性。您必须引用 DepthManager 包（例如 mx.managers.DepthManager.kCursor），或者使用 import 语句导入 DepthManager 包。

initObj 一个初始化对象。

返回

指向所创建的对象的引用。

描述

方法；在指定深度处创建一个对象。该方法用于访问特殊最高深度剪辑中保留的深度空间。

范例

下面的范例创建一个 TooltipSymbol 元件的实例，并将其放在为工具提示保留的深度：

```
import mx.managers.DepthManager
myCursorTooltip = createObjectAtDepth("TooltipSymbol", DepthManager.kTooltip);
```

DepthManager.setDepthAbove()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

movieClipInstance.setDepthAbove(*instance*)

参数

instance 实例名称。

返回

无。

描述

方法；将影片剪辑或组件实例的深度设置到 *instance* 参数指定的实例的深度之上。

DepthManager.setDepthBelow()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
movieClipInstance.setDepthBelow(instance)
```

参数

instance 实例名称。

返回

无。

描述

方法；将影片剪辑或组件实例的深度设置到 *instance* 参数指定的实例的深度之下。

范例

下列代码将 `textInput` 实例的深度设置到 `button` 的深度之下：

```
textInput.setDepthBelow(button);
```

DepthManager.setDepthTo()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
movieClipInstance.setDepthTo(depth)
```

参数

depth 深度级别。

返回

无。

描述

方法；将 *movieClipInstance* 的深度设置为 *depth* 指定的值。此方法将实例移动到其他深度，以便留出空间供其他对象使用。

范例

下面的范例将 mc1 实例的深度设置为深度 10：

```
mc1.setDepthTo(10);
```

有关深度和堆叠顺序的详细信息，请参阅 “动作脚本字典”帮助中的 “确定下一个最高可用深度”。

FocusManager 类

您可以使用 FocusManager 来指定一个顺序，当用户按 Tab 键在应用程序中定位时，组件将按此顺序接受焦点。您可以使用 FocusManager API 在文档中设置一个按钮，当用户按 Enter 键 (Windows) 或 Return 键 (Macintosh) 时，该按钮会接收键盘输入。例如，当用户填完一张表单后，他们应该能够使用 Tab 键在字段之间切换，然后按 Enter 键 (Windows) 或 Return 键 (Macintosh) 来提交表单。

所有组件都支持 FocusManager；您无需编写代码来调用它。FocusManager 也会与系统管理器进行交互，当激活或取消激活弹出窗口时，系统管理器会激活或取消激活 FocusManager 实例。每个模式窗口都有一个 FocusManager 实例，所以，该窗口中的组件就成为了它们自己的 Tab 集，这样就可以防止用户按 Tab 键切换到其他窗口中的组件。

FocusManager 可识别单选按钮组（这些按钮具有定义的 RadioButton.groupName 属性），并将焦点设置到该组中 selected 属性设置为 true 的实例。当按 Tab 键时，焦点管理器会查看下一个对象是否与当前对象具有相同的 groupName。如果是这样，那么它会自动将焦点移动到下一个具有不同 groupName 的对象。其他支持 groupName 属性的组件组也可以使用这一功能。

FocusManager 会处理由鼠标单击而引起的焦点变化。如果用户单击一个组件，则该组件就被赋予焦点。

FocusManager 不会自动给应用程序中的组件指定焦点。除非您对组件调用 focusManager.setFocus，否则默认情况下，主窗口和任何弹出窗口都不会在任何组件上设置焦点。

使用 FocusManager

要在应用程序中创建焦点导航，请在应接收焦点的任何对象（包括按钮）上设置 tabIndex 属性。当用户按下 Tab 键时，FocusManager 就会查找一个已启用对象，此对象应具有比 tabIndex 当前值更高的 tabIndex 属性。FocusManager 达到 tabIndex 属性的最高值后，它就会返回到零。因此，在以下范例中，首先 comment 对象（可能是 TextArea 组件）接收焦点，然后 okButton 对象接收焦点：

```
comment.tabIndex = 1;
okButton.tabIndex = 2;
```

要创建一个当用户按下 Enter 键 (Windows) 或者 Return 键 (Macintosh) 时接收焦点的按钮，可将 FocusManager.defaultPushButton 属性设为所需按钮的实例名称，如下所示：

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

注意：FocusManager 与对象放在舞台上的时间（对象的深度顺序）有关，而与它们在舞台上的相对位置无关。这与 Flash Player 处理 Tab 键排序的方式不同。

FocusManager 参数

FocusManager 没有创作参数。你必须使用 “动作” 面板中 FocusManager 类的动作脚本方法和属性。有关详细信息，请参阅 FocusManager 类。

创建具有 FocusManager 的应用程序

以下步骤会在 Flash 应用程序中创建一个焦点方案。

1. 将 TextInput 组件从 “组件” 面板拖到舞台中。
2. 在属性检查器中，为它分配实例名称 **comment**。
3. 将 Button 组件从 “组件” 面板拖到舞台中。
4. 在属性检查器中，为它分配实例名称 **okButton**，并将标签参数设置为 **OK**。
5. 在 “动作” 面板的第 1 帧中，输入下列代码：

```
comment.tabIndex = 1;
okButton.tabIndex = 2;
focusManager.setFocus(comment);
focusManager.defaultPushButton = okButton;
lo = new Object();
lo.click = function(){
  trace("button was clicked");
}
okButton.addEventListener("click", lo);
```

这段代码设置了 Tab 键顺序，并指定一个默认按钮，在用户按 Enter 键 (Windows) 或 Return 键 (Macintosh) 时可以用来接收 `click` 事件。

自定义 FocusManager

通过更改 `themeColor` 样式的值，可以更改光晕主题中焦点环的颜色。

FocusManager 使用 FocusRect 外观来绘制焦点。可以替换或修改此外观，子类也可以覆盖 `UIComponent.drawFocus` 以绘制自定义的焦点指示符。

FocusManager 类

继承 UIObject > UIComponent > FocusManager

动作脚本类命名空间 mx.managers.FocusManager

FocusManager 类的方法摘要

方法	描述
FocusManager.getFocus()	返回对具有焦点的对象的引用。
FocusManager.sendDefaultPushButtonEvent()	给注册到默认普通按钮的侦听器对象发送一个 click 事件。
FocusManager.setFocus()	给指定对象设置焦点。

FocusManager 类的属性摘要

方法	描述
FocusManager.defaultPushButton	一个对象，该对象在用户按 Return 键或 Enter 键时接收 click 事件。
FocusManager.defaultPushButtonEnabled	指明是启用 (true) 还是禁用 (false) 默认普通按钮的键盘处理功能。默认值为 true。
FocusManager.enabled	指明是启用 (true) 还是禁用 (false) Tab 键的处理功能。默认值为 true。
FocusManager.nextTabIndex	tabIndex 属性的下一个值。

FocusManager.defaultPushButton

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
focusManager.defaultPushButton
```

描述

属性；为应用程序指定默认普通按钮。当用户按 Enter 键 (Windows) 或 Return 键 (Macintosh) 时，默认普通按钮的侦听器会接收到一个 `click` 事件。默认值未定义，而且这个属性的数据类型是对象类型。

FocusManager 使用 SimpleButton 类的强调样式声明以可视方式指明当前的默认普通按钮。

`defaultPushButton` 属性的值始终是具有焦点的按钮。设置 `defaultPushButton` 属性无法将初始焦点指定给默认的普通按钮。如果应用程序中有多个按钮，则在按下 Enter 键或 Return 键时，由当前具有焦点的按钮接收 `click` 事件。如果在按下 Enter 键或 Return 键时某个其他组件具有焦点，则将 `defaultPushButton` 属性重置为其原始值。

范例

下列代码将默认普通按钮设置为 `OKButton` 实例：

```
FocusManager.defaultPushButton = OKButton;
```

另请参见

`FocusManager.defaultPushButtonEnabled`, `FocusManager.sendDefaultPushButtonEvent()`

FocusManager.defaultPushButtonEnabled

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
focusManager.defaultPushButtonEnabled
```

描述

属性；一个布尔值，确定是启用 (`true`) 还是禁用 (`false`) 默认普通按钮的键盘处理功能。将 `defaultPushButtonEnabled` 设置为 `false`，可以使组件接收 Return 键或 Enter 键，并在内部对其进行处理。必须通过监视组件的 `onKillFocus()` 方法（请参阅 “动作脚本字典” 帮助中的 `MovieClip.onKillFocus`）或 `focusOut` 事件来重新启用默认普通按钮处理功能。默认值为 `true`。

范例

下列代码会禁用默认普通按钮处理功能：

```
focusManager.defaultPushButtonEnabled = false;
```

FocusManager.enabled

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
focusManager.enabled
```

描述

属性；一个布尔值，确定是为一组特定的具有焦点的对象启用 (`true`) 还是禁用 (`false`) Tab 键处理功能。（例如，其他弹出窗口可能具有它自己的 FocusManager。）将 `enabled` 设置为 `false`，可以使组件接收 Tab 处理按键，并在内部对它们进行处理。必须通过监视组件的 `onKillFocus()` 方法（请参阅“动作脚本字典”帮助中的 `MovieClip.onKillFocus`）或 `focusOut` 事件来重新启用 FocusManager 处理功能。默认值为 `true`。

范例

下面的代码禁用 Tab 键处理功能：

```
focusManager.enabled = false;
```

FocusManager.getFocus()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
focusManager.getFocus()
```

参数

无。

返回

对具有焦点的对象的引用。

描述

方法；返回对当前具有焦点的对象的引用。

范例

如果当前具有焦点的对象是 myInputText，下列代码会将焦点设置到 myOKButton：

```
if (focusManager.getFocus() == myInputText)
{
  focusManager.setFocus(myOKButton);
}
```

另请参见

FocusManager.setFocus()

FocusManager.nextTabIndex

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
FocusManager.nextTabIndex
```

描述

属性；下一个可用的 Tab 键索引号。该属性用于动态地设置对象的 tabIndex 属性。

范例

下列代码为 mycheckbox 实例赋予下一个最高的 tabIndex 值：

```
mycheckbox.tabIndex = focusManager.nextTabIndex;
```

另请参见

UIComponent.tabIndex

FocusManager.sendDefaultPushButtonEvent()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
focusManager.sendDefaultPushButtonEvent()
```

参数

无。

返回

无。

描述

方法；给注册到默认普通按钮的侦听器对象发送一个 `click` 事件。使用该方法可以用编程方式发送 `click` 事件。

范例

下面的代码在用户选择 CheckBox 实例 `chb`（复选框将标记为“自动登录”）时触发默认普通按钮的 `click` 事件，并填写用户名和密码字段：

```
name_txt.tabIndex = 1;
password_txt.tabIndex = 2;
chb.tabIndex = 3;
submit_ib.tabIndex = 4;

focusManager.defaultPushButton = submit_ib;

chbObj = new Object();
chbObj.click = function(o){
  if (chb.selected == true){
    name_txt.text = "Jody";
    password_txt.text = "foobar";
    focusManager.sendDefaultPushButtonEvent();
  } else {
    name_txt.text = "";
    password_txt.text = "";
  }
}
chb.addEventListener("click", chbObj);

submitObj = new Object();
submitObj.click = function(o){
  if (password_txt.text != "foobar"){
    trace("error on submit");
  } else {
    trace("Yeah! sendDefaultPushButtonEvent worked!");
  }
}
submit_ib.addEventListener("click", submitObj);
```

另请参见

`FocusManager.defaultPushButton`, `FocusManager.sendDefaultPushButtonEvent()`

FocusManager.setFocus()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
focusManager.setFocus(object)
```

参数

object　对要接收焦点的对象的引用。

返回

无。

描述

方法；将焦点设置为指定的对象。

范例

下列代码将焦点设置到 myOKButton：

```
focusManager.setFocus(myOKButton);
```

另请参见

FocusManager.getFocus()

Form 类

有关此功能的最新信息，请单击 “帮助”选项卡顶部的 “更新”按钮。

Label 组件

一个标签组件就是一行文本。您可以指定一个标签采用 HTML 格式。您也可以控制标签的对齐和大小。 Label 组件没有边框、不能具有焦点，并且不广播任何事件。

每个 Label 实例的实时预览反映了创作时在属性检查器中或在 “组件检查器”面板中对参数所做的更改。标签没有边框，因此，查看它的实时预览的唯一方法就是设置其文本参数。如果文本太长，并且选择设置 autoSize 参数，那么实时预览将不支持 autoSize 参数，而且不能调整标签边框大小。您必须在边框内单击才能选择舞台上的标签。

将 Label 组件添加到应用程序时，可以使用 “辅助功能”面板使其可以由屏幕读取器进行访问。首先，您必须添加以下代码行来启用辅助功能：

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash”帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 label 组件

使用一个 Label 组件为表单的另一个组件创建文本标签；例如， TextInput 字段左侧的 “姓名：”标签来接受用户的姓名。如果您要构建一个应用程序，这个程序使用基于 Macromedia Component Architecture 第 2 版 (v2) 的组件，那么，使用 Label 组件来替代普通文本字段就是一个好方法，因为您可以使用样式来维持一致的外观。

Label 参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 Label 组件设置的创作参数：

text 指明标签的文本；默认值是 Label。

html 指明标签是 (`true`) 否 (`false`) 采用 HTML 格式。如果将 `html` 参数设置为 `true`，就不能用样式来设定 Label 的格式。默认值为 `false`。

autoSize 指明标签的大小和对齐方式应如何适应文本。默认值为 none。参数可以是以下四个值之一：

- none：标签不会调整大小或对齐方式来适应文本。
- left：标签的右边和底部可以调整大小以适应文本。左边和上边不会进行调整。
- center：标签的底部会调整大小以适应文本。标签的水平中心锚定在它原始的水平中心位置。
- right：标签的左边和底部会调整大小以适应文本。上边和右边不会进行调整。

注意： Label 组件的 `autoSize` 属性与内置的动作脚本 TextField 对象的 `autoSize` 属性不同。

您可以使用 Label 实例的方法、属性和事件为其编写动作脚本来设置其他选项。有关详细信息，请参阅 Label 类。

创建具有 Label 组件的应用程序

以下过程解释了如何在创作时将 Label 组件添加到应用程序。在本例中，标签在组合框的旁边，日期在购物车应用程序中。

要创建具有 Label 组件的应用程序，请执行以下操作：

1. 将 Label 组件从 “组件” 面板拖至舞台。
2. 在 “组件检查器” 面板中，执行以下操作：
   - 为 label 参数输入**过期日期**。

自定义 label 组件

在创作时和运行时，您都可以在水平和垂直方向使 Label 组件变形。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。您可以设置 autoSize 创作参数；设置此参数不会改变实时预览中的边框，但是会调整标签的大小。有关详细信息，请参阅第 101 页的 “Label 参数”。在运行时，请使用 `setSize()` 方法 （请参阅 `UIObject.setSize()`）或 `Label.autoSize`。

对 Label 组件使用样式

您可以设置样式属性来更改标签实例的外观。 Label 组件实例中的所有文本必须采用相同的样式。例如，对同一标签内的单词应用 `color` 样式时，不能将一个单词设置为 `"blue"`，而将另一个单词设置为 `"red"`。

如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。

有关样式的详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

Label 组件支持下列样式：

样式	描述
`color`	文本的默认颜色。
`embedFonts`	文档中嵌入的字体。
`fontFamily`	文本的字体名称。
`fontSize`	字体的磅值。
`fontStyle`	字体样式，“常规” 或 “斜体”。
`fontWeight`	字体粗细，“常规” 或 “粗体”。
`textAlign`	文本对齐方式：“左”、“右” 或 “居中”。
`textDecoration`	文本修饰，“无” 或 “下划线”。

使用具有 Label 组件的外观

Label 组件不能使用外观。

有关设计组件外观的详细信息，请参阅第 33 页的 “关于设置组件外观”。

Label 类

继承 UIObject > Label

动作脚本类命名空间 mx.controls.Label

Label 类的属性允许您在运行时为标签指定文本，指明文本是否要采用 HTML 格式，以及标签是否自动调整大小以适应文本。

使用 “动作脚本” 设置的 Label 类的属性会覆盖在属性检查器中或在 “组件检查器” 面板中设置的同名参数。

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

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

注意： 下面的代码返回未定义的：`trace(myLabelInstance.version);`。

Label 类的方法摘要

继承 UIObject 类的所有方法。

Label 类的属性摘要

属性	描述
Label.autoSize	一个字符串，指明如何调整标签大小和对齐方式以适应其 text 属性值。有四种可能的值："none"、"left"、"center" 和 "right"。默认值为 "none"。
Label.html	一个布尔值，它指明标签是 (true) 否 (false) 采用 HTML 格式。
Label.text	标签上的文本。

继承 UIObject 类的所有属性。

Label 类的事件摘要

继承 UIObject 类的所有事件。

Label.autoSize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

labelInstance.autoSize

描述

属性；一个字符串，指明如何调整标签大小和对齐方式以适应其 text 属性的值。有四种可能的值："none"、"left"、"center" 和 "right"。默认值为 "none"。

- none：标签不会调整大小或对齐方式来适应文本。
- left：标签的右边和底边可以调整大小以适应文本。左边和上边不会进行调整。
- center：标签的底边会调整大小以适应文本。标签的水平中心锚定在它原始的水平中心位置。
- right：标签的左边和底边可以调整大小以适应文本。上边和右边不会进行调整。

注意： Label 组件的 autoSize 属性与内置的动作脚本 TextField 对象的 autoSize 属性不同。

Label.html

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

labelInstance.html

描述

属性；一个布尔值，它指明标签是 (true) 否 (false) 采用 HTML 格式。默认值为 false。html 属性设置为 true 的 Label 组件不能用样式设定格式。

即使当 Label.html 设置为 true 时，您也不能对 Label 组件使用 <font color> HTML 标签。例如，在以下范例中，文本 “Hello” 显示为黑色，而不是它原本应该显示的红色 （如果支持 <font color> 的情况下）：

```
lbl.html = true;
lbl.text = "<font color=\"#FF0000\">Hello</font> World";
```

为了从 HTML 格式的文本获得纯文本，可将 HTML 属性设置为 false，然后访问 text 属性。这将去掉 HTML 格式，所以在获取纯文本之前，可能需要将标签文本复制到屏幕以外的 Label 或 TextArea 组件中。

范例

以下范例将 html 属性设置为 true，以便可以使用 HTML 设定标签的格式。然后，将 text 属性设置为一个包含 HTML 格式的字符串，如下所示：

```
labelControl.html = true;
labelControl.text = "The <b>Royal</b> Nonesuch";
```

单词 “Royal” 以粗体显示。

Label.text

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

labelInstance.text

描述

属性；标签的文本。默认值为 "Label"。

范例

下列代码设置了 Label 实例 labelControl 的 text 属性，并将该值发送到 “输出” 面板：

```
labelControl.text = "The Royal Nonesuch";
trace(labelControl.text);
```

List 组件

List 组件是一个可滚动的单选或多选列表框。列表也可以显示图形，其中包含其他组件。您在单击标签或数据参数字段时，会出现 “值” 对话框，您可以使用该对话框来添加显示在 List 中的项目。您也可以使用 `List.addItem()` 和 `List.addItemAt()` 方法来将项目添加到列表。

List 组件使用基于零的索引，其中索引为 0 的项目就是显示在顶端的项目。当使用 List 类的方法和属性添加、删除或替换列表项时，您可能需要指定该列表项的索引。

在单击列表或按 Tab 键切换到列表时，列表获得焦点，您然后可使用以下键控制它：

按键	描述
字母或数字键	跳转到标签中以 Key.getAscii() 作为首字符的下一项。
Ctrl 键	切换键。允许多个不临近的选择和取消选择。
向下箭头	选区会向下移动一项。
Home 键	选区会移动到列表顶端。
Page Down 键	选区会向下移动一页。
Page Up 键	选区会向上移动一页。
Shift 键	连续选择键。允许进行连续选择。
向上箭头	选区会向上移动一项。

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

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航” 或第 93 页的 “FocusManager 类”。

舞台上的每个 List 实例的实时预览反映了创作时在属性检查器或 “组件检查器” 面板中的对参数所做的更改。

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

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 List 组件

您可以建立一个列表，以便用户可以选择一项或多项。例如，用户访问一个电子商务网站需要选择想要购买的项目。一共有 30 个项目，用户在列表中上下滚动，并通过单击选择一项。

您也可以设计一个列表，该列表使用自定义影片剪辑作为行，这样就可以向用户显示更多信息。例如，在电子邮件应用程序中，每个信箱可能就是一个 List 组件，而每行可能会有指明优先级和状态的图标。

List 组件参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 List 组件实例设置的创作参数：

data 填充列表数据的值数组。默认值为 [] （空数组）。没有相应的运行时属性。

labels 填充列表的标签值的文本值数组。默认值为 [] （空数组）。没有相应的运行时属性。

multipleSelection 一个布尔值，它指明是 (true) 否 (false) 可以选择多个值。默认值为 false。

rowHeight 指明每行的高度，以像素为单位。默认值是 20。设置字体不会更改行的高度。

您可以使用 List 实例的方法、属性和事件为其编写动作脚本来设置其他选项。有关详细信息，请参阅 List 类。

创建具有 List 组件的应用程序

以下过程解释了如何在创作时将 List 组件添加到应用程序。在本例中，列表是一个有三个项目的范例。

要将一个简单的 List 组件添加到一个应用程序中，请执行以下操作：

1 将 List 组件从 “组件” 面板拖到舞台。

2 选择列表，然后选择 “修改” > “变形”，调整大小以适应您的应用程序。

3 在属性检查器中，执行以下操作：

- 输入实例名称 **myList**。
- 为标签参数输入 Item1、 Item2 和 Item3。
- 为数据参数输入 item1.html、 item2.html、 item3.html。

4 选择 “控制” > “测试影片”，以查看带项目的列表。

您可以在应用程序中使用数据属性值来打开 HTML 文件。

以下过程解释了如何在创作时将 List 组件添加到应用程序。在本例中，列表是一个有三个项目的范例。

要将 List 组件添加到应用程序，请执行以下操作：

1 将 List 组件从 “组件” 面板拖到舞台。

2 选择列表，然后选择 “修改” > “变形”，调整大小以适应您的应用程序。

3 在 “动作” 面板中，输入实例名称 **myList**

4 在时间轴中选择第一帧，在 “动作” 面板中，输入下列代码：

```
myList.dataProvider = myDP;
```

如果已经定义了名为 `myDP` 的数据提供程序，那么列表中将填入数据。有关数据提供程序的详细信息，请参阅 `List.dataProvider`。

5 选择 “控制” > “测试影片”，以查看带项目的列表。

自定义 List 组件

在创作时和运行时，您都可以按水平方向和垂直方向将 List 组件变形。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `List.setSize()` 方法 （请参阅 `UIObject.setSize()`）。

当调整列表的大小后，列表的行会在水平方向收缩，剪下其中的任何文本。在垂直方向，列表根据需要增加或删除行。滚动条自动对它们定位。有关滚动条的详细信息，请参阅第 173 页的 “ScrollPane 组件”。

对 List 组件使用样式

您可以设置样式属性以更改 List 实例的外观。

List 组件使用下列光晕样式：

样式	描述
alternatingRowColors	指定交替模式中行的颜色。它的值可以是两个或多个颜色（例如 0xFF00FF、0xCC6699 和 0x996699）组成的数组。
backgroundColor	列表的背景颜色。该样式在类样式声明 ScrollSelectList 上进行定义。
borderColor	三维边框的黑色部分或二维边框的彩色部分。
borderStyle	边框样式。可能的值包括：“none”、“solid”、“inset”和“outset”。该样式在类样式声明 ScrollSelectList 上进行定义。
defaultIcon	用于列表行的默认图标的名称。默认值未定义。
rollOverColor	滑过的行的颜色。
selectionColor	所选行的颜色。
selectionEasing	对用于控制编程补间的扩大公式（函数）的引用。
disabledColor	禁用的文本颜色。
textRollOverColor	指针在文本上滑过时，该文本的颜色。
textSelectedColor	选定文本时，该文本的颜色。
selectionDisabledColor	行被选中并禁用之后的颜色。
selectionDuration	选择项目时的任何转换的长度。
useRollOver	确定滑过一行时是否激活突出显示该行。

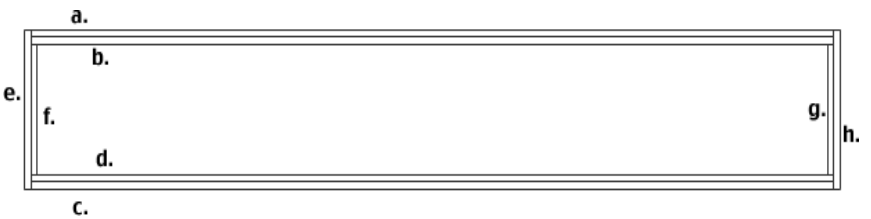
List 组件也使用 Label 组件（请参阅第 102 页的 “对 Label 组件使用样式”）、ScrollBar 和 RectBorder 的样式属性。

使用具有 List 组件的外观

List 组件中的所有外观都包含在组成列表的子组件（ScrollPane 组件和 RectBorder）中。有关详细信息，请参阅 第 173 页的 “ScrollPane 组件”。您可以使用 `setStyle()` 方法（请参阅 `UIObject.setStyle()`）来更改下列 RectBorder 样式属性：

RectBorder 样式	边框位置
`borderColor`	a
`highlightColor`	b
`borderColor`	c
`shadowColor`	d
`borderCapColor`	e
`shadowCapColor`	f
`shadowCapColor`	g
`borderCapColor`	h

这些样式属性设置边框上的下列位置：



List 类

继承 UIObject > UIComponent > View > ScrollView > ScrollSelectList > List

动作脚本类命名空间 mx.controls.List

List 组件由下列三个部分组成：

- 项
- 行
- 数据提供程序

项是用于将信息单位存储在列表中的一个 “动作脚本” 对象。可以将列表看作一个数组；数组中的每个索引空间就是一个项。项是一个对象，它通常有一个 `label` 属性（用于进行显示）和一个 `data` 属性（用于存储数据）。

行是用于显示项的一个组件。默认情况下，列表会提供行（使用 SelectableRow 类），或者您也可以提供行，它们通常作为 SelectableRow 类的子类。SelectableRow 类会实现 CellRenderer 接口，该接口是属性和方法的集合，通过它们，列表可以操作各行并将数据和状态信息（例如，已突出显示、已被选中，等等）发送到该行来进行显示。

数据提供程序是列表中的项列表的数据模型。同一帧中作为列表的任何数组都会自动得到一些方法，这些方法允许您操作数据并将更改广播给多个视图。您可以创建一个 Array 实例或者从服务器上获取一个实例，然后将它用作多个 List、ComboBox、DataGrid 等的数据模型。List 组件有一组代理其数据提供程序的方法（例如，`addItem()` 和 `removeItem()`）。如果没有为列表提供外部数据提供程序，则这些方法会自动创建一个数据提供程序实例，该实例会通过 `List.dataProvider` 被公开。

要将一个 List 组件添加到应用程序的 Tab 键顺序，请设置其 `tabIndex` 属性（请参阅 `UIComponent.tabIndex`）。List 组件用 FocusManager 覆盖默认的 Flash Player 焦点矩形，并绘制一个带有圆角的自定义焦点矩形。有关详细信息，请参阅第 23 页的 “创建自定义焦点导航”。

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

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

注意：下面的代码返回未定义的：`trace(myListInstance.version);`。

List 类的方法摘要

方法	描述
`List.addItem()`	向列表的结尾添加项目。
`List.addItemAt()`	将项目添加到指定索引处的列表。
`List.getItemAt()`	返回指定索引处的项目。
`List.removeAll()`	删除列表中的所有项目。
`List.removeItemAt()`	删除指定索引处的项目。
`List.replaceItemAt()`	用其他项目替换指定索引处的项目。
`List.setPropertiesAt()`	将指定的属性应用到指定的项目。
`List.sortItems()`	按照指定的比较函数对列表中的项目进行排序。
`List.sortItemsBy()`	按照指定的属性对列表中的项目进行排序。

继承 UIObject 类和 UIComponent 类中的所有方法。

List 类的属性摘要

属性	描述
List.cellRenderer	指定 cellRenderer 以用于列表的每行。
List.dataProvider	列表项目的来源。
List.hPosition	列表的水平位置。
List.hScrollPolicy	指明是 ("on") 否 ("off") 显示水平滚动条。
List.iconField	各项目中用于指定图标的字段。
List.iconFunction	一个函数，它确定要使用的图标。
List.labelField	指定各项目中用作标签文本的字段。
List.labelFunction	一个函数，它确定各个项目的哪些字段要用作标签文本。
List.length	项目中列表的长度。该属性为只读。
List.maxHPosition	当将 List.hScrollPolicy 设置为 "on" 时，指定列表可以向右滚动的像素数目。
List.multipleSelection	指明列表中允许 (true) 还是不允许 (false) 多选。
List.rowCount	列表中至少可以看到一部分的行数。
List.rowHeight	列表中每行的像素高度。
List.selectable	指定列表是 (true) 否 (false) 为可选择列表。
List.selectedIndex	单选列表中的选择索引。
List.selectedIndices	多选列表中的已选择项目的数组。
List.selectedItem	单选列表中的已选择项目。该属性为只读。
List.selectedItems	多选列表中的已选择的项目对象。该属性为只读。
List.vPosition	滚动列表，以便使最顶部可见的项目为指定的数。
List.vScrollPolicy	指明是显示 ("on")、不显示 ("off") 还是在需要时显示 ("auto") 垂直滚动条。

继承 UIObject 类和 UIComponent 类的所有属性。

List 类的事件摘要

事件	描述
List.change	所选内容因用户交互操作而更改时广播。
List.itemRollOut	指针在列表项目上滑过然后又滑离时广播。
List.itemRollOver	指针在列表项目上滑过时广播。
List.scroll	滚动列表时，进行广播。

继承 UIObject 类和 UIComponent 类的所有事件。

List.addItem()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.addItem(label[, data])
listInstance.addItem(itemObject)
```

参数

label 一个字符串，它指明新项目的标签。

data 项目的数据。此参数是可选的，可以是任何数据类型。

itemObject 项目对象，通常具有 label 属性和 data 属性。

返回

在其位置添加了项目的索引。

描述

方法；在列表的结尾添加新项目。

在第一个用法范例中，始终使用指定的 label 属性和 data 属性 （如果已指定）来创建项目对象。

第二个用法范例添加了指定的项目对象。

调用此方法会修改 List 组件的数据提供程序。如果与其他组件共享数据提供程序，那些组件也将会更新。

范例

下面两行代码都会将项目添加到 myList 实例。要试试这行代码，可将一个 List 拖到舞台，并为其设定实例名称 **myList**。将下面的代码添加到时间轴中的第一帧：

```
myList.addItem("this is an Item");
myList.addItem({label:"Gordon",age:"very old",data:123});
```

List.addItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.addItemAt(index, label[, data])
listInstance.addItemAt(index, itemObject)
```

参数

label 一个字符串，它指明新项目的标签。

data 项目的数据。此参数是可选的，可以是任何数据类型。

index 一个大于或等于零的数，指明项目的位置。

itemObject 项目对象，通常具有 `label` 属性和 `data` 属性。

返回

在其位置添加了项目的索引。

描述

方法；将新项目添加到 *index* 参数指定的位置。

在第一个用法范例中，始终使用指定的 `label` 属性和 `data` 属性 （如果已指定）来创建项目对象。

第二个用法范例添加了指定的项目对象。

调用此方法会修改 List 组件的数据提供程序。如果与其他组件共享数据提供程序，那些组件也将会更新。

范例

下面代码行会将一个项目添加到第三索引位置，该位置是列表中的第四个项目：

```
myList.addItemAt(3,{label:'Red',data:0xFF0000});
```

List.cellRenderer

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.cellRenderer
```

描述

属性；指定要用于列表各行的单元格渲染器。这个属性必须是一个类对象引用，或者是单元格渲染器使用的元件链接标识符。用于此属性的任何类都必须实现第54页的 “CellRenderer 接口”。

范例

下面的范例使用链接标识符来设置一个新的单元格渲染器：

```
myList.cellRenderer = "ComboBoxCell";
```

List.change

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(change){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
  // 此处是您的代码
}
listInstance.addEventListener("change", listenerObject)
```

描述

事件；当所选的列表的索引因用户交互操作而改变时，向所有已注册的侦听器广播。

第一个用法范例使用了 `on()` 处理函数，并且必须直接附加到一个 list 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 list 组件实例 `myBox`，它将 “_level0.myBox” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本例中为 `change`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

最后，对广播该事件的组件实例调用 `addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

范例

下面的范例将生成 change 事件的组件的实例名称发送到 “输出” 面板：

```
form.change = function(eventObj){
  trace("Value changed to " + eventObj.target.value);
}
myList.addEventListener("change", form);
```

另请参见

UIEventDispatcher.addEventListener()

List.dataProvider

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listnstance.dataProvider

描述

属性；在列表中查看的项目的数据模型。该属性的值可以是一个数组或任何实现 DataProvider 接口的对象。默认值为 []。有关 DataProvider 接口的详细信息，请参阅第 87 页的 “DataProvider 组件”。

List 组件以及其他支持数据的组件会将方法添加到 Array 对象的原型，以便它们符合 DataProvider 接口。因此，任何同时作为列表存在的数组都可以自动获得作为列表的数据模型所需的所有方法 （addItem()、getItemAt() 等），并可用于向多个组件广播模型更改。

如果数组包含对象，则会访问 List.labelField 或 List.labelFunction 属性，以确定要显示项目的哪些部分。默认值为 "label"，如果存在 label 字段，则会选择显示它，如果不存在该字段，则显示用逗号分隔的所有字段的列表。

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

任何实现 DataProvider 接口的实例都可以作为 List 组件的数据提供程序。这包括 Flash Remoting RecordSets、Firefly DataSets，等等。

范例

在本例使用一个字符串数组填充列表：

```
list.dataProvider = ["Ground Shipping","2nd Day Air","Next Day Air"];
```

本例创建一个数据提供程序数组，并将其赋予 dataProvider 属性，如下所示：

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

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

List.getItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.getItemAt(index)
```

参数

index 一个大于或等于 0，且小于 List.length 的数字。要检索的项目的索引。

返回

索引项目对象。如果索引已超出范围，则是未定义的。

描述

方法；检索所指定索引处的项目。

范例

下面代码显示索引位置 4 处的项目的标签：

```
trace(myList.getItemAt(4).label);
```

List.hPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.hPosition
```

描述

属性；水平滚动列表到指定的像素数。只有在 hScrollPolicy 的值是 "on"，而且列表的 maxHPosition 大于 0 的情况下，才能设置 hPosition。

范例

下面的范例获取 myList 的水平滚动位置：

```
var scrollPos = myList.hPosition;
```

以下范例将水平滚动位置始终设置到左侧：

```
myList.hPosition = 0;
```

List.hScrollPolicy

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.hScrollPolicy

描述

属性；确定是否显示水平滚动条的字符串；该值可以是 "on" 或 "off"。默认值为 "off"。水平滚动条不会测量文本，您必须设置最大水平滚动位置，请参阅 List.maxHPosition。

注意： List.hScrollPolicy 不支持值 "auto"。

范例

下面代码最多可以使列表水平滚动 200 像素：

```
myList.hScrollPolicy = "on";
myList.Box.maxHPosition = 200;
```

另请参见

List.hPosition, List.maxHPosition

List.iconField

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.iconField

描述

属性；指定了要用作图标标识符的字段名称。如果字段的值为未定义，则使用 defaultIcon 样式指定的默认图标。如果 defaultIcon 样式是未定义的，则不使用任何图标。

范例

下面的范例将 iconField 属性设置为每个项目的 icon 属性：

```
list.iconField = "icon"
```

另请参见

List.iconFunction

List.iconFunction

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.iconFunction
```

描述

属性；指定一个函数，该函数用于确定每行将使用哪个图标来显示其项目。此函数会接收参数 *item*（该参数指示要被呈现的项目），并且必须返回一个表示图标元件标识符的字符串。

范例

下面范例添加了一些图标，这些图标指明文件是图像还是文本文档。如果 data.fileExtension 字段包含值 "jpg" 或 "gif"，那么所用的图标将是 "pictureIcon"，依此类推：

```
list.iconFunction = function(item){
  var type = item.data.fileExtension;
  if (type=="jpg" || type=="gif") {
    return "pictureIcon";
  } else if (type=="doc" || type=="txt") {
    return "docIcon";
  }
}
```

List.itemRollOut

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(itemRollOut){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.itemRollOut = function(eventObject){
  // 此处是您的代码
}
listInstance.addEventListener("itemRollOut", listenerObject)
```

事件对象

除了事件对象的标准属性外，itemRollOut 事件还有一个附加属性：index。index 是已经滚动出的项目的数量。

描述

事件；当滚动出列表项目时，向所有已注册的侦听器广播。

第一个用法范例使用了一个 `on()` 处理函数，并且必须直接附加到一个 List 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 List 实例 `myList` 上，它将 “_level0.myList” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本例中为 `itemRollOut`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例向 “输出” 面板发送一条消息，指明已在哪个项目索引编号上滑过：

```
form.itemRollOut = function (eventObj) {
  trace("Item #" + eventObj.index + " has been rolled out.");
}
myList.addEventListener("itemRollOut", form);
```

另请参见

`List.itemRollOver`

List.itemRollOver

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(itemRollOver){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.itemRollOver = function(eventObject){
  // 此处是您的代码
}
listInstance.addEventListener("itemRollOver", listenerObject)
```

事件对象

除了事件对象的标准属性外，itemRollOver 事件还有一个附加属性：index。index 是滑过的项目的数量。

描述

事件；当滑过列表项目时，向所有已注册的侦听器广播。

第一个用法范例使用了一个 on() 处理函数，并且必须直接附加到一个 List 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，以下代码附加到 List 实例 myList 上，它将 “_level0.myList” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本例中为 itemRollOver），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例向 “输出” 面板发送一条消息，指明已在哪个项目索引编号上滑过：

```
form.itemRollOver = function (eventObj) {
  trace("Item #" + eventObj.index + " has been rolled over.");
}
myList.addEventListener("itemRollOver", form);
```

另请参见

List.itemRollOut

List.labelField

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.labelField

描述

属性；指定每个项目中的一个字段用作显示文本。此属性会获得该字段的值，并将其用作标签。默认值为 "label"。

范例

下面范例将 labelField 属性设置为每个项目的 "name" 字段 “Nina” 将显示为第二行代码中添加的项目的标签：

```
list.labelField = "name";
list.addItem({name:"Nina", age: 25});
```

另请参见

List.labelFunction

List.labelFunction

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.labelFunction

描述

属性；指定一个函数用于决定要显示每个项目的哪个字段（或字段组合）。此函数会接收一个参数 *item*（该参数指示要被呈现的项目），并且必须返回一个表示要显示的文本的字符串。

范例

以下范例让标签显示项目的一些设置了格式的细节：

```
list.labelFunction = function(item){
  return "The price of product " + item.productID + ", " + item.productName + "
  is $"
  + item.price;
}
```

另请参见

List.labelField

List.length

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.length

描述

属性 （只读）；列表中的项目数。

范例

下面范例将 length 的值放入一个变量中：

```
var len = myList.length;
```

List.maxHPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.maxHPosition

描述

属性；指定当 List.hScrollPolicy 设置为 "on" 时，列表可以滚动的像素数。列表无法精确地测量所包含的文本的宽度。必须设置 maxHPosition 以指明列表需要的滚动量。如果不设置此属性，则列表将不水平滚动。

范例

以下范例创建了一个带 400 像素的可以水平滚动的列表：

```
myList.hScrollPolicy = "on";
myList.maxHPosition = 400;
```

另请参见

List.hScrollPolicy

List.multipleSelection

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.multipleSelection
```

描述

属性；指明是可以多选 (true) 还是只可以单选 (false)。默认值为 false。

范例

以下范例会进行测试，以确定是否可以选择多个项目：

```
if (myList.multipleSelection){
  // 此处是您的代码
}
```

以下范例允许列表进行多选：

```
myList.selectMultiple = true;
```

List.removeAll()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.removeAll()
```

参数

无。

返回

无。

描述

方法；删除列表中的所有项目。

调用此方法会修改 List 组件的数据提供程序。如果与其他组件共享数据提供程序，那些组件也将会更新。

范例

下列代码会清除列表：

```
myList.removeAll();
```

List.removeItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.removeItemAt(index)
```

参数

index 一个字符串，它指明新项目的标签。它是一个大于零但小于 List.length 的值。

返回

一个对象；已删除的项目 （如果项目不存在，则它是未定义的）。

描述

方法；删除指定 *index* 位置处的项目。 *index* 参数指明的索引后面的列表索引会折叠一项。

调用此方法会修改 List 组件的数据提供程序。如果与其他组件共享数据提供程序，那些组件也将会更新。

范例

下面的代码会删除在索引位置 3 的项目：

```
myList.removeItemAt(3);
```

List.replaceItemAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.replaceItemAt(index, label[, data])
listInstance.replaceItemAt(index, itemObject)
```

参数

index 一个大于零且小于 List.length 的数字，指明要插入项目的位置 （新项目的索引）。

label 一个字符串，它指明新项目的标签。

data 项目的数据。此参数是可选参数，它可以是任何类型。

itemObject。 用作项目的对象，通常含有 label 和 data 属性。

返回

无。

描述

方法；替换 index 参数指定的索引处的项目内容。

调用此方法会修改 List 组件的数据提供程序。如果与其他组件共享数据提供程序，那些组件也将会更新。

范例

以下范例更改了第四索引的位置：

```
myList.replaceItemAt(3, "new label");
```

List.rowCount

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.rowCount
```

描述

属性；列表中至少可以看到一部分的行数。如果您已经使用像素缩放了一个列表，但需要计算它的行数，则该属性很有用。相反，设置行数可以保证显示准确的行数，而不会在底部出现部分行。

代码 `myList.rowCount = num` 相当于代码 `myList.setSize(myList.width, h)`（其中 `h` 为显示 `num` 个项目所需的高度）。

默认值基于创作时设置的列表高度，或者由 `list.setSize()` 方法（请参阅 `UIObject.setSize()`）设置。

范例

以下范例会发现列表中的可以看到的项目数量：

```
var rowCount = myList.rowCount;
```

下面范例会使列表显示四个项目：

```
myList.rowCount = 4;
```

在本例会删除列表底部的部分行（如果存在部分行）：

```
myList.rowCount = myList.rowCount;
```

该范例将列表设置为它可以完整显示的最小行数：

```
myList.rowCount = 1;
trace("myList has "+myList.rowCount+" rows");
```

List.rowHeight

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.rowHeight
```

描述

属性；列表中每行的像素高度。字体设置不会增加行高以适应文字，因此设置 `rowHeight` 属性是确保项目完全显示的最佳方法。默认值为 20。

范例

以下范例将每行设置为 30 像素：

```
myList.rowHeight = 30;
```

List.scroll

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(scroll){
  // 此处是您的代码
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.scroll = function(eventObject){
  // 此处是您的代码
}
listInstance.addEventListener("scroll", listenerObject)
```

事件对象

除了标准的事件对象属性之外，`scroll` 事件还有另一个附加属性：`direction`。它是一个字符串，有两个可能的值：`"horizontal"` 或 `"vertical"`。对于 ComboBox 滚动事件，该值始终是 `"vertical"`。

描述

事件；当列表滚动时，向所有已注册的侦听器广播。

第一个用法范例使用了一个 `on()` 处理函数，并且必须直接附加到一个 List 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 List 实例 `myList` 上，它将 “_level0.myList” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*listInstance*) 调度一个事件（在本例中为 `scroll`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例将生成 `change` 事件的组件的实例名称发送到 “输出” 面板：

```
form.scroll = function(eventObj){
  trace("list scrolled");
}
myList.addEventListener("scroll", form);
```

List.selectable

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.selectable

描述

属性；一个指明列表是 (`true`) 否 (`false`) 为可选列表的布尔值。默认值为 `true`。

List.selectedIndex

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.selectedIndex

描述

属性；单选列表的已选择索引。如果没有选中任何内容，则该值是未定义的；如果有多项所选内容，则该值等于最后选中的项目。如果您为 selectedIndex 分配一个值，就会清除当前所做任何选择，并且会选中所指明的项目。

范例

在本例将选择当前所选项目之后的项目。如果没有可选择的项目，则选择项目 0，如下所示：

```
var selIndex = myList.selectedIndex;
myList.selectedIndex = (selIndex==undefined ?0 : selIndex+1);
```

另请参见

List.selectedIndices, List.selectedItem, List.selectedItems

List.selectedIndices

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.selectedIndices

描述

属性；所选项目的索引数组。指定此属性将替换当前选择。将 selectedIndices 设置为 0 长度数组（或未定义的）将清除当前选择。如果没有可选择的项目，则无法定义该值。

selectedIndices 属性以项目的选择顺序列出。如果您单击第二个项目，再单击第三个项目，然后单击第一个项目，则 selectedIndices 将返回 [1,2,0]。

范例

以下范例将获取所选的索引：

```
var selIndices = myList.selectedIndices;
```

以下范例选择了四个项目：

```
var myArray = new Array (1,4,5,7);
myList.selectedIndices = myArray;
```

另请参见

List.selectedIndex, List.selectedItem, List.selectedItems

List.selectedItem

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.selectedItem

描述

属性（只读）；单选列表中的项目对象。（在有多个选中项目的多选列表中，selectedItem 将返回最近选中的项目。）如果没有所选内容，则该值是未定义的。

范例

在本例将显示选中的标签：

```
trace(myList.selectedItem.label);
```

另请参见

List.selectedIndex, List.selectedIndices, List.selectedItems

List.selectedItems

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.selectedItems

描述

属性（只读）；所选项目对象组成的数组。在多选列表中，selectedItems 允许您以项目对象的形式访问选中的项目组。

范例

以下范例会获取一个所选项目对象的数组：

```
var myObjArray = myList.selectedItems;
```

另请参见

List.selectedIndex, List.selectedItem, List.selectedIndices

List.setPropertiesAt()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.setPropertiesAt(index, styleObj)
```

参数

index 一个大于零且小于 `List.length` 的数字，它指明要更改的项目的索引。

styleObj 一个对象，它枚举要设置的属性和值。

返回

无。

描述

方法；将 *styleObj* 参数指定的属性应用到 *index* 参数指定的项目。受支持的属性为 `icon` 和 `backgroundColor`。

范例

以下范例会将第四个项目更改为黑色，并为它指定一个图标：

```
myList.setPropertiesAt(3, {backgroundColor:0x000000, icon:"file"});
```

List.sortItems()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
listInstance.sortItems(compareFunc)
```

参数

compareFunc 引用一个函数。此函数用于比较两个项目，以确定它们的排序顺序。

有关详细信息，请参阅 “动作脚本字典”帮助中的 `Array.sort()`。

返回

在其位置添加了项目的索引。

描述

方法；根据 compareFunc 参数对列表中的项目进行排序。

范例

以下范例基于大写标签对项目进行排序。请注意，传递给函数的参数 a 和 b 指示具有 label 和 data 属性的项目：

```
myList.sortItems(upperCaseFunc);
function upperCaseFunc(a,b){
    return a.label.toUpperCase() > b.label.toUpperCase();
}
```

另请参见

List.sortItemsBy()

List.sortItemsBy()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.sortItemsBy(*fieldName, order*)

参数

fieldName 指定用于排序的属性名称的字符串。通常情况下，该值为 "label" 或 "data"。

order 一个字符串，指定是以升序 ("ASC") 还是以降序 ("DESC") 对项目进行排序。

返回

无。

描述

方法；按字母或数值次序、以指定的顺序、根据指定的 *fieldName* 对列表中的项目进行排序。如果 *fieldName* 项目既包括文本字符串项也包括整数项，则首先列出整数项。通常情况下，*fieldName* 参数为 "label" 或 "data"，但您可以指定任何原始数据值。

范例

下列代码根据列表项的标签、以升序对列表 surnameMenu 中的项目进行排序：

```
surnameMenu.sortItemsBy("label", "ASC");
```

另请参见

List.sortItems()

List.vPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.vPosition

描述

属性；滚动列表以便索引成为可以最先看到的项目。如果索引超出了边框，则转到离边框最近的边框内的索引。默认值为 0。

范例

以下范例将列表位置设置为第一个索引项目：

```
myList.vPosition = 0;
```

List.vScrollPolicy

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

listInstance.vScrollPolicy

描述

属性；一个字符串，确定列表是否支持垂直滚动。该属性可以是下列值之一："on"、"off" 或 "auto"。值 "auto" 指定在需要时显示滚动条。

范例

以下范例将禁用滚动条：

```
myList.vScrollPolicy = "off";
```

您仍然可以通过使用 List.vPosition 创建滚动功能。

另请参见

List.vPosition

Loader 组件

Loader 组件是一个容器，它可以显示 SWF 或 JPEG。您可以缩放加载器的内容，或者调整加载器自身的大小来匹配内容的大小。默认情况下，缩放内容要适合 Loader 的大小。您也可以在运行时加载内容，并监视加载进度。

Loader 组件不能接收焦点。但是，Loader 组件中加载的内容可以接受焦点，并且可以有自己的焦点交互操作。有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航”或第 93 页的 “FocusManager 类”。

每个 Loader 实例的实时预览反映了创作时在属性检查器或 “组件检查器” 面板中对参数所做的更改。

可以启用加载到 Loader 组件中的内容来使用辅助功能。如果启用了，可以使用 “辅助功能” 面板以使屏幕读取程序可以访问内容。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 Loader 组件

当需要从一个远程位置获取内容并将其拖到 “Flash” 应用程序中时，您可以使用加载器。例如，您可以使用加载器将公司徽标 （JPEG 文件）添加到表单。您也可以使用加载器来继承并利用已经完成的 Flash 作品。例如，如果您已经创建了一个 Flash 应用程序，但想扩展该应用程序，可以使用加载器将旧的应用程序拖到新应用程序中，或者将旧应用程序作为某个选项卡界面的一部分。再例如，您可以在显示相片的应用程序中使用加载器组件。使用 `Loader.load()`、`Loader.percentLoaded` 和 `Loader.complete` 可以在加载期间控制图像加载的计时，并为用户显示进度栏。

Loader 组件参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 Loader 组件设置的创作参数：

autoload 指明内容是应该自动加载 (true)，还是应该等到调用 `Loader.load()` 方法时再进行加载 (false)。默认值为 true。

content 一个绝对或相对的 URL，指明要加载到加载器的文件。相对路径必须是相对于加载内容的 SWF 的路径。该 URL 必须与 Flash 内容当前驻留的 URL 在同一子域中。为了在独立的 Flash Player 中使用 SWF 文件，或者在影片测试模式下测试 SWF 文件，必须将所有 SWF 文件存储在同一文件夹中，并且其文件名不能包含文件夹或磁盘驱动器说明。默认值要在开始加载后才定义。

scaleContent 指明是内容缩放以适应加载器 (true)，还是加载器进行缩放以适应内容 (false)。默认值为 true。

您可以使用 Loader 实例的方法、属性和事件来编写动作脚本，以为其设置其他选项。有关详细信息，请参阅 Loader 类。

创建具有 Loader 组件的应用程序

以下过程解释了如何在创作时将 Loader 组件添加到应用程序。在本例中，加载器将从一个虚拟的 URL 中加载一个徽标 JPEG。

要创建具有 Loader 组件的应用程序，请执行以下操作：

1. 将 Loader 组件从 “组件” 面板上拖到舞台上。
2. 在舞台上选择加载器，然后使用 “任意变形” 工具将其调整到公司徽标的尺寸。
3. 在属性检查器中，输入实例名称 **logo**。
4. 选择舞台上的加载器，在 “组件检查器” 面板中执行以下操作：
   - 为 contentPath 参数输入 **http://corp.com/websites/logo/corplogo.jpg**。

自定义 Loader 组件

在创作时和在运行时，都可以在水平和垂直方向上改变 Loader 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `setSize()` 方法 （请参阅 `UIObject.setSize()`）。

Loader 组件的调整大小的行为由 `scaleContent` 属性控制。当 `scaleContent = true` 时，内容会缩放以适合加载器的边界 （当调用 `UIObject.setSize()` 时，重新进行缩放）。当属性为 `scaleContent = false` 时，组件的大小固定为内容的大小，而 `UIObject.setSize()` 方法将无效。

使用具有 Loader 组件的样式

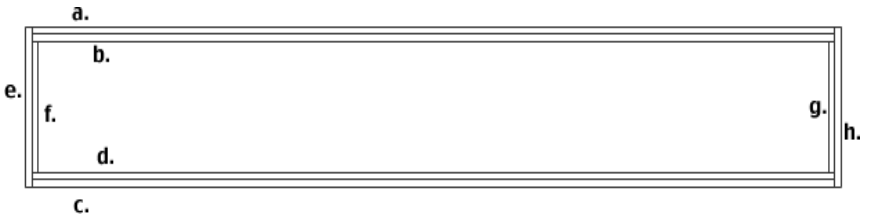
Loader 组件不使用样式。

使用具有 Loader 组件的外观

Loader 组件使用 RectBorder，该 RectBorder 使用动作脚本绘制 API。可以使用 `setStyle()` 方法 （请参阅 `UIObject.setStyle()`）更改下列 RectBorder 样式属性 ：

RectBorder 样式
`borderColor`
`highlightColor`
`borderColor`
`shadowColor`
`borderCapColor`
`shadowCapColor`
`shadowCapColor`
`borderCapColor`

这些样式属性设置边框上的下列位置 ：



Loader 类

继承 UIObject > UIComponent > View > Loader

动作脚本类命名空间 mx.controls.Loader

Loader 类的属性允许您设置要加载的内容并在运行时监视它的加载进程。

使用 “动作脚本” 设置 Loader 类的属性会覆盖在属性检查器中或在 “组件检查器” 面板中设置的同名参数。

有关详细信息，请参阅第 23 页的 “创建自定义焦点导航”。

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

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

注意：下面的代码返回未定义的：`trace(myLoaderInstance.version);`。

Loader 类的方法摘要

方法	描述
`Loader.load()`	加载由 contentPath 属性指定的内容。

继承 UIObject 类和 UIComponent 类中的所有方法。

Loader 类的属性摘要

属性	描述
`Loader.autoLoad`	一个布尔值，指明内容会自动加载 (`true`) 还是您必须调用 `Loader.load()` 才进行加载 (false)。
`Loader.bytesLoaded`	只读属性，指明已经加载的字节数。
`Loader.bytesTotal`	指明内容中的总字节数的只读属性。
`Loader.content`	引用由 `Loader.contentPath` 属性指定的内容。该属性为只读。
`Loader.contentPath`	一个字符串，它指明要加载的内容的 URL。
`Loader.percentLoaded`	一个数字，它指明已加载内容的百分比。该属性为只读。
`Loader.scaleContent`	一个布尔值，它指明是内容会进行缩放以适合加载器 (`true`)，还是加载器会进行缩放以适合内容 (`false`)。

继承 UIObject 类和 UIComponent 类的所有属性。

Loader 类的事件摘要

事件	描述
`Loader.complete`	当内容加载完成时触发。
`Loader.progress`	在内容加载过程中触发。

继承 UIObject 类和 UIComponent 类的所有属性

Loader.autoLoad

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

loaderInstance.autoLoad

描述

属性；一个布尔值，它指明是自动加载内容 (true)，还是等到调用 Loader.load() 才加载内容 (false)。默认值为 true。

范例

下列代码将 loader 组件设置为等待 Loader.load() 调用：

```
loader.autoload = false;
```

Loader.bytesLoaded

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

loaderInstance.bytesLoaded

描述

属性 （只读）；已经加载的内容的字节数目。在开始加载内容之前，默认值为 0。

范例

以下代码创建一个 ProgressBar 和一个 Loader 组件。然后，它创建一个具有 progress 事件处理函数的侦听器对象，该事件处理函数会显示加载的进程。侦听器注册到 loader 实例，如下所示：

```
createClassObject(mx.controls.ProgressBar, "pBar", 0);
createClassObject(mx.controls.Loader, "loader", 1);
loadListener = new Object();
loadListener.progress = function(eventObj){
  // eventObj.target 是生成 change 事件的组件，
  // 即 Loader。
  pBar.setProgress(loader.bytesLoaded, loader.bytesTotal); // 显示进程
}
loader.addEventListener("progress", loadListener);
loader.content = "logo.swf";
```

用 createClassObject() 方法创建实例时，必须用 move() 和 setSize() 方法将其放置在舞台上。请参阅 UIObject.move() 和 UIObject.setSize()。

另请参见

Loader.bytesTotal, UIObject.createClassObject()

Loader.bytesTotal

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

loaderInstance.bytesTotal

描述

属性 （只读）；内容的大小，以字节为单位。在内容开始加载前，默认值为 0。

范例

以下代码创建一个 ProgressBar 和一个 Loader 组件。然后，它创建一个具有 progress 事件处理函数的 load 侦听器对象，该事件处理函数会显示加载的进程。侦听器与 loader 实例一起注册，如下所示：

```
createClassObject(mx.controls.ProgressBar, "pBar", 0);
createClassObject(mx.controls.Loader, "loader", 1);
loadListener = new Object();
loadListener.progress = function(eventObj){
  // eventObj.target 是生成 change 事件的组件，
  // 即 Loader。
  pBar.setProgress(loader.bytesLoaded, loader.bytesTotal); // 显示进程
}
loader.addEventListener("progress", loadListener);
loader.content = "logo.swf";
```

另请参见

Loader.bytesLoaded

Loader.complete

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(complete){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
  ...
}
loaderInstance.addEventListener("complete", listenerObject)
```

描述

事件；当加载完内容时向所有已注册的侦听器广播。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 Loader 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 Loader 组件实例 `myLoaderComponent`，它将 “_level0.myLoaderComponent” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*loaderInstance*) 调度一个事件 （在本例中为 `complete`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下范例创建一个 Loader 组件，然后定义一个带有 `complete` 事件处理函数的侦听器对象，该事件处理函数将 loader 的 `visible` 属性设置为 `true`：

```
createClassObject(mx.controls.Loader, "loader", 0);
loadListener = new Object();
loadListener.complete = function(eventObj){
  loader.visible = true;
}
loader.addEventListener("complete", loadListener);S
loader.contentPath = "logo.swf";
```

Loader.content

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

loaderInstance.content

描述

属性 （只读）；对加载器内容的引用。到加载开始时才会定义该值。

另请参见

Loader.contentPath

Loader.contentPath

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

loaderInstance.contentPath

描述

属性；一个字符串，指明要加载到 loader 中的文件的绝对 URL 或相对 URL。相对路径必须相对于加载内容的 SWF。该 URL 必须与作为加载 SWF 的 URL 在同一子域中。

如果您要使用独立的 Flash Player，或者在 Flash 中使用测试影片模式，必须将所有 SWF 文件存储在同一文件夹中，并且其文件名不能包含文件夹或磁盘驱动器信息。

范例

以下范例指示 loader 实例显示 “logo.swf” 文件的内容：

```
loader.contentPath = "logo.swf";
```

Loader.load()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

loaderInstance.load(*path*)

参数

path 可选参数，指定在开始加载之前 contentPath 属性的值。如果未指定值，则使用 contentPath 的当前值。

返回

无。

描述

方法；指示 loader 开始加载其内容。

范例

以下代码创建一个 Loader 实例并将 `autoload` 属性设置为 `false`，这样加载器必须等待调用 `load()` 方法时才能开始加载内容。然后，它调用 `load()` 并指明要加载的内容：

```
createClassObject(mx.controls.Loader, "loader", 0);
loader.autoload = false;
loader.load("logo.swf");
```

Loader.percentLoaded

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
loaderInstance.percentLoaded
```

描述

属性（只读）；一个数字，指明已加载内容的百分比。通常，使用该属性是为了以一种易于阅读的形式向用户展示进程。使用以下代码将数字舍入为最接近的整数：

```
Math.round(bytesLoaded/bytesTotal*100))
```

范例

以下范例创建一个 Loader 实例，然后创建一个带有进程处理函数的侦听器对象，该处理函数跟踪加载的百分比并将其发送到 “输出” 面板：

```
createClassObject(Loader, "loader", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
  // eventObj.target 是生成 change 事件的组件,
  // 即 Loader。
  trace("logo.swf is " + loader.percentLoaded + "% loaded."); // 跟踪加载进程
}
loader.addEventListener("complete", loadListener);
loader.content = "logo.swf";
```

Loader.progress

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(progress){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.progress = function(eventObject){
  ...
}
loaderInstance.addEventListener("progress", listenerObject)
```

描述

事件；在加载内容时向所有已注册的侦听器广播。当 autoload 参数或对 `Loader.load()` 的调用触发加载操作时，触发该事件。progress 事件并不会始终广播。`complete` 事件可以在没有调度任何 `progress` 事件的情况下广播。如果加载的内容是本地文件，尤其会出现这种情况。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 Loader 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，以下代码附加到 Loader 组件实例 `myLoaderComponent`，它将 “_level0.myLoaderComponent” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*loaderInstance*) 调度一个事件 （在本例中为 `progress`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下代码创建一个 Loader 实例，然后为 progress 事件创建一个带有事件处理函数的侦听器对象，该处理函数会将一条关于已加载内容的百分比的消息发送到 “输出” 面板：

```
createClassObject(mx.controls.Loader, "loader", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
  // eventObj.target 是生成 change 事件的组件，
  // 即 Loader。
  trace("logo.swf is " + loader.percentLoaded + "% loaded."); // 跟踪加载进程
}
loader.addEventListener("progress", loadListener);
loader.contentPath = "logo.swf";
```

Loader.scaleContent

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
loaderInstance.scaleContent
```

描述

属性；指明是内容进行缩放以适应加载器 (true) 还是加载器进行缩放以适应内容 (false)。默认值为 true。

范例

以下代码指示加载器调整自身的大小以匹配内容的大小：

```
loader.strechContent = false;
```

MediaController 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

MediaDisplay 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

MediaPlayback 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

Menu 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

MenuBar 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

NumericStepper 组件

NumericStepper 组件允许用户逐个通过一组经过排序的数字。该组件由显示在小上下箭头按钮旁边的数字组成。用户按下这些按钮时，数字将逐渐增大或减小。如果用户单击其中任一箭头按钮，数字将根据 stepSize 参数的值增大或减小，直到用户松开鼠标按钮或达到最大 / 最小值为止。

NumericStepper 只处理数值数据。此外，要显示两个以上的数值位置 （例如，数字 5246 或 1.34），您在创作时必须调整步进器的大小。

在应用程序中，可以启用或禁用步进器。在禁用状态下，步进器不接收鼠标或键盘输入。如果您单击或按 Tab 键切换到启用的步进器，则它将接收焦点并且其内部焦点会设置为文本框。当 NumericStepper 实例有焦点时，您可以使用以下按键来控制它：

按键	描述
向下箭头	值一次变化一个单位。
向左箭头	在文本框中将插入点移动到左侧。
向右箭头	在文本框中将插入点移动到右侧。
Shift + Tab	将焦点移到前一个对象。
Tab 键	将焦点移到下一个对象。
向上箭头	值一次变化一个单位。

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航”或第 93 页的 “FocusManager 类”。

每个步进器实例的实时预览会反映创作过程中属性检查器或 “组件检查器” 面板指明的值参数的值。但是，在实时预览中，鼠标或键盘与步进器按钮之间不能进行交互操作。

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

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 NumericStepper 组件

NumericStepper 可用于任何您想让用户选择数值的场合。例如，您可以在表格中使用 NumericStepper 组件来允许用户设置信用卡到期时间。另一个范例是可以使用 NumericStepper 组件来允许用户改变字体大小。

NumericStepper 参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 NumericStepper 组件设置的创作参数：

value 设置当前步进的值。默认值为 0。

minimum 设置步进的最小值。默认值为 0。

maximum 设置步进的最大值。默认值为 10。

stepSize 设置步进的变化单位。默认值为 1。

您可以编写动作脚本，通过利用 NumericStepper 的属性、方法和事件来控制它的这些选项以及其他选项。有关详细信息，请参阅 NumericStepper 类。

创建具有 NumericStepper 组件的应用程序

以下过程解释了如何在创作时将 NumericStepper 组件添加到应用程序。在该范例中，步进器允许用户从 0-5 星级的影片中选择一个影片，增量为半个星。

要创建具有 Button 组件的应用程序，请执行以下操作：

1 将 NumericStepper 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，输入实例名称 **starStepper**。

3 在属性检查器中，执行以下操作：

- 输入 0 作为最小参数。
- 输入 5 作为最大参数。
- 输入 .5 作为 stepSize 参数。
- 输入 0 作为参数的值。

4 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

```
movieRate = new Object();
movieRate.change = function (eventObject){
  starChart.value = eventObject.target.value;
}
starStepper.addEventListener("change", movieRate);
```

最后一行代码将 `change` 事件处理函数添加到 `starStepper` 实例。该处理函数会设置 `starChart` 影片剪辑，以显示由 `starStepper` 实例指明的星级。（要查看此代码的运行效果，您必须创建一个具有 `value` 属性 （该属性显示星级）的 `starChart` 影片剪辑。）

自定义 NumericStepper 组件

在创作过程中以及在运行时，您都可以在水平和垂直方向上改变 NumericStepper 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `setSize()` 方法 （请参阅 `UIObject.setSize()`）或任何适用的 NumericStepper 类的属性和方法。请参阅 NumericStepper 类。

调整 NumericStepper 组件的大小不会改变上下箭头按钮的大小。如果将步进器的大小调整为大于默认的高度，则该步进器的按钮将被固定在组件的顶部或底部。 步进器按钮会始终出现在文本框的右侧。

对 NumericStepper 组件使用样式

您可以设置样式属性以更改步进器实例的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

NumericStepper 组件支持下列光晕样式：

样式	描述
themeColor	组件的背景。这是唯一不继承样式值的颜色样式。可能的值包括“haloGreen”、“haloBlue”和“haloOrange”。
color	组件标签的文本。
disabledColor	禁用的文本颜色。
fontFamily	文本的字体名称。
fontSize	字体的磅值。
fontStyle	字体样式；“常规”或“斜体”。
fontWeight	字体粗细；“常规”或“粗体”。
textDecoration	文本修饰；“无”或“下划线”。
textAlign	文本对齐方式；“左”、“右”或“居中”。

对 NumericStepper 组件使用外观

NumericStepper 组件会使用外观来表现其视觉效果。要在创作时设置 NumericStepper 组件的外观，请在库中修改外观元件并将组件作为 SWC 重新导出。外观元件位于库中的 Flash UI Components 2/Themes/MMDefault/Stepper Elements/states 文件夹中。有关详细信息，请参阅第 33 页的 “关于设置组件外观”。

如果启用了步进器，当指针移到向上按钮和向下按钮的上方时，这些按钮会显示其悬停状态。这些按钮在被单击后时，会显示为按下状态。当松开鼠标后，这些按钮又会返回其悬停状态。如果鼠标按下时指针移离按钮，按钮会恢复到其原始状态。

如果禁用了步进器，不论用户进行什么交互操作，它都会显示其禁用状态。

NumericStepper 组件使用以下外观属性：

属性	描述
upArrowUp	向上箭头的弹起状态。默认值为 StepUpArrowUp。
upArrowDown	向上箭头的按下状态。默认值为 StepUpArrowDown。
upArrowOver	向上箭头的悬停状态。默认值为 StepUpArrowOver。
upArrowDisabled	向上箭头的禁用状态。默认值为 StepUpArrowDisabled。
downArrowUp	向下箭头的弹起状态。默认值为 StepDownArrowUp。
downArrowDown	向下箭头的按下状态。默认值为 StepDownArrowDown。
downArrowOver	向下箭头的悬停状态。默认值为 StepDownArrowOver。
downArrowDisabled	向下箭头的禁用状态。默认值为 StepDownArrowDisabled。

NumericStepper 类

继承 UIObject > UIComponent > NumericStepper

动作脚本类名称 mx.controls.NumericStepper

NumericStepper 类属性允许您添加最大或最小的步长值、每一步长的单位数量以及运行时该步长的当前值。

使用 “动作脚本”设置 NumericStepper 类的属性会覆盖在属性检查器或 “组件检查器”面板中设置的同名参数。

NumericStepper 组件会使用 FocusManager 覆盖默认的 Flash Player 焦点矩形，并画出一个带有圆角的自定义焦点矩形。有关详细信息，请参阅第 23 页的 “创建自定义焦点导航”。

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

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

注意：下面的代码返回未定义的：`trace(myNumericStepperInstance.version);`。

NumericStepper 类的方法摘要

继承 UIObject 类和 UIComponent 类的所有属性。

NumericStepper 类的属性摘要

属性	描述
`NumericStepper.maximum`	一个指明最大范围值的数字。
`NumericStepper.minimum`	一个指明最小范围值的数字。
`NumericStepper.nextValue`	一个指明下一个连续值的数字。该属性为只读。
`NumericStepper.previousValue`	一个指明前一个连续值的数字。该属性为只读。
`NumericStepper.stepSize`	一个显示每一步长的变化单位的数字。
`NumericStepper.value`	一个显示步进器当前值的数字。

继承 UIObject 类和 UIComponent 类的所有属性。

NumericStepper 类的事件摘要

事件	描述
`NumericStepper.change`	当步长的值更改时触发。

继承 UIObject 类和 UIComponent 类的所有属性。

NumericStepper.change

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

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

用法 2：

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
   ...
}
stepperInstance.addEventListener("change", listenerObject)
```

描述

事件；当步进器的值更改时向所有已注册的侦听器广播。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 NumericStepper 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下列代码附加到步进器 `myStepper`，它将 “_level0.myStepper” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*stepperInstance*) 调度一个事件 （在本例中为 `change`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

此范例是在时间轴上的某一帧上编写的，它会在一个名为 myNumericStepper 的步进器发生更改时向“输出”面板发送一条消息。代码的第一行创建一个名为 form 的侦听器对象。第二行代码为侦听器对象的 change 事件定义一个函数。该函数内部有一个 trace 动作，它使用自动传递到该函数的事件对象（在本例中是 eventObj）来生成消息。事件对象的 target 属性是生成该事件的组件，在这个范例中是 myNumericStepper。从事件对象的 target 属性中可以访问 NumericStepper.value 属性。最后一行从 myNumericStepper 调用 UIEventDispatcher.addEventListener() 方法，并把 change 事件和 form 侦听器对象作为参数传递给该方法，如下所示：

```
form = new Object();
form.change = function(eventObj){
  // eventObj.target 是生成 change 事件的组件,
  // 即, “数字步进器”。
  trace("Value changed to " + eventObj.target.value);
}
myNumericStepper.addEventListener("change", form);
```

NumericStepper.maximum

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

stepperInstance.maximum

描述

属性；步进器的最大范围值。此属性可能包含一个最多可达三位的十进制数字。默认值为 10。

范例

下面的范例设置步进器范围的最大值为 20：

```
myStepper.maximum = 20;
```

另请参见

NumericStepper.minimum

NumericStepper.minimum

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

stepperInstance.minimum

描述

属性；步进器的最小范围值。此属性可能包含一个最多可达三位的十进制数字。默认值为 0。

范例

下面的范例设置步进器范围的最小值为 100：

```
myStepper.minimum = 100;
```

另请参见

NumericStepper.maximum

NumericStepper.nextValue

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

stepperInstance.nextValue

描述

属性 （只读），下一个连续的值。此属性可能包含一个最多可达三位的十进制数字。

范例

以下范例将 stepSize 属性设置为 1、初始值设置为 4，这样就会使 previousValue 的值为 5：

```
myStepper.stepSize = 1;
myStepper.value = 4;
trace(myStepper.nextValue);
```

另请参见

NumericStepper.previousValue

NumericStepper.previousValue

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

stepperInstance.previousValue

描述

属性 （只读），前一个连续的值。此属性可能包含一个最多可达三位的十进制数字。

范例

以下范例将 stepSize 属性设置为 1、初始值设置为 4，这样就会使 previousValue 的值为 3：

```
myStepper.stepSize = 1;
myStepper.value = 4;
trace(myStepper.previousValue);
```

另请参见

NumericStepper.nextValue

NumericStepper.stepSize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

stepperInstance.stepSize

描述

属性；要从当前值改变的单位数量。默认值为 1，该值不能为 0。该属性可以包含一个带最多三位十进制的数字。

范例

下面的范例将当前的 value 设置为 2， stepSize 单位设置为 2，则 nextValue 的值为 4：

```
myStepper.value = 2;
myStepper.stepSize = 2;
trace(myStepper.nextValue);
```

NumericStepper.value

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

stepperInstance.value

描述

属性；步进器的文本区域显示的当前值。如果该值与 stepSize 属性中定义的步进器的范围和步进增量不符，将不被赋值。此属性可能包含一个最多可达三位的十进制数字。

范例

下面的范例将步进器的当前 value 设置为 10，并将该值发送到 “输出”面板：

```
myStepper.value = 10;
trace(myStepper.value);
```

PopUpManager 类

动作脚本类命名空间 mx.managers.PopUpManager

PopUpManager 类允许您创建模式的或非模式的重叠窗口。（模式窗口在处于活动状态时不允许与其他窗口进行交互操作。）您可以调用 `PopUpManager.createPopUp()` 来创建重叠窗口，也可以对窗口实例调用 `PopUpManager.deletePopUp()` 来破坏弹出窗口。

PopUpManager 类的方法摘要

事件	描述
`PopUpManager.createPopUp()`	创建弹出窗口。
`PopUpManager.deletePopUp()`	删除由对 `PopUpManager.createPopUp()` 的调用创建的弹出窗口。

PopUpManager.createPopUp()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
PopUpManager.createPopUp(parent, class, modal [, initobj, outsideEvents])
```

参数

parent　一个引用，引用弹出窗口所在的窗口。

class　一个引用，引用要创建的对象的类。

modal　一个布尔值，它表明该窗口是 (`true`) 否 (`false`) 是模式的。

initobj　一个包含初始化属性的对象。此参数是可选的。

outsideEvents　一个布尔值，指明在用户单击窗口以外的区域时是 (`true`) 否 (`false`) 触发事件。此参数是可选的。

返回

一个引用，引用所创建的窗口。

说明

方法；如果为模式窗口，对 `createPopUp()` 的调用会找到以父级开始的最顶层的父窗口，然后创建一个类的实例。如果是非模式的，对 `createPopUp()` 的调用会创建一个类的实例作为父窗口的子窗口。

示例

下面的代码在单击按钮时创建一个模式窗口：

```
lo = new Object();
lo.click = function(){
  mx.managers.PopUpManager.createPopUp(_root, mx.containers.Window, true);
}
button.addEventListener("click", lo);
```

PopUpManager.deletePopUp()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

```
windowInstance.deletePopUp();
```

参数

无。

返回

无。

说明

方法；删除一个弹出窗口并移除模式状态。当窗口被破坏时，被重叠的窗口负责调用 PopUpManager.deletePopUp()。

示例

下面的代码创建一个名为 win 的模式窗口，该窗口具有一个关闭按钮，当单击该关闭按钮时删除该窗口：

```
import mx.managers.PopUpManager
import mx.containers.Window
win = PopUpManager.createPopUp(_root, Window, true, {closeButton:true});
lo = new Object();
lo.click = function(){
  win.deletePopUp();
}
win.addEventListener("click", lo);
```

ProgressBar 组件

ProgressBar 组件在用户等待加载内容时，会显示加载进程。加载进程可以是确定的也可以是不确定的。确定的进程栏是一段时间内任务进程的线性表示，当要载入的内容量已知时使用。不确定的进程栏在不知道要加载的内容量时使用。您可以添加标签来显示加载内容的进程。

默认情况下，组件被设置为在第一帧导出。这意味着这些组件在第一帧呈现前被加载到应用程序中。如果要为应用程序创建预加载器，则需要在每个组件的“链接属性”对话框（该对话框的访问路径为：“库”面板选项 > “链接”）中取消选中“在第一帧导出”。但是对于 ProgressBar 应设置为“在第一帧导出”，因为 ProgressBar 必须在其他内容流进入 Flash Player 之前首先显示。

每个 ProgressBar 实例的实时预览都会反映在创作过程中对属性检查器或“组件检查器”面板中的参数所做的更改。实时预览中会反映以下参数：conversion、direction、label、labelPlacement、mode 和 source。

使用 ProgressBar 组件

进程栏允许您在内容加载过程中显示内容的进程。当用户与应用程序交互操作时，这是必需的反馈信息。

使用 ProgressBar 组件有几种模式；您可以使用模式参数来设置模式。最常用的模式是“事件”和“轮询”。这些模式使用 source 参数来指定一个加载进程，该进程发出 `progress` 和 `complete` 事件（事件模式）或公开 `getBytesLoaded` 和 `getsBytesTotal` 方法（轮询模式）。您也可以在手动模式下使用 ProgressBar 组件，方法是：手动设置 `maximum`、`minimum` 和 `indeterminate` 属性，并调用 `ProgressBar.setProgress()` 方法。

ProgressBar 参数

以下是您可以在属性检查器中或在“组件检查器”面板中为每个 ProgressBar 组件实例设置的创作参数：

mode 进度栏运行的模式此值可以是下列之一：事件、轮询或手动。默认值为事件。

source 一个要转换为对象的字符串，它表示源的实例名。

direction 进度栏填充的方向。该值可以在右侧或左侧，默认值为右侧。

label 指明加载进度的文本。该参数是一个字符串，其格式是“已加载 %2 的 %1 (%3%%)”；%1 是当前已加载字节数的占位符，%2 是加载的总字节数，%3 是当前加载的百分比的占位符。字符“%%”是字符“%”的占位符。如果某个 %2 的值未知，它将被替换为“??”。如果某个值未定义，则不显示标签。

labelPlacement 与进程栏相关的标签位置。此参数可以是下列值之一：顶部、底部、左侧、右侧、中间。默认值为底部。

conversion 一个数字，在显示标签字符串中的 %1 和 %2 的值之前，用这些值除以该数字。默认值为 1。

您可以编写“动作脚本”，通过利用其属性、方法和事件来控制 ProgressBar 组件的这些选项以及其他选项。有关详细信息，请参阅 ProgressBar 类。

创建具有 ProgressBar 组件的应用程序

以下过程解释了如何在创作时将 ProgressBar 组件添加到应用程序。在该范例中，进程栏用于事件模式。在事件模式中，加载内容必须发出 progress 事件和 complete 事件，进度栏使用这些事件来显示进度。 Loader 组件会发出这些事件。有关详细信息，请参阅第 131 页的 “Loader 组件”。

要创建带有事件模式 ProgressBar 组件的应用程序，请执行以下操作：

1 将 ProgressBar 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，执行以下操作：

- 输入实例名称 **pBar**。
- 对于 mode 参数，选择事件。

3 将 Loader 组件从 “组件” 面板拖到舞台上。

4 在属性检查器中，输入实例名称 **loader**。

5 在舞台上选择进度栏，并在属性检查器中为 source 参数输入 **loader**。

6 在时间轴上选择第一帧，打开 “动作” 面板，输入以下代码，该代码会将一个 JPEG 文件加载到 Loader 组件中：

```
loader.autoLoad = false;
loader.content = "http://imagecache2.allposters.com/images/86/
  017_PP0240.jpg";
pBar.source = loader;
// 直到调用加载方法，才开始加载。
loader.load();
```

在下面的范例中，在轮询模式下使用进度栏。在轮询模式中， ProgressBar 使用源对象的 getBytesLoaded 和 getBytesTotal 方法来显示其进度。

要在轮询模式中创建具有 ProgressBar 组件的应用程序，请执行以下操作：

1 将 ProgressBar 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，执行以下操作：

- 输入实例名称 **pBar**。
- 对于 mode 参数，选择轮询。
- 为 source 参数输入 **loader**。

3 在时间轴上选择第一帧，打开 “动作” 面板，然后输入以下代码，该代码会创建一个名为 loader 的 Sound 对象，并调用 loadSound() 方法，以便将一个声音加载到 Sound 对象中：

```
var loader:Object = new Sound();
loader.loadSound("http://soundamerica.com/sounds/sound_fx/A-E/air.wav",
  true);
```

在下面的范例中，在手动模式下使用进度栏。在手动模式中，您必须设置 maximum、minimum 和 indeterminate 属性，并使用 setProgress() 方法来显示进度。在手动模式中不需设置 source 属性。

要创建具有手动模式 ProgressBar 组件的应用程序，请执行以下操作：

1 将 ProgressBar 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，执行以下操作：

- 输入实例名称 **pBar**。
- 对于 mode 参数，选择手动。

3 在时间轴中选择帧 1，打开 “动作” 面板，然后输入以下代码，该代码会通过调用 setProgress() 方法，对每个文件下载过程手动更新进度栏：

```
for(var:Number i=1; i <= total; i++){
   // 插入代码以加载文件
   // 插入代码以加载文件
pBar.setProgress(i, total);
}
```

自定义 ProgressBar 组件

在创作过程中和运行时，您都可以在水平方向上改变 ProgressBar 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 UIObject.setSize()。

进程栏的左右两端以及跟踪图形是固定大小的。当您重新调整进程栏的大小时，进程栏的中间部分会重新调整大小，以便能在它们之间放下。如果进程栏太小，则可能会无法正确呈现。

对 ProgressBar 组件使用样式

您可以设置样式属性来改变进程栏实例的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

ProgressBar 组件支持下列光晕样式：

样式	描述
themeColor	组件的背景。这是唯一不继承样式值的颜色样式。可能的值包括 “haloGreen”、“haloBlue” 和 “haloOrange”。
color	组件标签的文本。
disabledColor	禁用的文本颜色。
fontFamily	文本的字体名称。
fontSize	字体的磅值。
fontStyle	字体样式；“常规” 或 “斜体”。
fontWeight	字体粗细；“常规” 或 “粗体”。
textDecoration	文本修饰；“无” 或 “下划线”。

对 ProgressBar 组件使用外观

进程栏组件使用以下影片剪辑元件来显示其状态：TrackMiddle、TrackLeftCap、TrackRightCap 和 BarMiddle、BarLeftCap、BarRightCap 和 IndBar。IndBar 元件用于不确定的进程栏。要在创作过程中设计 ProgressBar 组件的外观，请在库中修改元件并将组件作为 SWC 重新导出。这些元件位于 HaloTheme.fla 文件或 SampleTheme.fla 文件中，这些文件位于库中的 Flash UI Components 2/Themes/MMDefault/ProgressBar Elements 文件夹下。有关详细信息，请参阅第 33 页的 “关于设置组件外观”。

如果您使用 `UIObject.createClassObject()` 方法动态（在运行时）创建 ProgressBar 组件实例，则也可以动态设计其外观。要在运行时设计组件的外观，请设置传递给 `createClassObject()` 方法的 *`initObject`* 参数的外观属性。这些外观属性设置用作进度栏状态的元件的名称。

ProgressBar 组件使用以下外观属性：

属性	描述
`progTrackMiddleName`	轨道的可扩展的中部。默认值为 `ProgTrackMiddle`。
`progTrackLeftName`	固定大小的左端。默认值为 `ProgTrackLeft`。
`progTrackRightName`	固定大小的右端。默认值为 `ProgTrackRight`。
`progBarMiddleName`	可扩展的中间栏图形。默认值为 `ProgBarMiddle`。
`progBarLeftName`	固定大小的左栏端。默认值为 `ProgBarLeft`。
`progBarRightName`	固定大小的右栏端。默认值为 `ProgBarRight`。
`progIndBarName`	不确定的栏图形。默认值为 `ProgIndBar`。

ProgressBar 类

继承 UIObject > ProgressBar

动作脚本类命名空间 mx.controls.ProgressBar

使用 “动作脚本” 设置 ProgressBar 类的属性会覆盖在属性检查器或 “组件检查器” 面板中设置的同名参数。

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

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

注意：下面的代码返回未定义的：`trace(myProgressBarInstance.version);`。

ProgressBar 类的方法摘要

方法	描述
`ProgressBar.setProgress()`	在手动模式中设置栏的进程。

继承 UIObject 类的所有方法。

ProgressBar 类的属性摘要

属性	描述
ProgressBar.conversion	一个数字，用于转换当前所加载字节的值和所加载字节总值。
ProgressBar.direction	进程栏填充的方向。
ProgressBar.indeterminate	指明源的总字节数为未知。
ProgressBar.label	进程栏随附的文本。
ProgressBar.labelPlacement	与进程栏相关的标签位置。
ProgressBar.maximum	手动模式中进程的最大值。
ProgressBar.minimum	手动模式中进程的最小值。
ProgressBar.mode	进程栏加载内容的模式。
ProgressBar.percentComplete	指明加载百分比的数字。
ProgressBar.source	要加载的内容，其进程由进程栏监视。
ProgressBar.value	指明已建立的进程的数量。该属性为只读。

继承 UIObject 类的所有属性。

ProgressBar 类的事件摘要

事件	描述
ProgressBar.complete	加载完成时触发。
ProgressBar.progress	在事件模式或轮询模式中加载内容时触发。

继承 UIObject 类的所有事件。

ProgressBar.complete

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(complete){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
  ...
}
pBar.addEventListener("complete", listenerObject)
```

事件对象

除了标准的事件对象属性之外，还为 ProgressBar.complete 事件定义了另外两个属性：current （加载的值等于总计）和 total （总值）。

描述

事件；内容加载结束时，向所有已注册的侦听器广播。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 ProgressBar 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下列代码附加到实例 pBar，它将 “_level0.pBar” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*pBar*) 调度一个事件 （在本例中为 complete），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

本范例创建一个带有 complete 回调函数的 form 侦听器对象，该回调函数将一条消息发送到 “输出” 面板，其中有 pBar 实例的值，如下所示：

```
form.complete = function(eventObj){
  // eventObj.target 是生成 change 事件的组件，
  // 即 Progress Bar。
  trace("Value changed to " + eventObj.target.value);
}
pBar.addEventListener("complete", form);
```

另请参见

UIEventDispatcher.addEventListener()

ProgressBar.conversion

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.conversion

描述

属性；一个数字，设置进入值的转换值。它除当前值和总值，将它们向下取整，然后在 label 属性中显示转换后的值。 默认值为 1。

范例

以下代码显示加载进程的值 （字节）：

```
pBar.conversion = 1024;
```

ProgressBar.direction

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
pBarInstance.direction
```

描述

属性；指明进程栏的填充方向。默认值为 "right"。

范例

以下代码使进程栏从右向左填充：

```
pBar.direction = "left";
```

ProgressBar.indeterminate

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
pBarInstance.indeterminate
```

描述

属性；一个布尔值，指明进度栏是填充条纹图案并且加载源的大小未知 (true)，还是实心填充且加载源大小已知 (false)。

范例

以下代码创建一个确定的进度栏，该进度栏为实心填充、从左向右移动：

```
pBar.direction = "right";
pBar.indeterminate = false;
```

ProgressBar.label

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.label

描述

属性；指明加载进程的文本。该属性是一个字符串，其格式为 “已加载 %2 中的 %1（%3%%）”；%1 是当前所加载字节数的占位符，%2 是总加载字节数的占位符，%3 是当前加载内容百分比的占位符。字符 “%%” 是字符 “%” 的占位符。如果某个 %2 的值未知，它将被替换为 “??”。如果某个值未定义，则不显示标签。默认值为 "LOADING %3%%"

范例

以下代码将进程栏旁边显示的文字设置为 “已加载 4 个文件” 格式：

```
pBar.label = "已加载 %1 个文件";
```

另请参见

ProgressBar.labelPlacement

ProgressBar.labelPlacement

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.labelPlacement

描述

属性；设置标签相对于进程栏的位置。可能的值有 "left"、"right"、"top"、"bottom" 和 "center"。

范例

以下代码设置标签在进程栏上方显示：

```
pBar.label = "%1 out of %2 loaded (%3%%)";
pBar.labelPlacement = "top";
```

另请参见

ProgressBar.label

ProgressBar.maximum

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.maximum

描述

属性；ProgressBar.mode 属性设置为 "manual" 时进度栏的最大值。

范例

以下代码将最大值属性设置为要加载的 Flash 应用程序的总帧数：

```
pBar.maximum = _totalframes;
```

另请参见

ProgressBar.minimum, ProgressBar.mode

ProgressBar.minimum

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.minimum

描述

属性；ProgressBar.mode 属性设置为 "manual" 时进度栏的最小进度值。

范例

以下代码设置进程栏的最小值：

```
pBar.minimum = 0;
```

另请参见

ProgressBar.maximum, ProgressBar.mode

ProgressBar.mode

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.mode

描述

属性；进程栏加载内容的模式。此值可以是下列之一：`"event"`、`"polled"` 或 `"manual"`。最常用的模式为 `"event"` 和 `"polled"`。这些模式使用 `source` 参数指定加载进程是发出 `progress` 和 `complete` 事件，像 Loader 组件 （事件模式），还是公开 `getBytesLoaded` 和 `getsBytesTotal` 方法，像 MovieClip 对象 （轮询模式）。您也可以在手动模式下使用 ProgressBar 组件，方法是：手动设置 `maximum`、`minimum` 和 `indeterminate` 属性，并调用 `ProgressBar.setProgress()` 方法。

在事件模式中， Loader 对象应该用作源。在轮询模式中，任何公开 `getBytesLoaded()` 和 `getBytesTotal()` 方法的对象都可用作源。（包括自定义对象或 `_root` 对象）

范例

以下代码将进度栏设置为事件模式：

```
pBar.mode = "event";
```

ProgressBar.percentComplete

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
pBarInstance.percentComplete
```

描述

属性 （只读）；返回进程完成的百分比。该值已向下取整。以下是用于计算百分比的公式：

```
100* （值 - 最小值） / （最大值 - 最小值）
```

范例

以下代码将 `percentComplete` 属性的值发送到 “输出” 面板：

```
trace("percent complete = " + pBar.percentComplete);
```

ProgressBar.progress

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(progress){
   ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.progress = function(eventObject){
  ...
}
pBarInstance.addEventListener("progress", listenerObject)
```

事件对象

除了标准的事件对象属性外，还为 ProgressBar.progress 事件定义了另外两个属性：current（加载的值等于总计）和 total（总值）。

描述

事件；当进程栏的值更改时向所有已注册的侦听器广播。事件仅在 ProgressBar.mode 设置为 "manual" 或 "polled" 时广播。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 ProgressBar 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下列代码附加到实例 myPBar，它将 "_level0.myPBar" 发送到 "输出" 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*pBarInstance*) 调度一个事件（在本例中为 progress），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 "事件对象"。

范例

本范例创建一个侦听器对象 form，并且在其上定义一个 progress 事件处理函数。在最后一行代码中，form 侦听器注册到 pBar 实例。progress 事件触发时，pBar 向 form 侦听器广播该事件，该侦听器调用 progress 回调函数，如下所示：

```
var form:Object = new Object();
form.progress = function(eventObj){
  // eventObj.target 是生成 change 事件的组件，
  // 即 Progress Bar。
  trace("Value changed to " + eventObj.target.value);
}
pBar.addEventListener("progress", form);
```

另请参见

UIEventDispatcher.addEventListener()

ProgressBar.setProgress()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
pBarInstance.setProgress(completed, total)
```

参数

completed 一个数字，指明已完成的进度量。您可以使用 ProgressBar.label 和 ProgressBar.conversion 属性以百分比形式或所选择的任何单位来显示该数字，这具体取决于进度栏的源。

total 一个数字，指明要达到百分之百必须完成的总进度。

返回

一个数字，指明已完成的进度。

描述

方法；ProgressBar.mode 属性设置为 "manual" 时，设置栏的状态以反映完成的进度量。您可以调用该方法，使栏除了反映加载的状态外还反映进度的状态。参数 completed 被分配给 value 值属性，参数 total 被分配给 maximum 属性。 minimum 属性不变。

范例

以下代码基于 Flash 应用程序时间轴的进度调用 setProgress() 方法：

```
pBar.setProgress(_currentFrame, _totalFrames);
```

ProgressBar.source

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
pBarInstance.source
```

描述

属性；对要加载的实例的引用，该实例的加载进程将会显示。加载内容应该发出一个 progress 事件，从其中可以检索当前值和总值。只是在 ProgressBar.mode 设置为 "event" 或 "polled" 时才使用该属性。默认值未定义。

ProgressBar 可与应用程序（包括 _root）内的内容一起使用。

范例

本范例设置 pBar 实例，以显示实例名为 loader 的加载器组件的加载进度。

```
pBar.source = loader;
```

另请参见

ProgressBar.mode

ProgressBar.value

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

pBarInstance.value

描述

属性 （只读）；指明已完成的进度。该属性为介于 ProgressBar.minimum 和 ProgressBar.maximum 之间的一个数字。默认值为 0。

RadioButton 组件

使用 RadioButton 组件可以强制用户只能选择一组选项中的一项。 RadioButton 组件必须用于至少有两个 RadioButton 实例的组。在任何给定的时刻，都只有一个组成员被选中。选择组中的一个单选按钮将取消选择组内当前选定的单选按钮。您可以设置 groupName 参数，以指明单选按钮属于哪个组。

可以启用或禁用单选按钮。用户按 Tab 键切换到单选按钮组时，只有选中的单选按钮会接收焦点。用户可以通过按箭头键来改变组内的焦点。在禁用状态下，单选按钮不接收鼠标或键盘输入。

如果您单击或按 Tab 键切换到 RadioButton 组件组，它就会接收焦点。当 RadioButton 组有焦点时，您可以使用下列按键来控制它：

按键	描述
向上箭头键 / 向右箭头键	所选项会移至单选按钮组内的前一个单选按钮。
向下箭头键 / 向左箭头键	选择将移到单选按钮组的下一个单选按钮。
Tab 键	将焦点从单选按钮组移动到下一个组件。

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航”或第 93 页的 “FocusManager 类”。

每个 RadioButton 实例在舞台上的实时预览会反映在创作过程中对属性检查器或 “组件检查器” 面板中的参数所做的更改。但是，实时预览中不会显示互斥的所选项。如果将同组的两个单选按钮的 selected 参数设置为 true，它们都会显示为选中状态，尽管在运行时将只显示最后创建的实例。有关详细信息，请参阅第 165 页的 “RadioButton 参数”。

将 RadioButton 组件添加到应用程序时，您可以使用 “辅助功能” 面板，以便让屏幕读取器能够访问到该组件。 首先，您必须添加以下代码行来启用辅助功能 ：

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 RadioButton 组件

单选按钮是任何表单或 Web 应用程序中的一个基础部分。如果您需要让用户从一组选项中做出一个选择，可以使用单选按钮。例如，在表单上询问客户要使用哪种信用卡付款时，您就可以使用单选按钮。

RadioButton 参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 RadioButton 组件设置的创作参数 ：

label 设置按钮上的文本值，默认值是 “单选按钮”。

data 是与单选按钮相关的值。没有默认值。

groupName 是单选按钮的组名称。默认值为 radioGroup。

selected 将单选按钮的初始值设置为被选中 (true) 或取消选中 (false)。被选中的单选按钮中会显示一个圆点。一个组内只有一个单选按钮可以有被选中的值 true。如果组内有多个单选按钮被设置为 true，则会选中最后实例化的单选按钮。默认值为 false。

labelPlacement 确定按钮上标签文本的方向。该参数可以是下列四个值之一：left、right、top 或 bottom，默认值是 right。有关详细信息，请参阅 `RadioButton.labelPlacement`。

您可以编写 “动作脚本”，通过利用 RadioButton 类的方法、属性和事件来设置 RadioButton 实例的其他选项。有关详细信息，请参阅 RadioButton 类。

创建具有 RadioButton 组件的应用程序

以下过程解释了如何在创作时将 RadioButton 组件添加到应用程序。在该范例中，单选按钮用于显示是非问题，如 “您是一个 Flashist 吗？” 单选按钮组的数据和实例名称 `Verdict` 一起显示在 TextArea 组件中。

要创建具有 RadioButton 组件的应用程序，请执行以下操作：

1. 将两个 RadioButton 组件从 “组件” 面板拖到舞台上。
2. 选择一个单选按钮，然后在 “组件检查器” 中执行以下操作 ：
   - 为 label 参数输入 Yes。
   - 为 data 参数输入 Flashist。
3. 选择另一个单选按钮，然后在 “组件检查器” 中执行以下操作 ：
   - 为 label 参数输入 No。
   - 为 data 参数输入 Anti-Flashist。

4 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

```
flashistListener = new Object();
flashistListener.click = function (evt){
    theVerdict.text = evt.target.selection.data
}
radioGroup.addEventListener("click", flashistListener);
```

代码最后一行将一个 `click` 事件处理函数添加到 `radioGroup` 单选按钮组。处理函数会将 TextArea 组件实例 `theVerdict` 的 `text` 属性设置为 `radioGroup` 单选按钮组中被选定的单选按钮的 `data` 属性的值。有关详细信息，请参阅 `RadioButton.click`。

自定义 RadioButton 组件

在创作过程中和运行时，您都可以在水平和垂直方向上改变 RadioButton 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `setSize()` 方法 （请参阅第 229 页的 “UIObject.setSize()”）。

RadioButton 组件的边框是不可见的，它同时也指定了组件的点击区。 如果您增加组件的大小，也就增加了点击区的大小。

如果组件边框太小而无法容纳组件标签，标签将被裁剪以适合边框。

对 RadioButton 组件使用样式

您可以设置样式属性以更改 RadioButton 的外观。如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

RadioButton 组件使用下列 “光晕” 样式：

样式	描述
`themeColor`	组件的背景。这是唯一不继承样式值的颜色样式。可能的值包括 “haloGreen”、“haloBlue” 和 “haloOrange”。
`color`	组件标签的文本。
`disabledColor`	禁用的文本颜色。
`fontFamily`	文本的字体名称。
`fontSize`	字体的磅值。
`fontStyle`	字体样式；“常规” 或 “斜体”。
`fontWeight`	字体粗细；“常规” 或 “粗体”。

对 RadioButton 组件使用外观

通过修改库中的组件元件，可以在创作时设置 RadioButton 组件的外观。RadioButton 组件的外观位于 HaloTheme.fla 或 SampleTheme.fla 中，这些文件位于库中的下列文件夹中：Flash UI Components 2/Themes/MMDefault/RadioButton Assets/States。请参阅第 33 页的 “关于设置组件外观”。

如果单选按钮已启用但未被选中，当用户将指针移到它上方时，它会显示其滑过状态。当用户单击未被选中的单选按钮时，该单选按钮将接收输入焦点并显示其 “false” 按下状态。当用户松开鼠标时，单选按钮显示其 “true” 状态，组内先前被选中的单选按钮显示其 “false” 状态。如果用户在按下鼠标时将指针移离单选按钮，单选按钮的外观会恢复到其 “false” 状态并保留输入焦点。

如果单选按钮或单选按钮组被禁用，则不论用户进行什么交互操作，它都会显示禁用状态。

如果您使用 `UIObject.createClassObject()` 方法动态创建 RadioButton 组件实例，那么，您也可以动态设置组件外观。要动态设置 RadioButton 组件的外观，请将外观属性传递给 `UIObject.createClassObject()` 方法。有关详细信息，请参阅第 33 页的 “关于设置组件外观”。外观属性指明使用哪个元件显示组件。

RadioButton 组件使用以下外观属性：

名称	描述
`falseUpIcon`	未选中状态。默认值为 `radioButtonFalseUp`。
`falseDownIcon`	按下且未选中的状态。默认值为 `radioButtonFalseDown`。
`falseOverIcon`	悬停而未选中的状态。默认值为 `radioButtonFalseOver`。
`falseDisabledIcon`	禁用而未选中的状态。默认值为 `radioButtonFalseDisabled`。
`trueUpIcon`	选中状态。默认值为 `radioButtonTrueUp`。
`trueDisabledIcon`	禁用的选中状态。默认值为 `radioButtonTrueDisabled`。

RadioButton 类

继承 UIObject > UIComponent > SimpleButton > Button > RadioButton

动作脚本包名称 mx.controls.RadioButton

RadioButton 类的属性允许您在运行时创建文字标签并确定它相对于单选按钮的位置。您还可以为单选按钮指定数据值，将单选按钮分组，并根据数据值或实例名称进行选择。

使用 “动作脚本” 设置 RadioButton 类的属性将覆盖在属性检查器或 “组件检查器” 面板中设置的同名参数。

RadioButton 组件使用 FocusManager 来覆盖默认的 Flash Player 焦点矩形并绘制一个带圆角的自定义焦点矩形。有关创建焦点导航的信息，请参阅第 23 页的 “创建自定义焦点导航”。

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

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

注意：下面的代码返回未定义的：`trace(myRadioButtonInstance.version);`。

RadioButton 类的方法摘要

继承 UIObject 类、 UIComponent 类、 SimpleButton 和 Button 类的所有方法。

RadioButton 类的属性摘要

属性	描述
RadioButton.data	与单选按钮实例相关的值。
RadioButton.groupName	单选按钮组的组名或单选按钮实例。
RadioButton.label	单选按钮旁边显示的文本。
RadioButton.labelPlacement	标签文本相对于单选按钮的方向。
RadioButton.selected	将单选按钮实例的状态设置为选中，并取消选中先前选中的单选按钮。
RadioButton.selectedData	选择具有指定数据值的单选按钮组中的单选按钮。
RadioButton.selection	对单选按钮组中当前选中的单选按钮的引用。

继承 UIObject 类、 UIComponent 类、 SimpleButton 和 Button 类的所有属性。

RadioButton 类的事件摘要

事件	描述
RadioButton.click	在按钮实例上按下鼠标时触发。

继承 UIObject 类、 UIComponent 类、 SimpleButton 和 Button 类 的所有事件。

RadioButton.click

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

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

用法 2：

```
listenerObject = new Object();
listenerObject.click = function(eventObject){
   ...
}
radioButtonGroup.addEventListener("click", listenerObject)
```

描述

事件；在单选按钮上单击鼠标（按下然后松开）或使用箭头键选中单选按钮时，向所有已注册的侦听器广播。当单选按钮组具有焦点，但组内没有单选按钮被选中时，如果按空格键或箭头键，该事件也会广播。

第一个用法范例使用 `on()` 处理函数，必须直接附加到 RadioButton 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，附加到单选按钮 `myRadioButton` 的以下代码将 “_level0.myRadioButton” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*radioButtonInstance*) 调度一个事件（在本例中为 `click`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象有一组属性，这些属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

本范例是在时间轴的某帧上编写的，当 `radioGroup` 中的某个单选按钮被单击时向 “输出” 面板发送一条消息。代码的第一行创建一个名为 `form` 的侦听器对象。第二行代码为侦听器对象上的 `click` 事件定义一个函数。该函数内部有一个 `trace` 动作，它使用自动传递到该函数的事件对象（在本例中是 `eventObj`）来生成消息。事件对象的 `target` 属性是生成该事件的组件。您可以从 `target` 属性访问实例属性（本例中访问 `RadioButton.selection` 属性）。最后一行从 `radioGroup` 调用 `UIEventDispatcher.addEventListener()` 方法并将 `click` 事件和 `form` 侦听器对象作为参数传递给它，如下所示：

```
form = new Object();
form.click = function(eventObj){
  trace("The selected radio instance is " + eventObj.target.selection);
}
radioGroup.addEventListener("click", form);
```

下列代码还会在单击 `radioButtonInstance` 时向 “输出” 面板发送一条消息。`on()` 处理函数必须直接附加到 `radioButtonInstance`，如下所示：

```
on(click){
  trace("radio button component was clicked");
}
```

RadioButton.data

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

radioButtonInstance.data

描述

方法；指定与单选按钮实例关联的数据。设置该属性覆盖创作过程中在属性检查器或 “组件检查器”面板中设置的数据参数值。 `data` 属性可以为任何数据类型。

范例

以下范例将数据值 `"#FF00FF"` 指定给 `radioOne` 单选按钮实例：

```
radioOne.data = "#FF00FF";
```

RadioButton.groupName

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
radioButtonInstance.groupName
radioButtonGroup.groupName
```

描述

属性；为单选按钮实例或单选按钮组设置组名。您可以使用该属性来获取或设置单选按钮实例的或单选按钮组的组名。调用此方法会覆盖在创作过程中设置的组件名参数值。默认值为 `"radioGroup"`。

范例

以下范例将单选按钮实例的组名设置为 “`colorChoice`”，然后将组名更改为 “`sizeChoice`”。要测试此范例，请将单选按钮放在舞台上，实例名为 `myRadioButton`，然后在帧 1 中输入下列代码：

```
myRadioButton.groupName = "colorChoice";
trace(myRadioButton.groupName);
colorChoice.groupName = "sizeChoice";
trace(colorChoice.groupName);
```

RadioButton.label

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
radioButtonInstance.label
```

描述

属性；为单选按钮指定文字标签。默认情况下，标签出现在单选按钮的右侧。调用此方法会覆盖在创作中指定的 label 参数。如果标签文字太长无法匹配组件的边界框，则缩短文字。

范例

下面的范例设置实例 radioButton 的标签属性：

```
radioButton.label = "Remove from list";
```

RadioButton.labelPlacement

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
radioButtonInstance.labelPlacement
radioButtonGroup.labelPlacement
```

描述

属性；一个字符串，它指明该标签相对于某个单选按钮的位置。您可以为某个单独的实例或为某个单选按钮组设置此属性。如果您为一个组设置该属性，标签就会放置在组中各个单选按钮的相应的位置。

以下是四个可能的值：

- "right" 单选按钮固定于边界区域的左上角。标签设置在单选按钮的右边。
- "left" 单选按钮固定于边界区域的右上角。标签设置在单选按钮的左边。
- "bottom" 标签放置在单选按钮的下方。单选按钮和标签组合在水平和垂直方向上居中。如果单选按钮的边界框不够大，标签就会被裁剪。
- "top" 标签放置在单选按钮的上方。单选按钮和标签组合在水平和垂直方向上居中。如果单选按钮的边界框不够大，标签就会被裁剪。

范例

以下代码将标签放置在 radioGroup 中的每个单选按钮的左边：

```
radioGroup.labelPlacement = "left";
```

RadioButton.selected

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
radioButtonInstance.selected
radioButtonGroup.selected
```

描述

属性；一个布尔值，它将单选按钮的状态设置为被选中 (true)，并取消选中先前被选中的单选按钮，或者，它将单选按钮的状态设置为取消选中 (false)。

范例

代码的第一行将 mcButton 实例的状态设置为 true。代码的第二行返回所选属性的值，如下所示：

```
mcButton.selected = true;
trace(mcButton.selected);
```

RadioButton.selectedData

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

radioButtonGroup.selectedData

描述

属性；选中具有指定数据值的单选按钮，并取消选中先前被选中的单选按钮。如果选中的实例没有指定 data 属性，则会选择并返回所选实例的标签值。 selectedData 属性可以是任何数据类型。

范例

以下范例从单选按钮组 colorGroup 中选择具有 "#FF00FF" 值的单选按钮，并把该值发送到“输出”面板：

```
colorGroup.selectedData = "#FF00FF";
trace(colorGroup.selectedData);
```

RadioButton.selection

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

radioButtonInstance.selection
radioButtonGroup.selection

描述

属性；在您获取或设置该属性时行为会有所不同。如果您获取该属性，它会返回单选按钮组中当前被选单选按钮的对象引用。如果您设置该属性，它会选择单选按钮组中指定的单选按钮 （传递为对象引用），并取消选中先前被选中的单选按钮。

范例

以下范例选择实例名为 color1 的单选按钮，并将其实例名发送到 “输出”面板：

```
colorGroup.selection = color1;
trace(colorGroup.selection._name)
```

RDBMSResolver 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

RemoteProcedureCall 接口

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

Screen 类

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

ScrollPane 组件

“滚动窗格” 组件在一个可滚动区域中显示影片剪辑、JPEG 文件和 SWF 文件。您可以让滚动条能够在一个有限的区域中显示图像。您可以显示从本地位置或 Internet 加载的内容。在创作过程中以及在运行时，您都可以使用 “动作脚本” 来设置滚动窗格的内容。

一旦滚动窗格具有焦点，如果滚动窗格的内容具有有效的制表位，那些标记将接收焦点。在内容中的最后一个制表位之后，焦点将切换到下一个组件。滚动窗格中的垂直和水平滚动条从不接收焦点。

如果用户单击或切换到 ScrollPane 实例，该实例将接收焦点。当 ScrollPane 实例具有焦点时，您可以使用下列按键来控制它 ：

按键	描述
向下箭头	内容向上移动一垂直滚动行。
End 键	内容移动到滚动窗格的底部。
向左箭头	内容向右移动一水平滚动行。
Home 键	内容移动到滚动窗格的顶部。
Page Down 键	内容向上移动一垂直滚动页。
Page Up 键	内容向下移动一垂直滚动页。
向右箭头	内容向左移动一水平滚动行。
向上箭头	内容向下移动一垂直滚动行。

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航” 或第 93 页的 “FocusManager 类” 。

每个 ScrollPane 实例的实时预览反映了创作时在属性检查器或 “组件检查器” 面板中对参数所做的更改。

使用 ScrollPane 组件

如果某些内容对于它们要加载到其中的区域而言过大，您可以使用滚动窗格来显示这些内容。例如，如果您有一幅大图像，而在应用程序中只有很小的空间来显示它，则可以将其加载到滚动窗格中。

您可以通过将 `scrollDrag` 参数设为 `true` 来允许用户在窗格中拖动内容；一个手形光标会出现在内容上。与其他大多数组件不同的是，当按下鼠标按键时，事件开始广播，一直到松开按键才结束。如果滚动窗格的内容具有有效的 Tab 键停靠位，您必须将 `scrollDrag` 设为 `false`，否则每次鼠标与内容进行的交互操作都将会调用滚动拖动。

ScrollPane 参数

下面是您可以在属性检查器或 “组件检查器” 面板中为每个 ScrollPane 组件实例设置的创作参数：

contentPath 指明要加载到滚动窗格中的内容。该值可以是本地 SWF 或 JPEG 文件的相对路径，或 Internet 上的文件的相对或绝对路径。它也可以是设置为 “为动作脚本导出” 的库中的影片剪辑元件的链接标识符。

hLineScrollSize 指明每次按下箭头按钮时水平滚动条移动多少个单位。默认值为 5。

hPageScrollSize 指明每次按下轨道时水平滚动条移动多少个单位。默认值为 20。

hScrollPolicy 显示水平滚动条。该值可以为 “on”、“off” 或 “auto”。默认值为 “auto”。

scrollDrag 是一个布尔值，它允许 (true) 或不允许 (false) 用户在滚动窗格中滚动内容。默认值为 false。

vLineScrollSize 指明每次按下箭头按钮时垂直滚动条移动多少个单位。默认值为 5。

vPageScrollSize 指明每次按下轨道时垂直滚动条移动多少个单位。默认值为 20。

vScrollPolicy 显示垂直滚动条。该值可以为 “on”、“off” 或 “auto”。默认值为 “auto”。

您可以编写 “动作脚本”，通过利用其属性、方法和事件来控制 ScrollPane 组件的这些选项以及其他选项。有关详细信息，请参阅 ScrollPane 类。

创建具有 ScrollPane 组件的应用程序

以下过程解释了如何在创作时将 ScrollPane 组件添加到应用程序。在此范例中，滚动窗格加载一个包含徽标的 SWF 文件。

要创建具有 ScrollPane 组件的应用程序，请执行以下操作：

1 将 ScrollPane 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，输入实例名称 **myScrollPane**。

3 在属性检查器中，为 contentPath 参数输入 **logo.swf**。

4 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

```
scrollListener = new Object();
scrollListener.scroll = function (evt){
  txtPosition.text = myScrollPane.vPosition;
}
myScrollPane.addEventListener("scroll", scrollListener);

completeListener = new Object;
completeListener.complete = function() {
  trace("logo.swf has completed loading.");
}
myScrollPane.addEventListener("complete", completeListener);
```

第一块代码是 `myScrollPane` 实例上的一个 `scroll` 事件处理函数，它显示一个名为 `txtPosition` 的 TextField 实例（该实例已经置于舞台上）中的 `vPosition` 属性的值。第二块代码为 `complete` 事件创建一个事件处理函数，它向 “输出” 面板发送一条消息。

自定义 ScrollPane 组件

在创作过程中和运行时，您都可以在水平和垂直方向上改变 ScrollPane 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `setSize()` 方法（请参阅 `UIObject.setSize()`）或任何适用的 ScrollPane 类的属性和方法。请参阅 ScrollPane 类。如果 ScrollPane 太小，则内容可能无法正确显示。

ScrollPane 将其内容的注册点设置在窗格的左上角。

当关闭水平滚动条时，垂直滚动条沿滚动窗格右侧从上到下显示。当关闭垂直滚动条时，水平滚动条沿滚动窗格底部从左到右显示。也可以同时关闭两个滚动条。

当重新调整滚动窗格的大小时，按钮会保持相同的大小，而滚动轨道和滑块会扩展或收缩，其点击区也会重新调整大小。

对 ScrollPane 组件使用样式

ScrollPane 不支持样式，但它使用的滚动条支持样式。有关详细信息，请参阅第 175 页的 “对 ScrollPane 组件使用样式”。

对 ScrollPane 组件使用外观

ScrollPane 组件本身没有任何外观，但它使用的滚动条有外观。有关详细信息，请参阅第 175 页的 “对 ScrollPane 组件使用外观”。

ScrollPane 类

继承 UIObject > UIComponent > View > ScrollView > ScrollPane

动作脚本类命名空间 mx.containers.ScrollPane

ScrollPane 类的属性允许您设置内容、监视加载进程以及在运行时调整滚动量。

使用“动作脚本”设置 ScrollPane 类的属性会覆盖在属性检查器或“组件检查器”面板中设置的同名参数。

您可以通过将 `scrollDrag` 属性设为 `true` 来允许用户在窗格中拖动内容；一个手形光标会出现在内容上。与其他大多数组件不同的是，当按下鼠标按键时，事件开始广播，一直到松开按键才结束。如果滚动窗格的内容具有有效的 Tab 键停靠位，您必须将 `scrollDrag` 设为 `false`，否则每次鼠标与内容进行的交互操作都将会调用滚动拖动。

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

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

注意：下面的代码返回未定义的：`trace(myScrollPaneInstance.version);`.

ScrollPane 类的方法摘要

方法	描述
`ScrollPane.getBytesLoaded()`	返回已加载的内容的字节数。
`ScrollPane.getBytesTotal()`	返回要加载的内容的总字节数。
`ScrollPane.refreshPane()`	重新加载滚动窗格的内容。

继承 UIObject 类和 UIComponent 类中的所有方法。

ScrollPane 类的属性摘要

方法	描述
`ScrollPane.content`	对加载到滚动窗格中的内容的引用。
`ScrollPane.contentPath`	加载到滚动窗格中的 SWF 或 JPEG 文件的绝对 URL 或相对 URL。
`ScrollPane.hLineScrollSize`	箭头按钮被按下时要水平滚动的内容数量。
`ScrollPane.hPageScrollSize`	轨道被按下时要水平滚动的内容数量。
`ScrollPane.hPosition`	滚动窗格的水平像素位置。
`ScrollPane.hScrollPolicy`	水平滚动条的状态。它可以为始终打开 (`"on"`)、始终关闭 (`"off"`)，或者只是在需要时才打开 (`"auto"`)。默认值为 `"auto"`。
`ScrollPane.scrollDrag`	指明当用户在 ScrollPane 中按下并拖动内容时，是否会出现滚动。为 `true`，则滚动，为 `false`，则不滚动。默认值为 `false`。
`ScrollPane.vLineScrollSize`	箭头按钮被按下时要垂直滚动的内容数量。
`ScrollPane.vPageScrollSize`	轨道被按下时要垂直滚动的内容数量。

方法	描述
ScrollPane.vPosition	滚动窗格的垂直像素位置。
ScrollPane.vScrollPolicy	垂直滚动条的状态。它可以为始终打开 ("on")、始终关闭 ("off")，或者只是在需要时才打开 ("auto")。默认值为 "auto"。

继承 UIObject 类和 UIComponent 类的所有属性。

ScrollPane 类的事件摘要

方法	描述
ScrollPane.complete	加载了滚动窗格内容时广播。
ScrollPane.progress	在加载滚动条内容时广播。
ScrollPane.scroll	滚动条被按下时广播。

继承 UIObject 类和 UIComponent 类的所有事件。

ScrollPane.complete

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(complete){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
  ...
}
scrollPaneInstance.addEventListener("complete", listenerObject)
```

描述

事件；当加载完内容时向所有已注册的侦听器广播。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到 ScrollPane 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下列代码附加到 ScrollPane 组件实例 myScrollPaneComponent，它将 “_level0.myScrollPaneComponent” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*scrollPaneInstance*) 调度一个事件（在本例中为 `complete`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例为 `scrollPane` 实例创建一个具有 `complete` 事件处理函数的侦听器对象：

```
form.complete = function(eventObj){
  // 插入代码以便处理事件
}
scrollPane.addEventListener("complete",form);
```

ScrollPane.content

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.content

描述

属性 （只读）；对滚动窗格内容的引用。到加载开始时才会定义该值。

范例

该范例将 `mcLoaded` 变量设为 `content` 属性的值：

```
var mcLoaded = scrollPane.content;
```

另请参见

`ScrollPane.contentPath`

ScrollPane.contentPath

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.contentPath

描述

属性；一个字符串，它指明加载到滚动窗格中的 SWF 或 JPEG 文件的绝对 URL 或相对 URL。相对路径必须相对于加载内容的 SWF。

如果使用相对 URL 加载内容，加载的内容必须相对于包含滚动窗格的 SWF 文件的位置。例如，使用位于 /scrollpane/nav/example.swf 目录中的 ScrollPane 组件的应用程序可从具有以下 contentPath 属性的目录 /scrollpane/content/flash/logo.swf 加载内容："../content/flash/logo.swf"

范例

下面的范例是让滚动窗格显示来自 Internet 的图像的内容：

```
scrollPane.contentPath ="http://imagecache2.allposters.com/images/43/
  033_302.jpg";
```

下面的范例通知滚动窗格显示来自库的元件的内容：

```
scrollPane.contentPath ="movieClip_Name";
```

下面的范例是让滚动窗格显示本地文件 “logo.swf” 的内容：

```
scrollPane.contentPath = "logo.swf";
```

另请参见

ScrollPane.content

ScrollPane.getBytesLoaded()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.getBytesLoaded()

参数

无。

返回

滚动窗格中已加载的字节数。

描述

方法；返回 ScrollPane 实例中已加载的字节数。加载内容时可按固定时间间隔调用该方法以查看其进度。

范例

该范例创建名为 scrollPane 的 ScrollPane 类实例。然后定义一个名为 loadListener 的侦听器对象，该对象带有一个 progress 事件处理函数，它调用 getBytesLoaded() 方法以帮助确定加载进度：

```
createClassObject(mx.containers.ScrollPane, "scrollPane", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
  // eventObj.target 为生成更改事件的组件
  var bytesLoaded = scrollPane.getBytesLoaded();
  var bytesTotal = scrollPane.getBytesTotal();
  var percentComplete = Math.floor(bytesLoaded/bytesTotal);

  if (percentComplete < 5 )  // 加载开始
  {
    trace( "开始从 Internet 加载内容" );
  }
  else if(percentComplete = 50) //50% complete
  {
    trace( "已下载 50% 的内容" );
  }
}
scrollPane.addEventListener("progress", loadListener);
scrollPane.contentPath = "http://www.geocities.com/hcls_matrix/Images/
  homeview5.jpg";
```

ScrollPane.getBytesTotal()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.getBytesTotal()

参数

无。

返回

一个数字。

描述

方法；返回要加载到 ScrollPane 实例中的总字节数。

另请参见

ScrollPane.getBytesLoaded()

ScrollPane.hLineScrollSize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.hLineScrollSize

描述

属性；水平滚动条中的向左箭头或向右箭头被按下时内容要移动的像素数。默认值为 5。

范例

该范例将水平滚动单位增加到 10：

```
scrollPane.hLineScrollSize = 10;
```

ScrollPane.hPageScrollSize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.hPageScrollSize

描述

属性；水平滚动条中的轨道被按下时内容要移动的像素数。默认值为 20。

范例

该范例将水平页滚动单位增加到 30：

```
scrollPane.hPageScrollSize = 30;
```

ScrollPane.hPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.hPosition

描述

属性；水平滚动条的像素位置。位置 0 表示滚动条的左端。

范例

该范例将滚动条的位置设为 20：

```
scrollPane.hPosition = 20;
```

ScrollPane.hScrollPolicy

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
scrollPaneInstance.hScrollPolicy
```

描述

属性；确定水平滚动条是始终存在 ("on")、从不存在 ("off")，还是根据图像大小自动出现 ("auto")。默认值为 "auto"。

范例

下列代码将滚动条设置为一直启用：

```
scrollPane.hScrollPolicy = "on";
```

ScrollPane.progress

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(progress){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.progress = function(eventObject){
  ...
}
scrollPaneInstance.addEventListener("progress", listenerObject)
```

描述

事件；在加载内容时向所有已注册的侦听器广播。progress 事件并不会始终广播；complete 事件可能在未调度任何 progress 事件的情况下广播。如果加载的内容是本地文件，尤其会出现这种情况。当通过设置 contentPath 属性的值开始加载内容时会触发此事件。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到 ScrollPane 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，以下附加到 ScrollPane 组件实例 mySPComponent 的代码将 “_level0.mySPComponent” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*scrollPaneInstance*) 会调度一个事件（在本例中为 progress），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的代码创建一个名为 scrollPane 的 ScrollPane 实例，然后创建一个侦听器对象，该对象具有一个用于 progress 事件的事件处理函数，它将一条有关已加载内容的字节数的消息发送到 “输出” 面板：

```
createClassObject(mx.containers.ScrollPane, "scrollPane", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
  // eventObj.target 为生成进程事件的组件
  // 在本例中为 scrollPane
  trace("logo.swf has loaded " + scrollPane.getBytesLoaded() + " Bytes.");
  // 跟踪加载进程
}
scrollPane.addEventListener("complete", loadListener);
scrollPane.contentPath = "logo.swf";
```

ScrollPane.refreshPane()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.refreshPane()

参数

无。

返回

无。

描述

方法；加载内容以后刷新滚动窗格。该方法会重新加载内容。例如，如果您已经将一个表单加载到 ScrollPane 中，并且已经使用 “动作脚本” 更改了一个输入属性 （例如，在一个文本字段中），那么，您就可以使用该方法。调用 `refreshPane()` 来重新加载具有这些输入属性的新值的同一个表单。

范例

下面的范例会刷新滚动窗格实例 `sp`：

```
sp.refreshPane();
```

ScrollPane.scroll

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(scroll){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.scroll = function(eventObject){
  ...
}
scrollPaneInstance.addEventListener("scroll", listenerObject)
```

事件对象

除了标准的事件对象属性之外，还有一个为 `scroll` 事件定义的 `type` 属性，其值为 `"scroll"`。另外还有一个 `direction` 属性，其可能值为 `"vertical"` 和 `"horizontal"`。

描述

事件；当用户按下滚动条按钮、滑块或轨道时，向所有已注册的侦听器广播。与其他事件不同的是，当用户按下滚动条时，`scroll` 事件开始持续广播，直到用户松开滚动条。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到 ScrollPane 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下面附加到实例 `sp` 的代码将 “_level0.sp” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*scrollPaneInstance*) 调度一个事件 （在本例中为 scroll），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

该范例创建一个具有 scroll 回调函数的 form 侦听器对象，该侦听器对象注册到 spInstance 实例：您必须为 spInstance 填写内容，如下所示：

```
spInstance.contentPath = "mouse3.jpg";
form = new Object();
form.scroll = function(eventObj){
  trace("ScrollPane scrolled");
}
spInstance.addEventListener("scroll", form);
```

另请参见

UIEventDispatcher.addEventListener()

ScrollPane.scrollDrag

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.scrollDrag

描述

属性；一个布尔值，指明当用户在 ScrollPane 中按下并拖动内容时，是否会滚动内容，为 true 则滚动，为 false 则不滚动。默认值为 false。

范例

该范例在滚动窗格内启用鼠标滚动：

```
scrollPane.scrollDrag = true;
```

ScrollPane.vLineScrollSize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.vLineScrollSize

描述

属性；当按下垂直滚动条中的向上或向下箭头按钮时，显示区域移动的像素数。默认值为 5。

范例

该代码会将显示区域在垂直滚动条箭头按钮被按下时的移动量增加到 10：

```
scrollPane.vLineScrollSize = 10;
```

ScrollPane.vPageScrollSize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

scrollPaneInstance.vPageScrollSize

描述

属性；当按下垂直滚动条中的轨道时显示区域移动的像素数。默认值为 20。

范例

该代码会将显示区域在垂直滚动条箭头按钮被按下时的移动量增加到 30：

```
scrollPane.vPageScrollSize = 30;
```

ScrollPane.vPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.vPosition

描述

属性；垂直滚动条的像素位置。默认值为 0。

ScrollPane.vScrollPolicy

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

scrollPaneInstance.vScrollPolicy

描述

属性；确定垂直滚动条是始终显示 ("on")、从不显示 ("off") 还是根据图像尺寸自动出现 ("auto")。默认值为 "auto"。

范例

下列代码将垂直滚动条设为始终打开：

```
scrollPane.vScrollPolicy = "on";
```

StyleManager 类

动作脚本类命名空间 mx.styles.StyleManager

StyleManager 类会跟踪已知的继承样式和颜色。只有在您要创建组件并希望添加新的继承样式或颜色时，才需要使用该类。

要确定哪些样式是继承样式，请参考 W3C Web 站点。

StyleManager 类的方法摘要

方法	描述
StyleManager.registerColorName()	使用 StyleManager 注册新的颜色名称。
StyleManager.registerColorStyle()	使用 StyleManager 注册新的颜色样式。
StyleManager.registerInheritingSyle()	使用 StyleManager 注册新的继承样式。

StyleManager.registerColorName()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

StyleManager.registerColorName(*colorName*, *value*)

参数

colorName 指明颜色名称的字符串（例如，"gray"、"darkGrey" 等等）。

value 指明颜色的十六进制数（例如，0x808080、0x404040 等等）。

返回

无。

描述

方法；将颜色名称与十六进制数值相关联并使用 StyleManager 注册颜色名称。

范例

下面的范例将 “gray” 注册为颜色的名称，该颜色是以十六进制数值 0x808080 表示的颜色：

```
StyleManager.registerColorName("gray", 0x808080 );
```

StyleManager.registerColorStyle()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
StyleManager.registerColorStyle(colorStyle)
```

参数

colorStyle 指明颜色名称的字符串 （例如，"highlightColor"、"shadowColor"、"disabledColor" 等等）。

返回

无。

描述

方法；将新的颜色样式添加到 StyleManager。

范例

下面的范例将 “highlightColor” 注册为颜色样式：

```
StyleManager.registerColorStyle("highlightColor");
```

StyleManager.registerInheritingSyle()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
StyleManager.registerInheritingStyle(propertyName)
```

参数

propertyName 指明样式属性名称的字符串 （例如，"newProp1"、"newProp2" 等等）。

返回

无。

描述

方法；将此样式属性标记为继承。使用该方法注册未在 CSS 规范中列出的样式属性。请不要使用该方法将非继承样式属性更改为继承。

范例

下面的范例将 `newProp1` 注册为继承样式：

```
StyleManager.registerInheritingStyle("newProp1");
```

Slide 类

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

TextArea 组件

TextArea 组件环绕着本机 “动作脚本” TextField 对象。您可以使用样式自定义 TextArea 组件；当实例被禁用时，其内容以 “disabledColor” 样式所代表的颜色显示。TextArea 组件也可以采用 HTML 格式，或者作为掩饰文本的密码字段。

在应用程序中可以启用或禁用 TextArea 组件。在禁用状态下，它不接收鼠标或键盘输入。当启用时，它遵循与 “动作脚本” TextField 对象相同的焦点、选择和导航规则。当 TextArea 实例具有焦点时，您可以使用下列按键对其进行控制：

按键	描述
箭头键	将插入点向上、向下、向左或向右移动一行。
Page Down 键	向下移动一屏。
Page Up 键	向上移动一屏。
Shift + Tab	将焦点移到前一个对象。
Tab 键	将焦点移到下一个对象。

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航” 或第 93 页的 “FocusManager 类”。

每个 TextArea 实例的实时预览都会反映创作时在属性检查器或 “组件检查器” 面板中对参数所做的更改。如果需要滚动条，它会出现在实时预览中，但并不起作用。在实时预览中，文本是不可选定的，并且无法在舞台上的组件实例中输入文本。

在将 TextArea 组件添加到应用程序中时，您可以使用 “辅助功能” 面板使其可由屏幕读取器访问。使用 TextArea 组件

在需要多行文本字段的任何地方都可使用 TextArea 组件。如果需要单行文本字段，请使用第 200 页的 “TextInput 组件”。例如，您可以在表单中使用 TextArea 组件作为注释字段。您可以设置侦听器来检查当用户切换到字段外时，字段是否为空。侦听器可能会显示错误信息，以指明必须在该字段中输入注释。

TextArea 组件参数

下列是您可以在属性检查器中或在 “组件检查器” 面板中为每个 TextArea 组件设置的创作参数：

text 指明 TextArea 的内容。您无法在属性检查器或 “组件检查器” 面板中输入回车。默认值为 "" （空字符串）。

html 指明文本是 (true) 否 (false) 采用 HTML 格式。默认值为 false。

editable 指明 TextArea 组件是 (true) 否 (false) 可编辑。默认值为 true。

wordWrap 指明文本是 (true) 否 (false) 自动换行。默认值为 true。

您可以编写 “动作脚本”，通过利用其属性、方法和事件来控制 TextArea 组件这些选项及其他选项。有关详细信息，请参阅 TextArea 类。

创建具有 TextArea 组件的应用程序

以下过程解释了如何在创作时将 TextArea 组件添加到应用程序。在该范例中，组件为一个具有事件侦听器的 “注释” 字段，该侦听器确定用户是否输入了文本。

要创建具有 TextArea 组件的应用程序，请执行以下步骤：

1 将 TextArea 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，输入实例名称 **comment**。

3 在属性检查器中，按照需要设置参数。但是，请将文本参数保留为空，将可编辑参数设为 true，将密码参数设为 false。

4 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

```
textListener = new Object();
textListener.handleEvent = function (evt){
  if (comment.length < 1)  {
  Alert(_root, "错误", "您必须在此字段中至少输入一个注释", mxModal | mxOK);
  }
}
comment.addEventListener("focusOut", textListener);
```

该代码在 TextArea 实例 `comment` 上设置了一个 `FocusOut` 事件处理函数，该事件处理函数用来验证用户是否在文本字段中键入了内容。

5 在注释实例中输入文本之后，您可以获取文本的值，如下所示：

```
var login = comment.text;
```

自定义 TextArea 组件

您可以在创作时或在运行时，在水平和垂直方向上改变 TextArea 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `UIObject.setSize()` 或 TextArea 类中任何适用的属性和方法。

调整了 TextArea 组件的大小后，边框大小就会调整到新的边框。如果需要滚动条，它们会被放置在下边缘和右边缘。然后，文本字段的大小会在其余区域内调整；TextArea 组件中没有大小固定的元素。如果 TextArea 组件太小不能显示文本，文本就会被裁剪。

对 TextArea 组件使用样式

对于字段中的所有文本来说， TextArea 组件支持一组组件样式。但是，您也可以显示与 Flash Player 的 HTML 呈现兼容的 HTML。要显示 HTML 文本，请将 `TextArea.html` 设为 `true`。

TextArea 组件在类样式声明中定义了它的 `backgroundColor` 和 `borderStyle` 样式属性。类样式覆盖 _global 样式，因此，如果要设置 `backgroundColor` 和 `borderStyle` 样式属性，必须在实例上创建一个不同的自定义样式声明。

如果样式属性的名称以 “Color” 结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

TextArea 组件支持下列样式：

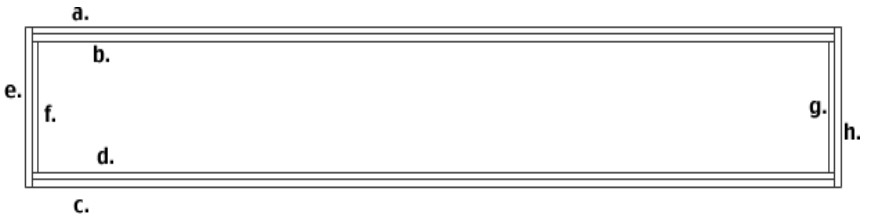
样式	描述
`color`	文本的默认颜色。
`embedFonts`	文档中嵌入的字体。
`fontFamily`	文本的字体名称。
`fontSize`	字体的磅值。
`fontStyle`	字体样式，“常规” 或 “斜体”。
`fontWeight`	字体粗细，“常规” 或 “粗体”。
`textAlign`	文本对齐方式：“左”、“右” 或 “居中”。
`textDecoration`	文本修饰，“无” 或 “下划线”。

对 TextArea 组件使用外观

TextArea 组件使用 RectBorder 类来绘制其边框。您可以使用 `setStyle()` 方法（请参阅 `UIObject.setStyle()`）来更改下列 RectBorder 样式属性：

RectBorder 样式
`borderColor`
`highlightColor`
`borderColor`
`shadowColor`
`borderCapColor`
`shadowCapColor`
`shadowCapColor`
`borderCapColor`

这些样式属性设置边框上的下列位置：



TextArea 类

继承 UIObject > UIComponent > View > ScrollView > TextArea

动作脚本类命名空间 mx.controls.TextArea

TextArea 类的属性允许您在运行时设置文本内容、格式以及水平和垂直位置。您也可以指明该字段是否可编辑，以及该字段是否为 “密码” 字段。您还可以限制用户可以输入的字符。

使用 “动作脚本” 设置 TextArea 类的属性会覆盖在属性检查器或 “组件检查器” 面板中设置的同名参数。

TextArea 组件会覆盖默认的 Flash Player 焦点矩形，并绘制一个带圆角的自定义焦点矩形。

TextArea 组件支持 CSS 样式和 Flash Player 所支持的任何其他 HTML 样式。

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

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

注意：下面的代码返回未定义的：`trace(myTextAreaInstance.version);`。

TextArea 类的属性摘要

属性	描述
TextArea.editable	一个布尔值，指明该字段是 (true) 否 (false) 可编辑。
TextArea.hPosition	定义滚动窗格中文本的水平位置。
TextArea.hScrollPolicy	指明水平滚动条是始终打开 ("on")，从不打开 ("off")，还是在需要时打开 ("auto")。
TextArea.html	一个标记，指明文本字段是否可以采用 HTML 格式。
TextArea.length	文本字段中的字符数。该属性为只读。
TextArea.maxChars	文本字段最多可以容纳的字符数。
TextArea.maxHPosition	TextArea.hPosition 的最大值。
TextArea.maxVPosition	TextArea.vPosition 的最大值。
TextArea.password	一个布尔值，指明字段是密码字段 (true) 还是不是密码字段 (false)。
TextArea.restrict	用户可在文本字段中输入的字符集。
TextArea.text	TextArea 组件的文本内容。
TextArea.vPosition	一个数字，指明垂直滚动位置
TextArea.vScrollPolicy	指明垂直滚动条是始终打开 ("on")，从不打开 ("off")，还是在需要时打开 ("auto")。
TextArea.wordWrap	一个布尔值，指明文本是 (true) 否 (false) 自动换行。

TextArea 类的事件摘要

事件	描述
TextArea.change	通知侦听器文本已更改。

TextArea.change

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(change){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
  ...
}
textAreaInstance.addEventListener("change", listenerObject)
```

描述

事件；通知侦听器文本已更改。在文本更改后广播该事件。不能使用该事件防止某些字符添加到组件的文本字段，而应使用 `TextArea.restrict`。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到 TextArea 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下列代码附加到 `myTextArea` 实例，它将 “_level0.myTextArea” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*textAreaInstance*) 调度一个事件 （在本例中为 `change`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

该范例跟踪文本字段已更改的总次数：

```
myTextArea.changeHandler = function(obj) {
  this.changeCount++;
  trace(obj.target);
  trace("text has changed " + this.changeCount + " times now! it now contains "
  +
this.text);
}
```

另请参见

`UIEventDispatcher.addEventListener()`

TextArea.editable

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.editable

描述

属性；一个布尔值，指明组件是 (true) 否 (false) 可编辑。默认值为 true。

TextArea.hPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.hPosition

描述

属性；定义文本在字段中的水平位置。默认值为 0。

范例

以下代码显示字段中最左边的字符：

```
myTextArea.hPosition = 0;
```

TextArea.hScrollPolicy

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.hScrollPolicy

描述

属性；确定水平滚动条是始终显示 ("on")、从不显示 ("off")，还是根据字段大小自动显示 ("auto")。默认值为 "auto"。

范例

以下代码使水平滚动条始终打开：

```
text.hScrollPolicy = "on";
```

TextArea.html

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.html

描述

属性；一个布尔值，指明该文本字段是 (true) 否 (false) 采用 HTML 格式。如果 html 属性为 true，则文本字段为 HTML 文本字段。如果 html 为 false，则文本字段为非 HTML 文本字段。默认值为 false。

范例

以下范例使 myTextArea 字段成为 HTML 文本字段，然后使用 HTML 标记设置文本格式：

```
myTextArea.html = true;
myTextArea.text = "The <b>Royal</b> Nonesuch"; // 显示 “The Royal Nonesuch”
```

TextArea.length

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.length

描述

属性（只读）；指明文本字段中的字符数。此属性与动作脚本 text.length 属性返回的值相同，但速度更快。制表符（“\t”）这样的字符会被算作一个字符。默认值为 0。

范例

以下范例获取文本字段的长度并将它复制到 length 变量中：

```
var length = myTextArea.length; // 查明文本字符串的长度
```

TextArea.maxChars

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.maxChars

描述

属性；该文本字段最多可容纳的字符数。脚本插入的文本可能会比 maxChars 属性允许的字符数多；maxChars 属性只是指明用户可以输入多少文本。如果此属性的值为 null，则对用户可以输入的文本量没有限制。默认值为空。

范例

以下范例将用户可以输入的字符数限制为 255：

```
myTextArea.maxChars = 255;
```

TextArea.maxHPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.maxHPosition

描述

属性（只读）；TextArea.hPosition 的最大值。默认值为 0。

范例

以下代码将文本滚动到最右侧：

```
myTextArea.hPosition = myTextArea.maxHPosition;
```

另请参见

TextArea.vPosition

TextArea.maxVPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.maxVPosition

描述

属性 （只读）；指明 TextArea.vPosition 的最大值。默认值为 0。

范例

以下代码将文本滚动到组件的底部：

```
myTextArea.vPosition = myTextArea.maxVPosition;
```

另请参见

TextArea.hPosition

TextArea.password

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.password

描述

属性；一个布尔值，指明文本字段是 (true) 否 (false) 为密码字段。如果 password 的值为 true，则此文本字段为密码文本字段并且会隐藏输入字符。如果为 false，则此文本字段不是密码文本字段。默认值为 false。

范例

以下代码使文本字段为密码字段，该字段将所有字符显示为星号 (*)：

```
myTextArea.password = true;
```

TextArea.restrict

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.restrict

描述

属性；指明用户可输入到文本字段中的字符集。默认值未定义。如果 restrict 属性的值为空，用户就可以输入任何字符。如果 restrict 属性的值为空字符串，则不能输入任何字符。如果 restrict 属性的值为一个字符串，则只能向文本字段中输入该字符串中的字符；系统将从左向右扫描该字符串。可以使用短划线 (-) 指定范围。

restrict 属性只限制用户交互；脚本可将任何文本放入文本字段中。此属性与属性检查器中的“嵌入字体轮廓”复选框不同步。

如果此字符串以“^”开头，则先接受所有字符，然后从已接受的字符集中排除字符串中 ^ 之后的字符。如果此字符串不以“^”开头，则最初不接受任何字符，然后将字符串中的字符包括在接受字符集中。

范例

在以下范例中，第一行代码将文本字段限定为大写字母、数字和空格。第二行代码允许除小写字母之外的所有字符。

```
my_txt.restrict = "A-Z 0-9"; // 将控件限定为大写字母、数字和空格。
my_txt.restrict = "^a-z"; // 允许除小写字母之外的所有字符。
```

TextArea.text

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.text

描述

属性；TextArea 组件的文本内容。默认值为 "" （空字符串）。

范例

以下代码在 myTextArea 实例中放置一个字符串，然后用 trace 命令将该字符串发送到“输出”面板：

```
myTextArea.text = "The Royal Nonesuch";
trace(myTextArea.text); // 跟踪 “The Royal Nonesuch”
```

TextArea.vPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.vPosition

描述

属性；定义文本在文本字段中的垂直位置。scroll 属性很有用，该属性可以帮助用户定位到长篇文章的特定段落，还可用于创建滚动文本字段。您可以获取并设置该属性。默认值为 0。

范例

以下代码会使字段中最顶端的字符显示出来：

```
myTextArea.vPosition = 0;
```

TextArea.vScrollPolicy

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.vScrollPolicy

描述

属性；确定垂直滚动条是始终显示 ("on")、从不显示 ("off")，还是根据字段大小自动出现 ("auto")。默认值为 "auto"。

范例

以下代码使垂直滚动条始终关闭：

```
text.vScrollPolicy = "off";
```

TextArea.wordWrap

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textAreaInstance.wordWrap

描述

属性，一个布尔值，指明文本是 (`true`) 否 (`false`) 自动换行。默认值为 `true`。

TextInput 组件

TextInput 是一个环绕本机 “动作脚本” TextField 对象的单行组件。您可以使用样式自定义 TextInput 组件；当实例被禁用时，它的内容会显示为 “disabledColor” 样式表示的颜色。TextInput 组件也可以采用 HTML 格式，或作为掩饰文本的密码字段。

在应用程序中，TextInput 组件可以被启用或者禁用。在禁用状态下，它不接收鼠标或键盘输入。当启用时，它遵循与 “动作脚本” TextField 对象相同的焦点、选择和导航规则。当一个 TextInput 实例有焦点时，您还可以使用以下按键来控制它：

按键	描述
箭头键	向左和向右移动一个字符的距离。
Shift + Tab	将焦点移到前一个对象。
Tab 键	将焦点移到下一个对象。

有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航” 或第 93 页的 “FocusManager 类”。

每个 TextInput 实例的实时预览都会反映在创作过程中对属性检查器或 “组件检查器” 面板中的参数所做的更改。在实时预览中，文本是不可选定的，并且无法在舞台上的组件实例中输入文本。

在将 TextInput 组件添加到应用程序时，您可以使用 “辅助功能” 面板来使其可由屏幕读取器访问。

使用 TextInput 组件

在任何需要单行文本字段的地方，都可以使用 TextInput 组件。如果您需要多行文本字段，请使用第 189 页的 “TextArea 组件”。例如，您可以在表单中将 TextInput 组件用作密码字段。您可以设置一个侦听器，检查用户按 Tab 键切换到字段之外时，该字段是否有足够的字符。该侦听器可以显示一条错误信息，指明必须输入正确的字符数。

TextInput 组件参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 TextInput 组件设置的创作参数：

text 指定 TextInput 的内容。您无法在属性检查器或 “组件检查器” 面板中输入回车。默认值为 "" （空字符串）。

editable 指明 TextInput 组件是 (true) 否 (false) 可编辑。默认值为 true。

password 指明字段是 (true) 否 (false) 为密码字段。默认值为 false。

您可以编写 “动作脚本”，通过利用其属性、方法和事件来处理 TextInput 组件的这些和其他选项。有关详细信息，请参阅 TextInput 类。

创建具有 TextInput 组件的应用程序

以下过程解释了如何在创作时将 TextInput 组件添加到应用程序。在本范例中，组件是带有事件侦听器的密码字段，事件侦听器确定输入的字符数是否正确。

要创建具有 TextInput 组件的应用程序，请执行以下操作：

1 将 TextInput 组件从 “组件” 面板拖到舞台上。

2 在属性检查器中，输入实例名称 **passwordField**。

3 在属性检查器中，执行以下操作：

- 将 text 参数保留为空。
- 将 editable 参数设置为 true。
- 将 password 参数设置为 true。

4 在时间轴中选择第一帧，打开 “动作” 面板，然后输入以下代码：

```
textListener = new Object();
textListener.handleEvent = function (evt){
  if (evt.type == "enter"){
    trace("You must enter at least 8 characters");
  }
}
passwordField.addEventListener("enter", textListener);
```

该代码在 TextInput `passwordField` 实例上设置了一个 `enter` 事件处理函数，该事件处理函数用来验证用户输入的字符数是否正确。

5 在 `passwordField` 实例中输入文本后，您可以获取它的值，如下所示：

```
var login = passwordField.text;
```

自定义 TextInput 组件

在创作过程中和在运行时，您都可以在水平方向上改变 TextInput 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 `UIObject.setSize()` 或 TextInput 类中任何适用的属性和方法。

在调整 TextInput 组件大小时，边框将相应调整为新边框。 TextInput 组件不使用滚动条，但当用户与文本交互操作时插入点会自动滚动。然后在剩余区域中调整文本字段大小，在 TextInput 组件中没有固定大小的元素。如果 TextInput 组件太小而无法显示文本，则该文本将会被裁剪。

对 TextInput 组件使用样式

TextInput 组件在类样式声明中定义了它的 `backgroundColor` 和 `borderStyle` 样式属性。类样式覆盖 _global 样式，因此，如果要设置 `backgroundColor` 和 `borderStyle` 样式属性，必须在实例上创建一个不同的自定义样式声明。

TextInput 组件支持以下样式：

样式	描述
`color`	文本的默认颜色。
`embedFonts`	文档中嵌入的字体。
`fontFamily`	文本的字体名称。

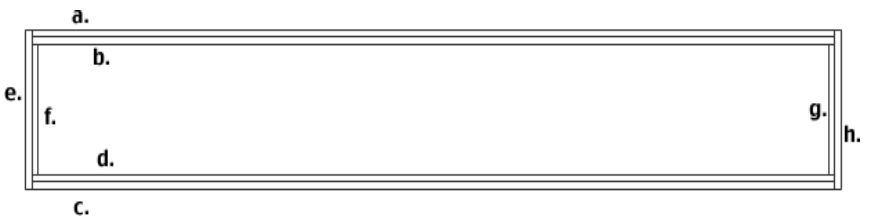
样式	描述
fontSize	字体的磅值。
fontStyle	字体样式，“常规”或“斜体”。
fontWeight	字体粗细，“常规”或“粗体”。
textAlign	文本对齐方式：“左”、“右”或“居中”。
textDecoration	文本修饰，“无”或“下划线”。

使用具有 TextInput 组件的外观

TextArea 组件使用 RectBorder 类来绘制其边框。您可以使用 setStyle() 方法（请参阅 UIObject.setStyle()）来更改下列 RectBorder 样式属性：

RectBorder 样式
borderColor
highlightColor
borderColor
shadowColor
borderCapColor
shadowCapColor
shadowCapColor
borderCapColor

这些样式属性设置边框上的下列位置：



TextInput 类

继承 UIObject > UIComponent > TextInput

动作脚本类命名空间 mx.controls.TextInput

使用 TextInput 类的属性，您可以在运行时设置文本内容、格式和水平位置。您也可以指明该字段是否可编辑，以及该字段是否为“密码”字段。您还可以限制用户可以输入的字符。

使用“动作脚本”设置 TextInput 类的属性会覆盖在属性检查器或“组件检查器”面板中设置的同名参数。

TextInput 组件使用 FocusManager 来覆盖默认的 Flash Player 焦点矩形并绘制一个带圆角的自定义焦点矩形。有关详细信息，请参阅第 93 页的“FocusManager 类”。

TextInput 组件支持 CSS 样式和 Flash Player 支持的任何其他 HTML 样式。有关 CSS 支持的信息，请参阅 W3C 规范。

您可以通过使用由文本对象返回的字符串来操作文本字符串。

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

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

注意：下面的代码返回未定义的：`trace(myTextInputInstance.version);`。

TextInput 类的方法摘要

继承 UIObject 类和 UIComponent 类中的所有方法。

TextInput 类的属性摘要

属性	描述
`TextInput.editable`	一个布尔值，指明该字段是 (`true`) 否 (`false`) 可编辑。
`TextInput.hPosition`	文本字段的水平滚动位置。
`TextInput.length`	TextInput 文本字段中的字符数。该属性为只读。
`TextInput.maxChars`	用户可以在 TextInput 文本字段输入的最大字符数。
`TextInput.maxHPosition`	TextField.hPosition 的最大可能值。该属性为只读。
`TextInput.password`	一个布尔值，指明该输入文本字段是否为隐藏所输入字符的密码字段。
`TextInput.restrict`	指明用户可以在文本字段输入哪些字符。
`TextInput.text`	设置 TextInput 文本字段的文本内容。

继承 UIObject 类和 UIComponent 类中的所有方法。

TextInput 类的事件摘要

事件	描述
`TextInput.change`	在输入字段更改时触发。
`TextInput.enter`	在按 Enter 键时触发。

继承 UIObject 类和 UIComponent 类中的所有方法。

TextInput.change

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(change){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
  ...
}
textInputInstance.addEventListener("change", listenerObject)
```

描述

事件；通知侦听器文本已更改。在文本更改后广播该事件。不能使用该事件防止某些字符添加到组件的文本字段，而应使用 `TextInput.restrict`。该事件只能通过用户输入触发，不能通过编程方式的更改来触发。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 TextInput 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下列代码附加到 `myTextInput` 实例，它将 “_level0.myTextInput” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*textInputInstance*) 调度一个事件 （在本例中为 `change`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

该范例在应用程序中设置一个标记，该标记指明 TextInput 字段中的内容是否已更改：

```
form.change = function(eventObj){
  // eventObj.target 是生成 change 事件的组件，
  // 即 Input 组件。
  myFormChanged.visible = true;  // 如果内容已更改，设置一个更改指示符；
}
myInput.addEventListener("change", form);
```

另请参见

`UIEventDispatcher.addEventListener()`

TextInput.editable

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
textInputInstance.editable
```

描述

属性；一个布尔值，指明组件是 (`true`) 否 (`false`) 可编辑。默认值为 `true`。

TextInput.enter

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(enter){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.enter = function(eventObject){
  ...
}
textInputInstance.addEventListener("enter", listenerObject)
```

描述

事件；通知侦听器 Enter 键已被按下。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 TextInput 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下列代码附加到 `myTextInput` 实例，它将 “_level0.myTextInput” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*textInputInstance*) 调度一个事件 （在本例中为 `enter`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

该范例在应用程序中设置一个标记，该标记指明 TextInput 字段中的内容是否已更改：

```
form.enter = function(eventObj){
  // eventObj.target 是生成 enter 事件的组件，
  // 即 Input 组件。
  myFormChanged.visible = true;
// 如果用户按下 Enter 键，则设置一个更改指示符；
}
myInput.addEventListener("enter", form);
```

另请参见

`UIEventDispatcher.addEventListener()`

TextInput.hPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textInputInstance.hPosition

描述

属性；定义文本在字段中的水平位置。默认值为 0。

范例

以下代码显示字段中最左边的字符：

```
myTextInput.hPosition = 0;
```

TextInput.length

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

inputInstance.length

描述

属性（只读）；一个数字，指明 TextInput 组件中的字符数。制表符（“\t”）这样的字符会被算作一个字符。默认值为 0。

范例

以下代码确定 myTextInput 字符串中的字符数，并将该数值复制到 length 变量中：

```
var length = myTextInput.length;
```

TextInput.maxChars

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textInputInstance.maxChars

描述

属性；该文本字段最多可容纳的字符数。脚本插入的文本可能会比 maxChars 属性允许的字符数多；maxChars 属性只是指明用户可以输入多少文本。如果此属性的值为 null，则对用户可以输入的文本量没有限制。默认值为空。

范例

以下范例将用户可以输入的字符数限制为 255：

```
myTextInput.maxChars = 255;
```

TextInput.maxHPosition

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textInputInstance.maxHPosition

描述

属性（只读）；指明 TextInput.hPosition 的最大值。默认值为 0。

范例

以下代码会滚动到最右边：

```
myTextInput.hPosition = myTextInput.maxHPosition;
```

TextInput.password

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textInputInstance.password

描述

属性；一个布尔值，指明文本字段是 (true) 否 (false) 为密码字段。如果 password 的值为 true，则此文本字段为密码文本字段并且会隐藏输入字符。如果为 false，则此文本字段不是密码文本字段。默认值为 false。

范例

以下代码使文本字段为密码字段，该字段将所有字符显示为星号 (*)：

```
myTextInput.password = true;
```

TextInput.restrict

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textInputInstance.restrict

描述

属性；指明用户可输入到文本字段中的字符集。默认值未定义。如果 restrict 属性的值为 null 或空字符串 ("")，则用户可以输入任意字符。如果 restrict 属性的值为一个字符串，则只能向文本字段中输入该字符串中的字符；系统将从左向右扫描该字符串。可以使用短划线 (-) 指定范围。

restrict 属性只限制用户交互；脚本可将任何文本放入文本字段中。此属性与属性检查器中的“嵌入字体轮廓”复选框不同步。

如果此字符串以“^”开头，则先接受所有字符，然后从已接受的字符集中排除字符串中 ^ 之后的字符。如果此字符串不以“^”开头，则最初不接受任何字符，然后将字符串中的字符包括在接受字符集中。

反斜线字符可以用于输入字符“-”、“^”和“\”，如下所示：

```
\^
\-
\\
```

在 “动作” 面板中，当在 "" （双引号）中输入 \ 字符时，对于 “动作” 面板的双引号解释器，该字符有特殊的含义。它表示 \ 之后的字符应被视为其本身的含义。例如，下列代码用于输入单个引号 ：

```
var leftQuote = "\"";
```

“动作” 面板的 .restrict 解释器也将 \ 用作转义符。因此，您可能会认为下列代码应该起作用 ：

```
myText.restrict = "0-9\-\^\\";
```

但是，因为此表达式包含在双引号内，所以会将下面的值发送到 .restrict 解释器 ：0-9-^\，.restrict 解释器将不能识别此值。

因为必须在双引号中输入此表达式，所以不仅要为 .restrict 解释器提供表达式，而且还必须转义 “动作” 面板中双引号的内置解释器。若要将值 0-9\-\^\\ 发送到 .restrict 解释器，您必须输入下列代码 ：

```
myText.restrict = "0-9\\-\\^\\\\";
```

范例

在以下范例中，第一行代码将文本字段限定为大写字母、数字和空格。第二行代码允许除小写字母之外的所有字符。

```
my_txt.restrict = "A-Z 0-9";
my_txt.restrict = "^a-z";
```

以下代码允许用户在实例 myText 中输入字符 “0 1 2 3 4 5 6 7 8 9 - ^ \” 。您必须使用双反斜线使字符 “-” 、“^” 和 “\” 转义。第一个 “\” 转义为 “ ”，第二个 “\” 指示解释器不应将下一个字符视为特殊字符，如下所示 ：

```
myText.restrict = "0-9\\-\\^\\\\";
```

TextInput.text

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

textInputInstance.text

描述

属性 ；TextInput 组件的文本内容。默认值为 "" （空字符串）。

范例

以下代码将字符串放入 myTextInput 实例中，然后用 trace 命令将该字符串发送到 “输出” 面板 ：

```
myTextInput.text = "The Royal Nonesuch";
trace(myTextInput.text); // 跟踪 “The Royal Nonesuch”
```

Tree 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

UIComponent 类

继承 UIObject > UIComponent

动作脚本类命名空间 mx.core.UIComponent

所有 v2 组件均扩展 UIComponent ；它不是可视组件。 UIComponent 类包含的功能和属性使 Macromedia 组件能够共享一些常规行为。 UIComponent 类允许您执行以下操作 ：

- 接收焦点和键盘输入
- 启用和禁用组件
- 按布局调整大小

要使用 UIComponent 的方法和属性，您可以直接从您正在使用的任意一个组件中调用它们。例如，要从 RadioButton 组件调用 `UIComponent.setFocus()` 方法，您要编写以下代码 ：

```
myRadioButton.setFocus();
```

如果您要使用 Macromedia 组件 V2 结构创建新组件，只需创建 UIComponent 的实例即可。即使在这种情况下，其他子类（如 Button）通常仍会隐式创建 UIComponent。如果您确实需要创建 UIComponent 的实例，请使用以下代码 ：

```
class MyComponent extends UIComponent;
```

UIComponent 类的方法摘要

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

继承 UIObject 类的所有方法。

UIComponent 类的属性摘要

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

继承 UIObject 类的所有属性。

UIComponent 类的事件摘要

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

继承 UIObject 类的所有事件。

UIComponent.focusIn

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
on(focusIn){
  ...
}
listenerObject = new Object();
listenerObject.focusIn = function(eventObject){
  ...
}
componentInstance.addEventListener("focusIn", listenerObject)
```

描述

事件；通知侦听器对象已接收到键盘焦点。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （在本例中为 `focusIn`），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下代码在用户在文本字段 `txt` 中键入时禁用按钮：

```
txtListener.handleEvent = function(eventObj) {
  form.button.enabled = false;
}
txt.addEventListener("focusIn", txtListener);
```

另请参见

`UIEventDispatcher.addEventListener()`

UIComponent.focusOut

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
on(focusOut){
  ...
}
listenerObject = new Object();
listenerObject.focusOut = function(eventObject){
  ...
}
componentInstance.addEventListener("focusOut", listenerObject)
```

描述

事件；通知侦听器对象已丢失键盘焦点。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （在本例中为 focusOut），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下代码在用户离开文本字段 `txt` 时启用按钮：

```
txtListener.handleEvent = function(eventObj) {
  if (eventObj.type == focusOut){
      form.button.enabled = true;
  }
}
txt.addEventListener("focusOut", txtListener);
```

另请参见

`UIEventDispatcher.addEventListener()`

UIComponent.enabled

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.enabled
```

描述

属性；指明组件是否可以接受焦点和鼠标输入。如果值为 `true`，则组件可以接收焦点和输入；如果值为 `false`，则不能接收。默认值为 `true`。

范例

以下范例将 CheckBox 组件的 `enabled` 属性设置为 `false`：

```
checkBoxInstance.enabled = false;
```

UIComponent.getFocus()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.getFocus();
```

参数

无。

返回

对当前具有焦点的对象的引用。

描述

方法；返回对具有键盘焦点的对象的引用。

范例

下列代码返回对具有焦点的对象的引用，并将它分配给 `tmp` 变量：

```
var tmp = checkbox.getFocus();
```

UIComponent.keyDown

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
on(keyDown){
  ...
}
listenerObject = new Object();
listenerObject.keyDown = function(eventObject){
  ...
}
componentInstance.addEventListener("keyDown", listenerObject)
```

描述

事件；当某个按键被按下时通知侦听器。这是一个级别很低的事件，除非必要，否则不应使用，因为它可能会影响系统性能。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （在本例中为 keyDown），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下代码会在某个按键被按下时使图标闪烁：

```
formListener.handleEvent = function(eventObj)
{
  form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyDown", formListener);
```

UIComponent.keyUp

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
on(keyUp){
  ...
}
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
  ...
}
componentInstance.addEventListener("keyUp", listenerObject)
```

描述

事件；当某个按键被松开时通知侦听器。这是一个级别很低的事件，除非必要，否则不应使用，因为它可能会影响系统性能。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 keyUp），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下代码在某个按键被松开时使图标闪烁：

```
formListener.handleEvent = function(eventObj)
{
  form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyUp", formListener);
```

UIComponent.setFocus()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.setFocus();
```

参数

无。

返回

无。

描述

方法；设置此组件实例的焦点。具有焦点的实例接收所有的键盘输入。

范例

下列代码将 `checkbox` 实例设置为具有焦点：

```
checkbox.setFocus();
```

UIComponent.tabIndex

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

instance.tabIndex

描述

属性；指明文档中组件的 Tab 键顺序的值。

范例

下列代码将 tmp 的值设置为 checkbox 实例的 tabIndex 属性：

```
var tmp = checkbox.tabIndex;
```

UIEventDispatcher 类

动作脚本类命名空间 mx.events.EventDispatcher ； mx.events.UIEventDispatcher

通过事件可以了解用户何时与组件进行了交互操作，也可以了解组件的外观或生命周期何时发生了重要的更改，例如创建或破坏组件或者组件的大小发生更改。

每个组件都会播放不同的事件，而且这些事件会列在每个组件项目中。在 “动作脚本” 代码中使用组件事件的方法有若干种。有关详细信息，请参阅第 21 页的 “关于组件事件”。

使用 UIEventDispatcher.addEventListener() 将侦听器注册到组件实例。触发组件事件时，侦听器会被调用。

UIEventDispatcher.addEventListener()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004 和 Flash MX Professional 2004。

用法

componentInstance.addEventListener(*event*, *listener*)

参数

event 表示事件名称的字符串。

listener 对侦听器对象或函数的引用。

返回

无。

描述

方法；在播放事件的组件实例上注册侦听器对象。事件触发时，将通知侦听器对象或函数。您可以从任何组件实例调用此方法。例如，下列代码将侦听器注册到组件实例 myButton：

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

在调用 addEventListener() 将侦听器注册到组件实例之前，您必须先将该侦听器定义为对象或函数。如果侦听器是对象，则必须定义一个回调函数，当触发该事件时，将会调用该回调函数。通常，回调函数与注册侦听器所使用的事件同名。如果侦听器是函数，则触发事件时，将调用该函数。有关详细信息，请参阅第 22 页的 “使用组件事件侦听器”。

您可以将多个侦听器注册到一个组件实例，但必须为每个侦听器单独调用 addEventListener()。而且，也可以将一个侦听器注册到多个组件实例，但必须为每个实例单独调用 addEventListener()。例如，下列代码定义一个侦听器对象，并将它分配给两个 Button 组件实例：

```
lo = new Object();
lo.click = function(evt){
  if (evt.target == button1){
    trace("button 1 clicked");
  } else if (evt.target == button2){
    trace("button 2 clicked");
  }
}
button1.addEventListener("click", lo);
button2.addEventListener("click", lo);
```

事件对象作为参数传递给侦听器。事件对象具有包含有关所发生事件的信息的属性。您可以在侦听器回调函数内使用事件对象来访问有关所发生的事件类型的信息，以及哪个实例广播该事件的信息。在上面的范例中，事件对象是 evt（您可以将任何标识符用作事件对象名称），它在 if 语句内使用，用于确定单击了哪个按钮实例。有关详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例定义了一个侦听器对象 myListener，并定义了回调函数 click。然后，它调用 addEventListener() 将 myListener 侦听器对象注册到组件实例 myButton。要测试该代码，请将实例名为 myButton 的按钮组件放在舞台上，并将下列代码置于第一帧中：

```
myListener = new Object();
myListener.click = function(evt){
  trace(evt.type + " triggered");
}
myButton.addEventListener("click", myListener);
```

事件对象

事件对象作为参数传递到侦听器。事件对象是一种动作脚本对象，这种对象具有的属性中包含有关所发生的事件的信息。您可以在侦听器回调函数内使用事件对象来访问所广播的事件的名称，或者访问广播该事件的组件的实例名称。例如，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：

```
listener = new Object();
listener.click = function(evtObj){
  trace("The " + evtObj.target.label + " button was clicked");
}
myButton.addEventListener("click", listener);
```

一些事件对象的属性在 W3C 规范 中进行了定义，但在 Macromedia Component Architecture 第 2 版 (v2) 中没有实现。下表中列出每个 v2 事件对象所具有的属性。一些事件还定义有其他属性，如果是这样的话，这些属性将在该事件的条目中列出。

事件对象的属性

属性	描述
type	指明事件名称的字符串。
target	对广播事件的组件实例的引用。

UIObject 类

继承 MovieClip > UIObject

动作脚本类命名空间 mx.core.UIObject

UIObject 是所有第 2 版组件的基类；它是不可视组件。UIObject 类环绕 “动作脚本” MovieClip 对象，并包含允许 Macromedia v2 组件共享某些常用行为的函数和属性。UIObject 类实现了以下内容：

- 样式
- 事件
- 按缩放比例调整大小

要使用 UIObject 的方法和属性，请直接从正在使用的任何组件中调用它们。例如，要从 RadioButton 组件调用 `UIObject.setSize()` 方法，您要编写以下代码：

```
myRadioButton.setSize(30, 30);
```

如果您使用 Macromedia Component V2 Architecture 创建新组件，则只需创建一个 UIObject 实例。即使在这种情况下，UIObject 也经常由其他子类（例如 Button）隐式创建。如果您确实需要创建一个 UIObject 实例，请使用下列代码：

```
class MyComponent extends UIObject;
```

UIObject 类的方法摘要

方法	描述
`UIObject.createObject()`	创建对象的子对象。
`UIObject.createClassObject()`	创建指定类的对象。
`UIObject.destroyObject()`	破坏组件实例。
`UIObject.invalidate()`	标记对象以便在到达下一个帧间隔时进行重绘。
`UIObject.move()`	将对象移动到要求的位置。
`UIObject.redraw()`	迫使对象有效以便能在当前帧中绘制。
`UIObject.setSize()`	将对象调整为所要求的大小。
`UIObject.setSkin()`	设置对象的外观。

UIObject 类的属性摘要

属性	描述
UIObject.bottom	返回对象底边位置 （相对于其父对象的底边）。
UIObject.height	对象的高度 （以像素为单位）。
UIObject.left	对象的左侧位置 （以像素为单位）。
UIObject.right	对象的右侧位置 （相对于其父对象的右边）。
UIObject.scaleX	一个数字，指明对象相对于其父对象在 x 方向上的缩放系数。
UIObject.scaleY	一个数字，指明对象相对于其父对象在 y 方向上的缩放系数。
UIObject.top	对象上边缘的位置 （相对于其父对象）。
UIObject.visible	一个布尔值，指明对象是可见的 (true) 还是不可见的 (false)。
UIObject.width	对象的宽度 （以像素为单位）。
UIObject.x	对象的左侧位置 （以像素为单位）。
UIObject.y	返回对象上边缘的位置 （相对于其父对象）。

UIObject 类的事件摘要

事件	描述
UIObject.draw	当对象将要绘制它的图形时进行广播。
UIObject.load	创建子对象时广播。
UIObject.move	移动了对象时广播。
UIObject.resize	卸载子对象时广播。
UIObject.unload	卸载子对象时广播。

UIObject.bottom

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.bottom

描述

属性 （只读）；一个数字，指明对象底边相对于其父对象底边的位置 （以像素为单位）。

范例

将 tmp 的值设置为复选框的底边位置：

```
var tmp = checkbox.bottom;
```

UIObject.createObject()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.createObject(linkageName, instanceName, depth, initObject)
```

参数

linkageName 一个字符串，指明 “库”面板中元件的链接标识符。

instanceName 一个字符串，指明新实例的实例名称。

depth 一个数字，指明新实例的深度。

initObject 一个对象，它包含了新实例的初始化属性。

返回

UIObject，是一个元件的实例。

描述

方法；创建对象的子对象。通常只由组件或高级开发人员使用。

范例

下面的范例在 `form` 对象上创建了一个 CheckBox 实例：

```
form.createObject("CheckBox", "sym1", 0);
```

UIObject.createClassObject()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.createClassObject(className, instanceName, depth, initObject)
```

参数

className 一个对象，指明新实例的类。

instanceName 一个字符串，指明新实例的实例名称。

depth 一个数字，指明新实例的深度。

initObject 一个对象，它包含了新实例的初始化属性。

返回

一个 UIObject，它是所指定类的一个实例。

描述

方法；创建对象的子对象。通常只由组件或高级开发人员使用。使用该方法，您可以在运行时创建组件。

您需要指定类包名称。请执行以下操作之一：

```
import mx.controls.Button;
createClassObject(Button,"button2",5,{label:"Test Button"});
```

或者

```
createClassObject(mx.controls.Button,"button2",5,{label:"Test Button"});
```

范例

下面的范例创建了一个 CheckBox 对象：

```
form.createClassObject(CheckBox, "cb", 0, {label:"Check this"});
```

UIObject.destroyObject()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.destroyObject(*instanceName*)

参数

instanceName 一个字符串，指明要破坏的对象的实例名称。

返回

无。

描述

方法；破坏组件实例。

UIObject.draw

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
on(draw){
  ...
}
listenerObject = new Object();
listenerObject.draw = function(eventObject){
  ...
}
componentInstance.addEventListener("draw", listenerObject)
```

描述

事件；通知侦听器对象将要绘制它的图形。这是一个级别很低的事件，除非必要，否则不应使用，因为它可能会影响系统性能。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （本例中为 draw），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下代码在 `form` 对象被绘制时，重绘对象 `form2`：

```
formListener.draw = function(eventObj){
  form2.redraw(true);
}
form.addEventListener("draw", formListener);
```

另请参见

`UIEventDispatcher.addEventListener()`

UIObject.height

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.height
```

描述

属性 （只读）；指明对象高度的数字 （以像素为单位）。

范例

下面的范例将 tmp 的值设为 checkbox 实例的高度：

```
var tmp = checkbox.height;
```

UIObject.getStyle()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.getStyle(propertyName)
```

参数

propertyName 一个字符串，它指明样式属性的名称（例如，"fontWeight"、"borderStyle"，等等）。

返回

样式属性的值。这些值可以是任何数据类型：

描述

方法；从 styleDeclaration 或对象中获取样式属性。如果样式属性是继承性样式，那么该对象的父级就可能会是样式值的源。

有关各组件所支持的样式的列表，请参阅其各自所属的条目。

范例

如果 cb 实例的 fontWeight 样式属性是粗体，下面的代码会将 ib 实例的 fontWeight 样式属性设置为粗体：

```
if (cb.getStyle("fontWeight") == "bold")
{
   ib.setStyle("fontWeight", "bold");
};
```

UIObject.invalidate()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.invalidate()
```

参数

无。

返回

无。

描述

方法；标记对象以便在下一帧时间间隔时进行重绘。

范例

下面的范例将标记 ProgressBar 实例 `pBar` 以便进行重绘：

```
pBar.invalidate();
```

UIObject.left

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.left
```

描述

属性（只读）；一个数字，它指明对象的左边缘（以像素为单位）。

范例

下面的范例将 `tmp` 的值设为 `checkbox` 实例的左侧位置：

```
var tmp = checkbox.left; // 将 tmp 的值设为复选框的左侧位置；
```

UIObject.load

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(load){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.load = function(eventObject){
  ...
}
componentInstance.addEventListener("load", listenerObject)
```

描述

事件；通知侦听器正在创建该对象的子对象。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件（在本例中为 load），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例将在加载 `form` 实例后创建一个 MySymbol 实例。

```
formListener.handleEvent = function(eventObj)
{
  form.createObject("MySymbol", "sym1", 0);
}
form.addEventListener("load", formListener);
```

UIObject.move

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(move){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.move = function(eventObject){
  ...
}
componentInstance.addEventListener("move", listenerObject)
```

描述

事件；通知侦听器对象已移动。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （在本例中为 move），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

下面的范例调用 `move()` 方法，以使 `form2` 相对于 `form1` 向下和向右移动 100 个像素：

```
formListener.handleEvent = function(eventObj)
{
  // eventObj.target 是生成更改事件的组件
  form2.move(form1.x + 100, form1.y + 100);
}
form1.addEventListener("move", formListener);
```

UIObject.move()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.move(*x*, *y*)

参数

x 一个数字，指明对象的左上角相对于其父对象的位置。

y 一个数字，指明对象的左上角相对于其父对象的位置。

返回

无。

描述

方法；将对象移动到要求的位置。您应该只将整数值传递给 `UIObject.move()`，否则，组件可能会模糊不清。

如果不遵循这些规则，很可能会导致控件的外观十分模糊。

范例

此范例将 ProgressBar 实例 `pBar` 移动到左上角的 (100, 100) 位置：

```
pBar.move(100, 100);
```

UIObject.redraw()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.redraw()
```

参数

always 一个布尔值；`true` 表示总是重绘，`false` 表示只在失效时才重绘。

返回

无。

描述

方法；迫使对象有效以便在当前帧中绘制。

范例

此范例迫使 `pBar` 实例立即重绘：

```
pBar.validate(true);
```

UIObject.resize

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(resize){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.resize = function(eventObject){
  ...
}
componentInstance.addEventListener("resize", listenerObject)
```

描述

事件；通知侦听器正在卸载该对象的子对象。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （在本例中为 resize），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 `UIEventDispatcher.addEventListener()` 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

在以下范例中，当移动 `form` 时，将调用 `setSize()` 方法，以使 `sym1` 的宽度变为原来的二分之一，高度变为原来的四分之一：

```
formListener.handleEvent = function(eventObj) {
  form.sym1.setSize(sym1.width / 2, sym1.height / 4);
}
form.addEventListener("resize", formListener);
```

UIObject.right

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.right

描述

属性 （只读）；一个数字，指明对象的右侧相对于其父对象右侧的位置 （以像素为单位）。

范例

以下范例将 `tmp` 的值设置为 `checkbox` 实例的右侧位置：

```
var tmp = checkbox.right;
```

UIObject.scaleX

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.scaleX

描述

属性；一个数字，指明对象相对于其父对象在 x 方向上的缩放系数。

范例

以下范例将复选框的宽度放大两倍并将 `tmp` 变量设置为水平缩放系数：

```
checkbox.scaleX = 200;
var tmp = checkbox.scaleX;
```

UIObject.scaleY

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.scaleY

描述

属性；一个数字，指明对象相对于其父对象在 y 方向上的缩放系数。

范例

以下范例将复选框的高度放大两倍并将 `tmp` 变量设置为垂直缩放系数：

```
checkbox.scaleY = 200;
var tmp = checkbox.scaleY;
```

UIObject.setSize()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.setSize(*width*, *height*)

参数

width 一个数字，指明对象的宽度（以像素为单位）。

height 一个数字，指明对象的高度（以像素为单位）。

返回

无。

描述

方法；将对象调整到要求的大小。您应该只将整数值传递给 `UIObject.setSize()`，否则，组件可能会模糊不清。该方法（以及 UIObject 的所有方法和属性）可以从任何组件实例中使用。

对 ComboBox 的实例调用此方法时，将会调整组合框的大小，而且所包含列表的 `rowHeight` 属性也会更改。

范例

此范例将 pBar 组件实例的大小调整为宽 100 个像素，高 100 个像素：

```
pBar.setSize(100, 100);
```

UIObject.setSkin()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.setSkin(id, linkageName)
```

参数

id 一个数字，指明变量。该值通常为在类定义中定义的一个常数。

linkageName 一个字符串，指明库中的一个资源。

返回

无。

描述

方法；设置组件实例中的外观。在开发组件时可以使用此方法。在运行时不能使用此方法来设置组件的外观。

范例

本范例设置 checkbox 实例中的一个外观：

```
checkbox.setSkin(CheckBox.skinIDCheckMark, "MyCustomCheckMark");
```

UIObject.setStyle()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.setStyle(propertyName, value)
```

参数

propertyName 一个字符串，它指明样式属性的名称（例如，"fontWeight"、"borderStyle"，等等）。

value 属性的值。

返回

一个 UIObject，它是所指定类的一个实例。

描述

方法；设置样式声明或对象的样式属性。如果样式属性是继承的样式，则会将新值通知给该对象的子对象。

有关各组件所支持的样式的列表，请参阅其各自所属的条目。

范例

下面的代码将复选框实例 `cb` 的样式属性 `fontWeight` 设置为粗体：

```
cb.setStyle("fontWeight", "bold");
```

UIObject.top

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.top
```

描述

属性（只读）；一个数字，指明对象的上边缘（以像素为单位）。

范例

以下范例将 `tmp` 变量设置为 `checkbox` 实例的顶部位置：

```
var tmp = checkbox.top; // 将 tmp 的值设置为复选框的顶部位置；
```

UIObject.unload

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(unload){
   ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.unload = function(eventObject){
  ...
}
componentInstance.addEventListener("unload", listenerObject)
```

描述

事件；通知侦听器正在卸载该对象的子对象。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个组件实例。

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*componentInstance*) 调度一个事件 （在本例中为 unload），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。每个事件对象都有一组属性，这些属性包含有关该事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下范例会在 unload 事件被触发时删除 sym1：

```
formListener.handleEvent = function(eventObj) {
  // eventObj.target 是生成 change 事件的组件，
  form.destroyObject(sym1);
}
form.addEventListener("unload", formListener);
```

UIObject.visible

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
componentInstance.visible
```

描述

属性；一个布尔值，指明对象是可见的 (true) 还是不可见的 (false)。

范例

下面的范例使 myLoader 加载器实例可见：

```
myLoader.visible = true;
```

UIObject.width

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.width

描述

属性 （只读）；一个数字，指明对象的宽度 （以像素为单位）。

范例

下面的范例将 TextArea 组件的宽度设置为 450 个像素：

```
mytextarea.width = 450;
```

UIObject.x

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.x

描述

属性 （只读）；一个数字，它指明对象的左边缘 （以像素为单位）。

范例

下面的范例将复选框的左边缘设置为 150 个像素：

```
checkbox.x = 150;
```

UIObject.y

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

componentInstance.y

描述

属性 （只读）；一个数字，指明对象的上边缘 （以像素为单位）。

范例

下面的范例将复选框的上边缘设置为 200 个像素：

```
checkbox.y = 200;
```

WebServices 包

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

WebServiceConnector 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

Window 组件

Window 组件在一个具有标题栏、边框和关闭按钮 （可选）的窗口内显示电影剪辑的内容。

Window 组件可以是模式的，也可以是非模式的。模式窗口会防止鼠标和键盘输入转至该窗口之外的其他组件。 Window 组件还支持拖动操作；用户可以单击标题栏并将窗口及其内容拖动到另一个位置。拖动边框不会更改窗口的大小。

如果使用 PopUpManager 向文档添加 Window 组件，则该 Window 实例将具有它自己的 FocusManager，与文档其他对象所具有的 FocusManager 不同。如果不使用 PopUpManager，窗口的内容会参与焦点排序。有关控制焦点的详细信息，请参阅第 23 页的 “创建自定义焦点导航” 或第 93 页的 “FocusManager 类”。

每个 Window 实例的实时预览反映在创作过程中对属性检查器或 “组件检查器” 面板中的所有参数所做的更改，但 contentPath 除外。

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

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

不管一个组件有多少实例，都只对其启用一次辅助功能。有关详细信息，请参阅 “使用 Flash” 帮助中的 “创建具有辅助功能的内容”。您可能需要更新帮助系统才能看到此信息。

使用 Window 组件

无论何时您需要向用户提供信息或最优先的选择时，您都可以在应用程序中使用一个窗口。例如，您可能会需要用户填写登录窗口或者发生了更改并需要确认新密码的窗口。

将窗口添加到应用程序有几种方式。您可以将窗口从 “组件” 面板拖动到舞台。您也可以通过调用 `createClassObject()` （请参阅 `UIObject.createClassObject()`）将窗口添加到应用程序。将窗口添加到应用程序的第三种方法是使用 PopUpManager 类。使用 PopUpManager 可以创建与舞台上其他对象重叠的模式窗口。有关详细信息，请参阅 Window 类。

Window 组件参数

以下是您可以在属性检查器中或在 “组件检查器” 面板中为每个 Window 组件设置的创作参数：

contentPath 指定窗口的内容。这可以是电影剪辑的链接标识符，或者是屏幕、表单或包含窗口内容的幻灯片的元件的名称。它也可以是要加载到窗口的 SWF 或 JPG 文件的绝对或相对 URL。默认值为 ""。加载的内容会被裁剪，以适合窗口大小。

title 指明窗口的标题。

closeButton 指明是 (true) 否 (false) 显示关闭按钮。单击关闭按钮会广播一个 `click` 事件，但不关闭窗口。您必须编写调用 `Window.deletePopUp()` 的处理函数，以显式关闭窗口。有关 `click` 事件的详细信息，请参阅 `Window.click`。

您可以编写 “动作脚本”，使用其属性、方法和事件来控制 Window 组件的这些和其他选项。有关详细信息，请参阅 Window 类。

创建具有 Window 组件的应用程序

以下过程解释了如何将 Window 组件添加到应用程序。在本范例中，窗口会要求用户更改其密码并确认新密码。

要创建具有 Button 组件的应用程序，请执行以下操作：

1 新建一个影片剪辑，其中包含密码和密码确认字段，以及 “确定” 和 “取消” 按钮。将该影片剪辑命名为 **PasswordForm**。

2 在库中，选择 PasswordForm 影片剪辑并从 “选项” 菜单中选择 “链接”。

3 选中 “为动作脚本导出” 并在 “标识符” 框中输入 **PasswordForm**。

4 在类字段中输入 **mx.core.View**。

5 将一个 Window 组件从 “组件” 面板中拖动到舞台上，然后从舞台上删除该组件。这会将该组件添加到库中。

6 在库中，选择 Window SWC 并从 “选项” 菜单中选择 “链接”。

7 选中 “为动作脚本导出”。

8 打开 “动作” 面板，然后在第一帧上输入下列 click 处理函数：

```
buttonListener = new Object();
buttonListener.click = function(){
  mx.managers.PopUpManager.createPopUp(_root, mx.containers.Window, true, {
  title:"Change Password", contentPath:"PasswordForm" })
}
button.addEventListener("click", buttonListener);
```

该处理函数调用 `PopUpManager.createPopUp()` 以实例化标题栏为 “更改密码” 的 Window 组件，该组件显示 PasswordForm 影片剪辑的内容。

自定义 Window 组件

在创作过程中和在运行时，您都可以在水平和垂直方向上改变 Window 组件的形状。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，请使用 `UIObject.setSize()` 或 Window 类中任何适用的属性和方法。有关详细信息，请参阅 Window 类。

调整窗口大小不会更改关闭按钮或标题题注的大小。标题题注为左对齐，关闭栏为右对齐。

对 Window 组件使用样式

Window 组件标题栏的样式声明由 `Window.titleStyleDeclaration` 属性指明。

Window 组件支持下列光晕样式：

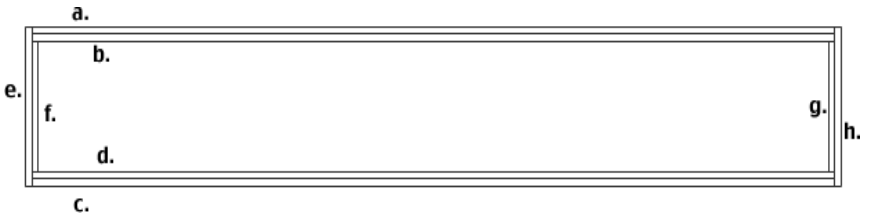
样式	描述
borderStyle	组件边框；其值为 “none”、“inset”、“outset” 或 “solid”。此样式不继承其值。

对 Window 组件使用外观

Window 组件使用 RectBorder 类，该类使用“动作脚本”绘图 API 来绘制其边框。您可以使用 `setStyle()` 方法 （请参阅 `UIObject.setStyle()`）来更改下列 RectBorder 样式属性：

RectBorder 样式
`borderColor`
`highlightColor`
`borderColor`
`shadowColor`
`borderCapColor`
`shadowCapColor`
`shadowCapColor`
`borderCapColor`

这些样式属性设置边框上的下列位置：



如果使用 `UIObject.createClassObject()` 或 `PopUpManager.createPopUp()` 动态 （在运行时）创建 Window 实例，则也可以动态设计其外观。要在运行时设计组件的外观，请设置传递给 `createClassObject()` 方法的 *`initObject`* 参数的外观属性。这些外观属性设置用作按钮状态的元件的名称，元件可以带有图标，也可以没有图标。有关详细信息，请参阅 `UIObject.createClassObject()` 和 `PopUpManager.createPopUp()`。

Window 组件使用下列外观属性：

属性	描述
`skinTitleBackgroud`	标题栏。默认值为 `TitleBackground`。
`skinCloseUp`	关闭按钮。默认值为 `CloseButtonUp`。
`skinCloseDown`	处于按下状态的关闭按钮。默认值为 `CloseButtonDown`。
`skinCloseDisabled`	处于禁用状态的关闭按钮。默认值为 `CloseButtonDisabled`。
`skinCloseOver`	处于悬停状态的关闭按钮。默认值为 `CloseButtonOver`。

Window 类

继承 UIObject > UIComponent > View > ScrollView > Window

动作脚本类命名空间 mx.containers.Window

Window 类的属性允许您在运行时设置标题题注、添加关闭按钮以及设置显示内容。 使用动作脚本设置 Window 类的属性会覆盖在属性检查器或 “组件检查器”面板中设置的同名参数。

实例化窗口的最佳方法是调用 PopUpManager.createPopUp()。此方法既可创建模式窗口（重叠并禁用应用程序中的现有对象），也可创建非模式窗口。例如，下面的代码创建模式 Window 实例（最后一个参数指明是模式窗口）：

```
var newWindow = PopUpManager.createPopUp(this, Window, true);
```

形态是通过在 Window 组件下方创建一个大的透明窗口模拟的。由于透明窗口的呈现方式，您可能会注意到透明窗口下的对象略显暗淡。有效透明度可以进行设置，方法是：更改 _global.style.modalTransparency 值，范围为从 0（完全透明）到 100（不透明）。如果使窗口部分透明，还可以设置窗口的颜色，方法是：在默认主题中更改“模式”外观。

如果使用 PopUpManager.createPopUp() 创建模式窗口，则在删除时，必须调用 Window.deletePopUp()，以便也可以删除透明窗口。例如，如果在窗口上使用 closeButton，则要编写下列代码：

```
obj.click = function(evt){
   this.deletePopUp();
}
window.addEventListener("click", obj);
```

注意：代码不会在创建模式窗口时停止执行。在其他环境（例如 Microsoft Windows）中，如果创建一个模式窗口，则创建窗口之后的代码行在窗口关闭之前不会运行。在 Flash 中，这些代码行在创建窗口之后、关闭窗口之前运行。

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

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

注意：下面的代码返回未定义的：trace(myWindowInstance.version);。

Window 类的方法摘要

方法	描述
Window.deletePopUp()	删除由 PopUpManager.createPopUp() 创建的窗口实例。

继承 UIObject 类、UIComponent 类和 View 的所有方法。

Window 类的属性摘要

属性	描述
Window.closeButton	指明标题栏上是 (true) 否 (false) 包含关闭按钮。
Window.content	对在 contentPath 属性中指定的内容的引用。
Window.contentPath	在窗口中所显示内容的路径。
Window.title	标题栏中显示的文本。
Window.titleStyleDeclaration	设置标题栏中文本格式的样式声明。

继承 UIObject 类、UIComponent 类和 ScrollView 的所有属性。

Window 类的事件摘要

事件	描述
Window.click	松开关闭按钮时触发。
Window.mouseDownOutside	在模式窗口外按下鼠标时触发。

继承 UIObject 类、 UIComponent 类、 View 和 ScrollView 的所有事件。

Window.click

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

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

用法 2：

```
listenerObject = new Object();
listenerObject.click = function(eventObject){
  ...
}
windowInstance.addEventListener("click", listenerObject)
```

描述

事件；在关闭按钮上单击 （松开）鼠标时向所有已注册的侦听器广播。

第一个用法范例使用一个 on() 处理函数，并且必须直接附加到一个 Window 组件实例。附加到组件的 on() 处理函数内部使用的关键字 this 是指该组件实例。例如，下列代码附加到 Window 组件实例 myWindow，它将 “_level0.myWindow” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*windowInstance*) 调度一个事件 （在本例中为 click），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象有一组属性，这些属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下范例创建一个模式窗口，然后定义一个删除此窗口的 click 处理函数：您必须向舞台添加一个 Window 组件，然后删除它以将其添加到文档库中，随后将下列代码添加到第 1 帧：

```
import mx.managers.PopUpManager
import mx.containers.Window
var myTW = PopUpManager.createPopUp(_root, Window, true, {closeButton:true,
  title:"My Window"});
windowListener = new Object();
windowListener.click = function(evt){
  _root.myTW.deletePopUp();
}
myTW.addEventListener("click", windowListener);
```

另请参见

UIEventDispatcher.addEventListener(), Window.closeButton

Window.closeButton

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

windowInstance.closeButton

描述

属性；一个布尔值，指明标题栏是 (true) 否 (false) 应包含一个关闭铵钮。此属性必须在 PopUpManager.createPopUp() 方法的 *initObject* 参数中设置。默认值为 false。

范例

下列代码创建一个窗口，该窗口显示影片剪辑 “LoginForm” 中的内容，其标题栏上有一个关闭按钮：

```
var myTW = PopUpManager.createPopUp(_root, Window, true,
  {contentPath:"LoginForm", closeButton:true});
```

另请参见

Window.click, PopUpManager.createPopUp()

Window.content

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

windowInstance.content

描述

属性；对窗口的内容（根影片剪辑）的引用。此属性返回一个 MovieClip 对象。从库中附加一个元件时，默认值为所附加元件的一个实例。从 URL 加载内容时，直到开始加载操作，才会定义默认值。

范例

设置窗口组件内部的内容中的文本属性值：

```
loginForm.content.password.text = "secret";
```

Window.contentPath

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
windowInstance.contentPath
```

描述

属性；设置要在窗口中显示的内容的名称。该值可以是库中的一个影片剪辑的链接标识符，也可以是要加载的 SWF 或 JPG 文件的绝对或相对 URL。默认值为 "" （空字符串）。

范例

下列代码创建一个 Window 实例，该实例显示链接标识符为 “loginForm” 的影片剪辑：

```
var myTW = PopUpManager.createPopUp(_root, Window, true,
  {contentPath:"LoginForm"});
```

Window.deletePopUp()

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

```
windowInstance.deletePopUp();
```

参数

无。

返回

无。

描述

方法；删除窗口实例和移除模式状态。只能对由 `PopUpManager.createPopUp()` 创建的窗口实例调用此方法。

范例

以下代码创建一个模式窗口，然后创建一个侦听器（单击关闭按钮时该侦听器将删除此窗口）：

```
var myTW = PopUpManager.createPopUp(_root, Window, true);
twListener = new Object();
twListener.click = function(){
  myTW.deletePopUp();
}
myTW.addEventListener("click", twListener);
```

Window.mouseDownOutside

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

用法 1：

```
on(mouseDownOutside){
  ...
}
```

用法 2：

```
listenerObject = new Object();
listenerObject.mouseDownOutside = function(eventObject){
  ...
}
windowInstance.addEventListener("mouseDownOutside", listenerObject)
```

描述

事件；在模式窗口外部单击（松开）鼠标时向所有已注册的侦听器广播。此事件很少使用，但如果用户尝试与窗口之外的内容进行交互时，您可以用它退出窗口。

第一个用法范例使用一个 `on()` 处理函数，并且必须直接附加到一个 Window 组件实例。附加到组件的 `on()` 处理函数内部使用的关键字 `this` 是指该组件实例。例如，下列代码附加到 Window 组件实例 `myWindowComponent`，它将 “_level0.myWindowComponent” 发送到 “输出” 面板：

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

第二个用法范例使用一个调度程序 / 侦听器事件模型。组件实例 (*windowInstance*) 调度一个事件 （在本例中为 mouseDownOutside），而该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。该事件对象有一组属性，这些属性包含有关事件的信息。您可以使用这些属性来编写处理该事件的代码。最后，对广播该事件的组件实例调用 UIEventDispatcher.addEventListener() 方法，以将侦听器注册到实例。当该实例调度该事件时，就会调用该侦听器。

有关事件对象的详细信息，请参阅第 217 页的 “事件对象”。

范例

以下范例创建一个窗口实例并定义一个 mouseDownOutside 处理函数，当用户单击窗口外部时，该处理函数调用一个 beep() 方法：

```
var myTW = PopUpManager.createPopUp(_root, Window, true, undefined, true);
// 创建一个侦听器
twListener = new Object();
twListener.mouseDownOutside = function()
{
    beep(); // 当用户单击外部时发出噪音
}
myTW.addEventListener("mouseDownOutside", twListener);
```

另请参见

UIEventDispatcher.addEventListener()

Window.title

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

windowInstance.title

描述

属性；一个字符串，指明标题栏的题注。默认值为 "" （空字符串）。

范例

下列代码将窗口的标题设置为 “Hello World”：

```
myTW.title = "Hello World";
```

Window.titleStyleDeclaration

可用性

Flash Player 6.0.79。

版本

Flash MX 2004。

用法

windowInstance.titleStyleDeclaration

描述

属性；一个字符串，指明设置窗口标题栏格式的样式声明。默认值未定义，指明粗体、白色文本。

范例

下列代码创建一个窗口，此窗口显示链接标识符为 “changePassword” 的影片剪辑的内容并使用 CSSStyleDeclaration “MyTWStyles”：

```
var myTW = PopUpManager.createPopUp(_root, Window, true,
  {contentPath:"LoginForm",
                                    titleStyleDeclaration:"MyTWStyles"});
```

有关样式的详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

XMLConnector 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

XUpdateResolver 组件

有关此功能的最新信息，请单击 “帮助” 选项卡顶部的 “更新” 按钮。

第 5 章
创建组件

本章说明了如何创建您自己的组件，让其他开发人员能够使用这些组件，以及将这些组件打包以便进行部署。

新增功能

Macromedia Component Architecture 的最新版本（第 2 版）与 Macromedia Flash MX 版本（第 1 版）有着很大的不同。 Macromedia 做了一些更改，旨在提高开发人员所用组件的可缩放性、性能和可扩展性。下面的列表概要说明其中的某些更改：

- 识别动作脚本元数据的 “组件检查器” 面板
- 可扩展的管理器和基类
- 内置 “实时预览”
- 改进了编译器消息
- 新增事件模型
- 焦点管理
- 基于 CSS 的样式

在 Flash 环境中工作

设置 Macromedia Flash MX 2004 和 Flash MX Professional 2004 环境的目的是使类和组件的结构具备逻辑性。本节说明了应在何处存储组件文件。

FLA 文件资源

在创建组件时，首先创建 FLA 文件，然后添加外观、图形和其他资源。这些资源可以存储在 FLA 文件中的任何位置，因为 Flash 组件用户只需要编译后的组件文件，而不需要原始资源。

在 Flash 中创建组件时，应使用具备两个帧、两个图层的 SWF 文件。第一个图层是动作图层，它指向组件的动作脚本类文件。第二个图层是资源图层，它包含由组件使用的图形、元件及其他资源。

类文件

FLA 文件包含对组件的动作脚本类文件的引用。这就是所谓的将组件绑定到类文件。

动作脚本代码指定组件的属性和方法，并且，如果存在供组件继承的类，它还会定义组件从哪些类继承。对于动作脚本的源代码，必须使用 *.as 的文件命名惯例，并且以组件自身的名称来对源代码文件命名。例如， MyComponent.as 包含 MyComponent 组件的源代码。

Flash MX 2004 核心类 .as 文件位于称为 Classes/mx/Core 的单个文件夹中。其他动作脚本类文件按包名称分组，分别位于 /Classes 下各自的文件夹中。

对于自定义组件，请在 /Classes 下新建一个目录，然后将动作脚本类文件存储在该目录中。

类路径

本节对 Flash 类路径进行说明。

了解类路径

类路径是目录的编号列表，Flash 在组件的导出或 SWF 文件的生成过程中搜索该列表，以查找类文件。类路径中各条目的顺序非常重要，因为 Flash 根据先进先服务的原则使用类。在导出时，类路径中与 FLA 文件内的链接标识符相匹配的类被导入 FLA 文件，并向它们的元件注册。

全局类路径适用于 Flash 生成的所有 FLA 文件。本地类路径只适用于当前的 FLA 文件。

默认的本地类路径为空。默认的全局类路径包含两个路径：

- $(UserConfig)/Classes
- .

点 (.) 表示当前的工作目录。 Flash 在 FLA 文件的当前目录中搜索动作脚本类。

$(UserConfig)/Classes 路径表示每个用户的配置目录。此目录指向以下位置：

- 在 Windows 中，此目录为 C:\Documents and Settings\ *用户名* \Application Data\Macromedia\Flash MX 2004\en\Configuration。
- 在 Macintosh 中，此目录为 *volume*:Users:*username*:Library:Application Support:Macromedia:Flash MX 2004:en:configuration。

UserConfig 目录是 *Flash_root*/en/Configuration 中各目录的镜像。不过，类路径不直接包含这些目录，它是相对于 UserConfig 目录的路径。

更改类路径

您可以为单个 FLA 文件更改类路径 （本地类路径），也可以为在 Flash 中处理的所有 FLA 文件更改类路径 （全局路径）。

更改全局类路径：

1 选择 “编辑” > “首选参数”。

显示 “首选参数” 对话框。

2 选择 “动作脚本” 选项卡。

3 单击 “动作脚本 2.0 设置” 按钮。

屏幕上会出现 “动作脚本设置” 对话框。

4 在 “类路径” 框中添加、删除或编辑条目。

5 保存所做的更改。

更改本地类路径：

1 选择 “文件” > “发布设置”。

出现 “发布设置” 对话框。

2 选择 Flash 选项卡。

3 单击 “设置” 按钮。

屏幕上会出现 “动作脚本设置” 对话框。

4 在 “类路径” 框中添加、删除或编辑条目。

5 保存所做的更改。

查找组件源文件

在开发组件时，您可以将源文件存储在任何目录中。不过，为了确保 Flash 在导出组件时能够找到必要的类文件，必须在 Flash MX 2004 类路径设置中包含该目录。此外，如果要测试组件，必须将组件存储在 Flash Components 目录中。有关存储 SWC 文件的更多信息，请参阅第 266 页的 “使用 SWC 文件”。

编辑元件

每个元件都有自己的时间轴。您可以给元件时间轴添加帧、关键帧和图层，就像可以给主时间轴添加这些项目一样。

创建组件时，要从元件开始。 Flash 提供了以下三种编辑元件的方法：

- 使用 “在当前位置编辑” 命令，在舞台上其他对象的上下文中编辑元件。其他对象显示暗淡，以便将它们与正在编辑的元件区分开。正在编辑的元件名称显示在舞台顶部的编辑栏内，位于当前场景名称的右侧。
- 使用 “在新窗口中编辑” 命令，在单独的窗口中编辑元件。如果在单独的窗口中编辑元件，您可以同时查看元件和主时间轴。正在编辑的元件名称显示在舞台顶部的编辑栏内。
- 在元件编辑模式下，通过将窗口从舞台视图更改为只显示该元件的单独视图来编辑元件。正在编辑的元件名称显示在舞台顶部的编辑栏内，位于当前场景名称的右侧。

组件代码范例

Flash MX 2004 和 Flash MX Professional 2004 包含以下组件源文件，它们有助于您开发自己的组件：

- FLA 文件源代码：*Flash MX 2004_install_dir*/en/First Run/ComponentFLA/StandardComponents.fla
- 动作脚本类文件：*Flash MX 2004_install_dir*/en/First Run/Classes/mx

创建组件

本节说明了如何创建作为现有 Flash MX 2004 类的子类的组件。随后的各节说明了如何编写组件的动作脚本类文件，以及如何针对组件的可用性和品质来编辑组件。

创建新组件元件

所有组件都是 MovieClip 对象，它们是一种元件。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 为该行为选择 “影片剪辑” 单选按钮。

“影片剪辑” 对象拥有自己的多帧时间轴，该时间轴独立于主时间轴播放。

5 单击 “高级” 按钮。

高级设置在对话框中显示。

6 选择 “为动作脚本导出”。这样就会让 Flash 默认将组件与使用该组件的任何 Flash 内容打包在一起。

7 输入链接标识符。

此标识符用作元件名称、链接名称和相关的类名称。

8 在 “AS 2.0 类” 文本框中，输入动作脚本 2.0 类的完全限定路径，它是相对于类路径设置的路径。

注意：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。

如果动作脚本文件位于包内，必须包含该包的名称。此字段的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）。

有关设置 Flash MX 2004 类路径的更多信息，请参阅第 246 页的 “了解类路径”。

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）。有关更多信息，请参阅第 267 页的 “设计组件的最佳做法”。

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，并且有一个十字丝表明该元件的注册点。

现在，您可以编辑该元件，并将它添加到组件的 FLA 文件。

编辑元件图层

在创建了新元件并为它定义链接之后，您可以在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。

有关如何编辑元件的信息，请参阅第 247 页的 “编辑元件”。

编辑元件图层：

1 进入元件编辑模式。

2 重命名空图层，或创建名为 Actions 的图层。

3 在 “动作”面板中添加一行，该行的作用是导入组件的完全限定动作脚本类文件。

该语句依赖 Flash MX 2004 类路径设置。（有关更多信息，请参阅第 246 页的 “了解类路径”。）以下范例导入 myPackage 包中的 MyComponent.as 文件：

```
import myPackage.MyComponent;
```

注意：在导入动作脚本类文件时，应使用 `import` 语句，而不应使用 `include` 语句。不要在类名称或包名称两边加引号。

4 重命名空图层，或创建名为 Assets 的图层。

Assets 图层包含由该组件使用的所有资源。

5 在 “动作”面板中，在第一帧内添加 `stop()` 动作，如以下范例所示：

```
stop();
```

不要向该帧添加任何图形资源。Flash Player 将在第二帧之前停止，您可以在第二帧内添加资源。

6 如果在扩展现有组件，请找到该组件和您使用的任何其他基类，然后将该元件的实例置于图层的第二帧内。为此，请从 “组件”面板选择元件，然后将它拖至组件的 “资产”图层第二帧中的舞台上。

组件使用的任何资源（不论它是另一个组件，还是诸如位图的介质）在该组件内都应有实例。

7 在组件的 Assets 图层的第二帧上添加由该组件使用的所有图形资源。例如，如果要创建自定义按钮，请添加表示按钮状态（弹起、按下，等等）的图形。

8 创建完元件内容后，执行下列操作之一，返回文档编辑模式：

- 单击舞台上方编辑栏左侧的 “返回”按钮。
- 选择 “编辑” > “编辑文档”。
- 单击舞台上方编辑栏内的场景名称。

添加参数

组件开发的下一步是定义组件参数。参数是用户据以修改您创建的组件实例的主要方法。

在 Flash 的以前版本中，应使用 “组件检查器”面板定义参数。在 Flash MX 2004 和 Flash MX Professional 2004 中，应在动作脚本类文件中定义参数，而 “组件检查器”面板负责找出公共参数并向用户显示这些参数。

下一节说明如何编写组件的外部动作脚本文件，其中包含有关添加组件参数的信息。

编写组件的动作脚本

大多数组件都会包含一些动作脚本代码。组件的类型决定着您编写动作脚本的位置和要编写的动作脚本的数量。开发组件有两种基本方法：

- 创建不带父类的新组件
- 扩展现有组件类

本节着重说明如何扩展现有组件。如果您在创建的组件源自其他组件的类文件，应该按照本节的说明编写外部动作脚本类文件。

扩展现有组件类

在创建源自父类的组件元件时，应将它链接到外部动作脚本 2.0 类文件。（有关定义此文件的信息，请参阅第 248 页的 “创建新组件元件”。）

外部动作脚本类为组件扩展其他类、添加方法、添加 getter 和 setter，以及定义事件处理函数。要编辑动作脚本类文件，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)。

只能从一个类继承。动作脚本 2.0 不允许多继承。

类文件的简单范例

下面是类文件的一个简单范例，该文件名为 MyComponent.as。此范例中包含从 UIObject 类继承的组件至少应有的一组导入、方法和声明。

```
import mx.core.UIObject;

  class myPackage.MyComponent extends UIObject {
  static var symbolName:String = "MyComponent";
  static var symbolOwner:Object = Object(myPackage.MyComponent);
  var className:String = "MyComponent";
  #include "../core/ComponentVersion.as"
  function MyComponent() {
  }
  function init(Void):Void{
    super.init();
  }
  function size(Void):Void {
    super.size();
  }
}
```

编写类文件的一般过程

在编写组件的动作脚本时，请使用下面的一般过程。根据所创建组件类型的不同，某些步骤是可选步骤。

本章后面会对此过程进行更详细的说明。

编写组件的动作脚本文件：

1 导入所有必需的类。

2 使用 `class` 关键字定义类；如有必要，请扩展父类。

3 定义 `symbolName` 和 `symbolOwner` 变量；它们分别是动作脚本类的元件名称和类的完全限定包名称。

4 将类名称定义为 `className` 变量。

5 添加版本控制信息。

6 输入默认成员变量。

7 为组件中使用的每个外观元素 / 链接创建变量。这样，用户就可以通过更改组件中的参数来设置不同的外观元素。

8 添加类常数。

9 为每个具有 getter/setter 的变量添加元数据关键字和声明。

10 定义未初始化的成员变量。

11 定义 getter 和 setter。

12 编写构造函数。它一般为空。

13 添加初始化方法。在创建类时调用此方法。

14 添加大小方法。

15 添加自定义方法或覆盖继承的方法。

导入类

外部动作脚本类文件的第一行应导入类所用的必要类文件。其中包括提供功能的类，如果类扩展了超类，还会包括超类。

在使用 `import` 语句时，导入的是完全限定的类名称，而不是类的文件名，如以下范例所示：

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

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一定会创建很大的 SWF 文件。

选择父类

大多数组件都有一些共同的行为和功能。Flash 中有两个基类专门提供这些共同的行为和功能。通过创建这些类的子类，组件一开始即可具备一组基本的方法、属性和事件。

下表简要说明这两个基类：

完整类	扩展	描述
mx.core.UIObject	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： • 编辑样式 • 事件处理 • 按缩放比例调整大小
mx.core.UIComponent	UIObject	UIComponent 是所有组件的基类。它可以参与切换，接受低级事件（如键盘和鼠标输入），还可以被禁用，以便不接收鼠标和键盘输入。 UIComponent 提供以下功能： • 创建焦点导航 • 启用和禁用组件 • 调整组件大小

了解 UIObject 类

基于 Macromedia Component Architecture 第 2 版的组件源自 UIObject 类，该类包装 MovieClip 类。MovieClip 类是 Flash 中可以在屏幕上绘制的所有类的基类。许多 MovieClip 属性和方法与时间轴相关，刚开始接触 Flash 的开发人员不太熟悉时间轴这一工具。 UIObject 类的创建目的是提取其中的许多细节。 MovieClip 的子类只记录必要的 MovieClip 属性和方法。不过，如果需要，您也可以访问这些属性和方法。

UIObject 尝试在 MovieClip 中隐藏鼠标处理和帧处理。在加载和卸载时，在它的布局更改时（`move`、`resize`），它都会在绘制之前将事件张贴到它的侦听器（等同于 `onEnterFrame`）。

UIObject 提供了替代的只读变量来确定影片剪辑的位置和大小。您可以使用 `move()` 和 `setSize()` 方法改变对象的位置和大小。

了解 UIComponent 类

UIComponent 类是 UIObject 的子类。它是具有用户交互（鼠标和键盘输入）的所有组件的基类。

扩展其他类

为了能够更方便地构造组件，您可以创建任何类的子类，这样也就不需要直接扩展 UIObject 或 UIComponent 类。如果扩展任何其他组件的类，则会默认扩展这些类。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。

Flash 包含一组可以在屏幕上绘制并且是从 UIObject 继承的类。例如， Border 类绘制其他对象周围的边框。另一个范例是 RectBorder，它是 Border 的子类，知道如何相应地调整其可视元素的大小。支持边框的所有组件应使用某个 Border 类或使用某个 Border 子类。有关这些类的详细说明，请参阅第 41 页的第 4 章 “组件字典”。

例如，如果您要创建一个组件，其行为与 Button 组件的行为几乎完全相同，您可以扩展 Button 类，而不必从基类重新创建 Button 类的所有功能。

编写构造函数

构造函数是具有唯一用途的方法：在初始化组件的新实例时设置属性和执行其他任务。您可以识别构造函数，因为它的名称与组件类自身的名称相同。例如，以下代码显示 ScrollBar 组件的构造函数：

```
function ScrollBar() {
}
```

在此情况下，初始化新的 ScrollBar 组件时，即会调用 `ScrollBar()` 构造函数。

一般情况下，组件构造函数应为空，这样才能用对象的属性接口来自定义对象。此外，根据初始化调用顺序的不同，有时在构造函数中设置属性会导致覆盖默认值。

一个类只可以包含一个构造函数；动作脚本 2.0 中不允许重载构造函数。

版本控制

在发布组件时，您应该定义版本号。这样，开发人员就能知道他们是否应该升级，并且有助于解决技术支持问题。设置组件的版本号时，请使用静态变量 version，如以下范例所示：

```
static var version:String = "1.0.0.42";
```

如果创建许多组件作为组件包的一部分，则可在外部文件中包含版本号。这样即可在一个位置更新版本号。例如，以下代码导入将版本号存储于一个位置的外部文件的内容：

```
#include "../myPackage/ComponentVersion.as"
```

ComponentVersion.as 文件的内容与上述变量声明相同，如以下范例所示：

```
static var version:String = "1.0.0.42";
```

类、元件和所有者名称

要帮助 Flash 查找适当的动作脚本类和包，并且保留组件的命名，必须在组件的动作脚本类文件中设置 symbolName、 symbolOwner 和 className 属性。

下表对这些变量进行说明：

变量	描述
symbolName	对象的元件名称。此变量为静态、 String 类型的变量。
symbolOwner	对 createClassObject() 方法的内部调用中使用的类。该值应为完全限定的类名称，它包含包的路径。此变量为静态、 Object 类型的变量。
className	组件类的名称。此变量也在计算样式值中使用。如果存在 _global.styles[className]，它将设置组件的默认值。此变量为 String 类型。

以下范例显示自定义组件的命名：

```
static var symbolName:String = "MyComponent";
static var symbolOwner:Object = custom.MyComponent;
var className:String = "MyComponent";
```

定义 getter 和 setter

getter 和 setter 向组件属性提供可见性，并控制其他对象对这些属性的访问。

定义 getter 和 setter 方法的惯例是在方法名前加 get 或 set，接着加一个空格，然后才是属性名。最好让 get 或 set 后面的每个单词首字母大写。

存储属性值的变量不能与 getter 或 setter 同名。按照惯例， getter 和 setter 变量名前面应加两个下划线。

以下范例显示 initialColor 的声明，以及获取和设置此属性值的 getter 和 setter 方法：

```
...
public var __initialColor:Color = 42;
...
public function get initialColor():Number {
  return __initialColor;
}
public function set initialColor(newColor:Number) {
  __initialColor = newColor;
}
```

Getter 和 setter 通常与元数据关键字结合使用，用于定义可见、可绑定且具有其他属性的属性。

组件元数据

Flash 识别外部动作脚本类文件中的组件元数据语句。元数据标记可以定义组件属性、数据绑定属性和事件。 Flash 解释这些语句并相应地更新开发环境。这样，您就可以一次定义这些成员，而不必在动作脚本代码和开发面板中定义。

元数据标记只可以在外部动作脚本类文件中使用。不能在 FLA 文件的动作帧中使用元数据标记。

使用元数据关键字

元数据与类声明或单个的数据字段相关。如果属性的值是 String 类型，必须在该属性两边加引号。

元数据语句绑定到动作脚本文件的下一行。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。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。

在下面的范例中，Inspectable 元数据关键字应用于 `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”面板的“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

下表说明可以在动作脚本类文件中使用的元数据标记：

标记	描述
Inspectable	定义在“组件检查器”面板中向组件用户显示的属性。请参阅第 255 页的“Inspectable”。
InspectableList	标识可检查参数的哪个子集应在属性检查器中列出。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 256 页的“InspectableList”。
事件	定义组件事件。请参阅第 257 页的“事件”。
Bindable	在“组件检查器”面板的“绑定”选项卡中显示属性。请参阅第 257 页的“Bindable”。
ChangeEvent	标识导致数据绑定发生的事件。请参阅第 258 页的“ChangeEvent”。
ComponentTask	指向与某个组件相关的一个或多个 JSFL 文件，该组件为指定的组件实例执行任务。请参阅第 259 页的“ComponentTask”。
IconFile	图标的文件名，该图标在 Flash“组件”面板中表示这一组件。请参阅第 266 页的“添加图标”。

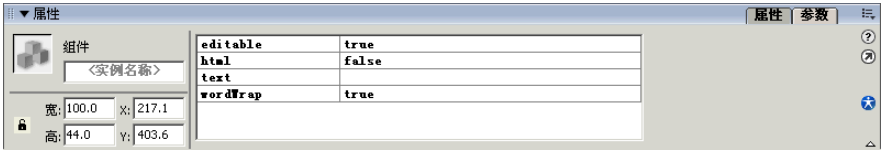
下面各节更详细地说明组件元数据标记。

Inspectable

您在组件的类定义中指定用户可编辑（或“可检查”）的组件参数，这些参数在“组件检查器”面板中显示。这样，您就可以在同一个位置维护可检查属性和基本的动作脚本代码。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”面板中选择“参数”选项卡。

下图显示“文本区域”控件的“组件检查器”面板中的“参数”选项卡：

或者，您可以在属性检查器的“参数”选项卡上查看组件属性的子集，如下图所示：



Flash 使用 Inspectable 元数据关键字确定应在创作环境中显示哪些参数。此关键字的语法如下：

```
[Inspectable(value_type=value[,attribute=value,...])]
property_declaration name:type;
```

以下范例将 `enabled` 参数定义为可检查参数：

```
[Inspectable(defaultValue=true, verbose=1, category="Other")]
var enabled:Boolean;
```

Inspectable 关键字也支持随意键入的属性，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。

下表说明 Inspectable 元数据关键字的属性：

属性	类型	描述
名称	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： • Array • Object • List • String • Number • Boolean • Font Name • Color
defaultValue	String 或 Number	（必需）可检查的属性的默认值。
enumeration	String	（可选）指定以逗号分隔的属性合法值列表。
verbose	Number	（可选）指明只有在用户指定包含详细属性时，才显示这一可检查属性。如果未指定该属性， Flash 假定应显示这一属性。
category	String	（可选）将属性划分到属性检查器中的某个特定子类别中。
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。
variable	String	（可选）其作用是向后兼容 Flash MX 组件。它用来指定此参数所绑定的变量。

InspectableList

InspectableList 元数据关键字用于确切地指定属性检查器中应显示可检查参数的哪个子集。请将 InspectableList 与 Inspectable 组合使用，这样即可隐藏子类组件的继承属性。如果不给组件的类添加 InspectableList 元数据关键字，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"[,...])]
// class definition
```

InspectableList 关键字必须紧挨着类定义且在它之前，因为它应用于整个类。

下面的范例允许 `flavorStr` 和 `colorStr` 属性在属性检查器中显示，但排除 DotParent 类的其他可检查属性：

```
[InspectableList("flavorStr","colorStr")]
class BlackDot extends DotParent {
  [Inspectable(defaultValue="strawberry")]
  public var flavorStr:String;
  [Inspectable(defaultValue="blue")]
  public var colorStr:String;
  ...
}
```

事件

Event 元数据关键字用于定义此组件发出的事件。

此关键字的语法如下：

```
[Event("event_name")]
```

例如，下面的代码定义 `click` 事件：

```
[Event("click")]
```

在动作脚本文件中将 Event 语句添加到类定义之外，以便将它们绑定到类，而不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 {
  ...
}
```

Bindable

数据绑定使各组件之间相互连接在一起。您可以通过 “组件检查器”面板的 “绑定”选项卡获得可视数据绑定。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，它们可被绑定到其他组件属性。“组件检查器”面板是 Flash MX Professional 2004 中用来进行数据绑定的主要工具。

通过使用 Bindable 元数据关键字，可以让动作脚本类中的属性和 getter/setter 函数在 “组件检查器”面板的 “绑定”选项卡中显示。

Bindable 元数据关键字的语法如下：

```
[Bindable[readonly|writeonly[,type="datatype"]]]
```

Bindable 关键字必须位于属性、getter/setter 函数或其他（位于属性或 getter/setter 函数之前的）元数据关键字之前。

下面的范例将变量 `flavorStr` 定义为公共、可检查的变量，用户也可以从 “组件检查器”面板的 “绑定”选项卡访问该变量：

```
[Bindable]
[Inspectable(defaultValue="strawberry")]
public var flavorStr:String = "strawberry";
```

Bindable 元数据关键字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	描述
`readonly`	指示 Flash 允许属性只作为绑定的源，如以下范例所示： `[Bindable("readonly")]`
`writeonly`	指示 Flash 允许属性只作为绑定的目标，如以下范例所示： `[Bindable("writeonly")]`
`type="datatype"`	指定正在被绑定的属性的数据类型。 如果不指定此选项，数据绑定将使用动作脚本代码中声明的属性数据类型。 如果 *datatype* 是注册的数据类型，则可在 “架构” 选项卡的 “数据类型” 弹出菜单中使用该功能。 以下范例将属性的数据类型设置为 String： `[Bindable(type="String")]`

您可以组合使用访问选项和数据类型选项，如以下范例所示：

```
[Bindable(param1="writeonly",type="DataProvider")]
```

在使用 ChangeEvent 元数据关键字时，必须使用 Bindable 关键字。有关更多信息，请参阅第 258 页的 “ChangeEvent”。

有关在 Flash 创作环境中创建数据绑定的信息，请参阅 “使用 Flash 帮助” 中的 “数据绑定（仅限 Flash Professional）”。

ChangeEvent

ChangeEvent 元数据关键字用于在对可绑定属性进行更改时，生成一个或多个组件事件。

此关键字的语法如下：

```
[Bindable]
[ChangeEvent("event"[,...)]
property_declaration or get/set function
```

与 Bindable 类似，此关键字只能与变量声明或 getter 及 setter 函数一起使用。

在下面的范例中，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;
```

在元数据中指定的事件发生时， Flash 将向绑定到属性的任何条目通知该属性已更改。

您还可以指示组件在调用 getter 或 setter 函数时生成事件，如以下范例所示：

```
[Bindable]
[ChangeEvent("change")]
function get selectedDate():Date
  ...
```

大多数情况下，您要在 getter 上设置 `change` 事件，在 setter 上发送该事件。

您可以在元数据中注册多个 `change` 事件，如以下范例所示：

```
[ChangeEvent("change1", "change2", "change3")]
```

其中的任一事件均表示变量有更改。不一定要发生所有事件才表示变量有更改。

ComponentTask

组件可以具有一个或多个相关的 JSFL 文件，这些文件为给定的组件实例执行有用任务。您应使用 ComponentTask 元数据关键字来为组件定义这些任务并声明这些任务。

ComponentTask 元数据关键字使用以下语法：

```
[ComponentTask(taskName,taskFile,otherFile1[,...])]
```

下表对 ComponentTask 关键字的属性进行说明：

属性	类型	描述
taskName	String	（必需）任务的名称，显示为字符串。
taskFile	String	（必需）执行任务的 JSFL 文件的名称。
otherFile1, ...	String	（必需） JSFL 文件所需的一个或多个文件的名称；例如，您的任务可能需要 XML2UI 描述文件。

在动作脚本文件中将 ComponentTask 语句添加到类定义之外，以便将它们绑定到类，而不绑定到类的特定成员。

以下范例为 MyComponent 类定义两个 ComponentTask 语句：

```
[ComponentTask("Do Some
  Setup","MyComponentSetup.jsfl","myForm.xml","myOtherForm.xml")]
[ComponentTask("Do Some More Setup","MyOtherComponentSetup.jsfl")]
class myComponent {
  ...
}
```

“任务”弹出菜单显示在 “组件检查器”面板的 “架构”选项卡上。要激活此菜单，请在舞台上选择组件实例，然后单击面板右侧的按钮。此菜单包含组件元数据中定义的所有任务的名称。

从 “任务”菜单选择任务名称时，将调用对应的 JSFL 文件。您可以从该 JSFL 文件访问当前选定的组件，如下所示：

```
var aComponent = fl.getDocumentDOM().selection[0];
```

在导出 SWC 文件时，Flash 会随组件提供相关的 JSFL 文件。在将组件导出为 SWC 文件时，JSFL 和帮助文件必须与 FLA 文件位于同一个文件夹中。有关生成 SWC 文件的更多信息，请参阅第 265 页的 “导出组件”。

定义组件参数

在构建组件时，您可以添加定义组件外观和行为的参数。最常用的属性在 “组件检查器”面板中显示为创作参数。您可以使用 Inspectable 关键字定义这些属性（请参阅第 255 页的 “Inspectable”）。您也可以使用动作脚本设置所有可检查参数。使用动作脚本设置的参数将覆盖在创作过程中设置的任何值。

下面的范例在 JellyBean 类文件中设置多个组件参数，并使用 Inspectable 元数据关键字让它们显示在 “组件检查器”面板中：

```
class JellyBean{
  // a string parameter
  [Inspectable(defaultValue="strawberry")]
  public var flavorStr:String;
  // a string list parameter
  [Inspectable(enumeration="sour,sweet,juicy,rotten",defaultValue="sweet")]
  public var flavorType:String;
```

```
  // an array parameter
  [Inspectable(name="Flavors", defaultValue="strawberry,grape,orange",
  verbose=1, category="Fruits")]
  var flavorList:Array;
  // an object parameter
  [Inspectable(defaultValue="belly:flop,jelly:drop")]
  public var jellyObject:Object;
  // a color parameter
  [Inspectable(defaultValue="#ffffff")]
  public var jellyColor:Color;
}
```

参数可以是以下任一受支持的类型：

- Array
- Object
- List
- String
- Number
- Boolean
- Font Name
- Color

实现核心方法

所有组件必须实现两个核心方法：大小和初始化方法。如果不在自定义组件中覆盖这两个方法，Flash Player 可能会产生错误。

实现初始化方法

Flash 在创建类时调用初始化方法。初始化方法至少应调用超类的初始化方法。在调用此方法之后，才能正确设置 `width`、`height` 和 `clip` 参数。

下面是 Button 类的初始化方法示例，它调用超类的初始化方法、设置缩放和其他默认属性值，并从 UIObject 对象获取颜色属性的值：

```
function init(Void):Void {
  super.init();
  labelField.selectable = false;
  labelField.styleName = this;
  useHandCursor = false;
  // mark as using color "color"
  _color = UIObject.textColorList;
}
```

实现大小方法

Flash 从 `setSize()` 方法调用组件的大小方法，以设置组件内容的版式。大小方法至少应调用超类的大小方法，如以下范例所示：

```
function size(Void):Void {
  super.size();
}
```

处理事件

组件可以通过事件了解用户何时与界面进行了交互操作，也可以了解组件的外观或生命周期何时发生了重要的更改，例如创建或破坏组件或者组件的大小发生更改。

事件模型是基于 XML Events （XML 事件）规范的发送器 / 侦听器模型。您编写的代码向目标对象注册侦听器，这样即可在目标对象发送事件时调用侦听器。

侦听器是函数或对象，但不是方法。侦听器会接收一个事件对象作为自己的参数，该参数包含事件的名称并提供有关该事件的所有相关信息。

组件生成和发送事件并使用 （侦听）其他事件。如果对象需要了解其他对象的事件，它应向该对象注册。当事件发生时，该对象通过调用一个在注册过程中请求的函数将该事件发送到所有注册的侦听器。要从同一个对象接收多个事件，必须为每个事件进行注册。

Flash MX 2004 扩展了动作脚本 `on()` 处理函数，以支持组件事件。在组件的类文件中声明事件并实现 `addEventListener()` 方法的任何组件均受支持。

公共事件

下面列出由各种不同的类广播的公共事件。如果事件适用于组件，每个组件都应尝试广播这些事件。这里没有列出所有组件的全部事件，只列出了可能会由其他组件重用的事件。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	参数	使用
`click`	无	由 Button 使用，或在鼠标单击没有其他含义时。
`scroll`	`Scrollbar.lineUp`、`lineDown`、`pageUp`、`pageDown`、`thumbTrack`、`thumbPosition`、`endScroll`、`toTop`、`toBottom`、`lineLeft`、`lineRight`、`pageLeft`、`pageRight`、`toLeft`、`toRight`	由 ScrollBar 和其他导致滚动 （在滚动弹出菜单上滚动 “缓冲器”）的控件使用。
`change`	无	由 List、ComboBox 和其他文本输入组件使用。
`maxChars`	无	当用户尝试在文本输入组件中输入过多的字符时使用。

此外，由于来自 UIComponent 的继承性，所有组件均广播下列事件 ：

UIComponent 事件	描述
`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）也可能发出焦点，但所有组件均发出 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	描述
keyDown	键已被按下。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	键已被松开。

使用事件对象

事件对象作为参数传递到侦听器。事件对象是一种动作脚本对象，这种对象具有的属性中包含有关所发生的事件的信息。您可以在侦听器回调函数内使用事件对象来访问所广播的事件的名称，或者访问广播该事件的组件的实例名称。

例如，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，并跟踪该值：

```
listener = new Object();
listener.click = function(evtObj){
  trace("The " + evtObj.target.label + " button was clicked");
}
myButton.addEventListener("click", listener);
```

下表列出了所有事件对象公用的属性：

属性	描述
type	指明事件名称的字符串。它是必需的属性。
target	对广播事件的组件实例的引用。一般情况下，不必明确描述该引用对象。

最常见的事件（例如，click 和 change）除了 type 外，没有其他必需的属性。

您可以在发送事件之前，明确地构建事件对象，如以下范例所示：

```
var eventObj = new Object();
eventObj.type = "myEvent";
eventObj.target = this;
dispatchEvent(eventObj);
```

您还可以使用快捷语法在单行上设置 type 属性的值和发送事件：

```
ancestorSlide.dispatchEvent({type:"revealChild", target:this});
```

在上面的范例中，设置 target 属性为可选，因为它是隐式的。

Flash MX 2004 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，请参阅第 41 页的第 4 章 “组件字典”中的事件说明。

发送事件

在组件动作脚本类文件的正文中，您使用 dispatchEvent() 方法来广播事件。dispatchEvent() 方法的签名如下：

```
dispatchEvent(eventObj)
```

eventObj 参数是描述事件的事件对象（请参阅第 262 页的 “使用事件对象”）。

标识事件处理函数

您在应用程序的动作脚本中定义侦听组件事件的事件处理函数对象或事件处理函数。

以下范例创建侦听器对象，处理 `click` 事件，并将它作为事件侦听器添加到 myButton：

```
listener = new Object();
listener.click = function(evtObj){
  trace("The " + evtObj.target.label + " button was clicked");
}
myButton.addEventListener("click", listener);
```

除了使用侦听器对象外，您还可以将函数用作侦听器。如果侦听器不属于对象，它就是函数。例如，以下代码创建侦听器函数 `myHandler()` 并将它注册到 myButton：

```
function myHandler(eventObj){
  if (eventObj.type == "click"){
    // your code here
  }
}
myButton.addEventListener("click", myHandler);
```

有关使用 `addEventListener()` 方法的详细信息，请参阅第 22 页的 “使用组件事件侦听器”。

如果您知道某个特定对象是某个事件唯一的侦听器，就可以利用新事件模型始终会在组件实例上调用方法这一情况。此方法是事件名称加 `Handler` 一词。例如，要处理 `click` 事件，请编写以下代码：

```
myComponentInstance.clickHandler = function(o){
  // insert your code here
}
```

在上面的代码中，如果在回调函数中使用关键字 `this`，则该关键字的作用范围是 myComponentInstance。

您也可以使用支持 `handleEvent()` 方法的侦听器对象。不论事件的名称是什么，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，因此，该语法不够灵活。例如，以下代码使用 `if...else` 语句处理 `click` 和 `enter` 事件：

```
myObj.handleEvent = function(o){
  if (o.type == "click"){
    // your code here
  } else if (o.type == "enter"){
    // your code here
  }
}
target.addEventListener("click", myObj);
target2.addEventListener("enter", myObj);
```

使用 Event 元数据

在动作脚本类文件中为每个事件侦听器添加 Event 元数据。Event 关键字的值成为 `addEventListener()` 方法调用中的第一个变量，如以下范例所示：

```
[Event("click")] // event declaration
...
class FCheckBox{
  function addEventListener(eventName:String, eventHandler:Object) {
    ... //eventName is String
  }
}
```

有关 Event 元数据关键字的详细信息，请参阅第 257 页的 “事件”。

设置外观

用户界面 (UI) 控件完全由附加的影片剪辑组成。这意味着 UI 控件的所有资源都可以是 UI 控件影片剪辑的外部资源，因此其他组件可以使用它们。例如，如果组件需要按钮功能，则可重用现有的 Button 组件资源。

Button 组件使用单独的影片剪辑来表示它的每个状态 （FalseDown、 FalseUp、 Disabled、 Selected，等等）。不过，您可以使自定义影片剪辑的 “调用外观” 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。在创作时及运行时更改外观的这一功能称为设置外观。

要在组件中使用设置外观，请为组件中使用的每个外观元素 / 链接创建变量。这样，任何用户均可通过更改组件中的参数来设置不同的外观元素，如以下范例所示 ：

```
var falseUpSkin = "mySkin";
```

“mySkin” 名称随后用作 MovieClip 元件的链接名称，该元件显示 false up （假弹起）外观。

以下范例显示 Button 组件各种状态的外观变量 ：

```
var falseUpSkin:String  = "ButtonSkin";
var falseDownSkin:String  = "ButtonSkin";
var falseOverSkin:String = "ButtonSkin"
var falseDisabledSkin:String = "ButtonSkin";
var trueUpSkin:String = "ButtonSkin";
var trueDownSkin:String = "ButtonSkin";
var trueOverSkin:String = "ButtonSkin";
var trueDisabledSkin:String = "ButtonSkin";
var falseUpIcon:String = "";
var falseDownIcon:String = "";
var falseOverIcon:String = "";
var falseDisabledIcon:String = "";
var trueUpIcon:String = "";
var trueDownIcon:String = "";
var trueOverIcon:String = "";
var trueDisabledIcon:String = "";
```

添加样式

添加样式是这样一种过程 : 它向某个类注册组件中的所有图形元素，并让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。

有关样式的详细信息，请参阅第 26 页的 “使用样式自定义组件的颜色和文本”。

使组件可访问

使各类残障人士均可访问 Web 内容的需求在不断增长。使用屏幕阅读器软件使视觉障碍者能够使用 Flash 应用程序中的可视内容，这种软件可以提供对屏幕内容的声音描述。

Flash MX 2004 包含以下辅助功能 ：

- 自定义焦点导航
- 自定义键盘快捷键
- 基于屏幕的文档和屏幕创作环境
- Accessibility 类

在创建组件时，您可以包含使组件与屏幕阅读器进行通信的动作脚本。随后，在开发人员使用这些组件在 Flash 中构建应用程序时，他们可以使用 “辅助功能” 面板配置每个组件实例。

在组件的 FLA 文件中添加下面这行，它应与您添加的其他动作脚本调用位于同一图层：

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

例如，下面一行启用 MyButton 组件的辅助功能：

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

开发人员将 MyButton 组件添加到应用程序时，可以使用 “辅助功能” 面板来使其可由屏幕阅读器访问。

有关 “辅助功能” 面板和 Flash 其他辅助功能的信息，请参阅 “使用 Flash 帮助” 中的 “创建可访问内容”。

导出组件

Flash MX 2004 将组件导出为组件包（SWC 文件）。在分发组件时，您只需向用户提供 SWC 文件。此文件包含与组件相关的所有代码、SWF 文件、图像和元数据，因此用户可以方便地将它放到自己的 Flash 环境中。

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），它是由 Flash 创作工具生成。

下表对 SWC 文件的内容进行说明。

文件	描述
catalog.xml	（必需）列出组件包及其各个组件的内容，并用作 SWC 文件内其他文件的目录。
源代码	如果使用 Flash MX 2004 创建组件，源代码就是包含组件类声明的一个或多个动作脚本文件。 源代码仅在创建组件子类时用来检查类型，它不由创作工具编译，因为编译后的字节代码已在 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。
实现 SWF 文件	（必需）实现组件的 SWF 文件。它们是可以在单个 SWF 文件中定义的一个或多个组件。如果使用 Flash MX 2004 创建组件，那么每个 SWF 文件只会导出一个组件。
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，则将实现 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，才应包含此类文件。
调试信息	（可选）与实现 SWF 文件对应的 SWD 文件。它的文件名始终与 SWF 文件的文件名相同，但扩展名为 .swd。如果在 SWC 文件中包含此类文件，则允许调试组件。
图标	（可选）包含 18 x 18、每像素 8 位图标的 PNG 文件，用来在创作工具用户界面中显示组件。如果未提供图标，则显示默认图标。（请参阅第 266 页的 “添加图标”。）
属性检查器	（可选）如果指定，此 SWF 文件将用作创作工具中的自定义属性检查器。如果省略，则向用户显示默认属性检查器。

要查看 SWC 文件的内容，您可以使用任何支持 PKZip 格式的压缩工具（包括 WinZip）打开该文件。

从 Flash 环境生成 SWC 文件后，您可以选择在 SWC 文件中包含其他文件。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。

多个 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。

使用 SWC 文件

本节说明了如何创建和导入 SWC 文件。您应向组件用户提供有关导入 SWC 文件的说明。

创建 SWC 文件

在 Flash MX 2004 和 Flash MX Professional 2004 中，您可以将影片剪辑导出为 SWC 文件，以创建 SWC 文件。在创建 SWC 文件时，Flash 报告编译时错误，就像在测试 Flash 应用程序一样。

导出 SWC 文件：

1 在 Flash 库中选择一个项目。

2 右键单击 (Windows) 或按住 Control 键单击 (Macintosh) 该项目，然后选择 “导出 SWC 文件”。

3 保存 SWC 文件。

将组件 SWC 文件导入 Flash

在向其他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。

在 Flash 创作环境中使用 SWC 文件：

1 关闭 Flash 创作环境。

2 将 SWC 文件复制到 *flash_root*/en/First Run/Components 目录中。

3 启动 Flash 创作环境或重新加载 “组件”面板。

组件的图标应显示在 “组件”面板中。

使组件更易用

您在创建完组件并准备将其打包之后，可以使组件更易于您的用户使用。本节说明了向组件添加可用性的一些技巧。

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。

该图像必须为 18 x 18 像素，并且保存为 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 在组件动作脚本类文件中的类定义之前添加以下定义：

```
[IconFile("component_name.png")]
```

3 将该图像添加到 FLA 文件所在的同一目录。在导出 SWC 文件时，Flash 将在归档的根级包含该图像。

使用 “实时预览”

“实时预览”功能在默认情况下处于启用状态，它使您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。

使用 V2 结构创建组件时，无需添加 “实时预览” 。组件 SWC 文件包含实现 SWF 文件，以及在 Flash 舞台上使用该 SWF 文件的组件。

添加工具提示

当用户将鼠标滚到组件名称上，或滚到 Flash 创作环境的 “组件” 面板中的图标上时，即会显示工具提示。

要向组件添加工具提示，请在组件动作脚本类文件中的类定义之外使用 `tiptext` 关键字。您必须使用星号 (*) 注释掉该关键字并在它前面加 @ 符号，这样编译器才能正确识别它。

以下范例显示 CheckBox 组件的工具提示 ：

```
* @tiptext 基本 CheckBox 组件。扩展按钮。
```

设计组件的最佳做法

设计组件时请采用以下做法 ：

- 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。
- 使用新事件模型，而不使用 `on(event)` 语法。
- 使用 Border 类 （而不使用图形元素）绘制对象周围的边框。
- 使用基于标记的外观设置。
- 使用 `symbolName` 属性。
- 假设初始状态。因为样式属性现在位于对象上，所以您可以为样式和属性设置初始设置，这样，在构造对象时，您的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。

索引

K

L

M

X

Y

Z