

MX



macromedia®

FLASH™MX
2004

使用组件

商标

Add Life to the Web、Afterburner、Aftershock、Andromedia、Allaire、Animation PowerPack、Aria、Attain、Authorware、Authorware Star、Backstage、Bright Tiger、Clustercats、ColdFusion、Contribute、Design In Motion、Director、Dream Templates、Dreamweaver、Drumbeat 2000、EDJE、EJIPT、Extreme 3D、Fireworks、Flash、Fontographer、FreeHand、Generator、HomeSite、JFusion、JRun、Kawa、Know Your Site、Knowledge Objects、Knowledge Stream、Knowledge Track、LikeMinds、Lingo、Live Effects、MacRecorder 徽标和图案、Macromedia、Macromedia Action!、Macromedia Flash、Macromedia M 徽标和图案、Macromedia Spectra、Macromedia xRes 徽标和图案、MacroModel、Made with Macromedia、Made with Macromedia 徽标和图案、MAGIC 徽标和图案、Mediamaker、Movie Critic、Open Sesame!、Roundtrip、Roundtrip HTML、Shockwave、Sitespring、SoundEdit、Titlemaker、UltraDev、Web Design 101、what the web can be 和 Xtra 是 Macromedia, Inc. 的注册商标或商标, 可能已经在美国或其他管辖区乃至世界范围内注册。本出版物中提到的其他产品名称、徽标、图案、标题、文字或短语可能是 Macromedia, Inc. 或其他实体的商标、服务标志或商品名称, 并且可能已经在特定的管辖区甚至世界范围内注册。

第三方信息

本指南包含指向第三方 Web 站点的链接, 这些站点不由 Macromedia 控制, Macromedia 不对所链接的任何站点的内容负责。如果要访问本指南提到的第三方 Web 站点, 您应自己承担因此而带来的风险。Macromedia 提供这些链接只是为您提供方便。包含这些链接并不意味着 Macromedia 为这些第三方站点的内容提供担保或承担责任。

语音压缩和解压缩技术得到了 Nellymoser, Inc. (www.nellymoser.com) 的授权。



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

Opera® 浏览器。版权所有 © 1995-2002 Opera Software ASA 及其供应商。保留所有权利。

Apple 公司免责声明

APPLE COMPUTER, INC. 对所附计算机软件包的适销性或用于特定目的的适用性不提供任何明示或暗示的担保。某些州不允许排除暗示的担保。上述排除可能不适用于您。此担保赋予您特定的法律权利。您可能还有其他权利, 在不同的州内, 这些权利可能不同。

版权所有 © 2003 Macromedia, Inc. 保留所有权利。未经 Macromedia, Inc. 事先书面许可, 本手册及其任何部分都不允许拷贝、影印、复制、翻译或转换成任何电子形式或机器可读的形式。部件号 **ZFL70M500X**

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

第一版 : 2003 年 10 月

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

目 录

引 言：组件入门.....	7
目标读者.....	7
系统要求.....	7
安装组件.....	8
关于本文档.....	9
印刷惯例.....	9
本手册中使用的术语.....	9
其他资源.....	10
 第 1 章：关于组件.....	11
第 2 版组件的优点.....	11
组件类别.....	12
组件结构.....	12
第 2 版组件的新功能.....	12
关于编译剪辑和 SWC 文件.....	13
辅助功能和组件.....	13
 第 2 章：使用组件.....	15
“组件”面板.....	15
“库”面板中的组件.....	16
“组件检查器”面板和属性检查器中的组件.....	16
“实时预览”中的组件.....	17
处理 SWC 文件和编译剪辑.....	17
向 Flash 文档中添加组件.....	18
设置组件参数.....	20
从 Flash 文档删除组件.....	20
使用代码提示.....	20
关于组件事件.....	21
创建自定义焦点导航.....	23
管理文档中的组件深度.....	24
关于对组件使用预加载器.....	24
将第 1 版组件升级到第 2 版结构.....	24

第 3 章：自定义组件	25
使用样式自定义组件的颜色和文本	25
关于主题	31
关于设置组件外观	33
第 4 章：组件字典	39
用户界面 (UI) 组件	39
数据组件	40
媒体组件	41
管理器	41
屏幕	41
Accordion 组件（仅限 Flash Professional）	41
Alert 组件（仅限 Flash Professional）	53
Button 组件	61
CellRenderer API	70
CheckBox 组件	76
ComboBox 组件	83
数据绑定类（仅限 Flash Professional）	108
DataGrid 组件（仅限 Flash Professional）	137
DataHolder 组件（仅限 Flash Professional）	165
DataProvider API	167
DataSet 组件（仅限 Flash Professional）	176
DateChooser 组件（仅限 Flash Professional）	218
DateField 组件（仅限 Flash Professional）	228
DepthManager 类	243
FocusManager 类	248
Form 类（仅限 Flash Professional）	255
Label 组件	260
List 组件	264
Loader 组件	289
媒体组件（仅限 Flash Professional）	299
Menu 组件（仅限 Flash Professional）	334
MenuBar 组件（仅限 Flash Professional）	359
NumericStepper 组件	368
PopUpManager 类	376
ProgressBar 组件	377
RadioButton 组件	389
RDBMSResolver 组件（仅限 Flash Professional）	398
远程过程调用 (RPC) 组件 API	407
Screen 类（仅限 Flash Professional）	411
ScrollPane 组件	422
Slide 类（仅限 Flash Professional）	435
StyleManager 类	457
TextArea 组件	459
TextInput 组件	469
TransferObject 接口	479
Tree 组件（仅限 Flash Professional）	481
TreeDataProvider 接口（仅限 Flash Professional）	497
UIComponent	502

UIEventDispatcher	508
UIObject	510
Web 服务类 (仅限 Flash Professional)	528
WebServiceConnector (仅限 Flash Professional)	549
Window 组件	557
XMLConnector 组件 (仅限 Flash Professional)	567
XUpdateResolver 组件 (仅限 Flash Professional)	574
第 5 章：创建组件	579
新增功能	579
在 Flash 环境中工作	579
创建组件	581
编写组件的动作脚本	583
导入类	585
选择父类	585
编写构造函数	586
版本控制	586
类、元件和所有者名称	586
定义 getter 和 setter	587
组件元数据	587
定义组件参数	593
实现核心方法	593
处理事件	594
设置外观	597
添加样式	598
使组件可访问	598
导出组件	598
使组件更易用	600
设计组件的最佳做法	601
索引	603

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
- 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（仅限 Flash Professional）
- Alert 组件（仅限 Flash Professional）
- 数据绑定类（仅限 Flash Professional）
- DateField 组件（仅限 Flash Professional）
- DataGrid 组件（仅限 Flash Professional）
- DataHolder 组件（仅限 Flash Professional）
- DataSet 组件（仅限 Flash Professional）
- DateChooser 组件（仅限 Flash Professional）
- Form 类（仅限 Flash Professional）
- 媒体组件（仅限 Flash Professional）
- Menu 组件（仅限 Flash Professional）
- MenuBar 组件（仅限 Flash Professional）
- RDBMSResolver 组件（仅限 Flash Professional）
- Screen 类（仅限 Flash Professional）
- Slide 类（仅限 Flash Professional）
- Tree 组件（仅限 Flash Professional）
- WebServiceConnector 类（仅限 Flash Professional）
- XMLConnector 组件（仅限 Flash Professional）
- XUpdateResolver 组件（仅限 Flash Professional）

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](#) 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：
 - \Program Files\Macromedia\Flex MX 2004\<language>\First Run\Components (Windows)
 - HD/Applications/Macromedia Flex MX 2004/First Run/Components (Macintosh)
- 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 Flash 入门（或“入门”帮助）、“使用 Flash”帮助、《动作脚本参考指南》帮助以及动作脚本字典帮助。
- 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 支持中心，网址为：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》帮助和“动作脚本字典”帮助。

有关使用组件的介绍，请参阅 Macromedia On Demand 研究会“Flash MX 2004 系列：使用 UI 组件”，网址为：www.macromedia.com/macromedia/events/online/ondemand/index.html。

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](#) 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](#) 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 版组件升级到第 2 版结构”。

组件类别

Flash MX 2004 和 Flash MX Professional 2004 中包含的组件分为五类：用户界面组件、数据组件、媒体组件、管理器和屏幕。用户界面组件使您可以与应用程序进行交互操作；例如，RadioButton、CheckBox 和 TextInput 组件都是用户界面控件。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。媒体组件使您可以播放和控制媒体流；MediaController、MediaPlayback 和 MediaDisplay 都是媒体组件。管理器是不可见的组件，此类组件使您可以在应用程序中管理诸如焦点或深度等功能；FocusManager、DepthManager、PopUpManager 和 StyleManager 都是 Flash MX 2004 和 Flash MX Professional 2004 包含的管理器组件。屏幕类别包括了动作脚本类，这些类使您可以在 Flash MX Professional 2004 中控制表单和幻灯片。有关每个类别的完整列表，请参阅第 39 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 版组件升级到第 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 版组件是用动作脚本 2.0 编写的。每个组件都是一个类，而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 39 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 25 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 31 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 502 页的“UIComponent”和第 510 页的“UIObject”。）

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。请参阅第 579 页的第 5 章“创建组件”。

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 2.0，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。请参阅第 579 页的第 5 章“创建组件”和《动作脚本参考指南》帮助。

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，请参阅第 579 页的第 5 章“创建组件”。

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轻松地感知 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，请查看第 39 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.ComponentNameAccImpl.enableAccessibility());`，并在“辅助功能”面板中设置辅助功能参数。组件辅助功能的运行方式与所有 Flash 影片剪辑辅助功能的运行方式相同。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”。

由 Macromedia 构建的大多数组件也能通过键盘来导航。第 39 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

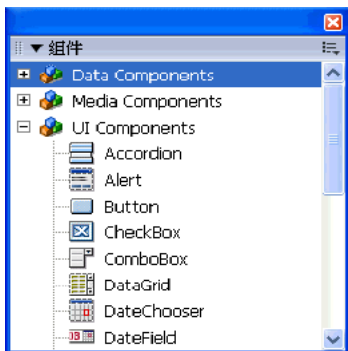
在 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中使用组件有多种方法。您可以使用“组件”面板来查看组件，并可以在创作过程中将组件添加到文档中。在将组件添加到文档中后，即可在属性检查器或“组件检查器”面板中查看组件属性。组件可以与其他组件通信，方法是侦听其他组件的事件并使用“动作脚本”处理这些事件。您还可以管理文档中的组件深度以及控制组件接收焦点的时间。

“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 (Windows) 或 Macromedia Flash 2004\zh_cn\First Run/Components (Macintosh) 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（SWC 文件）元件。



“库”面板中的 *ComboBox* 组件。

通过将组件图标从库拖到舞台上可以添加该组件的多个实例。

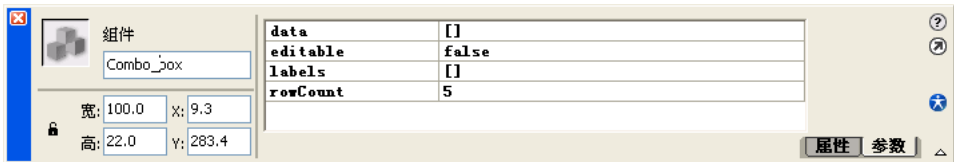
有关编译剪辑的详细信息，请参阅第 17 页的“处理 SWC 文件和编译剪辑”。

“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” > “开发面板” > “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。要了解“实时预览”中所反映的组件参数的有关信息，请参阅第 39 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，请参阅第 579 页的第 5 章“创建组件”。

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，请参阅第 579 页的第 5 章“创建组件”。

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“组件检查器”面板中的“参数”选项卡指定基本参数，然后使用 `on()` 事件处理函数来控制组件。
- 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“组件检查器”面板的“参数”选项卡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” > “开发面板” > “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 39 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 25 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 31 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 39 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 39 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中调用 `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 39 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，诸如 `UIObject.setSize()`、`UIObject.setStyle()`、`UIObject.x` 和 `UIObject.y` 等。每个组件还具有特有的属性和方法，其中的某些属性和方法可用作创作参数。例如，`ProgressBar` 组件具有 `percentComplete` 属性（`ProgressBar.percentComplete`），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：

```
import mx.controls.CheckBox;
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 `on()` 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()` 事件处理函数分配给组件实例，就像将处理函数分配给按钮或影片剪辑一样。

当您使用 `on()` 事件处理函数时，将会在触发事件并将其传递到处理函数时自动生成事件对象 `eventObj`。该事件对象的属性包含有关事件的信息。传递到 `on()` 处理函数的事件对象始终是 `eventObj`。有关详细信息，请参阅第 508 页的“[UIEventDispatcher](#)”。

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);
}
```

要使用 `on()` 事件处理函数：

- 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("The " + eventObj.type + " event was broadcast");
}
```

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”面板中选择要查看跟踪的复选框。
有关详细信息，请参阅第 39 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 508 页的“[UIEventDispatcher](#)”。

有关组件所广播事件的信息，请参阅第 39 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 `target` 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 `target` 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，才应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，请参阅第 579 页的第 5 章 “创建组件”。

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，在下面的范例如中，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;
```

您还可以使用“辅助功能”面板来分配 Tab 键索引值。

如果舞台上没有任何对象具有 Tab 键索引值，则 FocusManager 使用 Z 顺序。Z 顺序最初按组件拖到舞台上的顺序设置；不过，您也可以使用“修改”/“排列”/“移至顶层”或“移至底层”命令来确定最终的 Z 顺序。

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

要创建在用户按 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、对话框浮于内容上方，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 243 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 版组件升级到第 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 版组件和第 2 版组件可能会导致不可预料的情况。有关升级第 1 版组件以使用第 2 版的事件处理、样式和 getter/setter 来访问属性（而不是方法）的信息，请参阅[第 579 页的第 5 章“创建组件”](#)。

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《动作脚本参考指南》帮助中的“使用动作脚本 2.0 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()` 及 `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。
- 创建默认的类型样式声明。
您还可以定义默认的类型样式声明，以使类的每个实例共享一个默认外观。

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 29 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 26 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 39 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 457 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 457 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）	0xFFFFFFFF
red（红色）	0xFF0000
green（绿色）	0x00FF00
blue（蓝色）	0x0000FF
magenta（洋红色）	0xFF00FF
yellow（黄色）	0xFFFF00
cyan（青色）	0x00FFFF

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 39 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 Microsoft 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 26 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 39 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
<code>backgroundColor</code>	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。
<code>borderColor</code>	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）。
<code>borderStyle</code>	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”。
<code>buttonColor</code>	按钮的表面和三维边框的一部分。默认值为 0xEFEEEF（浅灰）。
<code>color</code>	组件标签的文本。默认值为 0x000000（黑色）。
<code>disabledColor</code>	禁用的文本颜色。默认值为 0x848384（深灰）。
<code>fontFamily</code>	文本的字体名称。默认值为 <code>_sans</code> 。
<code>fontSize</code>	字体的磅值。默认值为 10。
<code>fontStyle</code>	字体样式：可以是“normal”或“italic”。默认值为“normal”。
<code>fontWeight</code>	字体粗细：可以是“normal”或“bold”。默认值为“normal”。

样式	描述
highlightColor	三维边框的一部分。默认值为 0xFFFFFFFF（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 0xEFEDEF（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 0xFFFFFFFF（白色）。
symbolBackgroundDisabledColor	复选框和单选按钮在禁用时的背景色。默认值为 0xEFEDEF（浅灰）。
symbolBackgroundPressedColor	复选框和单选按钮在按下时的背景色。默认值为 0xFFFFFFFF（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：

- First Run\ComponentFLA (Windows)
- First Run\ComponentFLA (Macintosh)

您可以创建新主题并将它们应用到应用程序，以便更改所有组件的外观和行为。例如，您可以创建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。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 25 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，返回到文档编辑模式。
- 7 重复执行第 4 步至第 6 步，直到编辑完所有要更改的外观。
- 8 执行前一节第 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 31 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 子组件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 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：

- 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 31 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。

- 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（请参阅下一节，第 33 页的“编辑组件外观”和第 34 页的“将经过编辑的外观应用到组件”）。
- 要更改子组件中的外观（例如，List 组件中的滚动条），可将该组件分成子类（请参阅第 35 页的“将经过编辑的外观应用到子组件”）。
- 要更改无法从主组件直接访问的子组件（例如 ComboBox 组件中的 List 组件）的外观，请替换原型中的外观属性（请参阅第 37 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.flc**。
- 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，在元件编辑模式下打开该外观。
- 7 修改该影片剪辑，或删除它然后创建一个新影片剪辑。
您可能需要选择“视图” > “放大”来增加缩放比例。您在编辑外观时，必须您可以注册点，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，返回到文档编辑模式。
- 9 选择“文件” > “保存”，但不关闭 MyTheme.flc。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，请参阅下一节，第 34 页的“将经过编辑的外观应用到组件”、第 35 页的“将经过编辑的外观应用到子组件”或第 37 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.flc** 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.flc` 拖到 `DynamicSkinning.flc` 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.flc** 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.flc` 拖到 `ManualSkinning.flc` 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 39 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 33 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 33 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`，在元件编辑模式下将其打开。
- 7 在 `MyScrollDownArrowDown` 元件的第 1 帧上输入以下代码：

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

- 8 执行下列操作之一将 `List` 组件添加到文档中：

- 将 `List` 组件从“组件”面板拖到舞台。输入足够多的标签参数，以便垂直滚动条能够显示出来。
- 将 `List` 组件从“组件”面板拖到舞台上并将其删除。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 33 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 39 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 33 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 （仅限 Flash Professional）	一组垂直的互相重叠的视图，视图顶部有一些按钮，用户利用这些按钮可以在视图之间进行切换。
Alert 组件 （仅限 Flash Professional）	一个窗口，用于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选择的文本字段，以允许用户搜索此列表。
DateChooser 组件 （仅限 Flash Professional）	允许用户从日历中选择一个或多个日期。
DateField 组件 （仅限 Flash Professional）	一个不可选择的文本字段，并带有日历图标。当用户在组件边框内的任何位置单击时，就会显示一个 DateChooser 组件。
DataGrid 组件 （仅限 Flash Professional）	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。

组件	描述
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（仅限 Flash Professional）	允许用户从列表选择一个命令；它是一个标准的桌面应用程序菜单。
MenuBar 组件（仅限 Flash Professional）	水平的菜单栏。
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（仅限 Flash Professional）	允许用户处理分级信息。
Window 组件	一个可拖动的窗口，带有标题栏、题注、边框和“关闭”按钮，用于向用户显示内容。

数据组件

组件	描述
数据绑定类（仅限 Flash Professional）	这些类实现 Flash 的运行时数据绑定功能。
DataHolder 组件（仅限 Flash Professional）	保存数据，并可用作组件之间的连接器。
DataProvider API	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（仅限 Flash Professional）	一个构造块，用于创建数据驱动的应用程序。
RDBMSResolver 组件（仅限 Flash Professional）	用于将数据保存回任何支持的数据源。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。
Web 服务类（仅限 Flash Professional）	这些类使您能够访问 mx.services 包中的使用简单对象访问协议 (SOPAP) 的 Web 服务。
WebServiceConnector 类（仅限 Flash Professional）	提供对 Web 服务方法调用的无脚本访问。
XMLConnector 组件（仅限 Flash Professional）	使用 HTTP GET 和 POST 方法来读写 XML 文档。
XUpdateResolver 组件（仅限 Flash Professional）	用于将数据保存回任何支持的数据源。此解析程序组件将 DeltaPacket 翻译为 XUpdate。

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。

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

Accordion 组件（仅限 Flash Professional）

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

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

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

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

无法使用屏幕阅读器来访问 Accordion 组件。

使用 Accordion 组件（仅限 Flash Professional）

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

Accordion 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 Accordion 组件实例设置的创作参数：

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

childNames 一个数组，指定 accordion 子项的实例名称。默认值为 []（空数组）。

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

childIcons 一个数组，指定要用作 accordion 标题中的图标库元件的链接标识符。默认值为 []（空数组）。

您可以编写动作脚本，以便使用 Accordion 组件的属性、方法和事件来控制该组件的这些和其他选项。有关详细信息，请参阅第 46 页的“[Accordion 类（仅限 Flash Professional）](#)”。

创建具有 Accordion 组件的应用程序

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

要使用屏幕将 Accordion 组件添加到应用程序：

- 1 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 表单应用程序”。
- 2 双击文本“Form1”并输入名称 **addressForm**。
尽管 addressForm 屏幕未显示在库中，但它是 Screen 类（View 类的子类）的一个元件，accordion 可使用该元件作为子项。
- 3 在表单处于选定状态的情况下，在属性检查器中将其 visible 属性设置为 false。
这将在应用程序中隐藏表单的内容；表单只会出现在 Accordion 中。
- 4 将诸如 Label 和 TextInput 等组件从“组件”面板拖到表单上，以便创建一个模拟地址表单；对这些组件进行排列，然后在“组件检查器”面板的“参数”窗格中设置它们的属性。
将表单元素定位在表单的左上角。表单的左上角位于 Accordion 的左上角中。
- 5 重复步骤 2-4，以便创建一个名为 **checkoutForm** 的屏幕。
- 6 创建一个名为 **accordionForm** 的新表单。
- 7 将 Accordion 组件从“组件”面板拖到 accordionForm 表单中，并将其命名为 **myAccordion**。
- 8 在 myAccordion 处于选定状态的情况下，在属性检查器中执行以下操作：
 - 对于 childSymbols 属性，输入 **addressForm**、**addressForm** 和 **checkoutForm**。
这些字符串指定用于创建 accordion 子项的屏幕的名称。
注意：前两个子项是同一屏幕的实例，因为发运地址表单和开单地址表单具有相同的组件。
 - 对于 childNames 属性，输入 **shippingAddress**、**billingAddress** 和 **checkout**。
这些字符串是该 accordion 的子项的动作脚本名称。
 - 对于 childLabels 属性，输入发运地址、开单地址和结帐。
这些字符串是 accordion 标题上的文本标签。
- 9 选择“控制”>“测试影片”。

要将 Accordion 组件添加到应用程序，请执行以下操作：

- 1 选择“文件”>“新建”，然后创建新的 Flash 文档。
- 2 选择“插入”>“新元件”，并将其命名为 **AddressForm**。
- 3 在“创建新元件”对话框中，单击“高级”按钮并选择“为动作脚本导出”。在“AS 2.0 类”字段中，输入 **mx.core.View**。
要保持某个 accordion 子项中的 Tab 键顺序，该子项必须同时是 View 类的实例。
- 4 将诸如 Label 和 TextInput 等组件从“组件”面板拖到舞台上，以便创建一个模拟地址表单；对这些组件进行排列，然后在“组件检查器”面板的“参数”窗格中设置它们的属性。
将表单元素定位在舞台上相对于 0, 0（中心）的位置。影片剪辑的 0, 0 坐标位于 Accordion 的左上角。
- 5 选择“编辑”>“编辑文档”返回到主时间轴。
- 6 重复步骤 2-5，以便创建一个名为 **CheckoutForm** 的影片剪辑。

- 7 将 Accordion 组件从“组件”面板拖到主时间轴上的舞台上。
- 8 在属性检查器中，执行以下操作：
 - 输入实例名称 **myAccordion**。
 - 对于 childSymbols 属性，输入 **AddressForm**、**AddressForm** 和 **CheckoutForm**。
这些字符串指定用于创建 accordion 子项的影片剪辑的名称。
注意：前两个子项是同一影片剪辑的实例，因为发运地址表单和开单地址表单相同。
 - 对于 childNames 属性，输入 **shippingAddress**、**billingAddress** 和 **checkout**。
这些字符串是该 accordion 的子项的动作脚本名称。
 - 对于 childLabels 属性，输入发运地址、开单地址和结帐。
这些字符串是 accordion 标题上的文本标签。
 - 对于 childIcons 属性，输入 **AddressIcon**、**AddressIcon** 和 **CheckoutIcon**。
这些字符串指定用作 accordion 标题上的图标影片剪辑元件的链接标识符。如果希望图标出现在标题中，您必须创建这些影片剪辑元件。
- 9 选择“控制”>“测试影片”。

要使用动作脚本将子项添加到 Accordion 组件，请执行以下操作：

- 1 选择“文件”>“新建”，然后创建 Flash 文档。
- 2 将 Accordion 组件从“组件”面板拖到舞台上。
- 3 在属性检查器中，输入实例名称 **myAccordion**。
- 4 将 TextInput 组件拖到舞台上，然后将其删除。
此操作会将该组件添加到库中，以便能够在步骤 6 中将其动态地实例化。
- 5 在时间轴第 1 帧的“动作”面板中，输入下列代码：

```
myAccordion.createChild("View", "shippingAddress", { label:"Shipping Address"
});
myAccordion.createChild("View", "billingAddress", { label:"Billing Address"
});
myAccordion.createChild("View", "payment", { label:"Payment" });
```

此代码调用 createChild() 方法来创建其子项视图。

- 6 在第 1 帧的“动作”面板中您在步骤 4 中输入的代码的下方，输入以下代码：

```
var o = myAccordion.shippingAddress.createChild("TextInput", "firstName");
o.move(20, 38);
o.setSize(116, 20);
o = myAccordion.shippingAddress.createChild("TextInput", "lastName");
o.move(175, 38);
o.setSize(145, 20);
```

此代码将组件实例（两个 TextInput 组件）添加到 accordion 的子项。

自定义 Accordion 组件（仅限 Flash Professional）

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

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

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

对 Accordion 组件使用样式

您可以设置样式属性，以便更改 Accordion 组件的边框和背景的外观。

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 25 页的“使用样式自定义组件的颜色和文本”。

Accordion 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”。
backgroundColor	背景色。
borderColor	边框颜色。
borderStyle	边框的样式；可能的值包括“none”、“solid”、“inset”、“outset”、“default”、“alert”。“default”值是 Window 组件边框的外观，“alert”值是 Alert 组件边框的外观。
headerHeight	标题按钮的高度（以像素为单位）。
color	标题的文本颜色。
disabledColor	禁用 accordion 的颜色。
fontFamily	标题标签的字体名称。
fontSize	标题标签字体的磅值。
fontStyle	标题标签的字体样式；“normal”或“italic”。
fontWeight	标题标签的字体粗细；“normal”或“bold”。
textDecoration	文本修饰；“无”或“下划线”。
openDuration	过渡动画的持续时间（以毫秒为单位）。
openEasing	动画使用的补间函数。

对 Accordion 组件使用外观

Accordion 组件使用外观来表示其标题按钮的可视状态。要在创作时为按钮和标题栏设置外观，在其中一个主题 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/Accordion Assets skins states 文件夹中的外观元件。有关详细信息，请参阅第 33 页的“关于设置组件外观”。

Accordion 组件由边框和背景、标题按钮和子项组成。边框和背景可设置样式，但不可设置外观。使用从下列按钮继承的外观的子集，标题可设置外观，但不可设置样式。Accordion 组件使用以下外观属性来动态设置标题按钮的外观：

属性	描述	默认值
falseUpSkin	弹起状态。	accordionHeaderSkin
falseDownSkin	按下状态。	accordionHeaderSkin
falseOverSkin	滑过状态。	accordionHeaderSkin
trueUpSkin	切换状态。	accordionHeaderSkin

Accordion 类（仅限 Flash Professional）

继承 UIObject > UIComponent > View > Accordion

动作脚本类名称 mx.containers.Accordion

Accordion 是一种组件，它包含一次显示一个的子项。每个子项都有在创建子项时创建的对应该标题按钮。子项必须是 UIObject 的实例。

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

如果使用动作脚本设置 Accordion 类的属性，则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.Accordion.version);
```

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

Accordion 类的方法摘要

方法	描述
Accordion.createChild()	创建 accordion 实例的子项。
Accordion.createSegment()	创建 accordion 实例的子项。此方法的参数与 createChild() 方法的参数不同。
Accordion.destroyChildAt()	破坏位于指定索引位置的子项。
Accordion.getChildAt()	获取对位于指定索引位置的子项的引用。

继承 UIObject、UIComponent 和 mx.core.View 的所有方法。

Accordion 类的属性摘要

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

继承 [UIObject](#)、[UIComponent](#) 和 `mx.core.View` 的所有属性。

Accordion 类的事件摘要

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

继承 [UIObject](#)、[UIComponent](#) 和 `mx.core.View` 的所有事件。

Accordion.change

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

描述

事件；在 accordion 的 `selectedIndex` 和 `selectedChild` 属性更改时广播到所有注册的侦听器。只有在用户的鼠标单击或按键动作更改了 `selectedChild` 或 `selectedIndex` 值时才会广播该事件，而不是在使用动作脚本更改值时。在过渡动画出现之前广播该事件。

V2 组件使用调度程序 / 侦听器事件模型。Accordion 组件会在它的其中一个按钮被按下时发送 `change` 事件，该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称为处理函数）进行处理。您调用 `addEventListener()` 方法并将对处理函数的引用作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myAccordionListener` 的处理函数，并且将该处理函数传递到 `myAccordion.addEventListener()` 方法，作为第二个参数。`change` 处理函数在 `evtObject` 参数中捕获事件对象。在广播 `change` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
myAccordionListener = new Object();
myAccordionListener.change = function(){
    trace("Changed to different view");
}
myAccordion.addEventListener("change", myAccordionListener);
```

Accordion.createChild()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myAccordion.createChild(classOrSymbolName, instanceName[, initialProperties])
```

参数

classOrSymbolName 此参数可以是待实例化的 `UIObject` 的类的构造函数，也可以是链接名称（即指向待实例化的元件的引用）。类必须是 `UIObject` 或 `UIObject` 的子类，但大多数情况下是 `View` 或 `View` 的子类。

instanceName 新实例的实例名称。

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

- `label` 此字符串指定新的子实例在其标题上使用的文本标签。
- `icon` 此字符串指定子项用作其标题上的图标的库元件的链接标识符。

返回

指向 `UIObject`（新创建子项）的实例的引用。

说明

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

在创建每一子项时，按创建顺序给各子项分配一个索引编号，并且 `numChildren` 属性加 1。

示例

以下代码将创建名为 `payment` 的影片剪辑元件 `PaymentForm` 的一个实例，作为 `myAccordion` 的最后一个子项：

```
var child = myAccordion.createChild("PaymentForm", "payment", {
    label:"Payment", icon:"payIcon" });
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```


以下代码创建作为 **View** 类的实例的子项：

```
var child = myAccordion.createChild(mx.core.View, "payment", { label:"Payment",  
    Icon:"payIcon" });  
child.cardType.text = "Visa";  
child.cardNumber.text = "1234567887654321";
```

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

```
import mx.core.View  
var child = myAccordion.createChild(View, "payment", { label:"Payment",  
    Icon:"payIcon" });  
child.cardType.text = "Visa";  
child.cardNumber.text = "1234567887654321";
```

Accordion.createSegment()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

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

参数

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

instanceName 新实例的实例名称。

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

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

返回

指向新创建的 **UIObject** 实例的引用。

说明

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

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

在创建每一子项时，按创建顺序给各子项分配一个索引编号，并且 **numChildren** 属性加 1。

示例

以下范例将创建名为 **payment** 的 **PaymentForm** 影片剪辑元件的实例，作为 **myAccordion** 的最后一个子项：

```
var child = myAccordion.createSegment("PaymentForm", "payment", "Payment",  
    "payIcon");
```

```
child.cardType.text = "Visa";  
child.cardNumber.text = "1234567887654321";
```

以下代码创建作为 **View** 类的实例的子项：

```
var child = myAccordion.createSegment(mx.core.View, "payment", {  
    label:"Payment", Icon:"payIcon" });  
child.cardType.text = "Visa";  
child.cardNumber.text = "1234567887654321";
```

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

```
import mx.core.View  
var child = myAccordion.createSegment(View, "payment", { label:"Payment",  
    Icon:"payIcon" });  
child.cardType.text = "Visa";  
child.cardNumber.text = "1234567887654321";
```

Accordion.destroyChildAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myAccordion.destroyChildAt(index)
```

参数

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

返回

无。

说明

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

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

注意：调用 **destroyChildAt()** 方法会将 **numChildren** 属性减 1。

示例

以下代码将破坏 **myAccordion** 的最后一个子项：

```
myAccordion.destroyChildAt(myAccordion.numChildren - 1);
```

另请参见

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

Accordion.getChildAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myAccordion.getChildAt(index)
```

参数

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

返回

指向位于指定索引位置的 UIObject 实例的引用。

说明

方法；返回指向位于指定索引位置的子项的引用。为每一 accordion 子项都提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个子项为 0，第二个子项为 1，依此类推。

示例

以下代码将获取指向 myAccordion 最后一个子项的引用：

```
var lastChild:UIObject = myAccordion.getChildAt(myAccordion.numChildren - 1);
```

Accordion.numChildren

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myAccordion.numChildren
```

描述

属性（从 View 继承）；指明 accordion 实例中的子项数（UIObjects 子项）。标题不会被计作子项。

为每一 accordion 子项都提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个子项为 0，第二个子项为 1，依此类推。代码 myAccordion.numChild - 1 始终引用添加到 accordion 的最后一个子项。例如，如果某个 accordion 中有 7 个子项，则最后一个子项的索引为 6。numChildren 属性不是从零开始的，因此 myAccordion.numChildren 的值将为 7。7 减 1 的结果是 6，即最后一个子项的索引号。

范例

以下范例选择最后一个子项：

```
myAccordion.selectedIndex = myAccordion.numChildren - 1;
```

Accordion.selectedChild

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myAccordion.selectedChild
```

描述

属性；如果存在一个或多个子项，则为选定子项；如果不存在子项，则未定义。此属性的类型为 `UIObject` 或未定义。

如果 `accordion` 有子项，则代码 `myAccordion.selectedChild` 与代码 `myAccordion.getChildAt(myAccordion.selectedIndex)` 等效。

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

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

如果 `accordion` 有子项，则默认值为 `myAccordion.getChildAt(0)`。如果 `accordion` 没有子项，则默认值未定义。

范例

以下范例将获取选定子项视图的标签：

```
var selectedLabel = myAccordion.selectedChild.label;
```

以下范例将设置要作为选定子项视图的付款表单：

```
myAccordion.selectedChild = myAccordion.payment;
```

另请参见

[Accordion.selectedIndex](#)

Accordion.selectedIndex

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myAccordion.selectedIndex
```

描述

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

为每一 `accordion` 子项都提供一个索引编号以指示其位置。该索引编号从零开始，因此第一个子项为 0，第二个子项为 1，依此类推。`selectedIndex` 的有效值为 0、1、2...、 $n - 1$ ，其中 n 是子项的数量。

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

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

范例

以下范例将记录选定子项的索引：

```
var oldSelectedIndex = myAccordion.selectedIndex;
```

以下范例选择最后一个子项：

```
myAccordion.selectedIndex = myAccordion.numChildren - 1;
```

另请参见

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

Alert 组件（仅限 Flash Professional）

`Alert` 组件使您能够弹出一个窗口，该窗口向用户呈现一条消息和响应按钮。`Alert` 窗口包含一个可填充文本的标题栏、一个可自定义的消息和若干可更改标签的按钮。`Alert` 窗口只能包含以下按钮的任意组合：“是”、“否”、“确定”和“取消”。可以通过使用以下属性更改按钮上的文本标签：[Alert.yesLabel](#)、[Alert.noLabel](#)、[Alert.okLabel](#) 和 [Alert.cancelLabel](#)。您无法更改 `Alert` 窗口中按钮的顺序；按钮顺序始终为“确定”、“是”、“否”、“取消”。

要弹出 `Alert` 窗口，您必须调用 [Alert.show\(\)](#) 方法。为了成功调用该方法，`Alert` 组件必须位于库中。您必须将 `Alert` 组件从“组件”面板拖到舞台上，然后再将 `Alert` 组件从舞台中删除。此操作将组件添加到库中，但不会在文档中显示它。

`Alert` 组件的实时预览是一个空窗口。

可以使用屏幕阅读器来访问 `Alert` 窗口的文本和按钮。将 `Alert` 组件添加到应用程序时，可以使用“辅助功能”面板使其可由屏幕阅读器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.AlertAccImpl.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”。

使用 Alert 组件（仅限 Flash Professional）

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

Alert 参数

`Alert` 组件没有创作参数。必须调用动作脚本的 [Alert.show\(\)](#) 方法来弹出 `Alert` 窗口。可以使用其他动作脚本属性来修改应用程序中的 `Alert` 窗口。有关详细信息，请参阅第 55 页的“[Alert 类（仅限 Flash Professional）](#)”。

创建具有 Alert 组件的应用程序

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

要创建具有 Alert 组件的应用程序，请执行以下操作：

- 1 在“组件”面板中双击 Alert 组件，将其添加到舞台。
- 2 按 Backspace 键 (Windows) 或 Delete 键 (Macintosh) 将组件从舞台中删除，此操作将组件添加到库中，但不会在应用程序中显示它。
- 3 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以便为 click 事件定义事件处理函数：

```
import mx.controls.Alert
myClickHandler = function (evt){
    if (evt.detail == Alert.OK){
        trace("start stock app");
        // startStockApplication();
    }
}
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |
    Alert.CANCEL, this, myClickHandler, "stockIcon", Alert.OK);
```

此代码将创建带有“确定”和“取消”按钮的 Alert 窗口。在按下任一按钮时，将调用 myClickHandler 函数。但在按下“确定”按钮时，调用的是 startStockApplication() 方法。

- 4 “控制” > “测试影片”。

自定义 Alert 组件（仅限 Flash Professional）

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

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

Alert 不会响应 setSize() 方法。

对 Alert 组件使用样式

您可以设置样式属性以更改 Alert 组件的外观。如果样式属性的名称以“Color”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 25 页的“使用样式自定义组件的颜色和文本”。

Alert 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。

样式	描述
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”。
buttonStyleDeclaration	按钮文本样式的类（静态） CSSStyleDeclaration。
messageStyleDeclaration	消息文本、边框和背景样式的类（静态） CSSStyleDeclaration。
titleStyleDeclaration	标题文本样式的类（静态） CSSStyleDeclaration。

对 Alert 组件使用外观

Alert 组件使用窗口外观来表示其按钮和标题栏的可视状态。要在创作时为按钮和标题栏设置外观，请在其中一个主题 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/Window Assets skins states 文件夹中的外观元件。有关详细信息，请参阅第 33 页的“关于设置组件外观”。

Alert 组件使用 RectBorder.as 类中的动作脚本代码绘制其边框。可以使用 RectBorder 样式来修改 Alert 组件，如下所示：

```
var myAlert = Alert.show("This is a test of errors", "Error", Alert.OK |  
    Alert.CANCEL, this);  
myAlert.setStyle("borderStyle", "inset");
```

有关 RectBorder 样式的详细信息，请参阅第 267 页的“在 List 组件中使用外观”。

Alert 组件使用以下外观属性动态地设置按钮和标题栏的外观：

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

Alert 类（仅限 Flash Professional）

继承 UIObject > UIComponent > View > ScrollView > Window > Alert

动作脚本类名称 mx.controls.Alert

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

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

注意：在模块化的操作环境（例如，Microsoft Windows）中，在用户进行操作（如按下按钮）之前，对 Alert.show() 的调用不会返回。

Alert 类的方法摘要

事件	描述
<code>Alert.show()</code>	使用可选参数创建 Alert 窗口。

继承 `UIObject` 和 `UIComponent` 中的所有方法。

Alert 类的属性摘要

属性	描述
<code>Alert.buttonHeight</code>	每个按钮的高度（以像素为单位）。默认值为 22。
<code>Alert.buttonWidth</code>	每个按钮的宽度（以像素为单位）。默认值为 100。
<code>Alert.cancelLabel</code>	“取消”按钮的标签文本。
<code>Alert.noLabel</code>	“否”按钮的标签文本。
<code>Alert.okLabel</code>	“确定”按钮的标签文本。
<code>Alert.yesLabel</code>	“是”按钮的标签文本。

继承 `UIObject` 和 `UIComponent` 的所有属性。

Alert 类的事件摘要

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

继承 `UIObject` 和 `UIComponent` 的所有事件。

Alert.buttonHeight

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`Alert.buttonHeight`

描述

属性（类）；更改按钮高度的类属性（静态）。

另请参见

`Alert.buttonWidth`

Alert.buttonWidth

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

Alert.buttonWidth

描述

属性（类）；更改按钮宽度的类属性（静态）。

另请参见

[Alert.buttonHeight](#)

Alert.click

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

描述

事件；在单击“确定”、“是”、“否”或“取消”按钮时广播至注册的侦听器。

V2 组件使用调度程序/侦听器事件模型。Alert 组件会在它的其中一个按钮被单击时发送 click 事件，该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称为处理函数）进行处理。您调用 [Alert.show\(\)](#) 方法并将处理函数的名称作为参数传递给它。在单击 Alert 窗口中的某个按钮时，即会调用侦听器。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Alert.click 事件的事件对象具有一个附加的 detail 属性，取决于单击了哪个按钮，该属性的值可以是以下其中一个值：Alert.OK、Alert.CANCEL、Alert.YES、Alert.NO。有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 myClickHandler 的处理函数并将其作为第五个参数传递给 Alert.show() 方法。myClickHandler 在 evt 参数中捕获事件对象。然后，事件对象的 detail 属性将用在 trace 语句内，以便将所单击按钮的名称（Alert.OK 或 Alert.CANCEL）发送到“输出”面板，如下所示：

```
myClickHandler = function (evt){
    if (evt.detail == Alert.OK){
        trace(Alert.okLabel);
    }else if (evt.detail == Alert.CANCEL){
        trace(Alert.cancelLabel);
    }
}
Alert.show("This is a test of errors", "Error", Alert.OK | Alert.CANCEL, this,
myClickHandler);
```

Alert.cancelLabel

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
Alert.cancelLabel
```

描述

属性（类）；指明“取消”按钮上的标签文本的类属性（静态）。

范例

以下范例将“取消”按钮的标签设置为“cancellation”：

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

Alert.noLabel

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
Alert.noLabel
```

描述

属性（类）；指明“否”按钮上的标签文本的类属性（静态）。

范例

以下范例将“否”按钮的标签设置为“nyet”：

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

Alert.okLabel

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
Alert.okLabel
```

描述

属性（类）；指明“确定”按钮上的标签文本的类属性（静态）。

范例

以下范例将“确定”按钮的标签设置为“okay”：

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

Alert.show()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

参数

message 要显示的消息。

title Alert 标题栏中的文本。此参数是可选的。如果未指定 *title* 参数，则标题栏为空。

flags 可选参数，指明要显示在 Alert 窗口中的一个或多个按钮。默认值为 `Alert.OK`，它将显示“确定”按钮。在使用多个值时，请利用 `|` 字符分隔各个值。该值可以是以下一个或多个值：

- `Alert.OK`
- `Alert.CANCEL`
- `Alert.YES`
- `Alert.NO`

您还可以使用 `Alert.NONMODAL` 指明 Alert 窗口是非模式的。非模式窗口允许用户与应用程序中的其他窗口交互。

parent Alert 组件的父窗口。Alert 窗口会将自己置于父窗口的中心。使用值 `null` 或 `undefined` 来指定 `_root` Timeline。该父窗口必须继承自 `UIComponent` 类。可以使用 `mx.core.View` 注册父窗口，以使其继承自 `UIComponent`。此参数是可选的。

clickHandler 单击按钮时广播的 `click` 事件的处理函数。除了标准的 `click` 事件对象属性外，还有一个附加的 `detail` 属性，该属性包含所单击按钮标志的值（`Alert.OK`、`Alert.CANCEL`、`Alert.YES`、`Alert.NO`）。此处理函数可能是函数或对象。有关详细信息，请参阅第 21 页的第 2 章“使用组件事件侦听器”。

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

defaultButton 指明在用户按下 Enter 键 (Windows) 或 Return 键 (Macintosh) 时被单击的按钮。此参数可以是下列值之一：

- Alert.OK
- Alert.CANCEL
- Alert.YES
- Alert.NO

返回

所创建 Alert 类的实例。

说明

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

示例

以下代码显示一个带有“确定”按钮的 Alert 模式窗口的简单范例。

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

以下代码定义一个将有关所单击按钮的消息发送到“输出”面板的 click 处理函数。

```
myClickHandler = function (evt){  
    trace (evt.detail + "was clicked");  
}  
Alert.show("This is a test of errors", "Error", Alert.OK | Alert.CANCEL, this,  
    myClickHandler);
```

注意：事件对象的 detail 属性将返回一个数字，用于代表每个按钮。“确定”按钮为 4，“取消”按钮为 8，“是”按钮为 1，“否”按钮为 2。

Alert.yesLabel

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
Alert.yesLabel
```

描述

属性（类）；指明“是”按钮上的标签文本的类属性（静态）。

范例

以下范例将“确定”按钮的标签设置为“da”：

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

Button 组件

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

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

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 248 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”。

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，然后输入一个链接标识符。该值是为属性检查器或“组件检查器”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 为按钮添加自定义图标。该值是库中影片剪辑或图形元件的链接标识符；没有默认值。

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

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 toggle 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 25 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，就可以动态设置其外观。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何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	图标按下切换状态。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

动作脚本类名称 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 和 UIComponent 中的所有方法。

Button 类的属性摘要

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

继承 `UIObject` 和 `UIComponent` 的所有属性。

Button 类的事件摘要

方法	描述
<code>Button.click</code>	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` 和 `UIComponent` 的所有事件。

Button.click

可用性

Flash Player 6 版本 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`) 发送一个事件（在本例中为 `click`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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和普通按钮外观。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 版本 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

[SimpleButton.emphasized](#)

Button.icon

可用性

Flash Player 6 版本 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。在元件编辑模式下，选择舞台上的该元件，并在属性检查器中的“X”和“Y”框内都输入 0。在“库”面板中，选择影片剪辑并从“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 版本 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 版本 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 版本 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 版本 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 API

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

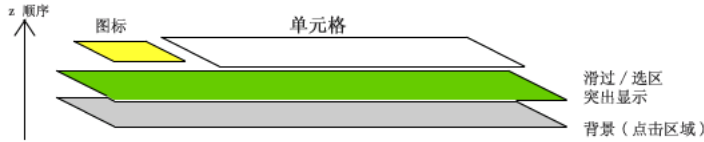
了解 List 类

要使用 CellRenderer API，深刻了解 List 类至关重要。DataGrid、Tree 和 Menu 组件是 List 类的扩展，因此，了解 List 类使您能够同时了解这些组件。

注意：组件是一种类，但类不一定是组件。

关于 List 类的构成

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



List 或 DataGrid 组件中行的堆叠顺序。

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

关于 List 类的滚动行为

List 类使用一种非常复杂的算法进行滚动。一个列表只会列出它一次能显示的最多行数，超出 rowCount 属性的值的项目根本不会获得行。在列表滚动时，它会将所有行上下移动（取决于滚动方向）。然后，列表将重复使用滚出视图的行；列表会重新初始化这些行，并使用它们作为正在滚入视图的新行，方法是将旧行的值设置为视图中的新行，然后将旧行移到新项目滚入视图的位置。

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

使用 CellRenderer API

您必须编写包含四个方法（`CellRenderer.getPreferredHeight()`、`CellRenderer.getPreferredWidth()`、`CellRenderer.setSize()`、`CellRenderer.setValue()`）的类，基于列表的组件将使用该类与单元格通信。

系统将为单元格自动指定两个方法和一个属性（`CellRenderer.getCellIndex()`、`CellRenderer.getDataLabel()` 和 `CellRenderer.listOwner`），以便允许它与基于列表的组件通信。例如，假设单元格内包含一个复选框，该复选框导致行在单击时被选中。单元格渲染器需要引用包含它的基于列表的组件，以便调用基于列表的组件的 `selectedIndex` 属性。同时，单元格需要知道它当前正在渲染的项目索引，以便能够将 `selectedIndex` 设置为正确的编号；单元格可以使用 `CellRenderer.listOwner` 和 `CellRenderer.getCellIndex()` 达到此目的。您不需要实现这些 API；在将单元格放到基于列表的组件内时，单元格将自动接收这些 API。

要为 CellRenderer API 实现的方法

您必须编写包含以下方法的类，以便 List、DataGrid、Tree 或 Menu 能够与单元格通信：

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

CellRenderer API 提供的方法

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

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

CellRenderer API 提供的属性

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

名称	描述
<code>CellRenderer.listOwner</code>	指向包含单元格的列表的引用。

CellRenderer.getDataLabel()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

`componentInstance.getDataLabel()`

参数

无。

返回

一个字符串。

描述

方法；返回包含单元格渲染器数据字段的名称的字符串。

范例

以下代码有助于使单元格知道它正在渲染数据字段 “Price”。变量 p 现在等同于 “Price”：

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

CellRenderer.getCellIndex()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

```
componentInstance.getCellIndex()
```

参数

无。

返回

包含以下两个字段的对象：columnIndex 和 itemIndex。

描述

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

范例

此范例从单元格内编辑 DataGrid 的 dataProvider：

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

CellRenderer.getPreferredHeight()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

```
componentInstance.getPreferredHeight()
```

参数

无。

返回

单元格的正确高度。

描述

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

范例

此范例返回值 20，指明单元格的高度需要为 20 个像素：

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

CellRenderer.getPreferredWidth()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

componentInstance.getPreferredWidth()

参数

无。

返回

无。

描述

方法；单元格的首选宽度。如果指定的宽度超出组件的宽度，单元格将被截断。

范例

此范例返回值 3，指明单元格需要比字符串所渲染的长度大三倍。

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

CellRenderer.listOwner

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

componentInstance.listOwner

描述

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

范例

此范例查找列表在某个单元格中的选定项目：

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

CellRenderer.setSize()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

```
componentInstance.setSize(width, height)
```

参数

width 指明布置组件所依据宽度的数字。

height 指明布置组件所依据高度的数字。

返回

无。

描述

方法；允许列表告知其单元格应按什么大小安排自己的位置。CellRenderer 应进行规划，以便它适合所述区域的边界，否则单元格的可视显示内容可能会溢出到列表的其他部分中，因此看起来不完整。

范例

此范例将调整单元格内图像的大小，以适合列表所指定的边框：

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

CellRenderer.setValue()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

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

参数

suggested 用于代表单元格渲染器的文本的值（如果需要）。

item 代表要渲染的整个项目的对象。单元格渲染器可以使用此对象的任何所需属性进行渲染。

selected 指明是否选定单元格所在行的布尔值 (true) 或 (false)。

返回

无。

描述

方法；采用指定的值，并在单元格内显示它们。这将使单元格中已显示内容以及要为新项目在单元格中所显示内容不会有任何不同。请务必记住，任何单元格处于列表中时都可能显示多个值。这是任何单元格渲染器中的最重要的方法。

范例

取决于传递的值，此范例会将图像加载到单元格内的 Loader 组件中：

```
function setValue(suggested, item, selected) :Void
{
    // 清除加载器
    loader.contentPath = undefined;
    // 列表数据提供程序中的不同图像有不同的 URL
    if (suggested!=undefined)
        loader.contentPath = suggested;
}
```

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 248 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”。

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；默认值为 defaultLabel。

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 469 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 25 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
<code>themeColor</code>	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“ <code>haloGreen</code> ”、“ <code>haloBlue</code> ”和“ <code>haloOrange</code> ”。
<code>color</code>	组件标签的文本。
<code>disabledColor</code>	禁用的文本颜色。
<code>fontFamily</code>	文本的字体名称。
<code>fontSize</code>	字体的磅值。
<code>fontStyle</code>	字体样式：“常规”或“斜体”。
<code>fontWeight</code>	字体粗细：“常规”或“粗体”。
<code>textDecoration</code>	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
<code>falseUpSkin</code>	弹起状态。默认值为 <code>RectBorder</code> 。
<code>falseDownSkin</code>	按下状态。默认值为 <code>RectBorder</code> 。
<code>falseOverSkin</code>	悬停状态。默认值为 <code>RectBorder</code> 。
<code>falseDisabledSkin</code>	禁用状态。默认值为 <code>RectBorder</code> 。

属性	描述
<code>trueUpSkin</code>	切换状态。默认值为 <code>RectBorder</code> 。
<code>trueDownSkin</code>	按下切换状态。默认值为 <code>RectBorder</code> 。
<code>trueOverSkin</code>	悬停切换状态。默认值为 <code>RectBorder</code> 。
<code>trueDisabledSkin</code>	禁用切换状态。默认值为 <code>RectBorder</code> 。

CheckBox 类

继承 `UIObject` > `UIComponent` > `SimpleButton` > `Button` > `CheckBox`

动作脚本类名称 `mx.controls.CheckBox`

`CheckBox` 类的属性允许您在运行时创建一个文本标签，并将其放在复选框的左、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 类的方法摘要

继承 `UIObject` 和 `UIComponent` 中的所有方法。

CheckBox 类的属性摘要

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

继承 `UIObject` 和 `UIComponent` 中的所有属性。

CheckBox 类的事件摘要

事件	描述
<code>CheckBox.click</code>	在按钮实例上按下鼠标时触发。

继承 `UIObject` 和 `UIComponent` 中的所有事件。

CheckBox.click

可用性

Flash Player 6 版本 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`) 发送一个事件（在本例中为 `click`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 79。

版本

Flash MX 2004。

用法

checkBoxInstance.labelPlacement

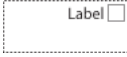
描述

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 78 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 版本 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：

```
checkboxbox1.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 组件拥有焦点并且可编辑时，所有键盘输入都会传递到文本框并根据 TextInput 组件（请参阅第 469 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（如果已在 Flash Player 的独立和浏览器版本中打开）。
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，并在“测试影片”模式下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，并且下拉列表将在独立或浏览器版本的 Flash Player 中关闭。
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。

按键	描述
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，并且下拉列表会在“测试影片”模式下关闭。
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 248 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，使其可由屏幕阅读器访问。首先，您必须添加以下代码行来启用辅助功能：

```
mx.accessibility.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”。

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共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一个城市时，您就可以加载这些文件。
- 5 在时间轴中选择第一帧，打开“动作”面板，然后输入以下代码：

```
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 25 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 62 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 471 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 266 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”。
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's website （Robert Penner 的 Web 站点）下载列表，地址为 www.robertpenner.com/easing/ 。

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

动作脚本类名称 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 和 UIComponent 中的所有方法。

ComboBox 类的属性摘要

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

继承 `UIObject` 和 `UIComponent` 中的所有属性。

ComboBox 类的事件摘要

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

继承 `UIObject` 和 `UIComponent` 中的所有事件。

ComboBox.addItem()

可用性

Flash Player 6 版本 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 版本 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 版本 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`) 发送一个事件（在本例中为 `change`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，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 510 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 版本 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 版本 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`) 发送一个事件（在本例中为 `close`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您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 510 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 版本 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"];
```

此范例将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 });
}
```

ComboBox.dropdown

可用性

Flash Player 6 版本 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 版本 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 版本 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 版本 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格广播。这是一个从组合格广播的 `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`) 发送一个事件（在本例中为 `enter`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。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 510 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 版本 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 版本 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) 发送一个事件（在本例中为 itemRollOut），而该事件由您创建的侦听器对象 (listenerObject) 上的函数（也称作处理函数）处理。您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 510 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 版本 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) 发送一个事件（在本例中为 itemRollOver），而该事件由您创建的侦听器对象 (listenerObject) 上的函数（也称作处理函数）处理。您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 510 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 版本 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 版本 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 版本 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 版本 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 `comboBox` 实例的下拉列表：

```
comboBox.open();
```

另请参见

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

ComboBox.open

可用性

Flash Player 6 版本 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`) 发送一个事件（在本例中为 `open`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您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 510 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 版本 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 版本 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 版本 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* 参数指定的索引位置的项目内容。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 版本 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 版本 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*) 发送一个事件（在本例中为 `scroll`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您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 510 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 版本 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 版本 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 版本 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 版本 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 版本 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);
```

数据绑定类（仅限 Flash Professional）

数据绑定类为 Flash MX Professional 2004 中的数据绑定特性提供了运行时功能。您可以使用 “组件检查器” 面板中的 “绑定” 选项卡在 Flash 创作环境中直观地创建和配置数据绑定，也可以使用 `mx.data.binding` 包中的类以编程方式创建和配置绑定。

有关数据绑定的概述，以及如何在 Flash 创作工具中直观地创建数据绑定，请参阅 “使用 Flash” 帮助中的 “数据绑定（仅限 Flash Professional）”。

使数据绑定类在运行时可用（仅限 Flash Professional）

为了能够在运行时使用数据绑定服务类，`DataBindingClasses` 组件必须位于 FLA 文件的库中。当您在 Flash 创作环境中直观地创建绑定时，此组件将自动添加到文档的库中。但是，如果只是使用动作脚本在运行时创建绑定，则您必须手动将此组件添加到文档的库中。有关如何将此组件添加到文档中的信息，请参阅 “使用 Flash” 帮助中的 “在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

mx.data.binding 包中的类（仅限 Flash Professional）

下表列出了 `mx.data.binding` 包中的类。

类	描述
Binding 类（仅限 Flash Professional）	在两个端点之间创建绑定。
ComponentMixins 类（仅限 Flash Professional）	将特定于数据绑定的功能添加到组件。
CustomFormatter 类（仅限 Flash Professional）	用于创建自定义格式程序类的基类。
CustomValidator 类（仅限 Flash Professional）	用于创建自定义验证程序类的基类。
DataType 类（仅限 Flash Professional）	提供对组件属性的数据字段的读写访问权限。

类	描述
EndPoint 类 （仅限 Flash Professional）	定义绑定的来源或目标。
TypedValue 类 （仅限 Flash Professional）	包含数据值和有关值的数据类型的信息。

Binding 类（仅限 Flash Professional）

动作脚本类名称 mx.data.binding.Binding

Binding 类定义两个端点（来源 和目标）之间的关联。它侦听来源端点的更改，并在每次来源更改时将更改后的数据复制到目标端点。

您可以使用 Binding 类（和提供支持的类）编写自定义绑定，也可以使用“组件检查器”面板中的“绑定”选项卡（“窗口” > “开发面板” > “组件检查器”）。

注意：要使此类在运行时可用，您必须在 FLA 文档中包括 DataBindingClasses 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 mx.data.binding 包中类的概述，请参阅第 108 页的“数据绑定类（仅限 Flash Professional）”。

Binding 类的方法摘要

方法	描述
Binding.execute()	从来源组件中获取数据，对数据进行格式化，然后将数据分配给目标组件。

Binding 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
new Binding(source, destination, [format], [isTwoWay])
```

参数

source 绑定的来源端点。此参数在名义上属于 mx.data.binding.EndPoint 类型，但可以是具有所需“端点”字段的任何动作脚本对象（请参阅 [EndPoint 类](#)（仅限 Flash Professional））。

destination 绑定的目标端点。此参数在名义上属于 mx.data.binding.EndPoint 类型，但可以是具有所需“端点”字段的任何动作脚本对象（请参阅 [EndPoint 类](#)（仅限 Flash Professional））。

format（可选）包含格式设置信息的对象。该对象必须具有以下属性：

- `cls` 扩展 mx.data.binding.DataAccessor 类的动作脚本类。
- `settings` 一个对象，其属性为 `cls` 指定的格式程序类提供可选设置。

isTwoWay (可选) 布尔值, 指定新的 Binding 对象是 (true) 否 (false) 为双向。默认值为 false。

返回

无。

说明

构造函数；创建新的 Binding 对象。可以将数据绑定到具有属性并发出事件的任何动作脚本对象（包括但不限于组件）。

只要处于最里层的影片剪辑同时包含来源和目标组件，就会存在绑定对象。例如，如果名为“A”的影片剪辑包含组件“X”和“Y”，并且“X”和“Y”之间存在绑定，则只要影片剪辑 A 存在，绑定就有效。

注意：不必保留对新 Binding 对象的引用（尽管您可以这样做）。Binding 对象一经创建就会立即开始侦听任一“端点”发出的“已更改”事件。但是，在某些情况下，您可能需要保存对新 Binding 对象的引用，以便稍后能够调用它的 `execute()` 方法（请参阅 [Binding.execute\(\)](#)）。

示例

范例 1：在此范例中，一个 TextInput 组件 (`src_txt`) 的 `text` 属性被绑定到另一个 TextInput 组件 (`dest_txt`) 的 `text` 属性。在 `src_txt` 文本字段失去焦点时（也就是说，在生成 `focusOut` 事件时），它的 `text` 属性的值将被复制到 `dest_txt.text` 中。

```
import mx.data.binding.*;
var src = new EndPoint();
src.component = src_txt;
src.property = "text";
src.event = "focusOut";

var dest= new EndPoint();
dest.component = dest_txt;
dest.property = "text";

new Binding(src, dest);
```

范例 2：此范例演示如何创建使用自定义格式程序类的 Binding 对象。有关创建自定义格式程序类的详细信息，请参阅第 111 页的“[CustomFormatter 类（仅限 Flash Professional）](#)”。

```
import mx.data.binding.*;
var src = new EndPoint();
src.component = src_txt;
src.property = "text";
src.event = "focusOut";

var dest= new EndPoint();
dest.component = text_dest;
dest.property = "text";

new Binding(src, dest, {cls:mx.data.formatters.Custom,
    settings:{classname:"com.mycompany.SpecialFormatter"}});
```

Binding.execute()

可用性

Flash Player 6。

版本

Flash MX Professional 2004。

用法

```
myBinding.execute([reverse])
```

参数

reverse 布尔值，指定是同时执行从目标到来源的绑定 (true)，还是只执行从来源到目标的绑定 (false)。默认情况下，此值为 false。

返回

如果绑定成功执行，则值为 null；否则将会返回一个错误消息（字符串）数组，这些错误消息描述阻碍绑定执行的一个或多个错误。

说明

方法；从来源组件获取数据，并将其分配给目标组件。如果绑定使用格式程序，则会在将数据分配给目标之前对其进行格式化。

此方法还会对数据进行验证，并会造成目标和来源组件发出 valid 或 invalid 事件。数据即使无效也会被分配给目标，除非目标是只读的。

如果 *reverse* 参数设置为 true，并且绑定是双向的，则会反向执行绑定（从目标到来源）。

示例

Button 组件实例附带的以下代码将在按钮被单击时反向执行绑定（从目标组件到来源组件）。

```
on(click){  
    _root.myBinding.execute(true);  
}
```

CustomFormatter 类（仅限 Flash Professional）

动作脚本类名称 mx.data.binding.CustomFormatter

CustomFormatter 类定义两个方法（format() 和 unformat()），它们提供了将数据值从特定数据类型转换为 String（反之亦然）的能力。默认情况下，这些方法不会执行任何操作；您必须在 mx.data.binding.CustomFormatter 的子类中实现它们。

要创建自己的自定义格式程序，您需要首先创建实现 format() 和 unformat() 方法的 CustomFormatter 的子类。然后，您可以将该类分配给组件之间的绑定，方法是使用动作脚本创建一个新的 Binding 对象（请参阅第 109 页的“[Binding 类（仅限 Flash Professional）](#)”），或使用“组件检查器”面板中的“绑定”选项卡。有关使用组件检查器分配格式程序类的信息，请参阅“使用 Flash”帮助中的“[架构格式程序（仅限 Flash Professional）](#)”。

您也可以在“组件检查器”面板的“架构”选项卡上将格式程序类分配给组件属性。但是，在这种情况下，只有在需要字符串格式的数据时才会使用格式程序。相反，使用“绑定”面板分配或通过动作脚本创建的格式程序会在每次执行绑定时使用。

有关使用动作脚本编写和分配自定义格式程序的范例，请参阅第 112 页的“自定义格式程序范例”。

注意：要使此类在运行时可用，您必须在 FLA 文档中包括 DataBindingClasses 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 mx.data.binding 包中类的概述，请参阅第 108 页的“数据绑定类（仅限 Flash Professional）”。

自定义格式程序范例

以下范例演示如何创建自定义格式程序类，然后使用动作脚本将其应用到两个组件之间的绑定。在此范例中，NumericStepper 组件的当前值（其 value 属性）被绑定到 TextInput 组件的当前值（其 text 属性）。在将 NumericStepper 组件的当前数字值（例如，1、2 或 3）分配给 TextInput 组件之前，自定义格式程序类会将该值格式化为其对等的英文单词（例如，“one”、“two”或“three”）。

要创建和使用自定义格式程序：

- 1 在 Flash MX Professional 2004 中，创建一个新的动作脚本文件。
- 2 将以下代码添加到该文件：

```
// NumberFormatter.as
class NumberFormatter extends mx.data.binding.CustomFormatter {
    // 格式化数字，返回字符串
    function format(rawValue) {
        var returnValue;
        var strArray = new Array('one', 'two', 'three');
        var numArray = new Array(1, 2, 3);
        returnValue = 0;
        for (var i = 0; i < strArray.length; i++) {
            if (rawValue == numArray[i]) {
                returnValue = strArray[i];
                break;
            }
        }
        return returnValue;
    } // 转换格式化后的值，返回原始值
    function unformat(formattedValue) {
        var returnValue;
        var strArray = new Array('one', 'two', 'three');
        var numArray = new Array(1, 2, 3);
        returnValue = "invalid";
        for (var i = 0; i < strArray.length; i++) {
            if (formattedValue == strArray[i]) {
                returnValue = numArray[i];
                break;
            }
        }
        return returnValue;
    }
}
```

- 3 将动作脚本文件另存为 NumberFormatter.as。
- 4 创建新的 Flash (FLA) 文档。
- 5 打开“组件”面板（“窗口” > “开发面板” > “组件”）。
- 6 将 TextInput 组件拖到舞台上，然后将其命名为 textInput。
- 7 将 NumericStepper 组件拖到舞台上，然后将其命名为 stepper。

- 8 打开 “时间轴” （“窗口” > “时间轴”）并选择第 1 层上的第一帧。
- 9 打开 “动作” 面板 （“窗口” > “开发面板” > “动作”）。
- 10 将以下代码添加到 “动作” 面板：

```
import mx.data.binding.*;
var x:NumberFormatter;
var customBinding = new Binding({component:stepper, property:"value",
    event:"change"}, {component:textInput, property:"text",
    event:"enter,change"}, {cls:mx.data.formatters.Custom,
    settings:{classname:"NumberFormatter"}});
```

代码的第二行 (var x:NumberFormatter) 确保编译后的 SWF 文件中包含自定义格式程序类的字节代码。

- 11 选择 “窗口” > “面板” > “其他面板” > “类” 以打开 “类” 库。
- 12 通过选择 “窗口” > “库” 打开文档的库。
- 13 将 DataBindingClasses 组件从 “类” 库拖到文档的库中。
这将使文档可以使用数据绑定运行时类。。有关详细信息，请参阅 “使用 Flash” 帮助中的 “在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。
- 14 将 FLA 文件保存到包含 NumberFormatter.as 的同一文件夹中。
- 15 测试文件 （“控制” > “测试影片”）。
单击 NumericStepper 组件上的按钮，并观看 TextInput 组件的内容发生更新。

CustomFormatter 类的方法摘要

方法	描述
<code>CustomFormatter.format()</code>	从原始数据类型转换为文本字符串。
<code>CustomFormatter.unformat()</code>	从文本字符串转换为原始数据类型。

CustomFormatter.format()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

此方法是自动调用的；您不必直接调用它。

参数

rawData 要格式化的数据。

返回

格式化后的值。

说明

方法；从原始数据类型转换为新的对象。

此方法不会默认执行。您必须在 `mx.data.binding.CustomFormatter` 的子类中定义此方法。

示例

请参阅第 112 页的“自定义格式程序范例”。

CustomFormatter.unformat()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

此方法是自动调用的；您不必直接调用它。

参数

formattedData 要转换回原始数据类型的已格式化数据。

返回

取消格式的值。

说明

方法；从字符串（或其他数据类型）转换为原始数据类型。此转换应执行与 `CustomFormatter.format()` 恰好相反的转换。

此方法不会默认执行。您必须在 `mx.data.binding.CustomFormatter` 的子类中定义此方法。有关详细信息，请参阅第 112 页的“自定义格式程序范例”。

CustomValidator 类（仅限 Flash Professional）

动作脚本类名称 `mx.data.binding.CustomValidator`

在需要对组件包含的数据字段执行自定义验证时，可以使用 `CustomValidator` 类。

要创建自定义验证类，您需要首先创建实现名为 `validate()` 的方法的 `mx.data.binding.CustomValidator` 的子类。待验证的值会自动传递到此方法。有关如何实现此方法的详细信息，请参阅 `CustomValidator.validate()`。

接着，您需要使用“组件检查器”面板的“架构”选项卡将自定义验证程序类分配给组件的字段。有关创建和使用自定义验证程序类的范例，请参阅 `CustomValidator.validate()` 条目中的“范例”部分。

要分配自定义验证程序，请执行以下操作：

- 1 在“组件检查器”面板（“窗口” > “组件检查器”）中，选择“架构”选项卡。
- 2 选择要验证的字段，然后从“数据类型”弹出菜单中选择“自定义”。
- 3 选择“验证选项”字段（位于“架构”选项卡的底部），然后单击放大镜图标打开“自定义验证设置”对话框。
- 4 在“动作脚本类”文本框中输入您创建的自定义验证类的名称。

为了使指定的类包含在发布的 SWF 内，它必须位于类路径中。

注意：要使该类在运行时可用，您必须在 FLA 文档中包括 DataBindingClasses 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 mx.data.binding 包中类的概述，请参阅第 108 页的“数据绑定类（仅限 Flash Professional）”。

CustomValidator 类的方法摘要

方法	描述
<code>CustomValidator.validate()</code>	对数据执行验证。
<code>CustomValidator.validationError()</code>	报告验证错误。

CustomValidator.validate()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

此方法是自动调用的；您不必直接调用它。

参数

value 要验证的数据；可以是任何类型。

返回

无。

说明

方法；自动调用以验证 *value* 参数包含的数据。必须在 CustomValidator 的子类中实现此方法；默认实现不会执行任何操作。

可以使用任何所需的动作脚本代码检查和验证数据。如果数据无效，此方法应调用带有相应消息的 `this.validationError()`。如果数据存在多个验证问题，则可以调用 `this.validationError()` 多次。

由于 `validate()` 方法可以被重复调用，因此应避免将需要很长时间才能完成的代码添加到此方法。此方法的实现只应检查有效性，然后使用 `CustomValidator.validationError()` 报告任何错误。同样，这种实现不应作为验证测试的任何结果（如警示最终用户），而应该为 `valid` 和 `invalid` 事件创建事件侦听器，然后从这些事件侦听器中警示最终用户（请参阅下面的范例）。

示例

以下过程演示如何创建和使用自定义验证类。CustomValidator 类的 `validate()` 方法 `OddNumbersOnly.as` 将不是奇数的任何值确定为无效。验证在 NumericStepper 组件的值发生更改时进行，该组件已绑定到 Label 组件的 `text` 属性。

要创建和使用自定义验证程序类：

- 1 在 Flash MX Professional 2004 中，创建一个新的动作脚本 (AS) 文件。
- 2 将以下代码添加到该 AS 文件：

```

class OddNumbersOnly extends mx.data.binding.CustomValidator
{
    public function validate(value) {
        // 确保值为数字
        var n = Number(value);
        if (String(n) == "NaN") {
            this.validationError("'" + value + "' is not a number.");
            return;
        }
        // 确定数字为奇数
        if (n % 2 == 0) {
            this.validationError("'" + value + "' is not a odd number.");
            return;
        }
        // 数据无误，无需执行任何操作，只需返回
    }
}

```

3 将该 AS 文件另存为 OddNumbersOnly.as。

注意：AS 文件的名称必须与类的名称匹配。

4 创建新的 Flash (FLA) 文档。

5 打开“组件”面板（“窗口” > “开发面板” > “组件”）。

6 将 NumericStepper 组件从“组件”面板拖到舞台上，然后将其命名为 **stepper**。

7 将 Label 组件拖到舞台上，然后将其命名为 **textLabel**。

8 将 TextArea 组件拖到舞台上，然后将其命名为 **status**。

9 选择 NumericStepper 组件，然后打开“组件检查器”面板（“窗口” > “开发面板” > “组件检查器”）。

10 在“组件检查器”面板中选择“绑定”选项卡，然后单击“添加绑定” (+) 按钮。

11 在“添加绑定”对话框中选择 Value 属性（唯一的属性），然后单击“确定”。

12 在“组件检查器”面板中，双击“绑定”选项卡“绑定属性”窗格中的“绑定到”，打开“绑定到”对话框。

13 在“绑定到”对话框的“组件路径”窗格中选择 Label 组件，并在“架构位置”窗格中选择其 text 属性。单击“确定”。

14 在舞台上选择 Label 组件，然后单击“组件检查器”面板中的“架构”选项卡。

15 在“架构属性”窗格中，从“数据类型”弹出菜单中选择“自定义”。

16 双击“架构属性”窗格中的“验证选项”字段，打开“自定义验证设置”对话框。

17 在“动作脚本类”文本框中输入 **OddNumbersOnly**，即之前创建的动作脚本类的名称。单击“确定”。

18 打开“时间轴”（“窗口” > “时间轴”）并选择第 1 层上的第一帧。

19 打开“动作”面板（“窗口” > “动作”）。

20 将以下代码添加到“动作”面板：

```

function dataIsValid(evt) {
    if (evt.property == "text") {
        status.text = evt.messages;
    }
}

function dataIsValid(evt) {
    if (evt.property == "text") {
        status.text = "OK";
    }
}

```

```

    }
}

textField.addEventListener("valid", dataIsValid);
textField.addEventListener("invalid", dataIsInvalid);

```

21 将 FLA 文件另存为 OddOnly.fla，保存到包含 OddNumbersOnly.as 的同一文件夹中。

22 测试 SWF（“控制” > “测试影片”）。

单击 NumericStepper 组件上的箭头更改它的值。注意在选择偶数和奇数时 TextArea 组件中出现的消息。

CustomValidator.validationError()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
this.validationError(errorMessage)
```

注意：只能从自定义验证程序类中调用此方法；关键字 `this` 指当前的 CustomValidator 对象。

参数

errorMessage 包含要报告的错误消息的字符串。

返回

无。

说明

方法；可以从 CustomValidator 的子类的 `validate()` 方法中调用此方法，以便报告验证错误。如果不调用此方法，则会在 `validate()` 完成时生成 `valid` 事件。如果从 `validate()` 方法内调用此方法一次或多次，则在 `validate()` 返回后会生成 `invalid` 事件。

传递给 `validationError()` 的每条消息都可在传递到 `invalid` 事件处理函数的事件对象的“messages”属性中找到。

示例

请参阅 [CustomValidator.validate\(\)](#) 的“范例”部分。

EndPoint 类（仅限 Flash Professional）

动作脚本类名称 mx.data.binding.EndPoint

EndPoint 类定义绑定的来源或目标。EndPoint 对象定义常数值、组件属性或组件属性的特定字段，您可以从中获取数据或将数据分配到其中。它们还能够定义 Binding 对象侦听的事件或事件列表；在发生指定事件时，绑定即会执行。

当使用 Binding 类构造函数创建新的绑定时，您需要为其传递两个 EndPoint 对象：一个用于来源，一个用于目标。

```
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```

EndPoint 对象 `srcEndPoint` 和 `destEndPoint` 可按以下方式进行定义：

```
var srcEndPoint = new mx.data.binding.EndPoint();
var destEndPoint = new mx.data.binding.EndPoint();
srcEndPoint.component = source_txt;
srcEndPoint.property = "text";
srcEndPoint.event = "focusOut";
destEndPoint.component = dest_txt;
destEndPoint.property = "text";
```

如果用文字表述，上面代码的意思是“在来源文本字段失去焦点时，将其 `text` 属性的值复制到目标文本字段的 `text` 属性上”。

您也可以将一般动作脚本对象传递给 `Binding` 构造函数，而不是传递明确构造的 `EndPoint` 对象。唯一的要求是这些对象应定义所需的 `EndPoint` 属性，即 `component` 和 `property`。以下代码与上面显示的代码等效。

```
var srcEndPoint = {component:source_txt, property:"text"};
var destEndPoint = {component:dest_txt, property:"text"};
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```

注意：要使此类在运行时可用，您必须在 FLA 文档中包括 `DataBindingClasses` 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 `mx.data.binding` 包中类的概述，请参阅[第 108 页的“数据绑定类（仅限 Flash Professional）”](#)。

EndPoint 类的属性摘要

方法	描述
EndPoint.constant	一个常数值。
EndPoint.component	指向组件实例的引用。
EndPoint.property	EndPoint.component 指定的组件实例的属性名称。
EndPoint.location	数据字段在组件实例的属性中的位置。
EndPoint.event	组件实例在数据更改时将发出的事件或事件列表的名称。

EndPoint 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`new EndPoint()`

返回

无。

说明

构造函数；创建新的 `EndPoint` 对象。

示例

此范例将创建名为 `source_txt` 的新 `EndPoint` 对象，并为其 `component` 和 `property` 属性赋值。

```
var source_obj = new mx.data.binding.EndPoint();
source_obj.component = myTextField;
source_obj.property = "text";
```

EndPoint.constant

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
endPoint_src.constant
```

说明

属性；分配给 `EndPoint` 对象的常数值。此属性只能应用于作为组件间绑定的来源（而不是目标）的“端点”。值可以是与绑定目标兼容的任何数据类型。如果指定了该值，则指定 `EndPoint` 对象的所有其他 `EndPoint` 属性将被忽略。

示例

在本范例中，字符串常数值“hello”被分配给 `EndPoint` 对象的 `constant` 属性。

```
var sourceEndPoint = new mx.data.binding.EndPoint();
sourceEndPoint.constant="hello";
```

EndPoint.component

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
endPointObj.component
```

说明

属性；指向组件实例的引用。

示例

此范例将 `List` 组件的某个实例 (`listBox1`) 分配为 `EndPoint` 对象的组件参数。

```
var sourceEndPoint = new mx.data.binding.EndPoint();
sourceEndPoint.component=listBox1;
```

EndPoint.property

可用性

Flash Player 6 版本 79

版本

Flash MX Professional 2004。

用法

endPointObj.property

说明

属性；指定由 `EndPoint.component`（包含可绑定数据）指定的组件实例的名称。

注意：`EndPoint.component` 和 `EndPoint.property` 必须结合使用才能形成有效的动作脚本对象 / 属性组合。

示例

此范例将一个 TextInput 组件 (text_1) 的 text 属性绑定到另一个 TextInput 组件 (text_2) 的相同属性。

```
var sourceEndPoint = {component:text_1, property:"text"};
var destEndPoint = {component:text_2, property:"text"};
new Binding(sourceEndPoint, destEndPoint);
```

EndPoint.location

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

endPointObj.location

说明

属性；指定数据字段在组件实例的属性中的位置。可以以四种方式来指定位置：作为包含 XPath 表达式或动作脚本路径的字符串、字符串数组或对象。

只有在数据是 XML 对象时，才能使用 XPath 表达式。有关支持的 XPath 表达式的列表，请参阅“使用 Flash”帮助中的“支持的 XPath 表达式”。（请参阅下面的范例 1。）

对于 XML 和动作脚本对象，您还可以指定包含动作脚本路径的字符串。动作脚本路径包含由点分隔的字段名称（例如，"a.b.c"）。

您也可以指定字符串数组作为位置。数组中的每个字符串均“下寻”到另一个嵌套级别。可以将此技术用于 XML 和动作脚本数据。（请参阅下面的范例 2。）当用于动作脚本数据时，字符串数组相当于使用动作脚本；也就是说，数组 ["a","b","c"] 相当于 "a.b.c"。

如果将某个对象指定为位置，则该对象必须指定两个属性：path 和 indices。如上所述，path 属性是一个字符串数组，除了一个或多个指定字符串可能是特殊标记 "[n]"。对于 path 中出现的每个这种标记，在 indices 中必须有对应的索引项。在计算路径的值时，这些索引用于为数组建立索引。索引项可以是任意“端点”。这种类型的位置只能应用于动作脚本数据，而不能应用于 XML。（请参阅下面的范例 3。）

示例

范例 1：此范例使用 XPath 表达式在 XML 对象中指定名为 zip 的节点的位置。

```
var sourceEndPoint = new mx.databinding.EndPoint();
var sourceObj=new Object();
sourceObj.xml=new XML("<zip>94103</zip>");
sourceEndPoint.component=sourceObj;
sourceEndPoint.property="xml";
sourceEndPoint.location="/zip";//
```

范例 2：此范例使用字符串数组“下寻”到嵌套的影片剪辑属性。

```
var sourceEndPoint = new mx.data.binding.EndPoint();
// 假设 movieClip1.ball.position 存在
sourceEndPoint.component=movieClip1;
sourceEndPoint.property="ball";
// 访问 movieClip1.ball.position.x
sourceEndPoint.location=["position","x"];
```

范例 3：此范例显示如何使用对象在复杂的数据结构中指定数据字段的位置。

```
var city=new Object();
city.theaters = [{theater:"t1", movies:[{name:"Good,Bad,Ugly"}, {name:"Matrix Reloaded"}]}, {theater:"t2", movies:[{name:"Gladiator"}, {name:"Catch me if you can"}]}}];
var srcEndPoint = new EndPoint();
srcEndPoint.component=city;
srcEndPoint.property="theaters";
srcEndPoint.location = {path:["[n]","movies","[n]","name"],
    indices:[{constant:0},{constant:0}]};
```

EndPoint.event

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

endPointObj.event

说明

属性；指定在分配给绑定属性的数据更改时由组件生成的事件名称或事件名称数组。当事件发生时，绑定即会执行。

指定的事件仅适用于用作绑定来源或双向绑定的目标的组件。有关创建双向绑定的详细信息，请参阅第 109 页的“Binding 类（仅限 Flash Professional）”。

示例

在此范例中，一个 TextInput 组件 (src_txt) 的 text 属性被绑定到另一个 TextInput 组件 (dest_txt) 的相同属性。当 src_txt 组件发出 focusOut 或 enter 事件时，绑定即会执行。

```
var source = {component:src_txt, property:"text", event:["focusOut", "enter"]};
var dest = {component:myTextArea, property:"text"};
var newBind = new mx.data.binding.Binding(source, dest);
```

ComponentMixins 类（仅限 Flash Professional）

动作脚本类名称 mx.data.binding.ComponentMixins

ComponentMixins 类定义自动添加到任何对象（充当绑定的来源或目标）或任何组件（充当 `ComponentMixins.initComponent()` 方法调用的目标）的属性和方法。这些属性和方法不会影响正常的组件功能；相反，它们还添加了对数据绑定十分有用的功能。

注意：要使用此类在运行时可用，您必须在 FLA 文档中包括 DataBindingClasses 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 mx.data.binding 包中类的概述，请参阅第 108 页的“数据绑定类（仅限 Flash Professional）”。

ComponentMixins 类的方法摘要

方法	描述
<code>ComponentMixins.getField()</code>	返回用于在组件属性的指定位置获取和设置字段值的对象。
<code>ComponentMixins.initComponent()</code>	将 ComponentMixin 方法添加到组件。
<code>ComponentMixins.refreshFromSources()</code>	执行所有将此组件作为目标“端点”的绑定。
<code>ComponentMixins.refreshDestinations()</code>	执行所有将此对象作为来源“端点”的绑定。
<code>ComponentMixins.validateProperty()</code>	检查以确定指定属性中的数据是否有效。

ComponentMixins.getField()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.getField(propertyName, [location])
```

参数

propertyName 包含指定组件的属性名称的字符串。

location （可选）字段在组件属性内的位置。如果 *propertyName* 指定的组件属性是复杂的数据结构，并且您对该结构的特定字段感兴趣，则此参数十分有用。此属性可以是以下三种格式之一：

- 包含 XPath 表达式的字符串。这种格式仅对 XML 数据结构有效。有关支持的 XPath 表达式的列表，请参阅“使用 Flash”帮助中的“支持的 XPath 表达式”。
- 包含由点分隔的字段名（例如，“a.b.c”）的字符串。此形式允许用于任何复杂数据（动作脚本或 XML）。
- 字符串数组，其中每个字符串都是一个字段名，例如，["a", "b", "c"]。此形式允许用于任何复杂数据（动作脚本或 XML）。

返回

一个 `DataType` 对象。

说明

方法；返回一个 `DataType` 对象，您可以使用该对象的方法在组件属性的指定字段位置获取或设置数据值。有关详细信息，请参阅第 126 页的“[DataType 类（仅限 Flash Professional）](#)”。

示例

此范例使用 `DataType.setAsString()` 方法设置位于组件属性中的字段的值。在这种情况下，属性 (`results`) 是一种复杂数据结构。

```
import mx.data.binding.*;
var field :DataType = myComponent.getField("results", "po.address.name1");
field.setAsString("Teri Randall");
```

另请参见

[DataType.setAsString\(\)](#)

ComponentMixins.initComponent()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mx.data.binding.ComponentMixins.initComponent(componentInstance)
```

参数

componentInstance 指向组件实例的引用。

返回

无。

说明

方法（静态）；将所有 `ComponentMixins` 方法添加到 `componentInstance` 指定的组件中。系统将为数据绑定中涉及的所有组件自动调用此方法。要使 `ComponentMixins` 方法可供数据绑定中未涉及的某个组件使用，您必须明确地为该组件调用此方法。

示例

以下代码使 `ComponentMixins` 方法可供 `DataSet` 组件使用。

```
mx.data.binding.ComponentMixins.initComponent(_root.myDataSet);
```

ComponentMixins.refreshFromSources()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.refreshSources()
```

返回

无。

说明

方法；执行 `componentInstance` 是目标 `EndPoint` 对象的所有绑定。此方法使您能够执行具有常数来源或未发出任何“数据已更改”事件的来源的绑定。

示例

以下范例执行 `ListBox` 组件实例（名为 `cityList`）是目标 `EndPoint` 对象的所有绑定。

```
cityList.refreshFromSources();
```

ComponentMixins.refreshDestinations()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.refreshDestinations()
```

返回

无。

说明

方法；执行 `componentInstance` 是来源“端点”的所有绑定。此方法使您能够执行来源未发出“数据已更改”事件的绑定。

示例

以下范例执行 DataSet 组件实例（名为 `user_data`）是来自 EndPoint 对象的所有绑定。

```
user_data.refreshDestinations();
```

ComponentMixins.validateProperty()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.validateProperty(propertyName)
```

参数

propertyName 字符串，包含属于 *componentInstance* 的属性的名称。

返回

一个数组，或 `null`。

说明

方法；确定 *propertyName* 中的数据依据属性的架构设置是否有效。属性的架构设置是指“组件检查器”面板的“架构”选项卡上所指定的内容。

如果数据有效，方法将返回 `null`；否则将返回错误消息数组作为字符串。

验证仅适用于具有可用架构信息的字段。如果某个字段是包含其他字段的对象，则会以递归方式验证每个“子”字段，依此类推。每个单独的字段都会在必要时发送 `valid` 或 `invalid` 事件。对于 *propertyName* 包含的每个数据字段，此函数将发送 `valid` 或 `invalid` 事件，如下所示：

- 如果字段的值为 `null`，并且不是必需的，则方法将返回 `null`。不会生成任何事件。
- 如果字段的值为 `null`，并且是必需的，则会返回错误并生成 `invalid` 事件。
- 如果字段的值非空并且字段的架构没有验证程序，则方法将返回 `null`；不会生成任何事件。
- 如果值非空并且字段的架构已定义了验证程序，则数据将由验证程序对象处理。如果数据有效，则会生成 `valid` 事件并返回 `null`；否则生成 `invalid` 事件并返回错误字符串数组。

示例

以下范例显示如何使用 `validateProperty()` 来确保用户所输入文本的长度有效。您将通过在“组件检查器”面板的“架构”选项卡中为字符串数据类型设置“验证选项”来确定有效长度。如果用户在文本字段中输入了长度无效的字符串，则 `validateProperty()` 方法返回的错误消息将显示在“输出”面板中。

要对用户在 TextInput 组件中输入的文本进行验证：

- 1 将 TextInput 组件从“组件”面板（“窗口” > “开发面板” > “组件”）拖到舞台上，然后将其命名为 `zipCode_txt`。
- 2 选择 TextInput 组件，然后在“组件检查器”面板（“窗口” > “开发面板” > “组件”）中单击“架构”选项卡。
- 3 在“架构树”窗格（“架构”选项卡的顶部窗格）中，选择 `text` 属性。

- 4 在“架构属性”窗格（“架构”选项卡的底部窗格）中，从“数据类型”弹出菜单中选择“ZipCode”。
- 5 通过选择“窗口” > “时间轴”打开时间轴（如果尚未打开）。
- 6 单击时间轴中“第 1 层”上的第一帧，然后打开“动作”面板（“窗口” > “动作”）。
- 7 将以下代码添加到“动作”面板：

```
// 将 ComponentMixin 方法添加到 TextInput 组件。  
// 请注意，只有在组件尚未作为  
// 来源或目标涉及在数据绑定中时，  
// 才必须进行此步骤。  
mx.data.binding.ComponentMixins.initComponent(zipCode_txt);  
// 为组件定义事件侦听器函数：  
validateResults = function (eventObj) {  
    var errors:Array = eventObj.target.validateProperty("text");  
    if (errors != null) {  
        trace(errors);  
    }  
};  
// 将侦听器函数注册到组件：  
zipCode_txt.addEventListener("enter", validateResults);
```

- 8 选择“窗口” > “其他面板” > “公用库” > “类”以打开“类”库。
- 9 通过选择“窗口” > “库”打开文档的库。
- 10 将 DataBindingClasses 组件从“类”库拖到文档的“库”面板中。
要使数据绑定运行时类可供 SWF 在运行时使用，则必须执行此步骤。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。
- 11 通过选择“控制” > “测试影片”来测试 SWF。
在舞台上的 TextInput 组件中，输入一个无效的美国邮政编码，例如，全部由字母组成或包含少于五个数字的邮政编码。注意“输出”面板中显示的错误消息。

DataType 类（仅限 Flash Professional）

动作脚本类名称 mx.data.binding.DataType

DataType 类提供对组件属性的数据字段的读写权限。要获取 DataType 对象，您需要对组件调用 `ComponentMixins.getField()` 函数。然后，您可以调用 DataType 对象的方法来获取和设置字段的值。

使用 DataType 对象方法获取和设置字段值以及直接在组件实例上获取或设置值之间的差异在于：后一种情况以“原始”形式提供数据。相反，在使用 DataType 类的方法获取或设置字段值时，系统将根据字段的架构设置处理这些值。

例如，以下代码将直接获取组件属性的值，并将其赋给变量。变量 `propVar` 包含作为 `propName` 属性的当前值的任何“原始”值。

```
var propVar = myComponent.propName;
```

下一范例将使用 `DataType.getAsString()` 方法获取同一属性的值。在这种情况下，分配给 `stringVar` 的值就是根据架构设置处理后作为字符串返回的 `propName` 的值。

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");  
var stringVar:String = dataTypeObj.getAsString();
```

有关如何指定字段的架构设置的详细信息，请参阅“使用 Flash”帮助中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

您也可以使用 `DataType` 类的方法来获取或设置不同数据类型的字段。如果可能，`DataType` 类会自动将原始数据转换为要求的类型。例如，在上面的代码范例中，检索出的数据被转换为字符串类型（即使原始数据是另一种类型）。

`ComponentMixins.getField()` 方法可供已包括在数据绑定中（作为来源、目标或索引）或已使用 `ComponentMixins.initComponent()` 方法初始化的组件使用。有关详细信息，请参阅第 122 页的“[ComponentMixins 类（仅限 Flash Professional）](#)”。

注意：要使此类在运行时可用，您必须在 FLA 文档中包括 `DataBindingClasses` 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 `mx.data.binding` 包中类的概述，请参阅第 108 页的“[数据绑定类（仅限 Flash Professional）](#)”。

DataType 类的方法摘要

方法	描述
<code>DataType.getAsBoolean()</code>	以布尔值的形式获取字段的当前值。
<code>DataType.getAsNumber()</code>	以数值的形式获取字段的当前值。
<code>DataType.getAsString()</code>	以字符串值的形式获取字段的当前值。
<code>DataType.getAnyTypedValue()</code>	获取字段的当前值。
<code>DataType.getTypedValue()</code>	以要求数据类型的形式获取字段的当前值。
<code>DataType.setAnyTypedValue()</code>	在该字段中设置一个新值。
<code>DataType.setAsBoolean()</code>	将字段设置为以布尔值形式提供的新值。
<code>DataType.setAsNumber()</code>	将字段设置为以数值形式提供的新值。
<code>DataType.setAsString()</code>	将字段设置为以字符串形式提供的新值。
<code>DataType.setTypedValue()</code>	在该字段中设置一个新值。

DataType 类的属性摘要

属性	描述
<code>DataType.encoder</code>	提供对与此字段关联的 <code>Encoder</code> 对象的引用。
<code>DataType.formatter</code>	提供对与此字段关联的 <code>Formatter</code> 对象的引用。
<code>DataType.kind</code>	提供对与此字段关联的 <code>Kind</code> 对象的引用。

DataType.encoder

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`dataTypeObject.encoder`

说明

属性；提供对与此字段关联的编码器对象的引用（如果存在）。可以使用此属性访问由特定编码器（应用于“组件检查器”面板的“架构”选项卡中的字段）定义的任何属性和方法。

如果未将任何编码器应用于所述字段，则此属性将返回 `undefined`。

有关随 Flash MX Professional 2004 提供的编码器的详细信息，请参阅“使用 Flash”帮助中的“架构编码器（仅限 Flash Professional）”。

示例

以下范例假设所访问的字段 (`isValid`) 使用布尔值编码器 (`mx.data.encoders.Bool`)。此编码器随 Flash MX Professional 2004 提供，并包含名为 `trueStrings` 的属性，该属性指定应将哪些字符串解释为布尔值 `true`。下面的代码将字段编码器的 `trueStrings` 属性设置为字符串“yes”和“si”。

```
var myField:mx.data.binding.DataType = dataSet.getField("isValid");
myField.encoder.trueStrings = "Yes,Oui";
```

DataType.formatter

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

dataTypeObject.formatter

说明

属性；提供对与字段关联的格式程序对象的引用（如果存在）。可以使用此属性访问格式程序对象（应用于“组件检查器”面板“架构”选项卡中的字段）的任何属性和方法。

如果未将任何格式程序应用于所述字段，则此属性将返回 `undefined`。

有关随 Flash MX Professional 2004 提供的格式程序的详细信息，请参阅“使用 Flash”帮助中的“架构格式程序（仅限 Flash Professional）”。

示例

此范例假设所访问的字段正在使用随 Flash MX Professional 2004 提供的数值格式程序 (`mx.data.formatters.NumberFormatter`)。此格式程序包含一个名为 `precision` 的属性，该属性指定小数点后显示的数字位数。以下代码为使用此格式程序的字段将 `precision` 属性设置为两位小数。

```
var myField:DataType = dataGrid.getField("currentBalance");
myField.formatter.precision = 2;
```

DataType.getAsBoolean()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.getAsBoolean()
```

返回

一个布尔值。

说明

方法；以布尔值的形式获取字段的当前值。如有必要，该值会被转换为布尔值形式。

示例

在此范例中，将以布尔值的形式检索名为 `propName` 的字段（属于名为 `myComponent` 的组件），并赋给变量。

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");  
var propValue:Boolean = dataTypeObj.getAsBoolean();
```

DataType.getAsNumber()

可用性

Flash Player 6。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.getAsNumber()
```

返回

一个数字。

说明

方法；以数值的形式获取字段的当前值。如有必要，该值会被转换为数值形式。

示例

在此范例中，将以数值的形式检索名为 `propName` 的字段（属于名为 `myComponent` 的组件），并赋给变量。

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");  
var propValue:Number = dataTypeObj.getAsNumber();
```

另请参见

[DataType.getAnyTypedValue\(\)](#)

DataType.getString()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.getAsString()
```

返回

一个字符串。

说明

方法；以字符串的形式获取字段的当前值。如有必要，该值会被转换为字符串形式。

示例

在此范例中，将以字符串的形式检索名为 `propName` 的组件（属于名为 `myComponent` 的组件）的属性，并赋给变量。

```
var dataTypeObj:mx.data.binding.DataType = myComponent.getField("propName");  
var propValue:String = dataTypeObj.getAsString();
```

另请参见

[DataType.getAnyTypedValue\(\)](#)

DataType.getAnyTypedValue()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.getAnyTypedValue(suggestedTypes)
```

参数

suggestedTypes 字符串数组，按需要程度依降序为字段指定所需的首选数据类型。有关详细信息，请参阅下面的“说明”部分。

返回

字段的当前值，采用 *suggestedTypes* 数组中指定的某个数据类型的形式。

说明

方法；获取字段的当前值，并使用字段架构中的信息来处理值。如果字段能够提供值作为 *suggestedTypes* 数组中指定的第一个数据类型，则方法将返回字段的值作为该数据类型。否则，方法将尝试抽取字段的值作为 *suggestedTypes* 数组中指定的第二个数据类型，依此类推。

如果指定 `null` 作为 *suggestedTypes* 数组中的项目之一，则方法将以“架构”面板中指定的数据类型返回字段的值。指定 `null` 将始终会导致正在返回值的现象，因此，请仅在数组的结尾使用 `null`。

如果值无法以某个建议类型的形式返回，则会以“架构”面板中指定的类型返回。

示例

此范例将首先尝试以数值形式获取 XMLConnector 组件的 results 属性中某个字段 (productInfo.available) 的值, 如果失败, 则以字符串的形式获取。

```
import mx.data.binding.DataType;
import mx.data.binding.TypedValue;
var f:DataType = myXmlConnector.getField("results", "productInfo.available");
var b:TypedValue = f.getAnyTypedValue(["Number", "String"]);
```

另请参见

[ComponentMixins.getField\(\)](#)

DataType.getTypedValue()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

dataTypeObject.getTypedValue(requestedType)

参数

requestedType 包含数据类型的名称或 null 的字符串。

返回

一个 TypedValue 对象 (请参阅[第 135 页的 “TypedValue 类 \(仅限 Flash Professional\)”](#))

说明

方法 ; 以 *requestedType* 指定的形式返回字段的值 (如果已指定, 并且字段能够以该形式提供值)。如果字段无法以要求的形式提供值, 则方法将返回 null。

如果将 null 指定为 *requestedType*, 则方法将以默认类型返回字段的值。

示例

```
var bool:TypedValue = field.getTypedValue("Boolean");
```

DataType.kind

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

dataTypeObject.kind

说明

属性；提供对与此字段关联的 Kind 对象的引用。可以使用此属性来访问 Kind 对象的属性和方法。

DataType.setAnyTypedValue()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.setAnyTypedValue(newTypedValue)
```

参数

newValue 要在该字段中设置的 TypedValue 对象值。

有关 TypedValue 对象的详细信息，请参阅第 135 页的“TypedValue 类（仅限 Flash Professional）”。

返回

一个字符串数组，描述在尝试设置此新值时发生的任何错误。在以下条件下可能发生错误：

- 无法将提供的数据转换为此字段的数据类型（例如，无法将“abc”转换为数值）。
- 数据是可接受类型，但不满足该字段的验证条件。
- 该字段为只读。

注意：消息的实际文本将因在该字段的架构中定义的数据类型、格式程序和编码器而异。

说明

方法；将新值设置到字段中，并使用字段架构中的信息来处理字段。

此方法会通过首先调用 `DataType.setTypedValue()` 方法来设置值。如果失败，方法将检查以确定目标对象是否愿意接受字符串、布尔值或数值数据，如果答案肯定，则会尝试使用对应的动作脚本转换函数。

示例

此范例创建新的 TypedValue 对象（一个布尔值），然后将该值分配给名为 field 的一个 DataType 对象。发生的任何错误都被分配给 errors 数组。

```
import mx.data.binding.*;
var t:TypedValue = new TypedValue (true, "Boolean");
var errors:Array = field.setAnyTypedValue (t);
```

另请参见

[DataType.setTypedValue\(\)](#)

DataType.setAsBoolean()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.setAsBoolean(newBooleanValue)
```

参数

newBooleanValue 布尔值。

返回

无。

说明

方法；将字段设置为以布尔值形式提供的新值。该值被转换并存储为适合该字段的数据类型。

示例

```
var bool:Boolean = true;  
field.setAsBoolean (bool);
```

DataType.setAsNumber()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.setAsNumber(newNumberValue)
```

参数

newNumberValue 数值。

返回

无。

说明

方法；将字段设置为以数值形式提供的新值。该值被转换并存储为适合该字段的数据类型。

示例

```
var num:Number = 32;  
field.setAsNumber (num);
```

DataType.setAsString()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.setAsString(newStringValue)
```

参数

newStringValue 字符串。

返回

无。

说明

方法；将字段设置为以字符串形式提供的新值。该值被转换并存储为适合该字段的数据类型。

示例

```
var stringValue:String = "The new value";  
field.setAsString (stringValue);
```

DataType.setTypedValue()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataTypeObject.setTypedValue(newTypedValue)
```

参数

newValue 要在该字段中设置的 TypedValue 对象值。

有关 TypedValue 对象的详细信息，请参阅第 135 页的“TypedValue 类（仅限 Flash Professional）”。

返回

一个字符串数组，描述在尝试设置此新值时发生的任何错误。在以下条件下可能发生错误：

- 所提供数据的类型不可接受。
- 无法将提供的数据转换为此字段的数据类型（例如，无法将“abc”转换为数值）。
- 数据是可接受类型，但不满足该字段的验证条件。
- 该字段为只读。

注意：消息的实际文本将因在该字段的架构中定义的数据类型、格式程序和编码器而异。

说明

方法；将新值设置到字段中，并使用字段架构中的信息来处理字段。此方法的行为方式与 `DataType.setAnyTypedValue()` 类似，只是它不会同样努力地将数据转换为可接受的数据类型。有关详细信息，请参阅 `DataType.setAnyTypedValue()`。

示例

此范例创建新的 `TypedValue` 对象（一个布尔值），然后将该值分配给名为 `field` 的一个 `DataType` 对象。发生的任何错误都被分配给 `errors` 数组。

```
import mx.data.binding.*;
var bool:TypedValue = new TypedValue (true, "Boolean");
var errors:Array = field.setTypedValue (bool);
```

另请参见

[DataType.setTypedValue\(\)](#)

TypedValue 类（仅限 Flash Professional）

动作脚本类名称 `mx.data.binding.TypedValue`

`TypedValue` 是一个对象，它包含数据值以及有关该值的数据类型的信息。`TypedValue` 对象以 `DataType` 类各种方法的参数的形式提供，并且是从这些方法中返回的。`TypedValue` 对象中的数据类型信息帮助 `DataType` 对象确定何时需要执行数据类型转换，以及如何进行。

注意：要使此类在运行时可用，您必须在 FLA 文档中包括 `DataBindingClasses` 组件。有关详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

有关 `mx.data.binding` 包中的类的概述，请参阅第 108 页的“数据绑定类（仅限 Flash Professional）”。

TypedValue 类的属性摘要

属性	描述
TypedValue.type	包含与 <code>TypedValue</code> 对象的值关联的架构。
TypedValue.typeName	包含 <code>TypedValue</code> 对象的值的数据类型名称。
TypedValue.value	包含 <code>TypedValue</code> 对象的数据值。

TypedValue 类的构造函数

可用性

Flash Player 6 版本 79。

用法

```
new mx.data.binding.TypedValue(value, typeName, [type])
```

参数

- value* 数据值。此值可以是任何类型。
- typeName* 包含值的数据类型名称的字符串。
- type* （可选）更详细描述数据的架构的 `Schema` 对象。只有在某些情形下（诸如将数据设置为 `DataSet` 组件的 `dataProvider` 属性时），此字段才是必需的。

描述

构造函数；创建新的 TypedValue 对象。

TypedValue.type

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

typedValueObject.type

说明

属性；包含与 TypedValue 对象的值关联的架构。它仅用在某些情形中。

示例

此范例将在“输出”面板中显示“null”。

```
var t:TypedValue = new TypedValue (true, "Boolean", null);
trace(t.type);
```

TypedValue.typeName

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

typedValueObject.typeName

说明

属性；包含 TypedValue 对象的值的数据类型名称。

示例

此范例将在“输出”面板中显示“布尔值”。

```
var t:TypedValue = new TypedValue (true, "Boolean", null);
trace(t.typeName);
```

TypedValue.value

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
typedValueObject.value
```

说明

属性；包含 TypedValue 对象的数据值。

示例

```
此范例将在“输出”面板中显示“true”。

var t:TypedValue = new TypedValue (true, "Boolean", null);
trace(t.value);
```

DataGrid 组件（仅限 Flash Professional）

DataGrid 组件使您能够创建强大的数据驱动的显示和应用程序。可以使用 DataGrid 组件来实例化使用 Macromedia Flash Remoting 的记录集（从 ColdFusion、Java 或 .Net 中的数据库查询中检索），然后将其显示在列中。您也可以使用数据集或数组中的数据来填充 DataGrid 组件。第 2 版 DataGrid 组件已进行改进，包括了水平滚动、更好的事件支持（包括对可编辑单元格的事件支持）、增强的排序功能以及性能优化。

您可以调整和自定义诸如网格中的字体、颜色以及列边框等特征。对于网格中的任何列，您都可以使用自定义影片剪辑作为“单元格渲染器”。（单元格渲染器显示单元格的内容。）可以使用滚动条来浏览网格中的数据；也可以禁用滚动条并使用 DataGrid 方法来创建页面视图样式的显示。

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

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

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”。

与 DataGrid 组件进行交互（仅限 Flash Professional）

可以使用鼠标和键盘与 DataGrid 组件进行交互。

如果 DataGrid.sortableColumns 为 true 并且 DataGridColumn.sortOnHeaderRelease 为 true，在某个列标题内单击将会使网格根据列的单元格值进行排序。

如果 DataGrid.resizableColumns 为 true，在列之间的区域中单击将使您能够调整列的大小。

在某个可编辑单元格内单击会将焦点发送给该单元格；单击不可编辑的单元格不会影响焦点。如果某个单元格的 DataGrid.editable 和 DataGridColumn.editable 属性均为 true，则该单元格是可编辑的。

当 DataGrid 实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

按键	描述
向下箭头	如果正在编辑单元格，插入点将移到单元格文本的末尾。如果单元格不可编辑，则向下箭头处理选区的方式与 List 组件相同。
向上箭头	如果正在编辑单元格，插入点将移到单元格文本的开头。如果单元格不可编辑，则向上箭头处理选区的方式与 List 组件相同。
右箭头	如果正在编辑单元格，插入点将向右移动一个字符。如果单元格不可编辑，右箭头不会执行任何操作。

按键	描述
左箭头	如果正在编辑单元格，插入点将向左移动一个字符。如果单元格不可编辑，左箭头不会执行任何操作。
Return/Enter/ Shift+Enter 键	如果单元格可编辑，则会提交更改，并且插入点将移到同一列中单元格的下一行（向上或向下，视 shift 切换而定）。
Shift+Tab/Tab 键	将焦点移到前一项。当按下 Tab 键时，焦点将从网格中的最后一列回绕到下一行上的第一列。当按下 Shift+Tab 键时，回绕将反向进行。

使用 DataGrid 组件（仅限 Flash Professional）

可以使用 DataGrid 组件作为许多种数据驱动应用程序的基础。您不但可以轻松地显示数据库查询（或其他数据）的格式化表格视图，而且可以使用单元格渲染器功能建立更为复杂和可编辑的用户界面片段。以下是 DataGrid 组件的实际用途：

- Webmail 客户端
- 搜索结果页
- 电子表格应用程序，如借贷计算器和纳税申请表应用程序。

DataGrid 组件由两组 API 组成：DataGrid 类和 DataGridColumn 类。

了解 DataGrid 组件：数据模型和视图

从概念上来说，DataGrid 组件由数据模型和显示数据的视图组成。数据模型包含以下三个主要部分：

- DataProvider
用于填充数据网格的项目列表。同一帧中作为 DataGrid 组件的任何数组都会自动（从 DataProvider API 中）获得一些方法，这些方法允许您处理数据并将更改广播给多个视图。可以将实现 DataProvider 界面的任何对象分配给 `DataGrid.dataProvider` 属性（其中包括记录集、数据集等）。以下代码将创建名为 myDP 的数据提供程序：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
price:"Cheap"});
```

- 项
用于在列的单元格中存储信息单元的动作脚本对象。数据网格实际上是一个可以显示多列数据的列表。可以将列表看作一个数组；列表的每个索引空间就是一个项。对于 DataGrid 组件，每个项均由多个字段组成。在以下代码中，大括号 ({}) 之间的内容就是一个项：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
price:"Cheap"});
```

- 字段
指明各项内列名称的标识符。它与列列表中的 `columnNames` 属性相对应。在 List 组件中，字段通常为 label 和 data，但在 DataGrid 组件中，字段可以是任何标识符。在以下代码中，字段为 name 和 price：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
price:"Cheap"});
```

视图包含以下三个主要部分：

- 行
负责通过布置单元格来呈现网格项目的视图对象。每一行都水平布置在前一行的下方。

- 列

列由负责显示每一列的视图对象（DataGridColumn 类的实例，例如宽度、颜色、大小等）组成。

可以使用三种方式将列添加到数据网格中：将 DataProvider 对象分配给 `DataGrid.dataProvider`（此操作会自动为第一项中的每个字段生成一个列）；设置 `DataGrid.columnNames` 以指定将显示哪些字段；或者使用 DataGridColumn 类的构造函数来创建列，并调用 `DataGrid.addColumn()` 将它们添加到网格中。

要设置列的格式，请为整个数据网格设置样式属性，或者定义 DataGridColumn 对象，单独设置它们的样式格式，然后将它们添加到数据网格中。

- 单元格

负责呈现每一项的个别字段的视图对象。要与数据网格通信，这些组件必须实现 CellRenderer 接口（请参阅第 70 页的“CellRenderer API”）。对于基本数据网格，单元格是内置的动作脚本 TextField 对象。

DataGrid 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 DataGrid 组件实例设置的创作参数：

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

rowHeight 每行的高度（以像素为单位）。更改字体大小不会更改行高度。默认值为 20。

editable 一个布尔值，它指明网格是 (true) 否 (false) 可编辑。默认值为 false。

您可以编写动作脚本，以便使用其属性、方法和事件来控制 DataGrid 组件的这些和其他选项。有关详细信息，请参阅第 141 页的“DataGrid 类（仅限 Flash Professional）”。

创建具有 DataGrid 组件的应用程序

要创建具有 DataGrid 组件的应用程序，您必须首先确定数据的来源。网格的数据可以来源于使用 Flash Remoting 从 Macromedia ColdFusion、Java 或 .Net 内的数据库查询中馈入的记录集，也可以来源于数据集或数组。要将数据拉到网格中，您需要将 `DataGrid.dataProvider` 属性设置为记录集、数据集或数组。也可以使用 DataGrid 和 DataGridColumn 类的方法以本地方式创建数据。与 DataGrid 组件在同一帧中的任何 Array 对象均会复制 DataProvider 类的方法、属性和事件。

要使用 Flash Remoting 将 DataGrid 组件添加到应用程序：

- 1 在 Flash 中，选择“文件” > “新建”，然后选择“Flash 文档”。
- 2 在“组件”面板中，双击 DataGrid 组件以将其添加到舞台上。
- 3 在属性检查器中，输入实例名称 **myDataGrid**。
- 4 在“动作”面板中的第 1 帧上，输入以下代码：

```
myDataGrid.dataProvider = recordSetInstance;
```

Flash Remoting 记录集 recordSetInstance 即已分配给 myDataGrid 的 dataProvider 属性。

- 5 选择“控制” > “测试影片”。

要使用本地数据提供程序将 DataGrid 组件添加到应用程序：

- 1 在 Flash 中，选择 “文件” > “新建”，然后选择 “Flash 文档”。
- 2 在 “组件” 面板中，双击 DataGrid 组件以将其添加到舞台上。
- 3 在属性检查器中，输入实例名称 **myDataGrid**。
- 4 在 “动作” 面板中的第 1 帧上，输入以下代码：

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",
price:"Cheap"});
myDataGrid.dataProvider = myDP;
```

name 和 price 字段被用作列标题，它们的值将填充每一行中的单元格。
- 5 选择 “控制” > “测试影片”。

自定义 DataGrid 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平和垂直方向上将 DataGrid 组件变形。在创作时，在舞台上选择组件并使用 “任意变形” 工具或任何 “修改” > “变形” 命令。在运行时，使用 setSize() 方法（请参阅 [UIObject.setSize\(\)](#)）。如果没有水平滚动条，列宽度将按比例进行调整。如果调整了列大小（并因此调整了单元格大小），则单元格中的文本可能会被裁剪。

对 DataGrid 组件使用样式

您可以设置样式属性以更改 DataGrid 组件的外观。DataGrid 组件从 List 组件继承 “光晕” 样式。（有关更多信息，请参阅第 266 页的 “对 List 组件使用样式”。）DataGrid 组件还支持以下 “光晕” 样式：

样式	描述
backgroundColor	可以为整个网格或每一列设置背景色。
labelStyle	可以为整个网格或每一列设置字体样式。
headerStyle	可以将列标题的 CSS 样式声明应用到网格或列。
vGridLines	一个布尔值，指明是 (true) 否 (false) 显示垂直网格线。
hGridLines	一个布尔值，指明是 (true) 否 (false) 显示水平网格线。
vGridLineColor	垂直网格线的颜色。
hGridLineColor	水平网格线的颜色。
headerColor	列标题的颜色。

如果上表指明能够为列设置样式，则可以使用以下语法来设置样式：

```
grid.getColumnAt(3).setStyle("backgroundColor", 0xff00aa)
```

对 DataGrid 组件使用外观

DataGrid 组件用于表示其可视状态的外观包括在构成数据网格所依据的子组件内（ScrollPane 和 RectBorder）。有关这些子组件的外观的信息，请参阅第 424 页的 “对 ScrollPane 组件使用外观” 和第 267 页的 “在 List 组件中使用外观”。

但是，变换图像和选区覆盖图使用动作脚本绘图 API。要在创作时为数据网格的这些部分设置外观，请在其中一个主题 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/datagrid/ skins states 文件夹中外观元件中的动作脚本代码。有关详细信息，请参阅[第 33 页的“关于设置组件外观”](#)。

DataGrid 类（仅限 Flash Professional）

继承 mx.core.UIObject > mx.core.UIComponent > mx.core.View > mx.core.ScrollView > mx.controls.listclasses.ScrollSelectList > mx.controls.List

动作脚本类名称 mx.controls.DataGrid

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.DataGrid.version);

注意：下面的代码返回 undefined：trace(myDataGridInstance.version);。

DataGrid 类的方法摘要

方法	描述
DataGrid.addColumn()	将列添加到数据网格。
DataGrid.addColumnAt()	将列添加到数据网格的特定位置。
DataGrid.addItem()	将项目添加到数据网格。
DataGrid.addItemAt()	将项目添加到数据网格的特定位置。
DataGrid.editField()	替换位于指定位置的单元格数据。
DataGrid.getColumnAt()	获取对位于指定位置的列的引用。
DataGrid.getColumnIndex()	获取列的索引。
DataGrid.removeAllColumns()	从数据网格中删除所有列。
DataGrid.removeColumnAt()	从数据网格的指定位置删除列。
DataGrid.replaceItemAt()	使用另一项目替换位于指定位置的项目。
DataGrid.spaceColumnsEqually()	平均间隔所有列。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有属性。

DataGrid 类的属性摘要

属性	描述
DataGrid.columnCount	只读。显示列数。
DataGrid.columnNames	显示为列的每个项目内的字段名数组。
DataGrid.dataProvider	数据网格的数据模型。
DataGrid.editable	一个布尔值，指明数据网格是 (true) 否 (false) 可编辑。
DataGrid.focusedCell	定义具有焦点的单元格。

属性	描述
<code>DataGrid.headerHeight</code>	列标题的高度（以像素为单位）。
<code>DataGrid.hScrollPolicy</code>	指明是显示 ("on")、不显示 ("off") 还是在必要时显示 ("auto") 水平滚动条。
<code>DataGrid.resizableColumns</code>	一个布尔值，它指明列是 (true) 否 (false) 可以调整大小。
<code>DataGrid.selectable</code>	一个布尔值，它指明数据网格是 (true) 否 (false) 可选择。
<code>DataGrid.showHeaders</code>	一个布尔值，它指明列标题是 (true) 否 (false) 可见。
<code>DataGrid.sortableColumns</code>	一个布尔值，它指明列是 (true) 否 (false) 可排序。

DataGrid 类的事件摘要

事件	描述
<code>DataGrid.cellEdit</code>	在单元格值更改时广播。
<code>DataGrid.cellFocusIn</code>	在单元格获得焦点时广播。
<code>DataGrid.cellFocusOut</code>	在单元格失去焦点时广播。
<code>DataGrid.cellPress</code>	在单元格被按下时广播。
<code>DataGrid.change</code>	在选中项目时广播。
<code>DataGrid.columnStretch</code>	在用户调整列大小时广播。
<code>DataGrid.headerRelease</code>	在用户按下和松开标题时广播。

DataGrid.addColumn()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.addColumn(dataGridColumn)
myDataGrid.addColumn(name)
```

参数

dataGridColumn `DataGridColumn` 类的一个实例。
name 一个字符串，它指明要插入的新 `DataGridColumn` 对象的名称。

返回

对已添加的 `DataGridColumn` 对象的引用。

说明

方法；将新列添加到数据网格的结尾。有关详细信息，请参阅第 159 页的 “[DataGridColumn 类（仅限 Flash Professional）](#)”。

示例

以下代码将添加名为 Purple 的新 `DataGridColumn` 对象：

```
import mx.controls.gridclasses.DataGridColumn;
myGrid.addColumn(new DataGridColumn("Purple"));
```

`DataGrid.addColumnAt()`

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
myDataGrid.addColumnAt(index, name)
```

用法 2：

```
myDataGrid.addColumnAt(index, dataGridColumn)
```

参数

index 索引位置，`DataGridColumn` 对象将添加在该处。第一个位置是 0。

name 一个字符串，指明 `DataGridColumn` 对象的名称。必须指定 *index* 参数或 *dataGridColumn* 参数中的任意一个。

dataGridColumn `DataGridColumn` 类的一个实例。

返回

对已添加的 `DataGridColumn` 对象的引用。

描述

方法；在指定位置添加新列。列将向右移动，并且其索引值会增加。有关详细信息，请参阅[第 159 页的“DataGridColumn 类（仅限 Flash Professional）”](#)。

范例

以下范例将名为 "Green" 的新 `DataGridColumn` 对象添加到第二列和第四列：

```
import mx.controls.gridclasses.DataGridColumn;
myGrid.addColumnAt(1, "Green");
myGrid.addColumnAt(3, new DataGridColumn("Purple"));
```

`DataGrid.addItem()`

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.addItem(item)
```

参数

item 要添加到网格的对象的实例。

返回

指向已添加实例的引用。

说明

方法；将项目添加到网格的结尾（最后一个项目索引之后）。

注意：此方法与 `List.addItem()` 方法（其中传递的是对象而不是字符串）不同。

示例

以下范例将一个新对象添加到 `myGrid` 网格：

```
var anObject= {name:"Jim!!", age:30};  
var addedObject = myGrid.addItem(anObject);
```

DataGrid.addItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.addItemAt(index, item)
```

参数

index 应遵循的节点添加顺序（在多个子节点之间）。第一个位置是 0。

item 显示节点的字符串。

返回

指向已添加对象实例的引用。

说明

方法；将项目添加到网格的指定位置。

示例

以下范例将对象实例插入到网格的索引位置 4：

```
var anObject= {name:"Jim!!", age:30};  
var addedObject = myGrid.addItemAt(4, anObject);
```


DataGrid.cellEdit

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.cellEdit = function(eventObject){
    // 此处插入您的代码
}
myDataGridInstance.addEventListener("cellEdit", listenerObject)
```

描述

事件；在单元格值发生更改时广播到所有注册的侦听器。

V2 组件使用调度程序 / 侦听器事件模型。DataGrid 组件在单元格的值更改时发送 cellEdit 事件，而该事件由附加到您创建的侦听器对象 (listenerObject) 的函数（也称为处理函数）处理。您调用 addEventListener() 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (eventObject) 传递到处理函数。每个事件对象都有一组属性，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。DataGrid.cellEdit 事件的事件对象具有四个附加属性：

columnIndex 一个数字，指明目标列的索引。

itemIndex 一个数字，指明目标行的索引。

oldValue 单元格先前的值。

type 字符串 "cellEdit"。

有关详细信息，请参阅第 510 页的“事件对象”。

范例

在以下范例中，定义了名为 myDataGridListener 的处理函数并将其作为第二个参数传递给 myDataGrid.addEventListener() 方法。cellEdit 处理函数在 eventObject 参数中捕获事件对象。在广播 cellEdit 事件时，trace 语句被发送到“输出”面板，如下所示：

```
myDataGridListener = new Object();
myDataGridListener.cellEdit = function(event){
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The value of the cell at " + cell + " has changed");
}
myDataGrid.addEventListener("cellEdit", myDataGridListener);
```

注意：要使上述代码正常运行，网格必须可编辑。

DataGrid.cellFocusIn

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.cellFocusIn = function(eventObject){
    // 此处插入您的代码
}
myDataGridInstance.addEventListener("cellFocusIn", listenerObject)
```

描述

事件；在特定单元格获得焦点时向所有注册的侦听器广播。在广播了之前编辑的单元格的 `editCell` 和 `cellFocusOut` 事件后，将广播此事件。

V2 组件使用调度程序 / 侦听器事件模型。在 `DataGrid` 组件发送 `cellFocusIn` 事件时，该事件由附加到您创建的侦听器对象 (`listenerObject`) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象都有一组属性，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。`DataGrid.cellFocusIn` 事件的事件对象具有三个附加属性：

`columnIndex` 一个数字，指明目标列的索引。

`itemIndex` 一个数字，指明目标行的索引。

`type` 字符串 "cellFocusIn"。

有关详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myListener` 的处理函数并将其作为第二个参数传递给 `grid.addEventListener()` 方法。`cellFocusIn` 处理函数在 `eventObject` 参数中捕获事件对象。在广播 `cellFocusIn` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
var myListener = new Object();
myListener.cellFocusIn = function(event) {
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The cell at " + cell + " has gained focus");
};
grid.addEventListener("cellFocusIn", myListener);
```

注意：要使上述代码正常运行，网格必须可编辑。

DataGrid.cellFocusOut

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.cellFocusOut = function(eventObject){
    // 此处插入您的代码
}
```

```
}  
myDataGridInstance.addEventListener("cellFocusOut", listenerObject)
```

描述

事件；在用户移离具有焦点的单元格时向所有注册的侦听器广播。可以使用事件对象属性来隔离留下的单元格。在广播了 `cellEdit` 事件之后，并在下一个单元格广播任何后续的 `cellFocusIn` 事件之前，将广播此事件。

V2 组件使用调度程序 / 侦听器事件模型。在 `DataGrid` 组件发送 `cellFocusOut` 事件时，该事件由附加到您创建的侦听器对象的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。`DataGrid.cellFocusOut` 事件的事件对象具有三个附加属性：

`columnIndex` 一个数字，指明目标列的索引。第一个位置是 0。

`itemIndex` 一个数字，指明目标行的索引。第一个位置是 0。

`type` 字符串 "cellFocusOut"。

有关详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myListener` 的处理函数并将其作为第二个参数传递给 `grid.addEventListener()` 方法。`cellFocusOut` 处理函数在 *eventObject* 参数中捕获事件对象。在广播 `cellFocusOut` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
var myListener = new Object();  
myListener.cellFocusOut = function(event) {  
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";  
    trace("The cell at " + cell + " has lost focus");  
};  
grid.addEventListener("cellFocusOut", myListener);
```

注意：要使上述代码正常运行，网格必须可编辑。

DataGrid.cellPress

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.cellPress = function(eventObject){  
    // 此处插入您的代码  
}  
myDataGridInstance.addEventListener("cellPress", listenerObject)
```

描述

事件；当用户在单元格上按下鼠标按钮时向所有注册的侦听器广播。

V2 组件使用调度程序 / 侦听器事件模型。在 `DataGrid` 组件广播 `cellPress` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。`DataGrid.cellPress` 事件的事件对象具有三个附加属性：

`columnIndex` 一个数字，指明目标列的索引。第一个位置是 0。

`itemIndex` 一个数字，指明目标行的索引。第一个位置是 0。

`type` 字符串 "cellPress"。

有关详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myListener` 的处理函数并将其作为第二个参数传递给 `grid.addEventListener()` 方法。`cellPress` 处理函数在 *eventObject* 参数中捕获事件对象。在广播 `cellPress` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
var myListener = new Object();
myListener.cellPress = function(event) {
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The cell at " + cell + " has been clicked");
};
grid.addEventListener("cellPress", myListener);
```

DataGrid.change

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

描述

事件；在选中项目时向所有注册的侦听器广播。

V2 组件使用调度程序 / 侦听器事件模型。在 `DataGrid` 组件发送 `change` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。`DataGrid.change` 事件的事件对象具有一个附加属性 `type`，该属性的值为 `"change"`。有关详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myListener` 的处理函数并将其作为第二个参数传递给 `grid.addEventListener()` 方法。`change` 处理函数在 `eventObject` 参数中捕获事件对象。在广播 `change` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
var myListener = new Object();
myListener.change = function(event) {
    trace("The selection has changed to " + event.target.selectedIndex);
};
grid.addEventListener("change", myListener);
```

DataGrid.columnCount

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.columnCount
```

描述

属性（只读）；显示的列数。

范例

以下范例获取 `DataGrid` 实例 `grid` 中显示的列数：

```
var c = grid.columnCount;
```

DataGrid.columnNames

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.columnNames
```

描述

属性；显示为列的每个项目内的字段名数组。

范例

以下范例指示 `grid` 实例仅将三个字段显示为列：

```
grid.columnNames = ["Name", "Description", "Price"];
```

DataGrid.columnStretch

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.columnStretch = function(eventObject){
    // 此处插入您的代码
}
myDataGridInstance.addEventListener("columnStretch", listenerObject)
```

描述

事件；当用户在水平方向调整列的大小时向所有注册的侦听器广播。

V2 组件使用调度程序 / 侦听器事件模型。在 `DataGrid` 组件发送 `columnStretch` 事件时，该事件由附加到您创建的侦听器对象 (`listenerObject`) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象都有一组属性，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。`DataGrid.columnStretch` 事件的事件对象具有两个附加属性：

`columnIndex` 一个数字，指明目标列的索引。第一个位置是 0。

`type` 字符串 "columnStretch"。

有关详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myListener` 的处理函数并将其作为第二个参数传递给 `grid.addEventListener()` 方法。`columnStretch` 处理函数在 `eventObject` 参数中捕获事件对象。在广播 `columnStretch` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
var myListener = new Object();
myListener.columnStretch = function(event) {
    trace("column " + event.columnIndex + " was resized");
};
grid.addEventListener("columnStretch", myListener);
```

DataGrid.dataProvider

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.dataProvider
```

描述

属性；在 DataGrid 组件中查看的项目的数据模型。

数据网格将方法添加到 Array 类的原型，以使每个 Array 对象都符合 DataProvider 接口（请参阅 Classes/mx/controls/listclasses 文件夹中的 DataProvider.as）。任何在相同帧或屏幕中作为数据网格存在的数组都会自动具有成为数据网格的数据模型所需的所有方法（addItem()、getItemAt() 等等），并可用于向多个组件广播数据模型更改。

在 DataGrid 组件中，您需要指定字段以便在 DataGrid.columnNames 属性中显示。

如果在设置 DataGrid.dataProvider 属性之前未定义列集（通过设置 DataGrid.columnNames 属性或调用 DataGrid.addColumn() 方法），一旦到达数据提供程序的第一个项目，数据网格将为该项目中的每个字段生成列。

可以使用任何实现 DataProvider 接口的对象作为数据网格的数据提供程序（包括 Flash Remoting 记录集、数据集和数组）。

范例

以下范例将创建要用作数据提供程序的数组，并将其直接分配给 dataProvider 属性：

```
grid.dataProvider = [{name:"Chris", price:"Priceless"}, {name:"Nigel",  
    Price:"cheap"}];
```

以下范例将创建带有 DataProvider 类的新 Array 对象。它使用 for 循环将 20 个项目添加到网格。

```
myDP = new Array();  
for (var i=0; i<20; i++)  
    myDP.addItem({name:"Nivesh", price:"Priceless"});  
list.dataProvider = myDP
```

DataGrid.editable

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.editable
```

描述

属性；确定用户是 (true) 否 (false) 能够编辑数据网格。要使单独的列可编辑并且任何单元格都能获得焦点，此属性必须为 true。默认值为 false。

范例

以下范例将滚动位置设置为显示区的顶部：

```
myDataGrid.editable = true;
```

DataGrid.editField()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.editField(index, colName, data)
```

参数

index 目标单元格的索引。该数值是从零开始的。

colName 一个字符串，指明包含目标单元格的列（字段）的名称。

data 要存储在目标单元格中的值。此参数可以是任何数据类型。

返回

单元格中的数据。

说明

方法；替换位于指定位置的单元格数据。

示例

以下示例将一个值放在网格中：

```
var prevValue = myGrid.editField(5, "Name", "Neo");
```

DataGrid.focusedCell

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.focusedCell
```

描述

属性；（仅适用于可编辑模式）定义具有焦点的单元格的对象实例。该对象必须具有字段 `columnIndex` 和 `itemIndex`，这两个字段均为整数，指明单元格的列和项目的索引。原点为 (0,0)。默认值未定义。

范例

以下范例将具有焦点的单元格设置为第三列第四行：

```
grid.focusedCell = {columnIndex:2, itemIndex:3};
```

DataGrid.getColumnAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index)
```

参数

index 要返回的 DataGridColumn 对象的索引。该数值是从零开始的。

返回

一个 DataGridColumn 对象。

说明

方法；获取对位于指定索引的 DataGridColumn 对象的引用。

示例

以下范例获取位于索引 4 处的 DataGridColumn 对象：

```
var aColumn = myGrid.getColumnAt(4);
```

DataGrid.getColumnIndex()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnIndex(index)
```

参数

index 要返回的 DataGridColumn 对象的索引。

返回

一个 DataGridColumn 对象。

说明

方法；获取对位于指定索引的 DataGridColumn 对象的引用。

DataGrid.headerHeight

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.headerHeight
```

描述

属性；数据网格的标题栏的高度。默认值为 20。

范例

以下范例将滚动位置设置为显示区的顶部：

```
myDataGrid.headerHeight = 30;
```

DataGrid.headerRelease

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.headerRelease = function(eventObject){  
    // 此处插入您的代码  
}  
myDataGridInstance.addEventListener("headerRelease", listenerObject)
```

描述

事件；在松开列标题时向所有注册的侦听器广播。可以将此事件与

`DataGridColumn.sortOnHeaderRelease` 属性一起使用，以便防止自动排序并允许您按所需方式进行排序。

V2 组件使用调度程序 / 侦听器事件模型。在 `DataGrid` 组件发送 `headerRelease` 事件时，该事件由附加到您创建的侦听器对象 (`listenerObject`) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象都有一组属性，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。`DataGrid.headerRelease` 事件的事件对象具有两个附加属性：

`columnIndex` 一个数字，指明目标列的索引。

`type` 字符串 "headerRelease"。

有关详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myListener` 的处理函数并将其作为第二个参数传递给 `grid.addEventListener()` 方法。`headerRelease` 处理函数在 *eventObject* 参数中捕获事件对象。在广播 `headerRelease` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
var myListener = new Object();
myListener.headerRelease = function(event) {
    trace("column " + event.columnIndex + " header was pressed");
};
grid.addEventListener("headerRelease", myListener);
```

DataGrid.hScrollPolicy

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.hScrollPolicy
```

描述

属性；指定数据网格是否具有水平滚动条。此属性可以具有以下三个值之一：“on”、“off”和“auto”。默认值为“off”。

如果将 `hScrollPolicy` 设置为“off”，则会按比例缩放列以适应限定的宽度。

范例

以下范例将水平滚动策略设置为自动：

```
myDataGrid.hScrollPolicy = "auto";
```

DataGrid.removeAllColumns()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.removeAllColumns()
```

参数

无。

返回

无。

说明

方法；从数据网格中删除所有 `DataGridColumn` 对象。调用此方法不会影响数据提供程序。

示例

以下范例将从 `myDataGrid` 中删除所有 `DataGridColumn` 对象：

```
myDataGrid.removeAllColumns();
```

DataGrid.removeColumnAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.removeColumnAt(index)
```

参数

index 要删除的列的索引。

返回

指向已删除 `DataGridColumn` 对象的引用。

说明

方法；删除位于指定索引处的 `DataGridColumn` 对象。

示例

以下范例将删除 `myDataGrid` 中位于索引 2 处的 `DataGridColumn` 对象：

```
myDataGrid.removeColumnAt(2);
```

DataGrid.replaceItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.replaceItemAt(index, item)
```

参数

index 要替换的项目的索引。

item 充当要用作替换项的项目值的对象。

返回

先前的值。

说明

方法；替换位于指定索引处的项目。

示例

以下范例将位于索引 4 处的项目替换为 `aNewValue` 中定义的项目：

```
var aNewValue = {name:"Jim", value:"tired"};
var prevValue = myGrid.replaceItemAt(4, aNewValue);
```

DataGrid.resizableColumns

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.resizableColumns
```

描述

属性；一个布尔值，它确定查看器是 (`true`) 否 (`false`) 能够伸展网格的列。此属性必须为 `true` 才能调整单独列的大小。默认值为 `true`。

范例

以下范例防止用户调整列的大小：

```
myDataGrid.resizableColumns = false;
```

DataGrid.selectable

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.selectable
```

描述

属性；一个布尔值，它指明用户是 (`true`) 否 (`false`) 能够选择数据网格。默认值为 `true`。

范例

以下范例防止网格被选中：

```
myDataGrid.selectable = false;
```

DataGrid.showHeaders

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.showHeaders
```

描述

属性；一个布尔值，它指明数据网格是 (true) 否 (false) 显示列标题。列标题将被加上阴影，以区别于网格中的其他行。如果 `DataGrid.sortableColumns` 设置为 true，用户将能够单击列标题对列的内容进行排序。默认值为 true。

范例

以下范例将隐藏列标题：

```
myDataGrid.showHeaders = false;
```

另请参见

[DataGrid.sortableColumns](#)

DataGrid.sortableColumns

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.sortableColumns
```

描述

属性；一个布尔值，它确定在用户单击列标题时是 (true) 否 (false) 能够对数据网格的列进行排序。此属性必须为 true 才能对单独的列进行排序。要广播 `headerRelease` 事件，此属性必须设置为 true。默认值为 true。

范例

以下范例将禁用排序：

```
myDataGrid.sortableColumns = false;
```

另请参见

[DataGrid.headerRelease](#)

DataGrid.spaceColumnsEqually()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.spaceColumnsEqually()
```

参数

无。

返回

无。

说明

方法；平均地重新间隔列。

示例

以下范例将在按下并松开任何列标题时重新间隔 myGrid 的列：

```
myGrid.showHeaders = true
myGrid.dataProvider = [{guitar:"Flying V", name:"maggot"}, {guitar:"SG",
    name:"dreschie"}, {guitar:"jagstang", name:"vitapup"}];
gridLO = new Object();
gridLO.headerRelease = function(){
    myGrid.spaceColumnsEqually();
}
myGrid.addEventListener("headerRelease", gridLO);
```

DataGridColumn 类（仅限 Flash Professional）

动作脚本类名称 mx.controls.gridclassesDataGridColumn

可以创建并配置 DataGridColumn 对象以用作数据网格的列。DataGrid 类的许多方法都专用于管理 DataGridColumn 对象。DataGridColumn 对象存储在数据网格中一个从零开始的数组内；0 是最左边的列。在添加或创建了列后，您可以调用 DataGrid.getColumnAt(index) 来访问它们。

可以通过三种方法在网格中添加或创建列。如果要配置列，请最好在将数据添加到数据网格之前使用第二种或第三种方法，以便不必创建列两次。

- 如果将 DataProvider 或具有多个字段的项目添加到没有已配置 DataGridColumn 对象的网格，则会按 for..in 循环的反向顺序为每个字段自动生成列。
- DataGrid.columnNames 接受所需项目字段的字段名，并按顺序为列出的每个字段生成 DataGridColumn 对象。此方法使您能够利用最少量的配置快速选择列并对列排序。此方法将删除所有以前的列信息。
- 添加列的最灵活的方法是将它们预先建立为 DataGridColumn 对象，然后使用 DataGrid.addColumn() 将这些对象添加到数据网格。由于该方法使您能够在列无论何时到达网格之前使用适当的大小设置和格式设置添加列（这样减少了处理器需求），因此该方法特别有用。有关详细信息，请参阅第 160 页的“DataGridColumn 类的构造函数”。

DataGridColumn 类的属性摘要

属性	描述
<code>DataGridColumn.cellRenderer</code>	要用于在此列中显示单元格的元件的链接标识符。
<code>DataGridColumn.columnName</code>	只读。与列关联的字段名称。
<code>DataGridColumn.editable</code>	一个布尔值，它指明列是 (true) 否 (false) 可编辑。
<code>DataGridColumn.headerRenderer</code>	要用于显示此列的标题的类的名称。
<code>DataGridColumn.headerText</code>	此列的标题文本。
<code>DataGridColumn.labelFunction</code>	确定显示项目的哪个字段的函数。
<code>DataGridColumn.resizable</code>	一个布尔值，它指明列是 (true) 否 (false) 能够调整大小。
<code>DataGridColumn.sortable</code>	一个布尔值，它指明列是 (true) 否 (false) 可排序。
<code>DataGridColumn.sortOnHeaderRelease</code>	一个布尔值，它指明在用户按下列标题时是 (true) 否 (false) 对列进行排序。
<code>DataGridColumn.width</code>	列的宽度（以像素为单位）。

DataGridColumn 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
new DataGridColumn(name)
```

参数

name 一个字符串，指明 DataGridColumn 对象的名称。此参数是要显示的每个项目的字段。

返回

无。

描述

构造函数；创建一个 DataGridColumn 对象。使用此构造函数来创建要添加到 DataGrid 组件的列。在创建 DataGridColumn 对象后，您可以通过调用 `DataGrid.addColumn()` 将它们添加到数据网格。

范例

以下范例将创建名为 Location 的 DataGridColumn 对象：

```
import mx.controls.gridclasses.DataGridColumn;
var column = new DataGridColumn("Location");
```


DataGridColumn.cellRenderer

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).cellRenderer
```

描述

属性；要用于在此列中显示单元格的元件的链接标识符。用于此属性的任何类必须实现 CellRenderer 接口（请参阅[第 70 页的“CellRenderer API”](#)）。默认值未定义。

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：

```
myGrid.getColumnAt(3).cellRenderer = "MyCellRenderer";
```

DataGridColumn.columnName

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).columnName
```

描述

属性（只读）；与此列关联的字段名称。默认值是在 DataGridColumn 构造函数中调用的名称。

范例

以下范例将位于第三个索引位置的列的列名称分配给变量 name：

```
var name = myGrid.getColumnAt(3).columnName;
```

另请参见

[DataGridColumn 类的构造函数](#)

DataGridColumn.editable

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).editable
```

描述

属性；确定用户是 (true) 否 (false) 能编辑列。要使单独的列可编辑，`DataGrid.editable` 属性必须为 true（即使 `DataGridColumn.editable` 设置为 true）。默认值为 true。

范例

以下范例将使网格中的第一个列不可编辑：

```
myDataGrid.getColumnAt(0).editable = false;
```

另请参见

[DataGrid.editable](#)

DataGridColumn.headerRenderer

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).headerRenderer
```

描述

属性；一个字符串，它指明要用于显示此列的标题的类名称。用于此属性的任何类必须实现 `CellRenderer` 接口（请参阅第 70 页的“[CellRenderer API](#)”）。默认值未定义。

范例

以下范例使用链接标识符来设置一个新的标题渲染器：

```
myGrid.getColumnAt(3).headerRenderer = "MyHeaderRenderer";
```

DataGridColumn.headerText

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).headerText
```

描述

属性；列标题中的文本。默认值为列名称。

范例

以下范例将列标题文本设置为 “The Price”：

```
var myColumn = new DataGridColumn("price");
myColumn.headerText = "The Price";
```

DataGridColumn.labelFunction

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).labelFunction
```

描述

属性；指定用于确定显示每个项目的哪个字段（或字段组合）的函数。此函数会接收一个参数 *item*（该参数指示所呈现的项目），并且必须返回一个表示要显示的文本的字符串。此属性可用于创建在项目中没有相应字段的虚拟列。

范例

以下范例将创建一个虚拟列：

```
var myCol = myGrid.addColumn("Subtotal");
myCol.labelFunction = function(item) {
    return "$" + (item.price + (item.price * salesTax));
};
```

DataGridColumn.resizable

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).resizable
```

描述

属性；一个布尔值，它指明用户是 (true) 否 (false) 能调整列的大小。要使此属性生效，[DataGrid.resizableColumns](#) 属性必须设置为 true。默认值为 true。

范例

以下范例防止调整位于索引 1 处的列的大小：

```
myGrid.getColumnAt(1).resizable = false;
```

DataGridColumn.sortable

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).sortable
```

描述

属性；一个布尔值，它指明用户是 (true) 否 (false) 能对列进行排序。要使此属性生效，[DataGrid.sortableColumns](#) 属性必须设置为 true。默认值为 true。

范例

以下范例防止对位于索引 1 处的列进行排序：

```
myGrid.getColumnAt(1).sortable = false;
```

DataGridColumn.sortOnHeaderRelease

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).sortOnHeaderRelease
```

描述

属性；一个布尔值，它指明在用户单击标题时是 (true) 否 (false) 自动对列进行排序。只有在 [DataGridColumn.sortable](#) 设置为 true 时，此属性才能设置为 true。如果 [DataGridColumn.sortOnHeaderRelease](#) 设置为 false，您可以捕获 headerRelease 事件并执行自己的排序。

默认值为 true。

范例

以下范例使您能够捕获 headerRelease 事件以执行自己的排序：

```
myGrid.getColumnAt(7).sortOnHeaderRelease = false;
```

DataGridColumn.width

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDataGrid.getColumnAt(index).width
```

描述

属性；指明列宽度的数值（以像素为单位）。默认值为 50。

范例

以下范例使列的大小是默认值的一半：

```
myGrid.getColumnAt(4).width = 25;
```

DataHolder 组件（仅限 Flash Professional）

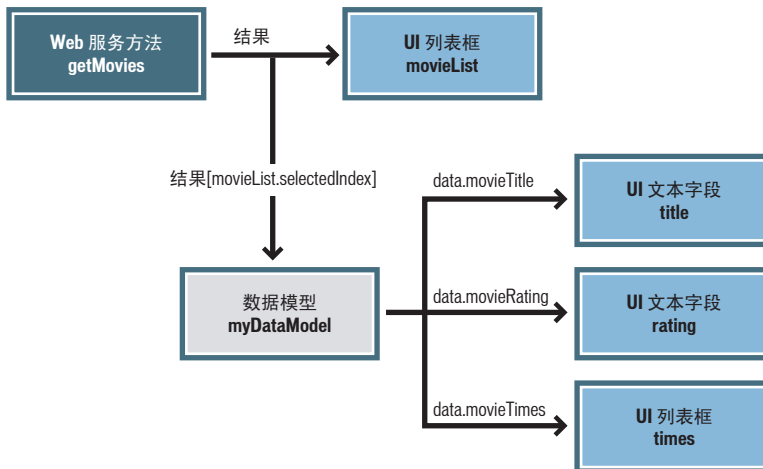
DataHolder 组件是数据的储备库，并且可用于在数据更改时生成事件。它的主要用途是容纳数据，并充当使用数据绑定的其他组件之间的连接器。

DataHolder 组件最初只有一个名为 data 的可绑定属性。您可以使用“组件检查器”面板（“窗口” > “开发面板” > “组件检查器”）中的“架构”选项卡添加更多属性。有关使用“架构”选项卡的详细信息，请参阅“使用 Flash”帮助中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

您可以将任何类型的数据分配给 DataHolder 属性，方法是在数据和另一属性之间创建绑定，或者使用自己的动作脚本代码。当该数据的值更改时，DataHolder 组件将发出一个名称与属性相同的事件，并执行与该属性关联的任何绑定。

在无法直接将组件（如连接器、用户界面组件或 DataSet 组件）绑定在一起时，DataHolder 组件十分有用。下面是您可以使用 DataHolder 组件的一些情况：

- 如果数据值是由动作脚本生成的，则可能需要将其绑定到某些其他组件。在这种情况下，您可以使用包含按需要绑定的属性的 DataHolder 组件。在将新值分配给这些属性时（例如，通过动作脚本），这些值将被分发到绑定了数据的对象。
- 您可能从索引复杂的数据绑定中产生的数据值，如下图中所示。



在这种情况下，将数据值绑定到某个 DataHolder 组件（在上图中称为 DataModel）然后使用该组件绑定到用户界面较为方便。

创建具有 DataHolder 组件的应用程序（仅限 Flash Professional only）

在此范例中，您需要将数组属性添加到 DataHolder 组件的架构（一个数组）中，该架构的值由您编写的动作脚本代码确定。然后，您需要将该数组属性绑定到 DataGrid 组件的 `dataProvider` 属性，方法是使用“组件检查器”面板中的“绑定”选项卡。

要在简单应用程序中使用 DataHolder 组件：

- 1 在 Flash MX Professional 2004 中，创建一个新文件。
- 2 打开“组件”面板（“窗口” > “开发面板” > “组件”），将 DataHolder 组件拖到舞台上，然后将其命名为 **dataHolder**。
- 3 将 DataGrid 组件拖到舞台上，然后将其命名为 **namesGrid**。
- 4 选择 DataHolder 组件，然后打开“组件检查器”面板（“窗口” > “开发面板” > “组件检查器”）。
- 5 在“组件检查器”面板中，单击“架构”选项卡。
- 6 单击位于“架构”选项卡顶部窗格中的“添加组件属性” (+) 按钮。
- 7 在“架构”选项卡底部窗格的“字段名称”字段中键入 **namesArray**，然后从“数据类型”弹出菜单中选择“数组”。
- 8 在“组件检查器”面板中单击“绑定”选项卡，然后在 DataHolder 组件的 `namesArray` 属性和 DataGrid 组件的 `dataProvider` 属性之间添加绑定。
有关使用“绑定”选项卡创建绑定的详细信息，请参阅“使用 Flash”帮助中的“在“绑定”选项卡中处理绑定（仅限 Flash Professional）”。
- 9 在“时间轴”（“窗口” > “时间轴”）中，选择“第 1 层”上的第一个帧，然后打开“动作”面板（“窗口” > “开发面板” > “动作”）。
- 10 在“动作”面板中输入以下代码：

```
dataHolder.namesArray= [{name:"Tim"},{name:"Paul"},{name:"Jason"}];
```

此代码使用若干对象填充 `namesArray` 数组。当此变量赋值执行时，将会执行您之前在 DataHolder 组件和 DataGrid 组件之间建立的绑定。

- 11 通过选择“控制” > “测试影片”对文件进行测试。

DataHolder 类的属性摘要

属性	描述
<code>DataHolder.data</code>	DataHolder 组件的默认可绑定属性。

DataHolder.data

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
dataHolder.data
```

说明

属性；DataHolder 对象架构中的默认项目。此属性不是 DataHolder 组件的“永久”成员，而是组件每个实例的默认可绑定属性。通过使用“组件检查器”面板的“架构”选项卡，您可以添加自己的可绑定属性，也可以删除默认的 data 属性。

有关使用“架构”选项卡的详细信息，请参阅“使用 Flash”帮助中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

示例

有关使用此组件的范例，请参阅第 166 页的“创建具有 DataHolder 组件的应用程序（仅限 Flash Professionalonly）”。

DataProvider API

动作脚本类名称 mx.controls.listclasses.DataProvider

DataProvider API 是一组方法和属性，数据源需要这些方法和属性才能使基于 List 的类与其通信。数组、记录集和数据集均实现此 API。您可以通过实现本文中描述的所有方法来创建符合 DataProvider 的类。然后，基于列表的组件将能够使用该类作为数据提供程序。

DataProvider API 的方法使您能够在显示数据的任何组件（也称为视图）中查询和修改数据。DataProvider API 还会在数据更改时广播更改事件。多个视图可以使用同一数据提供程序并都接收更改事件。

数据提供程序是项目的线性集合（类似于数组）。每个项目都是一个由多个数据字段组成的对象。使用 DataProvider.getItemAt()，您可以通过项目索引来访问这些项目（就好像处理数组一样）。

使用数据提供程序的最常见情况是与数组一起使用。当 Array 对象与支持数据的组件处于同一帧或屏幕中时，支持数据的组件会将 DataProvider API 的所有方法应用到 Array.prototype。这样，您将可以使用任何现有数组作为具有 dataProvider 属性的视图的数据。

由于 DataProvider API 的存在，提供数据视图的第 2 版组件（DataGrid、List、Tree 等）还可以显示 Flash Remoting 记录集以及 DataSet 组件中的数据。DataProvider API 是支持数据的组件用于与其数据提供程序通信的语言。

在 Macromedia Flash 文档中，“DataProvider”是 API 的名称，dataProvider 是充当数据视图的每个组件的属性，而“数据提供程序”是表示数据源的一般术语。

DataProvider API 的方法

名称	描述
DataProvider.addItem()	将项目添加到数据提供程序的结尾。
DataProvider.addItemAt()	将项目添加到数据提供程序的指定位置。
DataProvider.editField()	更改数据提供程序的某个字段。
DataProvider.getEditingData()	从数据提供程序中获取用于编辑的数据。
DataProvider.getItemAt()	获取对位于指定位置的项目的引用。
DataProvider.getItemID()	返回项目的唯一 ID。
DataProvider.removeAll()	从数据提供程序中删除所有项目。
DataProvider.removeItemAt()	从数据提供程序的指定位置删除项目。

名称	描述
<code>DataProvider.replaceItemAt()</code>	用其他项目替换位于指定位置的项目。
<code>DataProvider.sortItems()</code>	对数据提供程序中的项目进行排序。
<code>DataProvider.sortItemsBy()</code>	依据指定比较函数对数据提供程序中的项目进行排序。

DataProvider API 的属性

名称	描述
<code>DataProvider.length</code>	数据提供程序中的项目数。

DataProvider API 的事件

名称	描述
<code>DataProvider.modelChanged</code>	在数据提供程序更改时广播。

DataProvider.addItem()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDP.addItem(item)`

参数

item 包含数据的对象。它包含数据提供程序中的项目。

返回

无。

说明

方法；在数据提供程序的结尾添加新项目。

此方法触发事件名称为 `addItem` 的 `modelChanged` 事件。

示例

以下范例将一个新项目添加到数据提供程序 `myDP` 的结尾：

```
myDP.addItem({ label:"this is an Item"});
```


DataProvider.addItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDP.addItemAt(index, item)
```

参数

index 一个等于或大于 0 的数字。插入项目的位置；新项目的索引。

item 包含项目数据的对象。

返回

无。

说明

方法；将新项目添加到数据提供程序的指定索引处。大于数据提供程序长度的索引将被忽略。

此方法触发事件名称为 `addItem` 的 `modelChanged` 事件。

示例

以下范例将一个新项目添加到数据提供程序 `myDP` 的第 4 个位置：

```
myDP.addItemAt(3, {label : "this is the fourth Item"});
```

DataProvider.editField()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDP.editField(index, fieldName, newData)
```

参数

index 一个大于或等于 0 的数字。项目的索引。

fieldName 一个字符串，指明要修改的项目中字段的名称。

newData 要放入数据提供程序的新数据。

返回

无。

说明

方法；更改数据提供程序的某个字段。

此方法触发事件名称为 `updateField` 的 `modelChanged` 事件。

示例

以下代码修改第三个项目的 `label` 字段：

```
myDP.editField(2, "label", "mynewData");
```

DataProvider.getEditingData()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDP.getEditingData(index, fieldName)
```

参数

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

fieldName 一个字符串，指明所编辑字段的名称。

返回

要使用的可编辑格式数据。

说明

方法；从数据提供程序中检索要编辑的数据。这样，数据模型将能够提供不同格式的数据进行编辑和显示。

示例

以下代码将获取 `price` 字段的可编辑字符串：

```
trace(myDP.getEditingData(4, "price");
```

DataProvider.getItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDP.getItemAt(index)
```

参数

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

返回

对检索出的项目的引用；如果索引超出范围，则为未定义。

说明

方法；检索对位于指定位置的项目的引用。

示例

以下代码将显示第 5 个项目的标签：

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

DataProvider.getItemID()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004 Professional。

用法

```
myDP.getItemID(index)
```

参数

index 一个大于或等于 0 的数字。

返回

一个数字，充当项目的唯一 ID。

描述

方法；返回项目的唯一 ID。此方法主要用于跟踪选择。此 ID 用于支持数据的组件中，对哪些项目的列表处于选定状态进行记录。

范例

此范例获取第 4 个项目的 ID：

```
var ID = myDP.getItemID(3);
```

DataProvider.modelChanged

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.modelChanged = function(eventObject){  
    // 此处插入您的代码  
}  
myMenu.addEventListener("modelChanged", listenerObject
```

描述

事件；在数据提供程序被修改时向所有其视图侦听器广播。侦听器通常是通过分配其 `dataProvider` 属性添加到模型的。

V2 组件使用调度程序 / 侦听器事件模型。当数据提供程序在某些方面发生更改时，它将会广播 `modelChanged` 事件，并且支持数据的组件将捕获该事件来更新显示，以便反映数据的更改。

`Menu.modelChanged` 事件的事件对象具有五个附加属性：

- `eventName` `eventName` 属性用于划分 `modelChanged` 事件的子类别。支持数据的组件使用此信息来避免完全刷新使用数据提供程序的组件实例（视图）。下面是 `eventName` 属性的支持值：
 - `updateAll` 整个视图都需要刷新（除滚动位置外）。
 - `addItem` 已添加一系列项目。
 - `removeItems` 已删除一系列项目。
 - `updateItems` 一系列项目需要刷新。
 - `sort` 数据已排序。
 - `updateField` 项目内的某个字段已更改，并且需要刷新。
 - `updateColumn` `dataProvider` 内的整个字段定义需要刷新。
 - `filterModel` 已筛选模型，并且视图需要刷新（重置 `scrollPosition`）。
 - `schemaLoaded` 已声明 `dataProvider` 的字段定义。
- `firstItem` 第一个受影响项目的索引。
- `lastItem` 最后一个受影响项目的索引。如果只有一个项目受影响，则该值与 `firstItem` 相等。
- `removedIDs` 已删除项目标识符的数组。
- `fieldName` 一个字符串，指明受影响字段的名称。

有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `listener` 的处理函数并将其作为第二个参数传递给 `addEventListener()` 方法。`modelChanged` 处理函数在 `evt` 参数捕获事件对象。在广播 `modelChanged` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
listener = new Object();
listener.modelChanged = function(evt){
    trace(evt.eventName);
}
myList.addEventListener("modelChanged", listener);
```

DataProvider.length

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDP.length`

描述

属性（只读）；数据提供程序中的项目数。

范例

此范例将 `myArray` 数据提供程序中的项目数发送到“输出”面板：
`trace(myArray.length);`

DataProvider.removeAll()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDP.removeAll()`

参数

无。

返回

无。

描述

方法；删除数据提供程序中的所有项目。

此方法触发事件名称为 `removeItems` 的 `modelChanged` 事件。

范例

此范例将删除数据提供程序中的所有项目：
`myDP.removeAll();`

DataProvider.removeItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDP.removeItemAt(index)`

参数

`index` 一个大于或等于 0 的数字。要删除的项目的索引。

返回

无。

描述

方法；删除位于指定索引处的项目。已删除索引后面的索引将依次减 1。

此方法触发事件名称为 `removeItems` 的 `modelChanged` 事件。

范例

此范例将删除位于第 4 个位置的项目：

```
myDP.removeItemAt(3);
```

DataProvider.replaceItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDP.replaceItemAt(index, item)
```

参数

index 一个大于或等于 0 的数字。要更改的项目的索引。

item 充当新项目的对象。

返回

无。

描述

方法；替换位于指定索引处的项目的内容。

此方法触发事件名称为 `removeItems` 的 `modelChanged` 事件。

范例

此范例将使用具有 “new label” 标签的项目替换位于索引 3 处的项目：

```
myDP.replaceItemAt(3, {label : "new label"});
```

DataProvider.sortItems()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myDP.sortItems([compareFunc], [optionsFlag])
```

参数

compareFunc 一个指向函数的引用，该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()`。此参数是可选的。

optionsFlag 允许您不必复制整个数组或反复对其重新排序，即可对单个数组执行不同类型的多次排序。此参数是可选的。

以下是 *optionsFlag* 的可能值：

- `Array.DESENDING` - 按从高到低的顺序排序。
- `Array.CASEINSENSITIVE` - 按不区分大小写的方式排序。
- `Array.NUMERIC` - 如果所比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（字符串比较可以不区分大小写，只要指定该标记即可）。
- `Array.UNIQUESORT` - 如果数组中有两个对象相同或具有相同的排序字段，则此方法返回错误代码 (0)，而不是排序后的数组。
- `Array.RETURNINDEXEDARRAY` - 返回作为排序结果的整数索引数组。例如，如果用包含 `Array.RETURNINDEXEDARRAY` 值的 *optionsFlag* 参数排序，则以下数组将返回代码的第二行，并且该数组将保持不变：

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 3 和 1：

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

返回

无。

描述

方法；依据 *compareFunc* 参数指定的比较函数或 *optionsFlag* 参数指定的一个或多个排序选项对数据提供程序中的项目进行排序。

此方法触发事件名称为 `sort` 的 `modelChanged` 事件。

范例

此范例依据大写标签进行排序。项目 `a` 和 `b` 被传递到函数，并包含 `label` 和 `data` 字段：

```
myList.sortItems(upperCaseFunc);  
function upperCaseFunc(a,b){  
    return a.label.toUpperCase() > b.label.toUpperCase();  
}
```

DataProvider.sortItemsBy()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myDP.sortItemsBy(fieldName, order, [optionsFlag])
```

参数

fieldName 一个字符串，指定要用于排序的字段名称。通常情况下，此值为 "label" 或 "data"。

order 一个字符串，指定是以升序 ("ASC") 还是以降序 ("DESC") 对项目进行排序。

optionsFlag 允许您不必复制整个数组或反复对其重新排序，即可对单个数组执行不同类型的多次排序。此参数是可选的。

以下是 *optionsFlag* 的可能值：

- `Array.DECENDING` - 按从高到低的顺序排序。
- `Array.CASEINSENSITIVE` - 按不区分大小写的方式排序。
- `Array.NUMERIC` - 如果所比较的两个元素是数字，则按数字顺序排序。如果它们不是数字，则使用字符串比较（字符串比较可以不区分大小写，只要指定该标记即可）。
- `Array.UNIQUESORT` - 如果数组中有两个对象相同或具有相同的排序字段，则此方法返回错误代码 (0)，而不是排序后的数组。

• `Array.RETURNINDEXEDARRAY` - 返回作为排序结果的整数索引数组。例如，如果用包含 `Array.RETURNINDEXEDARRAY` 值的 *optionsFlag* 参数排序，则以下数组将返回代码的第二行，并且该数组将保持不变：

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

您可以将这些选项合并成一个值。例如，以下代码将合并选项 3 和 1：

```
array.sort (Array.NUMERIC | Array.DECENDING)
```

返回

无。

描述

方法；按字母或数值顺序、按指定的顺序、使用指定的字段名对数据提供程序中的项目进行排序。如果 *fieldName* 项目既包括文本字符串也包括整数，则首先列出整数项目。*fieldName* 参数通常为 "label" 或 "data"，但高级编程人员可能会指定任何原始值。您可以根据需要使用 *optionsFlag* 参数来指定排序样式。

此方法触发事件名称为 `sort` 的 `modelChanged` 事件。

范例

下列代码使用列表项的标签按升序对列表中的项目进行排序：

```
myDP.sortItemsBy("label", "ASC");
```

DataSet 组件（仅限 Flash Professional）

`DataSet` 组件使您能够将数据处理为可进行索引、排序、搜索、筛选和修改的对象的集合。

`DataSet` 组件功能包括 `DataSetIterator`（一组用于遍历和处理数据集合的方法）和 `DeltaPacket`（一组用于处理数据集合更新的接口和类）。大多数情况下，您不会直接使用这些类和接口；您将通过 `DataSet` 类提供的方法间接使用它们。

DataSet 组件管理的项目也称为传送对象。传送对象使用公共属性或存取器方法（用于读取和写入数据）显示驻留在服务器上的业务数据。DataSet 组件使开发人员能够处理复杂的客户端对象（反映其服务器端对应对象）或最简单的匿名对象集合（具有表示数据记录内的字段的公共属性）。有关传送对象的详细信息，请参阅 Core J2EE Patterns Transfer Object（核心 J2EE 模式传送对象），网址为：java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html。

注意：DataSet 组件需要 Flash Player 7 或更高版本。

使用 DataSet 组件（仅限 Flash Professional）

通常情况下，您在应用程序中将 DataSet 组件与其他组件结合使用以处理和更新数据源：Connector 组件（用于连接到外部数据源）、用户界面组件（用于显示数据源中的数据）以及 Resolver 组件（用于将对数据集进行的更新转换为适当的格式，以便发送到外部数据源）。然后，您可以使用数据绑定将这些不同组件的属性绑定在一起。

有关 DataSet 组件以及如何将它与其他组件一起使用的一般详细信息，请参阅“使用 Flash”帮助中的“数据管理（仅限 Flash Professional）”。

DataSet 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 DataSet 组件实例设置的创作参数：

itemClassName 在每次需要新项目时将实例化的传送对象类的名称。

注意：要使指定的类在运行时可用，您必须同时在 SWF 文件的代码内的某处对该类进行完全限定引用。（例如，`var myItem:my.package.myItem;`）。

filtered 如果为 `true`，则会对数据集应用过滤器，以使其仅包含匹配过滤条件的对象。

logChanges 如果为 `true`，数据集会将所有变化（对数据或方法调用进行的更改）记录到它的 `deltaPacket` 属性。

readOnly 如果为 `true`，则无法修改数据集。

您可以编写动作脚本，以便使用 DataSet 组件的属性、方法和事件来控制该组件的这些和其他选项。有关详细信息，请参阅第 179 页的“DataSet 类（仅限 Flash Professional）”。

创建具有 DataSet 组件的应用程序

通常情况下，您可以将 DataSet 组件与其他用户界面组件一起使用，并通常与 Connector 组件（如 XMLConnector 或 WebServiceConnector 组件）一起使用。数据集中的项目是借助于 Connector 组件或原始动作脚本数据填充的，并随后绑定到用户界面控件（如 List 或 DataGrid 组件）。

要使用 DataSet 组件创建应用程序：

- 1 在 Flash MX Professional 2004 中，选择“文件”>“新建”。在“类型”列中选择“Flash 文档”，然后单击“确定”。
- 2 如果“组件”面板尚未打开，请将其打开（“窗口”>“开发面板”>“组件”）。
- 3 将 DataSet 组件从“组件”面板拖到舞台上。在属性检查器中将其命名为 **userData**。
- 4 将 DataGrid 组件拖到舞台上，然后将其命名为 **userGrid**。
- 5 将 DataGrid 组件的大小调整为约 300 像素宽，100 像素高。
- 6 将 Button 组件拖到舞台上，然后将其命名为 **nextBtn**。

7 在“时间轴”中选择“第 1 层”上的第一帧，然后打开“动作”面板（“窗口” > “开发面板” > “动作”）。

8 将以下代码添加到“动作”面板：

```
var recData = [{id:0, firstName:"Mick", lastName:"Jones"},
               {id:1, firstName:"Joe", lastName:"Strummer"},
               {id:2, firstName:"Paul", lastName:"Simonon"}];
userData.items = recData;
```

这将使用一个对象数组填充 DataSet 对象的 items 属性，其中每个对象具有三个属性：firstName、lastName 和 id。

9 要将 DataSet 组件的内容绑定到 DataGrid 组件的内容，请打开“组件检查器”面板（“窗口” > “开发面板” > “组件检查器”），然后单击“绑定”选项卡。

10 在舞台上选择 DataGrid 组件 (userGrid)，然后单击“组件检查器”面板中的“添加绑定” (+) 按钮。

11 在“添加绑定”对话框中，选择“dataProvider : 数组”，然后单击“确定”。

12 在“组件检查器”面板中双击“绑定到”字段。

13 在出现的“绑定到”对话框中，从“组件路径”列中选择“DataSet <userData>”，然后从“架构位置”列中选择“dataProvider : 数组”。

14 要将 DataSet 组件的选定索引绑定到 DataGrid 组件的选定索引，请在“组件检查器”面板中再次单击“添加绑定” (+) 按钮。

15 在出现的对话框中，选择“selectedIndex : 数值”。单击“确定”。

16 在“组件检查器”面板中双击“绑定到”字段，打开“绑定到”对话框。

17 在“组件路径”字段中，从“组件路径”列中选择“DataSet <userData>”，然后从“架构位置”列中选择“selectedIndex : 数值”。

18 选择 Button 组件 (nextBtn)，如果“动作”面板尚未打开，请将其打开（“窗口” > “开发面板” > “动作”）。

19 在“动作”面板中输入以下代码：

```
on(click){
    _parent.userData.next();
}
```

此代码使用 DataSet.next() 方法浏览到 DataSet 对象项目集合中的下一个项目。由于之前已将 DataGrid 对象的 selectedIndex 属性绑定到 DataSet 对象的同一属性，因此，如果更改 DataSet 对象中的当前项目，DataGrid 对象中的当前（选定）项目也会随之更改。

20 保存文件，然后选择“控制” > “测试影片”来测试 SWF 文件。

DataGrid 对象即会被指定项目填充。注意单击按钮将如何更改 DataGrid 对象中的选定项目。

DataSet 类（仅限 Flash Professional）

动作脚本类名称 `mx.data.components.DataSet`

DataSet 类的方法摘要

方法	描述
<code>DataSet.addItem()</code>	将指定项目添加到集合。
<code>DataSet.addSort()</code>	创建集合中项目的新的已排序视图。
<code>DataSet.applyUpdates()</code>	通知侦听器对 DataSet 对象进行的更改已就绪。
<code>DataSet.changesPending()</code>	指明 DeltaPacket 对象中是否存在项目。
<code>DataSet.clear()</code>	从集合的当前视图中清除所有项目。
<code>DataSet.createItem()</code>	返回最新初始化的集合项目。
<code>DataSet.disableEvents()</code>	停止向侦听器发送 DataSet 事件。
<code>DataSet.enableEvents()</code>	继续向侦听器发送 DataSet 事件。
<code>DataSet.find()</code>	在集合的当前视图中定位项目。
<code>DataSet.findFirst()</code>	在集合的当前视图中定位出现的第一个项目。
<code>DataSet.findLast()</code>	在集合的当前视图中定位出现的最后一个项目。
<code>DataSet.first()</code>	移至集合当前视图中的第一个项目。
<code>DataSet.getItemId()</code>	返回指定项目的唯一 ID。
<code>DataSet.getIterator()</code>	返回当前重复值的克隆。
<code>DataSet.hasNext()</code>	指明当前的重复值是否位于集合视图的结尾。
<code>DataSet.hasPrevious()</code>	指明当前的重复值是否位于集合视图的开头。
<code>DataSet.hasSort()</code>	指明是否存在指定的排序。
<code>DataSet.isEmpty()</code>	指明集合是否包含任何项目。
<code>DataSet.last()</code>	移至集合当前视图中的最后一个项目。
<code>DataSet.loadFromSharedObj()</code>	从共享对象中检索 DataSet 对象的内容。
<code>DataSet.locateById()</code>	将当前重复值移到具有指定 ID 的项目。
<code>DataSet.next()</code>	移至集合当前视图中的下一个项目。
<code>DataSet.previous()</code>	移至集合当前视图中的上一个项目。
<code>DataSet.removeAll()</code>	从集合中删除所有项目。
<code>DataSet.removeItem()</code>	从集合中删除指定项目。
<code>DataSet.removeRange()</code>	删除当前重复值的范围设置。
<code>DataSet.removeSort()</code>	从 DataSet 对象中删除指定排序。
<code>DataSet.saveToSharedObj()</code>	将 DataSet 对象中的数据保存到共享对象。
<code>DataSet.setIterator()</code>	设置 DataSet 对象的当前重复值。

方法	描述
<code>DataSet.setRange()</code>	设置当前重复值的范围设置。
<code>DataSet.skip()</code>	在集合的当前视图中按指定项目数向前或向后移动。
<code>DataSet.useSort()</code>	使指定排序处于活动状态。

DataSet 类的属性摘要

属性	描述
<code>DataSet.currentItem</code>	返回集合中的当前项目。
<code>DataSet.dataProvider</code>	返回 <code>DataProvider</code> 接口。
<code>DataSet.deltaPacket</code>	返回对集合所做的更改，或分配要对集合进行的更改。
<code>DataSet.filtered</code>	指明是否已筛选项目。
<code>DataSet.filterFunc</code>	用于在集合中筛选项目的用户定义函数。
<code>DataSet.items</code>	集合中的项目。
<code>DataSet.itemClassName</code>	要在分配项目时创建的对象。
<code>DataSet.length</code>	指定集合当前视图中的项目数。
<code>DataSet.logChanges</code>	指明是否记录对集合或其项目所做的更改。
<code>DataSet.properties</code>	包含此集合内任何传送对象的属性（字段）。
<code>DataSet.readOnly</code>	指明是否能修改集合。
<code>DataSet.schema</code>	以 XML 格式指定集合的架构。
<code>DataSet.selectedIndex</code>	包含集合内当前项目的索引。

DataSet 类的事件摘要

事件	描述
<code>DataSet.addItem</code>	在将项目添加到集合之前广播。
<code>DataSet.afterLoaded</code>	在分配了 <code>items</code> 属性之后广播。
<code>DataSet.deltaPacketChanged</code>	在 <code>DataSet</code> 对象的变量增量包已更改并且可以使用时广播。
<code>DataSet.calcFields</code>	在应更新计算所得字段时广播。
<code>DataSet.iteratorScrolled</code>	在重复值的位置更改时广播。
<code>DataSet.modelChanged</code>	在集合中项目的某些方面已被修改时广播。
<code>DataSet.newItem</code>	在 <code>DataSet</code> 对象构建了新项目时（但在将其添加到集合之前）广播。
<code>DataSet.removeItem</code>	在删除了项目时广播。
<code>DataSet.resolveDelta</code>	在将 <code>DeltaPacket</code> 对象分配给包含消息的 <code>DataSet</code> 对象时广播。

DataSet.addItem

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(addItem) {  
    // insert your code here  
}  
listenerObject = new Object();  
listenerObject.addItem = function (eventObj) {  
    // insert your code here  
}  
dataSet.addEventListener("addItem", listenerObject)
```

说明

事件；就在将新的传送对象插入此集合之前生成。

如果将事件对象的 `result` 属性设置为 `false`，则添加操作将被取消；如果将其设置为 `true`，则允许添加操作。

事件对象 (*eventObj*) 包含以下属性：

`target` 生成该事件的 `DataSet` 对象。

`type` 字符串 "addItem"。

`item` 指向集合中要添加的项目的引用。

`result` 一个布尔值，它指定是否应添加指定项目。默认情况下，该值为 `true`。

示例

如果名为 `userHasAdminPrivs()` 的用户定义函数返回 `false`，下面的 `on(addItem)` 事件处理函数（附加到 `DataSet` 对象）将取消新项目的添加操作；否则允许添加项目。

```
on(addItem) {  
    if(globalObj.userHasAdminPrivs()) {  
        // 允许添加项目。  
        eventObj.result = true;  
    } else {  
        // 不允许添加项目；用户没有管理权限。  
        eventObj.result = false;  
    }  
}
```

另请参见

[DataSet.removeItem](#)

DataSet.addItem()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.addItem([obj])
```

参数

obj 要添加到此集合的对象。此参数是可选的。

返回

如果项目已添加到集合，则返回 `true` ；否则返回 `false`。

说明

方法 ；将指定传送对象添加到集合以便管理。新添加的项目将成为数据集的当前项目。如果未指定 *obj* 参数，则会通过 `DataSet.createItem()` 自动创建新的对象。

新项目在集合中的位置取决于是否为当前重复值指定了排序。如果未使用排序，则会将指定项目添加到集合的结尾。如果使用了排序，则会依据项目在当前排序中的位置将其添加到集合中。

有关初始化和构建传送对象的详细信息，请参阅 `DataSet.createItem()`。

示例

```
myDataSet.addItem(myDataSet.createItem());
```

另请参见

[DataSet.createItem\(\)](#)

DataSet.addSort()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.addSort(name, fieldList, sortOptions)
```

参数

name 一个字符串，它指定排序的名称。

fieldList 一个字符串数组，它指定要进行排序的字段名。

sortOptions 以下一个或多个整数（常数）值，它们指明为此排序使用哪些选项。使用逐位 OR 运算符 (`|`) 分隔多个值。值必须是下列之一：

- `DataSetIterator.Ascending` 按升序对项目进行排序。这是默认的排序选项（如果未指定任何选项）。
- `DataSetIterator.Descending` 依据指定的项目属性按降序对项目进行排序。
- `DataSetIterator.Unique` 防止排序（如果任何字段包含相同的值）。
- `DataSetIterator.CaseInsensitive` 在排序操作期间比较两个字符串时忽略大小写。默认情况下，如果正在进行排序的属性是字符串，则排序是区分大小写的。

如果将 `DataSetIterator.Unique` 指定为排序选项且正在排序的数据不唯一，添加了指定的排序名称或者 `fieldList` 数组中指定的属性在此数据集中不存在，则会抛出 `DataSetError` 例外。

返回

无。

说明

方法；依据 `fieldList` 参数指定的属性为当前重复值创建新的升序或降序排序。在创建了新的排序并将其存储在排序集合中以供稍后检索之后，新的排序将被自动分配给当前重复值。

示例

以下代码将创建名为 "rank" 的新排序，该排序将对 `DataSet` 对象的 "classRank" 字段按降序进行区分大小写的唯一排序。

```
myDataSet.addSort("rank", ["classRank"], DataSetIterator.Descending |
    DataSetIterator.Unique | DataSetIterator.CaseInsensitive);
```

另请参见

[DataSet.removeSort\(\)](#)

DataSet.afterLoaded

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(afterLoaded) {
    // 此处插入您的代码
}
listenerObject = new Object();
listenerObject.afterLoaded = function (eventObj) {
    // 此处插入您的代码
}
dataSet.addEventListener("afterLoaded", listenerObject)
```

说明

事件；在分配了 `DataSet.items` 属性后立即广播。

事件对象 (`eventObj`) 包含以下属性：

`target` 生成该事件的 `DataSet` 对象

type 字符串 "afterLoaded"。

示例

在此范例中，一旦分配了 `DataSet` `contact_ds` 中的项目，名为 `contactForm` 的表单（未显示）将变得可见。

```
contact_ds.addEventListener("afterLoaded", loadListener);
loadListener = new Object();
loadListener.afterLoaded = function (eventObj) {
    if(eventObj.target == "contact_ds") {
        contactForm.visible = true;
    }
}
```

DataSet.applyUpdates()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.applyUpdates()
```

返回

无。

说明

方法；指出 `DataSet.deltaPacket` 属性具有您可以使用数据绑定或直接通过动作脚本访问的值。在调用此方法之前，`DataSet.deltaPacket` 属性为 `null`。如果已通过 `DataSet.disableEvents()` 方法禁用了事件，则此方法没有效果。

调用此方法还会为当前 `DataSet.deltaPacket` 属性创建事务 ID，并发出 `deltaPacketChanged` 事件。有关详细信息，请参阅 `DataSet.deltaPacket`。

示例

以下代码将对 `myDataSet` 调用 `applyUpdates()` 方法。

```
myDataSet.applyUpdates();
```

另请参见

`DataSet.deltaPacket`

DataSet.calcFields

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(calcFields) {  
    // 此处插入您的代码  
}  
listenerObject = new Object();  
listenerObject.calcFields = function (eventObj) {  
    // insert your code here  
}  
dataSet.addEventListener("calcFields", listenerObject)
```

说明

事件；在需要确定集合中当前项目的计算所得字段的值时生成。计算所得字段是指 Kind 属性在“组件检查器”面板的“架构”选项卡上设置为“已计算”的字段。您创建的 calcFields 事件侦听器应执行必需的计算，并设置计算所得字段的值。

在非计算所得字段（即 Kind 属性在“组件检查器”面板的“架构”选项卡上设置为“数据”的字段）的值更新时，也会调用此事件。

有关 Kind 属性的详细信息，请参阅“使用 Flash”帮助中的“架构种类（仅限 Flash Professional）”。

小心：不要更改此事件中任何非计算所得字段的值，因为这将导致“无限循环”。只能设置 calcFields 事件内计算所得字段的值。

DataSet.changesPending()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.changesPending()
```

返回

一个布尔值。

说明

方法；如果集合或集合内的任何项目具有尚未发送到 DeltaPacket 对象中的待定更改，则返回 true；否则返回 false。

示例

如果 DataSet 集合或该集合内的任何项目具有尚未提交到 DeltaPacket 对象的修改，以下代码将启用“保存更改”按钮（未显示）。

```
if( data_ds.changesPending() ) {  
    saveChanges_btn.enabled = true;  
}
```

DataSet.clear()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.clear()
```

返回

无。

说明

方法；删除集合当前视图中的项目。哪些项目被视为“可查看”取决于任何当前的过滤器和当前重复值的范围设置。因此，调用此方法可能不会清除集合中的所有项目。要清除集合中的所有项目而不考虑当前重复值的视图，请使用 `DataSet.removeAll()`。

如果在调用此方法时 `DataSet.logChanges` 设置为 `true`，则会为集合内的所有项目将“remove”条目添加到 `DataSet.deltaPacket` 中。

示例

此范例将从 `DataSet` 集合的当前视图中删除所有项目。由于 `logChanges` 属性设置为 `true`，因此将记录这些项目的删除。

```
myDataSet.logChanges= true;  
myDataSet.clear();
```

另请参见

`DataSet.deltaPacket`、`DataSet.logChanges`

DataSet.createItem()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.createItem([itemData])
```

参数

itemData 与项目关联的数据。此参数是可选的。

返回

新构建的项目。

说明

方法；创建不与集合关联的项目。可以指定使用 `DataSet.itemClassName` 属性创建的对象。如果未指定 `DataSet.itemClassName` 值并且忽略了 `itemData` 参数，则会构建匿名对象。视 `DataSet.schema` 当前指定的架构而定，此匿名对象的属性将被设置为默认值。

在调用此方法时，`DataSet.newItem` 事件的所有侦听器将收到通知，并且能够在此方法返回项目之前对项目进行处理。指定的可选项目用于初始化通过 `DataSet.itemClassName` 属性指定的类，并在 `DataSet.itemClassName` 为空白时用作项目。

如果无法加载通过 `DataSet.itemClassName` 属性指定的类，则会抛出 `DataSetError` 例外。

示例

```
contact.itemClassName = "Contact";
var itemData = new XML("<contact_info><name>John Smith</name><phone>555.555.4567</phone><zip><pre>94025</pre><post>0556</post></zip></contact_info>");
contact.addItem(contact.createItem(itemData));
```

另请参见

[DataSet.itemClassName](#)、[DataSet.newItem](#)、[DataSet.schema](#)

DataSet.currentItem

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.currentItem
```

说明

属性（只读）；返回 `DataSet` 集合中的当前项目，或者，如果集合为空或集合当前重复值的视图为空，则会返回 `null`。

此属性提供对集合内的项目的直接访问。通过直接访问此对象所进行的更改既不会被跟踪（在 `DataSet.deltaPacket` 属性中），也不会是应用于此对象任何属性的任何架构设置。

示例

以下范例将显示 `customerName` 属性（在名为 `customerData` 的数据集的当前项目中定义）的值。

```
trace(customerData.currentItem.customerName);
```

DataSet.dataProvider

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`dataSet.dataProvider`

说明

属性；此数据集的 DataProvider 接口。此属性为用户界面控件（如 List 和 DataGrid 组件）提供数据。

示例

以下代码将 DataSet 对象的 dataProvider 属性分配给 DataGrid 组件的对应属性。

```
myGrid.dataProvider = myDataSet.dataProvider;
```

DataSet.deltaPacket

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`dataSet.deltaPacket`

说明

属性；返回 DeltaPacket 对象，该对象包含对 `dataSet` 集合及其项目所做的所有更改操作。在对 `dataSet` 调用 `DataSet.applyUpdates()` 之前，此属性为 `null`。

在调用 `DataSet.applyUpdates()` 时，会将一个事务 ID 分配给 DeltaPacket 对象。此事务 ID 用于在从服务器开始并返回到客户机的更新往返行程中标识 DeltaPacket 对象。DeltaPacket 对象（具有匹配事务 ID）为 `deltaPacket` 属性分配的任何后续值都将被视为对之前所发送更改的服务器响应。具有匹配 ID 的 DeltaPacket 对象用于更新集合，并报告包内指定的错误。

将会向 `DataSet.resolveDelta` 事件的侦听器报告错误或服务器消息。请注意，在将具有匹配 ID 的 DeltaPacket 对象分配给 `DataSet.deltaPacket` 时，将忽略 `DataSet.logChanges` 设置。没有匹配事务 ID 的 DeltaPacket 对象将更新集合，就像直接使用 DataSet API 一样。视 `dataSet` 和 DeltaPacket 对象的当前 `DataSet.logChanges` 设置而定，这样可能会创建额外的变量增量条目。

如果为 DeltaPacket 对象分配了匹配的事务 ID，并且无法在原始 DeltaPacket 对象中找到新分配的 DeltaPacket 对象中的某个项目，则会抛出 `DataSetError` 例外。

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.logChanges](#)、[DataSet.resolveDelta](#)

DataSet.deltaPacketChanged

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(deltaPacketChanged) {  
    // 此处插入您的代码  
}  
listenerObject = new Object();  
listenerObject.deltaPacketChanged = function (eventObj) {  
    // 此处插入您的代码  
}  
dataSet.addEventListener("deltaPacketChanged", listenerObject)
```

说明

事件；在指定 DataSet 对象的 `deltaPacket` 属性已更改并且可供使用时广播。

另请参见

[DataSet.deltaPacket](#)

DataSet.disableEvents()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.disableEvents()
```

返回

无。

说明

方法；为 DataSet 对象禁用事件。如果禁用了事件，则在对集合中的项目进行更改或 DataSet 对象滚动到集合中的另一项目时，将不会更新用户界面控件（如 DataGrid 组件）。

要重新启用事件，您必须调用 [DataSet.enableEvents\(\)](#)。disableEvents() 方法可被调用多次，因此 enableEvents() 必须被调用相同的次数才能重新启用事件发送。

示例

在此范例中，在对集合中的项进行更改前禁用事件，因此 DataSet 对象将不会尝试刷新控件和影响性能。

```
// 禁用数据集的事件
myDataSet.disableEvents();
myDataSet.last();
while(myDataSet.hasPrevious()) {
    var price = myDataSet.price;
    price = price * 0.5; // 所有项均为五折。
    myDataSet.price = price;
    myDataSet.previous();
}
// 告诉数据集现在应该更新控件
myDataSet.enableEvents();
```

另请参见

[DataSet.enableEvents\(\)](#)

DataSet.enableEvents()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.enableEvents()
```

返回

无。

说明

方法；在通过调用 [DataSet.disableEvents\(\)](#) 禁用了事件之后为 DataSet 对象重新启用事件。要为 DataSet 对象重新启用事件，enableEvents() 方法的调用次数必须等于或大于 disableEvents() 的调用次数。

示例

在此范例中，在对集合中的项进行更改前禁用事件，因此 DataSet 对象将不会尝试刷新控件和影响性能。

```
// 禁用数据集的事件
myDataSet.disableEvents();
myDataSet.last();
while(myDataSet.hasPrevious()) {
    var price = myDataSet.price;
    price = price * 0.5; // 所有项均为五折。
    myDataSet.price = price;
    myDataSet.previous();
}
// 告诉数据集现在应该更新控件
myDataSet.enableEvents();
```

另请参见

[DataSet.disableEvents\(\)](#)

DataSet.filtered

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

dataSet.filtered

说明

属性；一个布尔值，它指明是否过滤当前重复值中的数据。如果设置为 `true`，则会为集合中的每个项目调用 [DataSet.filterFunc](#) 指定的过滤函数。

示例

在以下范例中，对名为 `employee_ds` 的 `DataSet` 对象启用过滤。假设 `DataSet` 集合中的每条记录都包含一个名为 `empType` 的字段。如果当前项目中的 `empType` 字段设置为 "management"，以下过滤函数将返回 `true`；否则返回 `false`。

```
employee_ds.filtered = true;
employee_ds.filterFunc = function(item:Object) {
    // 过滤出职务为经理的那些员工 ...
    return(item.empType != "management");
}
```

另请参见

[DataSet.filterFunc](#)

DataSet.filterFunc

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.filterFunc = function(item:Object) {
    // 返回 true|false；
};
```

说明

属性；指定一个函数，该函数确定在集合的当前视图中包括哪些项目。如果 [DataSet.filtered](#) 设置为 `true`，则会为集合中的每个传送对象调用分配给此属性的函数。对于传送到函数的每个项目，如果项目应包括在当前视图中，则它应返回 `true`，或者，如果项目不应包括在当前视图中，则应返回 `false`。

示例

在以下范例中，对名为 `employee_ds` 的 `DataSet` 对象启用过滤。如果每个项目中的 `empType` 字段设置为 "management"，则指定的过滤函数返回 `true`；否则返回 `false`。

```
employee_ds.filtered = true;
employee_ds.filterFunc = function(item:Object) {
    // 过滤出职务为经理的那些员工 ...
    return(item.empType != "management");
}
```

另请参见

[DataSet.filtered](#)

DataSet.find()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.find(searchValues)
```

参数

searchValues 一个数组，包含要在当前排序中查找的一个或多个字段值。

返回

如果找到这些值，则返回 `true`；否则返回 `false`。

说明

方法；使用 *searchValues* 指定的字段值搜索集合的当前视图以查找项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。如果找到，则找到的项目将成为 `DataSet` 对象中的当前项目。

searchValues 指定的值的顺序必须与当前排序指定的字段列表的顺序相同（请参阅下面的范例）。

如果当前排序不唯一，则找到的传送对象不确定。如果要在非唯一排序中查找出现的第一个或最后一个传送对象，请使用 [DataSet.findFirst\(\)](#) 或 [DataSet.findLast\(\)](#)。

指定的数据转换基于基础字段的类型以及数组中指定的内容。例如，如果指定 `["05-02-02"]` 作为搜索值，则会使用基础日期字段，通过日期的 `DataType.setAsString()` 方法来转换值。如果指定了 `[new Date().getTime()]`，则使用日期的 `DataType.setAsNumber()` 方法。

示例

此范例将在当前集合中查找 `name` 和 `id` 字段相应包含值 "Bobby" 和 105 的项目。如果找到，则会使用 `DataSet.getItemId()` 方法来获取项目在集合中的唯一标识符，并使用 `DataSet.locateById()` 方法将当前重复值放在该项目上。

```
var studentID:String = null;
studentData.addSort("id", ["name","id"]);
// 查找由 "Bobby" 和 105 标识的传送对象。
// 请注意，搜索字段的顺序匹配
// addSort() 方法中指定的顺序。
if(studentData.find(["Bobby", 105])) {
    studentID = studentData.getItemId();
}
// 现在使用 locateById() 方法将当前重复值定位在
// 集合中 ID 与 studentID 匹配的项上
if(studentID != null) {
    studentData.locateById(studentID);
}
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.getItemId\(\)](#)、[DataSet.locateById\(\)](#)

DataSet.findFirst()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.findFirst(searchValues)
```

参数

searchValues 一个数组，包含要在当前排序中查找的一个或多个字段值。

返回

如果找到这些项，则返回 `true` ；否则，返回 `false`。

说明

方法；使用 *searchValues* 指定的字段值搜索集合的当前视图以查找第一个项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

searchValues 指定的值的顺序必须与当前排序指定的字段列表的顺序相同（请参阅下面的范例）。

指定的数据转换基于基础字段的类型以及数组中指定的内容。例如，如果指定的搜索值为 ["05-02-02"]，则会使用基础日期字段通过日期的 `setAsString()` 方法来转换值。如果指定的值是 `[new Date().getTime()]`，则使用日期的 `setAsNumber()` 方法。

示例

此范例将在当前集合中搜索 `name` 和 `age` 字段包含 "Bobby" 和 "13" 的第一个项目。如果找到，则会使用 `DataSet.getItemId()` 来获取项目在集合中的唯一标识符，并使用 `DataSet.locateById()` 将当前重复值放在该项目上。

```
var studentID:String = null;
studentData.addSort("nameAndAge", ["name", "age"]);
// 查找具有指定值的第一个传送对象。
// 请注意，搜索字段的顺序匹配
// addSort() 方法中指定的顺序。
if(studentData.findFirst(["Bobby", "13"])) {
    studentID = studentData.getItemId();
}
// 现在使用 locateById() 方法将当前重复值定位在
// 集合中 ID 与 studentID 匹配的项上
if(studentID != null) {
    studentData.locateById(studentID);
}
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.getItemId\(\)](#)、[DataSet.locateById\(\)](#)

DataSet.findLast()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.findLast(searchValues)
```

参数

searchValues 一个数组，包含要在当前排序中查找的一个或多个字段值。

返回

如果找到这些项，则返回 `true` ；否则，返回 `false`。

说明

方法；使用 *searchValues* 指定的字段值搜索集合的当前视图以查找最后一个项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

searchValues 指定的值的顺序必须与当前排序指定的字段列表的顺序相同（请参阅下面的范例）。

指定的数据转换基于基础字段的类型以及数组中指定的内容。例如，如果指定的搜索值为 ["05-02-02"]，则会使用基础日期字段通过日期的 `setAsString()` 方法来转换值。如果指定的值是 `[new Date().getTime()]`，则使用日期的 `setAsNumber()` 方法。

示例

此范例将在当前集合中搜索 `name` 和 `age` 字段包含 "Bobby" 和 "13" 的最后一个项目。如果找到，则会使用 `DataSet.getItemId()` 方法来获取项目在集合中的唯一标识符，并使用 `DataSet.locateById()` 方法将当前重复值放在该项目上。

```
var studentID:String = null;
studentData.addSort("nameAndAge", ["name", "age"]);
// 查找具有指定值的最后一个传送对象。
// 请注意，搜索字段的顺序匹配
// addSort() 方法中指定的顺序。
if(studentData.findLast(["Bobby", "13"])) {
    studentID = studentData.getItemId();
}
// 现在使用 locateById() 方法将当前重复值定位在
// ID 与 studentID 匹配的集合中项目的当前重复值。
if(studentID != null){
    studentData.locateById(studentID);
}
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.getItemId\(\)](#)、[DataSet.locateById\(\)](#)

DataSet.first()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.first()
```

返回

无。

说明

方法；使集合当前视图中的第一个项目成为当前项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

示例

以下代码将 `DataSet` `userData` 放在其集合中的第一个项目处，然后使用 `DataSet.currentItem` 属性显示该项目包含的 `price` 属性的值。

```
inventoryData.first();
trace("The price of the first item is:"+ inventoryData.currentItem.price);
```

另请参见

[DataSet.last\(\)](#)

DataSet.getItemId()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.getItemId([index])
```

参数

index 一个数值，指定要获取其 ID 的项目当前视图中的项目。此参数是可选的。

返回

一个字符串。

说明

方法；返回当前项目在集合中的标识符，或返回由 *index* 指定的项目的标识符。此标识符只有在此集合中才是唯一的，它由 [DataSet.addItem\(\)](#) 自动分配。

示例

以下代码将获取当前项目在集合中的唯一 ID，然后在“输出”面板中显示该 ID。

```
var itemNo:String = myDataSet.getItemId();  
trace("Employee id("+ itemNo+ ")");
```

另请参见

[DataSet.addItem\(\)](#)

DataSet.getIterator()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.getIterator()
```

返回

一个 ValueListIterator 对象。

说明

方法；返回此集合的一个新重复值；此重复值是正在使用的当前重复值的克隆，其中包括它在集合内的当前位置。此方法主要供想要访问同一集合同时出现的多个视图的高级用户使用。

示例

```
myIterator:ValueListIterator = myDataSet.getIterator();
```

```
myIterator.sortOn(["name"]);  
myIterator.find({name:"John Smith"}).phone = "555-1212";
```

DataSet.hasNext()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.hasNext()
```

返回

一个布尔值。

说明

方法；如果当前重复值位于其集合视图的结尾处，则返回 `false`；否则返回 `true`。

示例

此范例将重复集合当前视图中的所有项目（从开头位置开始），并对每个项目的 `price` 属性执行计算。

```
myDataSet.first();  
while(myDataSet.hasNext()) {  
    var price = myDataSet.currentItem.price;  
    price = price * 0.5; // 所有项均为五折。  
    myDataSet.currentItem.price = price;  
    myDataSet.next();  
}
```

另请参见

[DataSet.currentItem](#)、[DataSet.first\(\)](#)、[DataSet.next\(\)](#)

DataSet.hasPrevious()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.hasPrevious()
```

返回

一个布尔值。

说明

方法；如果当前重复值位于其集合视图的开头处，则返回 `false`；否则返回 `true`。

示例

此范例将重复集合当前视图中的所有项目（从最后一个项目开始），并对每个项目的 `price` 属性执行计算。

```
myDataSet.last();
while(myDataSet.hasPrevious()) {
    var price = myDataSet.currentItem.price;
    price = price * 0.5; // 所有项均为五折。
    myDataSet.currentItem.price = price;
    myDataSet.previous();
}
```

另请参见

[DataSet.currentItem](#)、[DataSet.skip\(\)](#)、[DataSet.previous\(\)](#)

DataSet.hasSort()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.hasSort(sortName)
```

参数

sortName 一个字符串，它包含使用 [DataSet.addSort\(\)](#) 创建的排序的名称

返回

一个布尔值。

说明

方法；如果 *sortName* 指定的排序存在，则返回 `true`；否则返回 `false`。

示例

以下代码将测试名为“customerSort”的排序是否存在。如果该排序已存在，则会通过 [DataSet.useSort\(\)](#) 方法使其成为当前排序。如果具有该名称的排序不存在，则会通过 [DataSet.addSort\(\)](#) 方法创建一个。

```
if(myDataSet.hasSort("customerSort"))
    myDataSet.useSort("customerSort");
else {
    myDataSet.addSort("customerSort", ["customer"], DataSetIterator.Descending);
}
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.useSort\(\)](#)

DataSet.isEmpty()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.isEmpty()
```

返回

一个布尔值。

说明

方法；如果指定的 DataSet 对象不包含任何项目（也就是说，如果 `dataSet.length == 0`），则返回 `true`。

示例

以下代码将在“删除记录”按钮（未显示）所应用到的 DataSet 对象为空时禁用该按钮。

```
if(userData.isEmpty()){  
    delete_btn.enabled = false;  
}
```

另请参见

[DataSet.length](#)

DataSet.items

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myDataSet.items
```

说明

属性；由 `myDataSet` 管理的项目数组。

示例

此范例将对象数组分配给 DataSet 对象的 `items` 属性。

```
var recData = [{id:0, firstName:"Mick", lastName:"Jones"},  
               {id:1, firstName:"Joe", lastName:"Strummer"},  
               {id:2, firstName:"Paul", lastName:"Simonon"}];  
myDataSet.items = recData;
```

DataSet.itemClassName

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.itemClassName
```

说明

属性；一个字符串，指明在将项目添加到集合中时应创建的类的名称。您指定的类必须实现 **TransferObject** 接口，如下所示。

```
interface mx.data.to.TransferObject {  
    function clone():Object;  
    function getPropertyData():Object;  
    function setPropertyData(propData:Object):Void;  
}
```

您也可以在属性检查器中设置该属性。

要使指定的类在运行时可用，您必须同时在 SWF 文件的代码内的某处对该类进行完全限定引用，如下代码片断中所示：

```
var myItem:my.package.myItem;
```

如果试图在载入了 `DataSet.items` 数组之后修改此属性的值，则会抛出 `DataSetError` 例外。

有关 **TransferObject** 接口的详细信息，请参阅第 479 页的“**TransferObject 接口**”。

DataSet.iteratorScrolled

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(iteratorScrolled) {  
    // 此处插入您的代码  
}  
listenerObject = new Object();  
listenerObject.iteratorScrolled = function (eventObj) {  
    // 此处插入您的代码  
}  
dataSet.addEventListener("iteratorScrolled", listenerObject)
```


说明

事件；在当前重复值已滚动到集合中的新项目之后立即生成。

事件对象 (*eventObj*) 包含以下属性：

target 生成该事件的 DataSet 对象。

type 字符串 "iteratorScrolled"。

scrolled 一个数值，它指定重复值滚动了多少项目；正值指明重复值在集合中向前移动；负值指明它在集合中向后移动。

示例

在此范例中，应用程序的状态栏（未显示）将在当前重复值的位置更改时更新。

```
on(iteratorScrolled) {  
    var dataSet:mx.data.components.DataSet = eventObj.target;  
    var statusBarText = dataSet.fullname+" Acct  
    #:"+dataSet.getField("acctnum").getAsString();  
    setStatusBar(statusBarText);  
}
```

DataSet.last()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.last()
```

返回

无。

说明

方法；使集合当前视图中的最后一个项目成为当前项目。

示例

以下代码（附加到 Button 组件）将转到 DataSet 集合中的最后一个项目。

```
function goLast(eventObj:obj) {  
    inventoryData.last();  
}  
goLast_btn.addEventListener("click", goLast);
```

另请参见

[DataSet.first\(\)](#)

DataSet.length

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.length
```

说明

属性（只读）；指定集合当前视图中的项目数。这个可查看的项目数以当前过滤器和范围的设置为基础。

示例

如果用户在数据集中没有输入足够的内容，以下范例可能会使用可编辑的 DataGrid 组件警示用户。

```
if(myDataSet.length < MIN_REQUIRED) {  
    alert("You need at least "+MIN_REQUIRED);  
}
```

DataSet.loadFromSharedObj()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.loadFromSharedObj(objName, [localPath])
```

参数

objName 一个字符串，指定要检索的共享对象的名称。名称中可以包含斜杠（例如“work/addresses”）。在指定的名称中不允许包括空格和以下字符：

~ % & \ ; : " ' , < > ? #

localPath 一个可选字符串参数，指定创建了共享对象的 SWF 文件的完整或部分路径。此字符串用于确定对象存储在用户计算机上的何处。默认值是 SWF 文件的完整路径。

返回

无。

说明

方法；加载从共享对象中恢复此 DataSet 集合所需的所有相关数据。要将 DataSet 集合保存到共享对象，请使用 `DataSet.saveToSharedObj()`。DataSet.loadFromSharedObject() 方法将覆盖此 DataSet 集合内可能存在的任何数据或待定更改。请注意，DataSet 集合的实例名称用于在指定的共享对象内标识数据。

如果指定的共享对象未找到或从中检索数据时有问题，此方法将抛出 `DataSetError` 例外。

示例

此范例将尝试加载与数据集（名为 `myDataSet`）关联的共享对象（名为 `webapp/customerInfo`）。将在 `try...catch` 代码块内调用该方法。

```
try {
    myDataSet.loadFromSharedObj("webapp/customerInfo");
}
catch(e:DataSetError) {
    trace("Unable to load shared object.");
}
```

另请参见

[DataSet.saveToSharedObj\(\)](#)

DataSet.locateById()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.locateById(id)
```

参数

id 待查找集合中项目的字符串标识符。

返回

一个布尔值。

说明

方法；将当前重复值放在 ID 与 *id* 匹配的集合项目上。如果指定的 ID 可与集合中的某个项目匹配，此方法将返回 `true`；否则返回 `false`。

示例

此范例将使用 `DataSet.find()` 在当前集合中搜索 `name` 和 `id` 字段分别包含值 "Bobby" 和 105 的项目。如果找到，则会使用 `DataSet.getItemId()` 方法来获取该项目的唯一标识符，并使用 `DataSet.locateById()` 方法将当前重复值放在该项目上。

```
var studentID:String = null;
studentData.addSort("id", ["name","id"]);
if(studentData.find(["Bobby", 105])) {
    studentID = studentData.getItemId();
    studentData.locateById(studentID);
}
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.find\(\)](#)、[DataSet.getItemId\(\)](#)

DataSet.logChanges

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.logChanges
```

说明

属性；一个布尔值，它指定是 (true) 否 (false) 应在 `DataSet.deltaPacket` 中记录对数据集或其项目进行的更改。

如果此属性设置为 true，在集合级别和项目级别执行的操作将被记录。集合级别的更改包括在集合中添加项目或从中删除项目的操作。项目级别的更改包括对项目进行的属性更改，以及通过 DataSet 组件对项目进行的方法调用。

示例

以下范例将为名为 `userData` 的 DataSet 对象禁用记录操作。

```
userData.logChanges = false;
```

另请参见

[DataSet.deltaPacket](#)

DataSet.modelChanged

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(modelChanged) {  
    // 此处插入您的代码  
}  
listenerObject = new Object();  
listenerObject.modelChanged = function (eventObj) {  
    // 此处插入您的代码  
}  
dataSet.addEventListener("modelChanged", listenerObject)
```

说明

事件；在集合的某些方面已更改（例如，删除项目或将项目添加到集合、项目属性的值发生更改或对集合进行了过滤或排序）时广播。

事件对象 (*eventObj*) 包含以下属性：

`target` 生成该事件的 DataSet 对象。

`type` 字符串 "iteratorScrolled"。

`firstItem` 受更改影响的集合中第一个项目的索引（号）。

`lastItem` 受更改影响的集合中最后一个项目的索引（号），如果只有一个项目受影响，则它与 `firstItem` 相等。

`fieldName` 一个字符串，它包含受影响的字段的名称。除非对 `DataSet` 对象的属性进行了更改，否则此属性为 `undefined`。

`eventName` 一个字符串，它描述发生的更改。它可以是以下值之一：

字符串值	描述
"addItem"	已添加一系列项目。
"filterModel"	已过滤模型，并且视图需要刷新（重置滚动位置）。
"removeItem"	已删除一系列项目。
"schemaLoaded"	已声明数据提供程序的字段定义。
"sort"	数据已排序。
"updateAll"	整个视图都需要刷新（除滚动位置外）。
"updateColumn"	数据提供程序内整个字段的定义都需要刷新。
"updateField"	项目内的字段已更改，并且需要刷新。
"updateItems"	一系列项目需要刷新。

示例

在此范例中，如果已从集合中删除了项目且目标 `DataSet` 对象没有其他项目，则会禁用“删除项”按钮。

```
on(modelChanged) {
    delete_btn.enabled = ((eventObj.eventName == "removeItems") &&
        (eventObj.target.isEmpty()));
}
```

另请参见

[DataSet.isEmpty\(\)](#)

DataSet.newItem

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(newItem) {
    // 此处插入您的代码
}
listenerObject = new Object();
listenerObject.newItem = function (eventObj) {
    // 此处插入您的代码
}
```

```
}  
dataSet.addEventListener("newItem", listenerObject)
```

说明

事件；在通过 `DataSet.createItem()` 构建了新的传送对象时广播。此事件的侦听器可在将项目添加到集合之前对其进行修改。

事件对象 (*eventObj*) 包含以下属性：

target 生成该事件的 `DataSet` 对象。

type 字符串 "iteratorScrolled"。

item 指向所创建项目的引用。

示例

此范例在将新创建的项目添加到集合之前对其进行修改。

```
function newItemEvent(evt:Object):Void {  
    var employee:Object = evt.item;  
    employee.name = "newGuy";  
    // 属性数据恰好是 XML  
    employee.zip =  
        employee.getPropertyData().firstChild.childNodes[1].attributes.zip;  
}  
employees_ds.addEventListener("newItem", newItemEvent);
```

DataSet.next()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.next()
```

返回

无。

说明

方法；使集合当前视图中的下一个项目成为当前项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

示例

此范例将遍历 `DataSet` 对象中的所有项目（从第一个项目开始），并对每个项目中的字段执行计算。

```
myDataSet.first();  
while(myDataSet.hasNext()) {  
    var price = myDataSet.price;  
    price = price * 0.5; // 所有项均为五折。  
    myDataSet.price = price;  
    myDataSet.next();  
}
```

另请参见

[DataSet.first\(\)](#)、[DataSet.hasNext\(\)](#)

DataSet.previous()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.previous()
```

返回

无。

说明

方法；使集合当前视图中的上一个项目成为当前项目。在当前视图中包含哪些项取决于任何当前过滤器和范围设置。

此范例将集合当前视图中的所有项目（从最后一个项目开始），并对每个项目中的字段执行计算。

```
myDataSet.last();
while(myDataSet.hasPrevious()) {
    var price = myDataSet.price;
    price = price * 0.5; // 所有项均为五折。
    myDataSet.price = price;
    myDataSet.previous();
}
```

另请参见

[DataSet.first\(\)](#)、[DataSet.hasNext\(\)](#)

DataSet.properties

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.properties
```

说明

属性（只读）；返回一个对象，该对象包含此集合内任意传送对象所有公开的属性（字段）。

示例

此范例将显示名为 `myDataSet` 的 `DataSet` 对象中属性的所有名称。

```
for(var i in myDataSet.properties) {  
    trace("field '"+i+ "' has value "+ myDataSet.properties[i]);  
}
```

DataSet.readOnly

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.readOnly
```

说明

属性；一个布尔值，它指定此集合是可以修改 (`false`) 还是只读 (`true`)。将此属性设置为 `true` 可防止对集合进行更新。

您也可以在属性检查器中设置该属性。

示例

以下范例使名为 `myDataSet` 的 `DataSet` 对象变为只读，然后尝试更改属于集合中当前项目的属性的值。这将抛出一个例外。

```
myDataSet.readOnly = true;  
// 这将抛出一个例外  
myDataSet.currentItem.price = 15;
```

另请参见

[DataSet.currentItem](#)

DataSet.removeAll()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.removeAll()
```

参数

无。

返回

无。

描述

方法；删除 DataSet 集合中的所有项目。

范例

此范例将删除 DataSet 集合 `contact_ds` 中的所有项目：

```
contact_ds.removeAll();
```

DataSet.removeItem

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(removeItem) {  
    // 此处插入您的代码  
}  
listenerObject = new Object();  
listenerObject.removeItem = function (eventObj) {  
    // 此处插入您的代码  
}  
dataSet.addEventListener("removeItem", listenerObject)
```

说明

事件；就在从此集合中删除新项目之前生成。

如果将事件对象的 `result` 属性设置为 `false`，则删除操作将被取消；如果将其设置为 `true`，则允许删除操作。

事件对象 (`eventObj`) 包含以下属性：

`target` 生成该事件的 DataSet 对象。

`type` 字符串 "removeItem"。

`item` 指向集合中要删除的项目的引用。

`result` 一个布尔值，它指定是否应删除项目。默认情况下，该值为 `true`。

示例

在此范例中，如果名为 `userHasAdminPrivs()` 的用户定义函数返回 `false`，则 `on(removeItem)` 事件处理函数取消新项目的删除操作；否则允许删除操作。

```
on(removeItem) {  
    if(globalObj.userHasAdminPrivs()) {  
        // 允许删除项目。  
        eventObj.result = true;  
    } else {  
        // 不允许删除项目；用户没有管理权限。  
        eventObj.result = false;  
    }  
}
```

另请参见

[DataSet.addItem](#)

DataSet.removeItem()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.removeItem([item])
```

参数

item 应删除的项目。此参数是可选的。

返回

一个布尔值。如果成功删除了项目，则返回 `true`；否则返回 `false`。

说明

方法；从集合中删除指定项目；或者，如果忽略了 *item* 参数，则删除当前项目。如果 [DataSet.logChanges](#) 为 `true`，则将此操作记录到 [DataSet.deltaPacket](#)。

示例

以下代码（附加到 `Button` 组件的实例）将删除名为 `usersData` 的 `DataSet` 对象中的当前项目，该对象驻留在与 `Button` 实例相同的时间轴上。

```
on(click){  
    _parent.usersData.removeItem();  
}
```

另请参见

[DataSet.deltaPacket](#)、[DataSet.logChanges](#)

DataSet.removeRange()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.removeRange()
```

返回

无。

说明

方法；删除通过当前重复值的 `DataSet.setRange()` 指定的当前端点设置。

示例

```
myDataSet.addSort("name_id", ["name", "id"]);
myDataSet.setRange(["Bobby", 105],["Cathy", 110]);
while(myDataSet.hasNext()) {
    myDataSet.gradeLevel = "5"; // 更改此范围中的所有级别
    myDataSet.next();
}
myDataSet.removeRange();
myDataSet.removeSort("name_id");
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.hasNext\(\)](#)、[DataSet.next\(\)](#)、
[DataSet.removeSort\(\)](#)、[DataSet.setRange\(\)](#)

DataSet.removeSort()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.removeSort(sortName)
```

参数

sortName 一个字符串，它指定要删除的排序的名称。

返回

无。

说明

方法；从此 `DataSet` 对象中删除指定排序（如果排序存在）。如果指定的排序不存在，此方法将抛出 `DataSetError` 例外。

示例

```
myDataSet.addSort("name_id", ["name", "id"]);
myDataSet.setRange(["Bobby", 105],["Cathy", 110]);
while(myDataSet.hasNext()) {
    myDataSet.gradeLevel = "5"; // 更改此范围中的所有级别
    myDataSet.next();
}
myDataSet.removeRange();
myDataSet.removeSort("name_id");
```

另请参见

[DataSet.applyUpdates\(\)](#)、[DataSet.hasNext\(\)](#)、[DataSet.next\(\)](#)、
[DataSet.removeRange\(\)](#)、[DataSet.setRange\(\)](#)

DataSet.resolveDelta

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
on(resolveDelta) {  
    // 此处插入您的代码  
}  
listenerObject = new Object();  
listenerObject.resolveDelta = function (eventObj) {  
    // 此处插入您的代码  
}  
dataSet.addEventListener("resolveDelta", listenerObject)
```

说明

事件；在将 `DeltaPacket` 对象分配给 `DataSet.deltaPacket` 时广播，它的事务 ID 与之前从 `DataSet` 对象中检索的 `DeltaPacket` 对象的事务 ID 匹配，并且具有与该 `DeltaPacket` 对象所包含任何 `Delta` 或 `DeltaItem` 对象关联的消息。

此事件使您能够在尝试应用先前提交的更改时协调从服务器返回的任何错误。通常，您可以使用此事件来显示包含冲突值的“协调对话框”，从而允许用户对数据进行修改以便能够重新发送。

事件对象 (`eventObj`) 包含以下属性：

`target` 生成该事件的 `DataSet` 对象。

`type` 字符串 "resolveDelta"。

`data` `Delta` 对象及关联 `DeltaItem` 对象的数组，它具有长度不为零的消息。

示例

此范例显示一个名为 `reconcileForm` 的表单（未显示），并对该表单对象调用一个方法 (`setReconcileData()`)，该方法使用户能够协调服务器返回的任何冲突值。

```
myDataSet.addEventListener("resolveDelta", resolveDelta);  
function resolveDelta(eventObj:Object) {  
    reconcileForm.visible = true;  
    reconcileForm.setReconcileData(eventObj.data);  
}  
// 在 reconcileForm 代码中  
function setReconcileData(data:Array):Void {  
    var di:DeltaItem;  
    var ops:Array = ["property", "method"];  
    var cl:Array;  
    // 更改列表  
    var msg:String;  
    for (var i = 0; i<data.length; i++) {  
        cl = data[i].getChangeList();  
        for (var j = 0; j<cl.length; j++) {  
            di = cl[j];  
            msg = di.getMessage();  
            if (msg.length>0) {
```

```

        trace("The following problem occurred '"+msg+"' while performing a
        '"+ops[di.kind]+"' modification on/with '"+di.name+"' current server value
        ["+di.curValue+"] , value sent ["+di.newValue+"] Please fix!");
    }
}
}
}
}

```

DataSet.saveToSharedObj()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.saveToSharedObj(objName, [localPath])
```

参数

objName 一个字符串，它指定要创建的共享对象的名称。名称中可以包含斜杠（例如“work/addresses”）。在指定的名称中不允许包括空格和以下字符：

~ % & \ ; : " ' , < > ? #

localPath 一个可选字符串参数，指定创建了共享对象的 SWF 文件的完整或部分路径。此字符串用于确定对象将存储在用户计算机上的何处。默认值是 SWF 文件的完整路径。

返回

无。

说明

方法；保存将此 DataSet 集合恢复到共享对象所需的所有相关数据。它使用户能够在与来源数据断开连接的情况下工作（如果来源数据是网络资源）。此方法将覆盖此 DataSet 集合的指定共享对象内可能存在的任何数据。要从共享对象恢复 DataSet 集合，请使用 [DataSet.loadFromSharedObj\(\)](#)。请注意，DataSet 集合的实例名称用于在指定的共享对象内标识数据。

如果无法创建共享对象或在将数据刷新到其中时有问题，此方法将抛出 `DataSetError` 例外。

示例

此范例将在 `try...catch` 块内调用 `saveToSharedObj()`，并且，如果在将数据保存到共享数据时有问题，此范例将显示一个错误。

```

try {
    myDataSet.saveToSharedObj("webapp/customerInfo");
}
catch(e:DataSetError) {
    trace("Unable to create shared object");
}

```

另请参见

[DataSet.loadFromSharedObj\(\)](#)

DataSet.schema

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

dataSet.schema

说明

属性；为此 DataSet 对象提供以 XML 方式表示的架构。分配给此属性的 XML 必须具有以下格式：

```
<?xml version="1.0"?>
<properties>
  <property name="propertyName">
    <type name="dataType" />
    <encoder name="dataType">
      <options>
        <dataFormat>format options</dataFormat>
      </options>
    </encoder>
    <kind name="dataKind">
      <options>
      </options>
    </kind>
  </property>
  <property> ...</property>
  ...
</properties>
```

如果指定的 XML 不符合上面的格式，则会抛出 DataSetError 例外。

示例

```
myDataSet.schema = new XML("<properties><property name='billable'> ..etc..</properties>");
```

DataSet.selectedIndex

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

dataSet.selectedIndex

说明

属性；指定集合内的选定索引。可以将此属性绑定到 DataGrid 或 List 组件中的选定项目，反之亦然。有关演示此操作的完整范例，请参阅第 177 页的“创建具有 DataSet 组件的应用程序”。

示例

以下范例将 DataSet 对象 (userData) 的选定索引设置为 DataGrid 组件 (userGrid) 中的选定索引。

```
userData.selectedIndex = userGrid.selectedIndex;
```

DataSet.setIterator()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.setIterator(iterator)
```

参数

iterator 对 DataSet.getIterator() 的调用所返回的重复值对象。

返回

无。

说明

方法；将指定的重复值分配给此 DataSet 对象，并使其成为当前重复值。指定的重复值必须来自之前对其所分配到的 DataSet.getIterator() 的调用；否则，将抛出 DataSetError 例外。

示例

```
myIterator:ValueListIterator = myDataSet.getIterator();  
myIterator.sortOn(["name"]);  
myDataSet.setIterator(myIterator);
```

另请参见

[DataSet.getIterator\(\)](#)

DataSet.setRange()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.setRange(startValues, endValues)
```

参数

startValues 范围内第一个传送对象的属性的关键值数组。

endValues 范围内最后一个传送对象的属性的关键值数组。

返回

无。

说明

方法；设置当前重复值的端点。这些端点定义一个范围，重复值将在其中进行运算。只有在通过 `DataSet.applyUpdates()` 为当前重复值设置了有效的排序时，此方法才有效。

如果需要分组的值，则为当前重复值设置一个范围比使用过滤函数更有效（请参阅 `DataSet.filterFunc()`）。

示例

```
myDataSet.addSort("name_id", ["name", "id"]);
myDataSet.setRange(["Bobby", 105], ["Cathy", 110]);
while(myDataSet.hasNext()) {
    myDataSet.gradeLevel = "5"; // 更改此范围中的所有级别
    myDataSet.next();
}
myDataSet.removeRange();
myDataSet.removeSort("name_id");
```

另请参见

`DataSet.applyUpdates()`、`DataSet.hasNext()`、`DataSet.next()`、`DataSet.removeRange()`、`DataSet.removeSort()`

DataSet.skip()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.skip(offSet)
```

参数

offSet 一个整数，指定重复值位置要移动的记录数。

返回

无。

说明

方法；将当前重复值的位置在集合内向前或向后移动一定距离（由 *offSet* 指定）。如果 *offSet* 值为正，则重复值的位置向前移动；如果值为负，则向后移动。

如果指定的 *offset* 超出了集合的开头（或结尾），重复值将被定位在集合的开头（或结尾）。

示例

此范例将当前重复值定位在集合中的第一个项目，然后移动到倒数第二个项目，并对属于该项目的字段执行计算。

```
myDataSet.first();  
// 移动到倒数第二个项目  
var itemsToSkip = myDataSet.length - 2;  
myDataSet.skip(itemsToSkip).price = myDataSet.amount * 10;
```

DataSet.useSort()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
dataSet.useSort(sortName, order)
```

参数

sortName 一个字符串，它包含要使用的排序的名称。

order 一个整数值，它指明排序的排序顺序；该值必须为 `DataSetIterator.Ascending` 或 `DataSetIterator.Descending`。

返回

无。

说明

方法；将当前重复值的排序切换到 *sortName* 指定的排序（如果存在）。如果 *sortName* 指定的排序不存在，则会抛出 `DataSetError` 例外。

要创建排序，请使用 `DataSet.applyUpdates()`。

示例

此代码使用 `DataSet.hasSort()` 来确定名为 "customer" 的排序是否存在。如果存在，代码将调用 `DataSet.useSort()` 使 "customer" 成为当前排序。否则，代码将使用 `DataSet.addSort()` 创建一个具有该名称的排序。

```
if(myDataSet.hasSort("customer")) {  
    myDataSet.useSort("customer");  
} else {  
    myDataSet.addSort("customer", ["customer"], DataSetIterator.Descending);  
}
```

另请参见

`DataSet.applyUpdates()`、`DataSet.hasSort()`

DateChooser 组件（仅限 Flash Professional）

DateChooser 组件是一个允许用户选择日期的日历。它包含一些按钮，这些按钮允许用户在月份之间来回滚动并单击某个日期将其选中。可以设置指明月份和日名称、星期的第一天和任何禁用日期以及突出显示当前日期的参数。

每个 DateChooser 实例的实时预览反映了创作过程中属性检查器或“组件检查器”面板指明的值。

使用 DateChooser 组件（仅限 Flash Professional）

DateChooser 可用于任何您想让用户选择日期的场合。例如，您可以将 DateChooser 组件用于酒店预订系统中，其中某些日期是可选择的，其他日期则是禁用的。您也可以在用户选择日期时显示当前事件（如表演或会议）的应用程序中使用 DateChooser 组件。

DateChooser 参数

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 DateChooser 组件实例设置的创作参数：

monthNames 设置在日历的标题行中显示的月份名称。该值是一个数组，并且默认值是 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

dayNames 设置一周中各天的名称。该值是一个数组，并且默认值是 ["S", "M", "T", "W", "T", "F", "S"]。

firstDayOfWeek 指明一周中的哪一天（其值为 0-6，0 是 dayNames 数组的第一个元素）显示在 DateChooser 的第 1 列中。此属性更改“日”列的显示顺序。

disabledDays 指明一周中禁用的各天。该参数是一个数组，并且最多具有 7 个值。默认值为 []（空数组）。

showToday 指明是否突出显示今天的日期。默认值为 true。

您可以编写动作脚本，使用其属性、方法和事件来控制 DateChooser 组件的这些和其他选项。有关详细信息，请参阅第 220 页的“DateChooser 类（仅限 Flash Professional）”。

创建具有 DateChooser 组件的应用程序

以下过程解释了如何在创作时将 DateChooser 组件添加到应用程序。在此范例中，DateChooser 使用户能够从航空公司预订系统中选择一个日期。10 月 15 日之前的所有日期必须被禁用。同时，12 月中的某个范围必须被禁用以创建一个假日关闭期，并且星期一也必须被禁用。

要创建具有 DateChooser 组件的应用程序，请执行以下操作：

- 1 在“组件”面板中双击 DateChooser 组件，将其添加到舞台。
- 2 在属性检查器中，输入实例名称 **flightCalendar**。
- 3 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以设置可选日期的范围：

```
flightCalendar.selectableRange = {rangeStart:new Date(2003, 9, 15),  
rangeEnd:new Date(2003, 11, 31)}
```

此代码将一个值分配给动作脚本对象中的 selectableRange 属性，该属性包含两个 Date 对象，这两个对象分别具有变量名称 rangeStart 和 rangeEnd。这样，此代码就定义了用户可以在其内选择日期的范围的上限和下限。

- 4 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以设置假日禁用日期的范围：

```
flightCalendar.disabledRanges = [{rangeStart:new Date(2003, 11, 15),
rangeEnd:new Date(2003, 11, 26)}];
```
- 5 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以禁用各星期一：

```
flightCalendar.disabledDays=[1];
```
- 6 “控制” > “测试影片”。

自定义 DateChooser 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平和垂直方向上改变 DateChooser 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改” > “变形”命令。在运行时，使用 setSize() 方法（请参阅 [UIObject.setSize\(\)](#)）。

对 DateChooser 组件使用样式

您可以设置样式属性来更改日期选择器实例的外观。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 25 页的“使用样式自定义组件的颜色和文本”。

DateChooser 组件支持下列光晕样式：

样式	描述
themeColor	用于变换图象和所选日期的发亮的颜色。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	文本的字体系列。
fontSize	字体的磅值。
fontStyle	字体样式；“normal”或“italic”。
fontWeight	字体粗细；“normal”或“bold”。
textDecoration	文本修饰：“none”或“underline”。

对 DateChooser 组件使用外观

DateChooser 组件会使用外观来表现其可视状态。要在创作时设置 DateChooser 组件的外观，请在其中一个主题 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/DateChooser Assets/Elements/Month skins states 文件夹中的外观元件。有关详细信息，请参阅第 33 页的“关于设置组件外观”。

在此组件中，只有月份滚动按钮才能被动态地设置外观。DateChooser 组件使用以下外观属性：

属性	描述
falseUpSkin	弹起状态。默认值为 fwdMonthUp 和 backMonthUp。
falseDownSkin	按下状态。默认值为 fwdMonthDown 和 backMonthDown。
falseDisabledSkin	禁用状态。默认值为 fwdMonthDisabled 和 backMonthDisabled。

DateChooser 类（仅限 Flash Professional）

继承 [UIObject](#) > [UIComponent](#) > [DateChooser](#)

动作脚本类名称 [mx.controls.DateChooser](#)

[DateChooser](#) 类的属性使您能够访问选定的日期以及显示的月份和年份。您也可以设置日和月的名称，指明禁用的日期和可选的日期，设置一周中的第一天，以及指明当前日期是否应突出显示。

使用动作脚本设置 [DateChooser](#) 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.DateChooser.version);
```

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

DateChooser 类的方法摘要

继承 [UIObject](#) 和 [UIComponent](#) 中的所有方法。

DateChooser 类的属性摘要

属性	描述
DateChooser.dayNames	一个数组，指明一周中各天的名称。
DateChooser.disabledDays	一个数组，指明为日期选择器中所有适用日期禁用的一周中的各天。
DateChooser.disabledRanges	禁用日期的范围或单个禁用的日期。
DateChooser.displayedMonth	一个数字，指明 <code>monthNames</code> 数组中要在日期选择器中显示的元素。
DateChooser.displayedYear	一个数字，指明要显示的年份。
DateChooser.firstDayOfWeek	一个数字，指明 <code>dayNames</code> 数组中要在日期选择器的第一列中显示的元素。
DateChooser.monthNames	一个字符串数组，指明月份名称。
DateChooser.selectableRange	单个可选日期或可选日期的范围。
DateChooser.selectedDate	一个 <code>Date</code> 对象，指明可选日期。
DateChooser.showToday	一个布尔值，指明是否突出显示当前日期。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有属性。

DateChooser 类的事件摘要

事件	描述
DateChooser.change	在选择一个日期时广播。
DateChooser.scroll	在按下月份按钮时广播。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有事件。

DateChooser.change

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
on(change){  
    ...  
}
```

用法 2：

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

描述

事件；当选择日期时，向所有已注册的侦听器广播。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到 `DateChooser` 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（附加到日期选择器 `myDC`）将 “_level0.myDC” 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 (*chooserInstance*) 调度一个事件（在本例中为 `change`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，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 510 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 myDC 的 DateChooser 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（在本例中是 myDC）。从事件对象的 target 属性中可以访问 NumericStepper.maximum 属性。最后一行从 myDC 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.change = function(eventObj){
    trace("date selected " + eventObj.target.selectedDate);
}
myDC.addEventListener("change", form);
```

DateChooser.dayNames

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDC.dayNames

描述

属性；包含一周中各天的名称的数组。星期日是第一天（索引位置为 0），其他各天的名称按顺序跟在后面。默认值为 ["S", "M", "T", "W", "T", "F", "S"]。

范例

以下范例将一周的第 5 天（星期四）的值从 “T” 更改为 “R”：

```
myDC.dayNames[4] = "R";
```

DateChooser.disabledDays

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDC.disabledDays

描述

属性；指明一周中被禁用的各天的数组。一个月中属于指定日期范围内的所有日期都被禁用。此数组的元素可以具有介于 0（星期日）和 6（星期六）之间的值。默认值为 []（空数组）。

范例

以下范例将禁用星期日和星期六，以使用户只能选择每周工作日：

```
myDC.disabledDays = [0, 6];
```

DateChooser.disabledRanges

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDC.disabledRanges
```

描述

属性；禁用某一天或一个范围中的各天。此属性是对象数组。该数组中的每个对象必须或者是指定要禁用的某一天的 Date 对象，或者是包含 rangeStart 和 / 或 rangeEnd 属性（这两个属性的值必须是 Date 对象）的对象。rangeStart 和 rangeEnd 属性描述日期范围的界限。如果这两个属性中的任何一个属性被省略，则范围在该方向上无限制。

disabledRanges 的默认值未定义。

在为 disabledRanges 属性定义日期时应指定一个完整日期。例如，应指定 new Date(2003,6,24)，而不是 new Date()。如果没有指定一个完整日期，则该时间返回为 00:00:00。

范例

以下范例定义一个数组，该数组具有 rangeStart 和 rangeEnd Date 对象，禁用 5 月 7 日和 6 月 7 日之间的日期：

```
myDC.disabledRanges = [ {rangeStart:new Date(2003, 4, 7), rangeEnd:new Date(2003, 5, 7)}];
```

以下范例禁用 11 月 7 日之后的所有日期：

```
myDC.disabledRanges = [ {rangeStart:new Date(2003, 10, 7)}];
```

以下范例禁用 10 月 7 日之前的所有日期：

```
myDC.disabledRanges = [ {rangeEnd:new Date(2002, 9, 7)}];
```

以下范例仅禁用 11 月 7 日：

```
myDC.disabledRanges = [ new Date(2003, 11, 7) ];
```

DateChooser.displayedMonth

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDC.displayedMonth`

描述

属性；指明显示哪一月份的数字。该数字指明 `monthNames` 数组中的元素，其中 0 是第一个月份。默认值是当前日期的月份。

范例

以下范例将显示的月份设置为 December（12 月）：

```
myDC.displayedMonth = 11;
```

另请参见

[DateChooser.displayedYear](#)

DateChooser.displayedYear

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDC.displayedYear`

描述

属性；指明显示哪一年份的四位数数字。默认值为当前年份。

范例

以下范例将显示的年份设置为 2010：

```
myDC.displayedYear = 2010;
```

另请参见

[DateChooser.displayedMonth](#)

DateChooser.firstDayOfWeek

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myDC.firstDayOfWeek`

描述

属性；一个数值，指明一周中的哪一天（0-6，0 是 `dayNames` 数组的第一个元素）显示在 `DateChooser` 组件的第一列中。更改该属性将更改日列的顺序，但对 `dayNames` 属性的顺序没有影响。默认值为 0（星期日）。

范例

以下范例将一周的第一天设置为 Monday（星期一）：

```
myDC.firstDayOfWeek = 1;
```

另请参见

[DateChooser.dayNames](#)

DateChooser.monthNames

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDC.monthNames
```

描述

属性；一个字符串数组，在 `DateChooser` 组件的顶部指明月份名称。默认值为 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

范例

以下范例将设置 `myDC` 实例的月份名称：

```
myDC.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",  
"Sept", "Oct", "Nov", "Dec"];
```

DateChooser.scroll

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
on(scroll){  
  ...  
}
```

用法 2：

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

描述

事件；在按下月份按钮时广播到所有注册的侦听器。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到 `DateChooser` 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 `myDC`）将 “_level0.myDC” 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 (`myDC`) 发送一个事件（在本例中为 `scroll`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您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。`scroll` 事件的事件对象还有另一个属性 `detail`，该属性可以具有以下值之一：`nextMonth`、`previousMonth`、`nextYear`、`previousYear`。

最后，对广播该事件的组件实例调用 `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 510 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，当在名为 `myDC` 的 `DateChooser` 实例上按下月份按钮时，它将向 “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 `scroll` 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 `myDC`。最后一行从 `myDC` 调用 `UIEventDispatcher.addEventListener()` 方法，并把 `scroll` 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.scroll = function(eventObj){  
  trace(eventObj.detail);  
}  
myDC.addEventListener("scroll", form);
```

DateChooser.selectableRange

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDC.selectableRange

描述

属性；设置可选的某一天或一定范围的可选日期。用户将无法超出可选范围滚动。此属性的值是由名为 `rangeStart` 和 `rangeEnd` 的两个 `Date` 对象组成的对象。`rangeStart` 和 `rangeEnd` 属性指定可选日期范围的界限。如果只定义 `rangeStart`，则启用 `rangeStart` 后的所有日期。如果只定义 `rangeEnd`，则启用 `rangeEnd` 前的所有日期。默认值未定义。

如果您想要只启用某一天，则可以使用单个 `Date` 对象作为 `selectableRange` 的值。

在定义日期时指定完整日期例如，应指定 `new Date(2003,6,24)`，而不是 `new Date()`。如果没有指定一个完整日期，则该时间返回为 `00:00:00`。

`DateChooser.selectedDate` 的值在超出可选范围时将被设置为未定义。

如果当前月份超出可选范围，`DateChooser.displayedMonth` 和 `DateChooser.displayedYear` 的值将被设置为可选范围中最接近的最后一个月份。例如，如果当前显示的月份是 8 月，并且可选范围是从 2003 年 6 月到 2003 年 7 月，则显示的月份将更改为 2003 年 7 月。

范例

以下范例将可选范围定义为 5 月 7 日和 6 月 7 日之间的日期（包括这两个日期）：

```
myDC.selectableRange = {rangeStart:new Date(2001, 4, 7), rangeEnd:new Date(2003, 5, 7)};
```

以下范例将可选范围定义为 5 月 7 日之后的日期（包括 5 月 7 日）：

```
myDC.selectableRange = {rangeStart:new Date(2003, 4, 7)};
```

以下范例将可选范围定义为 6 月 7 日之前的日期（包括 6 月 7 日）：

```
myDC.selectableRange = {rangeEnd:new Date(2003, 5, 7)};
```

以下范例将可选日期定义为仅限 6 月 7 日：

```
myDC.selectableRange = new Date(2003, 5, 7);
```

DateChooser.selectedDate

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDC.selectedDate

描述

属性；一个 Date 对象，如果该值属于 selectableRange 属性的值的范围内，则指明所选日期。默认值未定义。

selectedDate 属性不能设置为处于 disabledRange 内、超出 selectableRange 或与某个已禁用的日期相同。如果将 selectedDate 属性设置为以前的某个日期，则值将为未定义。

范例

以下范例将所选日期设置为 6 月 7 日：

```
myDC.selectedDate = new Date(2003, 5, 7);
```

DateChooser.showToday

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDC.showToday
```

描述

属性；此属性确定是否突出显示当前日期。默认值为 true。

范例

以下范例关闭对今天日期的突出显示：

```
myDC.showToday = false;
```

DateField 组件（仅限 Flash Professional）

DateField 组件是一个不可选择的文本字段，它显示右边带有日历图标 的日期。如果未选定日期，则该文本字段为空白，并且当前日期的月份显示在日期选择器中。当用户在日期字段边框内的任意位置单击时，将会弹出一个日期选择器，并显示选定日期所在月份内的日期。当日期选择器打开时，用户可以使用月份滚动按钮在月份和年份之间来回滚动，并选择一个日期。选定了日期时，日期选择器将关闭。

DateField 的实时预览不会反映创作时属性检查器或“组件检查器”面板指明的值，因为该组件是一个弹出组件，在创作时是不可见的。

使用 DateField 组件（仅限 Flash Professional）

DateField 组件可用于您想让用户选择日期的任何场合。例如，您可以将 DateField 组件用于酒店预订系统中，其中某些日期是可选择的，其他日期则是禁用的。您也可以在用户选择日期时显示当前事件（如表演或会议）的应用程序中使用 DateField 组件。

DateField 参数

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 DateField 组件设置的创作参数：

monthNames 设置在日历的标题行中显示的月份名称。该值是一个数组，并且默认值是 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

dayNames 设置一周中各天的名称。该值是一个数组，并且默认值是 ["S", "M", "T", "W", "T", "F", "S"]。

firstDayOfWeek 指明一周中的哪一天（其值为 0-6，0 是 dayNames 数组的第一个元素）显示在 DateChooser 的第 1 列中。此属性更改“日”列的显示顺序。

默认值为 0（即“S”）。

disabledDays 指明一周中禁用的各天。该参数是一个数组，并且最多具有 7 个值。默认值为 []（空数组）。

showToday 指明是否突出显示今天的日期。默认值为 true。

您可以编写动作脚本，以便使用其属性、方法和事件来控制 DateField 组件的这些和其他选项。有关详细信息，请参阅第 231 页的“DateField 类（仅限 Flash Professional）”。

创建具有 DateField 组件的应用程序

以下过程解释了如何在创作时将 DateField 组件添加到应用程序。在此范例中，DateField 组件使用户能够从航空公司预订系统中选择一个日期。当前日期之前的所有日期必须被禁用。12 月中一个 15 天的范围也必须被禁用，以便创建一个假日关闭期。同时，由于某些航班在星期一不可用，因此必须为这些航班禁用所有星期一。

要创建具有 DateField 组件的应用程序，请执行以下操作：

- 1 在“组件”面板中双击 DateField 组件，将其添加到舞台。
- 2 在属性检查器中，输入实例名称 **flightCalendar**。
- 3 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以设置可选日期的范围：

```
flightCalendar.selectableRange = {rangeStart:new Date(2001, 9, 1),  
    rangeEnd:new Date(2003, 11, 1)};
```

此代码将一个值分配给动作脚本对象中的 selectableRange 属性，该属性包含两个 Date 对象，这两个对象分别具有变量名称 rangeStart 和 rangeEnd。这样，此代码就定义了用户可以在其内选择日期的范围的上限和下限。

- 4 在“动作”面板中，在时间轴的第一帧上输入以下代码以设置禁用日期的范围，一个范围处于 12 月，另一个代表当前日期之前的所有日期：

```
flightCalendar.disabledRanges = [{rangeStart:new Date(2003, 11, 15),  
    rangeEnd:new Date(2003, 11, 31)}, {rangeEnd:new Date(2003, 6, 16)}];
```

- 5 在“动作”面板中，在时间轴的第 1 帧上输入以下代码，以禁用各星期一：

```
flightCalendar.disabledDays=[1];
```

- 6 “控制” > “测试影片”。

自定义 DateField 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平方向上改变 DateField 组件的形状。在创作时，在舞台上选择组件并使用“任意变形”工具或任何“修改”>“变形”命令。在运行时，使用 `setSize()` 方法（请参阅 `UIObject.setSize()`）。设置宽度不会改变 DateField 组件内日期选择器的尺寸。但是，您可以使用 `pullDown` 属性来访问 DateChooser 组件并设置其尺寸。

对 DateField 组件使用样式

您可以设置样式属性来更改日期字段实例的外观。如果样式属性的名称以“Color”结尾，则它是颜色样式属性，并且行为与非颜色样式属性不同。有关详细信息，请参阅第 25 页的“使用样式自定义组件的颜色和文本”。

DateField 组件支持下列光晕样式：

样式	描述
<code>themeColor</code>	用于变换图象和所选日期的发亮的颜色。这是唯一不继承样式值的颜色样式。可能的值包括“haloGreen”、“haloBlue”和“haloOrange”。
<code>color</code>	组件标签的文本。
<code>disabledColor</code>	禁用的文本颜色。
<code>fontFamily</code>	文本的字体名称。
<code>fontSize</code>	字体的磅值。
<code>fontStyle</code>	字体样式：“normal”或“italic”。
<code>fontWeight</code>	字体粗细：“normal”或“bold”。
<code>textDecoration</code>	文本修饰：“none”或“underline”。

对 DateField 组件使用外观

DateField 组件使用外观来表示弹出图标的可视状态。要在创作时设置弹出图标的外观，请在其中一个主题 FLA 文件的库中修改 Flash UI Components 2/Themes/MMDefault/DateField Elements skins states 文件夹中的外观元件。有关详细信息，请参阅第 33 页的“关于设置组件外观”。

在此组件中，只有弹出图标按钮才能被设置外观。DateField 组件使用以下外观属性动态地设置弹出图标的外观：

属性	描述
<code>openDateUp</code>	弹出图标的弹起状态。
<code>openDateDown</code>	弹出图标的按下状态。
<code>openDateOver</code>	弹出图标的滑过状态。
<code>openDateDisabled</code>	弹出图标的禁用状态。

DateField 类（仅限 Flash Professional）

继承 [UIObject](#) > [UIComponent](#) > [ComboBase](#) > [DateField](#)

动作脚本类名称 `mx.controls.DateField`

`DateField` 类的属性使您能够访问选定的日期以及显示的月份和年份。您也可以设置日和月的名称，指明禁用的日期和可选的日期，设置一周中的第一天，以及指明当前日期是否应突出显示。

使用动作脚本设置 `DateField` 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.DateField.version);
```

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

DateField 类的方法摘要

方法	描述
DateField.close()	关闭弹出日期选择器子组件。
DateField.open()	打开弹出日期选择器子组件。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有方法。

DateField 类的属性摘要

属性	描述
DateField.dateFormatter	对要在文本字段中显示的日期进行格式化的函数。
DateField.dayNames	一个数组，指明一周中各天的名称。
DateField.disabledDays	一个数组，指明为日期选择器中所有适用日期禁用的一周中的各天。
DateField.disabledRanges	禁用日期的范围或单个禁用的日期。
DateField.displayedMonth	一个数字，指明 <code>monthNames</code> 数组中要在日期选择器中显示的元素。
DateField.displayedYear	一个数字，指明要显示的年份。
DateField.firstDayOfWeek	一个数字，指明 <code>dayNames</code> 数组中要在日期选择器的第一列中显示的元素。
DateField.monthNames	一个字符串数组，指明月份名称。
DateField.pullDown	指向 <code>DateChooser</code> 子组件的引用。该属性为只读。
DateField.selectableRange	单个可选日期或可选日期的范围。
DateField.selectedDate	一个 <code>Date</code> 对象，指明可选日期。
DateField.showToday	一个布尔值，指明是否突出显示当前日期。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有属性。

DateField 类的事件摘要

事件	描述
<code>DateField.change</code>	在选择一个日期时广播。
<code>DateField.close</code>	在日期选择器子组件关闭时广播。
<code>DateField.open</code>	在日期选择器子组件打开时广播。
<code>DateField.scroll</code>	在按下月份按钮时广播。

继承 `UIObject` 和 `UIComponent` 中的所有事件。

DateField.change

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
on(change){  
    ...  
}
```

用法 2：

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

描述

事件；当选择日期时，向所有已注册的侦听器广播。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到一个 `DateField` 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，以下代码附加到日期字段 `myDF` 上，它将 “_level0.myDF” 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 (*chooserInstance*) 调度一个事件（在本例中为 `change`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，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 510 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 myDF 的日期字段发生更改时向“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 myDF。从事件对象的 target 属性中可以访问 DateField.selectedDate 属性。最后一行从 myDF 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.change = function(eventObj){
    trace("date selected " + eventObj.target.selectedDate) ;
}
myDF.addEventListener("change", form);
```

DateField.close()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.close()
```

参数

无。

返回

无。

说明

方法；关闭弹出菜单。

示例

以下代码将关闭 myDF 日期字段实例的日期选择器弹出窗口。

```
myDF.close();
```

DateField.close

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
on(close){  
  ...  
}
```

用法 2：

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

描述

事件；当用户在图标外单击或选择一个日期后 `DateChooser` 子组件关闭时向所有注册的侦听器广播。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到一个 `DateField` 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，以下代码附加到日期字段 `myDF` 上，它将 “_level0.myDF” 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 (`myDF`) 发送一个事件（在本例中为 `close`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您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 510 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 `myDF` 内的日期选择器关闭时向 “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 `close` 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 `myDF`。从事件对象的 `target` 属性中可以访问 `selectedDate` 属性。最后一行从 `myDF` 调用 `UIEventDispatcher.addEventListener()` 方法，并把 `close` 事件和 `form` 侦听器对象作为参数传递给该方法，如下所示：

```
form.close = function(eventObj){  
  trace("PullDown Closed" + eventObj.target.selectedDate);  
}
```

```
myDF.addEventListener("close", form);
```

DateField.dateFormatter

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.dateFormatter
```

描述

属性；对要在文本字段中显示的日期进行格式化的函数。该函数必须接收 Date 对象作为参数，并以要显示的格式返回字符串。

范例

以下范例将设置函数以返回要显示的日期的格式：

```
myDF.dateFormatter = function(d:Date){  
    return d.getFullYear()+" / "+(d.getMonth()+1)+" / "+d.getDate();  
};
```

DateField.dayNames

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.dayNames
```

描述

属性；包含一周中各天的名称的数组。星期日是第一天（索引位置为 0），其他各天的名称按顺序跟在后面。默认值为 ["S", "M", "T", "W", "T", "F", "S"]。

范例

以下范例将一周的第 5 天（星期四）的值从 “T” 更改为 “R”：

```
myDF.dayNames[4] = "R";
```

DateField.disabledDays

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.disabledDays
```

描述

属性；指明一周中被禁用的各天的数组。一个月中属于指定日期范围内的所有日期都被禁用。此数组的元素可以具有介于 0（星期日）和 6（星期六）之间的值。默认值为 []（空数组）。

范例

以下范例将禁用星期日和星期六，以使用户只能选择每周工作日：

```
myDF.disabledDays = [0, 6];
```

DateField.disabledRanges

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.disabledRanges
```

描述

属性；禁用某一天或一个范围中的各天。此属性是对象数组。该数组中的每个对象必须或者是指定要禁用的某一天的 Date 对象，或者是包含 rangeStart 和 / 或 rangeEnd 属性（这两个属性的值必须是 Date 对象）的对象。rangeStart 和 rangeEnd 属性描述日期范围的界限。如果这两个属性中的任何一个属性被省略，则范围在该方向上无限制。

disabledRanges 的默认值未定义。

在为 disabledRanges 属性定义日期时应指定一个完整日期。例如，应指定 new Date(2003,6,24)，而不是 new Date()。如果没有指定一个完整日期，则该时间返回为 00:00:00。

范例

以下范例定义一个数组，该数组具有 rangeStart 和 rangeEnd Date 对象，禁用 5 月 7 日和 6 月 7 日之间的日期：

```
myDF.disabledRanges = [ {rangeStart:new Date(2003, 4, 7), rangeEnd:new  
    Date(2003, 5, 7)}];
```

以下范例禁用 11 月 7 日之后的所有日期：

```
myDF.disabledRanges = [ {rangeStart:new Date(2003, 10, 7)}];
```

以下范例禁用 10 月 7 日之前的所有日期：

```
myDF.disabledRanges = [ {rangeEnd:new Date(2002, 9, 7)}];
```

以下范例仅禁用 11 月 7 日：

```
myDF.disabledRanges = [ new Date(2003, 11, 7) ];
```

DateField.displayedMonth

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDF.displayedMonth

描述

属性；指明显示哪一月份的数字。该数字指明 `monthNames` 数组中的元素，其中 0 是第一个月份。默认值是当前日期的月份。

范例

以下范例将显示的月份设置为 December（12 月）：

```
myDF.displayedMonth = 11;
```

另请参见

[DateField.displayedYear](#)

DateField.displayedYear

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDF.displayedYear

描述

属性；指明显示哪一年份的数字。默认值为当前年份。

范例

以下范例将显示的年份设置为 2010：

```
myDF.displayedYear = 2010;
```

另请参见

[DateField.displayedMonth](#)

DateField.firstDayOfWeek

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.firstDayOfWeek
```

描述

属性；一个数值，指明一周中的哪一天（0-6，0 是 `dayNames` 数组的第一个元素）显示在 `DateField` 组件的第一列中。更改该属性将更改日列的顺序，但对 `dayNames` 属性的顺序没有影响。默认值为 0（星期日）。

范例

以下范例将一周的第一天设置为 Monday（星期一）：

```
myDF.firstDayOfWeek = 1;
```

另请参见

[DateField.dayNames](#)

DateField.monthNames

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.monthNames
```

描述

属性；一个字符串数组，在 `DateField` 组件的顶部指明月份名称。默认值为 ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]。

范例

以下范例将设置 `myDF` 实例的月份名称：

```
myDF.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",  
"Sept", "Oct", "Nov", "Dec"];
```

DateField.open()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.open()
```

参数

无。

返回

无。

说明

方法；打开弹出 DateChooser 子组件。

示例

以下代码将打开 df 实例的弹出日期选择器：

```
df.open();
```

DateField.open

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
on(open){  
    ...  
}
```

用法 2：

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

描述

事件；当用户单击图标后日期选择器子组件打开时向所有注册的侦听器广播。

第一个用法范例使用 on() 处理函数，并且必须直接附加到一个 DateField 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，以下代码附加到日期字段 myDF 上，它将 “_level0.myDF” 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 (myDF) 发送一个事件（在本例中为 open），而该事件由您创建的侦听器对象 (listenerObject) 上的函数（也称作处理函数）处理。您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 510 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 myDF 的步进器打开时向“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 open 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 myDF。从事件对象的 target 属性中可以访问 DateField.selectedDate 属性。最后一行从 myDF 调用 UIEventDispatcher.addEventListener() 方法，并把 open 事件和 form 侦听器对象作为参数传递给该方法，如下所示：

```
form.open = function(eventObj){
    trace("Pop-up opened and date selected is " + eventObj.target.selectedDate)
};
myDF.addEventListener("open", form);
```

DateField.pullDown

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.pullDown
```

描述

属性（只读）；对 DateField 组件包含的 DateChooser 组件的引用。当用户单击 DateField 组件时，DateChooser 子组件将被实例化。但是，如果在用户单击该组件之前引用了 pullDown 属性，则 DateChooser 将被实例化并随后被隐藏。

范例

以下范例将 DateChooser 子组件的可见性设置为 false，然后将 DateChooser 子组件的大小设置为 300 像素高，300 像素宽：

```
myDF.pullDown._visible = false;
myDF.pullDown.setSize(300,300);
```

DateField.scroll

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
on(scroll){  
    ...  
}
```

用法 2：

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

描述

事件；在按下月份按钮时广播到所有注册的侦听器。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到一个 `DateField` 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，以下代码附加到日期字段 `myDF` 上，它将 “_level0.myDF” 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 (`myDF`) 发送一个事件（在本例中为 `scroll`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您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。`scroll` 事件的事件对象还有另一个属性 `detail`，该属性可以具有以下值之一：`nextMonth`、`previousMonth`、`nextYear`、`previousYear`。

最后，对广播该事件的组件实例调用 `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 510 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，当在名为 `myDF` 的 `DateField` 实例上按下月份按钮时，它将向 “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 `scroll` 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 `myDF`。最后一行从 `myDateField` 调用 `UIEventDispatcher.addEventListener()` 方法，并把 `scroll` 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.scroll = function(eventObj){  
    trace(eventObj.detail);  
}  
myDF.addEventListener("scroll", form);
```

DateField.selectableRange

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDF.selectableRange

描述

属性；设置可选的某一天或一定范围的可选日期。此属性的值是由名为 `rangeStart` 和 `rangeEnd` 的两个 `Date` 对象组成的对象。`rangeStart` 和 `rangeEnd` 属性指定可选日期范围的界限。如果只定义 `rangeStart`，则启用 `rangeStart` 后的所有日期。如果只定义 `rangeEnd`，则启用 `rangeEnd` 前的所有日期。默认值未定义。

如果您想要只启用某一天，则可以使用单个 `Date` 对象作为 `selectableRange` 的值。

在定义日期时指定完整日期例如，应指定 `new Date(2003,6,24)`，而不是 `new Date()`。如果没有指定一个完整日期，则该时间返回为 `00:00:00`。

`DateField.selectedDate` 的值在超出可选范围时将被设置为未定义。

如果当前月份超出可选范围，`DateField.displayedMonth` 和 `DateField.displayedYear` 的值将被设置为可选范围中最接近的最后一个月份。例如，如果当前显示的月份是 8 月，并且可选范围是从 2003 年 6 月到 2003 年 7 月，则显示的月份将更改为 2003 年 7 月。

范例

以下范例将可选范围定义为 5 月 7 日和 6 月 7 日之间的日期（包括这两个日期）：

```
myDF.selectableRange = {rangeStart:new Date(2001, 4, 7), rangeEnd:new Date(2003, 5, 7)};
```

以下范例将可选范围定义为 5 月 7 日之后的日期（包括 5 月 7 日）：

```
myDF.selectableRange = {rangeStart:new Date(2003, 4, 7)};
```

以下范例将可选范围定义为 6 月 7 日之前的日期（包括 6 月 7 日）：

```
myDF.selectableRange = {rangeEnd:new Date(2003, 5, 7)};
```

以下范例将可选日期定义为仅限 6 月 7 日：

```
myDF.selectableRange = new Date(2003, 5, 7);
```

DateField.selectedDate

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myDF.selectedDate

描述

属性；一个 `Date` 对象，如果该值属于 `selectableRange` 属性的值的范围内，则指明所选日期。默认值未定义。

范例

以下范例将所选日期设置为 6 月 7 日：

```
myDF.selectedDate = new Date(2003, 5, 7);
```

DateField.showToday

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myDF.showToday
```

描述

属性；此属性确定是否突出显示当前日期。默认值为 `true`。

范例

以下范例关闭对今天日期的突出显示：

```
myDF.showToday = false;
```

DepthManager 类

动作脚本类名称 `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
<code>DepthManager.createChildAtDepth()</code>	在指定深度处创建指定元素的子级。
<code>DepthManager.createClassChildAtDepth()</code>	在该指定深度处创建指定类的对象。
<code>DepthManager.createClassObjectAtDepth()</code>	在特殊最深剪辑中的指定深度处创建指定类的实例。
<code>DepthManager.createObjectAtDepth()</code>	在最深剪辑中的指定深度处创建一个对象。
<code>DepthManager.setDepthAbove()</code>	将深度设置到指定实例之上。
<code>DepthManager.setDepthBelow()</code>	将深度设置到指定实例之下。
<code>DepthManager.setDepthTo()</code>	将深度设置为最深剪辑中的指定实例。

DepthManager.createChildAtDepth()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

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

参数

linkageName 一个链接标识符。此参数是一个字符串。

depthFlag 它是下列值之一：DepthManager.kTop、DepthManager.kBottom、DepthManager.kTopmost、DepthManager.kNotopmost。所有深度标记都是 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

指向所创建的对象引用。

描述

方法；创建一个由 *linkageName* 参数指定的元素的子实例，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 版本 79。

版本

Flash MX 2004 and 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 版本 79。

版本

Flash MX 2004 and 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.createClassObjectAtDepth()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004 and Flash MX Professional 2004。

用法

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

参数

linkageName 一个链接标识符。

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 版本 79。

版本

Flash MX 2004 and Flash MX Professional 2004。

用法

```
movieClipInstance.setDepthAbove(instance)
```

参数

instance 一个实例名称。

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 版本 79。

版本

Flash MX 2004 and Flash MX Professional 2004。

用法

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

参数

instance 一个实例名称。

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 版本 79。

版本

Flash MX 2004 and 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《动作脚本参考指南》帮助中的“确定下一个最高可深度”。

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

FocusManager 不会自动给组件指定焦点。如果想要某个组件在加载应用程序时具有焦点，必须编写一个对组件调用 `FocusManager.setFocus()` 的脚本。

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;
```

您还可以使用“辅助功能”面板来分配 Tab 键索引值。

如果舞台上没有任何对象具有 Tab 键索引值，则 FocusManager 使用 Z 顺序。Z 顺序最初按组件拖到舞台上的顺序设置；不过，您也可以使用“修改”/“排列”/“移至顶层”或“移至底层”命令来确定最终的 Z 顺序。

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(evt){
    trace(evt.target + " was clicked");
}
okButton.addEventListener("click", lo);
```

这段代码设置了 Tab 键顺序，并指定在用户按 Enter 键 (Windows) 或 Return 键 (Macintosh) 时用来接收 click 事件的默认按钮。

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`

动作脚本类名称 `mx.managers.FocusManager`

FocusManager 类的方法摘要

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

FocusManager 类的属性摘要

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

FocusManager.defaultPushButton

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004 and 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 版本 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 版本 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 版本 79。

版本

Flash MX 2004 and 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 版本 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 版本 79。

版本

Flash MX 2004 and 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 版本 79。

版本

Flash MX 2004 and 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.setFocus\(\)](#)

Form 类（仅限 Flash Professional）

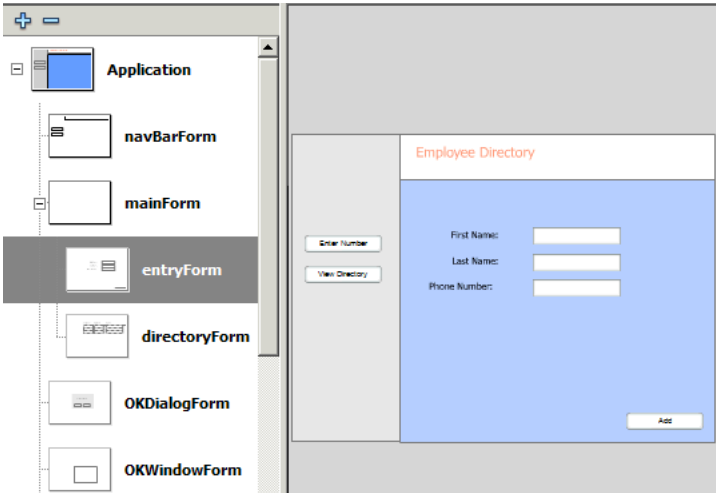
继承 UIObject > UIComponent > View > Loader > Screen > Form

动作脚本类名称 mx.screens.Form

Form 类提供您在 Flash MX Professional 2004 的“屏幕大纲”窗格中所创建窗体的运行时行为。有关使用屏幕的概述，请参阅“使用 Flash”帮助中的“使用屏幕（仅限 Flash Professional）”。

使用 Form 类（仅限 Flash Professional）

窗体用作图形对象（例如，应用程序中的用户界面元素）和应用程序状态的容器。可以使用“屏幕大纲”窗格使您正在创建的应用程序的不同状态可视化，其中每个窗体都是不同的应用程序状态。例如，以下图示显示了使用窗体设计的应用程序范例的“屏幕大纲”窗格。



窗体应用程序范例的“屏幕大纲”视图

此图示显示名为“Employee Directory”的应用程序范例的大纲，其中包含一些窗体。名为“entryForm”的窗体（在上面的图示中处于选中状态）包含一些用户界面对象，包括输入文本字段、标签和普通按钮。通过切换此窗体的可见性（使用 [Form.visible](#) 属性），同时也切换其他窗体的可见性，开发人员可以轻松地使用户看到此窗体。

使用“行为”面板（“窗口” > “开发面板” > “行为”），您还可以将行为和控件附加到窗体。有关向屏幕添加过渡和控件的详细信息，请参阅“使用 Flash”帮助中的“通过行为创建在屏幕上使用的控件和过渡（仅限 Flash Professional）”。

因为 Form 类扩展了 Loader 类，所以可以轻松地将外部内容（SWF 或 JPEG）加载到窗体中。例如，窗体的内容可以是单独的 SWF，其本身可能包含窗体。通过此方法，可以使窗体应用程序模块化，从而能更轻松地维护应用程序，同时还降低了最初的下载时间。有关详细信息，请参阅第 412 页的“将外部内容加载到屏幕（仅限 Flash Professional）”。

窗体对象参数

以下列出了一些创作参数，您可以在属性检查器或“组件检查器”面板中为每个窗体对象实例设置这些参数：

autoload 指示 contentPath 参数指定的内容是应该自动加载 (true)，还是应该等到调用 `Loader.load()` 方法时再进行加载 (false)。默认值为 true。

contentPath 指定窗体的内容。该参数可以是一个影片剪辑的链接标识符，也可以是要加载到幻灯片中的 SWF 或 JPG 文件的绝对或相对 URL。默认情况下，加载的内容会进行剪辑以适合幻灯片的大小。

visible 指定窗体在第一次加载时是 (true) 否 (false) 可见。

Form 类的方法摘要

方法	描述
<code>Form.getChildForm()</code>	返回指定索引处的子窗体。

继承 `UIObject`、`UIComponent`、`View`、`Loader` 组件和 `Screen` 类（仅限 Flash Professional）的所有方法。

Form 类的属性摘要

属性	描述
<code>Form.currentFocusedForm</code>	返回包含全局当前焦点的“最低叶”窗体。
<code>Form.indexInParentForm</code>	返回此窗体在其父窗体的子窗体列表中的索引（从零开始）。
<code>Form.visible</code>	指定窗体在其父窗体、幻灯片、影片剪辑或 SWF 可见时是否可见。
<code>Form.numChildForms</code>	返回此窗体包含的子窗体的数量。
<code>Form.parentIsForm</code>	返回此窗体的父对象是否也是窗体。
<code>Form.rootForm</code>	返回包含此窗体的窗体树（或子树）的根。

继承 `UIObject`、`UIComponent`、`View`、`Loader` 组件和 `Screen` 类（仅限 Flash Professional）的所有属性。

Form.currentFocusedForm

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mx.screens.Form.currentFocusedForm
```

说明

属性（只读）；返回包含全局当前焦点的窗体对象。实际的焦点可能在窗体自身上，或者在影片剪辑、文本对象或者该窗体内的组件上。如果没有当前焦点，则可以为 `null`。

示例

以下代码附加到一个按钮（未显示），并显示具有当前焦点的窗体的名称。

```
trace("The form with the current focus is:" +  
      mx.screens.Form.currentFocusedForm);
```

Form.getChildForm()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myForm.getChildForm(childIndex)
```

参数

childIndex 一个数字，指示要返回的子窗体的索引（从零开始）。

返回

一个窗体对象。

说明

方法；返回 *myForm* 的子窗体（其索引为 *childIndex*）。

示例

下面的范例在“输出”面板中显示了属于名为 `Application` 的窗体根对象的所有子窗体对象名称。

```
for (var i:Number = 0; i < _root.Application.numChildForms; i++) {  
    var childForm:mx.screens.Form = _root.Application.getChildForm(i);  
    trace(childForm._name);  
}
```

另请参见

[Form.numChildForms](#)

Form.indexInParentForm

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myForm.indexInParentForm

说明

属性（只读）；包含 *myForm* 在其父对象的子窗体列表中的索引（从零开始）。如果 *myForm* 的父对象是屏幕而不是窗体（例如，是幻灯片），则 *indexInParentForm* 始终为 0。

示例

```
var myIndex:Number = myForm.indexInParent;  
if (myForm == myForm._parent.getChildForm(myIndex)) {  
    trace("I'm where I should be");  
}
```

另请参见

[Form.getChildForm\(\)](#)

Form.numChildForms

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myForm.numChildForms

说明

属性（只读）；返回 *myForm* 包含的子窗体数量。此属性不包括 *myForm* 所包含的任何幻灯片，只包括窗体。

示例

以下代码循环访问 *myForm* 所包含的所有子窗体，并在“输出”面板中显示它们的名称。

```
var howManyKids:Number = myForm.numChildForms;  
for(i=0; i<howManyKids; i++) {  
    var childForm = myForm.getChildForm(i);  
    trace(childForm._name);  
}
```

另请参见

[Form.getChildForm\(\)](#)

Form.parentIsForm

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myForm.parentIsForm

说明

属性（只读）：返回一个布尔值（true 或 false），指示所指定窗体的父对象是 (true) 否 (false) 也是窗体。如果为 false，则 *myForm* 位于其窗体层次结构的根位置。

示例

```
if (myForm.parentIsForm) {
    trace("I have "+myForm._parent.numChildScreens+" sibling screens");
} else {
    trace("I am the root form and have no siblings");
}
```

Form.rootForm

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myForm.rootForm

说明

属性（只读）；返回位于窗体层次结构顶层的包含 *myForm* 的窗体。如果 *myForm* 包含于非窗体的对象（即幻灯片）中，则此属性会返回 *myForm*。

示例

在下面的范例中，将一个对 *myForm* 的根窗体的引用放入到名为 *root* 的变量中。如果指定给 *root* 的值引用 *myForm*，则 *myForm* 位于其窗体树的顶层。

```
var root:my.screens.Form = myForm.rootForm;
if(rootForm == myForm) {
    trace("myForm is the top form in its tree");
}
```

Form.visible

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myForm.visible

说明

属性；确定 *myForm* 在其父窗体、幻灯片、影片剪辑或影片可见时是否可见。也可以使用 Flash 创作环境中的属性检查器设置此属性。

此属性设置为 `true` 时，*myForm* 会接收到 `reveal` 事件；当设置为 `false` 时，*myForm* 会接收到 `hide` 事件。您可以将在窗体接收到上述一种事件时执行的过渡附加到窗体上。有关向屏幕添加过渡的详细信息，请参阅“使用 Flash”帮助中的“通过行为创建在屏幕上使用的控件和过渡（仅限 Flash Professional）”。

示例

以下代码附加到 Button 组件的一个实例，它将包含按钮的窗体的 `visible` 属性设置为 `false`。

```
on(click){  
    _parent.visible = true;  
}
```

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。实时预览不支持 `autoSize` 参数。

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

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

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”。

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，标签位于购物车应用程序中某个带有日期的组合框的旁边。

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 261 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 25 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

动作脚本类名称 mx.controls.Label

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

使用“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 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](#) 的所有属性。

Label 类的事件摘要

继承 [UIObject](#) 的所有事件。

Label.autoSize

可用性

Flash Player 6 版本 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：标签的右边和底边会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。

Label.html

可用性

Flash Player 6 版本 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) 可采用 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 `` 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 `` 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 版本 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。

按键	描述
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 248 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”。

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，用户访问电子商务 Web 站点时需要选择想要购买的项目。一共有 30 个项目，用户在列表中上下滚动，并通过单击选择一项。

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

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 422 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
<code>alternatingRowColors</code>	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。
<code>backgroundColor</code>	列表的背景颜色。该样式在类样式声明 <code>ScrollSelectList</code> 上进行定义。
<code>borderColor</code>	三维边框的黑色部分或二维边框的彩色部分。
<code>borderStyle</code>	边框样式。可能的值包括：“none”、“solid”、“inset”和“outset”。该样式在类样式声明 <code>ScrollSelectList</code> 上进行定义。
<code>defaultIcon</code>	用于列表行的默认图标的名称。默认值未定义。
<code>rollOverColor</code>	滑过的行的颜色。
<code>selectionColor</code>	所选行的颜色。

样式	描述
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。

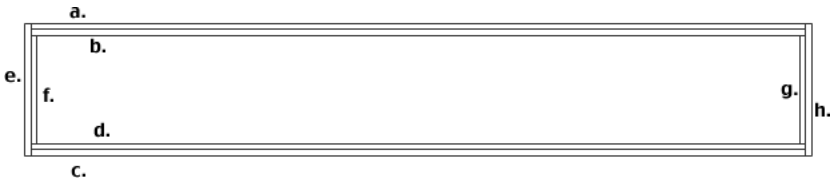
List 组件也使用 Label 组件（请参阅第 261 页的“对 Label 组件使用样式”）、ScrollPane 组件（请参阅第 422 页的“ScrollPane 组件”）和 RectBorder 的样式属性。

在 List 组件中使用外观

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 422 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`

动作脚本类名称 `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，它通常有一个显示出来的 `label` 属性和一个 `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` 类实现 `CellRenderer` 接口，该接口是属性和方法的集合，通过这些属性和方法，列表可以操作各行并将数据和状态信息（例如，大小、已被选中等等）发送到行以供显示。

数据提供程序是列表中的项列表的数据模型。同一帧中作为列表的任何数组都会自动得到一些方法，这些方法允许您操作数据并将更改广播给多个视图。您可以创建一个 `Array` 实例或者从服务器上获取一个实例，然后将它用作多个 `List`、`ComboBox`、`DataGrid` 等的的数据模型。`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
<code>List.addItem()</code>	向列表的结尾添加项目。
<code>List.addItemAt()</code>	将项目添加到指定索引处的列表。
<code>List.getItemAt()</code>	返回指定索引处的项目。
<code>List.removeAll()</code>	删除列表中的所有项目。
<code>List.removeItemAt()</code>	删除指定索引处的项目。
<code>List.replaceItemAt()</code>	用其他项目替换指定索引处的项目。
<code>List.setPropertiesAt()</code>	将指定的属性应用到指定的项目。
<code>List.sortItems()</code>	按照指定的比较函数对列表中的项目进行排序。
<code>List.sortItemsBy()</code>	按照指定的属性对列表中的项目进行排序。

继承 `UIObject` 和 `UIComponent` 中的所有方法。

List 类的属性摘要

属性	描述
List.cellRenderer	指定要使用的类或元件以显示列表的每一行。
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数据项。
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数据项。
List.labelFunction	一个函数，它确定各个项目的哪些数据项要用作标签文本。
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	指明列表中是 (true) 否 (false) 允许多选。
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](#) 和 [UIComponent](#) 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](#) 和 [UIComponent](#) 的所有事件。

List.addItem()

可用性

Flash Player 6 版本 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 一个项目对象，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。要试用这行代码，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 版本 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 一个项目对象，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

以下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 版本 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。此属性必须是类对象引用，或者是元件链接标识符。用于此属性的任何类都必须实现第 70 页的“CellRenderer API”。

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 版本 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 `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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 79。

版本

Flash MX 2004。

用法

*list*instance.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 167 页的“DataProvider API”](#)。

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 版本 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 版本 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 版本 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 版本 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 版本 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*（该参数指示所呈现的项目），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 版本 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 `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 `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*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 `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 `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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 版本 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*（该参数指示所呈现的项目），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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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` 参数指定的索引位置的项目内容。

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 版本 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 版本 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 版本 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 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 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），而该事件由您创建的侦听器对象 (listenerObject) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (eventObject) 传递到侦听器对象方法。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 510 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 版本 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 版本 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，并且会选中所指定的项目。

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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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也包括整数，则首先列出整数项目。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 版本 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 版本 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 248 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](#)”。

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。

contentPath 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 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 文件存储在同一个文件夹中，并且其文件名不能包含文件夹或磁盘驱动器说明。默认值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/corpllogo.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	字母
<code>borderColor</code>	a
<code>highlightColor</code>	b
<code>borderColor</code>	c
<code>shadowColor</code>	d
<code>borderCapColor</code>	e
<code>shadowCapColor</code>	f
<code>shadowCapColor</code>	g
<code>borderCapColor</code>	h

这些样式属性设置边框上的下列位置：



Loader 类

继承 UIObject > UIComponent > View > Loader

动作脚本类名称 mx.controls.Loader

Loader 类的属性允许您设置要加载的内容并在运行时监视它的加载进程。

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
<code>Loader.load()</code>	加载由 <code>contentPath</code> 属性指定的内容。

继承 `UIObject` 和 `UIComponent` 中的所有方法。

Loader 类的属性摘要

属性	描述
<code>Loader.autoLoad</code>	一个布尔值，指明内容是自动加载 (<code>true</code>) 还是只在您调用 <code>Loader.load()</code> 时才进行加载 (<code>false</code>)。
<code>Loader.bytesLoaded</code>	只读属性，指明已经加载的字节数。
<code>Loader.bytesTotal</code>	指明内容中的总字节数的只读属性。
<code>Loader.content</code>	引用由 <code>Loader.contentPath</code> 属性指定的内容。该属性为只读。
<code>Loader.contentPath</code>	一个字符串，它指明要加载的内容的 URL。
<code>Loader.percentLoaded</code>	一个数字，它指明已加载内容的百分比。该属性为只读。
<code>Loader.scaleContent</code>	一个布尔值，它指明是内容会进行缩放以适合加载器 (<code>true</code>)，还是加载器会进行缩放以适合内容 (<code>false</code>)。

继承 `UIObject` 和 `UIComponent` 的所有属性。

Loader 类的事件摘要

事件	描述
<code>Loader.complete</code>	当内容加载完成时触发。
<code>Loader.progress</code>	在内容加载过程中触发。

继承 `UIObject` 和 `UIComponent` 的所有属性

Loader.autoLoad

可用性

Flash Player 6 版本 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()` 时才加载内容 (`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 版本 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 版本 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 版本 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 `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 `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*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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);
loader.contentPath = "logo.swf";
```

Loader.content

可用性

Flash Player 6 版本 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 版本 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 Flash Player，或者在 Flash 中使用测试影片模式，必须将所有 SWF 文件存储在同一个文件夹中，并且其文件名不能包含文件夹或磁盘驱动器信息。

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 版本 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 版本 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 版本 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 autoloader 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 `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 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*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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.stretchContent = false;
```

媒体组件（仅限 Flash Professional）

媒体流组件能够很方便地将媒体流加入到 Flash 演示文稿中。这些组件可让您以多种方法显示媒体。

下面是三种可用的媒体组件：

- **MediaDisplay** 组件使媒体可以流入到 Flash 内容中，而不需要支持的用户界面。此组件可以用于处理视频和音频数据。当 **MediaDisplay** 组件由其自身使用时，应用程序的用户将无法控制媒体。
- **MediaController** 组件通过提供一个用户界面来补充 **MediaDisplay** 组件，该界面使用标准的控件（播放、暂停等）控制媒体播放。媒体不会加载到 **MediaController** 或由 **MediaController** 播放；它仅用于控制 **MediaPlayback** 或 **MediaDisplay** 实例中的播放。**MediaController** 组件起到“抽屉”的作用，它会在鼠标位于组件上方时显示播放控件的内容。
- **MediaPlayback** 组件是 **MediaDisplay** 和 **MediaController** 组件的结合；它提供对媒体内容进行流式处理的方法。

请记住关于媒体组件的以下要点：

- 媒体组件要求 Flash Player 7 或更高版本。
- 媒体组件不支持向前扫描和向后扫描功能。不过，可以通过移动播放滑块获得此功能。
- 只有组件大小和控制策略会在“实时预览”中反映出来。
- 媒体组件不支持辅助功能。

与媒体组件进行交互（仅限 Flash Professional）

流式 **MediaPlayback** 和 **MediaController** 组件会响应对鼠标和键盘活动。**MediaDisplay** 组件不响应键盘或鼠标事件。下表总结了 **MediaPlayback** 和 **MediaController** 组件在接收到焦点后的动作：

目标	导航	描述
给定控制器的播放控件	鼠标移过	按钮突出显示。
给定控制器的播放控件	单击鼠标左键	用户可以通过给定控制器的播放控件控制音频和视频媒体的播放，方法是单击播放控件以触发其相应的效果。 “暂停” / “播放” 和 “转到开头” / “转到结尾” 按钮遵从标准的按钮行为。按下鼠标按钮时，屏幕上的按钮会突出显示为其被按下状态，当松开鼠标按钮时，屏幕上的按钮会回到未选中的状态。 在播放 FLV 媒体文件时，“转到结尾” 按钮被禁用。
给定控制器的滑块控件	前后移动滑块	播放滑块指示用户在媒体内容中的位置；默认情况，显示手柄会水平移动，以便从开头（左）到结尾（右）指示播放。垂直摆放控件时，播放滑块从下往上移动。当指示器手柄从左向右移动时，它会突出显示上一个显示区，以表明此内容已播放或已选中。在指示器手柄前面的显示区会保持不突出显示状态，直到指示器移过。用户可以拖动指示器手柄，以改变媒体内容的播放位置。如果媒体正在播放，会从鼠标松开的位置点开始自动播放。如果媒体暂停，可以移动并松开滑块，且媒体仍然保持暂停。 还提供了一个音量滑块，可以在水平和垂直布局下从左（静音）向右（最大音量）移动。

目标	导航	描述
播放控制器导航	Tab 键, Shift+Tab 键	将焦点在控制器组件内的按钮之间移动, 获得焦点的元素将突出显示。此导航适用于 “暂停” / “播放”、“转到开头”、“转到结尾”、“静音” 和 “最大音量” 控件。当用户按 tab 键在元素之间移动时, 焦点会从左到右、从上到下移动。按 Shift+Tab 键会将焦点从右到左、从下到上移动。通过 Tab 键接收焦点后, 控件立即将焦点传给 “播放” / “暂停” 按钮。当焦点位于 “最大音量” 按钮上且按下 Tab 键时, 控件会向舞台上 Tab 键索引中的下一个控件提供焦点。
给定的控件按钮	空格键或 Enter/Return 键	选择具有焦点的元素。按下后, 按钮会显示为被其按下状态。松开后, 按钮会回复其具有焦点、鼠标移过的状态。

了解媒体组件（仅限 Flash Professional）

在开始使用媒体组件之前, 最好是了解一下它们是如何工作的。本节提供了有关媒体组件如何进行工作的概述。本节中列出的大多数属性可以通过 “组件检查器” 面板直接设置。请参阅第 304 页的 “使用 “组件检查器” 面板设置媒体组件”。

除了本节稍后讨论的布局属性, 可以为 `MediaDisplay` 和 `MediaPlayback` 组件设置以下属性：

- 媒体类型, 可以设置为 MP3 或 FLV（请参阅 `Media.mediaType` 和 `Media.setMedia()`）。
- 相对或绝对内容路径, 它保存了要进行流式处理的媒体文件（请参阅 `Media.contentPath`）。
- 线索点对象, 以及其名称、时间和播放器属性（请参阅 `Media.addCuePoint()` 和 `Media.cuePoints`）。可以任意设置线索点的名称, 但应将其设置为在使用侦听器和跟踪事件时有意义。在线索点的时间属性值等于它所关联的 `MediaPlayback` 或 `MediaDisplay` 组件的播放头位置值时, 它将广播 `cuePoint` 事件。播放器属性是对其所关联的 `MediaPlayback` 实例的引用。线索点可以随后通过 `Media.removeCuePoint()` 和 `Media.removeAllCuePoints()` 删除。

媒体流组件广播许多相关事件。以下广播事件可以用于设置动画中的其他项目：

- `change` 事件在播放媒体时由 `MediaDisplay` 和 `MediaPlayback` 组件连续广播。（请参阅 `Media.change`。）
- `progress` 事件在加载媒体时由 `MediaDisplay` 和 `MediaPlayback` 组件连续广播。（请参阅 `Media.progress`。）
- `click` 事件在每次单击 “暂停” / “播放” 按钮时由 `MediaController` 和 `MediaPlayback` 组件广播。在这种情况下, 事件对象的 `detail` 属性会提供关于所单击按钮的信息。（请参阅 `Media.click`。）
- `volume` 事件在用户调节音量控件时由 `MediaController` 和 `MediaPlayback` 组件广播。（请参阅 `Media.volume`。）
- `playheadChange` 事件在用户移动播放滑块时由 `MediaController` 和 `MediaPlayback` 组件广播。（请参阅 `Media.playheadChange`。）

`MediaDisplay` 组件与 `MediaController` 组件配合使用。配合使用时, 这两个组件的行为方式类似于 `MediaPlayback` 组件, 但在布局方面更为灵活。因此, 如果在播放媒体时需要灵活的外观和行为, 请使用 `MediaDisplay` 和 `MediaController` 组件。否则, `MediaPlayback` 组件是最佳选择。

了解 MediaDisplay 组件

当将 MediaDisplay 组件放置到舞台上时，所绘制出的该组件没有可见的用户界面。它只是一个容纳和播放媒体的容器。在 MediaDisplay 组件中播放的任何视频媒体的外观受以下属性影响：

- `Media.aspectRatio`
- `Media.autoSize`
- 高度
- 宽度

注意：除非正在播放媒体，否则用户看不到任何内容。

`Media.aspectRatio` 属性优先于其他属性。当 `Media.aspectRatio` 设置为 `true` 时（默认值），此组件将始终在设置了组件大小后重新调整播放媒体的大小，以确保保持媒体的高宽比。

对于 FLV 文件，当 `Media.autoSize` 设置为 `true` 时，要播放的媒体将以其首选大小显示，而与组件的大小无关。这意味着，除非 MediaDisplay 实例大小与该媒体的大小相同，否则，该媒体将溢出实例边界或无法填充实例大小。当 `Media.autoSize` 设置为 `false` 时，将会尽可能地使用实例大小，同时保持高宽比。如果 `Media.autoSize` 和 `Media.aspectRatio` 都设置为 `false`，将使用该组件的准确大小。

注意：由于在播放 MP3 文件时不会显示图像，因此，设置 `Media.autoSize` 将无效。对于 MP3 文件，最小的可用大小是 60 像素高乘以 256 像素宽（在默认的方向中）。

MediaDisplay 组件还支持 `Media.volume` 属性。此属性为 0 至 100 之间的整数，0 为静音，100 为最大音量。默认设置为 75。

了解 MediaController 组件

MediaController 组件的界面取决于它的 `Media.controllerPolicy` 和 `Media.backgroundStyle` 属性。`Media.controllerPolicy` 属性确定媒体控件设置是始终可见、折叠还是仅当鼠标停留在组件的控件部分时可见。折叠时，控制器会绘制一个经修改的进度栏，它是加载栏和播放栏的组合。它在栏底部显示加载的字节进度，并在其上面显示播放头的进度。展开状态绘制增强版的播放栏 / 加载栏，包含以下项目：

- 在左侧指示播放状态（正进行流式处理或暂停）的文本标签，以及在右侧指示播放头位置（以秒为单位）的文本标签。
- 播放头位置指示器
- 滑块，用户可以拖动它在媒体中导航

MediaController 组件还提供了以下项目：

- “播放” / “暂停” 状态按钮
- 由两个按钮组成的组按钮：“转到开头”和“转到结尾”，分别导航到媒体的开头和结尾
- 音量控件，由一个滑块、静音按钮和最大音量按钮组成

MediaController 组件的折叠和展开状态都使用 `Media.backgroundStyle` 属性。此属性确定控制器是绘制铬印染背景（默认值），还是允许从控件后面显示影片背景。

MediaController 组件有一个方向设置 (`Media.horizontal`)，可用于以水平方向（默认值）或垂直方向绘制组件。在水平方向下，播放栏从左到右跟踪播放的媒体。在垂直方向下，播放栏从下到上跟踪媒体。

MediaDisplay 和 MediaController 组件可以通过 `Media.associateDisplay()` 和 `Media.associateController()` 方法互相关联。调用这些方法时，它们使 MediaController 实例可以根据来自 MediaDisplay 实例的事件广播更新前者的控件，并使 MediaDisplay 组件可以响应用户通过 MediaController 进行的设置。

了解 MediaPlayer 组件

MediaPlayer 组件是 MediaController 和 MediaDisplay 控件的组合。两个子组件都包含在 MediaPlayer 内。MediaController 和 MediaDisplay 部分始终会根据 MediaPlayer 组件实例的整体大小进行缩放。

MediaPlayer 组件使用 `Media.controlPlacement` 确定控件的布局。可能的控件位置包括 `top`、`bottom`、`left` 和 `right`，它们指示控件相对于显示将绘制在哪个位置。例如，`right` 值使控件保持垂直并将其放在显示的右侧。

使用媒体组件（仅限 Flash Professional）

随着使用媒体向 Web 用户提供信息这些案例的迅速增多，向用户提供一种对媒体进行流式处理然后加以控制的方法成为了一种普遍的愿望。以下是媒体组件的用法方案范例：

- 显示介绍公司的媒体
- 对影片或影片预览进行流式处理
- 对歌曲或歌曲片段进行流式处理
- 提供媒体形式的学习材料

使用 MediaPlayer 组件

假设必须要为客户开发一个 Web 站点，使 Web 站点用户可以在丰富媒体环境下预览您销售的 DVD 和 CD。下面的范例显示了完成此过程的步骤，并假设您的 Web 站点已做好插入流组件的准备。

创建显示 CD 或 DVD 预览的 Flash 文档：

- 1 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
- 2 打开“组件”面板（“窗口”>“开发面板”>“组件”）并双击 MediaPlayer 组件，这会将该组件的一个实例放置到舞台上。
- 3 选择此 MediaPlayer 组件实例并在属性检查器中输入实例名称 **myMedia**。
- 4 在“组件检查器”面板（“窗口”>“开发面板”>“组件检查器”）中，根据要进行流式处理的媒体类型设置媒体类型（MP3 或 FLV）。
- 5 如果选择 FLV，请在“视频长度”文本框中输入视频的持续时间；请使用 HH:MM:SS 格式。
- 6 在“URL”文本框中输入预览视频的位置。例如，您可以输入 **http://my.web.com/videopreviews/AMovieName.flv**。
- 7 为“自动播放”、“使用首选媒体大小”和“保持高宽比”复选框设置所需选项。
- 8 将控件位置设置到所需的 MediaPlayer 组件一侧。
- 9 向媒体结尾添加一个线索点，它将与侦听器一起使用，用于打开一个通知用户该影片出售的弹出式窗口。将线索点的名称指定为 **cuePointName**，然后设置线索点时间，使其接近剪辑的结尾（在几秒钟内）。要达到此目的，请执行以下步骤：
 - a 双击某个 Window 组件，使其在舞台上显示。
 - b 删除此 Window 组件。这会将一个名为 Window 的项目放置到库中。

- c 创建一个文本框并写入一些文本，以通知用户此影片出售。
 - d 通过选择 “修改” > “转换为元件”，然后指定名称 **mySale_mc**，将此文本框转换为影片剪辑。
 - e 右键单击库中的 mySale_mc 影片剪辑，选择 “链接”，然后选择 “为动作脚本导出” 选项。这会将影片剪辑放置到您的运行时库中。
- 10 将以下动作脚本添加到第 1 帧。此代码会创建一个侦听器，用于打开一个通知用户此影片出售的弹出式窗口。

```
// 导入动态创建弹出式窗口所需的类

import mx.containers.Window;
import mx.managers.PopUpManager;

// 创建一个打开销售弹出式窗口的侦听器对象
var saleListener = new Object();

saleListener.cuePoint = function(evt){

var saleWin = PopUpManager.createPopUp(_root, Window, false,
    {closeButton:true, title:"Movie Sale ", contentPath:"mySale_mc"});

// 放大窗口以便使内容与之相适应

saleWin.setSize(80, 80);
var delSaleWin = new Object();
delSaleWin.click = function(evt){
    saleWin.deletePopUp();
}
saleWin.addEventListener("click", delSaleWin);

}

myMedia.addEventListener("cuePoint", saleListener);
```

使用 MediaDisplay 和 MediaController 组件

假设您决定要在更大程度上控制媒体播放的外观和行为。为此，需要同时使用 MediaDisplay 和 MediaController 来提供所需的效果。下面的范例显示上例的等效步骤，创建一个显示 CD 和 DVD 预览媒体的 Flash 应用程序。

创建显示 CD 或 DVD 预览的 Flash 文档：

- 1 在 Flash 中，选择 “文件” > “新建”，然后选择 “Flash 文档”。
- 2 在 “组件” 面板（“窗口” > “开发面板” > “组件”）中，双击 MediaController 和 MediaDisplay 组件，这会将每个组件的一个实例放置到舞台上。
- 3 选择 MediaDisplay 实例并在属性检查器中输入实例名称 **myDisplay**。
- 4 选择 MediaController 实例并在属性检查器中输入实例名称 **myController**。
- 5 从属性检查器启动 “组件检查器” 面板，并根据要进行流式处理的媒体类型设置媒体类型（MP3 或 FLV）。
- 6 如果选择 FLV，请在 “视频长度” 文本框中输入视频的持续时间，格式为 HH:MM:SS。
- 7 在 “URL” 文本框中输入预览视频的位置。例如，您可以输入 **http://my.web.com/video previews/AMovieName.flv**。
- 8 为 “自动播放”、“使用首选媒体大小” 和 “保持高宽比” 复选框设置所需选项。

- 9 选择 `MediaController` 实例，并在“组件检查器”面板中将方向设置为垂直，方法是将其 `horizontal` 属性设置为 `false`。
 - 10 在“组件检查器”面板中，将 `backgroundStyle` 设置为 `None`。
这样会指定 `MediaController` 实例不应绘制背景，而应在控件之间填充媒体。
 - 11 使用一个行为关联 `MediaController` 和 `MediaDisplay` 实例，使 `MediaController` 实例准确地反映 `MediaDisplay` 实例中的播放头移动和其他设置，并使 `MediaDisplay` 实例响应用户的单击动作：
 - a 选择 `MediaDisplay` 实例，并在属性检查器中输入实例名称 **myMediaDisplay**。
 - b 选择将触发该行为的 `MediaController` 实例。
 - c 在“行为”面板（“窗口” > “开发面板” > “行为”）中，单击“增加” (+) 按钮并选择“媒体” > “关联显示”。
 - d 在“关联显示”窗口中，选择 `_root` 下的 `myMediaDisplay`，然后单击“确定”。
- 有关使用行为控制媒体组件的详细信息，请参阅第 305 页的“通过使用行为控制媒体组件”。

使用“组件检查器”面板设置媒体组件

“组件检查器”面板使用户能轻松地设置媒体组件的参数、属性等。要使用此面板，请在舞台上单击所需的组件，并在属性检查器打开的情况下，单击“启动组件检查器”。“组件检查器”面板可用于以下用途：

- 自动播放媒体（请参阅 `Media.activePlayControl` 和 `Media.autoPlay`）
- 保持或忽略媒体的高宽比（请参阅 `Media.aspectRatio`）
- 确定媒体是否将自动调整大小以适应组件实例（请参阅 `Media.autoSize`）
- 启用或禁用铬印染背景（请参阅 `Media.backgroundStyle`）
- 以 URL 的形式指定媒体的路径（请参阅 `Media.contentPath`）
- 指定播放控件的可见性（请参阅 `Media.controllerPolicy`）
- 添加线索点对象（请参阅 `Media.addCuePoint()`）
- 删除线索点对象（请参阅 `Media.removeCuePoint()`）
- 设置 `MediaController` 实例的方向（请参阅 `Media.horizontal`）
- 设置所播放的媒体的类型（请参阅 `Media.setMedia()`）
- 设置 FLV 媒体的播放时间（请参阅 `Media.totalTime`）
- 设置时间显示的最后几位数，以指示毫秒数或每秒帧数 (fps)

在使用“组件检查器”面板时，了解一些概念很重要：

- 当选择 MP3 视频类型时，视频时间控件会被删除，原因是此信息会在使用 MP3 文件时自动读入。对于 FLV 文件，必须输入媒体的总时间 (`Media.totalTime`)，以便使 `MediaPlayback` 组件（或任何侦听的 `MediaController` 组件）的播放栏准确地反映播放进度。
- 文件类型设置为 FLV 时，您将观察到一个“毫秒数”选项和（如果取消选择“毫秒数”）“每秒的帧” (FPS) 弹出式菜单。当选择“毫秒数”选项时，FPS 控件不可见。在此模式下，运行时在播放栏显示的时间为 HH:MM:SS.mmm 格式（H = 小时、M = 分钟、S = 秒、m = 毫秒），而且线索点以秒为单位设置。当取消选择“毫秒数”时，FPS 控件即会启用，且播放栏时间为 HH:MM:SS.FF 格式（F = 每秒的帧），而线索点以帧为单位设置。

注意：只能使用“组件检查器”面板设置 FPS 属性。使用动作脚本设置 fps 值无效且将被忽略。

通过使用行为控制媒体组件

行为是预先编写的动作脚本，为了控制对象而添加到对象实例（例如 MediaDisplay 组件）中。行为可让您为文档增添动作脚本编码的功能、控制能力和灵活性，而无需自己创建动作脚本代码。

要通过行为控制媒体组件，请使用“行为”面板将行为应用到给定的媒体组件实例。请指定将会触发行为的事件（例如到达指定的线索点）、选择目标对象（将受行为影响的媒体组件），并在必要时选择行为的设置（例如要导航到的媒体内的影片剪辑）。

以下行为随附在 Flash MX Professional 2004 中，用于控制嵌入的媒体组件。

行为	用途	参数
关联控制器	将 MediaController 组件与 MediaDisplay 组件关联	目标 MediaController 组件的实例名称
关联显示	将 MediaDisplay 组件与 MediaController 组件关联	目标 MediaController 组件的实例名称
标签帧线索点导航	将动作放到 MediaDisplay 或 MediaPlayer 实例上，通知指定的影片剪辑导航到与给定的线索点名称相同的帧	帧的名称和线索点的名称（它们的名称应该相同）
幻灯片线索点导航	使基于幻灯片的 Flash 文档导航到与给定的线索点名称相同的幻灯片	幻灯片的名称和线索点的名称（它们的名称应该相同）

将 MediaDisplay 组件与 MediaController 组件关联：

- 1 将 MediaDisplay 实例和 MediaController 实例放置于舞台上。
- 2 选择 MediaDisplay 实例，并使用属性检查器输入实例名称 **myMediaDisplay**。
- 3 选择将触发该行为的 MediaController 实例。
- 4 在“行为”面板（“窗口” > “开发面板” > “行为”）中，单击“增加” (+) 按钮并选择“媒体” > “关联显示”。
- 5 在“关联显示”窗口中，选择 _root 下的 myMediaDisplay，然后单击“确定”。

注意：如果已将 MediaDisplay 组件与 MediaController 组件关联，则不需要将 MediaController 组件与 MediaDisplay 组件关联。

将 MediaController 组件与 MediaDisplay 组件关联：

- 1 将 MediaDisplay 实例和 MediaController 实例放置于舞台上。
- 2 选择 MediaController 实例，并使用属性检查器输入实例名称 **myMediaController**。
- 3 选择将触发该行为的 MediaDisplay 实例。
- 4 在“行为”面板（“窗口” > “开发面板” > “行为”）中，单击“增加” (+) 按钮并选择“媒体” > “关联控制器”。
- 5 在“关联控制器”窗口中，选择 _root 下的 myMediaController，然后单击“确定”。

使用“标签帧线索点导航”行为：

- 1 将 MediaDisplay 或 MediaPlayer 组件实例放置于舞台上。
- 2 选择希望媒体导航到的帧，并使用属性检查器输入帧的名称 **myLabeledFrame**。
- 3 选择 MediaDisplay 或 MediaPlayer 实例。

- 在“组件检查器”面板中，单击“增加” (+) 按钮并以 HH:MM:SS:mmm 或 HH:MM:SS:FF 格式输入线索点时间，然后为线索点指定名称 **myLabeledFrame**。
线索点指示在导航到选定的帧之前应经过的时间量。例如，如果要跳至 myLabeledFrame（影片中的第 5 秒位置），请在 SS 文本框中输入 **5**，并在“名称”文本框中输入 **myLabeledFrame**。
- 在“行为”面板（“窗口” > “开发面板” > “行为”）中，单击“增加” (+) 按钮并选择“媒体” > “标签帧线索点导航”。
- 在“标签帧线索点导航”窗口中，选择 _root 剪辑并单击“确定”。

使用“幻灯片线索点导航”行为：

- 打开新的 Flash 幻灯片演示文稿文档。
- 将 MediaDisplay 或 MediaPlayer 组件实例放置于舞台上。
- 在舞台左侧的“屏幕大纲”窗格中，单击“插入屏幕” (+) 按钮以添加第二张幻灯片，然后选择第二张幻灯片并重命名为 **mySlide**。
- 选择 MediaDisplay 或 MediaController 实例。
- 在“组件检查器”面板中，单击“增加” (+) 按钮并以 HH:MM:SS:mmm 或 HH:MM:SS:FF 格式输入线索点时间，然后为线索点指定名称 **MySlide**。
线索点指示在导航到选定的幻灯片之前应经过的时间量。例如，如果要跳至 mySlide（影片中的第 5 秒位置），请在 SS 文本框中输入 **5**，并在“名称”文本框中输入 **mySlide**。
- 在“行为”面板（“窗口” > “开发面板” > “行为”）中，单击“增加” (+) 按钮并选择“媒体” > “幻灯片线索点导航”。
- 在“幻灯片线索点导航”窗口中，选择 _root 剪辑下的 Presentation，然后单击“确定”。

媒体组件参数（仅限 Flash Professional）

以下表格列出了可以在属性检查器中为给定的媒体组件实例设置的创作参数：

MediaDisplay 组件参数

名称	类型	默认值	描述
自动播放 (Media.autoPlay)	Boolean	Selected	确定是否在加载媒体后立刻播放该媒体。
使用首选媒体大小 (Media.autoSize)	Boolean	Selected	确定与 MediaDisplay 实例关联的媒体是符合组件大小，还是仅使用其默认的大小。
FPS	整数	30	指示每秒的帧数。当选择了“毫秒数”选项时，此控件禁用。
线索点 (Media.cuePoints)	Array	Undefined	线索点对象的一个数组，这些对象各自具有一个名称和时间位置，时间格式为有效的 HH:MM:SS:FF（选择了“毫秒数”选项时）或 HH:MM:SS:mmm。
FLV 或 MP3 (Media.mediaType)	“FLV” 或 “MP3”	“FLV”	指定要播放的媒体类型。
毫秒	Boolean	Unselected	确定播放条是使用帧还是毫秒，以及线索点是使用秒还是帧。当选择此选项时，FPS 控件不可见。

名称	类型	默认值	描述
URL (Media.contentPath)	String	Undefined	一个字符串，保存要播放的媒体的路径和文件名。
视频长度 (Media.totalTime)	整数	Undefined	播放 FLV 媒体所需的总时间。此设置是确保播放条正常工作所必需的。此控件仅在媒体类型设置为 FLV 时可见。

MediaController 组件参数

名称	类型	默认值	描述
activePlayControl (Media.activePlayControl)	字符串： “pause” 或 “play”	“pause”	确定播放栏在实例化时是处于播放模式还是暂停模式。
backgroundStyle (Media.backgroundStyle)	字符串： “default” 或 “none”	“default”	确定是否为 MediaController 实例绘制铬印染背景。
controllerPolicy (Media.controllerPolicy)	“auto”、 “on”、或 “off”	“auto”	确定控制器是根据鼠标位置打开或关闭，还是锁定在打开或关闭状态。
horizontal (Media.horizontal)	Boolean	true	确定实例的控制器部分为垂直方向还是水平方向。true 值表示组件将为水平方向。
enabled	Boolean	true	确定此控件是否可由用户修改。true 值表示可修改此控件。
visible	Boolean	true	确定此控件是否对用户可见。true 值表示此控件可见。
minHeight	整数	0	此实例的允许最小高度，以像素为单位。
minWidth	整数	0	此实例的允许最小宽度，以像素为单位。

MediaPlayback 组件参数

名称	类型	默认值	描述
控件位置 (Media.controlPlacement)	“top”、 “bottom”、 “left”、 “right”	“bottom”	控制器的位置。值与方向有关。
Media.controllerPolicy	Boolean	true	确定控制器是否根据鼠标的位置而打开或关闭。
自动播放 (Media.autoPlay)	Boolean	Selected	确定是否在加载媒体后立刻播放该媒体。
使用首选媒体大小 (Media.autoSize)	Boolean	Selected	确定 MediaController 实例大小是与媒体相适应还是使用其他设置。

名称	类型	默认值	描述
FPS	整数	30	每秒的帧数。当选择了“毫秒数”选项时，此控件禁用。
线索点 (Media.cuePoints)	Array	Undefined	线索点对象的一个数组，这些对象各自具有一个名称和时间位置，时间格式为有效的 HH:MM:SS:mmm（选择了“毫秒数”选项时）或 HH:MM:SS:FF。
FLV 或 MP3 (Media.mediaType)	“FLV” 或 “MP3”		指定要播放的媒体类型。
毫秒	Boolean	Unselected	确定播放条是使用帧还是毫秒，以及线索点是使用秒还是帧。当选择此选项时，FPS 控件禁用。
URL (Media.contentPath)	String	Undefined	一个字符串，保存要播放的媒体的路径和文件名。
视频长度 (Media.totalTime)	整数	Undefined	播放 FLV 媒体所需的总时间。此设置是确保播放条正常工作所必需的。

使用媒体组件创建应用程序（仅限 Flash Professional）

使用媒体组件创建 Flash 内容非常简单，通常只需要几个步骤。

此范例显示如何创建一个播放公开提供的小型媒体文件的应用程序。

要将媒体组件添加到应用程序：

- 1 在 Flash 中，选择“文件” > “新建”，然后选择“Flash 文档”。
- 2 在“组件”面板（“窗口” > “开发面板” > “组件”）中，双击 MediaPlayer 组件以将其添加到舞台上。
- 3 在属性检查器中，输入实例名称 **myMedia**。
- 4 在属性检查器中，单击“启动组件检查器”。
- 5 在“组件检查器”面板中，在 URL 文本框内输入 **http://www.cathphoto.com/c.flv**。
- 6 选择“控制” > “测试影片”，查看媒体播放。

自定义媒体组件（仅限 Flash Professional）

如果要更改媒体组件的外观，可以使用外观设置。有关组件自定义的完整指南，请参阅 [第 25 页](#) 的 [第 3 章 “自定义组件”](#)。

在媒体组件中使用样式

媒体组件不支持样式。

在媒体组件中使用外观

尽管可以打开媒体组件源文档并更改其资源以获得需要的外观，但媒体组件不支持动态外观设置。最好对此文件制作一份副本并使用该副本，这样即可始终拥有可恢复的已安装的源文件。可以在以下位置找到媒体组件源文档：

- Windows : C:\Documents and Settings\用户\Local Settings\Application Data\Macromedia\Flash MX 2004\语言\Configuration\ComponentFLA fla

- Macintosh : HD Drive:Users: 用户名 :Library:Application Support:Macromedia:Flash MX 2004: 语言 :Configuration:ComponentFLA fla
- 有关组件外观的详细信息，请参阅第 33 页的“关于设置组件外观”。

Media 类（仅限 Flash Professional）

继承 mx.core.UIComponent

动作脚本类名称 mx.controls.MediaController、mx.controls.MediaDisplay、mx.controls.MediaPlayback

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.MediaPlayback.version);
```

注意：代码 trace(myMediaInstance.version); 返回 undefined。

Media 类的方法摘要

方法	组件	描述
Media.addCuePoint()	MediaDisplay、MediaPlayback	向组件实例添加线索点对象。
Media.associateController()	MediaDisplay	将 MediaDisplay 实例与 MediaController 实例关联。
Media.associateDisplay()	MediaController	将 MediaController 实例与 MediaDisplay 实例关联。
Media.displayFull()	MediaPlayback	将组件实例转换为全屏播放模式。
Media.displayNormal()	MediaPlayback	将组件实例转换回其原始屏幕大小。
Media.getCuePoint()	MediaDisplay、MediaPlayback	返回线索点对象。
Media.play()	MediaDisplay、MediaPlayback	在给定的开始点播放与组件实例关联的媒体。
Media.pause()	MediaDisplay、MediaPlayback	在媒体时间轴中的播放头当前位置暂停播放头。
Media.removeAllCuePoints()	MediaDisplay、MediaPlayback	删除与给定的组件实例关联的所有线索点对象。
Media.removeCuePoint()	MediaDisplay、MediaPlayback	删除与给定的组件实例关联的指定线索点。
Media.setMedia()	MediaDisplay、MediaPlayback	将媒体类型和路径设置为指定的媒体类型。
Media.stop()	MediaDisplay、MediaPlayback	停止播放头并将其移到位置 0，即媒体的开头。

Media 类的属性摘要

属性	组件	描述
<code>Media.activePlayControl</code>	MediaController	确定组件在运行时加载时的状态。
<code>Media.aspectRatio</code>	MediaDisplay、MediaPlayback	确定组件实例是否保持其视频高宽比。
<code>Media.autoPlay</code>	MediaDisplay、MediaPlayback	确定组件实例是否立即开始缓冲和播放。
<code>Media.autoSize</code>	MediaDisplay、MediaPlayback	确定 MediaDisplay 或 MediaPlayback 组件的媒体观看部分如何调整其自身的大小。
<code>Media.backgroundStyle</code>	MediaController	确定组件实例是否绘制其铬印染背景。
<code>Media.bytesLoaded</code>	MediaDisplay、MediaPlayback	已加载且可以播放的字节数。
<code>Media.bytesTotal</code>	MediaDisplay、MediaPlayback	要加载到组件实例中的字节数。
<code>Media.contentPath</code>	MediaDisplay、MediaPlayback	一个字符串，保存要进行流式处理并播放的媒体的相对路径和文件名。
<code>Media.controllerPolicy</code>	MediaController、MediaPlayback	确定组件内的控件在播放时是隐藏（仅在触发鼠标移过事件时显示），还是始终可见或始终隐藏。
<code>Media.controlPlacement</code>	MediaPlayback	确定组件的控件是否相对于组件定位。
<code>Media.cuePoints</code>	MediaDisplay、MediaPlayback	已指定到给定的组件实例的线索点对象数组。
<code>Media.horizontal</code>	MediaController	确定组件实例的方向。
<code>Media.mediaType</code>	MediaDisplay、MediaPlayback	确定要播放的媒体类型。
<code>Media.playheadTime</code>	MediaDisplay、MediaPlayback	对于正在播放的媒体时间轴，保存播放头的当前位置（以秒为单位）。
<code>Media.playing</code>	MediaDisplay、MediaPlayback	返回一个布尔值，指示给定的组件实例是否正在播放媒体。
<code>Media.preferredHeight</code>	MediaDisplay、MediaPlayback	FLV 媒体文件的默认高度值。
<code>Media.preferredWidth</code>	MediaDisplay、MediaPlayback	FLV 媒体文件的默认宽度值。
<code>Media.totalTime</code>	MediaDisplay、MediaPlayback	一个整数，指示媒体的总长度（以秒为单位）。
<code>Media.volume</code>	MediaDisplay、MediaPlayback	一个介于 0（最小值）和 100（最大值）之间的整数，代表音量级别。

Media 类的事件摘要

事件	组件	描述
Media.change	MediaDisplay、MediaPlayback	当媒体播放时连续广播。
Media.click	MediaController、MediaPlayback	当用户单击 “播放” / “暂停” 按钮时进行广播。
Media.complete	MediaDisplay、MediaPlayback	播放头已到达媒体结尾的通知。
Media.cuePoint	MediaDisplay、MediaPlayback	播放头已到达给定的线索点的通知。
Media.playheadChange	MediaController、MediaPlayback	当用户移动播放滑块或单击 “转到开头” 或 “转到结尾” 按钮时，由组件实例进行广播。
Media.progress	MediaDisplay、MediaPlayback	连续生成，直到媒体完全下载。
Media.volume	MediaController、MediaPlayback	当用户调整音量时进行广播。

Media.activePlayControl

适用于

MediaController

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.activePlayControl
```

描述

属性；一个布尔值，确定 MediaController 组件在运行时加载时的状态。true 值指示 MediaController 组件在运行时应处于播放状态，false 值指示它在运行时处于暂停状态。此属性应该与 autoPlay 属性一起设置，这样两者在运行时同时为暂停或播放状态。默认值为 true。

范例

下面的范例指示当在运行时首次加载时，控件为暂停状态：

```
myMedia.activePlayControl = false;
```

另请参见

[Media.autoPlay](#)

Media.addCuePoint()

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.addCuePoint(cuePointName, cuePointTime)
```

参数

cuePointName 一个字符串，可以用于对线索点命名。

cuePointTime 一个以秒数表示的数字，指示何时广播 `cuePoint` 事件。

返回

无。

说明

方法；向 `MediaPlayback` 或 `MediaDisplay` 组件实例添加线索点对象。当播放头时间等于线索点时间时，即会广播 `cuePoint` 事件。

示例

以下代码将名为 `Homerun` 的线索点添加到 `myMedia` 中时间 = 16 秒的位置。

```
myMedia.addCuePoint("Homerun", 16);
```

另请参见

[Media.cuePoint](#)、[Media.cuePoints](#)、[Media.getCuePoint\(\)](#)、[Media.removeAllCuePoints\(\)](#)、[Media.removeCuePoint\(\)](#)

Media.aspectRatio

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.aspectRatio
```


描述

属性；一个布尔值，确定 `MediaDisplay` 或 `MediaPlayback` 实例在播放时是否保持视频高宽比。`true` 值指示应保持高宽比；`false` 值指示高宽比可以在播放时变化。默认值为 `true`。

范例

下面的范例指示高宽比可以在播放时变化：

```
myMedia.aspectRatio = false;
```

Media.associateController()

适用于

`MediaDisplay`

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.associateController(instanceName)
```

参数

instanceName 一个字符串，指示要关联的 `MediaController` 组件的实例名称。

返回

无。

说明

方法；将 `MediaDisplay` 组件实例与给定的 `MediaController` 实例关联。

如果已经使用 `Media.associateDisplay()` 将 `MediaController` 实例与 `MediaDisplay` 实例关联，则无需使用 `Media.associateController()`。

示例

以下代码将 `myMedia` 与 `myController` 关联：

```
myMedia.associateController(myController);
```

另请参见

[Media.associateDisplay\(\)](#)

Media.associateDisplay()

适用于

`MediaController`

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.associateDisplay(instanceName)
```

参数

instanceName 一个字符串，指示要关联的 MediaDisplay 组件的实例名称。

返回

无。

说明

方法；将 MediaController 组件实例与给定的 MediaDisplay 实例关联。

如果已经使用 Media.associateController() 将 MediaDisplay 实例与 MediaController 实例关联，则无需使用 Media.associateDisplay()。

示例

以下代码将 myMedia 与 myDisplay 关联：

```
myMedia.associateDisplay(myDisplay);
```

另请参见

[Media.associateController\(\)](#)

Media.autoPlay

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.autoPlay
```

描述

属性；一个布尔值，确定 MediaPlayback 或 MediaDisplay 实例是否立即开始尝试缓冲和播放。true 值指示控件将在运行时进行缓冲和播放；false 值指示控件将在运行时停止。此属性取决于 contentPath 和 mediaType 属性。如果未设置 contentPath 和 mediaType，则在运行时不会进行播放。默认值为 true。

范例

下面的范例指示当在运行时首次加载时，控件不会启动：

```
myMedia.autoPlay = false;
```

另请参见

[Media.contentPath](#)、[Media.mediaType](#)

Media.autoSize

适用于

[MediaDisplay](#)、[MediaPlayback](#)

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.autoSize
```

描述

属性；一个布尔值，确定 [MediaDisplay](#) 或 [MediaPlayback](#) 组件的媒体观看部分如何调整其自身的大小。

对于 [MediaDisplay](#) 组件，此属性的作用如下所示：

- 如果将此属性设置为 `true`，Flash 将以媒体的首选大小显示媒体，而与组件大小无关。这意味着，除非 [MediaDisplay](#) 实例大小与该媒体的大小相同，否则，该媒体将溢出实例边界或无法填充实例大小。
- 如果将此属性设置为 `false`，Flash 将尽可能用实例的大小，同时保持高宽比。如果 [Media.autoSize](#) 和 [Media.aspectRatio](#) 都设置为 `false`，将使用该组件的准确大小。

对于 [MediaPlayback](#) 组件，此属性的作用如下所示：

- 如果将此属性设置为 `true`，Flash 将以媒体的首选大小显示媒体，除非播放媒体区小于首选的大小。在这种情况下，Flash 将缩小媒体以适应实例的内部大小，并保持高宽比。如果首选大小小于实例的媒体区，则将不使用部分媒体区。
- 如果将此属性设置为 `false`，Flash 将尽可能用实例的大小，同时保持高宽比。如果 [Media.autoSize](#) 和 [Media.aspectRatio](#) 都设置为 `false`，则将填充组件的媒体区。此区域定义为控件上面的区域（在默认的布局下），周围有 8 个像素的边距，组成了组件的边缘。

默认值为 `true`。

范例

下面的范例指示控件不会根据其媒体大小进行播放：

```
myMedia.autoSize = false;
```

另请参见

[Media.aspectRatio](#)

Media.backgroundColor

适用于

MediaController

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.backgroundColor
```

描述

属性；"default" 值指示将为 MediaController 实例绘制铬印染背景，而 "none" 值指示将不会绘制铬印染背景。默认值为 "default"。

它不是样式属性，因此不受样式设置影响。

范例

下面的范例指示将不会为控件绘制铬印染背景：

```
myMedia.backgroundColor = "none";
```

Media.bytesLoaded

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.bytesLoaded
```

描述

只读属性；已加载到组件且可以播放的字节数。默认值是 undefined。

范例

以下代码创建名为 PlaybackLoad 的变量，并将在 for 循环中利用加载的字节数设置该变量。

```
// 创建一个变量，用于保存所加载的字节数
var PlaybackLoad = myMedia.bytesLoaded;
// 执行 some 函数，直到准备好播放
for (PlaybackLoad < 150) {
    someFunction();
}
```

Media.bytesTotal

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.bytesTotal
```

描述

属性；要加载到 MediaPlayback 或 MediaDisplay 组件的字节数。默认值是 `undefined`。

范例

下面的范例告知用户要进行流式处理的媒体大小：

```
myTextField.text = myMedia.bytesTotal;
```

Media.change

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

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

描述

事件；当播放媒体时由 MediaDisplay 和 MediaPlayback 组件进行广播。完成的百分比可以从组件实例中获取。请参阅下面的范例。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Media.change 事件的事件对象有两个附加属性：

`target` 对广播中的对象的引用。

`type` 指示事件类型的字符串 "change"。

有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

示例

下面的范例使用对象侦听器确定播放头位置 (`Media.playheadTime`)，从中可计算出完成的百分比：

```
var myPlayerListener = new Object();
myPlayerListener.change = function(eventObject){
    var myPosition = myPlayer.playheadTime;
    var myPercentPosition = (myPosition/totalTime);
}
myPlayer.addEventListener("change", myPlayerListener);
```

另请参见

[Media.playing](#)、[Media.pause\(\)](#)

Media.click

适用于

[MediaController](#)、[MediaPlayback](#)

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
var myMediaListener = new Object()
myMediaListener.click = function(){
    // 此处插入您的代码
}
myPlayer.addEventListener("click", myMediaListener);
```

描述

事件；当用户单击“播放”/“暂停”按钮时进行广播。`detail` 字段将用于确定所单击的按钮。`Media.click` 事件对象具有以下属性：

`detail` 字符串 "pause" 或 "play"。

`target` 对 `MediaController` 或 `MediaPlayback` 组件实例的引用。

`type` 字符串 "click"。

范例

下面的范例会在用户单击“播放”时打开一个弹出式窗口：

```
var myMediaListener = new Object()
myMediaListener.click = function(){
    PopUpManager.createPopup(_root, mx.containers.Window, false,
    {contentPath:movieSale});
}
myMedia.addEventListener("click", myMediaListener);
```

Media.complete

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
    // 此处插入您的代码
}
myMedia.addEventListener("complete", listenerObject)
```

描述

事件；播放头已到达媒体结尾的通知。Media.complete 事件对象具有以下属性：

target 对 MediaDisplay 或 MediaPlayback 组件实例的引用。

type 字符串 "complete"。

范例

下面的范例使用对象侦听器确定媒体何时结束播放：

```
var myListener = new Object();
myListener.complete = function(eventObject) {
    trace("media is Finished");
};
myMedia.addEventListener("complete", myListener);
```

Media.contentPath

适用于

MediaController

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.contentPath
```

描述

属性；一个字符串，保存要进行流式处理和 / 或播放的媒体的相对路径和文件名。

[Media.setMedia\(\)](#) 方法是通过动作脚本设置此属性的唯一受支持的方法。默认值是 undefined。

范例

下面的范例在文本框中显示播放中影片的名称：

```
myTextField.text = myMedia.contentPath;
```

另请参见

[Media.setMedia\(\)](#)

Media.controllerPolicy

适用于

MediaController、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.controllerPolicy
```

描述

属性；确定 MediaController 组件（或 MediaPlayer 组件内的控制器子组件）是否在实例化时隐藏，而仅当用户将鼠标移过控制器的折叠状态时显示。

此属性的可能值如下：

- "on" 指示控件始终是展开的。
- "off" 指示控件始终是折叠的。
- "auto" 指示控件将保持折叠状态，直到用户将鼠标移过点击区。点击区与在其中绘制折叠的控件的区域匹配。控件将保持展开状态，直到鼠标移离点击区。

注意：点击区会随着控制器扩展和收缩。

范例

下面的范例将使控制器始终保持打开：

```
myMedia.controllerPolicy = "on";
```

Media.controlPlacement

适用于

MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.controlPlacement
```

描述

属性；确定 MediaPlayer 组件的控制器部分相对于其显示的位置。可能的值有 "top"、"bottom"、"left" 和 "right"。默认值为 "bottom"。

范例

在下面的范例中，MediaPlayer 组件的控制器部分将位于右侧：

```
myMedia.controlPlacement = "right";
```

Media.cuePoint

适用于

MediaDisplay、MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.cuePoint = function(eventObject){
    // 此处插入您的代码
}
myMedia.addEventListener("cuePoint", listenerObject)
```

描述

事件；播放头已到达线索点的通知。Media.cuePoint 事件对象具有以下属性：

name 一个字符串，指示线索点的名称。

time 一个数字，以帧数或秒数表示，指示何时到达线索点。

target 对线索点对象的引用。

type 字符串 "cuePoint"。

范例

下面的范例使用对象侦听器确定何时到达线索点：

```
var myCuePointListener = new Object();
myCuePointListener.cuePoint = function(eventObject){
    trace("heard " + eventObject.type + ", " + eventObject.target);
}
myPlayback.addEventListener("cuePoint", myCuePointListener);
```

另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoints](#)、[Media.getCuePoint\(\)](#)

Media.cuePoints

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.cuePoints[N]
```

描述

属性；已指定给 MediaPlayback 或 MediaDisplay 组件实例的线索点对象数组。在数组内，每个线索点对象可以具有名称、以秒或帧数表示的时间，以及一个播放器属性（即对象所关联的组件的实例名称）。默认值为空数组 []。

范例

如果正在播放动作预览，下面的范例会删除第三个线索点：

```
if(myVariable == actionPreview) {  
    myMedia.removeCuePoint(myMedia.cuePoints[2]);  
}
```

另请参见

[Media.addCuePoint\(\)](#)、[Media.getCuePoint\(\)](#)、[Media.removeCuePoint\(\)](#)

Media.displayFull()

适用于

MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.displayFull()
```

参数

无。

返回

无。

说明

方法；将 MediaPlayer 组件实例设置为全屏模式。也就是说，组件进行扩展以填充整个舞台。要将组件返回到正常大小，请使用 `Media.displayNormal()`。

示例

以下代码使组件进行扩展以填满舞台：

```
myMedia.displayFull();
```

另请参见

[Media.displayNormal\(\)](#)

Media.displayNormal()

适用于

MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.displayNormal();
```

参数

无。

返回

无。

说明

方法；在使用 `Media.displayFull()` 方法之后，将 MediaPlayer 实例设置回正常大小。

示例

以下代码会使 MediaPlayer 组件回复为其原始大小：

```
myMedia.displayNormal();
```

另请参见

[Media.displayFull\(\)](#)

Media.getCuePoint()

适用于

MediaDisplay、MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.getCuePoint(cuePointName)
```

参数

无。

返回

cuePointName 在使用 `Media.addCuePoint()` 时提供的字符串。

说明

方法；根据其线索点名称返回线索点对象。

示例

以下代码会获取名为 `myCuePointName` 的线索点。

```
myMedia.removeCuePoint(myMedia.getCuePoint("myCuePointName"));
```

另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoint](#)、[Media.cuePoints](#)、[Media.removeCuePoint\(\)](#)

Media.horizontal

适用于

`MediaController`

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.horizontal
```

描述

属性；确定 `MediaController` 组件是以垂直方向显示还是以水平方向显示。`true` 值指示组件将以水平方向显示；`false` 值指示以垂直方向显示。设置为 `false` 时，播放头和加载进度指示器从下往上移动。默认值为 `true`。

范例

下面的范例将以垂直方向显示 `MediaController` 组件：

```
myMedia.horizontal = false;
```

Media.mediaType

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.mediaType
```

描述

属性；保存要播放的媒体的类型值。可以选择 FLV 和 MP3 格式。默认值为 "FLV"。请参阅“使用 Flash”帮助中的“导入 Macromedia Flash 视频 (FLV) 文件”。

范例

下面的范例确定当前播放的媒体类型：

```
var currentMedia = myMedia.mediaType;
```

另请参见

[Media.setMedia\(\)](#)

Media.pause()

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.pause()
```

参数

无。

返回

无，

说明

方法；在当前位置暂停播放头。

示例

以下代码会暂停播放。

```
myMedia.pause();
```

Media.play()

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.play(startingPoint)
```

参数

startingPoint 一个非负整数值，指示媒体应开始播放的开始点（以秒为单位）。

返回

无。

说明

方法；在给定的开始点播放与组件实例关联的媒体。默认值为 `playheadTime` 的当前值。

示例

以下代码指示媒体组件应在 120 秒处开始播放：

```
myMedia.play(120);
```

另请参见

[Media.pause\(\)](#)

Media.playheadChange

适用于

MediaController、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.playheadChange = function(eventObject) {  
    // 此处插入您的代码
```

```
}  
myMedia.addEventListener("playheadChange", listenerObject)
```

描述

事件；当用户移动播放滑块或单击“转到开头”或“转到结尾”按钮时，由 MediaController 或 MediaPlayer 组件进行广播。Media.playheadChange 事件对象具有以下属性：

detail 一个数字，指示已播放的媒体的百分比。

type 字符串 "playheadChange"。

范例

下面的范例在用户停止拖动播放头时将已播放的百分比发送到“输出”面板：

```
var controlListen = new Object();  
controlListen.playheadChange = function(eventObject){  
    trace(eventObject.detail);  
}  
myMedia.addEventListener("playheadChange", controlListen);
```

Media.playheadTime

适用于

MediaDisplay、MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.playheadTime
```

描述

属性；对于正在播放的媒体时间轴，保存播放头的当前位置（以秒为单位）。默认值设置为播放头的位置。

范例

下面的范例为播放头位置设置一个变量，以秒为单位表示：

```
var myPlayhead = myMedia.playheadTime;
```

Media.playing

适用于

MediaDisplay、MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`myMedia.playing`

说明

只读属性；返回一个布尔值，指示媒体是否正在播放。`true` 值指示媒体正在播放；`false` 指示媒体由用户暂停。

示例

以下代码确定媒体是正在播放还是暂停：

```
if(myMedia.playing == true){
    some function;
}
```

另请参见

[Media.change](#)

Media.preferredHeight

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`myMedia.preferredHeight`

描述

属性；根据 FLV 的默认高度值设置。此属性仅适用于 FLV 媒体，因为高度对于 MP3 文件是固定的。此属性可以用于设置高度和宽度参数（加上组件自身的边距）。如果未设置 FLV 媒体，默认值为 `undefined`。

范例

下面的范例根据正在播放的实例调整 MediaPlayback 实例大小，并说明了组件实例所需的边距像素：

```
if(myPlayback.contentPath != !undefined){
    var mediaHeight = myPlayback.preferredHeight;
    var mediaWidth = myPlayback.preferredWidth;
    myPlayback.setSize((mediaWidth + 20), (mediaHeight + 70));
}
```


Media.preferredWidth

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.preferredWidth
```

描述

属性；根据 FLV 的默认宽度值设置。默认值是 `undefined`。

范例

下面的范例设置变量 `mediaWidth` 的所需宽度：

```
var mediaWidth = myMedia.preferredWidth;
```

Media.progress

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.progress = function(eventObject){  
    // 此处插入您的代码  
}  
myMedia.addEventListener("progress", listenerObject)
```

描述

事件；连续生成，直到媒体已完全下载。Media.progress 事件对象具有以下属性：

target 对 MediaDisplay 或 MediaPlayback 组件实例的引用。

type 字符串 "progress"。

范例

下面的范例会侦听进程：

```
var myProgressListener = new Object();  
myProgressListener.progress = function(){  
    // 使 lightMovieClip 在出现进程时闪烁
```

```
        var lightVisible = lightMovieClip.visible;  
        lightMovieClip.visible = !lightVisible;  
    }
```

Media.removeAllCuePoints()

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.removeAllCuePoints()
```

参数

无。

返回

无。

说明

方法；删除与组件实例关联的所有线索点对象。

示例

以下代码会删除所有线索点对象：

```
myMedia.removeAllCuePoints();
```

另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoints](#)、[Media.removeCuePoint\(\)](#)

Media.removeCuePoint()

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.removeCuePoint(cuePoint)
```

参数

cuePoint 对先前已通过 [Media.addCuePoint\(\)](#) 指定的线索点对象的引用。

返回

无。

说明

方法；删除与组件实例关联的特定线索点。

示例

以下代码会删除名为 `myCuePoint` 的线索点：

```
myMedia.removeCuePoint(getCuePoint("myCuePoint"));
```

另请参见

[Media.addCuePoint\(\)](#)、[Media.cuePoints](#)、[Media.removeAllCuePoints\(\)](#)

Media.setMedia()

适用于

[MediaDisplay](#)、[MediaPlayback](#)

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.setMedia(contentPath, mediaType)
```

参数

contentPath 一个字符串，指示要播放的媒体的路径和文件名。

mediaType 一个字符串，用于将媒体类型设置为 FLV 或 MP3。此参数是可选的。

返回

无。

说明

方法；使用 URL 参量将媒体类型和路径设置为指定的媒体类型。*contentPath* 的默认值未定义。

此方法提供了设置 [MediaPlayback](#) 和 [MediaDisplay](#) 组件的内容路径和媒体类型的唯一受支持的方法。

示例

以下代码为要播放的组件实例提供了新媒体。

```
myMedia.setMedia("http://www.RogerMoore.com/moonraker.flv", "FLV");
```

Media.stop()

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.stop()
```

参数

无。

返回

无。

说明

方法；停止播放头并将其移到位置 0，即媒体的开头。

示例

以下代码停止播放头，并将其移至时间 = 0 处。

```
myMedia.stop()
```

Media.totalTime

适用于

MediaDisplay、MediaPlayback

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.totalTime
```

描述

属性；媒体的总长度（以秒为单位）。由于 FLV 文件格式在完全加载之前不向媒体组件提供播放时间，因此需要手动输入 `Media.totalTime`，以便播放滑块可以准确地反映媒体的实际播放时间。MP3 文件的默认值为媒体的播放时间。对于 FLV 文件，默认值为 `undefined`。

不能为 MP3 文件设置此属性，因为信息包含在 `Sound` 对象内。

范例

下面的范例为 FLV 媒体设置播放时间（以秒为单位）：

```
myMedia.totalTime = 151;
```

Media.volume

适用于

MediaDisplay、 MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
myMedia.volume
```

描述

属性；存储音量设置整数值，范围为 0 到 100。此属性的默认值为 75。

范例

下面的范例为媒体播放设置最大音量：

```
myMedia.volume = 100;
```

另请参见

[Media.volume](#)、[Media.pause\(\)](#)

Media.volume

适用于

MediaController、 MediaPlayer

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.volume = function(eventObject){  
    // 此处插入您的代码  
}  
myMedia.addEventListener("volume", listenerObject)
```

描述

事件；当音量值由用户调整时进行广播。Media.volume 事件对象具有以下属性：

detail 0 到 100 之间的一个整数值，指示音量级别。

type 字符串 "volume"。

范例

下面的范例将通知用户音量已被调整：

```
var myVolListener = new Object();
myVolListener.volume = function(){
    mytextfield.text = "Volume adjusted!";
}
myMedia.addEventListener("volume", myVolListener);
```

另请参见

[Media.volume](#)

Menu 组件（仅限 Flash Professional）

Menu 组件使用户可以从弹出式菜单中选择一个项目，这与大多数软件应用程序的“文件”或“编辑”菜单很相似。

通常在用户滑过或单击一个像按钮的菜单激活器时，会在应用程序中打开 Menu。还可以对 Menu 组件编写脚本，使其在用户按下特定的键时打开。

Menu 组件始终在运行时动态创建。必须将组件从“组件”面板添加到文档，然后删除它以将其添加到库中。然后，使用以下代码通过动作脚本创建菜单：

```
var myMenu = mx.controls.Menu.createMenu(parent, menuDataProvider);
```

使用以下代码在应用程序中打开菜单：

```
myMenu.show(x, y);
```

menuShow 事件在菜单呈现的前一刻对所有 Menu 实例的侦听器进行广播，从而可以更新菜单项的状态。类似地，在 Menu 实例隐藏后，会立即广播 menuHide 事件。

菜单中的项目是以 XML 进行描述的。有关详细信息，请参阅第 335 页的“[了解 Menu 组件：视图和数据](#)”。

无法使用屏幕阅读器来访问 Menu 组件。

与 Menu 组件进行交互（仅限 Flash Professional）

可以使用鼠标和键盘与 Menu 组件进行交互。

Menu 打开后会保持可见，直到它由脚本关闭，或者用户在菜单外部或启用的菜单项内部单击鼠标。

进行单击即会选中菜单项，但以下菜单项类型除外：

- 禁用的菜单项或分隔符 滚动和单击不起作用（菜单仍保持可见）。
- 子菜单的锚记 滚动会激活子菜单；单击无效；滚动到子菜单的项目以外的任何项目会关闭子菜单。

选中一个项目时，Menu.change 事件会发送到菜单的所有侦听器、菜单会被隐藏，而且发生以下动作（取决于项目类型）：

- check 切换该项目的 selected 属性。
- radio 该项目成为其单选按钮组的当前选项。

移动鼠标会触发 Menu.rollOut 和 Menu.rollOver 事件。

在菜单外部按鼠标会关闭菜单并触发 Menu.menuHide 事件。

在启用的项目内部松开鼠标会按以下方式影响项目类型：

- **check** 切换该项目的 `selected` 属性。
- **radio** 项目的 `selected` 属性设置为 `true`，而且上次在单选按钮组中所选项目的 `selected` 属性设置为 `false`。会设置相应单选按钮组对象的 `selection` 属性，以引用选中的菜单项。
- **未定义和分层菜单的父项** 切换分层菜单的可见性。

当 `Menu` 实例从单击或 `Tab` 键切换中获得焦点时，您可以使用以下按键来控制它：

按键	描述
向下箭头 向上箭头	在菜单行中向下和向上移动选区。选区在顶部或底部行循环。
右箭头	打开子菜单，或者将选区移动到菜单栏（如果存在菜单栏）中的下一个菜单。
左箭头	关闭子菜单并将焦点返回到父菜单（如果存在父菜单），或者将选区移动到菜单栏（如果存在菜单栏）中的上一个菜单。
Enter 键	打开子菜单，或者如果不存在子菜单，在某一行单击并松开。

注意：如果打开了某个菜单，可以按 `Tab` 键移出该菜单。必须要进行选择，或者按 `Escape` 键退出菜单。

使用 Menu 组件（仅限 Flash Professional）

可以使用 `Menu` 组件创建具有可分别选择的选项的菜单，就像大多数软件应用程序的“文件”和“编辑”菜单。还可以使用 `Menu` 组件创建上下文关联菜单，这些菜单在用户按下热点或组合键时出现。与 `MenuBar` 组件一起使用 `Menu` 组件来创建水平菜单栏，它带有在每个菜单栏项目下扩展的菜单。

与标准的桌面菜单一样，`Menu` 组件支持功能属于以下一般类别的菜单项：

命令激活器 这些项目会触发事件；可以编写代码处理这些事件。

子菜单锚记 这些项目是用于打开子菜单的锚记。

单选按钮 这些项目成组使用；一次只能选择一个项目。

复选框项目 这些项目代表一个布尔值（`true` 或 `false`）。

分隔符 这些项目提供简单的水平线，将菜单中的项目在外观上分为不同的组。

了解 Menu 组件：视图和数据

从概念上来说，`Menu` 组件由数据模型和显示数据的视图组成。`Menu` 类就是这种视图，并且包含可视的配置方法。[MenuDataProvider](#) 类向全局 XML 原型对象添加方法（很类似于 `DataProvider` 类对 `Array` 对象所作的操作）；这些方法使您可以在外部构建数据提供程序并将其添加到多个菜单实例中。数据提供程序会向其所有客户端视图广播所有更改。（请参阅[第 355 页的“MenuDataProvider 类”](#)。）

`Menu` 实例是与各个菜单项对应的 XML 元素的分层集合。属性定义相应的菜单项在屏幕上的行为和外观。集合与 XML 之间可以方便地来回转换，它用于描述菜单（`menu` 标签）和项目（`menuitem` 标签）。内置的动作脚本 XML 类是 `Menu` 组件底层模型的基础。

具有两个项目的简单菜单可在 XML 中以两个菜单项子元素进行描述：

```
<menu>
  <menuitem label="Up" />
```

```
<menuitem label="Down" />
</menu>
```

注意：XML 节点的标签名称（menu 和 menuitem）不重要；属性及其嵌套关系在菜单中使用。

关于分层菜单

要创建分层菜单，应在父 XML 元素中嵌入 XML 元素，如下所示：

```
<menu>
  <menuitem label="MenuItem A" >
    <menuitem label="SubMenuItem 1-A" />
    <menuitem label="SubMenuItem 2-A" />
  </menuitem>
  <menuitem label="MenuItem B" >
    <menuitem label="SubMenuItem 1-B" />
    <menuitem label="SubMenuItem 2-B" />
  </menuitem>
</menu>
```

注意：这会将父菜单项转换为弹出式菜单锚记，因此在选中时不会生成事件。

关于菜单项 XML 属性

菜单项 XML 元素的属性确定显示的内容、菜单项的行为，以及如何向动作脚本揭示此菜单项。下表描述了 XML 菜单项的属性：

属性名称	类型	默认值	描述
label	String	undefined	代表菜单项的显示文本。此属性对所有项目类型是必需的，除了 separator 类型。
type	separator、check、radio、normal 或 undefined	undefined	菜单项类型：separator、check box、radio button 或 normal（命令或子菜单激活器）。如果此属性不存在，则默认值为 normal。
icon	String	undefined	图像资源的链接标识符。该属性不是必需的属性。此属性不适用于 check、radio 或 separator 类型。
instanceName	String	undefined	一个标识符，可以用于引用根菜单实例中的菜单项实例。例如，名为 yellow 的菜单项可以引用为 myMenu.yellow。该属性不是必需的属性。
groupName	String	undefined	一个标识符，可以用于关联单选按钮组中的几个单选按钮项，并显示根菜单实例中的单选按钮组的状态。例如，名为 colors 的单选按钮组可以引用为 myMenu.colors。只有类型 radio 才需要此属性。

属性名称	类型	默认值	描述
selected	false、true 或 false 或 true (字符串或布尔值)	false	一个布尔值，指示 check 或 radio 项是处于开启 (true) 还是关闭状态 (false)。该属性不是必需的属性。
enabled	false、true 或 false 或 true (字符串或布尔值)	true	一个布尔值，指示是 (true) 否 (false) 可以选择此菜单项。该属性不是必需的属性。

关于菜单项类型

有四种菜单项，由 type 属性指定：

```
<menu>
  <menuitem label="Normal Item" />
  <menuitem type="separator" />
  <menuitem label="Checkbox Item" type="check" instanceName="check_1"/>
  <menuitem label="RadioButton Item" type="radio" groupName="radioGroup_1" />
</menu>
```

普通菜单项

Normal Item 菜单项不具有 type 属性，这意味着 type 属性默认为 normal。Normal 项目可以是命令激活器或子菜单激活器，这取决于它们是否具有嵌套的子项目。

分隔符菜单项

type 属性设置为 separator 的菜单项用作菜单中的外观分隔符。下面的 XML 会创建三个菜单项：Top、Middle 和 Bottom，各项之间带有分隔符：

```
<menu>
  <menuitem label="Top" />
  <menuitem type="separator" />
  <menuitem label="Middle" />
  <menuitem type="separator" />
  <menuitem label="Bottom" />
</menu>
```

所有 separator 项目被禁用。单击或滑过分隔符不起作用。

复选框菜单项

type 属性设置为 check 的菜单项用作菜单中的复选框项目；当 selected 属性设置为 true 时，在菜单项的标签旁边会出现一个复选标记。当选定某个复选框时，其状态会自动切换，而且会对根菜单上的所有侦听器广播 change 事件。下面的范例定义三个复选框菜单项：

```
<menu>
  <menuitem label="Apples" type="check" instanceName="buyApples"
    selected="true" />
  <menuitem label="Oranges" type="check" instanceName="buyOranges"
    selected="false" />
```

```

        <menuitem label="Bananas" type="check" instanceName="buyBananas"
        selected="false" />
    </menu>

```

可以在动作脚本中使用实例名称直接从菜单本身访问菜单项，如下例所示：

```

myMenu.setMenuItemSelected(myMenu.buyapples, true);
myMenu.setMenuItemSelected(myMenu.buyoranges, false);

```

注意：selected 属性只能使用 setMenuItemSelected(item, b) 方法修改。可以直接检查 selected 属性，但它返回 true 或 false 的 String 值。

单选按钮菜单项

可以将 type 属性设置为 radio 的菜单项分到一组，使得一次只能选择其中一个项目。通过为菜单项指定与其 groupName 属性相同的值，可以创建单选按钮组，如下例所示：

```

<menu>
    <menuitem label="Center" type="radio" groupName="alignment_group"
        instanceName="center_item"/>
    <menuitem type="separator" />
    <menuitem label="Top" type="radio" groupName="alignment_group" />
    <menuitem label="Bottom" type="radio" groupName="alignment_group" />
    <menuitem label="Right" type="radio" groupName="alignment_group" />
    <menuitem label="Left" type="radio" groupName="alignment_group" />
</menu>

```

当用户选择一个项目时，当前的选项会自动改变，而且会对根菜单上的所有侦听器广播 change 事件。在动作脚本中，使用 selection 属性可访问单选按钮组中当前选中的项目，如下所示：

```

var selectedItem = myMenu.alignment_group.selection;
myMenu.alignment_group = myMenu.center_item;

```

每个 groupName 值在根菜单实例的范围内必须唯一。

注意：selected 属性只能使用 setMenuItemSelected(item, b) 方法修改。可以直接检查 selected 属性，但它返回 true 或 false 的 String 值。

向动作脚本揭示菜单项

可以在 instanceName 属性中为每个菜单项指定唯一的标识符，从而能从根菜单直接访问菜单项。例如，以下 XML 代码为每个菜单项提供了 instanceName 属性：

```

<menu>
    <menuitem label="Item 1" instanceName="item_1" />
    <menuitem label="Item 2" instanceName="item_2" >
        <menuitem label="SubItem A" instanceName="sub_item_A" />
        <menuitem label="SubItem B" instanceName="sub_item_B" />
    </menuitem>
</menu>

```

可以使用动作脚本直接从 Menu 组件访问相应的对象实例及其属性，如下所示：

```

var aMenuItem = myMenu.item_1;
myMenu.setMenuItemEnabled(item_2, true);
var aLabel = myMenu.sub_item_A.label;

```

注意：每个 instanceName 在根菜单组件实例（包括根菜单的所有子菜单）的范围内必须是唯一的。

关于初始化对象属性

initObject（初始化对象）参数是关于创建 Menu 组件的布局的一个基本概念。*initObject* 参数是具有属性的一个对象。每个属性代表菜单项的一个可能 XML 属性。（有关 *initObject* 参数中所允许属性的说明，请参阅第 336 页的“关于菜单项 XML 属性”。）

initObject 参数用于以下方法：

- `Menu.addItem()`
- `Menu.addItemAt()`
- `MenuDataProvider.addItem()`
- `MenuDataProvider.addItemAt()`

下面的范例创建 *initObject* 参数，它带有两个属性 - `label` 和 `instanceName`：

```
var i = myMenu.addItem({label:"myMenuItem", instanceName:"myFirstItem"});
```

几个属性可以协同工作来创建特定的菜单项类型。通过指定特定的属性来创建特定类型的菜单项（普通、分隔符、复选框或单选按钮）。

例如，可以使用以下 *initObject* 参数对普通菜单项进行初始化：

```
myMenu.addItem({label:"myMenuItem", enabled:true, icon:"myIcon",  
instanceName:"myFirstItem"});
```

可以使用以下 *initObject* 参数对分隔符菜单项进行初始化：

```
myMenu.addItem({type:"separator"});
```

可以使用以下 *initObject* 参数对复选框菜单项进行初始化：

```
myMenu.addItem({type:"check", label:"myMenuCheck", enabled:false,  
selected:true, instanceName:"myFirstCheckItem"})
```

可以使用以下 *initObject* 参数对单选按钮菜单项进行初始化：

```
myMenu.addItem({type:"radio", label:"myMenuRadio1", enabled:true,  
selected:false, groupName:"myRadioGroup" instanceName:"myFirstRadioItem"})
```

请注意，应该将菜单项的 `instanceName`、`groupName` 和 `type` 属性当作只读，这很重要。应该仅在创建项目（例如，在对 `addItem()` 的调用中）时设置这些属性。在创建后修改这些属性可能会产生不可预料的结果。

Menu 组件参数

Menu 组件没有创作参数。

可以使用 Menu 组件的属性、方法和事件编写动作脚本来控制 Menu 组件。有关详细信息，请参阅第 342 页的“Menu 类（仅限 Flash Professional）”。

创建具有 Menu 组件的应用程序

在下面的范例中，应用程序开发人员正构建一个应用程序，并使用 Menu 组件显示一些用户可以发出的命令，例如“打开”、“关闭”、“保存”等。

创建具有 Menu 组件的应用程序：

- 1 选择“文件” > “新建”，然后创建 Flash 文档。
- 2 将 Menu 组件从“组件”面板拖到舞台上并将其删除。

此操作会将 Menu 组件添加到库中，并且不会将其添加到应用程序中。菜单是使用动作脚本动态创建的。

- 3 将 Button 组件从“组件”面板拖到舞台上。
单击按钮会激活菜单。
- 4 在属性检查器中，为按钮指定实例名称 **commandBtn**，然后将其文本属性更改为**命令**。
- 5 在第一帧上的“动作”面板中，输入以下代码以添加事件侦听器，从而侦听 **commandBtn** 实例上的 click 事件：

```
var listener = new Object();
listener.click = function(evtObj){
    var button = evtObj.target;
    if(button.menu == undefined) {
        // 创建 Menu 实例并添加一些项目
        button.menu = mx.controls.Menu.createMenu();
        button.menu.addItem("Open");
        button.menu.addItem("Close");
        button.menu.addItem("Save");
        button.menu.addItem("Revert");

        // 添加更改侦听器以捕获项目选择
        var changeListener = new Object();
        changeListener.change = function(event) {
            var item = event.menuItem;
            trace("Item selected:" + item.attributes.label);
        }
        button.menu.addEventListener("change", changeListener);
    }
    button.menu.show(button.x, button.y + button.height);
}
commandBtn.addEventListener("click", listener);
```

- 6 选择“控制” > “测试影片”。
单击“命令”按钮以查看菜单是否出现。选择菜单项，以查看 trace 动作在“输出”窗口中报告选中了哪个项目。

使用服务器中的 XML 数据创建并填充菜单：

- 1 选择“文件” > “新建”，然后创建 Flash 文档。
- 2 将 Menu 组件从“组件”面板拖到舞台上并将其删除。
此操作会将 Menu 组件添加到库中，并且不会将其添加到应用程序中。菜单是使用动作脚本动态创建的。

- 3 在“动作”面板中，将以下代码添加到第一帧，以创建菜单和添加一些项目：

```
var myMenu = mx.controls.Menu.createMenu();
// 导入 XML 文件
var loader = new XML();
loader.menu = myMenu;
loader.ignoreWhite = true;
loader.onLoad = function(success) {
    // 当数据到达时，传给菜单
    if(success) {
        this.menu.dataProvider = this.firstChild;
    }
};
loader.load(url);
```

注意：菜单项由 XML 文档的第一个子项的子项进行描述。

4 选择 “控制” > “测试影片”。

使用良构的 XML 字符串创建并填充菜单：

1 选择 “文件” > “新建”，然后创建 Flash 文档。

2 将 Menu 组件从 “组件” 面板拖到舞台上并将其删除。

此操作会将 Menu 组件添加到库中，并且不会将其添加到应用程序中。菜单是使用动作脚本动态创建的。

3 在 “动作” 面板中，将以下代码添加到第一帧，以创建菜单和添加一些项目：

```
// 创建包含菜单定义的 XML 字符串
var s = "";
s += "<menu>";
s += "<menuitem label='Undo' />";
s += "<menuitem type='separator' />";
s += "<menuitem label='Cut' />";
s += "<menuitem label='Copy' />";
s += "<menuitem label='Paste' />";
s += "<menuitem label='Clear' />";
s += "<menuitem type='separator' />";
s += "<menuitem label='Select All' />";
s += "</menu>";

// 通过字符串创建 XML 对象
var xml = new XML(s);
xml.ignoreWhite = true;

// 通过 XML 对象的第一个子项创建菜单
var myMenu = mx.controls.Menu.createMenu(_root, xml.firstChild);
```

4 选择 “控制” > “测试影片”。

使用 MenuDataProvider 类创建并填充菜单：

1 选择 “文件” > “新建”，然后创建 Flash 文档。

2 将 Menu 组件从 “组件” 面板拖到舞台上并将其删除。

此操作会将 Menu 组件添加到库中，并且不会将其添加到应用程序中。菜单是使用动作脚本动态创建的。

3 在 “动作” 面板中，将以下代码添加到第一帧，以创建菜单和添加一些项目：

```
// 创建用作出厂设置的 XML 对象
var xml = new XML();

// 接下来创建的项目不会出现在菜单中。
// 'createMenu' 方法调用（见以下）预期会
// 接收到根元素，其子对象将成为
// 项目。这只是一个创建该
// 根元素并为其指定一个便利名称
// 的简单方法。
var theMenuElement = xml.addItem("Edit");

// 添加菜单项
theMenuElement.addItem({label:"Undo"});
theMenuElement.addItem({type:"separator"});
theMenuElement.addItem({label:"Cut"});
theMenuElement.addItem({label:"Copy"});
theMenuElement.addItem({label:"Paste"});
theMenuElement.addItem({label:"Clear", enabled:"false"});
theMenuElement.addItem({type:"separator"});
```

```
theMenuElement.addItem({label:"Select All"});  
// 创建 Menu 对象  
var theMenuControl = mx.controls.Menu.createMenu(_root, theMenuElement);
```

4 选择 “控制” > “测试影片”。

自定义 Menu 组件

菜单会调整其自身的大小，以在水平方向适合其最宽的文本。还可以调用 setSize() 方法调整组件大小。图标应调整到 16 个像素乘 16 个像素的最大大小。

在 Menu 组件中使用样式

可以调用 setStyle() 方法来更改菜单、其项目和子菜单的样式。Menu 组件支持下列光晕样式：

样式	描述
themeColor	菜单背景色。这是唯一不继承样式值的颜色样式。
color	菜单项的文本标签的颜色。
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；normal 或 italic。
fontWeight	字体粗细；normal 或 bold。
rolloverColor	菜单项的转滚颜色。
selectionColor	选中的项目和包含子菜单的项目。
selectionDisabledColor	选中的项目和包含子菜单且禁用的项目。
textRolloverColor	滑过项目时文本的颜色。
textDecoration	文本修饰；none 或 underline。
textDisabledColor	禁用文本的颜色。
textSelectedColor	选定菜单项的文本颜色。
popupDuration	当菜单打开时，过渡的持续时间。值为 0 表示没有过渡。

在 Menu 组件中使用外观

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 类（仅限 Flash Professional）

继承 UIObject > UIComponent > View > ScrollView > ScrollSelectList > Menu

动作脚本类名称 mx.controls.Menu

Menu 类的方法摘要

方法	描述
<code>Menu.addItem()</code>	向 Menu 添加菜单项。
<code>Menu.addItemAt()</code>	将菜单项添加到 Menu 的特定位置。
<code>Menu.createMenu()</code>	创建 Menu 类的实例。这是静态方法。
<code>Menu.getItemAt()</code>	获取对指定位置的菜单项的引用。
<code>Menu.hide()</code>	关闭菜单。
<code>Menu.indexOf()</code>	返回给定菜单项的索引。
<code>Menu.removeAll()</code>	删除菜单中的所有项目。
<code>Menu.removeItemAt()</code>	删除 Menu 中特定位置处的菜单项
<code>Menu.setMenuItemEnabled()</code>	指示菜单项是 (true) 否 (false) 启用。
<code>Menu.setMenuItemSelected()</code>	指示菜单项是 (true) 否 (false) 被选中。
<code>Menu.show()</code>	在特定的位置或上次的位置打开菜单。

继承 [UIObject](#)、[UIComponent](#)、[ScrollView](#) 和 [ScrollSelectList](#) 的所有方法。

Menu 类的属性摘要

属性	描述
<code>Menu.dataProvider</code>	菜单的数据源。

继承 [UIObject](#)、[UIComponent](#)、[ScrollView](#) 和 [ScrollSelectList](#) 的所有属性。

Menu 类的事件摘要

事件	描述
<code>Menu.change</code>	当用户选择项目时进行广播。
<code>Menu.menuHide</code>	当菜单关闭时进行广播。
<code>Menu.menuShow</code>	当菜单打开时进行广播。
<code>Menu.rollOut</code>	当指针滑离项目时进行广播。
<code>Menu.rollOver</code>	当指针滑过项目时进行广播。

继承 [UIObject](#)、[UIComponent](#)、[ScrollView](#) 和 [ScrollSelectList](#) 的所有事件

Menu.addItem()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1 :

```
myMenu.addMenu(initObject)
```

用法 2 :

```
myMenu.addMenu(childMenuItem)
```

参数

initObject 一个对象，它包含对菜单项的属性进行初始化的属性。请参阅第 336 页的“[关于菜单项 XML 属性](#)”。

childMenuItem 一个 XML 节点对象。

返回

对添加的 XML 节点的引用。

说明

方法 ; 用法 1 在菜单末尾添加菜单项。该菜单项是根据在 *initObject* 参数中提供的值构建的。用法 2 在菜单末尾添加菜单项，该菜单项是预先建立的 XML 节点（为 XML 对象形式）。添加预先存在的节点会将该节点从其先前的位置上删除。

示例

用法 1 : 下面的范例将菜单项加到菜单中 :

```
myMenu.addItem({ label:"Item 1", type:"radio", selected:false,  
    enabled:true, instanceName:"radioItem1", groupName:"myRadioGroup" });
```

用法 2 : 下面的范例将一个菜单中的某个节点移到另一个菜单的根 :

```
myMenu.addItem(mySecondMenu.getItemAt(mySecondMenu, 3));
```

Menu.addItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1 :

```
myMenu.addItemAt(index, initObject)
```

用法 2 :

```
myMenu.addItemAt(index, childMenuItem)
```

参数

index 一个整数，指示项目的添加顺序（在多个子节点之间）。

initObject 一个对象，它包含对菜单项的属性进行初始化的属性。请参阅第 336 页的“[关于菜单项 XML 属性](#)”。

childMenuItem 一个 XML 节点对象。

返回

对添加的 XML 节点的引用。

说明

方法；用法 1 在菜单中的指定位置添加菜单项（子节点）。该菜单项是根据在 *initObject* 参数中提供的值构建的。用法 2 在菜单中的指定位置添加菜单项，该菜单项是预先建立的 XML 节点（为 XML 对象形式）。添加预先存在的节点会将该节点从其先前的位置上删除。

示例

用法 1：以下范例将一个新节点添加为菜单根的第二个子项：

```
myMenu.addItemAt(1, { label:"Item 1", instanceName:"radioItem1",  
    type:"radio", selected:false, enabled:true, groupName:"myRadioGroup" } );
```

用法 2：下面的范例将一个菜单中的某个节点移到另一个菜单的根的第四个子项：

```
myMenu.addItemAt(3, mySecondMenu.getItemAt(mySecondMenu, 3));
```

Menu.change

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

描述

事件；每当用户造成菜单更改时对所有注册的侦听器进行广播。

V2 组件使用调度程序 - 侦听器事件模型。在 Menu 组件广播 change 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 *addEventListener()* 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Menu.change 事件的事件对象具有以下附加属性：

- *menuBar* 对作为目标 Menu 的父级的 MenuBar 实例的引用。当作为目标的 Menu 不属于 Menu 时，此值为未定义。
- *menu* 对目标项目所在的 Menu 实例的引用。
- *menuItem* 一个 XML 节点，是选中的菜单项。
- *groupName* 一个字符串，指示项目所属的单选按钮组的名称。如果项目未位于单选按钮组中，此值为未定义。

有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `listener` 的处理函数，并将其传递到 `myMenu.addEventListener()` 方法，作为第二个参数。`change` 处理函数在 `event` 参数中捕获事件对象。在广播 `change` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
listener = new Object();
listener.change = function(evt){
    trace("Menu item chosen:"+evt.menuItem.attributes.label);
}
myMenu.addEventListener("change", listener);
```

Menu.createMenu()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`Menu.createMenu(parent, mdp)`

参数

parent `MovieClip` 实例。影片剪辑是包含新 `Menu` 实例的父组件。此参数是可选的。

mdp 描述此 `Menu` 实例的 `MenuDataProvider` 实例。此参数是可选的。

返回

对新的 `Menu` 实例的引用。

说明

方法（静态）；对 `Menu` 实例进行实例化，并且选择性地将其附加到指定的父项，以指定的 `MenuDataProvider` 作为菜单项的数据源。

如果 *parent* 参量被省略或为 `null`，则 `Menu` 被附加到 `_root` 时间轴。

如果 *mdp* 参量被省略或为 `null`，则菜单没有菜单项；必须调用 `addMenu()` 或 `setDataProvider()` 方法来填充菜单。

示例

在下面的范例中，第 1 行创建一个 `MenuDataProvider`，它是用 `MenuDataProvider` 类的方法修饰的 XML 对象。下一行添加带有子菜单（“文件”、“项目”和“资源”）的菜单项（“新建”）。下一个代码块向主菜单添加更多项目。第三个代码块创建附加到 `myParentClip` 的空菜单，并用数据源 `myMDP` 填充该菜单，然后在坐标 100, 20 处打开它，如下所示：

```
var myMDP = new XML();

var newItem = myMDP.addMenuItem({label:"New"});
newItem.addMenuItem({label:"File..."});
newItem.addMenuItem({label:"Project..."});
newItem.addMenuItem({label:"Resource..."});

myMDP.addMenuItem({label:"Open", instanceName:"miOpen"});
myMDP.addMenuItem({label:"Save", instanceName:"miSave"});
myMDP.addMenuItem({type:"separator"});
```

```
myMDP.addItem({label:"Quit", instanceName:"miQuit"});

var myMenu = mx.controls.Menu.createMenu(myParentClip, myMDP);

myMenu.show(100, 20);
```

要测试此代码，将其放到主时间轴的第 1 帧上的“动作”面板中。将 Menu 组件从“组件”面板拖到舞台上并将其删除。这会将其添加到库中，而不放到文档中。

Menu.dataProvider

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.dataProvider
```

描述

属性；Menu 组件中项目的数据源。

Menu.dataProvider 是 XML 节点对象。设置此属性会替换 Menu 的现有数据源。

默认值未定义。

注意：所有 XML 或 XMLNode 实例在与 Menu 组件一起使用时，会自动接收 MenuDataProvider API 的方法和属性。

范例

下面的范例导入 XML 文件，并将其指定给 Menu.dataProvider 属性：

```
var myMenuDP = new XML();
myMenuDP.load("http://myServer.myDomain.com/source.xml");
myMenuDP.onLoad = function(){
    myMenuControl.dataProvider = myMenuDP;
}
```

Menu.getItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.getItemAt(index)
```

参数

index 一个整数，指示菜单中节点的索引。

返回

对指定节点的引用。

说明

方法；返回对菜单的指定子节点的引用。

示例

下面的范例获取对 `myMenu` 中的第二个子节点的引用，并将值复制到变量 `myItem` 中：

```
var myItem = myMenu.getMenuItemAt(1);
```

Menu.hide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.hide()
```

参数

index `Menu` 项的索引。

返回

无。

说明

方法；关闭具有可选过渡效果的菜单。

示例

下面的范例回缩展开的菜单：

```
myMenu.hide();
```

另请参见

[Menu.show\(\)](#)

Menu.indexOf()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.indexOf(item)
```

参数

item 对描述菜单项的 XML 节点的引用。

返回

指定的菜单项的索引，或者如果项目不属于此菜单，则为未定义。

说明

方法；返回此菜单实例中的指定菜单项的索引。

示例

下面的范例向父项目添加菜单项，然后获取此项目在其父项目中的索引：

```
var myItem = myMenu.addItem({label:"That item"});  
var myIndex = myMenu.indexOf(myItem);
```

Menu.menuHide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.menuHide = function(eventObject){  
    // 此处插入您的代码  
}  
myMenu.addEventListener("menuHide", listenerObject)
```

描述

事件；每当菜单关闭时，向所有已注册的侦听器广播。

V2 组件使用调度程序 - 侦听器事件模型。在 Menu 组件调度 menuHide 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法，并将处理函数的名称和侦听器对象的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Menu.menuHide 事件的事件对象具有两个附加属性：

- `menuBar` 对作为目标 Menu 的父项的 MenuBar 实例的引用。当作为目标的 Menu 不属于 MenuBar 时，此值为未定义。
- `menu` 对隐藏的 Menu 实例的引用。

有关事件对象的详细信息，请参阅[第 510 页的“事件对象”](#)。

范例

下面的范例定义了名为 `form` 的处理函数，并将其作为第二个参数传递到

`myMenu.addEventListener()` 方法。`menuHide` 处理函数在 `event` 参数中捕获事件对象。在广播 menuHide 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
form = new Object();  
form.menuHide = function(evt){
```

```
        trace("Menu closed:"+evt.menu);
    }
    myMenu.addEventListener("menuHide", form);
```

另请参见

[Menu.menuShow](#)

Menu.menuShow

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.menuShow = function(eventObject){
    // 此处插入您的代码
}
myMenu.addEventListener("menuShow", listenerObject)
```

描述

事件；当菜单打开时，向所有已注册的侦听器广播。所有父节点打开菜单以显示其子项。

V2 组件使用调度程序 - 侦听器事件模型。在 Menu 组件调度 menuShow 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法，并将处理函数的名称和侦听器对象作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Menu.menuShow 事件的事件对象具有两个附加属性：

- `menuBar` 对作为目标 Menu 的父级的 MenuBar 实例的引用。当作为目标的 Menu 不属于 Menu 时，此值为未定义。
- `menu` 对显示的 Menu 实例的引用。

有关事件对象的详细信息，请参阅[第 510 页的“事件对象”](#)。

范例

下面的范例定义了名为 `form` 的处理函数，并将其作为第二个参数传递到 `myMenu.addEventListener()` 方法。menuShow 处理函数在 `eventObject` 参数中捕获事件对象。在广播 menuShow 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
form = new Object();
form.menuShow = function(evt){
    trace("Menu opened:"+evt.menu);
}
myMenu.addEventListener("menuShow", form);
```

另请参见

[Menu.menuHide](#)

Menu.removeAll()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.removeAll()
```

参数

无。

返回

无。

说明

方法；删除所有项目并刷新菜单。

示例

下面的范例删除菜单中的所有节点：

```
myMenu.removeAll();
```

Menu.removeItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.removeItemAt(index)
```

参数

index 要删除的菜单项的索引。

返回

对返回的菜单项（XML 节点）的引用。如果在该位置不存在任何项目，则该值是未定义的。

说明

方法；删除指定索引处的菜单项及其所有子项。如果在该索引不存在任何菜单项，则调用此方法没有效果。

示例

下面的范例删除索引为 3 的菜单项：

```
var item = myMenu.removeItemAt(3);
```

Menu.rollOut

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.rollOut = function(eventObject){
    // 此处插入您的代码
}
myMenu.addEventListener("rollOut", listenerObject)
```

描述

事件；当指针滑离菜单项时，向所有已注册的侦听器广播。

V2 组件使用调度程序 - 侦听器事件模型。在 Menu 组件广播 rollOut 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Menu.rollOut 事件的事件对象具有一个附加属性：

- `menuItem` 对指针滑离的菜单项（XML 节点）的引用。

有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

范例

下面的范例定义了名为 `form` 的处理函数，并将其作为第二个参数传递到 `myMenu.addEventListener()` 方法。rollOut 处理函数在 `event` 参数中捕获事件对象。在广播 rollOut 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
form = new Object();
form.rollOut = function(evt){
    trace("Menu rollOut:"+evt.menuItem.attributes.label);
}
myMenu.addEventListener("rollOut", form);
```

Menu.rollOver

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.rollOver = function(eventObject){
    // 此处插入您的代码
}
myMenu.addEventListener("rollOver", listenerObject)
```


描述

事件；当指针滑过菜单项时，向所有已注册的侦听器广播。

V2 组件使用调度程序 - 侦听器事件模型。在 Menu 组件广播 `change` 事件时，该事件由附加到您创建的侦听器对象 (*listenerObject*) 的函数（也称作处理函数）处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Menu.rollOver 事件的事件对象具有一个附加属性：

`menuItem` 对指针滑过的菜单项（XML 节点）的引用。

有关事件对象的详细信息，请参阅[第 510 页的“事件对象”](#)。

范例

下面的范例定义了名为 `form` 的处理函数，并将其作为第二个参数传递到 `myMenu.addEventListener()` 方法。rollOver 处理函数在 `event` 参数中捕获事件对象。在广播 rollOver 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
form = new Object();
form.rollOver = function(evt){
    trace("Menu rollOver:"+evt.menuItem.attributes.label);
}
myMenu.addEventListener("rollOver", form);
```

Menu.setMenuItemEnabled()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.setMenuItemEnabled(item, enable)
```

参数

item 一个 XML 节点。数据提供程序内目标菜单项的节点。

enable 一个布尔值，它表明项目是 (`true`) 否 (`false`) 启用。

返回

无。

说明

方法；将目标项目的 `enabled` 属性更改为 `enable` 参数给定的状态。如果此调用导致状态更改，则用新状态重绘此项目。

示例

下面的范例禁用 `myMenu` 的第二个子对象：

```
var myItem = myMenu.getMenuItemAt(1);
myMenu.setMenuItemEnabled(myItem, false);
```

另请参见

`Menu.setMenuItemSelected()`

Menu.setMenuItemSelected()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.setMenuItemSelected(item, select)
```

参数

item 一个 XML 节点。数据提供程序内目标菜单项的节点。

select 一个布尔值，它表明项目是 (true) 否 (false) 被选中。如果项目是复选框，则其复选框为可见或不可见。如果项目为单选按钮，则项目成为单选按钮组中的当前选中项。

返回

无。

说明

方法；将项目的 `selected` 属性更改为 `select` 参数所指定的状态。如果此调用导致状态更改，则用新状态重绘此项目。这仅对 `type` 属性设置为 "radio" 或 "check" 的项目有意义，原因是这会使它们的点或复选标记出现或消失。对 `type` 为 "normal" 或 "separator" 的项目调用此方法是无效的。

示例

下面的范例取消选中 `myMenu` 的第二个子项：

```
var myItem = myMenu.getMenuItemAt(1);  
myMenu.setMenuItemSelected(myItem, false);
```

Menu.show()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.show(x, y)
```

参数

x *x* 坐标。

y *y* 坐标。

返回

无。

说明

方法；打开指定位置的菜单。菜单会自动重新调整大小，使其所有顶层项目都可见，而且左上角放在由组件的父项提供的坐标系中的给定位置。

如果省略了 *x* 和 *y* 参数，则菜单在其上次的位置显示。

示例

下面的范例会扩展菜单：

```
myMenu.show(10, 10);
```

另请参见

```
Menu.hide()
```

MenuDataProvider 类

MenuDataProvider 类是修饰器（混合）API，它向 XMLNode 全局类添加功能。此功能使指定给 Menu dataProvider 属性的 XML 实例可通过 MenuDataProvider API 处理其数据以及关联的 Menu 视图。

重要概念：

- MenuDataProvider 是修饰器（混合）API。无需对其实例化即可使用。
- 菜单以本机方式将 XML 作为 dataProvider 属性接收。
- 如果实例化 Menu 类，SWF 文件中的所有 XML 实例会由 MenuDataProvider API 进行修饰。
- 只有 MenuDataProvider API 方法向 Menu 控件广播事件。您仍然可以使用本机 XML 方法，但它们不是刷新 Menu 视图的广播事件。
 - 使用 MenuDataProvider API 方法可控制数据模型。
 - 将 XML 方法用于只读操作，例如在 Menu 层次结构中移动。
- Menu 中的所有项目是用 MenuDataProvider API 修饰的 XML 对象。
- 对项目属性的更改在重绘之前不会在屏幕菜单中反映出来。

MenuDataProvider 类的方法摘要

方法	描述
<code>MenuDataProvider.addItem()</code>	添加子项目。
<code>MenuDataProvider.addItemAt()</code>	在指定的位置添加子项目。
<code>MenuDataProvider.getItemAt()</code>	获取对指定位置的菜单项的引用。
<code>MenuDataProvider.indexOf()</code>	返回指定菜单项的索引。
<code>MenuDataProvider.removeItem()</code>	删除菜单项。
<code>MenuDataProvider.removeItemAt()</code>	删除指定位置处的菜单项。

MenuDataProvider.addItem()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
myMenu.addItem(initObject)
```

用法 2：

```
myMenu.addItem(childMenuItem)
```

参数

initObject 一个对象，包含初始化菜单项的属性的特定属性。有关详细信息，请参阅[第 336 页的“关于菜单项 XML 属性”](#)。

childMenuItem 一个 XML 节点。

返回

对 XMLNode 对象的引用。

说明

方法；用法 1 将子项目添加到父菜单项（可以是菜单本身）的末尾。该菜单项根据在 *initObject* 参数中传递的值构建。用法 2 将在指定的 XML *childMenuItem* 参数中定义的子项目添加到父菜单项的末尾。

示例

下面的范例将新节点添加到菜单中的指定节点处：

```
myMenuDP.firstChild.addItem("Inbox", { label:"Item 1",  
    icon:"radioItemIcon", type:"radio", selected:false, enabled:true,  
    instanceName:"radioItem1", groupName:"myRadioGroup" });
```

MenuDataProvider.addItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
myMenu.addItemAt(index, initObject)
```

用法 2：

```
myMenu.addItemAt(index, childMenuItem)
```

参数

index 一个整数。

initObject 一个对象，包含初始化菜单项的属性的特定属性。有关详细信息，请参阅第 336 页的“关于菜单项 XML 属性”。

childMenuItem 一个 XML 节点。

返回

对添加的 XML 节点的引用。

说明

方法；用法 1 将子项目添加到父菜单项（可以是菜单本身）中的指定索引位置。该菜单项根据在 *initObject* 参数中传递的值构建。用法 2 将在指定的 XML *childMenuItem* 参数中定义的子项目添加到父菜单项的指定索引处。

示例

用法 1：以下范例将一个新节点添加为菜单根的第二个子项：

```
myMenu.addItemAt(1, { label:"Item 1", type:"radio", selected:false,  
    enabled:true, instanceName:"radioItem1", groupName:"myRadioGroup" });
```

MenuDataProvider.getItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.getItemAt(index)
```

参数

index 一个整数，指明菜单的位置。

返回

对指定的 XML 节点的引用。

说明

方法；返回对当前菜单项的指定子菜单项的引用。

示例

下面的范例查找要获取的节点，然后获取 *myMenuItem* 的第二个子项：

```
var myMenuItem = myMenuDP.firstChild.firstChild;  
myMenuItem.getItemAt(1);
```

MenuDataProvider.indexOf()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.indexOf(item)
```

参数

item 对描述菜单项的 XML 节点的引用。

返回

指定的菜单项的索引；如果项目不属于此菜单，则返回未定义。

说明

方法；返回此父菜单项中的指定菜单项的索引。

示例

下面的范例向父项目添加菜单项，并获取该项目的索引：

```
var myItem = myParentItem.addMenuItem({label:"That item"});  
var myIndex = myParentItem.indexOf(myItem);
```

MenuDataProvider.removeItem()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.removeItem()
```

参数

无。

返回

对删除的菜单项（XML 节点）的引用；如果出错则返回未定义。

说明

方法；删除目标项目和任何子节点。

示例

下面的范例将 myMenuItem 从其父项中删除：

```
myMenuItem.removeItem();
```

MenuDataProvider.removeItemAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenu.removeItemAt(index)
```

参数

index Menu 项的索引。

返回

对删除的菜单项的引用。如果在该位置不存在任何项目，则该值是未定义的。

说明

方法；删除 *index* 参数指定的菜单项的子项目。如果在该索引不存在任何菜单项，则调用此方法没有效果。

示例

下面的范例删除第四个项目：

```
myMenuDP.removeItemAt(3);
```

MenuBar 组件（仅限 Flash Professional）

MenuBar 组件使您可以创建带有弹出式菜单和命令的水平菜单栏，就像最常见的软件应用程序（例如 Macromedia Flash）中的“文件”和“编辑”菜单栏一样。菜单栏对 Menu 组件进行了补充，方法是通过提供可单击的界面来显示和隐藏菜单，而这些菜单起到了组合鼠标和键盘交互性操作的作用。

菜单栏使您可以通过几个步骤创建应用程序菜单。要构建菜单栏，可以向描述一系列菜单的菜单栏指定 XML 数据提供程序，或者使用 `MenuBar.addMenu()` 方法一次添加一个菜单实例。

菜单栏中的每个菜单都由两部分组成：菜单和使菜单打开的按钮（称为菜单激活器）。这些可单击的菜单激活器作为文本标签出现在菜单栏中，并带有边框凹下和凸起的突出显示状态，这些状态响应来自鼠标和键盘的交互操作。

按下菜单激活器之后，相应的菜单会在其下面打开。菜单会保持活动状态，直到再次按下激活器，或选择了某个菜单项或者在菜单区域外进行了单击。

除了创建显示和隐藏菜单的菜单激活器外，菜单栏还在一系列菜单之间创建组行为。这使用户可以浏览许多命令选项，方法是滑过一系列激活器或使用箭头键在列表中移动。鼠标和键盘交互性操作协同工作，使用户可以在 MenuBar 组件内的菜单之间跳转。

用户不能在菜单栏上的菜单之间滚动。如果菜单超过了菜单栏的宽度，则会被遮盖。

不能使用屏幕阅读器访问 MenuBar 组件。

与 MenuBar 组件进行交互（仅限 Flash Professional）

可以使用鼠标和键盘与 MenuBar 组件进行交互。

滑过菜单激活器会在激活器标签周围显示凸起的边框高亮区。

当 MenuBar 实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

按键	描述
向下箭头	将选区下移一个菜单行。
向上箭头	将选区上移一个菜单行。
右箭头	将选区移到下一个按钮。
左箭头	将选区移到上一个按钮。
Enter/Escape 键	关闭打开的菜单。

注意：如果菜单打开，不能通过按 Tab 键关闭它。必须要进行选择，或者按 Escape 键关闭菜单。

使用 MenuBar 组件（仅限 Flash Professional）

可以使用 MenuBar 组件将一组菜单（例如，“文件”、“编辑”、“特殊”、“窗口”等）添加到应用程序的顶部。

MenuBar 组件参数

以下列出了一些创作参数，您可以在属性检查器或“组件检查器”面板中为每个 MenuBar 组件实例设置这些参数：

labels 一个数组，将菜单激活器添加到带有给定标签的 MenuBar。默认值为 []（空数组）。

您可以编写动作脚本，以便使用 MenuBar 组件的属性、方法和事件来控制该组件的这些和其他选项。有关详细信息，请参阅第 362 页的“MenuBar 类”。

创建具有 MenuBar 组件的应用程序

在本例中，您会将 MenuBar 组件拖到舞台上，添加代码以便用菜单填充实例，并将侦听器附加到菜单以便响应菜单项选择。

在应用程序中使用 MenuBar 组件：

- 1 选择“文件”>“新建”，创建新的 Flash 文档。
- 2 将 MenuBar 组件从“组件”面板拖到舞台中。
- 3 将菜单放置在舞台的顶部，形成标准布局。
- 4 选择 MenuBar，并在属性检查器中输入实例名称 **myMenuBar**。
- 5 在“动作”面板中的第 1 帧上，输入以下代码：

```
var menu = myMenuBar.addMenu("File");
menu.addMenuItem({label:"New", instanceName:"newInstance"});
menu.addMenuItem({label:"Open", instanceName:"openInstance"});
menu.addMenuItem({label:"Close", instanceName:"closeInstance"});
```

此代码向菜单栏实例添加“文件”菜单。然后，它使用 Menu API 添加三个菜单项：“新建”、“打开”和“关闭”。

- 6 在“动作”面板中的第 1 帧上，输入以下代码：

```
var listen = new Object();
listen.change = function(evt){
    var menu = evt.menu;
    var item = evt.menuItem
    if (item == menu.newInstance){
        myNew();
    }
}
```



```
        trace(item);
    }else if (item == menu.openInstance){
        myOpen()
        trace(item);
    }
}
menu.addEventListener("change",listen);
```

此代码创建侦听器对象 `listen`，此对象使用事件对象 `evt` 捕获菜单项选择。

注意：您必须调用 `addEventListener` 方法将侦听器注册到菜单实例（而不是菜单栏实例）。

7 选择“控制” > “测试影片”，测试 `MenuBar` 组件。

自定义 MenuBar 组件（仅限 Flash Professional）

此组件根据通过 `dataProvider` 属性提供的激活器标签或 `MenuBar` 类的方法调整自身的大小。当激活器按钮在菜单栏中时，它会保持固定的大小（取决于字体样式和文本长度）。

在 MenuBar 组件中使用样式

`MenuBar` 为组中的每个菜单创建一个激活器标签。可以使用样式更改激活器标签的外观。`MenuBar` 组件支持下列光晕样式：

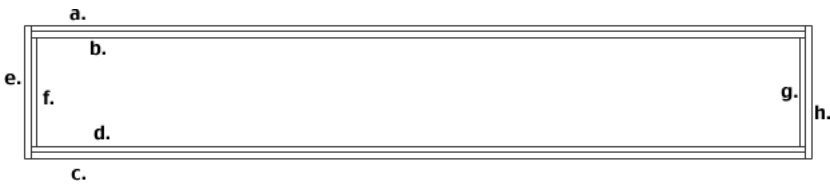
样式	描述
<code>themeColor</code>	选区突出显示颜色。这是唯一不继承样式值的颜色样式。可能的值包括“ <code>haloGreen</code> ”、“ <code>haloBlue</code> ”和“ <code>haloOrange</code> ”。
<code>color</code>	组件标签的文本。
<code>disabledColor</code>	禁用的文本颜色。
<code>fontFamily</code>	文本的字体名称。
<code>fontSize</code>	字体的磅值。
<code>fontStyle</code>	字体样式；“常规”或“斜体”。
<code>fontWeight</code>	字体粗细；“常规”或“粗体”。
<code>textDecoration</code>	文本修饰；“无”或“下划线”。
<code>popupDuration</code>	弹出菜单所用的时间量（以毫秒为单位）。默认值为 0。

`MenuBar` 组件还在用户与其交互时使用 `RectBorder` 类在标签周围绘制凹下和凸起的加亮区。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
<code>borderColor</code>	a
<code>highlightColor</code>	b
<code>borderColor</code>	c
<code>shadowColor</code>	d
<code>borderCapColor</code>	e

RectBorder 样式	边框位置
shadowCapColor	f
shadowCapColor	g
borderCapColor	h

这些样式属性设置边框上的下列位置：



在 MenuBar 组件中使用外观

MenuBar 组件使用 Menu 组件的外观来表示其外观状态。有关 Menu 组件外观的详细信息，请参阅第 342 页的“在 Menu 组件中使用外观”。

MenuBar 类

继承 UIObject > UIComponent > MenuBar

动作脚本类名称 mx.controls.MenuBar

MenuBar 类的方法摘要

方法	描述
MenuBar.addMenu()	将菜单添加到菜单栏。
MenuBar.addMenuAt()	将菜单添加到菜单栏的特定位置。
MenuBar.getMenuAt()	获取对位于指定位置的菜单的引用。
MenuBar.getMenuEnabledAt()	返回一个布尔值，指示菜单是 (true) 否 (false) 启用。
MenuBar.removeMenuAt()	删除位于菜单栏特定位置的菜单。
MenuBar.setMenuEnabledAt()	一个布尔值，指示菜单是 (true) 否 (false) 启用。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有方法。

MenuBar 类的属性摘要

属性	描述
MenuBar.dataProvider	菜单栏的数据模型。
MenuBar.labelField	一个字符串，确定将各个 XMLNode 的哪个属性用作菜单栏项目的标签文本。
MenuBar.labelFunction	一个函数，确定显示什么内容来作为每个菜单栏项目的标签。

继承 [UIObject](#) 和 [UIComponent](#) 中的所有方法。

MenuBar.addMenu()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
myMenuBar.addMenu(label)
```

用法 2：

```
myMenuBar.addMenu(label, menuDataProvider)
```

参数

label 一个字符串，指明新菜单的标签。

menuDataProvider 描述菜单及其项目的 XML 或 XMLNode 实例。如果该值是 XML 实例，则使用该实例的 `firstChild`。

返回

指向新的 Menu 对象的引用。

说明

方法；用法 1 使用在 *label* 参数中指定的值将一个菜单和菜单激活器添加到菜单栏末尾。用法 2 添加在指定的 XML *menuDataProvider* 参数中定义的一个菜单和菜单激活器。

示例

用法 1：下面的范例添加“文件”菜单，然后使用 `Menu.addItem()` 方法添加菜单项“新建”和“打开”：

```
menu = myMenuBar.addMenu("File");
menu.addItem({label:"New", instanceName:newInstance});
menu.addItem({label:"Open", instanceName:"openInstance"});
```

用法 2：下面的范例添加“字体”菜单，该菜单具有在 `menuDataProvider myMenuDP2` 中定义的菜单项“粗体”和“斜体”：

```
var myMenuDP2 = new XML();
myMenuDP2.addItem({type:"check", label:"Bold", instanceName:"check1"});
myMenuDP2.addItem({type:"check", label:"Italic", instanceName:"check2"});
menu = myMenuBar.addMenu("Font",myMenuDP2);
```

MenuBar.addMenuAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
myMenuBar.addMenuAt(index, label)
```

用法 2：

```
myMenuBar.addMenuAt(index, label, menuDataProvider)
```

参数

index 一个整数，指示菜单应插入的位置。第一个位置为 0。要加到菜单的末尾，请调用 `MenuBar.addMenu(label)`。

label 一个字符串，指明新菜单的标签。

menuDataProvider 描述菜单的 XML 或 XMLNode 实例。如果该值是 XML 实例，则使用该实例的 `firstChild`。

返回

指向新的 Menu 对象的引用。

说明

方法；用法 1 将一个菜单和菜单激活器添加到指定的 *index* 处，而值在 *label* 参数中指定。用法 2 在指定的索引处添加一个菜单和带标签的菜单激活器。菜单的内容在 *menuDataProvider* 参数中定义。

示例

用法 1：下面的范例在所有 MenuBar 菜单的左侧放置一个菜单：

```
menu = myMenuBar.addMenuAt(0,"Toreador");
menu.addMenuItem("About Macromedia Flash", instanceName:"aboutInst");
menu.addMenuItem("Preferences", instanceName:"PrefInst");
```

用法 2：下面的范例添加“编辑”菜单，该菜单具有在 *menuDataProvider* myMenuDP 中定义的菜单项“撤消”、“重做”、“剪切”和“复制”。

```
var myMenuDP = new XML();
myMenuDP.addMenuItem({label:"Undo", instanceName:"undoInst"});
myMenuDP.addMenuItem({label:"Redo", instanceName:"redoInst"});
myMenuDP.addMenuItem({type:"separator"});
myMenuDP.addMenuItem({label:"Cut", instanceName:"cutInst"});
myMenuDP.addMenuItem({label:"Copy", instanceName:"copyInst"});

myMenuBar.addMenuAt(0,"Edit",myMenuDP);
```

MenuBar.dataProvider

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.dataProvider
```

描述

属性；MenuBar 组件中的项目的数据模型。

menuBar.dataProvider 是 XML 节点对象。设置此属性会替换 MenuBar 组件的现有数据模型。数据提供程序可能具有的任何子节点用作菜单栏自身的项目；这些子节点的任何子节点用作其相应菜单的项目。

默认值未定义。

注意：所有 XML 或 XMLNode 实例在与 MenuBar 组件一起使用时，会自动接收 MenuDataProvider API 的方法和属性。

范例

下面的范例导入 XML 文件，并将其指定给 MenuBar.dataProvider 属性：

```
var myMenuBarDP = new XML();
myMenuBarDP.load("http://myServer.myDomain.com/source.xml");
myMenuBarDP.onLoad = function(success){
    if(success) {
        myMenuBar.dataProvider = myMenuBarDP;
    } else {
        trace("error loading XML file");
    }
}
```

MenuBar.getMenuAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.getMenuAt(index)
```

参数

index 一个整数，指明菜单的位置。

返回

对指定索引处的菜单的引用。如果该位置没有菜单，此值为未定义。

说明

方法；返回对指定索引处的菜单的引用。

示例

因为 getMenuAt() 方法返回一个实例，所以可以将项目添加到菜单的指定索引处。在下面的范例中，在使用 Label 创作参数创建菜单激活器“文件”、“编辑”和“视图”之后，以下代码会在运行时将“新建”和“打开”菜单项添加到“文件”菜单：

```
menu = myMenuBar.getMenuAt(0);
menu.addItem({label:"New",instanceName:"newInst"});
menu.addItem({label:"Open",instanceName:"openInst"});
```

MenuBar.getMenuEnabledAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.getMenuEnabledAt(index)
```

参数

index MenuBar 项的索引。

返回

一个布尔值，指示是 (true) 否 (false) 可以选择此菜单。

说明

方法；返回一个布尔值，指示是 (true) 否 (false) 可以选择此菜单。

示例

下面的范例对 myMenuBar 的第一个位置中的菜单调用此方法：

```
myMenuBar.getMenuEnabledAt(0);
```

MenuBar.labelField

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.labelField
```

描述

属性；一个字符串，确定将每个 XML 节点的哪个属性用作菜单的标签文本。此属性还会传递到从菜单栏创建的任何菜单。默认值为 "label"。

在设置 dataProvider 属性后，此属性为只读。

范例

下面的范例使用每个节点的名 `name` 属性作为标签文本：

```
myMenuBar.labelField = "name";
```

MenuBar.labelFunction

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.labelFunction
```

描述

属性；一个函数，确定在每个菜单的标签文本中显示什么内容。此函数接收与项目关联的 XML 节点作为参数，并返回要用作标签文本的字符串。此属性会传递到从菜单栏创建的任何菜单。默认值为未定义。

在设置 `dataProvider` 属性后，此属性为只读。

范例

下面的范例通过节点属性构建自定义的标签：

```
myMenuBar.labelFunction = function(node){  
    var a = node.attributes;  
    return "The Price for " + a.name + " is " + a.price;  
};
```

MenuBar.removeMenuAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.removeMenuAt(index)
```

参数

index MenuBar 项的索引。

返回

对返回的 MenuBar 项的引用。如果在该位置不存在任何项目，则该值为未定义。

说明

方法；删除指定索引处的菜单。如果在该索引不存在任何菜单项，则调用此方法没有效果。

示例

下面的范例删除索引为 4 的菜单：

```
myMenuBar.removeMenuAt(4);
```

MenuBar.setEnabledAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myMenuBar.setEnabledAt(index, boolean)
```

参数

- index* 要设置的 MenuBar 项的索引。
- boolean* 一个布尔值，指示指定索引处的菜单项是 (true) 否 (false) 启用。

返回

无。

说明

方法；启用给定索引处的菜单。如果在该索引处不存在任何菜单，则调用此方法无效。

示例

下面的范例获取索引为 3 的 MenuBarColumn 对象：

```
myMenuBar.setEnabledAt(3);
```

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 248 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”。

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和底部。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 25 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
<code>themeColor</code>	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“ <code>haloGreen</code> ”、“ <code>haloBlue</code> ”和“ <code>haloOrange</code> ”。
<code>color</code>	组件标签的文本。
<code>disabledColor</code>	禁用的文本颜色。
<code>fontFamily</code>	文本的字体名称。
<code>fontSize</code>	字体的磅值。
<code>fontStyle</code>	字体样式；“常规”或“斜体”。
<code>fontWeight</code>	字体粗细；“常规”或“粗体”。
<code>textDecoration</code>	文本修饰；“无”或“下划线”。
<code>textAlign</code>	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 和 UIComponent 中的所有方法。

NumericStepper 类的属性摘要

属性	描述
<code>NumericStepper.maximum</code>	一个指明最大范围值的数字。
<code>NumericStepper.minimum</code>	一个指明最小范围值的数字。
<code>NumericStepper.nextValue</code>	一个指明下一个连续值的数字。该属性为只读。
<code>NumericStepper.previousValue</code>	一个指明前一个连续值的数字。该属性为只读。
<code>NumericStepper.stepSize</code>	一个显示每一步长的变化单位的数字。
<code>NumericStepper.value</code>	一个显示步进器当前值的数字。

继承 [UIObject](#) 和 [UIComponent](#) 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](#) 和 [UIComponent](#) 的所有事件。

NumericStepper.change

可用性

Flash Player 6 版本 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，如果把下列代码附加到步进器 `myStepper`，它将把 “_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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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，将初始值设置为 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 版本 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，将初始值设置为 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 版本 79。

版本

Flash MX 2004。

用法

```
stepperInstance.stepSize
```

描述

属性；要从当前值改变的单元数量。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 版本 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

动作脚本类名称 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
<code>PopUpManager.createPopUp()</code>	创建弹出窗口。
<code>PopUpManager.deletePopUp()</code>	删除由对 <code>PopUpManager.createPopUp()</code> 的调用创建的弹出窗口。

PopUpManager.createPopUp()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004 and Flash MX Professional 2004

用法

```
PopUpManager.createPopUp(parent, class, modal [, initobj, outsideEvents])
```

参数

parent 一个引用，引用弹出窗口所基于的窗口。

class 一个引用，引用要创建的对象类。

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 版本 79。

版本

Flash MX 2004 and 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 `getBytesTotal` 方法（轮询模式）。您也可以在手动模式下使用 `ProgressBar` 组件，方法是：手动设置 `maximum`、`minimum` 和 `indeterminate` 属性，并调用 `ProgressBar.setProgress()` 方法。

`ProgressBar` 参数

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 与进度栏相关的标签位置。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 289 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”面板上拖到舞台上。
- 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.contentPath = "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 25 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](#)

动作脚本类名称 `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](#) 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，其进度由进度栏监视。
ProgressBar.value	指明已建立的进程的数量。该属性为只读。

继承 [UIObject](#) 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](#) 的所有事件。

ProgressBar.complete

可用性

Flash Player 6 版本 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 `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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 `getBytesTotal` 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 版本 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。该值已向下定整。以下是用于计算百分比的公式：

$$100 * (\text{值} - \text{最小值}) / (\text{最大值} - \text{最小值})$$

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 版本 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 `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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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 248 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 390 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”。

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 523 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 25 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。

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”。

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](#)、[UIComponent](#)、[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](#)、[UIComponent](#)、[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](#)、[UIComponent](#)、[SimpleButton](#) 和 [Button](#) 类的所有事件。

RadioButton.click

可用性

Flash Player 6 版本 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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 radio0ne 单选按钮实例：

```
radio0ne.data = "#FF00FF";
```

RadioButton.groupName

可用性

Flash Player 6 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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（仅限 Flash Professional）

请将解析程序组件与 DataSet 组件（Macromedia Flash 数据结构中的数据管理功能的一部分）组合使用。通过解析程序组件，您能够将对应用程序内的数据进行的更改转换为适合于您所更新的外部数据源的格式。这些组件在运行时没有可视外观。

如果在应用程序中使用 DataSet 组件，则会生成一组优化的说明（DeltaPacket），它们描述了在运行时对数据所作的更改。这组说明会由解析程序组件转换为适当的格式（更新包）。在将更新发送到服务器后，服务器会发送一个响应（结果包），响应中包含由该更新操作导致的附加更新或错误。解析程序组件可以将此信息转换回 DeltaPacket，DeltaPacket 可应用于 DataSet 组件以使其与外部数据源保持同步。通过解析程序组件，您不必编写额外的动作脚本代码，就能够令应用程序和外部数据源保持同步。

RDBMSResolver 组件对 Web 服务、JavaBean、servlet 或 ASP 页可接收并分析的 XML 进行翻译。XML 包含在更新任何标准的 SQL 关系数据库时所需的信息和格式。可以使用并行解析程序组件 XupdateResolver（请参阅第 574 页的“[XUpdateResolver 组件（仅限 Flash Professional）](#)”）将数据返回到基于 XML 的服务器。有关 DataSet 组件的详细信息，请参阅第 176 页的“[DataSet 组件（仅限 Flash Professional）](#)”。有关连接器的详细信息，请参阅第 549 页的“[WebServiceConnector（仅限 Flash Professional）](#)”和第 567 页的“[XMLConnector 组件（仅限 Flash Professional）](#)”。有关 Flash 数据结构的详细信息，请参阅“使用 Flash”帮助中的“解析程序组件（仅限 Flash Professional）”。

RDBMSResolver 组件将在应用程序中对数据所作的更改转换为可以发送到外部数据源的 XML 包。

注意：可以使用 RDBMSResolver 将数据更新发送到可以分析 XML 并针对数据库生成 SQL 语句的任何外部数据源；例如，ASP 页、Java servlet 或 ColdFusion 组件。

RDBMSResolver 组件的更新以 XML 更新包的形式发送，而该更新包通过连接器对象与数据库进行通信。解析程序组件连接到 DataSet 组件的 DeltaPacket 属性，将其自己的更新包发送到连接器，从连接器接收服务器错误，并且将这些错误传送回 DataSet 组件 - 全都使用可绑定的属性。

使用 RDBMSResolver 组件（仅限 Flash Professional）

仅当 Flash 应用程序包含 DataSet 组件且必须将更新发送回数据源时，才使用此 RDBMSResolver 组件。此组件对您要返回到关系数据库的数据进行解析。

有关使用 RDBMSResolver 组件的详细信息，请参阅“使用 Flash”帮助中的“解析程序组件（仅限 Flash Professional）”。

RDBMSResolver 组件参数

TableName 一个字符串，在 XML 中表示要更新的数据库表的表名称。它应该与要更新的 Resolver.fieldInfo 项目的输入值相同。如果不存在对此字段的更新，此值应该为空（默认值）。

UpdateMode 一个枚举数，确定在生成 XML 更新包的过程中标识关键字段的方式。默认值为 umUsingKey。可能值如下：

- **umUsingAll** 使用修改的所有字段的旧值标识要更新的记录。这是用于更新的最安全的值，原因是，它确保了其他用户在您检索记录后没有修改过该记录。然而，这种方法很耗时，而且会生成较大的更新包。
- **umUsingModified** 使用修改的所有字段的旧值标识要更新的记录。该值确保其他用户在您获取记录后不修改记录中的相同字段。
- **umUsingKey** 这是此属性的默认值。此设置使用关键字段的旧值。这意味着“乐观并发”模型（当今的大部分数据库系统都采用该模型），并确保您正在修改的记录是在数据库中检索的记录。您的更改会覆盖任何其他用户对相同数据的更改。

NullValue 一个字符串，代表空字段值。此值可自定义，以防止与空字符串("")或另一个有效值混淆。默认值为 {_NULL_}。

FieldInfo 一个集合，代表唯一标识记录的一个或多个关键字段。如果数据源是数据库表，则应该有一个或多个字段唯一地标识其中的记录。另外，可能已通过其他表计算或联接了一些字段。必须标识这些字段，以便可以在 XML 更新包中设置关键字段，而且在 XML 更新包中忽略任何不应更新的字段。

RDBMSResolver 组件包含了一个用于此用途的 FieldInfo 属性。此集合属性使您可以定义不限数量的字段，这些字段具有标识需要特殊处理的字段的属性。集合中的每个 FieldInfo 项目都包含三个属性：

- **FieldName** 字段的名称。此名称应该映射到 DataSet 组件中的字段。
- **OwnerName** 一个可选值，用于标识不“属于”在解析程序组件的 TableName 参数中定义的同一个表的字段。如果此值与 TableName 参数的值相同或为空，则字段通常会包含在 XML 更新包中。如果值不相同，则此字段会被排除在更新包之外。
- **IsKey** 一个布尔值属性，应将其设置为 true，以便更新表的所有关键字段。

以下范例显示为了更新客户表中的字段而创建的 FieldInfo 项目。必须标识客户表中的关键字段。客户表具有一个关键字段 - id，因此，您应该创建具有以下值的字段项目：

```
FieldName = "id"
OwnerName = <--! 将此值保留为空 -->
IsKey = "true"
```

另外，在查询中使用联接操作来添加 custType 字段。此字段应被排除在更新之外，因此，创建具有以下值的字段项目：

```
FieldName = "custType"
OwnerName = "JoinedField"
IsKey = "false"
```

定义了字段项目之后，Flash Player 可以使用它们自动生成用于更新表的完整 XML。

RDBMSResolver 组件的属性摘要

属性	描述
<code>RDBMSResolver.deltaPacket</code>	DataSet 组件的 DeltaPacket 属性的副本。
<code>RDBMSResolver.fieldInfo</code>	不限数量的字段，这些字段具有将需要特殊处理的 DataSet 字段标识为关键字段或不可更新字段的属性。
<code>RDBMSResolver.nullValue</code>	指示字段值为空的指示符。
<code>RDBMSResolver.tableName</code>	放入 XML 中的表名称，指示要更新的数据库表。
<code>RDBMSResolver.updateMode</code>	在生成 XML 更新包时，确定如何标识关键字段的值。
<code>RDBMSResolver.updatePacket</code>	连接器 updatePacket 属性的副本，包含最新的 XML 格式数据。在服务器收到此应用程序的更新请求后，会将这些数据从连接器返回到 DataSet 组件。
<code>RDBMSResolver.updateResults</code>	连接器的 Results 属性的副本，它返回 DataSet 组件的任何 XML 格式错误或更新。

RDBMSResolver 组件的方法摘要

方法	描述
<code>RDBMSResolver.addFieldInfo()</code>	向 fieldInfo 集合中添加新项目，用于在运行时动态设置解析程序组件，而不使用创作环境中的“组件检查器”。

RDBMSResolver 组件的事件摘要

事件	描述
<code>RDBMSResolver.beforeApplyUpdates</code>	在应用程序中定义；由解析程序组件调用，以便在将 updatePacket 属性的 XML 绑定到连接器之前对其进行自定义修改。
<code>RDBMSResolver.reconcileResults</code>	在应用程序中定义；由解析程序组件调用，以便对发送到服务器的 updatePacket 属性和从服务器返回的 updatePacket 属性之间的更新进行协调。
<code>RDBMSResolver.reconcileUpdates</code>	在应用程序中定义；由解析程序组件调用，以便协调服务器接收的更新和未决的更新。

RDBMSResolver.addFieldInfo()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.addFieldInfo("fieldName", "ownerName", "isKey")
```


参数

fieldName 字符串；提供此信息对象描述的字段的名称。

ownerName 字符串；提供拥有此字段的表的名称。如果此参数与解析程序实例的 *tablename* 属性相同，则可将其留空 ("")。

isKey 布尔值；指示此字段是否是关键字段。

返回

无。

描述

方法；将新项目添加到更新包中的 XML *fieldInfo* 集合。如果您必须在运行时动态设置解析程序组件，请使用此方法，而不使用创作环境中的“组件检查器”。

范例

下面的范例创建了一个解析程序组件，并提供了表的名称、关键字段的名称，而且防止 *personTypeName* 字段被更新：

```
var myResolver:RDBMSResolver = new RDBMSResolver();
myResolver.tableName = "Customers";
// 将 id 字段设置为关键字段
// 并设置 personTypeName 字段，使其不会被更新。
myResolver.addFieldInfo("id", "", true);
myResolver.addFieldInfo("personTypeName", "JoinedField", false);
// 设置数据绑定
//...
```

RDBMSResolver.beforeApplyUpdates

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.beforeApplyUpdates(eventObject)
```

参数

eventObject 解析程序事件对象；描述在通过连接器将更新发送到数据库之前对 XML 包进行的自定义。此解析程序事件对象应包含以下属性：

属性	描述
target	对象；触发此事件的解析程序。
type	字符串；事件的名称。
updatePacket	XML 对象；将要应用的 XML 对象。

返回

无。

描述

属性；deltaPacket 类型的属性，它接收将要转换为 xupdatePacket 的 deltaPacket，并根据放置于 updateResults 属性中的任何服务器结果输出 deltaPacket。此事件处理函数提供一个方法，您可以通过此方法在将更新的数据发送到连接器前对 XML 进行自定义修改。

updateResults 属性中的消息视作错误。这意味着，具有消息的增量被再次添加到 deltaPacket 中，这样，下次将 deltaPacket 发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个 deltaPacket 前将这些消息提供给用户并进行修改。

范例

以下范例将用户身份验证数据添加到 XML 包：

```
on (beforeApplyUpdates) {  
    // 添加用户身份验证数据  
    var userInfo = new XML( ""+getUserId()+ ""+getPassword()+" " );  
    updatePacket.firstChild.appendChild( userInfo );  
}
```

RDBMSResolver.deltaPacket

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.deltaPacket
```

描述

属性；deltaPacket 类型的属性，它接收将要转换为 updatePacket 的 deltaPacket，并从放在 updateResults 属性中的任何服务器结果中输出 deltaPacket。

updateResults 属性中的消息视作错误。这意味着，具有消息的增量被再次添加到 deltaPacket 中，这样，下次将 deltaPacket 发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个 deltaPacket 前将这些消息提供给用户并进行修改。

RDBMSResolver.fieldInfo

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.fieldInfo
```

描述

属性；Collection 类型的属性，它指定不限数量的字段的集合，这些字段具有标识需要特殊处理（由于字段是关键字段或不可更新字段）的 DataSet 字段的属性。集合中的每个 FieldInfo 项目都包含三个属性：

属性	描述
FieldName	需特殊处理的字段的名称。此字段名称应该映射到 DataSet 中的字段名称。
OwnerName	如果此字段不“属于”在组件的 TableName 参数中定义的同个表，则此可选属性是此字段的所有者的名称。如果此属性中填入了与该参数相同的值或者留空，则此字段通常会包含在 XML 更新包中。如果填入了不同的值，则此字段会被排除在更新包之外。
IsKey	布尔值，对于要更新的表的所有关键字段设置为 true。

RDBMSResolver.nullValue

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`resolveData.deltaPacket`

描述

属性；String 类型属性，用于为字段的值提供空值。此值可自定义，以防止与空字符串 ("") 或另一个有效值混淆。默认字符串为 {_NULL_}。

RDBMSResolver.reconcileResults

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`resolveData.reconcileResults(eventObject)`

参数

eventObject 解析程序事件对象；描述用于比较两个 `updatePacket` 的事件对象。 此解析程序事件对象应包含以下属性：

属性	描述
<code>target</code>	对象；触发此事件的解析程序。
<code>type</code>	字符串；事件的名称。

返回

无。

描述

事件；由解析程序组件调用，用于在从服务器接收到结果并应用到 `deltaPacket` 后比较两个包。

一个 `updateResults` 包可能包含 `deltaPacket` 中的操作结果以及有关其他客户端所执行更新的信息。接收到新的 `updatePacket` 时，操作结果和数据库更新被分为两个 `updatePacket`，并分别放入到 `deltaPacket` 属性中。 `reconcileResults` 事件在使用数据绑定发送包含操作结果的 `deltaPacket` 之前激发。

范例

下面的范例会协调两个 `updatePacket`，并在成功后返回和清除更新：

```
on (reconcileResults) {  
    // 检查结果  
    if( examine( updateResults ))  
        myDataSet.purgeUpdates();  
    else  
        displayErrors( results );  
}
```

RDBMSResolver.reconcileUpdates

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.reconcileResults(eventObject)
```

参数

eventObject 解析程序事件对象；描述在将更新通过连接器发送到数据库之前对 XML 包进行的自定义。此解析程序事件对象应包含以下属性：

属性	描述
<code>target</code>	对象；触发此事件的解析程序。
<code>type</code>	字符串；事件的名称。

返回

无。

描述

事件；在应用 `deltaPacket` 中的更新后从服务器接收到结果时，由解析程序组件调用。一个 `updateResults` 包可能包含 `deltaPacket` 中的操作结果以及有关其他客户端所执行更新的信息。接收到新的 `updatePacket` 时，操作结果和数据库更新被分为两个 `deltaPacket`，并分别放入到 `deltaPacket` 属性中。`reconcileUpdates` 事件在使用数据绑定发送包含任何数据库更新的 `deltaPacket` 之前激发

示例

下面的范例会协调两个结果并在成功后清除更新：

```
on (reconcileUpdates) {  
    // 检查结果  
    if( examine( updateResults ))  
        myDataSet.purgeUpdates();  
    else  
        displayErrors( results );  
}
```

RDBMSResolver.tableName

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`resolveData.deltaPacket`

描述

属性；String 类型属性，用于在 XML 中表示要更新的数据库表的表名称。它还确定在 `updatePacket` 中发送哪个字段。为了进行此确定，解析程序组件会将此属性的值与为 `fieldInfo.ownerName` 属性提供的值进行比较。如果某个字段在 `fieldInfo` 集合属性中没有条目，则此字段会放入到 `updatePacket` 中。如果某个字段在 `fieldInfo` 集合属性中有条目，且 `ownerName` 属性值为空或者与解析程序组件的 `tableName` 属性相同，则此字段会放入到 `updatePacket` 中。如果某个字段在 `fieldInfo` 集合属性中有条目，且 `ownerName` 属性值不为空或者与解析程序组件的 `tableName` 属性不相同，则此字段不会放入到 `updatePacket` 中。

RDBMSResolver.updateMode

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

`resolveData.deltaPacket`

描述

属性；此属性包含一些值，这些值在生成 XML 更新包时，确定如何标识关键字段。以下三个字符串是此属性的值：

值	描述
umUsingAll	使用修改过的所有字段的旧值以标识要更新的记录。这是用于更新的最安全的值，原因是，它确保了其他用户在您检索记录后，没有修改过所有这些记录。然而，这种方法较为耗时，而且会生成较大的更新包。
umUsingModified	使用修改过的所有字段的旧值以标识要更新的记录。该值确保其他用户在您获取记录后不修改记录中的相同字段。
umUsingKey	这是此属性的默认值。使用关键字段的旧值。这意味着“乐观并发”模型（当今的大部分数据库系统都采用该模型），并确保您正在修改的记录是在数据库中检索的记录。您的更改会覆盖任何其他用户对相同数据的更改。

RDBMSResolver.updatePacket

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

resolveData.deltaPacket

描述

描述

属性；XML 类型属性，用于绑定到某个连接器属性，后者将更改的已转换更新包传送回服务器，以便可以更新数据源。这是包含 DataSet 更改的包的 XML 文档。

RDBMSResolver.updateResults

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

resolveData.deltaPacket

描述

属性；`deltaPacket` 类型属性，包含通过连接器从服务器返回的更新的结果。使用此属性可将错误和更新的数据从服务器传送到 `DataSet`；例如，当服务器为自动分配的字段分配新 ID 时。将此属性绑定到连接器的 `Results` 属性，以便它可以接收更新的结果并将结果传送回 `DataSet`。

`updateResults` 属性中的消息视作错误。这意味着，具有消息的增量被再次添加到 `deltaPacket` 中，这样，下次将 `deltaPacket` 发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个 `deltaPacket` 前将这些消息提供给用户并进行修改。

远程过程调用 (RPC) 组件 API

动作脚本类名称 `mx.datacomponents.RPC`

远程过程调用 (RPC) 组件 API 是一种接口（一组方法、属性和事件），它可以由 Flash MX 2004 v2 组件执行。RPC 组件 API 定义了一种轻松地向外部资源（例如 Web 服务）发送参数或通过它接收结果的方法。

执行 RPC API 的组件包括 `WebServiceConnector` 和 `XMLConnector` 组件。这些组件充当外部数据源（例如 Web 服务或 XML 文件）之间的连接器，以及充当应用程序中的 UI 组件。

RPC 组件能够调用一个外部函数、传入参数以及接收结果。组件可以多次调用同一个函数。要调用多个函数，必须使用多个组件。

RPC 组件类的属性摘要

属性	描述
<code>RPC.multipleSimultaneousAllowed</code>	指明是否可以同时进行多个调用。
<code>RPC.params</code>	指定在执行下一个 <code>Trigger()</code> 操作时要发送到服务器的数据。
<code>RPC.results</code>	标识作为 <code>Trigger()</code> 操作的结果从服务器接收的数据。
<code>RPC.suppressInvalidCalls</code>	指明在参数无效的情况下是否禁止调用。

RPC 组件类的方法摘要

方法	描述
<code>RPC.trigger()</code>	启动远程过程调用。

RPC 组件类的事件摘要

事件	描述
<code>RPC.result</code>	远程过程调用成功完成后广播。
<code>RPC.send</code>	当 <code>Trigger</code> 函数执行时（在收集了参数数据之后，但在验证这些数据且启动“远程调用”之前）进行广播。
<code>RPC.status</code>	在远程过程调用启动时广播，以通知用户操作状态。

RPC.multipleSimultaneousAllowed

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.multipleSimultaneousAllowed;
```

描述

属性；指明是否可以同时进行多个调用。如果为 `false`，则 `Trigger()` 函数将在另一个调用已在执行时执行一个调用。将发出 `status` 事件，并且具有代码 `CallAlreadyInProgress`。如果值为 `true`，将发生调用。

当同时进行多个调用时，不保证这些调用的完成顺序与触发它们的顺序相同。此外，Flash Player 可能会对同时发生的网络操作的数目加以限制。该限制将因版本和平台而异。

RPC.params

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.params;
```

描述

属性；指定在执行下一个 `Trigger()` 操作时要发送到服务器的数据。每个 RPC 组件均定义使用此数据的方式以及有效的类型。

RPC.result

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("result", myListenerObject);
```

描述

事件；在远程过程调用操作成功完成后广播。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 “result”

- **target** : 对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）

您可以使用 `results` 属性获取实际结果值。

RPC.results

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.results;
```

描述

属性；标识作为 `Trigger()` 操作的结果从服务器接收的数据。每个 RPC 组件均定义获取此数据的方式以及有效的类型。这些数据在 RPC 操作成功完成后出现，由 `result` 事件通知。它在该组件卸载前或执行下一个 RPC 操作前可用。

返回的数据可能会非常大。您可以通过以下两种方式管理这些数据：

- 选择适当的影片剪辑、时间轴或屏幕作为 RPC 组件的父级。在父级退出后该组件的存储区将可用于垃圾回收。
- 在动作脚本中，您可以随时将空值分配给该属性。

RPC.send

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("send", myListenerObject);
```

描述

事件；在处理 `Trigger()` 操作期间（在收集了参数数据后，但在验证这些数据和启动远程过程调用前）广播。此位置适合于存放将在调用前修改参数数据的代码。

事件处理函数的参数是具有以下字段的对象：

- **type** : 字符串 “send”
- **target** : 对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）

您可以使用 `params` 属性获取或修改实际参数值。

RPC.status

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("status", myListenerObject);
```

描述

事件；在远程过程调用启动时广播，以通知用户操作状态。

事件处理函数的参数是具有以下字段的对象：

- type：字符串 “status”
- target：对发出该事件的对象的引用；例如，WebServiceConnector 组件
- code：一个字符串，提供已发生的特定情况的名称。
- data：一个对象，其内容取决于代码。

下面是可用于状态事件的代码和关联数据：

代码	数据	描述
StatusChange	{callsInProgress:nnn}	只要 Web 服务调用开始或结束就发出该事件。“nnn” 项提供当前正发生的调用的数目。
CallAlreadyInProgress	无数据	如果：(a) 调用了 Trigger 函数，且 (b) multipleSimultaneousAllowed 为 false，同时 (c) 调用已在执行中，则会发出此事件。在发生此事件后，则认为尝试的调用已完成，并且不会发生 “result” 或 “send” 事件。
InvalidParams	无数据	如果 Trigger 函数发现 “params” 属性不包含有效数据，则会发出此事件。如果 “suppressInvalidCalls” 属性为 true，则认为尝试的调用已完成，并且不会发生 “result” 或 “send” 事件。

RPC.suppressInvalidCalls

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.suppressInvalidCalls;
```

描述

属性；指明在参数无效的情况下是否禁止调用。如果为 true，则 Trigger() 函数在绑定的参数未通过验证的情况下将不执行调用。将发出 “status” 事件，并且具有代码 InvalidParams。如果为 false，将进行该调用，并且根据需要使用无效数据。

RPC.trigger()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
componentInstance.trigger();
```

描述

方法；启动远程过程调用。每一 RPC 组件均精确定义此操作涉及的内容。如果该操作成功，则操作结果将出现在 RPC 组件的 `results` 属性中。

`trigger()` 方法执行以下步骤：

- 1 如果任何数据绑定到 `params` 属性，则该方法执行所有绑定以确保提供最新数据。这样做还会导致发生数据验证。
- 2 如果数据无效且 `suppressInvalidCalls` 设置为 `true`，则操作将停止。
- 3 如果操作继续，则发出 `send` 事件。
- 4 使用指示的连接方法（例如 HTTP）启动实际远程调用。

Screen 类（仅限 Flash Professional）

继承 UIObject > UIComponent > View > Loader > Screen

动作脚本类名称 mx.screens.Screen

Screen 类是在 Flash MX Professional 2004 的“屏幕大纲”窗格中创建的屏幕的基类。屏幕是用于创建应用程序和演示文稿的高级容器。有关使用屏幕的概述，请参阅“使用 Flash”帮助中的“使用屏幕（仅限 Flash Professional）”。

Screen 类有两个主要的子类：Slide 和 Form。

Slide 类提供幻灯片演示文稿的运行时行为。Slide 类具有内置的导航和排序功能，并能够使用“行为”轻松地在幻灯片之间附加过渡。幻灯片对象含有“状态”概念，因此用户可以进入到下一或上一幻灯片 / 状态；显示下一张幻灯片时，上一张幻灯片即会隐藏。有关使用 Slide 类控制幻灯片演示文稿的详细信息，请参阅第 435 页的“Slide 类（仅限 Flash Professional）”。

Form 类为窗体应用程序提供运行时环境。窗体能够覆盖其他组件、作为其他组件的容器或者被其他组件包含。与幻灯片不同，窗体不提供任何排序或导航功能。有关使用 Form 类的详细信息，请参阅第 255 页的“Form 类（仅限 Flash Professional）”。

Screen 类提供幻灯片和窗体共有的功能。

屏幕知道如何管理其子项 每个屏幕都附带一种内置的属性（该属性是作为子项的屏幕的集合）。此集合由“屏幕窗格”中的屏幕层次结构布局确定。屏幕可以具有任意数量（包括零）的子项，这些子项自身也可以有子项。

屏幕可以隐藏 / 显示其子项 因为屏幕本质上是嵌套的影片剪辑的集合，所以屏幕可以控制其子项的可见性。对于窗体应用程序，在默认情况下，屏幕的所有子项同时可见；对于幻灯片演示文稿，通常是一次显示一个屏幕。

屏幕广播事件 例如，当特定的屏幕可见时，可以触发声音的播放或者开始播放一段视频。

将外部内容加载到屏幕（仅限 Flash Professional）

Screen 类扩展了 Loader 类（请参阅第 289 页的“Loader 组件”），从而提供了轻松管理和加载外部 SWF（和 JPEG）的能力。Loader 类包含一个名为 `contentPath` 的属性，此属性指定外部 SWF 或 JPEG 的 URL，或者“库”中的影片剪辑的链接标识符。

使用此功能，可以将外部屏幕树（或任何外部 SWF）作为任何屏幕节点的子项加载。这提供了一种有用的方法，利用此方法可将基于屏幕的影片模块化，并将这些影片分为多个单独的 SWF。

例如，假如您有一个幻灯片演示文稿，有三个人各自提供了其中的一个部分。您可以要求每个演示者各自创建一个单独的幻灯片演示文稿 (SWF)。然后，您可以创建一个“主幻灯片演示文稿”，其中包含三个占位符幻灯片，分别用于演示者创建的各个幻灯片演示文稿。对于每个占位符幻灯片，可以将它的 `contentPath` 属性指向每个 SWF。

例如，可以按下图所示安排“主”幻灯片演示文稿：



“主” SWF 幻灯片演示文稿结构

假设演示者提供了三个 SWF：`speaker_1.swf`、`speaker_2.swf` 和 `speaker_3.swf`。通过指定每个占位符幻灯片的 `contentPath` 属性（方法是使用动作脚本，或者为每张幻灯片设置属性检查器的 `contentPath` 属性），即可轻松地组合整个演示文稿。

```
Speaker_1.contentPath = speaker_1.swf;
Speaker_2.contentPath = speaker_2.swf;
Speaker_3.contentPath = speaker_3.swf;
```

也可以使用属性检查器设置每张幻灯片的 `contentPath` 属性。请注意，在默认情况下，当您在属性检查器中（或使用以上所示的代码）设置幻灯片的 `contentPath` 后，只要加载了“主演示文稿” SWF，即会加载指定的 SWF。为了缩短最初加载时间，可以考虑设置附加到每张幻灯片的 `on(reveal)` 处理函数中的 `contentPath` 属性。

```
// 附加到 Speaker_1 幻灯片
on(reveal) {
    this.contentPath="speaker_1.swf";
}
```

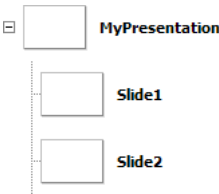
或者，您可以将幻灯片的 `autoLoad` 属性（从 Loader 类继承）设置为 `false`，然后在幻灯片显示出来时调用幻灯片上的 `load()` 方法（也是从 Loader 类继承）。

```
// 附加到 Speaker_1 幻灯片
on(reveal) {
    this.load();
}
```

使用动作脚本引用加载的屏幕

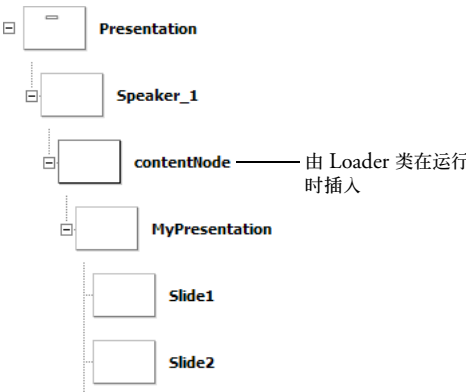
Loader 类创建一个名为 `contentNode` 的内部影片剪辑，它将 `contentPath` 属性指定的 SWF 或 JPEG 加载到其中。实际上，该影片剪辑会在“占位符”幻灯片（在上面的“主”演示文稿中创建）和加载的幻灯片演示文稿中的第一张幻灯片之间添加额外的屏幕节点。

例如，假设为上图所示的 `Speaker_1` 幻灯片占位符创建的 SWF 具有以下结构（如“屏幕大纲”窗格中所示）：



“Speaker 1” SWF 幻灯片演示文稿结构

在运行时，当 `Speaker 1` SWF 加载到占位符幻灯片中时，整个幻灯片演示文稿将会具有以下结构：



“主”演示文稿和“speaker”演示文稿的结构（运行时）

`Screen`、`Slide` 和 `Form` 类的属性和方法会尽可能地“忽略”此 `contentHolder` 节点。也就是说（请参考上面的图示），名为 `MyPresentation` 的幻灯片（及其子幻灯片）是以 `Presentation` 幻灯片为根的不断幻灯片树的一部分，而且不被视为单独的子树。

Screen 类的方法摘要

方法	描述
<code>Screen.getChildScreen()</code>	返回此屏幕位于特定索引处的子屏幕。

继承 `UIObject`、`UIComponent`、`View` 和 `Loader` 组件的所有方法。

Screen 类的属性摘要

属性	描述
<code>Screen.currentFocusedScreen</code>	返回包含全局当前焦点的屏幕。
<code>Screen.indexInParent</code>	返回该屏幕在其父屏幕的子屏幕列表中的索引（从零开始）。
<code>Screen.numChildScreens</code>	返回屏幕包含的子屏幕数量。
<code>Screen.parentIsScreen</code>	返回一个布尔值（true 或 false），指示屏幕的父对象本身是否是屏幕。
<code>Screen.rootScreen</code>	返回包含此屏幕的（子）树的根屏幕。

继承 `UIObject`、`UIComponent`、`View` 和 `Loader` 组件的所有属性。

Screen 类的事件摘要

事件	描述
<code>Screen.allTransitionsInDone</code>	当应用到屏幕的所有“进入”过渡结束时进行广播。
<code>Screen.allTransitionsOutDone</code>	当应用到屏幕的所有“退出”过渡结束时进行广播。
<code>Screen.mouseDown</code>	当在直接属于屏幕的对象（形状或影片剪辑）上按下鼠标按钮时进行广播。
<code>Screen.mouseDownSomewhere</code>	当在舞台上的某处（不一定要在属于此屏幕的对象上）按下鼠标按钮时进行广播。
<code>Screen.mouseMove</code>	当鼠标在屏幕上移动时进行广播。
<code>Screen.mouseOut</code>	当鼠标从屏幕内移出时进行广播。
<code>Screen.mouseOver</code>	当鼠标从屏幕外移入时进行广播。
<code>Screen.mouseUp</code>	当在直接属于屏幕的对象（形状或影片剪辑）上松开鼠标按钮时进行广播。
<code>Screen.mouseUpSomewhere</code>	当在舞台上的某处（不一定要在属于此屏幕的对象上）松开鼠标按钮时进行广播。

继承 `UIObject`、`UIComponent`、`View` 和 `Loader` 组件的所有事件。

Screen.allTransitionsInDone

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(allTransitionsInDone) {  
    // 此处是您的代码  
}  
listenerObject = new Object();
```

```
listenerObject.allTransitionsInDone = function(eventObject){
    // 此处插入您的代码
}
screenObj.addEventListener("allTransitionsInDone", listenerObject)
```

说明

事件；当应用到此屏幕的所有“进入”过渡结束时进行广播。allTransitionsInDone 事件由与 *myScreen* 关联的过渡管理器广播。

示例

在下面的范例中，当应用到名为 *mySlide* 的幻灯片的所有“进入”过渡结束后，*mySlide* 所包含的按钮 (*nextSlide_btn*) 即可见。

```
// 附加到 mySlide :
on(allTransitionsInDone) {
    this.nextSlide_btn._visible = true;
}
```

另请参见

[Screen.allTransitionsOutDone](#)

Screen.allTransitionsOutDone

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(allTransitionsOutDone) {
    // 此处是您的代码
}
listenerObject = new Object();
listenerObject.allTransitionsOutDone = function(eventObject){
    // 此处插入您的代码
}
screenObj.addEventListener("allTransitionsOutDone", listenerObject)
```

说明

事件；当应用到此屏幕的所有“退出”过渡结束时进行广播。allTransitionsOutDone 事件由与 *myScreen* 关联的过渡管理器广播。

另请参见

[Screen.currentFocusedScreen](#)

Screen.currentFocusedScreen

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mx.screens.Screen.currentFocusedScreen
```

描述

静态属性（只读）；返回对包含全局当前焦点的“最低叶”屏幕对象的引用。焦点可以在屏幕自身上，或者在影片剪辑、文本对象或者该屏幕内的组件上。如果没有当前焦点，则默认为 null。

范例

下面的范例在“输出”面板中显示当前具有焦点的屏幕的名称。

```
var currentFocus:mx.screens.Screen = mx.screens.Screen.currentFocusedScreen;
trace("Current screen is:" + currentFocus._name);
```

Screen.getChildScreen()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myScreen.getChildScreen(index)
```

参数

childIndex 指示要返回的子屏幕索引的一个数字（从零开始）。

返回

屏幕对象。

说明

方法；返回 *myScreen* 的子屏幕对象（其索引为 *childIndex*）。

示例

下面的范例在“输出”面板中显示了属于名为 *Presentation* 的根屏幕的所有子屏幕名称。

```
for (var i:Number = 0; i < _root.Presentation.numChildScreens; i++) {
    var childScreen:mx.screens.Screen = _root.Presentation.getChildScreen(i);
    trace(childScreen._name);
}
```


Screen.indexInParent

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myScreen.indexInParent

说明

属性（只读）；包含 *myScreen* 在其父屏幕的子屏幕列表中的索引（从零开始）。

示例

下面的范例显示了屏幕 *myScreen* 在其父屏幕的子屏幕列表中的相对位置。

```
var numChildren:Number = myScreen._parent.numChildScreens;  
var myIndex:Number = myScreen.indexInParent;  
trace("I   child slide # " + myIndex + " out of " + numChildren + " screens.");
```

Screen.mouseDown

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseDown) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseDown = function(eventObj){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseDown", listenerObject)
```

说明

事件；当在直接属于屏幕的对象（例如，形状或影片剪辑）上按下鼠标按钮时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”](#)。

示例

下面的代码在“输出”面板中显示了曾捕获鼠标事件的屏幕的名称。

```
on(mouseDown) {  
    trace("Mouse down event on:" + eventObj.target._name);  
}
```

Screen.mouseDownSomewhere

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseDown) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseDownSomewhere = function(eventObject){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseDownSomewhere", listenerObject)
```

说明

事件；当按下鼠标按钮（但不一定在指定的屏幕上）时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

Screen.mouseMove

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseDown) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseMove = function(eventObject){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseMove", listenerObject)
```

说明

事件；当鼠标在屏幕上移动时进行广播。此事件仅当鼠标移到此屏幕的边框上时才发送。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

注意：使用此事件可能会影响系统性能，因此使用时应小心。

Screen.mouseOut

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseOut) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseOut = function(eventObject){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseOut", listenerObject)
```

说明

事件；当鼠标从屏幕的边框内移到边框外时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

注意：使用此事件可能会影响系统性能，因此使用时应小心。

Screen.mouseOver

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseDown) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseOver = function(eventObject){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseOver", listenerObject)
```

说明

事件；当鼠标从屏幕的边框外移到边框内时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

注意：使用此事件可能会影响系统性能，因此使用时应小心。

Screen.mouseUp

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseUp) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseUP = function(eventObject){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseUp", listenerObject)
```

说明

事件；在屏幕上松开鼠标时进行广播。

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

Screen.mouseUpSomewhere

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
on(mouseUpSomewhere) {  
    // 此处是您的代码  
}  
listenerObject = new Object();  
listenerObject.mouseUpSomewhere = function(eventObject){  
    // 此处插入您的代码  
}  
screenObj.addEventListener("mouseUpSomewhere", listenerObject)
```

说明

事件；当按下鼠标按钮（但不一定在指定的屏幕上）时进行广播

该事件被触发时，它会自动将一个事件对象 (*eventObj*) 传递到处理函数。每个事件对象都有一组属性，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 510 页的“事件对象”。

Screen.numChildScreens

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myScreen.numChildScreens

说明

属性（只读）；返回 *myScreen* 包含的子屏幕数量。

示例

下面的范例显示属于 *myScreen* 的所有子屏幕的名称。

```
var howManyKids:Number = myScreen.numChildScreens;
for(i=0; i<howManyKids; i++) {
    var childScreen = myScreen.getChildScreen(i);
    trace(childScreen._name);
}
```

另请参见

[Screen.getChildScreen\(\)](#)

Screen.parentIsScreen

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myScreen.parentIsScreen

说明

属性（只读）：返回一个布尔值（true 或 false），指示所指定屏幕的父对象是 (true) 否 (false) 也是屏幕。如果为 false，则 *myScreen* 位于其屏幕层次结构的根位置。

示例

下面的代码确定屏幕 *myScreen* 的父对象是否也是屏幕。如果是，则假设 *myScreen* 为演示文稿中的根幻灯片或主幻灯片，因而没有同辈幻灯片。否则，如果 *myScreen.parentIsScreen* 为 true，则在“输出”面板中显示 *myScreen* 的同辈幻灯片数量。

```
if (myScreen.parentIsScreen) {
    trace("I have "+myScreen._parent.numChildScreens+" sibling screens");
} else {
    trace("I am the root screen and have no siblings");
}
```

Screen.rootScreen

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

myScreen.rootScreen

说明

属性（只读）；返回位于屏幕层次结构顶层的包含 *myScreen* 的屏幕。

示例

下面的范例显示名称

```
var myRoot:mx.screens.Screen = myScreen.rootScreen;
```

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 248 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);

completelListener = new Object;
completelListener.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。

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。

ScrollPane 类

继承 `UIObject > UIComponent > View > ScrollView > ScrollPane`

动作脚本类名称 `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](#) 和 [UIComponent](#) 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量。
ScrollPane.hPageScrollSize	按轨道时要水平滚动的内容量。
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量。
ScrollPane.vPageScrollSize	轨道被按下时要垂直滚动的内容数量。
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](#) 和 [UIComponent](#) 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](#) 和 [UIComponent](#) 的所有事件。

ScrollPane.complete

可用性

Flash Player 6 版本 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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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*) 调度一个事件（在本例中为 **progress**），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您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 510 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 版本 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 版本 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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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 版本 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 版本 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";
```

Slide 类（仅限 Flash Professional）

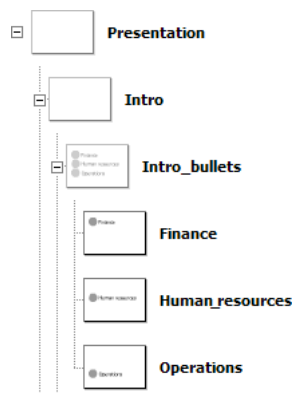
继承 UIObject > UIComponent > View > Loader > Screen > Slide

动作脚本类名称 mx.screens.Slide

Slide 类对应于分层的幻灯片演示文稿中的节点。在 Flash MX Professional 2004 中，可以使用“屏幕大纲”窗格创建幻灯片演示文稿。有关使用屏幕的概述，请参阅“使用 Flash”帮助中的“使用屏幕（仅限 Flash Professional）”。

Slide 类扩展了 Screen 类（请参阅第 411 页的“Screen 类（仅限 Flash Professional）”），并提供在幻灯片之间导航和排序的内置功能，它还能够使用“行为”轻松地在幻灯片之间附加过渡。幻灯片含有“状态”概念，因此用户可以进入到演示文稿中的下一张或上一张幻灯片，而且当显示演示文稿中的下一张幻灯片时，上一张幻灯片会隐藏。

请注意，您只可以导航到（或“停止于”）不包含任何子幻灯片的幻灯片（也称为“叶”幻灯片）。例如，以下图示显示幻灯片演示文稿范例的“屏幕大纲”窗格的内容。



当此演示文稿开始后，默认情况下，将在名为 Finance 的幻灯片（即演示文稿中第一张不包含任何子幻灯片的幻灯片）处“停止”。

还应注意，子幻灯片会“继承”其父幻灯片的可视外观（图形和其他内容）。例如，在上面的图示中，除了 Finance 幻灯片上的内容，用户还会看到 Intro 和 Presentation 幻灯片上的任何内容。

注意：Slide 类继承 Loader 类（请参阅第 291 页的“Loader 类”），从而可让您轻松地将外部 SWF（或 JPEG）加载到给定的幻灯片中。这就提供了一种模块化幻灯片演示文稿并降低初始下载时间的方法。有关详细信息，请参阅第 412 页的“将外部内容加载到屏幕（仅限 Flash Professional）”。

使用 Slide 类（仅限 Flash Professional）

使用 Slide 类的方法和属性来控制您使用“屏幕大纲”窗格（“窗口” > “屏幕”）创建的幻灯片演示文稿，以便获取有关幻灯片演示文稿的信息（例如，确定父幻灯片所包含的子幻灯片的数量），或者在幻灯片演示文稿中的幻灯片之间导航（例如，创建“下一张幻灯片”和“上一张幻灯片”按钮）。

也可以使用“行为”面板（“窗口” > “开发面板” > “行为”）中可用的一种内置行为控制幻灯片演示文稿。有关对幻灯片使用行为的详细信息，请参阅“使用 Flash”帮助中的“使用行为向屏幕添加控件（仅限 Flash Professional）”。

幻灯片参数

以下是您可以在属性检查器或“组件检查器”面板中为每张幻灯片设置的创作参数：

autoKeyNav 确定幻灯片如何或是否响应默认的键盘导航。有关详细信息，请参阅 [Slide.autoKeyNav](#)。

autoload 指示 `contentPath` 参数指定的内容是应该自动加载 (true)，还是应该等到调用 `Loader.load()` 方法时再进行加载 (false)。默认值为 true。

contentPath 指定幻灯片的内容。该参数可以是一个影片剪辑的链接标识符，也可以是要加载到幻灯片中的 SWF 或 JPG 文件的绝对或相对 URL。默认情况下，加载的内容会进行剪辑以适合幻灯片的大小。

overlayChildren 指定在从一张子幻灯片导航到下一张子幻灯片时，幻灯片的子幻灯片是保持可见 (true) 还是不可见 (false)。

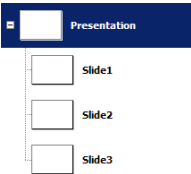
playHidden 指定幻灯片在隐藏时是继续播放 (true) 还是停止播放 (false)。

使用 Slide 类创建幻灯片演示文稿

使用 Slide 类的方法和属性来控制您在 Flash 创作环境的“屏幕大纲”窗格（“窗口” > “屏幕”）中创建的幻灯片演示文稿。请注意，“行为”面板（“窗口” > “开发面板” > “行为”）包含一些用于创建幻灯片导航的行为。在本例中，您将编写自己的动作脚本，以便为幻灯片演示文稿创建“下一张”和“上一张”按钮。

要创建带有导航的幻灯片演示文稿：

- 1 在 Flash 中，选择“文件” > “新建”。
- 2 单击“常规”选项卡并在“类型”下选择“Flash 幻灯片演示”。
- 3 在“屏幕大纲”窗格中，单击“插入屏幕”按钮(+)两次，以在 Presentation 幻灯片下面创建两张新幻灯片。
“屏幕大纲”窗格应如下所示：



- 4 在“屏幕大纲”窗格中选择 Slide1，并使用“文本”工具添加一个文本字段，内容为“这是第一张幻灯片”。
- 5 对 Slide2 和 Slide3 重复上述步骤，分别在每张幻灯片上创建内容为“这是第二张幻灯片”和“这是第三张幻灯片”的文本字段。
- 6 选择 Presentation 幻灯片并打开“组件”面板（“窗口” > “开发面板” > “组件”）。
- 7 将 Button 组件从“组件”面板拖到舞台底部。
- 8 在属性检查器（“窗口” > “属性”）中，为 Button 组件的 Label 属性输入“下一张幻灯片”。
- 9 如果仍未打开“动作”面板，选择“窗口” > “开发面板” > “动作”以将其打开。
- 10 在“动作”面板中输入以下代码：

```
on(click){
    _parent.currentSlide.gotoNextSlide();
}
```

- 11 测试 SWF（“控制” > “测试影片”）并单击“下一张幻灯片”按钮进入到下一张幻灯片。

Slide 类的方法摘要

属性	描述
Slide.getChildSlide()	返回此幻灯片位于给定的索引处的子幻灯片。
Slide.gotoFirstSlide()	导航到幻灯片的子幻灯片层次结构中的第一个叶节点。
Slide.gotoLastSlide()	导航到幻灯片的子幻灯片层次结构中的最后一个叶节点。
Slide.gotoNextSlide()	导航到下一个幻灯片。
Slide.gotoPreviousSlide()	导航到上一个幻灯片。
Slide.gotoSlide()	导航到任意一张幻灯片。

继承 [UIObject](#)、[UIComponent](#)、[View](#)、[Loader](#) 组件和 [Screen](#) 类（仅限 Flash Professional）的所有方法。

Slide 类的属性摘要

属性	描述
Slide.autoKeyNav	确定幻灯片是否使用默认的键盘处理来导航到下一张 / 上一张幻灯片。
Slide.currentSlide	返回包含当前活动幻灯片的幻灯片的直接子幻灯片。
Slide.currentSlide	返回当前活动的幻灯片。
Slide.currentFocusedSlide	返回包含全局当前焦点的 “最低叶” 幻灯片。
Slide.defaultKeyDownHandler	一种回调处理函数，它覆盖默认的按键幻灯片导航（向左和向右箭头）。
Slide.firstSlide	返回幻灯片的第一张没有子幻灯片的子幻灯片。
Slide.getChildSlide()	返回指定索引处的子幻灯片。
Slide.indexInParentSlide	返回幻灯片在其父幻灯片的子幻灯片列表中的索引（从零开始）。
Slide.lastSlide	返回幻灯片的最后一张没有子幻灯片的子幻灯片。
Slide.nextSlide	返回下一张叶节点幻灯片。
Slide.numChildSlides	返回幻灯片所包含的子幻灯片的数量。
Slide.overlayChildren	确定当控制从一张子幻灯片转到下一张子幻灯片时，幻灯片的子幻灯片是否可见。
Slide.parentIsSlide	返回一个布尔值，指示幻灯片的父对象也是 (true) 或不是 (false) 幻灯片。
Slide.playHidden	确定当幻灯片隐藏时是否继续播放。
Slide.previousSlide	返回上一个叶节点幻灯片。
Slide.revealChild	返回包含此幻灯片的幻灯片树的根。

继承 [UIObject](#)、[UIComponent](#)、[View](#)、[Loader](#) 组件和 [Screen](#) 类（仅限 Flash Professional）的所有属性。

Slide 类的事件摘要

事件	描述
Slide.hideChild	当幻灯片的所有子幻灯片从可见变为不可见时进行广播。
Slide.revealChild	当幻灯片的所有子幻灯片从不可见变为可见时进行广播。

继承 [UIObject](#)、[UIComponent](#)、[View](#)、[Loader](#) 组件和 [Screen](#) 类（仅限 Flash Professional）的所有事件。

Slide.autoKeyNav

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.autoKeyNav

说明

属性；确定当 *mySlide* 具有焦点时，幻灯片是否使用默认的键盘处理来导航到下一张 / 上一张幻灯片。该属性接受下列字符串值之一："true"、"false" 或 "inherit"。您也可以使用 [Slide.defaultKeydownHandler](#) 属性覆盖此默认的键盘处理行为。

也可以使用属性检查器设置该属性。

如果设置为 "true"，则在 *mySlide* 具有焦点时按向右箭头 (Key.RIGHT) 或空格键 (Key.SPACE) 会进入到下一张幻灯片；按向左箭头 (Key.Left) 会移到上一张幻灯片。

如果设置为 "false"，则在 *mySlide* 具有焦点时不发生默认的键盘处理。

如果设置为 "inherit"，则 *mySlide* 会检查其父幻灯片的 autoKeyNav 属性。如果 *mySlide* 的父幻灯片也设置为 "inherit"，则会检查 *mySlide* 的父幻灯片的父幻灯片，依此类推，直到找到 autoKeyNav 属性设置为 "true" 或 "false" 的父幻灯片为止。

如果 *mySlide* 没有父幻灯片（也即如果 (mySlide.parentIsSlide == false) 为 true），则它会表现出仿佛 autoKeyNav 设置为 true 的行为。

示例

此范例为名为 loginSlide 的幻灯片关闭自动键盘导航。

```
_root.Presentation.loginSlide.autoKeyNav = "false";
```

另请参见

[Slide.defaultKeydownHandler](#)

Slide.currentSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.currentSlide

说明

属性（只读）；返回当前活动的幻灯片。返回的始终是“叶”幻灯片，即不包含子幻灯片的幻灯片。

示例

以下代码附加到 **Presentation** 根幻灯片上的一个按钮，并会在每次按下此按钮时进入到幻灯片演示文稿的下一张幻灯片。

```
// 附加到 Presentation 幻灯片所包含的按钮实例 :
on(press) {
    _parent.currentSlide.gotoNextSlide();
}
```

另请参见

[Slide.gotoNextSlide\(\)](#)

Slide.currentChildSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.currentChildSlide

说明

属性（只读）；返回包含当前活动幻灯片的 *mySlide* 的直接子幻灯片；如果 *mySlide* 包含的子幻灯片都不具有当前焦点，则返回 `null`。

示例

考虑以下屏幕大纲：

```
Presentation
  Slide_1
    Bullet1_1
      SubBullet1_1_1
    Bullet1_2
      SubBullet1_2_1
  Slide_2
```

假设 `SubBullet1_1_1` 为当前幻灯片，则以下语句全部为 `true`：

```
Presentation.currentChildSlide == Slide_1;
Slide_1.currentChildSlide == Bullet_1_1;
SubBullet_1_1_1.currentChildSlide == null;
Slide_2.currentChildSlide == null;
```

另请参见

[Slide.currentSlide](#)

Slide.currentFocusedSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mx.screens.Slide.currentFocusedSlide
```

说明

属性（只读）；返回包含当前全局焦点的“最低叶”幻灯片。实际的焦点可能在幻灯片自身上，或者在该幻灯片内的影片剪辑、文本对象或组件上；如果没有当前焦点，则返回 `null`。

示例

```
var focusedSlide = mx.screens.Slide.currentFocusedSlide;
```

Slide.defaultKeydownHandler

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.defaultKeyDownHandler = function (eventObj) {  
    // 此处是您的代码  
}
```

参数

eventObj 具有以下属性的事件对象：

- `type` 一个指示事件类型的字符串。可能值为 "keyUp" 和 "keyDown"
- `ascii` 一个整数，表示上次按下的按键的 ASCII 值；与 `Key.getAscii()` 返回的值对应。
- `code` 一个整数，表示上次按下的按键的按键代码；与 `Key.getCode()` 所返回的值对应。
- `shiftKey` 一个布尔值（true 或 false），表示当前是按下了 (true) 还是没按下 (false) Shift 键。
- `ctrlKey` 一个布尔值（true 或 false），表示当前是按下了 (true) 还是没按下 (false) Control 键。

返回

无。

说明

回调处理函数；可让您用所创建的自定义键盘处理函数覆盖默认的键盘导航。例如，您可以不使用向左和向右箭头键分别导航到演示文稿中的上一张和下一张幻灯片，而使用向上和向下箭头键执行这些功能。有关默认的键盘处理行为的论述，请参阅 [Slide.autoKeyNav](#)。

在以下情况中会启用自动键盘处理：当前幻灯片的 `Slide.autoKeyNav` 属性设置为 "true"；或者该属性设置为 "inherit"，且非 "inherit" 的当前幻灯片的最直接始祖幻灯片是演示文稿的根幻灯片，或者其 `autoKeyNav` 值设置为 "true"。

如果为当前幻灯片启用了自动键盘处理，

示例

在该范例中，对于 `on(load)` 处理函数所附加到的幻灯片的子幻灯片，默认的键盘处理发生了改变。此处理函数使用向上 / 向下箭头而不是向左 / 向右箭头进行导航。

```
on(load){
    this.defaultKeyDownHandler = function(eventObj:Object) {
        switch (eventObj.code) {
            case Key.DOWN :
                this.currentSlide.gotoNextSlide();
                break;
            case Key.UP :
                this.currentSlide.gotoPreviousSlide();
                break;
            default :
                break;
        }
    };
}
```

另请参见

[Slide.autoKeyNav](#)

Slide.firstSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.firstSlide

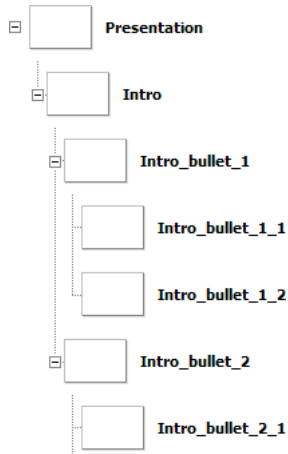
说明

属性（只读）；返回 *mySlide* 的第一个没有子幻灯片的子幻灯片。

示例

例如，在下面显示的幻灯片层次结构中，以下语句全部为 `true`：

```
Presentation.Intro.firstSlide == Intro_bullet_1_1;
Presentation.Intro_bullet_1.firstSlide == Intro_bullet_1-1;
```



Slide.getChildSlide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.getChildSlide(childIndex)
```

参数

childIndex 要返回的子幻灯片的索引（从零开始）。

返回

一个幻灯片对象。

说明

方法；返回 *mySlide* 的子幻灯片（其索引为 *childIndex*）。此方法在重复一组已知索引的子幻灯片时很有用，如以下范例所示。

示例

此范例在“输出”面板中显示 **Presentation** 根幻灯片的所有子幻灯片的名称。

```
var numSlides = _root.Presentation.numChildSlides;
for(var slideIndex=0; slideIndex < numSlides; slideIndex++) {
    var childSlide = _root.Presentation.getChildSlide(slideIndex);
    trace(childSlide._name);
}
```

另请参见

[Slide.numChildSlides](#)

Slide.gotoSlide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.gotoSlide(newSlide)
```

参数

newSlide 要导航到的幻灯片。

返回

一个布尔值 (true 或 false)，指示导航是成功 (true) 还是不成功 (false)。

说明

方法；导航到由 *newSlide* 指定的幻灯片。要使导航成功，必须满足以下条件：

- 当前幻灯片必须是 *mySlide* 的子幻灯片。
- 由 *newSlide* 指定的幻灯片和当前幻灯片必须具有共同的始祖幻灯片，也即当前幻灯片和 *newSlide* 必须位于同一个幻灯片子树中。

如果任何一个条件不满足，导航就会失败，且方法返回 false；否则，方法就会导航到指定的幻灯片并返回 true。

例如，考虑以下幻灯片层次结构：

```
Presentation
  Slide1
    Slide1_1
    Slide1_2
  Slide2
    Slide2_1
    Slide2_2
```

如果当前幻灯片为 Slide1_2，则下面的 gotoSlide() 方法调用将会失败，原因是当前幻灯片不是 Slide2 的子代。

```
Slide2.gotoSlide(Slide2_1);
```

也请考虑下面的屏幕层次结构，在该结构中，窗体对象是两个单独的幻灯片树的父屏幕。

```
Form_1
  Slide1
    Slide1_1
    Slide1_2
  Slide2
    Slide2_1
    Slide2_2
```

如果当前幻灯片是 `Slide1_2`，则下面的方法调用也会失败，原因是 `Slide1` 和 `Slide2` 位于不同的幻灯片子树中。

```
Slide1_2.gotoSlide(Slide2_2);
```

示例

以下附加到 `Button` 组件的代码使用 `Slide.currentSlide` 属性和 `gotoSlide()` 方法进入到演示文稿中的下一张幻灯片。

```
on(click){
    _parent.gotoSlide(_parent.currentSlide.nextSlide);
}
```

请注意，这段代码与以下使用 `Slide.gotoNextSlide()` 方法的代码等效。

```
on(click){
    _parent.currentSlide.gotoNextSlide();
}
```

另请参见

[Slide.currentSlide](#), [Slide.gotoNextSlide\(\)](#)

Slide.gotoFirstSlide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.gotoFirstSlide()
```

返回

无。

说明

方法；导航到 *mySlide* 下面的子幻灯片树中的第一张叶幻灯片。从某张幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数内进行调用时，如果该事件是幻灯片导航的结果，则会忽略此方法。

要转到演示文稿中的第一张幻灯片，请调用 `mySlide.rootSlide.gotoFirstSlide()`。有关 `rootSlide` 的详细信息，请参阅 [Slide.revealChild](#)

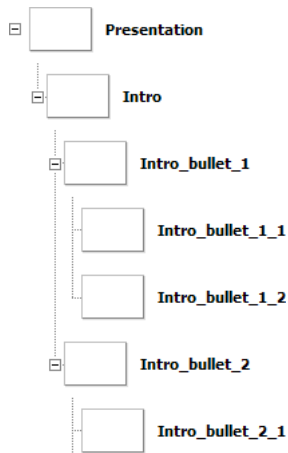
示例

在下面显示的幻灯片层次结构中，下面的方法调用都会导航到名为 `Intro_bullet_1_1` 的幻灯片。

```
Presentation.gotoFirstSlide();
Presentation.Intro.gotoFirstSlide();
Presentation.Intro.Intro_bullet_1.gotoFirstSlide();
```

此方法调用将导航到名为 Intro_bullet_2_1 的幻灯片。

```
Presentation.Intro.Intro_bullet_2.gotoFirstSlide();
```



另请参见

[Slide.firstSlide](#)、[Slide.revealChild](#)

Slide.gotoLastSlide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.gotoLastSlide()
```

返回

无。

说明

方法；导航到 *mySlide* 下面的子幻灯片树中的最后一张叶幻灯片。从某张幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数内进行调用时，如果该事件是另一个幻灯片导航的结果，则会忽略此方法。

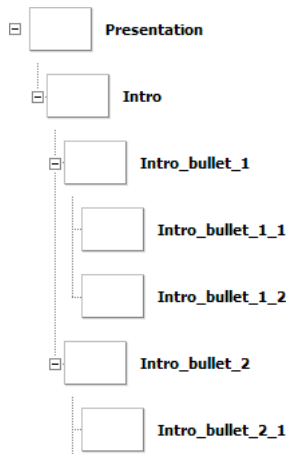
示例

在下面显示的幻灯片层次结构中，下面的方法调用都会导航到名为 Intro_bullet_1_2 的幻灯片。

```
Presentation.Intro.gotoLastSlide();  
Presentation.Intro.Intro_bullet_1.gotoLastSlide();
```

这些方法调用将导航到名为 Intro_bullet_2_1 的幻灯片。

```
Presentation.gotoLastSlide();
Presentation.Intro.gotoLastSlide();
```



另请参见

[Slide.gotoSlide\(\)](#)、[Slide.lastSlide](#)

Slide.gotoNextSlide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.gotoNextSlide()
```

返回

一个布尔值（true 或 false），或 null；如果方法成功地导航到下一张幻灯片，则返回 true；如果在调用方法时，演示文稿已经处于最后一张幻灯片（也即如果 `currentSlide.nextSlide` 为 null），则返回 false；如果不包含当前幻灯片的幻灯片上调用，则返回 null。

说明

方法；导航到幻灯片演示文稿中的下一张幻灯片。当控制从一张幻灯片传到下一张幻灯片时，会隐藏传出的幻灯片并显示传入的幻灯片。如果传出和传入幻灯片位于不同的幻灯片子树中，则所有始祖幻灯片（从传出幻灯片开始，一直到传入和传出幻灯片的公共始祖）都被隐藏起来并接收 `hide` 事件。紧接着，从传入幻灯片的所有始祖幻灯片一直到传出和传入幻灯片的公共始祖都会可见，并且接收 `reveal` 事件。

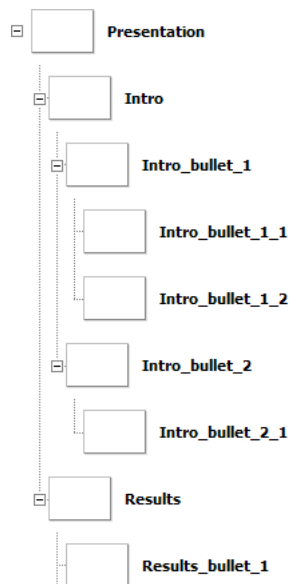
通常，`gotoNextSlide()` 是在代表当前幻灯片的叶节点上调用的。如果在非叶节点 `someNode` 上调用，则 `someNode.gotoNextSlide()` 会进入到下一张幻灯片或“部分”中的第一个叶节点。有关详细信息，请参阅下面的范例。

在不包含当前幻灯片的幻灯片上调用时此方法没有效果（请参见下面的范例）。

另外，从附加到幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数中调用此方法时，如果该处理函数是作为幻灯片导航的结果调用的，则此方法无效。

示例

假设在下面的幻灯片层次结构中，名为 `Intro_bullet_1_1` 的幻灯片是当前查看的幻灯片（即 `_root.Presentation.currentSlide._name == Intro_bullet_1_1`）。



在这种情况下，调用 `Intro_bullet_1_1.gotoNextSlide()` 将导航到 `Intro_bullet_1_2`，它是 `Intro_bullet_1_1` 的同辈幻灯片。

然而，调用 `Intro_bullet_1.gotoNextSlide()` 将导航到 `Intro_bullet_2_1`，它是 `Intro_bullet_2` 包含的第一张叶幻灯片，而 `Intro_bullet_2` 是 `Intro_bullet_1` 的下一张同辈幻灯片。类似地，调用 `Intro.gotoNextSlide()` 将导航到 `Results_bullet_1`，它是 `Results` 幻灯片所包含的第一张叶幻灯片。

另外，仍然假设当前幻灯片是 `Intro_bullet_1_1`，调用 `Results.gotoNextSlide()` 将无效，原因是 `Results` 不包含当前幻灯片（即 `Results.currentSlide` 为 `null`）。

另请参见

[Slide.currentSlide](#)、[Slide.gotoPreviousSlide\(\)](#)、[Slide.nextSlide](#)

Slide.gotoPreviousSlide()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.gotoPreviousSlide()
```

返回

一个布尔值（true 或 false），或 null；如果方法成功地导航到上一张幻灯片，则返回 true；如果在调用方法时，演示文稿处于第一张幻灯片（也即如果 `currentSlide.nextSlide` 为 null），则返回 false；如果在不包含当前幻灯片的幻灯片上调用，则返回 null。

说明

方法；导航到幻灯片演示文稿中的上一张幻灯片。当控制从一张幻灯片传到上一张幻灯片时，会隐藏传出的幻灯片并显示传入的幻灯片。如果传出和传入幻灯片位于不同的幻灯片子树中，则所有始祖幻灯片（从传出幻灯片开始，一直到传入和传出幻灯片的公共始祖）都被隐藏起来并接收 `hide` 事件。紧接着，从传入幻灯片的所有始祖幻灯片一直到传出和传入幻灯片的公共始祖都会可见，并且接收 `reveal` 事件。

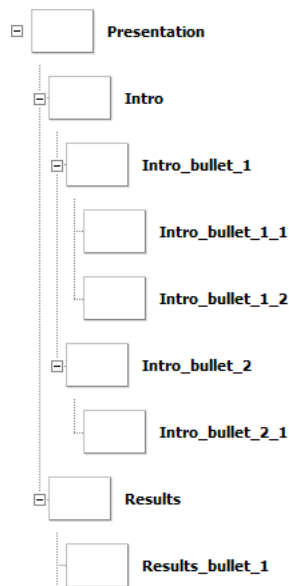
通常，`gotoPreviousSlide()` 是在代表当前幻灯片的叶节点上调用的。如果在非叶节点 `someNode` 上调用，则 `someNode.gotoPreviousSlide()` 会进入到下一张幻灯片或“部分”中的第一个叶节点。有关详细信息，请参阅下面的范例。

在不包含当前幻灯片的幻灯片上调用时此方法没有效果（请参见下面的范例）。

另外还应注意，从附加到幻灯片的 `on(hide)` 或 `on(reveal)` 事件处理函数中调用此方法时，如果该处理函数是作为幻灯片导航的结果调用的，则此方法无效。

示例

假设在下面的幻灯片层次结构中，名为 `Intro_bullet_1_2` 的幻灯片是当前查看的幻灯片（即 `_root.Presentation.currentSlide._name == Intro_bullet_1_2`）。



在这种情况下，调用 `Intro_bullet_1_2.gotoPreviousSlide()` 将导航到 `Intro_bullet_1_1`，它是 `Intro_bullet_1_2` 的上一张同辈幻灯片。

然而，调用 `Intro_bullet_2.gotoPreviousSlide()` 将导航到 `Intro_bullet_1_1`，它是 `Intro_bullet_1` 包含的第一张叶幻灯片，而后者是 `Intro_bullet_2` 的上一张同辈幻灯片。类似地，调用 `Results.gotoPreviousSlide()` 将导航到 `Intro_bullet_1_1`，它是 `Intro` 幻灯片包含的第一张叶幻灯片。

另外，如果当前幻灯片是 `Intro_bullet_1_1`，则调用 `Results.gotoPreviousSlide()` 将无效，原因是 `Results` 不包含当前幻灯片（即 `Results.currentSlide` 为 `null`）。

另请参见

[Slide.currentSlide](#)、[Slide.gotoNextSlide\(\)](#)、[Slide.previousSlide](#)

Slide.hideChild

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

说明

事件；每当幻灯片对象的子幻灯片从可见变为不可见时进行广播。此事件仅由幻灯片对象（而不由窗体对象）广播。`hideChild` 事件的主要用途是将“退出”过渡应用到给定幻灯片的所有子幻灯片。

示例

当附加到根幻灯片（例如，`Presentation` 幻灯片）时，以下代码将在属于根幻灯片的每张子幻灯片出现时显示其名称。

```
on(revealChild) {  
    var child = eventObj.target._name;  
    trace(child + " has just appeared");  
}
```

另请参见

[Slide.revealChild](#)

Slide.indexInParentSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
mySlide.indexInParent
```

说明

属性（只读）；返回 *mySlide* 在其父幻灯片的子幻灯片列表中的索引（从零开始）。

示例

以下代码使用 `indexInParent` 和 `Slide.numChildSlides` 属性显示当前正在查看的幻灯片的索引，以及其父幻灯片所包含的幻灯片总数。要使用这段代码，请将其附加到包含一张或多张子幻灯片的父幻灯片。

```
on(revealChild) {  
    trace("Displaying "+(currentSlide.indexInParentSlide+1)+" of  
        "+currentSlide._parent.numChildSlides);  
}
```

请注意，因为此属性是从零开始的索引，所以其值以一为增量 (`currentSlide.indexInParent+1`) 以显示更多有意义的值。

另请参见

[Slide.numChildSlides](#)、[Slide.revealChild](#)

Slide.lastSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.lastSlide

说明

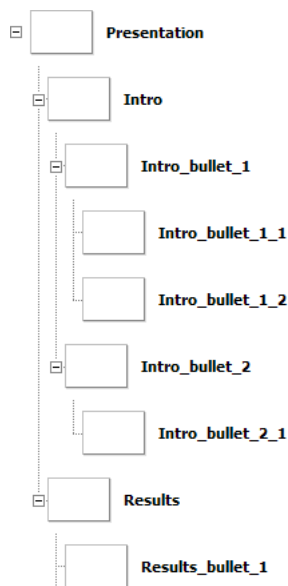
属性（只读）；返回 *mySlide* 的最后一张没有子幻灯片的子幻灯片。

示例

对于下面显示的幻灯片层次结构，以下语句全部为 `true`：

```
Presentation.lastSlide._name == Results_bullet_1;  
Intro.lastSlide._name == Intro_bullet_1_2;
```

```
Intro_bullet_1.lastSlide._name == Intro_bullet_1_2;
Results.lastSlide._name = Results_bullet_1;
```



Slide.nextSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.nextSlide

说明

属性（只读）；返回在调用 *mySlide*.gotoNextSlide() 时会导航到但实际并未导航到的幻灯片。例如，您可以使用此属性显示演示文稿中的下一张幻灯片的名称，并让用户选择是否要导航到该幻灯片。

示例

在此例中，名为 nextButton 的 Button 组件的标签会显示演示文稿中的下一张幻灯片的名称。如果没有下一张幻灯片，也即如果 myslide.nextSlide 为 null，则该按钮的标签会被更新，以指示用户位于此幻灯片演示文稿的末尾。

```
if (mySlide.nextSlide != null) {
    nextButton.label = "Next slide:" + mySlide.nextSlide._name + " > ";
} else {
    nextButton.label = "End of this slide presentation.";
}
```

另请参见

[Slide.gotoNextSlide\(\)](#)、[Slide.previousSlide](#)

Slide.numChildSlides

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.numChildSlides

说明

属性（只读）；返回 *mySlide* 包含的子幻灯片的数量。请注意，幻灯片可以包含窗体或其他幻灯片。如果 *mySlide* 既包含幻灯片又包含窗体，则此属性仅返回幻灯片的数量，而不对窗体计数。

示例

本例使用 `Slide.numChildSlide` 和 `Slide.getChildSlide()` 方法重复 `Presentation` 根幻灯片的所有子幻灯片，并在“输出”面板中显示它们的名称。

```
var numSlides = _root.Presentation.numChildSlides;
for(var slideIndex=0; slideIndex < numSlides; slideIndex++) {
    var childSlide = _root.Presentation.getChildSlide(slideIndex);
    trace(childSlide._name);
}
```

另请参见

[Slide.getChildSlide\(\)](#)

Slide.overlayChildren

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.overlayChildren

说明

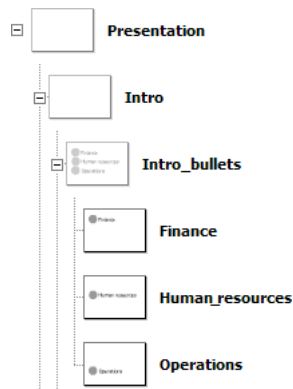
属性；确定在从一张子幻灯片导航到下一张时，*mySlide* 的子幻灯片是否保持可见。如果设置为 `true`，则当控制传到下一张同辈幻灯片时，上一张幻灯片保持可见；如果设置为 `false`，则当控制传到下一张同辈幻灯片时，上一张幻灯片不可见。

在下面的情况下，将此属性设置为 `true` 很有用：给定的幻灯片包含几张分别显示的子“项目符号”幻灯片（可能使用过渡），但都需要在新项目符号出现时保持可见。

注意：此属性仅适用于 *mySlide* 的直接子代，而并不适用于所有（嵌套的）子幻灯片。

示例

例如，以下图示中的 Intro_bullets 幻灯片包含三张子幻灯片（Finance、Human resources 和 Operations），而每张子幻灯片均显示一个单独的项目符号。通过将 Intro_bullets.overlayChildren 设置为 true，每张项目符号幻灯片将在其他项目符号出现时仍保留在舞台上。



Slide.parentIsSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.parentIsSlide

说明

属性（只读）；一个布尔值（true 或 false），指示 *mySlide* 的父对象是否也是幻灯片。如果 *mySlide* 的父对象是幻灯片或者幻灯片的子类，则此属性将返回 true，否则返回 false。

如果 *mySlide* 是演示文稿中的根幻灯片，则此属性将返回 false，原因是 Presentation 幻灯片的父对象是主时间轴 (_level0)，不是幻灯片。如果 *mySlide* 的父对象是窗体，此属性也会返回 false。

示例

下面的代码确定幻灯片 *mySlide* 的父对象本身是否也是幻灯片。如果是，则假设 *mySlide* 为演示文稿中的根幻灯片或主幻灯片，因而没有同辈幻灯片。否则，如果 *mySlide.parentIsSlide* 为 true，则会在“输出”面板中显示 *mySlide* 的同辈幻灯片数量。

```
if (mySlide.parentIsSlide) {  
    trace("I have " + mySlide._parent.numChildSlides+" sibling slides");  
} else {  
    trace("I am the root slide and have no siblings");  
}
```

另请参见

[Slide.numChildSlides](#)

Slide.playHidden

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.playHidden

说明

属性；一个布尔值，指定 *mySlide* 在隐藏时是否应继续播放。当此属性为 `true` 时，*mySlide* 将在隐藏时继续播放。设置为 `false` 时，*mySlide* 将在隐藏时停止播放；当显示时在 *mySlide* 的第 1 帧处重新开始播放。

也可以在 Flash 创作环境的属性检查器中设置此属性。

Slide.previousSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.previousSlide

说明

属性（只读）；返回在调用 *mySlide*.gotoPreviousSlide() 时会导航到但实际不导航到的幻灯片。例如，您可以使用此属性显示演示文稿中的上一张幻灯片的名称，并让用户选择是否要导航到该幻灯片。

示例

在此例中，名为 `previousButton` 的 `Button` 组件的标签会显示演示文稿中的上一张幻灯片的名称。如果没有上一张幻灯片，也即如果 `mySlide.previousSlide` 为 `null`，则该按钮的标签会被更新，以指示用户位于此幻灯片演示文稿的最开始处。

```
if (mySlide.previousSlide != null) {  
    previousButton.label = "Previous slide:" + mySlide.previous._name + " > ";  
} else {  
    previousButton.label = "You are at the beginning of this slide presentation.";
```

另请参见

[Slide.gotoPreviousSlide\(\)](#)、[Slide.nextSlide](#)

Slide.revealChild

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

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

说明

事件；每次当幻灯片对象的子幻灯片从不可见变为可见时进行广播。此事件主要用于将“进入”过渡附加到给定幻灯片的所有子幻灯片。

示例

当附加到根幻灯片（例如，Presentation 幻灯片）时，以下代码将在每张子幻灯片出现时显示其名称。

```
on(revealChild) {  
    var child = eventObj.target._name;  
    trace(child + " has just appeared");  
}
```

另请参见

[Slide.hideChild](#)

Slide.rootSlide

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

mySlide.rootSlide

说明

属性（只读）；返回包含 *mySlide* 的幻灯片树或幻灯片子树的根幻灯片。

示例

假设在幻灯片上有一个影片剪辑，单击它时会转到演示文稿中的第一张幻灯片。要达到此目的，可将以下代码附加到影片剪辑：

```
on(press) {  
    _parent.rootSlide.gotoFirstSlide();  
}
```

在本例中，_parent 是指包含影片剪辑对象的幻灯片。

StyleManager 类

动作脚本类名称 mx.styles.StyleManager

StyleManager 类会跟踪已知的继承样式和颜色。只有在您要创建组件并希望添加新的继承样式或颜色时，才需要使用该类。

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
<code>StyleManager.registerColorName()</code>	使用 StyleManager 注册新的颜色名称。
<code>StyleManager.registerColorStyle()</code>	使用 StyleManager 注册新的颜色样式。
<code>StyleManager.registerInheritingStyle()</code>	使用 StyleManager 注册新的继承样式。

StyleManager.registerColorName()

可用性

Flash Player 6 版本 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 版本 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.registerInheritingStyle()

可用性

Flash Player 6 版本 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");
```

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 248 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 469 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讲，TextArea 组件支持一组件样式。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 25 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。

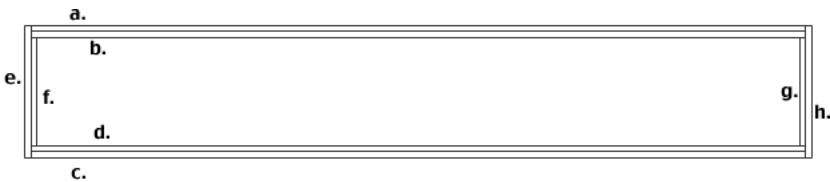
样式	描述
fontFamily	文本的字体名称。
fontSize	字体的磅值。
fontStyle	字体样式，“常规”或“斜体”。
fontWeight	字体粗细，“常规”或“粗体”。
textAlign	文本对齐方式：“左”、“右”或“居中”。
textDecoration	文本修饰，“无”或“下划线”。

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	字母
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：



TextArea 类

继承 UIObject > UIComponent > View > ScrollView > TextArea

动作脚本类名称 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
<code>TextArea.editable</code>	一个布尔值，指明该字段是 (true) 否 (false) 可编辑。
<code>TextArea.hPosition</code>	定义滚动窗格中文本的水平位置。
<code>TextArea.hScrollPolicy</code>	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")。
<code>TextArea.html</code>	一个标记，指明文本字段是否可以采用 HTML 格式。
<code>TextArea.length</code>	文本字段中的字符数。该属性为只读。
<code>TextArea.maxChars</code>	文本字段最多可以容纳的字符数。
<code>TextArea.maxHPosition</code>	<code>TextArea.hPosition</code> 的最大值。
<code>TextArea.maxVPosition</code>	<code>TextArea.vPosition</code> 的最大值。
<code>TextArea.password</code>	一个布尔值，指明字段是密码字段 (true) 还是非密码字段 (false)。
<code>TextArea.restrict</code>	用户可在文本字段中输入的字符集。
<code>TextArea.text</code>	TextArea 组件的文本内容。
<code>TextArea.vPosition</code>	一个数字，指明垂直滚动位置
<code>TextArea.vScrollPolicy</code>	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")。
<code>TextArea.wordWrap</code>	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
<code>TextArea.change</code>	通知侦听器文本已更改。

TextArea.change

可用性

Flash Player 6 版本 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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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 版本 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 版本 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 属性都返回相同的值，但速度更快。制表符（“\t”）这样的字符会被算作一个字符。默认值为 0。

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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 248 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 459 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
<code>color</code>	文本的默认颜色。
<code>embedFonts</code>	文档中嵌入的字体。
<code>fontFamily</code>	文本的字体名称。
<code>fontSize</code>	字体的磅值。
<code>fontStyle</code>	字体样式，“常规”或“斜体”。
<code>fontWeight</code>	字体粗细，“常规”或“粗体”。
<code>textAlign</code>	文本对齐方式：“左”、“右”或“居中”。
<code>textDecoration</code>	文本修饰，“无”或“下划线”。

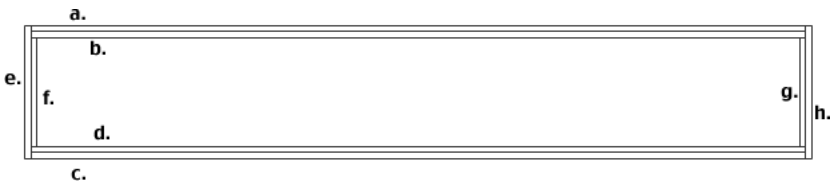
使用具有 `TextInput` 组件的外观

`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：

<code>RectBorder</code> 样式	字母
<code>borderColor</code>	a
<code>highlightColor</code>	b
<code>borderColor</code>	c
<code>shadowColor</code>	d
<code>borderCapColor</code>	e

RectBorder 样式	字母
shadowCapColor	f
shadowCapColor	g
borderCapColor	h

这些样式属性设置边框上的下列位置：



TextInput 类

继承 UIObject > UIComponent > TextInput

动作脚本类名称 mx.controls.TextInput

使用 TextInput 类的属性，您可以在运行时设置文本内容、格式和水平位置。您也可以指明该字段是否可编辑，以及该字段是否为“密码”字段。您还可以限制用户可以输入的字符。

使用“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 248 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 和 UIComponent 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。

属性	描述
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](#) 和 [UIComponent](#) 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](#) 和 [UIComponent](#) 中的所有方法。

TextInput.change

可用性

Flash Player 6 版本 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 `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，如果把下列代码附加到 `myTextInput` 实例，它 will 把 “_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*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 `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, 如果把下列代码附加到 `myTextInput` 实例, 它将把 “_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 `change`), 而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法 ; 当该事件被触发时, 就会调用该方法。该事件被触发时, 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 510 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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 版本 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"
```

TransferObject 接口

动作脚本类名称 mx.data.to.TransferObject

TransferObject 接口定义了 DataSet 组件所管理的项目必须实现的一组方法。
DataSet.itemClassName 属性指定了在每次需要新项目时将实例化的传送对象类的名称。也可以使用属性检查器为所选的 DataSet 组件指定此属性。

TransferObject 接口的方法摘要

方法	描述
TransferObject.clone()	创建传送对象的新实例。
TransferObject.getPropertyData()	返回此传送对象的数据。
TransferObject.setPropertyData()	设置此传送对象的数据。

TransferObject.clone()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
class itemClass implements mx.data.to.TransferObject
{
    function clone() {
        // 此处是您的代码
    }
}
```

返回

传送对象的副本。

说明

方法；创建传送对象的实例。此方法的实现会创建现有传送对象及其属性的副本，然后返回该对象。

示例

```
class itemClass implements mx.data.to.TransferObject {
    function clone():Object {
        var b:ContactClass = new ContactClass();
        for (var p in this) {
            b[p] = this[p];
        }
        return b;
    }
}
```

TransferObject.getPropertyData()

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
class itemClass implements mx.data.to.TransferObject {
    function getPropertyData() {
        // 此处是您的代码
    }
}
```

返回

对象。

说明

方法；返回此传送对象的数据。此方法的实现可能会返回具有属性和相应值的匿名动作脚本对象。

示例

```
class Contact implements mx.data.to.TransferObject {
    function getPropertyData():Object {
        var internalData:Object = { name:name, readOnly:_readOnly, phone:phone,
        zip:zip.zipPlus4 };
        return( internalData );
    }
}
```

TransferObject.setPropertyData()

可用性

Flash Player 7。

版本

Flash MX 2004。

用法

```
class yourClass implements TransferObject {
    function setPropertyData(propData) {
        // 此处是您的代码
    }
}
```


参数

propData 一个包含分配给此传送对象的数据的对象。

返回

无。

说明

方法；设置此传送对象的数据。*propData* 参数是一个对象，其字段包含 DataSet 组件分配给此传送对象的数据。

示例

```
class Contact implements mx.data.to.TransferObject {

    function setPropertyData( data:Object ):Void {
        _readOnly = data.readOnly;
        phone = data.phone;
        zip = new mx.data.types.ZipCode( data.zip );
    }

    public var name:String;
    public var phone:String;
    public var zip:ZipCode;
    private var _readOnly:Boolean; // 指示是否不可改变
}
```

Tree 组件（仅限 Flash Professional）

Tree 组件允许用户查看分层数据。树显示在类似 List 组件的框中，但树中的每个项目称为节点，并且可以是叶 或分枝。默认情况下，用旁边带有文件图标文本字段表示叶，用旁边带有文件夹图标的文本字段表示分枝，并且文件夹图标带有展示三角形，用户可以打开它以显示子项。分枝的子项可以是叶或分枝。

必须通过 XML 数据源提供树组件的数据。有关详细信息，请参阅下一节。

当树实例从单击或 Tab 键切换中获得焦点时，您可以使用以下按键来控制它：

按键	描述
向下箭头	将选区向下移动一项。
向上箭头	将选区向上移动一项。
右箭头	打开所选的分枝节点。如果分枝已打开，则会移到第一个子节点。
左箭头	关闭所选的分枝节点。如果是在已关闭分枝节点的叶节点上，则会移到父节点。
空格键	打开或关闭所选的分枝节点。
End 键	将选区移到列表底端。
Home 键	将选区移到列表顶端。
Page Down 键	将选区向下移动一页。
Page Up 键	将选区向上移动一页。
Ctrl 键	允许选择多个不相邻的项目。
Shift 键	允许选择多个相邻的项目。

无法使用屏幕阅读器来访问 Tree 组件。

使用 Tree 组件（仅限 Flash Professional）

Tree 组件可用于表示分层数据，如电子邮件客户端文件夹、文件浏览器窗格或库存的类别浏览系统。绝大多数情况下，是通过服务器以 XML 的形式来获得树的数据的，但此数据也可以是在 Flash 中创作时所创建的良好 XML。为树创建 XML 的最好方法是使用 [TreeDataProvider 接口（仅限 Flash Professional）](#)。也可以使用动作脚本 XML 类或建立 XML 字符串。创建 XML 数据源（或通过外部源加载一个 XML 数据源）后，需将其指定给 [Tree.dataProvider](#)。

Tree 组件由两组 API 组成：Tree 类和 TreeDataProvider 接口。Tree 类包含可视的配置方法和属性。TreeDataProvider 接口允许您构建 XML 并将其添加到多个树实例中。TreeDataProvider 对象会向任何使用它的树广播更改。另外，存在于与树或菜单相同的帧上的任何 XML 或 XMLNode 对象会自动获得 TreeDataProvider 方法和属性。有关详细信息，请参阅第 497 页的“[TreeDataProvider 接口（仅限 Flash Professional）](#)”。

为 Tree 组件设置 XML 格式

Tree 组件专为显示分层数据结构而设计。XML 是 Tree 组件的数据模型。了解 XML 数据源与 Tree 组件的关系很重要。

请考虑以下的 XML 数据源范例：

```
<node>
  <node label="Mail">
    <node label="INBOX"/>
    <node label="Personal Folder">
      <node label="Business" isBranch="true" />
      <node label="Demo" isBranch="true" />
      <node label="Personal" isBranch="true" />
      <node label="Saved Mail" isBranch="true" />
      <node label="bar" isBranch="true" />
    </node>
    <node label="Sent" isBranch="true" />
    <node label="Trash"/>
  </node>
</node>
```

注意：isBranch 属性是只读属性；无法直接设置它。要设置此属性，请调用 [Tree.setIsBranch\(\)](#) 方法。

XML 数据源中的节点可以有任何名称。请注意，上例中的每个节点都以一般性名称“node”进行命名。树会通读 XML，并根据各节点的嵌套关系建立显示层次结构。

在树中，每个 XML 节点可以显示为两种类型之一：分枝或叶。分枝节点可以包含多个子节点，并显示为带有展示三角形的文件夹图标，此展示三角形使用户可以打开和关闭文件夹。叶节点显示为文件图标，并且不能包含子节点。叶和分枝都可以是根；根节点显示在树的最顶层，并且没有父节点。图标是可自定义的；有关详细信息，请参阅第 486 页的“[使用具有 Tree 组件的外观](#)”。

构建 XML 的方法有很多种。Tree 组件并非设计为可使用所有类型的 XML 结构，因此，使用 Tree 组件可以解释的 XML 非常重要。请勿在子节点中嵌套节点属性；每个节点应包含所有必需的属性。另外，每个节点的属性应一致，这样才实用。例如，要利用 Tree 组件描述信箱结构，请在每个节点上都使用相同的属性（消息、数据、时间、附件等）。这可让树知道它预期要呈现的内容，并可让您以循环方式遍历层次结构，以便比较数据。

当树显示节点时，默认情况下，它使用节点的 label 属性作为文本标签。如果存在任何其他属性，它们会成为树中节点属性的附加属性。

实际的根节点被解释为 Tree 组件本身。这意味着 firstChild（上例中的 <node label="Mail">）在树视图中显示为根节点。这表示树可以有多个根文件夹。在上例中，树中只显示一个根文件夹：“Mail”。但是，如果您将在 XML 中向该层添加同辈节点，则树中会显示多个根节点。

Tree 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 Tree 组件设置的创作参数：

multipleSelection 一个布尔值，它指明用户是 (true) 否 (false) 可以选择多个项目。默认值为 false。

rowHeight 每行的高度（以像素为单位）。默认值为 20。

您可以编写动作脚本，以便使用 Tree 组件的属性、方法和事件来控制该组件的这些和其他选项。有关详细信息，请参阅第 486 页的“Tree 类（仅限 Flash Professional）”。

对于 Tree 组件，您不能像其他组件那样在属性检查器或“组件检查器”面板中为其输入数据参数。有关详细信息，请参阅第 482 页的“使用 Tree 组件（仅限 Flash Professional）”和第 483 页的“创建具有 Tree 组件的应用程序”。

创建具有 Tree 组件的应用程序

在本例中，开发人员正在创建一个电子邮件应用程序，并选择使用 Tree 组件来显示信箱。

不能像其他组件那样在属性检查器或“组件检查器”面板中输入数据参数。由于数据结构比 Tree 组件更复杂，因此，您必须在运行时导入 XML 对象，或在创作时在 Flash 中建立一个 XML 对象。要在 Flash 中创建 XML，可以使用 TreeDataProvider、使用动作脚本 XML 对象或建立 XML 字符串。这些选择会在以下过程中逐一说明。

要将 Tree 组件添加到应用程序：

- 1 在 Flash 中，选择“文件”>“新建”，然后选择“Flash 文档”。
- 2 在“组件”面板中，双击 Tree 组件以将其添加到舞台上。
- 3 在属性检查器中，输入实例名称 **myTree**。
- 4 在第 1 帧的“动作”面板中，输入会创建 change 事件处理函数的以下代码：

```
listenerObject = new Object();
listenerObject.change = function(evtObject){
    trace(evtObject.target.selectedItem.attributes.label + " was selected");
}
myTree.addEventListener("change", listenerObject);
```

每次选定树中的项目时，处理函数中的 trace 动作会向“输出”面板发送一条消息。

- 5 完成以下其中一个过程，为树加载或创建一个 XML 数据源。

要 from 外部文件中加载 XML，请执行以下操作：

- 1 执行上述过程中的各步骤，将 Tree 组件添加到应用程序并创建 change 事件处理函数。
- 2 在“动作”面板中的第 1 帧上，输入以下代码：

```
myTreeDP = new XML();
myTreeDP.ignoreWhite = true;
myTreeDP.load("http://myServer.myDomain.com/source.xml");
myTreeDP.onLoad = function(){
    myTree.dataProvider = myTreeDP;
}
```

此代码创建一个称为 `myTreeDP` 的动作脚本 XML 对象，并调用 `XML.load()` 方法来加载 XML 数据源。然后，代码定义了 `onLoad` 事件处理函数，该函数在 XML 加载时将 `myTree` 实例的 `dataProvider` 属性设置为新的 XML 对象。有关 XML 对象的详细信息，请参阅它在“动作脚本字典”帮助中的条目。

3 选择“控制” > “测试影片”。

在 SWF 文件中，您可以看到在 Tree 组件中显示的 XML 结构。单击 Tree 组件中的各项可以看到 `change` 事件处理函数中将数据值发送到“输出”面板的 `trace` 动作。

在创作时，要使用 `TreeDataProvider` 类在 Flash 中创建 XML，请执行以下操作：

1 执行上述第一个过程中的各步骤，将 Tree 组件添加到一个应用程序并创建 `change` 事件处理函数。

2 在“动作”面板中的第 1 帧上，输入以下代码：

```
var myTreeDP = new XML();
myTreeDP.addTreeNode("Local Folders", 0);

// 使用 XML.firstChild 嵌套 Local Folders 下的子节点
var myTreeNode = myTreeDP.firstChild;
myTreeNode.addTreeNode("Inbox", 1);
myTreeNode.addTreeNode("Outbox", 2);
myTreeNode.addTreeNode("Sent Items", 3);
myTreeNode.addTreeNode("Deleted Items", 4);

// 利用 myTree 组件指定 myTreeDP 数据源
myTree.dataProvider = myTreeDP;

// 将 4 个子节点逐个设置为分枝
for(var i=0; i<myTreeNode.childNodes.length; i++){
    var node = myTreeNode.getTreeNodeAt(i);
    myTree.setIsBranch(node, true);
}
```

此代码创建了一个名为 `myTreeDP` 的 XML 对象。在与 Tree 组件相同的帧上的任何 XML 对象会自动获得 `TreeDataProvider` API 的所有属性和方法。第二行代码创建了一个名为 `Local Folders` 的根节点。有关其余代码的详细信息，请参阅代码中的注释（以 `//` 开头的行）。

3 选择“控制” > “测试影片”。

在 SWF 文件中，您可以看到在 Tree 组件中显示的 XML 结构。单击 Tree 组件中的各项可以看到 `change` 事件处理函数中将数据值发送到“输出”面板的 `trace` 动作。

要使用动作脚本 XML 类创建 XML，请执行以下操作：

1 执行上述第一个过程中的各步骤，将 Tree 组件添加到一个应用程序并创建 `change` 事件处理函数。

2 在“动作”面板中的第 1 帧上，输入以下代码：

```
// 创建 XML 对象
var myTreeDP = new XML();

// 创建节点值
var myNode0 = myTreeDP.createElement("node");
myNode0.attributes.label = "Local Folders";
myNode0.attributes.data = 0;

var myNode1 = myTreeDP.createElement("node");
myNode1.attributes.label = "Inbox";
myNode1.attributes.data = 1;
```

```

var myNode2 = myTreeDP.createElement("node");
myNode2.attributes.label = "Outbox";
myNode2.attributes.data = 2;

var myNode3 = myTreeDP.createElement("node");
myNode3.attributes.label = "Sent Items";
myNode3.attributes.data = 3;

var myNode4 = myTreeDP.createElement("node");
myNode4.attributes.label = "Deleted Items";
myNode4.attributes.data = 4;

// 将节点指定到 XML 树中的层次结构
myTreeDP.appendChild(myNode0);
myTreeDP.firstChild.appendChild(myNode1);
myTreeDP.firstChild.appendChild(myNode2);
myTreeDP.firstChild.appendChild(myNode3);
myTreeDP.firstChild.appendChild(myNode4);

// 利用 Tree 控件指定 myTreeDP 数据源
myTree.dataProvider = myTreeDP;

```

请阅读代码中的注释（以 // 开头的行），以了解代码的说明。有关 XML 对象的详细信息，请参阅它在“动作脚本字典”帮助中的条目。

3 选择“控制” > “测试影片”。

在 SWF 文件中，您可以看到在 Tree 组件中显示的 XML 结构。单击 Tree 组件中的各项可以看到 change 事件处理函数中将数据值发送到“输出”面板的 trace 动作。

在创作时，要使用良构的字符串在 Flash 中创建 XML，请执行以下操作：

1 执行上述第一个过程中的各步骤，将 Tree 组件添加到一个应用程序并创建 change 事件处理函数。

2 在“动作”面板中的第 1 帧上，输入以下代码：

```

myTreeDP = new XML("<node label='Local Folders'><node label='Inbox' data='0' /><node label='Outbox' data='1' /></node>");
myTree.dataProvider = myTreeDP;

```

以上代码创建了一个 XML 对象 myTreeDP，并将其指定给 myTree 的 dataProvider 属性。

3 选择“控制” > “测试影片”。

在 SWF 文件中，您可以看到在 Tree 组件中显示的 XML 结构。单击 Tree 组件中的各项可以看到 change 事件处理函数中将数据值发送到“输出”面板的 trace 动作。

自定义 Tree 组件（仅限 Flash Professional）

在创作过程中和运行时，您都可以在水平和垂直方向上改变 Tree 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\(\)](#)）。当树的宽度不足以显示节点的文本时，文本会缩短。

对 Tree 组件使用样式

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。

使用具有 Tree 组件的外观

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。

Tree 类（仅限 Flash Professional）

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

动作脚本类名称 mx.controls.Tree

Tree 类的方法摘要

方法	描述
<code>Tree.addNode()</code>	向树实例添加节点。
<code>Tree.addNodeAt()</code>	在树实例中的特定位置添加节点。
<code>Tree.getDisplayIndex()</code>	返回给定节点的显示索引。
<code>Tree.getIsBranch()</code>	指定文件夹是否为分枝（具有文件夹图标和展开箭头）。
<code>Tree.getIsOpen()</code>	指示分枝是打开还是关闭。
<code>Tree.getNodeDisplayedAt()</code>	返回给定节点的显示索引。
<code>Tree.getTreeNodeAt()</code>	返回在树的根上的节点。
<code>Tree.removeAll()</code>	从树实例中删除所有节点并刷新树。
<code>Tree.removeTreeNodeAt()</code>	删除在指定位置的节点并刷新树。
<code>Tree.setIsBranch()</code>	指示节点是否为分枝（获得文件夹图标和展开箭头）。
<code>Tree.setIcon()</code>	指定节点是打开还是关闭。
<code>Tree.setIsOpen()</code>	指定将用作节点图标的元件。

继承 UIComponent、UIObject、View、ScrollView、ScrollSelectList 和 List 的所有方法。

Tree 类的属性摘要

属性	描述
<code>Tree.dataProvider</code>	指定 XML 数据源。
<code>Tree.firstVisibleNode</code>	指定显示在最顶层的第一个节点。
<code>Tree.selectedNode</code>	指定树实例中的一个选定节点。
<code>Tree.selectedNodes</code>	指定树实例中的多个选定节点。

继承 UIComponent、UIObject、View、ScrollView、ScrollSelectList 和 List 的所有属性。

Tree 类的事件摘要

事件	描述
<code>Tree.nodeClose</code>	在用户关闭节点时广播。
<code>Tree.nodeOpen</code>	在用户打开节点时广播。

继承 UIComponent、UIObject、View、ScrollView、ScrollSelectList 和 List 的所有事件。

Tree.addTreeNode()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
myTree.addTreeNode(label [, data])
```

用法 2：

```
myTree.addTreeNode(child)
```

参数

label 一个显示节点或带有“label”字段的对象的字符串（或由 `labelField` 属性指定的标签字段名称）。

data 一个与该节点关联的任何类型的对象。此参数是可选的。

child 任何 XMLNode 对象。

返回

添加的 XML 节点。

说明

方法；将子节点添加到树。该节点或者是通过标签和数据参数中提供的信息构建的（用法 1），或者是通过作为 XMLNode 对象的预先建立的子节点构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

注意：调用此方法将刷新视图。

示例

以下代码向 `myTree` 的根添加一个新节点。第二行代码将节点从 `mySecondTree` 的根移到 `myTree` 的根：

```
myTree.addTreeNode("Inbox", 3);  
myTree.addTreeNode(mySecondTree.getTreeNodeAt(3));
```

Tree.addTreeNodeAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1 :

```
myTree.addTreeNodeAt(index, label [, data])
```

用法 2 :

```
myTree.addTreeNodeAt(index, child)
```

参数

index 应遵循的节点添加顺序（在多个子节点之间）。

label 一个显示该节点的字符串。

data 一个与该节点关联的任何类型的对象。此参数是可选的。

child 任何 XMLNode 对象。

返回

添加的 XML 节点。

说明

方法；在树的指定位置添加节点。该节点或者是通过标签和数据参数中提供的信息构建的（用法 1），或者是通过预先建立的 XMLNode 对象构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

注意：调用此方法将刷新视图。

示例

以下范例添加一个新节点，作为 myTree 的根的第二个子项。第二行移动 mySecondTree 中的节点，使其成为 myTree 的根的第四个子项：

```
myTree.addTreeNodeAt(1, "Inbox", 3);  
myTree.addTreeNodeAt(3, mySecondTree.getTreeNodeAt(3));
```

Tree.dataProvider

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.dataProvider
```

描述

属性；dataProvider 属性可以是 XML 或字符串。如果 dataProvider 是 XML 对象，则直接将它添加到树。如果 dataProvider 是字符串，则它必须包含由树读取并被转换为 XML 对象的有效 XML。

可以在运行时从外部源加载 XML，或者在创作时在 Flash 中创建它。要创建 XML，可以使用 TreeDataProvider 的方法或内置的动作脚本 XML 类的方法和属性。也可以创建包含 XML 的字符串。

在与 Tree 组件相同的帧上的 XML 对象会自动包含 TreeDataProvider 的方法和属性。可以使用动作脚本 XML 或 XMLNode 对象。

范例

以下范例导入一个 XML 文件，并将其指定给 Tree 组件的 myTree 实例：

```
myTreeDP = new XML();
myTreeDP.ignoreWhite = true;
myTreeDP.load("http://myServer.myDomain.com/source.xml");
myTreeDP.onLoad = function(){
    myTree.dataProvider = myTreeDP;
}
```

注意：大多数 XML 文件都包含空白，而 Flash 不会忽略该空白，除非您将 ignoreWhite 属性设置为 true。

Tree.firstVisibleNode

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.firstVisibleNode
```

描述

属性；显示在最顶层的第一个节点。如果该节点在某个未展开的节点下，则对 firstVisibleNode 进行的设置将无效。默认值是第一个可见节点，或者，如果没有可见的节点，则默认值未定义。此属性的此值是 XMLNode 对象。

注意：设置此属性类似于设置 List.vPosition。

范例

以下范例将滚动位置设置为显示区的顶部：

```
myTree.firstVisibleNode = myTree.getTreeNodeAt(0);
```

Tree.getIsBranch()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.getIsBranch(node)
```

参数

node 一个 XMLNode 对象。

返回

一个布尔值，指示节点是 (true) 否 (false) 为分枝。

说明

方法；指示指定的节点是否具有文件夹图标和展开箭头（是否为分枝）。在向该节点添加子项时会自动设置此方法。只需调用 `setIsBranch()` 来创建空的文件夹即可。有关详细信息，请参阅 [Tree.setIsBranch\(\)](#)。

示例

以下代码将节点状态指定给变量：

```
var open = myTree.getIsBranch(myTree.getTreeNodeAt(1));
```

另请参见

[Tree.setIsBranch\(\)](#)

Tree.getIsOpen()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.getIsOpen(node)
```

参数

node 一个 XMLNode 对象。

返回

一个布尔值，指示树是 (true) 否 (false) 打开。

说明

方法；指示指定的节点是打开还是关闭。

示例

以下代码将节点状态指定给变量：

```
var open = myTree.getIsOpen(myTree.getTreeNodeAt(1));
```

Tree.getDisplayIndex()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.getDisplayIndex(node)
```

参数

node 一个 XMLNode 对象。

返回

指定节点的索引，或者，如果当前未显示该节点，则为未定义。

说明

方法；返回在 *node* 参数中指定的节点的显示索引。

显示索引是可以在树窗口中查看的项的数组。例如，任何未封闭项的子项都不在显示索引中。显示索引从 0 开始并继续到可见项，而与父项无关。换言之，显示索引是显示的行从 0 开始的行编号。

示例

以下代码获取 myNode 的显示索引：

```
var x = myTree.getDisplayIndex(myNode);
```

Tree.getNodeDisplayedAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.getNodeDisplayedAt(index)
```

参数

index 一个整数，表示树的可查看区中的显示位置。此数字从零开始；在第一个位置的节点为 0，第二个位置为 1，依此类推。

返回

指定的 XMLNode 对象。

说明

方法；将树的某个显示索引映射到在该索引处显示的节点上。例如，如果树的第五行显示了在层次结构中深居第八层的节点，则通过检查 getNodeDisplayedAt(4) 会返回该节点。

显示索引是可以在树窗口中查看的项的数组。例如，任何未封闭项的子项都不在显示索引中。显示索引从 0 开始并继续到可见项，而与父项无关。换言之，显示索引是显示的行从 0 开始的行编号。

注意：在每次节点打开和关闭时会显示索引的更改。

示例

以下代码获取对某个 XML 节点的引用，该节点是在 myTree 中显示的第二行：

```
myTree.getNodeDisplayedAt(1);
```

Tree.getTreeNodeAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.getTreeNodeAt(index)
```

参数

index 树的索引号。

返回

XMLNode 对象。

说明

方法；返回在 myTree 的根上的指定节点。

示例

以下代码获取在 myTree 树中位于第一层上的第二个节点：

```
myTree.getTreeNodeAt(1);
```

Tree.nodeClose

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();  
listenerObject.nodeClose = function(eventObject) {  
    // 此处插入您的代码  
}  
myTreeInstance.addEventListener("nodeClose", listenerObject)
```

说明

事件；在用户关闭 Tree 组件的节点时，向所有注册的侦听器广播。

V2 组件使用调度程序 / 侦听器事件模型。Tree 组件会在其一个节点被单击关闭时广播 nodeClose 事件，该事件由附加到您创建的侦听器对象 (*listenerObject*) 上的函数（也称为处理函数）进行处理。

该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到处理函数。每个事件对象都有一组属性，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。Tree.nodeClose 事件的事件对象还有一个附加属性：node（已关闭的 XML 节点）。有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

示例

在下面的范例中，定义了名为 `myTreeListener` 的处理函数，并且将该处理函数作为第二个参数传递到 `myTree.addEventListener()` 方法。`nodeClose` 处理函数在 `evtObject` 参数中捕获事件对象。在广播 `nodeClose` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
myTreeListener = new Object();
myTreeListener.nodeClose = function(evtObject){
    trace(evtObject.node + " node was closed");
}
myTree.addEventListener("nodeClose", myTreeListener);
```

Tree.nodeOpen

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
listenerObject = new Object();
listenerObject.nodeOpen = function(eventObject){
    // 此处插入您的代码
}
myTreeInstance.addEventListener("nodeOpen", listenerObject)
```

描述

事件；在用户打开 `Tree` 组件上的节点时，向所有注册的侦听器广播。

V2 组件使用调度程序/侦听器事件模型。`Tree` 组件会在用户单击打开一个节点时调度 `nodeOpen` 事件，该事件由附加到您创建的侦听器对象 (`listenerObject`) 上的函数（也称为处理函数）进行处理。您调用 `addEventListener()` 方法并将处理函数的名称作为参数传递给它。

该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到处理函数。每个事件对象都有一组属性，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。`Tree.nodeOpen` 事件的事件对象还有一个附加属性：`node`（已打开的 XML 节点）。有关事件对象的详细信息，请参阅第 510 页的“事件对象”。

范例

在下面的范例中，定义了名为 `myTreeListener` 的处理函数，并且将该处理函数作为第二个参数传递到 `myTree.addEventListener()` 方法。`nodeOpen` 处理函数在 `evtObject` 参数中捕获事件对象。在广播 `nodeOpen` 事件时，`trace` 语句被发送到“输出”面板，如下所示：

```
myTreeListener = new Object();
myTreeListener.nodeOpen = function(evtObject){
    trace(evtObject.node + " node was opened");
}
myTree.addEventListener("nodeOpen", myTreeListener);
```

Tree.removeAll()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.removeAll()
```

参数

无。

返回

无。

说明

方法；删除所有节点并刷新树。

示例

以下代码清空 myTree：

```
myTree.removeAll();
```

Tree.removeTreeNodeAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.removeTreeNodeAt(index)
```

参数

index 树的子项的索引号。树的每个子项均按其创建顺序获得了一个从零开始的索引。

返回

XMLNode 对象，或者，如果出错，则为未定义。

说明

方法；删除树的根上的节点（由其索引位置指定）并刷新树。

示例

以下代码删除 myTree 树的根的第四个子项：

```
myTree.removeTreeNodeAt(3);
```

Tree.setIsBranch()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.setIsBranch(node, isBranch)
```

参数

node 一个 XML 节点。

isBranch 一个布尔值，它表明该节点是 (true) 否 (false) 是分枝。

返回

无。

说明

方法；指定节点是否具有文件夹图标和展开箭头，以及是否具有子项或者可能具有子项。如果节点具有子项，它会自动设置为分枝；在您想创建空的文件夹时，只需调用 `setIsBranch()` 即可。如果在用户打开文件夹时只希望加载子节点，则您可能要创建尚未拥有子项的分枝。

调用 `setIsBranch()` 方法会刷新所有视图。

示例

以下代码使 `myTree` 的一个节点变为一个分枝；

```
myTree.setIsBranch(myTree.getTreeNodeAt(1), true);
```

Tree.setIcon()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.setIcon(node, linkID[, linkID2])
```

参数

node 一个 XML 节点。

linkID 元件的链接标识符，该元件将被用作节点旁的图标。此参数用于叶节点和分枝节点的关闭状态。

linkID2 元件的链接标识符，该元件将被用作节点旁的图标。此参数用于表示分枝节点的打开状态的图标。

返回

无。

说明

方法；为指定的节点指定图标。对于叶节点，此方法使用一个参数 (*linkID*)，而对于分枝，此方法使用两个参数 (*linkID* 和 *linkID2*) (关闭和打开图标)。对于叶 (非分枝) 节点，第二个参数会被忽略。而且，如果只为分枝节点指定了一个参数，则对于关闭和打开状态，它们都使用一个图标。

示例

以下代码指定在 `myTree` 的第二个节点旁使用带有链接标识符 “`imageIcon`” 的元件：

```
myTree.setIcon(myTree.getTreeNodeAt(1), "imageIcon");
```

Tree.setIsOpen()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.setIsOpen(node, isOpen[, noEvent])
```

参数

node 一个 XML 节点。

isOpen 一个布尔值，用于打开一个节点 (`true`) 或关闭它 (`false`)。

noEvent 一个布尔值，它确定是 (`true`) 否 (`false`) 将打开节点的过渡制成动画。此参数是可选的。

返回

无。

说明

方法；打开或关闭节点。

示例

以下代码打开 `myTree` 的一个节点：

```
myTree.setIsOpen(myTree.getTreeNodeAt(1), true);
```

Tree.selectedNode

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.selectedNode
```

描述

属性；指定树实例中的一个选定节点。

范例

以下范例指定 `myTree` 中的第一个子节点：

```
myTree.selectedNode = myTree.getTreeNodeAt(0);
```

另请参见

[Tree.selectedNodes](#)

Tree.selectedNodes

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myTree.selectedNodes
```

描述

属性；指定树实例中的多个选定节点。

范例

以下范例选择 `myTree` 中的第一和第三个子节点：

```
myTree.selectedNodes = [myTree.getTreeNodeAt(0), myTree.getTreeNodeAt(2)];
```

另请参见

[Tree.selectedNode](#)

TreeDataProvider 接口（仅限 Flash Professional）

`TreeDataProvider` 是一个接口；无需对其实例化即可使用它。如果在 SWF 中打包了 `Tree` 类，则该 SWF 中的所有 XML 实例都包含 `TreeDataProvider` API。树中的所有节点都是包含 `TreeDataProvider` API 的 XML 对象。

最好是使用 `TreeDataProvider` API 的方法为 `Tree.dataProvider` 属性创建 XML，原因是，只有 `TreeDataProvider` 才向 `Tree` 组件广播刷新树的显示的事件。内置的 XML 类方法可用于创建 XML，但它们不会广播将刷新显示的事件。

可以使用 `TreeDataProvider` API 的方法来控制数据模型和数据显示。可以将内置的 XML 类方法用于只读的任务，如遍历树的层次结构。

对于保存将显示的文本的属性，可通过指定 `labelField` 或 `labelFunction` 属性来选择该属性。例如，code `myTree.labelField = "fred"`；会致使系统为显示的文本查询属性 `myTreeDP.attributes.fred` 的值。

TreeDataProvider 接口的方法摘要

事件	描述
<code>TreeDataProvider.addNode()</code>	在父节点的末尾添加一个子节点。
<code>TreeDataProvider.addNodeAt()</code>	在父节点上的指定位置添加一个子节点。
<code>TreeDataProvider.getTreeNodeAt()</code>	返回节点的指定子项。
<code>TreeDataProvider.removeTreeNode()</code>	从节点的父项中删除节点和节点的所有后代。
<code>TreeDataProvider.removeTreeNodeAt()</code>	从子节点的索引位置处删除节点和节点的所有后代。

TreeDataProvider 接口的属性摘要

属性	描述
<code>TreeDataProvider.attributes.data</code>	指定与节点关联的数据。
<code>TreeDataProvider.attributes.label</code>	指定将在节点旁边显示的文本。

TreeDataProvider.addNode()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
someNode.addNode(label, data)
```

用法 2：

```
someNode.addNode(child)
```

参数

label 一个显示该节点的字符串。

data 一个与该节点关联的任何类型的对象。

child 任何 XMLNode 对象。

返回

添加的 XML 节点。

说明

方法；在树的根添加子节点。该节点或者是根据标签和数据参数中提供的信息构建的（用法 1），或者是根据作为 XMLNode 对象的预先生成的子节点构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

调用此方法将刷新树实例的显示。

示例

在以下范例中，第一行代码确定将向其添加子项的节点的位置。第二行将一个新节点添加到指定的节点，如下所示：

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.addTreeNode("Inbox", 3);
```

以下代码将一棵树中的某个节点移到另一棵树的根：

```
myTreeNode.addTreeNode(mySecondTree.getTreeNodeAt(3));
```

TreeDataProvider.addTreeNodeAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

用法 1：

```
someNode.addTreeNodeAt(index, label, data)
```

用法 2：

```
someNode.addTreeNodeAt(index, child)
```

参数

index 一个整数，它指示应将节点添加到的子节点中的索引位置。

label 一个显示该节点的字符串。

data 一个与该节点关联的任何类型的对象。

child 任何 XMLNode 对象。

返回

添加的 XML 节点。

说明

方法；在父节点中的指定位置添加子节点。该节点或者是根据标签和数据参数中提供的信息构建的（用法 1），或者是根据作为 XMLNode 对象的预先生成的子节点构建的（用法 2）。添加预先存在的节点会将该节点从其先前的位置上删除。

调用此方法将刷新树实例的显示。

范例

以下代码确定将向其添加节点的节点的位置，并添加作为根的第二个子项的新节点：

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.addTreeNodeAt(1, "Inbox", 3);
```

以下代码移动一棵树中的某个节点，使其成为另一棵树的根的第四个子项：

```
myTreeNode.addTreeNodeAt(3, mySecondTree.getTreeNodeAt(3));
```

TreeDataProvider.attributes.data

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
someNode.attributes.data
```

说明

属性；指定与节点关联的数据。这会将值作为 XML 节点对象中的属性添加。设置此属性不会刷新任何树显示。此属性可以是任何数据类型。

示例

以下代码确定要调整的节点的位置，并设置其 data 属性：

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.attributes.data = "hi"; // <node data = "hi"> 中的结果；
```

另请参见

[TreeDataProvider.attributes.label](#)

TreeDataProvider.attributes.label

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
someNode.attributes.label
```

说明

属性；一个指定为节点显示的文本的字符串。此字符串会被写入到 XML 节点对象的一个属性。设置此属性不会刷新任何树显示。

示例

以下代码确定要调整的节点的位置，并设置其 label 属性。以下代码的结果是 “<node label="Mail">”：

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.attributes.label = "Mail";
```

另请参见

[TreeDataProvider.attributes.data](#)

TreeDataProvider.getTreeNodeAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
someNode.getTreeNodeAt(index)
```

参数

index 一个表示子节点在当前节点中的位置的整数。

返回

指定的节点。

说明

方法；返回节点的指定子节点。

示例

以下代码确定某个节点的位置，然后获取 `myTreeNode` 的第二个子项：

```
var myTreeNode = myTreeDP.firstChild.firstChild;  
myTreeNode.getTreeNodeAt(1);
```

TreeDataProvider.removeTreeNode()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
someNode.removeTreeNode()
```

参数

无。

返回

删除的 XML 节点或未定义（如果发生错误）。

说明

方法；从指定节点的父项中删除该节点和任何后代。

示例

以下代码删除某个节点：

```
myTreeDP.firstChild.removeTreeNode();
```

TreeDataProvider.removeTreeNodeAt()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
someNode.removeTreeNodeAt(index)
```

参数

index 一个整数，指示将被删除的节点的位置。

返回

删除的 XML 节点或未定义（如果发生错误）。

说明

方法；删除由当前节点和子节点的索引位置指定的节点（和所有后代）。调用此方法将刷新视图。

示例

以下代码删除给定节点的第四个子项：

```
myTreeDP.firstChild.removeTreeNodeAt(3);
```

UIComponent

继承 UIObject > UIComponent

动作脚本类名称 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
<code>UIComponent.setFocus()</code>	返回对具有焦点的对象的引用。
<code>UIComponent.setFocus()</code>	将焦点设置到组件实例中。

继承 [UIObject](#) 类的所有方法。

UIComponent 类的属性摘要

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

继承 [UIObject](#) 类的所有属性。

UIComponent 类的事件摘要

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

继承 [UIObject](#) 类的所有事件。

UIComponent.focusIn

可用性

Flash Player 6 版本 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 `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*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 `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*)，而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数 (也称作处理函数) 处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 版本 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 `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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 `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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 版本 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

动作脚本类名称 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 版本 79。

版本

Flash MX 2004 and 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 21 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);
```

不保证执行顺序。无法要求在调用一个侦听器之前先调用另一个侦听器。

事件对象作为参数传递给侦听器。事件对象具有包含有关所发生事件的信息的属性。您可以在侦听器回调函数内使用事件对象来访问有关所发生的事件类型的信息，以及哪个实例广播该事件的信息。在上面的范例中，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 510 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

继承 `MovieClip` > `UIObject`

动作脚本类名称 `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。环绕 `MovieClip` 类允许 [Macromedia](#) 在将来无需中断内容即可添加新事件和扩展功能。同时，对于不熟悉传统的 Flash“影片”和“帧”概念的用户，环绕 `MovieClip` 类还允许他们无需了解这些概念即可使用 API 来创建基于组件的应用程序。

`UIObject` 类实现了以下内容：

- 样式
- 事件
- 按缩放比例调整大小

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
<code>UIObject.createObject()</code>	创建对象的子对象。
<code>UIObject.createClassObject()</code>	创建指定类的对象。
<code>UIObject.destroyObject()</code>	破坏组件实例。
<code>UIObject.invalidate()</code>	标记对象以便在到达下一个帧间隔时进行重绘。
<code>UIObject.move()</code>	将对象移动到要求的位置。
<code>UIObject.redraw()</code>	迫使对象有效以便能在当前帧中绘制。
<code>UIObject.setSize()</code>	将对象调整为所要求的大小。
<code>UIObject.setSkin()</code>	设置对象的外观。

UIObject 类的属性摘要

属性	描述
<code>UIObject.bottom</code>	对象底边位置（相对于其父对象的底边）。只读。
<code>UIObject.height</code>	对象的高度（以像素为单位）。只读。
<code>UIObject.left</code>	对象的左侧位置（以像素为单位）。只读。
<code>UIObject.right</code>	对象的右侧位置（相对于其父对象的右边）。只读。
<code>UIObject.scaleX</code>	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。
<code>UIObject.scaleY</code>	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。
<code>UIObject.top</code>	对象上边缘的位置（相对于其父对象）。只读。
<code>UIObject.visible</code>	一个布尔值，指明对象是可见的 (<code>true</code>) 还是不可见的 (<code>false</code>)。
<code>UIObject.width</code>	对象的宽度（以像素为单位）。只读。
<code>UIObject.x</code>	对象的左侧位置（以像素为单位）。只读。
<code>UIObject.y</code>	返回对象上边缘的位置（相对于其父对象）。只读。

UIObject 类的事件摘要

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

UIObject.bottom

可用性

Flash Player 6 版本 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）。要设置此属性，请调用 `UIObject.move()` 方法。

范例

此范例移动复选框，使其对齐到列表框底边的下面：

```
myCheckbox.move(myCheckbox.x, form.height - listbox.bottom);
```

UIObject.createObject()

可用性

Flash Player 6 版本 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 版本 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 版本 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 版本 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 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 `draw`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您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 510 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 版本 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）。要更改 `height` 属性，请调用 `UIObject.setSize()` 属性。

范例

以下范例增加了复选框的高度：

```
myCheckbox.setSize(myCheckbox.width, myCheckbox.height + 10);
```

UIObject.hide

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

```
on(hide){
```

```

    } ...
    listenerObject = new Object();
    listenerObject.hide = function(eventObject){
    } ...
    componentInstance.addEventListener("hide", listenerObject)

```

描述

事件；在将对象的 `visible` 属性从 `true` 改为 `false` 时广播。

范例

以下处理函数在它所附加到的对象变为不可见时，在“输出”面板中显示一条消息。

```

on(hide){
    trace("I e become invisible.");
}

```

另请参见

[UIObject.reveal](#)

UIObject.getStyle()

可用性

Flash Player 6 版本 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 版本 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 版本 79。

版本

Flash MX 2004。

用法

```
componentInstance.left
```

描述

属性（只读）；一个数字，指明对象的左边相对于其父对象的位置（以像素为单位）。要设置此属性，请调用 [UIObject.move\(\)](#) 方法。

UIObject.load

可用性

Flash Player 6 版本 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 `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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 `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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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(){  
    form2.move(form1.x + 100, form1.y + 100);  
}  
form1.addEventListener("move", formListener);
```

UIObject.move()

可用性

Flash Player 6 版本 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。

范例

此范例将复选框向右移动 10 个像素：

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

UIObject.redraw()

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

```
componentInstance.redraw(always)
```

参数

always 一个布尔值。如果为 `true`，则即使未调用 `invalidate()`，也绘制对象。如果为 `false`，则只有在调用了 `invalidate()` 时才绘制对象。

返回

无。

描述

方法；迫使对象有效以便在当前帧中绘制。

范例

以下范例创建一个复选框和一个按钮并绘制它们，原因是未期望其他脚本修改窗体：

```
form.createClassObject(mx.controls.CheckBox, "cb", 0);  
form.createClassObject(mx.controls.Button, "b", 1);  
form.redraw(true)
```

UIObject.resize

可用性

Flash Player 6 版本 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；通知侦听器对象的大小已被调整。

第一个用法范例使用 `on()` 处理函数，并且必须直接附加到一个组件实例。

第二个用法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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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.reveal

可用性

Flash Player 6 版本 79。

版本

Flash MX 2004。

用法

```
on(reveal) {  
    ...  
}  
listenerObject = new Object();  
listenerObject.reveal = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("reveal", listenerObject)
```

描述

事件；在将对象的 `visible` 属性从 `false` 改为 `true` 时广播。

范例

以下处理函数在它所附加到的对象变为可见时，在“输出”面板中显示一条消息。

```
on(reveal) {  
    trace("I e become visible.");  
}
```

另请参见

[UIObject.hide](#)

UIObject.right

可用性

Flash Player 6 版本 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）。要设置此属性，请调用 [UIObject.move\(\)](#) 方法。

范例

以下范例移动复选框，使其对齐到列表框右边的下面：

```
myCheckbox.move(form.width - listBox.right, myCheckbox.y);
```

UIObject.scaleX

可用性

Flash Player 6 版本 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 版本 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 版本 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 版本 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 版本 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 版本 79。

版本

Flash MX 2004。

用法

componentInstance.top

描述

属性（只读）；一个数字，指明对象的上边缘相对于其父对象的位置（以像素为单位）。要设置此属性，请调用 `UIObject.move()` 方法。

UIObject.unload

可用性

Flash Player 6 版本 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 `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`），而该事件由您创建的侦听器对象 (*listenerObject*) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (*eventObject*) 传递到侦听器对象方法。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 510 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 版本 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 版本 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）。要更改宽度，请调用 [UIObject.setSize\(\)](#) 方法。

范例

以下范例增加了复选框的宽度：

```
myCheckbox.setSize(myCheckbox.width + 10, myCheckbox.height);
```

UIObject.x

可用性

Flash Player 6 版本 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）。要设置此属性，请调用 [UIObject.move\(\)](#) 方法。

范例

以下范例将复选框向右移 10 个像素：

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

UIObject.y

可用性

Flash Player 6 版本 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）。要设置此属性，请调用 [UIObject.move\(\)](#) 方法。

范例

以下范例将复选框向下移动 10 个像素：

```
myCheckbox.move(myCheckbox.x, myCheckbox.y + 10);
```

Web 服务类（仅限 Flash Professional）

在 mx.services 包中找到的类由用于访问使用简单对象访问协议 (SOAP) 的 Web 服务的类组成。此 WebService API 不同于 WebServiceConnector 组件 API。前者是只能在动作脚本代码中使用的一组类，在各种 Macromedia 产品中很常见。后者是 Flash MX 2004 所独有的 API，提供用于 WebServiceConnector 组件的直观创作工具的动作脚本接口。

使 Web 服务类在运行时可用（仅限 Flash Professional）

为了能够在运行时使用 Web 服务类，WebServiceClasses 组件必须位于 FLA 文件的库中。此组件包含运行时类，这些类可让您使用 Web 服务。有关将这些类添加到 FLA 的详细信息，请参阅“使用 Flash”帮助中的“在运行时使用数据绑定和 Web 服务（仅限 Flash Professional）”。

注意：当您向 WebServiceConnector 组件添加到 FLA 时，会自动使这些类对于 Flash 文档可用。

mx.services 包中的类（仅限 Flash Professional）

下表列出了 mx.services 包中的类。这些类紧密地集成在一起，因此当您初次了解这个包时，可能要按照下表中列出类的顺序来阅读信息。

类	描述
WebService 类（仅限 Flash Professional）	通过使用定义了 Web 服务的 WSDL 文件，为调用 Web 服务的方法和处理 Web 服务的回调构建新的 WebService 对象。
PendingCall 类（仅限 Flash Professional）	从 Web 服务的方法调用返回的对象，您实现该对象以处理调用的结果和错误。
Log 类（仅限 Flash Professional）	用于记录与 WebService 对象相关的活动的可选对象。
SOAPCall 类（仅限 Flash Professional）	高级的类，它包含有关 Web 服务操作的信息并提供对某些行为的控制。

Log 类（仅限 Flash Professional）

Log 类是 mx.services 包的一部分，并且计划与 WebService 类一起使用（请参阅第 541 页的“WebService 类（仅限 Flash Professional）”）。有关 mx.data.services 包中的类的概述，请参阅第 528 页的“Web 服务类（仅限 Flash Professional）”。

可以创建新的 Log 对象，以记录与 WebService 对象相关的活动。要在向 Log 对象发送消息时执行代码，请使用 Log.onLog() 回调函数。并不存在日志文件；记录机制是您已在 onLog() 回调中使用的任何举措，例如向 trace 命令发送日志消息。

此对象的构造函数创建一个 Log 对象，该对象可作为可选的参量传递给 WebService 构造函数（请参阅第 541 页的“WebService 类（仅限 Flash Professional）”）。

动作脚本类名称 mx.services.Log

Log 对象的回调摘要

回调	描述
Log.onLog()	向日志对象发送一条日志消息。

Log 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myWebSvcLog = new Log([logLevel] [, logName]);
```

参数

logLevel 用于指示您想在日志中记录的信息类型的日志级别。在 Web 服务代码中，日志消息被细分为类别或级别。Log 对象构造函数的 *logLevel* 参数与这些类别相关。可使用三个 *logLevel*：

- Log.BRIEF：日志记录主要的生命周期事件和错误通知。
- Log.VERBOSE：日志记录所有的生命周期事件和错误通知。
- Log.DEBUG：日志记录度量标准和优良纹理型的事件和错误。

默认的 *logLevel* 是 *log.BRIEF*。

logName 随附于每条日志消息的可选名称。如果您使用了多个日志对象，则可以使用 *logName* 来确定哪个日志记录了给定的消息。

返回

无。

描述

构造函数；创建一个 Log 对象。使用此构造函数创建日志。创建了 Log 对象后，可以将此对象传递给 Web 服务以获取消息。

范例

可以调用 new Log 构造函数，它返回一个传递给 Web 服务的日志对象：

```
// 创建新的日志对象
myWebSvcLog = new Log();
myWebSvcLog.onLog = function(txt)
{
    myTrace(txt)
}
```

然后，将此 Log 对象作为参数传递给 WebService 构造函数：

```
myWebSvc = new WebService("http://www.myco.com/info.wsdl", myWebSvcLog);
```

在 Web 服务代码执行以及向此日志对象发送消息时，会调用 Log 对象的 onLog() 函数。如果您想实时看到日志消息，这是唯一可以放置显示这些消息的代码的地方。

以下是日志消息的范例：

```
7/30 15:22:43 [INFO] SOAP:Decoding PendingCall response
7/30 15:22:43 [DEBUG] SOAP:Decoding SOAP response envelope
7/30 15:22:43 [DEBUG] SOAP:Decoding SOAP response body
7/30 15:22:44 [INFO] SOAP:Decoded SOAP response into result [16 millis]
```

```
7/30 15:22:46 [INFO] SOAP:Received SOAP response from network [6469 millis]
7/30 15:22:46 [INFO] SOAP:Parsed SOAP response XML [15 millis]
7/30 15:22:46 [INFO] SOAP:Decoding PendingCall response
7/30 15:22:46 [DEBUG] SOAP:Decoding SOAP response envelope
7/30 15:22:46 [DEBUG] SOAP:Decoding SOAP response body
7/30 15:22:46 [INFO] SOAP:Decoded SOAP response into result [16 millis]
```

Log.onLog()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myWebSvcLog.onLog = function(message)
```

参数

message 传递给处理函数的日志消息。有关日志消息的详细信息，请参阅第 528 页的“Log 类（仅限 Flash Professional）”。

返回

无。

描述

日志回调函数；向日志文件发送日志消息时，Flash Player 调用此函数。对于记录或显示日志消息的代码（如 trace 命令），此处理函数是一个放置这种代码的好地方。Log 构造函数在第 528 页的“Log 类（仅限 Flash Professional）”中介绍。

范例

以下范例创建一个新的日志对象，将其传递给新的 WebService 对象，并处理记录消息。

```
// 创建新的日志对象
myWebSvcLog = new Log();

// 将此日志对象传递给 Web 服务
myWebService = new WebService(wsdlURI, myWebSvcLog);

// 处理传入的日志消息
myWebSvcLog.onLog = function(message)
{
    mytrace("Log Event:\r myWebSvcLog.message="+message+);
}
```

PendingCall 类（仅限 Flash Professional）

PendingCall 类是 mx.services 包的一部分，并且计划与 WebService 类一起使用（请参阅第 541 页的“WebService 类（仅限 Flash Professional）”）。有关 mx.data.services 包中的类的概述，请参阅第 528 页的“Web 服务类（仅限 Flash Professional）”。

对 WebService 对象调用方法时，WebService 对象会返回一个 PendingCall 对象。PendingCall 对象不是开发人员构建的。使用 PendingCall 对象的 onResult 和 onFault 回调来处理 Web 服务方法的异步响应。如果 Web 服务方法返回错误，则 Flash Player 会调用 PendingCall.onFault 回调，并传递 SOAPFault 对象，该对象代表由服务器 /Web 服务返回的 XML SOAP 错误。如果成功调用 Web 服务，则 Flash Player 会调用 PendingCall.onResult 回调并传递结果对象。结果对象是来自在动作脚本中解码或反序列化的 Web 服务的 XML 响应。有关 WebService 对象的详细信息，请参阅第 541 页的“WebService 类（仅限 Flash Professional）”。

在输出参数多于一个时，PendingCall 对象还可让您访问这些参数。许多 Web 服务只返回一个结果，但有些 Web 服务会返回多个结果。在此 API 中，“返回值”只是指第一个（或唯一一个）结果。PendingCall.getOutputXXX 函数可让您访问所有结果，而不只是访问第一个。因此，当您收到在 onResult() 回调的参量中的“返回值”时，如果有其他您想访问的输出参数，则可使用 getOutputValues()（返回一个数组）和 getOutputValue(index)（返回单个参数）来获取已解码的动作脚本值。

也可以直接访问 SOAPParameter 对象。SOAPParameter 对象是具有两个属性的动作脚本对象：value 和 element，前者包含输出参数的动作脚本值，而后者包含输出参数的 XML 值。以下函数返回一个 SOAPParameter 对象或 SOAPParameter 对象的一个数组，它们包含值 (param.value) 和 XML 元素 (param.element)：getOutputParameters()、getOutputParameterByName(name) 和 getOutputParameter(index)。

动作脚本类名称 mx.services.PendingCall

PendingCall 对象的函数摘要

函数	描述
<code>PendingCall.getOutputParameter()</code>	根据传入的 index 获取 SOAPParameter 对象。
<code>PendingCall.getOutputParameterByName()</code>	根据传入的 localName 获取 SOAPParameter 对象。
<code>PendingCall.getOutputParameters()</code>	获取 SOAPParameter 对象的一个数组。
<code>PendingCall.getOutputValue()</code>	根据传入的索引获取输出值。
<code>PendingCall.getOutputValues()</code>	获取所有输出值的一个数组。

PendingCall 对象的属性摘要

属性	描述
<code>PendingCall.myCall</code>	用于 PendingCall 操作的 SOAPCall 操作描述符。
<code>PendingCall.request</code>	XML 原始格式的 SOAP 请求。
<code>PendingCall.response</code>	XML 原始格式的 SOAP 响应。

PendingCall 对象的回调摘要

回调	描述
<code>PendingCall.onFault()</code>	在方法失败时由 Web 服务调用。
<code>PendingCall.onResult()</code>	在方法已成功且返回结果时调用。

PendingCall 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

描述

PendingCall 对象不是开发人员构建的。相反，对 WebService 对象调用函数时，WebService 对象会返回一个 PendingCall 对象。

PendingCall.getOutputParameter()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myPendingCall.getOutputParameter(var index)
```

参数

index 参数的索引。

返回

具有以下元素的 SOAPParameter 对象：

元素	描述
值	该参数的动作脚本值。
元素	该参数在 SOAP 封套中的原始 XML。

描述

函数；获取 SOAPParameter 对象的另一个输出参数，包含值和 XML 元素。SOAP RPC 调用可以返回多个输出参数。您收到的第一个（或唯一一个）返回值总是在 onResult() 回调的结果参量中，但若访问其他返回值，您必须使用类似本函数的函数或 `getOutputValue()`。`getOutputParameter()` 函数返回第 *n* 个作为 SOAPParameter 对象的参数。

另请参阅 `PendingCall.getOutputValue()`、`PendingCall.getOutputValues()`、`PendingCall.getOutputParameterByName()` 和 `PendingCall.getOutputParameters()`。

范例

在给定以下 SOAP 描述符文件的情况下，`getOutputParameter(1)` 会返回一个 `value="Hi there!"` 和 `element=the <outParam2> XmlNode` 的 `SOAPParameter` 对象。

```
...
<SOAP:Body>
  <rpcResponse>
    <outParam1 xsi:type="xsd:int">54</outParam1>
    <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
    <outParam3 xsi:type="xsd:boolean">true</outParam3>
  </rpcResponse>
</SOAP:Body>
...
```

PendingCall.getOutputParameterByName()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myPendingCall.getOutputParameterByName(var localName)
```

参数

`localName` 参数的本地名称，也就是去掉了所有命名空间信息的 XML 元素的名称。例如，以下两个元素的本地名称都是 `bob`：

```
<bob abc="123">
<xsd:bob def="ghi">
```

返回

具有以下元素的 `SOAPParameter` 对象：

元素	描述
值	该参数的动作脚本值。
元素	该参数在 SOAP 封套中的原始 XML。

描述

函数；获取作为 `SOAPParameter` 对象的任何输出参数，包含值和 XML 元素。SOAP RPC 调用可以返回多个输出参数。您收到的第一个（或唯一一个）返回值总是在 `onResult()` 回调的结果参量中，但若要访问其他返回值，您必须使用类似本函数的 API。
`getOutputParameterByName()` 调用返回带有 `localName` 名称的输出参数。

另请参阅 [PendingCall.getOutputValue\(\)](#)、[PendingCall.getOutputValues\(\)](#)、[PendingCall.getOutputParameter\(\)](#) 和 [PendingCall.getOutputParameters\(\)](#)。

范例

在给定以下 SOAP 描述符文件的情况下，`getOutputParameterByName("outParam2")` 会返回一个 `value="Hi there!"` 和 `element=the <outParam2> XmlNode` 的 `SOAPParameter` 对象。

```
...
<SOAP:Body>
  <rpcResponse>
    <outParam1 xsi:type="xsd:int">54</outParam1>
    <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
    <outParam3 xsi:type="xsd:boolean">true</outParam3>
  </rpcResponse>
</SOAP:Body>
...
```

PendingCall.getOutputParameters()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myPendingCall.getOutputParameterByName()
```

参数

无。

返回

具有以下元素的 `SOAPParameter` 对象的数组：

元素	描述
值	该参数的动作脚本值。
元素	该参数在 SOAP 封套中的原始 XML。

描述

函数；获取 `SOAPParameter` 对象的其他输出参数，包含值和 XML 元素。SOAP RPC 调用可以返回多个输出参数。您收到的第一个（或唯一一个）返回值总是在 `onResult()` 回调的结果参量中，但若访问其他返回值，您必须使用类似本函数的 API 或 `getOutputValues()`。

另请参阅 [PendingCall.getOutputValue\(\)](#)、[PendingCall.getOutputValues\(\)](#)、[PendingCall.getOutputParameterByName\(\)](#) 和 [PendingCall.getOutputParameter\(\)](#)。

PendingCall.getOutputValue()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myPendingCall.getOutputValue(var index)`

参数

`index` 输出参数的索引。第一个参数的索引是 0。

返回

第 *n* 个输出参数。

描述

函数；获取某个输出参数的已解码动作脚本值。SOAP RPC 调用可以返回多个输出参数。您收到的第一个（或唯一一个）返回值总是在 `onResult()` 回调的结果参量中，但若要访问其他返回值，您必须使用类似本函数的 API 或 `getOutputParameter()`。`getOutputValue()` 调用会返回第 *n* 个输出参数。

另请参阅 [PendingCall.getOutputParameter\(\)](#)、[PendingCall.getOutputValues\(\)](#)、[PendingCall.getOutputParameterByName\(\)](#) 和 [PendingCall.getOutputParameters\(\)](#)。

范例

在给定以下 SOAP 描述符文件的情况下，`getOutputValue(2)` 会返回 `true`。

```
...
<SOAP:Body>
  <rpcResponse>
    <outParam1 xsi:type="xsd:int">54</outParam1>
    <outParam2 xsi:type="xsd:string">Hi there!</outParam2>
    <outParam3 xsi:type="xsd:boolean">true</outParam3>
  </rpcResponse>
</SOAP:Body>
...
```

PendingCall.getOutputValues()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`myPendingCall.getOutputValues()`

参数

无。

返回

所有输出参数的已解码值的数组。

描述

函数；获取所有输出参数的已解码动作脚本值。SOAP RPC 调用可以返回多个输出参数。您收到的第一个（或唯一一个）返回值总是在 `onResult()` 回调的结果参量中，但若要访问其他返回值，您必须使用类似本函数的 API 或 `getOutputParameters()`。

另请参阅 `PendingCall.getOutputValue()`、`PendingCall.getOutputParameter()`、`PendingCall.getOutputParameterByName()` 和 `PendingCall.getOutputParameters()`。

PendingCall.myCall

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

`PendingCall.myCall`

描述

属性；与 `PendingCall` 的操作对应的 `SOAPCall` 对象。此 `SOAPCall` 对象包含有关 Web 服务操作的信息并提供对某些行为的控制。有关详细信息，请参阅第 539 页的“[SOAPCall 类（仅限 Flash Professional）](#)”。

范例

以下的 `onResult` 回调通过 `trace` 命令发送 `SOAPCall` 操作的名称。

```
callback.onResult = function(result)
{
    // 检查我的操作名称
    trace("My operation name is " + this.myCall.name);
}
```

PendingCall.onFault()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myPendingCallObj.onFault = function(fault)
{
    // 处理所有错误，例如，通过
    // 通知用户服务器不可用或与技术
    // 支持部门联系
}
```

参数

`fault` 具有属性的错误的已解码动作脚本对象版本。如果 XML 形式的错误信息来自某一服务器，则 `SOAPFault` 对象将是该 XML 的已解码动作脚本版本。

返回到 `PendingCall.onfault()` 的错误对象的类型是 `SOAPFault` 对象。该对象不是由开发人员直接构建的，但作为失败的结果返回。它是 SOAP Fault XML 类型的动作脚本映射。

SOAPFault 属性	描述
<code>faultcode</code>	字符串；一个描述错误的短字符串。
<code>faultstring</code>	字符串；可由人直接读取的关于错误的说明。
<code>detail</code>	字符串；与错误相关的特定于应用程序的信息，例如由 Web 服务引擎返回的堆栈跟踪或其它信息。
<code>element</code>	XML；表示错误的 XML 版本的 XML 对象。
<code>faultactor</code>	字符串；错误的来源（如果未涉及中间项，则为可选）。

返回

无。

描述

`PendingCall` 对象回调函数；在 Web 服务方法失败并返回错误时，您提供这个由 Flash Player 调用的处理函数。错误参数是一个动作脚本 `SOAPFault` 对象。

范例

以下范例处理从 Web 服务方法返回的错误。

```
// 处理在使用 Web 服务方法时返回的所有错误
myPendingCallObj = myWebService.methodName(params)
myPendingCallObj.onFault = function(fault)
{
    // 捕获 SOAP 错误
    DebugOutputField.text = fault.faultstring;

    // 添加代码以处理任何错误，例如，通过
    // 通知用户服务器不可用或与技术
    // 支持部门联系
}
```

PendingCall.onResult()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myPendingCallObj.onResult = function(result)
{
    // 捕获结果并为此应用程序对其进行处理
}
```

参数

result 由 Web 服务方法利用 `myPendingCallObj = myWebService.methodName(params)` 调用时返回的 XML 结果（已解码的动作脚本对象版本）。

返回

无。

描述

PendingCall 对象回调函数；在 Web 服务方法成功并返回结果时，您提供这个由 Flash Player 调用的处理函数。结果是由操作返回的 XML 的已解码动作脚本对象版本。要获取返回的原始 XML 而不是已解码的结果，请访问 `PendingCall.response` 属性（请参阅 [PendingCall.response](#)）。

范例

以下范例处理从 Web 服务方法返回的结果。

```
// 处理在使用 Web 服务方法时返回的结果
myPendingCallObj = myWebService.methodName(params)
myPendingCallObj.onResult = function(result)
{
    // 捕获结果并为此应用程序对其进行处理
    ResultOutputField.text = result;
}
```

PendingCall.request

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
rawXML = myPendingCallback.request;
```

描述

PendingCall 属性；包含利用 `myPendingCallback = myWebService.methodName()` 发送的当前请求的原始 XML 格式。通常，您不会用到 `PendingCall.request`，但如果对通过线路发送的 SOAP 感兴趣，则可以使用该属性。使用此属性可访问请求的原始 XML。使用 `myPendingCallback.onResult()` 可获取请求结果的动作脚本版本。

PendingCall.response

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
rawXML = myPendingCallback.response;
```

描述

PendingCall 属性；包含对利用 myPendingCallback = myWebService.methodName() 发送的最新 Web 服务方法调用的响应的原始 XML 格式。通常，您不会用到 PendingCall.response，但如果对通过线路发送的 SOAP 感兴趣，则可以使用该属性。使用 myPendingCallback.onResult() 可获取请求结果的对应动作脚本版本。

SOAPCall 类（仅限 Flash Professional）

SOAPCall 类是 mx.services 包的一部分，并且计划作为高级功能与 WebService 类一起使用（请参阅第 541 页的“WebService 类（仅限 Flash Professional）”。有关 mx.data.services 包中的类的概述，请参阅第 528 页的“Web 服务类（仅限 Flash Professional）”。

创建新的 WebService 对象时，它包含与传入的 WSDL URL 中的操作对应的方法。另外，也会在后台为 WSDL 中的每个操作创建一个 SOAPCall 对象。SOAPCall 是该操作的描述符，因此包含与该特定操作有关的所有信息（该 XML 在网络上的外观和操作样式等）。它还提供对某些行为的控制。可以通过使用 getCall(operationName) 函数获取给定操作的 SOAPCall。对于每一操作只有一个 SOAPCall，由对该操作的所有活动调用共享。一旦获取 SOAPCall，即可通过执行以下操作自定义描述符：

- 打开 / 关闭 XML 响应的解码。
- 打开 / 关闭将 SOAP 数组转换为动作脚本对象的延迟。
- 更改给定操作的并发性配置。
- 向 SOAPCall 对象添加标题。

动作脚本类名称 mx.services.SOAPCall

SOAPCall 对象的函数摘要

函数	描述
SOAPCall.addHeader()	向 SOAPCall 对象添加标题。

SOAPCall 对象的属性摘要

属性	描述
SOAPCall.concurrency	更改给定操作的并发性配置。
SOAPCall.doDecoding	打开 / 关闭 XML 响应的解码。
SOAPCall.doLazyDecoding	打开 / 关闭将 SOAP 数组转换为动作脚本对象的延迟。

SOAPCall 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

描述

SOAPCall 对象不是开发人员构建的。相反,在对 WebService 对象调用方法时,WebService 对象会返回一个 PendingCall 对象。要访问关联的 SOAPCall 对象,请使用 `myPendingCall.myCall`。

SOAPCall.addHeader()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
SOAPCall.addHeader(var header)
```

参数

header 要添加的标题。

返回

无。

描述

函数 ; 向 SOAPCall 对象添加标题。

范例

以下范例创建一个新的 SOAP 标题并将其附加到 SOAPCall。以下代码 :

```
import mx.services.QName;

var qname = new QName("bar", "http://foo");
var value = "hi there!";
var header = new SOAPHeader(qname, value);
soapCall.addHeader(header);

创建以下 SOAP 标题 :

...
<SOAP:Header>
  <ns1:bar
    xmlns:ns1="http://foo"
    xsi:type="xsd:string">hi there!</ns1:bar>
</SOAP:Header>
...
```

SOAPCall.concurrency

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
SOAPCall.concurrency
```

描述

属性；并发请求的数量。下表列出了可能的值：

值	描述
SOAPCall.MULTIPLE_CONCURRENCY	允许多个活动的调用。
SOAPCall.SINGLE_CONCURRENCY	一次只允许一个调用，且前提是某个调用在处于活动状态后出错。
SOAPCall.LAST_CONCURRENCY	通过取消以前的调用，只允许一个调用。

SOAPCall.doDecoding

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
SOAPCall.doDecoding
```

描述

属性；打开 / 关闭 XML 响应的解码 - 默认情况下，XML 响应被转换（解码）为动作脚本对象。如果只想要 XML，可以设置 `SOAPCall.doDecoding = false`。

SOAPCall.doLazyDecoding

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
SOAPCall.doLazyDecoding
```

描述

属性；打开 / 关闭数组的“迟缓解码”。默认情况下，我们使用“迟缓的”解码算法将 SOAP 数组转换为动作脚本对象这一操作推迟到最后一刻，从而可大大加快操作在返回较大的数据集时的返回速度。这意味着您从远端取回的所有数组都是 `ArrayProxy` 对象。然后，当您访问特定的索引 (`foo[5]`) 时，该元素在必要时会自动解码。可通过设置 `SOAPCall.doLazyDecoding = false` 来关闭此行为（这将会导致所有数组全部解码）。

WebService 类（仅限 Flash Professional）

WebService 类是 `mx.services` 包的一部分，并且计划与以下类一起使用：

- [Log 类（仅限 Flash Professional）](#)

- [PendingCall 类（仅限 Flash Professional）](#)
- [SOAPCall 类（仅限 Flash Professional）](#)

注意：此 WebService API 不同于 WebServiceConnector 组件 API。前者是只能在动作脚本代码中使用的一组类，在各种 Macromedia 产品中很常见。后者是 Flash MX 2004 所独有的 API，提供用于 WebServiceConnector 组件的直观创作工具的动作脚本接口。

有关 mx.services 包中的类的概述，请参阅第 528 页的“[Web 服务类（仅限 Flash Professional）](#)”。

WebServices 对象充当远程 Web 服务的本地引用。创建新的 WebService 对象时，会下载和分析定义 Web 服务的 WSDL 文件，并将其放在该对象中。然后，可以直接对该 WebService 对象调用 Web 服务的方法，并处理该 Web 服务的任何回调。当成功处理了 WSDL 且 WebService 对象准备就绪时，会调用 onLoad() 回调。如果在加载 WSDL 时遇到问题，会调用 onFault() 回调。

对 WebService 对象调用方法时，返回值是一个回调对象。从所有 Web 服务方法调用返回的回调的对象类型是 PendingCall。这些对象通常不是开发人员构建的，而是作为 webServiceObject.webServiceMethodName() 命令的结果自动构建的。这些对象并不是稍后进行的 WebService 调用的结果。而 PendingCall 对象表示正在进行的调用。当 WebService 操作完成时（通常是在进行方法调用后的几秒钟），会填写各种 PendingCall 数据字段，而且会调用您提供的 onResult 或 onFault 回调。有关 PendingCall 对象的详细信息，请参阅第 531 页的“[PendingCall 类（仅限 Flash Professional）](#)”。

Player 会在分析 WSDL 之前将您进行的所有调用进行排队，并尝试在分析 WSDL 之后执行这些调用。这是因为，WSDL 包含正确编码和发送 SOAP 请求所需的信息。您在分析 WSDL 后进行的函数调用无需排队；它们会立即进行。如果某个排队的调用与 WSDL 中定义的任何操作的名称都不匹配，Flash Player 会向您在初次进行该调用时被指定的回调对象返回一个错误。

动作脚本类名称 mx.services.WebService

使用 WebServices API

随附在 mx.services 包中的 WebServices API 由 WebService 类、Log 类和 PendingCall 类组成。

支持的类型

WebService 功能支持在以下表格中定义的 XML 架构类型的子集。

也支持复杂的类型和 SOAP 编码的数组类型，而这些类型可以由其他复杂的类型、数组或内置的 XML 架构类型组成：

数字简单类型

XML 架构类型	动作脚本绑定
decimal	Number
integer	Number
negativeInteger	Number
nonNegativeInteger	Number
positiveInteger	Number

XML 架构类型	动作脚本绑定
long	Number
int	Number
short	Number
byte	Number
unsignedLong	Number
unsignedShort	Number
unsignedInt	Number
unsignedByte	Number
float	Number
double	Number

日期和时间简单类型

XML 架构类型	动作脚本绑定
date	Date 对象
datetime	Date 对象
duration	Date 对象
gDay	Date 对象
gMonth	Date 对象
gMonthDay	Date 对象
gYear	Date 对象
gYearMonth	Date 对象
time	Date 对象

名称和字符串简单类型

XML 架构类型	动作脚本绑定
string	动作脚本字符串
normalizedString	动作脚本字符串
QName	mx.services.Qname 对象

布尔型

XML 架构类型	动作脚本绑定
Boolean	Boolean

对象类型

XML 架构类型	动作脚本绑定
Any	XML 对象
Complex Type	由任何支持的类型的属性组成的动作脚本对象
Array	由任何支持的对象或类型组成的动作脚本数组

支持的 XML 架构对象元素

```
schema
  complexType
    complexContent
      限制
    sequence | simpleContent
      限制
  element
    complexType | simpleType
```

WebService 安全性

WebService API 符合 Flash Player 的安全性模型。

用户身份验证和授权

对于 WebService API，身份验证和授权规则与对于 Flash 的任何 XML 网络操作的规则相同。SOAP 本身并未指定任何身份验证和授权的方法。例如，当下层的 HTTP 传送给 HTTP 标题中返回 HTTP BASIC 响应时，浏览器会通过以下方式响应：为用户显示一个对话框，并随后将用户的输入附加到后续消息中的 HTTP 标题。此机制存在于比 SOAP 低的级别上，并且是 Flash HTTP 身份验证设计的一部分。

消息完整性

消息级的安全性包括：在网络包之上的传送 SOAP 消息的概念层对 SOAP 消息本身的加密。

传送安全性 Flash Player SOAP Web 服务的下层网络传送始终是 HTTP POST。因此，可以应用在 Flash HTTP 传送层上的任何安全性方式（例如 SSL）通过 Flash 中的 Web 服务调用获得支持。SSL/HTTPS 为 SOAP 消息传递提供最常用的传送安全性方式，而在传送层与 SSL 结合使用 HTTP BASIC 身份验证是当今最常用的 Web 站点安全性方式。

WebService 对象的函数摘要

函数	描述
<code>WebService.myMethodName()</code>	调用由 WSDL 定义的特定 Web 服务操作。
<code>WebService.getCall()</code>	获取给定操作的 SOAPCall

WebService 对象的回调摘要

回调	描述
<code>WebService.onLoad()</code>	当 Web 服务已成功加载并分析了其 WSDL 文件时调用。
<code>WebService.onFault()</code>	在 WSDL 分析的过程中出错时调用。

WebService 类的构造函数

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myWebServiceObject = new WebService(wsdlURI [, logObject]);
```

参数

构造函数的参数如下所示：

wsdlURI Web 服务 WSDL 文件的 URL。

logObject 为此 Web 服务指定 Log 对象名称的可选参数（请参阅第 528 页的“Log 类（仅限 Flash Professional）”）。

返回

无。

描述

要创建 WebService 对象，请调用 `new WebService()` 并提供 WSDL URL。Flash Player 返回一个 WebService 对象。WebService 对象构造函数可以选择性地接受 Log 对象和代理 URL：

```
myWebServiceObject = new WebService(wsdlURI [, logObject]);
```

如果需要，可以对 WebService 对象使用两个回调。在 Flash Player 完成 WSDL 文件的分析并且对象完成时，它会调用 `webServiceObject.onLoad(WSDLDocument)` 函数。对于您只想在完全分析 WSDL 文件后才执行的代码，这是一个放置此代码的好地方。例如，您可能会选择将第一个 Web 服务方法调用放在此函数中。

如果在查找或分析 WSDL 文件时出错，Flash Player 会调用 `webServiceObject.onFault(fault)`。对于调试代码和那些告知用户“服务器不可用，请稍后重试”之类信息的代码，这是放置它们的好地方。有关详细信息，请参阅这些函数的各个条目。

调用 Web 服务操作：将 Web 服务操作作为直接在 Web 服务上可用的方法进行调用。例如，如果 Web 服务具有方法 `getCompanyInfo(tickerSymbol)`，则按以下方式调用该方法：

```
myPendingCallObject = myWebServiceObject.getCompanyInfo(tickerSymbol);
```

在上例中，回调对象被命名为 `myPendingCallObject`。所有的方法调用都是异步的，并返回 `PendingCall` 类型的回调对象。异步表示不能立即获得 Web 服务调用的结果。

当您调用

```
x = stockService.getQuote("macr");
```

时，对象 `x` 不是 `getQuote` 的结果（它是 `PendingCall` 对象）。只能在稍后（通常是几秒钟后）Web 服务操作完成时获得实际结果。动作脚本代码由对 `onResult` 回调函数的调用进行通知。

处理 `PendingCall` 对象：此回调对象是一个 `PendingCall` 对象，用于处理已调用的 Web 服务方法的结果和错误（请参阅第 531 页的“[PendingCall 类（仅限 Flash Professional）](#)”）。例如：

```
MyPendingCallObject = myWebServiceObject.myMethodName(param1, ..., paramN);
MyPendingCallObject.onResult = function(result)
{
    OutputField.text = result
}
MyPendingCallObject.onFault = function(fault)
{
    DebugField.text = fault.faultCode + "," + fault.faultstring;

    // 添加代码以处理任何错误，例如，通过
    // 通知用户服务器不可用或与技术
    // 支持部门联系
}
```

WebService.getCall()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
getCall(var operationName)
```

参数

operationName 您想获取的相应 `SOAPCall` 的 Web 服务操作。

返回

`SOAPCall` 对象。

描述

创建新的 `WebService` 对象时，它包含与传入的 WSDL URL 中的操作对应的方法。另外，也会在后台为 WSDL 中的每个操作创建一个 `SOAPCall` 对象。`SOAPCall` 是该操作的描述符，因此包含与该特定操作有关的所有信息（该 XML 在网络上的外观和操作样式等）。它还提供对某些行为的控制。可以通过使用 `getCall(operationName)` 方法获取给定操作的 `SOAPCall`。对于每一操作只有一个 `SOAPCall`，由对该操作的所有活动调用共享。一旦获取了 `SOAPCall`，即可通过使用 `SOAPCall` API 更改操作符的描述符。有关详细信息，请参阅第 539 页的“[SOAPCall 类（仅限 Flash Professional）](#)”。

范例

有关使用此调用的范例，请参阅第 539 页的“[SOAPCall 类（仅限 Flash Professional）](#)”。

WebService.onFault()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

MyWebServiceObject.onFault

参数

fault 具有属性的错误的已解码动作脚本对象版本。如果 XML 形式的错误信息来自某一服务器，则 SOAPFault 对象将是该 XML 的已解码动作脚本版本。

返回给 *webservice.onFault()* 方法的错误对象类型是 SOAPFault 对象。该对象不是由开发人员直接构建的，但作为失败的结果返回。它是 SOAP Fault XML 类型的动作脚本映射。

SOAPFault 属性	描述
<i>faultcode</i>	字符串；描述错误的标准短 QName。
<i>faultstring</i>	字符串；可由人直接读取的关于错误的说明。
<i>detail</i>	字符串；与错误相关的特定于应用程序的信息，例如由 Web 服务引擎返回的堆栈跟踪或其它信息。
<i>element</i>	XML；表示错误的 XML 版本的 XML 对象。
<i>faultactor</i>	字符串；错误的来源（如果未涉及中间项，则为可选）。

返回

无。

描述

WebService 回调函数；当新的 *webService(WSDLurl)* 方法已失败且返回错误时，Flash Player 会调用此函数。这可能发生在无法分析或找到 WSDL 文件的时候。错误参数是一个动作脚本 SOAPFault 对象。

范例

以下范例处理在 WebService 对象的创建中返回的任何错误。

```
MyWebServiceObject.onFault = function(fault)
{
    // 捕获错误
    DebugOutputField.text = fault.faultstring;

    // 添加代码以处理任何错误，例如，通过
    // 通知用户服务器不可用或与技术
    // 支持部门联系
}
```

WebService.onLoad()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
myService.onLoad
```

参数

`wsdlDocument` WSDL XML 文档。

返回

无。

描述

Webservice 回调函数；当 WebService 对象已成功加载并分析了其 WSDL 文件时，Flash Player 会调用此回调。在此事件发生之前，可以在应用程序中调用操作，但当此事件发生时，将在内部对这些操作进行排队，并在 WSDL 加载之前不会实际传送这些操作。

范例

以下范例指定 WSDL URL，创建新的 Web 服务对象，并在加载后接收 WSDL 文档。

```
// 指定 WSDL URL
var wsdlURI = "http://www.flash-db.com/services/ws/companyInfo.wsdl";

// 创建新的 Web 服务对象
stockService = new WebService(wsdlURI);

// 加载后接收 WSDL 文档
stockService.onLoad = function(wsdlDocument);
{
    // 当 WSDL 加载完成且
    // 对象已被创建时执行的代码
}
```

WebService.myMethodName()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
callbackObj = myWebServiceObject.myMethodName(param1, ... paramN);
```

参数

所需的参数取决于所调用的 Web 服务方法。

返回

callbackObj 可以将用于处理有关调用的结果和错误的函数附加到的 PendingCall 对象。有关详细信息，请参阅第 531 页的“PendingCall 类（仅限 Flash Professional）”。

在从 WebService 方法返回的响应是 PendingCall.onResult() 或 onFault() 时调用的回调。通过唯一地标识回调对象，可以管理多个 onResult 回调，如下例所示：

```
myWebService = new WebService("http://www.myCompany.com/myService.wsdl");
callback1 = myWebService.getWeather("02451");
callback1.onResult = function(result)
{
    // 完成一些操作
}
callback2 = myWebService.getDetailedWeather("02451");
callback2.onResult = function(result)
{
    // 完成另一些操作
}
```

描述

要调用 Web 服务操作，将其作为在 Web 服务上直接可用的方法进行调用。例如，如果 Web 服务具有方法 getCompanyInfo(tickerSymbol)，则调用：

```
myCallbackObject.myservice.getCompanyInfo(tickerSymbol);
```

所有调用都是异步的，并返回 PendingCall 对象类型的回调对象。

WebServiceConnector（仅限 Flash Professional）

WebServiceConnector 组件使您可以使用业界标准的 SOAP（简单对象访问协议）协议来访问服务器公开的远程方法。Web 服务可以接收参数和返回结果。使用 Flash MX Professional 2004 创作工具和 WebServiceConnector 组件，您可以内省、访问和绑定远程 Web 服务和 Flash 应用程序之间的数据。WebServiceConnector 组件的单个实例可用于对同一操作进行多个调用。您需要为想调用的每个不同操作使用不同的 WebServiceConnector 实例。

Web 服务定义一些方法（有时称为操作），它们可用于通过使用 Web 服务描述语言 (WSDL) 格式的 XML 进行的消耗。WSDL 文件指定由 Web 服务公开的操作、参数和结果（称为架构）的列表。

使用 URL 可访问 WSDL 文件。在 Flash MX Professional 2004 中，可以通过使用“Web 服务”面板输入 Web 服务的 WSDL 文件的 URL 查看该服务的架构。一旦标识了 WSDL 文件，则 Web 服务可用于您创建的任何应用程序。

只有 WSDL 文件的创作者可以更改 WSDL 文件或操作参数。每当创作者更改 WSDL 文件时，params 和结果架构会随之更新。这些更改将覆盖开发人员对架构所作的所有编辑。要获取更新的 WSDL 文件，您可以从“Web 服务”面板菜单中选择“刷新 Web 服务”。

WebServiceConnector 组件和 XMLConnector 组件实现了 RPC（远程过程调用）组件 API、一组方法、属性，以及用于定义向外部数据源发送参数和从其接收结果的便捷方法的事件。

WebServiceConnector 组件的单个实例可用于对同一操作进行多个调用。您需要为想调用的每个不同操作使用不同的 WebServiceConnector 实例。

开发人员可以编辑架构，以便为了在应用程序中使用而对其进行自定义（例如，提供其他的格式设置或验证设置）。请参阅“使用 Flash”帮助中的“在“架构”选项卡中处理架构（仅限 Flash Professional）”。

使用 WebServiceConnector（仅限 Flash Professional）

可以使用 WebServiceConnector 连接到 Web 服务，并使 Web 服务的属性可用于绑定到应用程序中的 UI 组件的属性。要连接到 Web 服务，您必须首先为该 Web 服务输入 Web 服务 URL。WebServiceConnector 在应用程序创作的过程中显示在舞台上，但在运行时应用程序中并没有可视的外观。

可以在“组件检查器”面板或“Web 服务”面板中为 Web 服务输入 URL。请参阅“使用 Flash”帮助中的“WebServiceConnector 组件”。

有关使用 WebServiceConnector 组件的详细信息，请参阅“使用 Flash”帮助中的“数据绑定（仅限 Flash Professional）”。

WebServiceConnector 参数

以下列出了一些创作参数，您可以在“组件检查器”面板的“参数”选项卡中为每个 WebServiceConnector 组件实例设置这些参数：

`multipleSimultaneousAllowed`（布尔值类型）指明是否可以同时进行多个调用；默认值是 `false`。如果为 `false`，则当一个调用已在执行时，`trigger()` 函数将不执行调用。将发出 `status` 事件，并且具有代码 `CallAlreadyInProgress`。如果值为 `true`，将发生调用。

`operation`（字符串类型）是显示在 WSDL 文件中的 SOAP 端口内的操作名称。

`suppressInvalidCalls`（布尔型）指示在参数无效时是否禁止调用；默认值为 `false`。如果为 `true`，则 `trigger()` 函数在绑定数据的参数未通过验证的情况下将不执行调用。将发出 `status` 事件，并且具有代码 `InvalidParams`。如果为 `false`，将进行该调用，并且根据需要使用无效数据。

`WSDLURL`（字符串类型）是定义 Web 服务操作的 WSDL 文件的 URL。在创作期间设置此 URL 时，立即获取和分析该 WSDL 文件。可以在“组件检查器”面板的“架构”选项卡上看到产生的参数和结果信息。还会将服务描述添加到“Web 服务”面板。例如，请参阅 <http://www.xmethods.net/sd/2001/TemperatureService.wsdl>。

WebServiceConnector 类（仅限 Flash Professional）

继承 RPC > WebServiceConnector

动作脚本类名称 `mx.data.components.webServiceConnector`

WebServiceConnector 类的属性摘要

属性	描述
<code>webServiceConnector.multipleSimultaneousAllowed</code>	指明是否可以同时进行多个调用。
<code>webServiceConnector.multipleSimultaneousAllowed</code>	指示显示在 WSDL 文件中的 SOAP 端口内的操作名称。
<code>webServiceConnector.params</code>	指定在执行下一个 <code>trigger()</code> 操作时要发送到服务器的数据。
<code>webServiceConnector.results</code>	标识作为 <code>trigger()</code> 操作的结果从服务器接收的数据。
<code>webServiceConnector.suppressInvalidCalls</code>	指明在参数无效的情况下是否禁止调用。

属性	描述
<code>webServiceConnector.timeout</code>	指定在结果未返回的情况下 Web 服务连接将在多少秒内失败。
<code>webServiceConnector.WSDLURL</code>	指定定义 Web 服务操作的 WSDL 文件的 URL。

WebServiceConnector 类的方法摘要

方法	描述
<code>webServiceConnector.trigger()</code>	启动远程过程调用。

WebServiceConnector 类的事件摘要

事件	描述
<code>webServiceConnector.result</code>	在对 Web 服务进行的调用成功完成时广播。
<code>webServiceConnector.send</code>	当 <code>trigger()</code> 函数执行时（在收集了参数数据之后，但在验证这些数据 and 启动对 Web 服务的调用之前）进行广播。
<code>webServiceConnector.status</code>	在启动对 Web 服务的调用时广播，以通知用户操作状态。

webServiceConnector.multipleSimultaneousAllowed

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.multipleSimultaneousAllowed;
```

描述

属性；指明是否可以同时进行多个调用。如果为 `false`，则 `trigger()` 函数将在另一个调用已在执行时执行一个调用。将发出 `status` 事件，并且具有代码 `CallAlreadyInProgress`。如果值为 `true`，将发生调用。

当同时进行多个调用时，不保证这些调用的完成顺序与触发它们的顺序相同。此外，Flash Player 可能会对同时发生的网络操作的数目加以限制。该限制将因版本和平台而异。

范例

以下范例允许同时发生对 `myXmlUrl` 进行的多个调用：

```
myXmlUrl.multipleSimultaneousAllowed = true;
```

webServiceConnector.operation

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.operation;
```

描述

属性；显示在 WSDL 文件中的 SOAP 端口内的操作名称。

webServiceConnector.params

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.params;
```

描述

属性；指定在执行下一个 `trigger()` 操作时要发送到服务器的数据。数据类型由 Web 服务的 WSDL 描述确定。

调用 Web 服务方法时，`params` 属性的数据类型必须是一个动作脚本对象或数组，如下所示：

如果 Web 服务为文档格式，则 `params` 的数据类型是某种 XML 文档。

如果您在创作过程中使用属性检查器或“组件检查器”面板来设置 WSDLURL 和操作，则可以按照 Web 服务方法所要求的同一顺序提供作为参数数组的 `params`，如 `[1, "hello", 2432]`。

范例

以下范例为名为 `wsc` 的 Web 服务组件设置 `params` 属性：

```
wsc.params = [param_txt.text];
```

webServiceConnector.result

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("result", myListenerObject);
```


描述

事件；在远程过程调用操作成功完成后广播。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串“result”
- `target`：对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）

您可以使用 `results` 属性获取实际结果值。

范例

以下范例为 `result` 事件定义 `res` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var res = function (ev) {  
    trace(ev.target.results);  
};  
wsc.addEventListener("result", res);
```

`webServiceConnector.results`

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.results;
```

描述

属性；标识作为 `trigger()` 操作的结果从服务器接收的数据。每个 RPC 组件均定义获取此数据的方式以及有效的类型。这些数据在 RPC 操作成功完成后出现，由 `result` 事件通知。它在该组件卸载前或执行下一个 RPC 操作前可用。

返回的数据可能会非常大。您可以通过以下两种方式管理这些数据：

- 选择适当的影片剪辑、时间轴或屏幕作为 RPC 组件的父级。在父级退出后该组件的存储区将可用于垃圾回收。
- 在动作脚本中，您可以随时将空值分配给该属性。

`webServiceConnector.send`

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("send", myListenerObject);
```

描述

事件；在处理 `trigger()` 操作期间（在收集了参数数据后，但在验证这些数据 and 启动远程过程调用前）广播。此位置适合于存放将在调用前修改参数数据的代码。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 “send”
- `target`：对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）

您可以使用 `params` 属性获取或修改实际参数值。

范例

以下范例为 `send` 事件定义 `sendFunction` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var sendFunction = function (sendEnv) {
    sendEnv.target.params = [newParam_txt.text];
};
wsc.addEventListener("send", sendFunction);
```

webServiceConnector.status

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("status", myListenerObject);
```

描述

事件；在远程过程调用启动时广播，以通知用户操作状态。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 “status”
- `target`：对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）
- `code`：一个字符串，提供已发生的特定情况的名称。
- `data`：一个对象，其内容取决于代码。

下面是可用于状态事件的代码和关联数据：

代码	数据	描述
StatusChange	{callsInProgress:nnn}	只要 Web 服务调用开始或结束就发出该事件。 “nnn” 项提供当前正发生的调用的数目。

代码	数据	描述
CallAlreadyInProgress	无数据	如果：(a) 调用了 trigger 函数，且 (b) multipleSimultaneousAllowed 为 false，同时 (c) 调用已在执行中，则会发出此事件。在发生此事件后，则认为尝试的调用已完成，并且不会发生 “result” 或 “send” 事件。
InvalidParams	无数据	如果 trigger() 函数发现 “params” 属性不包含有效数据，则会发出此事件。如果 “suppressInvalidCalls” 属性为 true，则认为尝试的调用已完成，并且不会发生 “result” 或 “send” 事件。

以下是可能的 Web 服务值：

faultcode	faultstring	detail
Timeout	调用方法 xxx 时的超时	b
MustUnderstand	对于标题 xxx 无回调	b
Server.Connection	无法连接到端点：xxx	b
VersionMismatch	请求实现版本：xxx 响应实现版本 yyy	b
Client.Disconnected	无法加载 WSDL	无法加载 WSDL，如果当前在线，请验证 WSDL xxx 的 URI 和 / 或格式。
Server	出错的 WSDL 格式	定义必须是 WSDL 文档中的第一个元素
Server.NoServicesInWSDL	无法加载 WSDL	在 xxx 处找不到 WSDL 中的元素
WSDL.UnrecognizedNamespace	WSDL 分析器没有用于命名空间 xxxx 的注册文档	b
WSDL.UnrecognizedBindingName	WSDL 分析器在命名空间 yyy 中无法找到命名为 xxx 的绑定	b
WSDL.UnrecognizedPortTypeName	WSDL 分析器在命名空间 yyy 中无法找到命名为 xxx 的 portType	b
WSDL.UnrecognizedMessageName	WSDL 分析器在命名空间 yyy 中无法找到命名为 xxx 的消息	b
WSDL.BadElement	元素 xxx 不可解析	b
WSDL.BadType	类型 xxx 不可解析	b
Client.NoSuchMethod	无法在服务中找到 “xxx” 方法	b
yyy	yyy - 从服务器报告的错误，这取决于您与哪一台服务器通信	b
No.WSDLURL.Defined	WebServiceConnector 组件没有已定义的 WSDL URL	b

faultcode	faultstring	detail
Unknown.Call.Failure	WebService 调用因未知原因而失败	
Client.Disconnected	无法加载导入的架构	无法加载架构；如果当前在线，请在 (XXXX) 处验证架构的 URI 和 / 或格式

范例

以下范例为 status 事件定义 statusFunction 函数，并且将该函数分配给 addEventListener 事件处理函数：

```
var statusFunction = function (stat) {
    trace(stat.code);
    trace(stat.data.faultcode);
    trace(stat.data.faultstring);
};
wsc.addEventListener("status", statusFunction);
```

webServiceConnector.suppressInvalidCalls

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.suppressInvalidCalls;
```

描述

属性；指明在参数无效的情况下是否禁止调用。如果为 true，则 trigger() 函数在绑定参数未通过验证的情况下将不执行调用。将发出 “status” 事件，并且具有代码 InvalidParams。如果为 false，将进行该调用，并且根据需要使用无效数据。

webServiceConnector.timeout

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.timeout;
```

描述

属性；指定在结果未返回的情况下 Web 服务连接将在多少秒内失败。会发出 status 事件（继承自 RPC 组件），并且代码为 WebServiceFault，faultcode 为 Timeout。

webServiceConnector.trigger()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.trigger();
```

描述

方法；启动对 Web 服务的调用。每个 Web 服务都准确地定义了此调用所涉及的内容。如果操作成功，则操作的结果将显示在 Web 服务的 `results` 属性中。

`trigger()` 方法执行以下步骤：

- 1 如果任何数据绑定到 `params` 属性，则该方法执行所有绑定以确保提供最新数据。这样做还会导致发生数据验证。
- 2 如果数据无效且 `suppressInvalidCalls` 设置为 `true`，则操作将停止。
- 3 如果操作继续，则发出 `send` 事件。
- 4 使用指示的连接方法（例如 HTTP）启动实际远程调用。

webServiceConnector.WSDLURL

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.WSDLURL;
```

描述

属性；定义 Web 服务操作的 WSDL 文件的 URL。在创作期间设置此 URL 时，立即获取和分析该 WSDL 文件。产生的参数和结果显示在“组件检查器”面板的“架构”选项卡中。此外，服务描述显示在“Web 服务”面板中。

Window 组件

Window 组件在一个具有标题栏、边框和关闭按钮（可选）的窗口内显示影片剪辑的内容。

Window 组件可以是模式的，也可以是非模式的。模式窗口会防止鼠标和键盘输入转至该窗口之外的其他组件。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。

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，以使其可由屏幕阅读器访问。首先，您必须添加以下代码行来启用辅助功能：

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

不管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”。

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。

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。如果您不使用 [PopUpManager](#)，则窗口的内容会与文档中的其他组件一起参加焦点排序。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 248 页的“[FocusManager](#) 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 Window 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**。
这是将填写 Window 的内容。应将内容对齐 0,0，这是因为，内容位于 Window 的左上角。
- 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”。
链接标识符 **PasswordForm** 自动输入到“标识符”框中。
- 4 在类字段中输入 **mx.core.View** 并单击“确定”。

- 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 如果未选择“为动作脚本导出”，则选择它。
- 8 从“组件”面板中将按钮组件拖到舞台，并在属性检查器中为该组件指定实例名称 **button**。
- 9 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(){
    myWindow = mx.managers.PopUpManager.createPopUp(_root,
    mx.containers.Window, true, { title:"Change Password",
    contentPath:"PasswordForm"});
    myWindow.setSize(240,110);
}
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，该组件显示在按下按钮时 PasswordForm 影片剪辑的内容。要在按下“确定”或“取消”按钮时关闭 Window，您将需要编写另一个处理函数。

自定义 Window 组件

在创作过程中和在运行时，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，使用 `UIObject.setSize()`。

调整窗口大小不会更改关闭按钮或标题题注的大小。标题题注为左对齐，关闭栏为右对齐。

对 Window 组件使用样式

Window 组件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
<code>borderStyle</code>	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	字母
<code>borderColor</code>	a
<code>highlightColor</code>	b
<code>borderColor</code>	c
<code>shadowColor</code>	d
<code>borderCapColor</code>	e
<code>shadowCapColor</code>	f

RectBorder 样式	字母
shadowCapColor	g
borderCapColor	h

这些样式属性设置边框上的下列位置：



如果使用 `UIObject.createClassObject()` 或 `PopupMenu.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 `PopupMenu.createPopUp()`。

`Window` 组件使用下列外观属性：

属性	描述
<code>skinTitleBackground</code>	标题栏。默认值为 <code>TitleBackground</code> 。
<code>skinCloseUp</code>	关闭按钮。默认值为 <code>CloseButtonUp</code> 。
<code>skinCloseDown</code>	处于按下状态的关闭按钮。默认值为 <code>CloseButtonDown</code> 。
<code>skinCloseDisabled</code>	处于禁用状态的关闭按钮。默认值为 <code>CloseButtonDisabled</code> 。
<code>skinCloseOver</code>	处于悬停状态的关闭按钮。默认值为 <code>CloseButtonOver</code> 。

Window 类

继承 `UIObject > UIComponent > View > ScrollView > Window`

动作脚本类名称 `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 `PopupMenu.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 `PopupMenu.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](#)、[UIComponent](#) 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 <code>contentPath</code> 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](#)、[UIComponent](#) 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](#)、[UIComponent](#)、[View](#) 和 [ScrollView](#) 的所有事件。

Window.click

可用性

Flash Player 6 版本 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 `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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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) 应包含一个关闭按钮。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 版本 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 版本 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 版本 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。

示例

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 版本 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 `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`），而该事件由您创建的侦听器对象 (`listenerObject`) 上的函数（也称作处理函数）处理。您定义一个与侦听器对象上的事件同名的方法；当该事件被触发时，就会调用该方法。该事件被触发时，它会自动将一个事件对象 (`eventObject`) 传递到侦听器对象方法。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 510 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 版本 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 版本 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 25 页的“使用样式自定义组件的颜色和文本”](#)。

XMLConnector 组件（仅限 Flash Professional）

XMLConnector 组件是一种 Flash MX 2004 v2 组件，它的用途是使用 HTTP get 操作或 post 操作读或写 XML 文档。它充当其他组件和外部 XML 数据源之间的连接器。XMLConnector 使用 Flash MX Professional 2004 创作环境中的数据绑定功能或动作脚本代码与应用程序中的组件进行通信。XMLConnector 组件具有属性、方法和事件，但它没有运行时可视外观。

XMLConnector 组件和 WebServiceConnector 组件实现了 RPC（远程过程调用）组件 API、一组方法、属性，以及用于定义向外部数据源发送参数和从其接收结果的便捷方法的事件。

使用 XMLConnector 组件（仅限 Flash Professional）

XMLConnector 组件使应用程序可以访问通过 HTTP 返回或接收 XML 的任何外部数据源。连接外部 XML 数据源以及将该数据源的参数和结果用于您的应用程序的最简易方法是，指定一个架构 - XML 文档的结构，它标识可绑定到的文档中的数据元素。

架构显示在“组件检查器”面板中的“架构”选项卡中。架构标识可绑定到应用程序中的用户界面组件属性的 XML 文档中的字段。可以通过“组件检查器”面板手动创建架构，或利用创作环境自动创建一个架构。

注意：创作环境将接收一份您所连接到的外部 XML 文档的副本，以用作架构的模型。如果您熟悉 XML 脚本的撰写，则可以创建一个可用于生成架构的简单 XML 文件。

虽然 XMLConnector 组件具有属性和事件（像其他组件一样），但是它没有运行时可视外观。有关使用 XMLConnector 组件的详细信息，请参阅“使用 Flash”帮助中的“XMLConnector 组件（仅限 Flash Professional）”。

XMLConnector 组件的参数

以下列出了一些创作参数，您可以在“组件检查器”面板的“参数”选项卡中为每个 XMLConnector 组件实例设置这些参数：

`direction`（枚举）指示是发送、接收数据还是两者同时进行。

`ignoreWhite`（布尔型）设置为 `true` 时，在获取 XML 时将忽略空白。

`multipleSimultaneousAllowed`（布尔值类型）指明是否可以同时进行多个调用；默认值是 `false`。

`suppressInvalidCalls`（布尔型）指示在参数无效时是否禁止调用；默认值为 `false`。

`URL`（字符串类型）是在 HTTP 操作中使用的外部 XML 文档的 URL。

XMLConnector 类（仅限 Flash Professional）

继承 RPC > XMLConnector

动作脚本类名称 mx.data.components.XMLConnector

XMLConnector 类的属性摘要

属性	描述
<code>XMLConnector.direction</code>	指示是发送、接收数据还是两者同时进行。
<code>XMLConnector.multipleSimultaneousAllowed</code>	指明是否可以同时进行多个调用。
<code>XMLConnector.params</code>	指定在执行下一个 <code>trigger()</code> 操作时要发送到服务器的数据。
<code>XMLConnector.results</code>	标识作为 <code>trigger()</code> 操作的结果从服务器接收的数据。
<code>XMLConnector.suppressInvalidCalls</code>	指明在参数无效的情况下是否禁止调用。
<code>XMLConnector.URL</code>	组件在 HTTP 操作中使用的 URL。

XMLConnector 类的方法摘要

方法	描述
<code>XMLConnector.trigger()</code>	启动远程过程调用。

XMLConnector 类的事件摘要

事件	描述
<code>XMLConnector.result</code>	远程过程调用成功完成后广播。
<code>XMLConnector.send</code>	当 <code>trigger()</code> 函数执行时（在收集了参数数据之后，但在验证这些数据和启动远程调用之前）进行广播。
<code>XMLConnector.status</code>	在远程过程调用启动时广播，以通知用户操作状态。

XMLConnector.direction

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.direction;
```

描述

属性；指示是发送、接收数据还是两者同时进行。值如下所示：

- `send` `params` 属性的 XML 数据通过 HTTP POST 发送到 XML 文档的 URL。忽略任何返回的数据。对 `results` 属性不作任何设置，而且没有任何 `result` 事件。
注意： `params` 和 `results` 属性以及 `result` 事件继承自 RPC 组件 API。
- `receive` 没有 `params` 数据被发送到 URL。通过 HTTP GET 访问 XML 文档的 URL，并预期从该 URL 收到有效的 XML 数据。
- `send/receive Params` 数据被发送到 URL，并预期从该 URL 收到有效的 XML 数据。

范例

以下范例为文档 mysettings.xml 将 direction 设置为 receive：

```
myXMLConnector.direction = "receive";  
myXMLConnector.URL = "mysettings.xml";  
myXMLConnector.trigger();
```

XMLConnector.multipleSimultaneousAllowed

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.multipleSimultaneousAllowed;
```

描述

属性；指明是否可以同时进行多个调用。如果为 false，则 trigger() 函数将在另一个调用已在执行时执行一个调用。将发出 status 事件，并且具有代码 CallAlreadyInProgress。如果值为 true，将发生调用。

当同时进行多个调用时，不保证这些调用的完成顺序与触发它们的顺序相同。此外，Flash Player 可能会对同时发生的网络操作的数目加以限制。该限制将因版本和平台而异。

XMLConnector.params

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.params;
```

描述

属性；指定在执行下一个 trigger() 操作时要发送到服务器的数据。每个 RPC 组件均定义使用此数据的方式以及有效的类型。

范例

以下范例为 myXMLConnector 定义 name 和 city params：

```
myXMLConnector.params = new XML("<mydoc><name>Bob</name><city>Oakland</city></mydoc>");
```

XMLConnector.result

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("result", myListenerObject);
```

描述

事件；在远程过程调用操作成功完成后广播。

事件处理函数的参数是具有以下字段的对象：

- **type**：字符串 “result”
- **target**：对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）

您可以使用 `results` 属性获取实际结果值。

范例

以下范例为 `result` 事件定义 `res` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var res = function (ev) {  
    trace(ev.target.results);  
};  
xcon.addEventListener("result", res);
```

XMLConnector.results

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.results;
```

描述

属性；标识作为 `trigger()` 操作的结果从服务器接收的数据。每个 RPC 组件均定义获取此数据的方式以及有效的类型。这些数据在 RPC 操作成功完成后出现，由 `result` 事件通知。它在该组件卸载前或执行下一个 RPC 操作前可用。

返回的数据可能会非常大。您可以通过以下两种方式管理这些数据：

- 选择适当的影片剪辑、时间轴或屏幕作为 RPC 组件的父级。在父级退出后该组件的存储区将可用于垃圾回收。
- 在动作脚本中，您可以随时将空值分配给该属性。

范例

以下范例使用 `trace` 命令发送 `myXMLConnector` 的 `results` 属性：

```
trace(myXMLConnector.results);
```

XMLConnector.send

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("send", myListenerObject);
```

描述

事件；在处理 `trigger()` 操作期间（在收集了参数数据后，但在验证这些数据 and 启动远程过程调用前）广播。此位置适合于存放将在调用前修改参数数据的代码。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 “send”
- `target`：对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）

您可以使用 `params` 属性获取或修改实际参数值。

范例

以下范例为 `send` 事件定义 `sendFunction` 函数，并且将该函数分配给 `addEventListener` 事件处理函数：

```
var sendFunction = function (sendEnv) {  
    sendEnv.target.params = [newParam_txt.text];  
};  
xcon.addEventListener("send", sendFunction);
```

XMLConnector.status

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.addEventListener("status", myListenerObject);
```

描述

事件；在远程过程调用启动时广播，以通知用户操作状态。

事件处理函数的参数是具有以下字段的对象：

- `type`：字符串 “status”

- **target** : 对发出该事件的对象的引用（例如 `WebServiceConnector` 组件）
- **code** : 一个字符串，提供已发生的特定情况的名称。
- **data** : 一个对象，其内容取决于代码。

如果调用时发生问题，则 `status` 事件的代码字段设置为 `Fault`，如下所示：

代码	数据	描述
<code>Fault</code>	<code>{faultcode:code, faultstring:string, detail:detail, element:element, faultactor:actor}</code>	如果在处理调用的过程中发生其他问题，则会发出此事件。数据是 <code>SOAPFault</code> 对象。在发生此事件后，则认为尝试的调用已完成，并且不会发生“ <code>result</code> ”或“ <code>send</code> ”事件。

以下是可能伴随 `status` 事件发生的错误：

FaultCode	FaultString	注释
<code>XMLConnector.Not.XML</code>	<code>params</code> 不是 XML 对象	<code>params</code> 必须是动作脚本 XML 对象。
<code>XMLConnector.Parse.Error</code>	<code>params</code> 具有 XML 分析错误 <code>NN</code> 。	<code>params</code> XML 对象的“ <code>status</code> ”属性具有非零的值 <code>NN</code> 。请参阅“Flash 帮助”中有关 <code>XML.status</code> 的信息，以查看可能的错误 <code>NN</code> 。
<code>XMLConnector.No.Data.Received</code>	没有从服务器接收到任何数据	限制：由于不同的浏览器限制，此消息可能表示 (a) 服务器 URL 无效、无响应或返回 HTTP 错误代码；或 (b) 服务器请求成功但响应恰巧为 0 字节的数据。建议的解决方法是：设计应用程序时使服务器永不会返回 0 字节的数据。如果您获取“ <code>XMLConnector.No.Data.Received</code> ”，则将确实知道发生了服务器错误，并且可以相应地通知最终用户。
<code>XMLConnector.Results.Parse.Error</code>	收到的数据具有 XML 分析错误 <code>NN</code>	Flash Player 内置的 XML 分析器确定收到的 XML 无效。请参阅“Flash 帮助”中有关 <code>XML.status</code> 的信息，以查看可能的错误 <code>NN</code> 。
<code>XMLConnector.Params.Missing</code>	<code>direction</code> 为“ <code>send</code> ”或“ <code>send/receive</code> ”，但 <code>params</code> 为空。	<code>b</code>

范例

以下范例为 status 事件定义 statusFunction 函数，并且将该函数分配给 addEventListener 事件处理函数：

```
var statusFunction = function (stat) {  
    trace(stat.code);  
    trace(stat.data.faultcode);  
    trace(stat.data.faultstring);  
};  
xcon.addEventListener("status", statusFunction);
```

XMLConnector.suppressInvalidCalls

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.suppressInvalidCalls;
```

描述

属性；指明在参数无效的情况下是否禁止调用。如果为 true，则 trigger() 函数在绑定参数未通过验证的情况下将不执行调用。将发出“status”事件，并且具有代码 InvalidParams。如果为 false，将进行该调用，并且根据需要使用无效数据。

XMLConnector.trigger()

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.trigger();
```

描述

方法；启动远程过程调用。每一 RPC 组件均精确定义此操作涉及的内容。如果该操作成功，则操作结果将出现在 RPC 组件的 results 属性中。

trigger() 方法执行以下步骤：

- 1 如果任何数据绑定到 params 属性，则该方法执行所有绑定以确保提供最新数据。这样做还会导致发生数据验证。
- 2 如果数据无效且 suppressInvalidCalls 设置为 true，则操作将停止。
- 3 如果操作继续，则发出 send 事件。
- 4 使用指示的连接方法（例如 HTTP）启动实际远程调用。

XMLConnector.URL

可用性

Flash Player 6 版本 79。

版本

Flash MX Professional 2004。

用法

```
componentInstance.URL;
```

描述

属性；此组件在执行 HTTP 操作时所使用的 URL。此 URL 可以是绝对或相对的 URL。此 URL 受到所有标准的 Flash Player 安全性的保护。

XUpdateResolver 组件（仅限 Flash Professional）

解析程序组件是与 DataSet 组件（Macromedia Flash 数据结构中的数据管理功能的一部分）组合使用的。通过解析程序组件，您能够将对应应用程序内的数据进行的更改转换为适合于您所更新的外部数据源的格式。这些组件在运行时没有可视外观。

如果在应用程序中使用 DataSet 组件，则会生成一组优化的说明 (DeltaPacket)，它们描述了在运行时对数据所作的更改。这组说明会由解析程序组件转换为适当的格式（更新包）。在将更新发送到服务器后，服务器会发送一个响应（结果包），响应中包含由该更新操作导致的附加更新或错误。解析程序组件可以将此信息转换回 DeltaPacket，DeltaPacket 可应用于 DataSet 组件以使其与外部数据源保持同步。通过解析程序组件，您不必编写额外的动作脚本代码，就能够令应用程序和外部数据源保持同步。

XUpdate 是一种用于描述对 XML 文档所作的更改的标准，并受到各种 XML 数据库（如 Xindice 或 XHive）的支持。XUpdateResolver 组件将 DeltaPacket 翻译为 XUpdate 语句。外部数据源可以处理这些 XUpdate 语句。XML 文档包含在更新任何标准的 XUpdate 数据库时所需的信息和格式。

有关 XUpdate 语言规范的工作草案的信息，请参阅 www.xmldb.org/xupdate/xupdate-wd.html。可以使用并行解析程序组件 RDBMSResolver（请参阅第 398 页的“RDBMSResolver 组件（仅限 Flash Professional）”）将数据返回到基于 XML 的服务器。有关 DataSet 组件的详细信息，请参阅第 176 页的“DataSet 组件（仅限 Flash Professional）”。有关连接器的详细信息，请参阅第 549 页的“WebServiceConnector（仅限 Flash Professional）”和第 567 页的“XMLConnector 组件（仅限 Flash Professional）”。有关 Flash 数据结构的详细信息，请参阅“使用 Flash”帮助中的“解析程序组件（仅限 Flash Professional）”。

注意：也可以使用 XUpdateResolver 组件将数据更新发送到任何可以分析 XUpdate 语言的外部数据源；例如，ASP 页、Java servlet 或 ColdFusion 组件。

XUpdateResolver 组件的更新以 XUpdate 数据包的形式发送，并通过某个连接对象与数据库或服务器进行通信。XUpdate 包由一组优化的说明组成，它们描述了在 DataSet 组件上执行的插入、编辑和删除操作。解析程序组件从 DataSet 组件获取 DeltaPacket，将其自己的 XUpdate 包发送到连接器，从连接中接收服务器错误，并且将这些错误传送回 DataSet 组件 - 全都使用可绑定的属性。

使用 XUpdateResolver 组件（仅限 Flash Professional）

仅当 Flash 应用程序包含 DataSet 组件且必须将更新发送回数据源时，才使用此 XUpdateResolver 组件。此组件解析您想返回到 XML 格式的数据源的数据。

有关使用 XUpdateResolver 组件的详细信息，请参阅“使用 Flash”帮助中的“解析程序组件（仅限 Flash Professional）”。

XUpdateResolver 组件的参数

includeDeltaPacketInfo 布尔值；如果为 true，则会使 XUpdate 在 XUpdate 节点上的属性中包含 deltaPacket 的其他信息。此信息包含事务 ID 和操作 ID。

以下范例显示当此属性的布尔值设置为 false 时如何构建更新包：

```
<xupdate:modifications
  version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">

  <xupdate:remove select="/datapacket/row[@id='100']"/>

</xupdate:modifications>
```

以下范例显示当此属性的布尔值设置为 true 时如何构建更新包：

```
<xupdate:modifications
  version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate"
  transId="46386292065:Wed Jun 25 15:52:34 GMT-0700 2003">

  <xupdate:remove select="/datapacket/row[@id='100']" opId="0123456789"/>

</xupdate:modifications>
```

XUpdateResolver 组件的属性摘要

属性	描述
XUpdateResolver.deltaPacket	包含对 DataSet 组件所作更改的说明。
XUpdateResolver.includeDeltaPacketInfo	在 XUpdate 节点上的属性中包含 deltaPacket 的其他信息。
XUpdateResolver.updateResults	说明更新的结果。
XUpdateResolver.xupdatePacket	包含对 DataSet 组件所作更改的 XUpdate 翻译。

XUpdateResolver 组件的事件摘要

事件	描述
XUpdateResolver.beforeApplyUpdates	在创建 XML 包后并紧接着在发送该包之前由解析程序组件调用，以立即进行自定义的修改。
XUpdateResolver.reconcileResults	由解析程序组件调用以比较两个包。

XUpdateResolver.beforeApplyUpdates

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

resolveData.beforeApplyUpdates(eventObject)

参数

eventObject 解析程序事件对象；描述在将更新通过连接器发送到数据库之前对 XML 包进行的自定义。此解析程序事件对象应包含以下属性：

属性	描述
target	对象；触发此事件的解析程序。
type	字符串；事件的名称。
updatePacket	XML 对象；将要应用的 XML 对象。

返回

无。

描述

事件；在为新的 *deltaPacket* 创建 XML 包后，并紧接着在使用数据绑定发送该包之前由解析程序组件调用，以立即进行自定义的修改。在将更新的数据发送到连接器之前，可以使用此事件处理函数对 XML 进行自定义的修改。

范例

以下范例将用户身份验证数据添加到 XML 包：

```
on (beforeApplyUpdates) {  
    // 添加用户身份验证数据  
    var userInfo = new XML( ""+getUserId()+" "+getPassword()+" " );  
    xupdatePacket.firstChild.appendChild( userInfo );  
}
```

XUpdateResolver.deltaPacket

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

resolveData.deltaPacket

描述

属性；deltaPacket 类型的属性，它接收将要转换为 xupdatePacket 的 deltaPacket，并根据放置于 updateResults 属性中的任何服务器结果输出 deltaPacket。此事件处理函数提供一个方法，您可以通过此方法在将更新的数据发送到连接器前对 XML 进行自定义修改。

updateResults 属性中的消息视作错误。这意味着，具有消息的增量被再次添加到 deltaPacket 中，这样，下次将 deltaPacket 发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个 deltaPacket 前将这些消息提供给用户并进行修改。

XUpdateResolver.includeDeltaPacketInfo

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

resolveData.includeDeltaPacketInfo

描述

属性；Boolean 类型的属性，如果为 true，则在 XUpdate 节点上的属性中包含 deltaPacket 的其他信息。此信息将包含事务 ID 和操作 ID。

有关产生的 XML 的范例，请参阅第 575 页的“XUpdateResolver 组件的参数”。

XUpdateResolver.reconcileResults

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

resolveData.reconcileResults(eventObject)

参数

eventObject ResolverEvent 对象；描述用于比较两个 updatePacket 的事件对象。此解析程序事件对象应包含以下属性：

属性	描述
target	对象；触发此事件的解析程序。
type	字符串；事件的名称。

返回

无。

描述

事件；在从服务器收到结果后，并紧接着在通过数据绑定传送包含操作结果的 `deltaPacket` 之前，使用此回调来插入任何代码。这是放置对来自服务器的消息进行处理的代码的好地方。

范例

下面的范例会协调两个 `updatePacket`，并在成功后清除更新：

```
on (reconcileResults) {  
    // 检查结果  
    if( examine( updateResults ))  
        myDataSet.purgeUpdates();  
    else  
        displayErrors( results );  
}
```

XUpdateResolver.updateResults

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.updateResults
```

描述

属性；`deltaPacket` 类型的属性，包含使用连接器从服务器返回的更新的结果。使用此属性可将错误和更新的数据从服务器传送到 `DataSet` 组件；例如，当服务器为自动分配的字段分配新 ID 时。将此属性绑定到连接器的 `Results` 属性，以便它可以接收更新的结果并将结果传送回 `DataSet` 组件。

`updateResults` 属性中的消息视作错误。这意味着，具有消息的增量被再次添加到 `deltaPacket` 中，这样，下次将 `deltaPacket` 发送到服务器时可以重新发送它。您必须编写代码以处理具有消息的增量，以便在将消息添加到下一个 `deltaPacket` 前将这些消息提供给用户并进行修改。

XUpdateResolver.xupdatePacket

可用性

Flash Player 7。

版本

Flash MX Professional 2004。

用法

```
resolveData.xupdatePacket
```

描述

属性；`xm1` 类型的属性，包含对 `DataSet` 组件所作更改的 `XUpdate` 翻译。将此属性绑定到连接器组件的属性，后者将经过翻译的更改的更新包传送回 `DataSet` 组件。

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 (Macintosh) ； \$(LocalData)/Classes (Windows)
- .

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 和 \$(LocalData)/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:用户名:Library:Application Support:Macromedia:Flash MX 2004:en:configuration*。

UserConfig 和 LocalData 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 或 LocalData 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 599 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 580 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）。有关详细信息，请参阅第 601 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 581 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 580 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。参数是用户用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 582 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
<code>mx.core.UIObject</code>	<code>MovieClip</code>	<code>UIObject</code> 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。 <code>UIObject</code> 提供以下功能： <ul style="list-style-type: none">• 编辑样式• 事件处理• 按缩放比例调整大小
<code>mx.core.UIComponent</code>	<code>UIObject</code>	<code>UIComponent</code> 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。 <code>UIComponent</code> 提供以下功能： <ul style="list-style-type: none">• 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 39 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 子组件的构造函数：

```
function ScrollBar() {  
}
```

在此情况下，初始化新的滚动条时，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 588 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 590 页的“InspectableList”。
事件	定义组件事件。请参阅第 591 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 591 页的“Bindable”。
ChangeEvent	标识导致数据绑定发生的事件。请参阅第 592 页的“ChangeEvent”。
IconFile	图标文件名，该图标在 Flash “组件”面板中表示这一组件。请参阅第 600 页的“添加图标”。

下面各节更详细地说明组件元数据标记。

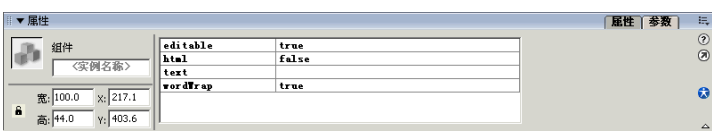
Inspectable

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： <ul style="list-style-type: none">• 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")]
writable	指示 Flash 允许属性只作为绑定的目标，如以下范例所示： [Bindable("writable")]
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="writable",type="DataProvider")]
```

在使用 ChangeEvent 元数据关键字时，必须使用 Bindable 关键字。有关详细信息，请参阅[第 592 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  
...  
function set selectedDate(value):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。

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 588 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（ <code>ListBox</code> 、 <code>ComboBox</code> 、 <code>Button</code> 、 <code>Text</code> ）也可能发出焦点，但所有组件均发出 <code>DOMFocusIn</code> 。
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>code</code> 属性包含被按下键的键控代码， <code>ascii</code> 属性包含它的 ASCII 代码。不要使用低级 <code>Key</code> 对象检查，因为 <code>Key</code> 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。

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 39 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 595 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 21 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 591 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 25 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 600 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。

A

- Accordion 组件 41
 - Accordion 类 46
 - 参数 42
 - 创建具有它的应用程序 43
 - 使用 42
 - 使用外观 45
 - 使用样式 45
 - 自定义 44
- addEventListener 596
- Alert 类
 - 方法 56
 - 属性 56
- Alert 组件 53
 - Alert 类 55
 - 参数 53
 - 创建具有它的应用程序 54
 - 事件 56
 - 使用 53
 - 使用外观 55
 - 使用样式 54
 - 自定义 54
- 安装
 - 检查 9
 - 指导 9
- 安装组件 8

B

- Binding 类 109
- Button 组件 61
 - Button 类 64
 - 参数 61, 436
 - 创建具有它的应用程序 62
 - 方法 64
 - 事件 65
 - 使用 61
 - 使用外观 63

- 使用样式 62
- 属性 65
- 自定义 62
- 包 12
- 本地类路径 580
- 编辑元件, 针对组件 581
- 编译剪辑 13
 - 处理 17
 - 在“库”面板中 16
- 标签 20
- 组件
 - 调整大小 20

C

- CellRenderer
 - 方法 72
 - 使用 71
 - 属性 72
- CellRenderer API 70
- CellRenderer 组件 70
- CheckBox 组件 76
 - CheckBox 类 79
 - 参数 77
 - 创建具有它的应用程序 77
 - 方法 79
 - 事件 79
 - 使用 77
 - 使用外观 78
 - 使用样式 78
 - 属性 79
- className 586
- clickHandler 23
- ComboBox 事件 88
- ComboBox 组件 83
 - ComboBox 类 87
 - 参数 84
 - 创建具有它的应用程序 85

- 方法 87
- 使用 84
- 使用外观 86
- 使用样式 85
- 属性 88
- ComponentMixins 类 122
- CSSStyleDeclaration 26, 27
- CustomFormatter 类 111
- CustomValidator 类 114
- 菜单激活器 359
- 参数
 - 查看 16
 - 定义 593
 - 可检查, 位于元数据语句中 588
 - 设置 16, 20
 - 添加到新组件 583
- 处理事件 22
- 初始化方法, 实现 593
- 创建组件
 - 编辑元件图层 582
 - 编写动作脚本 583
 - 编写动作脚本的过程 584
 - 编写构造函数 586
 - 处理事件 594
 - 创建 SWC 文件 599
 - 创建某个类的子类 586
 - 导出 598
 - 导入 SWC 文件 600
 - 定义版本号 586
 - 定义参数 593
 - 定义的 UIComponent 类 586
 - 定义的 UIObject 类 585
 - 辅助功能 598
 - 公共事件 594
 - 扩展组件类 583
 - 类文件的代码范例 584
 - 设置外观 597
 - 事件元数据 597
 - 实现核心方法 593
 - 使用 SWC 文件进行实时预览 600
 - 使用元数据语句 587
 - 添加参数 583
 - 添加事件 596
 - 添加提示文本 600
 - 添加图标 600
 - 选择父类 585
 - 选择类名称 586
 - 选择元件名称 586
 - 选择元件所有者 586

- 样式 598
- 组件元件 582

D

- DataGrid 类
 - 方法 141
 - 属性 141, 142
- DataGrid 组件 137
 - 参数 139
 - 创建具有它的应用程序 139
 - 进行交互 137
 - 类 141
 - 了解 138
 - 视图 138
 - 使用 138
 - 使用外观 140
 - 使用样式 140
 - 数据模型 138
 - 自定义 140
- DataGridColumn 类 159
 - 方法 160
- DataHolder 组件 165
- DataProvider API 167
 - 事件 168
 - 属性 168
- DataProvider 类
 - 方法 167
- DataSet 类 179
- DataSet 组件 176
- DataType 类 126
- DateChooser 类
 - 方法 220
 - 属性 220
- DateChooser 组件 218
 - DateChooser 类 220
 - 参数 218
 - 创建具有它的应用程序 218
 - 事件 220
 - 使用 218
 - 使用外观 219
 - 使用样式 219
 - 自定义 219
- DateField 类, 方法 231
- DateField 组件 228
 - DateField 类 231
 - 参数 229
 - 创建具有它的应用程序 229
 - 事件 232
 - 使用 228
 - 使用外观 230

- 使用样式 230
- 属性 231
- defaultPushButton 23
- DeltaPacket
 - 关于 574
 - 与组件一起使用 574
- DepthManager 24
 - 方法 244
 - 类 243
- detail 537, 547
- 大小方法, 实现 594
- 代码提示, 触发 20
- 导出自定义组件 598
- 导入类 585
- 第 1 版的组件结构, 与第 2 版的不同之处 579
- 第 1 版组件 24
 - 升级 24
- 第 2 版组件
 - 好处和说明 11
 - 和 Flash Player 12
- 第 2 版组件结构
 - 使用 SWC 文件进行实时预览 600
 - 与第 1 版的不同之处 579
- 动作脚本
 - 为新组件编写 583
 - 为新组件编写的工作流程 584

E

- EndPoint 类 117

F

- faultactor 537, 547
- faultcode 537, 547
- faultstring 537, 547
- FLA 文件资源, 为组件文件存储 579
- Flash MX 2004, 可用组件 8
- Flash MX Professional 2004, 可用组件 8
- Flash Player
 - 和组件 12
 - 支持 24
- Flash Professional
 - RDBMSResolver 组件 398
 - XUpdateResolver 组件 574
 - 组件类型 398, 574
- FocusManager 23
- FocusManager 类 248
- FocusManager 组件
 - FocusManager 类 250
 - 参数 249

- 创建具有它的应用程序 249
- 使用 249
- 自定义 250
- Form 类 255
- 范例主题 31
- 方法
 - 初始化, 实现 593
 - 大小, 实现 594
 - 定义 getter 和 setter 587
 - 实现 593
- 父类, 为新组件选择 585
- 辅助功能
 - 创作 13
 - 和组件 13
 - 用于自定义组件 598

G

- getter, 为属性定义 587
- 构造函数, 为新组件编写 586
- 光晕主题 31

H

- handleEvent 方法 22

J

- 继承
 - 在第 2 版组件中 12
- 焦点 23
- 焦点导航, 创建 23
- 接口
 - TransferObject 479
 - TreeDataProvider 497

K

- “库” 面板 16
- 扩展类 586

L

- Label 类 262
- Label 组件 260
 - Label 类 262
 - 参数 261
 - 创建具有它的应用程序 261
 - 方法 262
 - 事件 263
 - 使用 261
 - 使用样式 261
 - 属性 262
 - 自定义 261

- List 类 267
 - 构成 71
 - 滚动 71
- List 组件 264
 - List 类 267
 - 参数 265
 - 创建具有它的应用程序 265
 - 方法 268
 - 了解 70
 - 事件 269
 - 使用 265
 - 使用样式 266
 - 属性 269
 - 自定义 266
- Loader 组件 289
 - Loader 类 291
 - 参数 290
 - 创建具有它的应用程序 290
 - 方法 292
 - 事件 292
 - 使用 290
 - 属性 292
 - 自定义 290
- Log 类 528
- 类
 - Accordion 46
 - Alert 55
 - Binding 109
 - Button 64
 - CheckBox 79
 - ComboBox 87
 - ComponentMixins 122
 - CustomFormatter 111
 - CustomValidator 114
 - DataGrid 141
 - DataGridColumn 159
 - DataSet 179
 - DataType 126
 - DateChooser 220
 - DateField 231
 - EndPoint 117
 - FocusManager 250
 - Label 262
 - List 267
 - Loader 291
 - Log 528
 - Menu 342
 - MenuBar 362
 - NumericStepper 371
 - PendingCall 531

- ProgressBar 381
- RadioButton 392
- ScrollPane 424
- SOAPCall 539
- TextArea 461
- TextInput 472
- UIComponent 586
- UIObject 585
- WebService 541
- 创建子类 586
- 导入 585
- 和组件继承性 12
- 扩展 583, 586
- 媒体 309
- 名称, 用于自定义组件 586
- 数字步进器 486
- 文件, 为组件存储 580
- 选择父类 585
- 样式表 25
- 类别
 - UI 控件 39
 - 管理器 41
 - 媒体 41
 - 屏幕 41
 - 数据 40
- 类路径
 - 本地 580
 - 更改 580
 - 和 UserConfig 目录 580
 - 了解 580
 - 全局 580

M

- Macromedia DevNet 10
- Macromedia Flash 技术支持中心 10
- Media 类 309
 - 方法 309
 - 事件 311
 - 属性 310
- MediaController 组件
 - 参数 307
 - 了解 301
 - 使用 303
- MediaDisplay 组件
 - 参数 306
 - 了解 301
 - 使用 303
- MediaPlayback 组件
 - 参数 307
 - 了解 302

- 使用 302
- Menu 类 342
 - 方法 343
 - 属性 343
- Menu 组件 334
 - 菜单项类型 337
 - 参数 339
 - 初始化对象属性 339
 - 创建具有它的应用程序 339
 - 关于 XML 属性 336
 - 进行交互 334
 - 类 342
 - 使用 335
 - 使用外观 342
 - 使用样式 342
 - 添加分层菜单 336
 - 向动作脚本揭示项目 338
 - 自定义 342
- MenuBar 类
 - 方法 362
 - 属性 362
- MenuBar 组件 359
 - 参数 360
 - 创建具有它的应用程序 360
 - 进行交互 359
 - 类 362
 - 使用 360
 - 使用外观 362
 - 使用样式 361
 - 自定义 361
- 媒体
 - 组件
 - 使用行为 305
- 媒体组件 41, 299
 - 参数 306
 - 创建应用程序 308
 - 进行交互 299
 - 了解 300
 - 使用 302
 - 使用外观 308
 - 使用样式 308
 - 使用“组件检查器” 304
 - 行为, 将 MediaController 与 MediaDisplay 关联 305
 - 行为, 将 MediaDisplay 与 MediaController 关联 305
 - 行为, 使用标签帧线索点导航 305
 - 行为, 使用幻灯片线索点导航 306
 - 自定义 308

- 名称
 - 类 586
 - 元件, 用于自定义组件 586

N

- NumericStepper 类
 - 方法 46, 371
 - 属性 47, 231, 371
- NumericStepper 组件 218, 368
 - NumericStepper 类 371
 - 参数 218, 369
 - 创建具有它的应用程序 218, 369
 - 事件 46, 56, 231, 372
 - 使用 218, 369
 - 使用外观 219, 370
 - 使用样式 45, 219, 370
 - 自定义 219, 370

O

- on() 21
- onFault 537, 538, 547

P

- PendingCall 类 531
- PopUpManager 类 376
- PopUpManager 类, 方法 376
- ProgressBar 组件 377
 - ProgressBar 类 381
 - 参数 378
 - 创建具有它的应用程序 378
 - 方法 381
 - 事件 381
 - 使用 378
 - 使用外观 380
 - 使用样式 380
 - 属性 381
 - 自定义 379
- 屏幕 API 41
- 屏幕阅读器, 辅助功能 13

Q

- 全局
 - 类路径 580

R

- RadioButton 组件 389
 - RadioButton 类 392
 - 参数 390
 - 创建具有它的应用程序 391

- 方法 393
- 事件 393
- 使用 390
- 使用外观 392
- 使用样式 391
- 属性 393
 - 自定义 391
- RDBMSResolver 组件 398
 - 参数 399
 - 方法 400
 - 事件 400
 - 使用 398
 - 属性 400
- RPC 组件 API
 - 和 XMLConnector 567
 - 用于 WebServiceConnector 549

S

- ScrollPane 组件 422
 - ScrollPane 类 424
 - 参数 423
 - 创建具有它的应用程序 423
 - 方法 425
 - 事件 425
 - 使用 423
 - 使用外观 424
 - 使用样式 424
 - 属性 425
 - 自定义 424
- separator 337
- setSize() 20
- setter, 为属性定义 587
- SOAPCall 类 539
- SOAPFault 537, 547
- StyleManager 类 457
- StyleManager 类, 方法 457
- SWC 文件 13
 - 处理 17
 - 创建 599
 - 导入 600
 - 和编译剪辑 13
 - 解释的文件格式 598
- 设置外观 33
 - 用于自定义组件 597
- 深度, 管理 24
- 事件 21
 - 处理 594
 - 对象 21
 - 公共事件 594
 - 广播 21

- 添加 596
- 元数据 591, 597
- 侦听器 21
- 实例
 - 设置样式 26
- 实例, 设置样式 26
- 实例样式 25
- 实时预览 17
 - 用于自定义组件 600
- 视图
 - Menu 组件 335
- 使用动作脚本添加组件 19
- 数据绑定类 108
- 数据模型
 - Menu 组件 335
- 数据组件 40
- 属性检查器 16
- 数字步进器类
 - 方法 355
 - 属性 343
- 数字步进器组件
 - 创建具有它的应用程序 483
 - 事件 141, 160, 343, 355, 362

T

- Tab 键顺序, 组件的 248
- tabIndex 23
- TextArea 组件 459
 - TextArea 类 461
 - 参数 459
 - 创建具有它的应用程序 460
 - 事件 462
 - 使用外观 461
 - 使用样式 460
 - 属性 462
 - 自定义 460
- TextInput 组件 469
 - TextInput 类 472
 - 参数 470
 - 创建具有它的应用程序 470
 - 方法 472
 - 事件 473
 - 使用 470
 - 使用样式 471
 - 属性 472
 - 自定义 471
- TransferObject 组件 479
 - 方法 479
- Tree 类
 - 属性 486

- Tree 组件 481
 - XML 格式设置 482
 - 参数 483
 - 创建具有它的应用程序 483
 - 类 486
 - 使用 482
 - 使用外观 486
 - 使用样式 485
 - 自定义 485
- TreeDataProvider 接口
 - 方法 498
 - 属性 498
- 提示文本, 用于自定义组件 600
- 调整组件大小 20

U

- UIComponent 类
 - 定义的 586
 - 和组件继承性 12
- UIObject 类, 定义 585

W

- Web 服务, WSDL 文件 549
- Web 服务类
 - 类
 - Web 服务 528
- WebService 类 541
- WebServiceConnector
 - multipleSimultaneousAllowed 参数 550
 - suppressInvalidCalls 参数 550
 - WSDLURL 参数 550
 - 参数 550
 - 操作参数 550
 - 方法摘要 551
 - 事件摘要 551
 - 使用 550
 - 属性摘要 550
- WebServiceConnector 组件 549
- WSDL 文件
 - 获取更新 549
 - 用于 Web 服务 549
- 外观 33
 - 编辑 33
 - 应用 34
 - 应用到子组件 29
- 外观的链接标识符 33
- 外观属性
 - 设置 33
 - 在原型中更改 37

- 文档
 - 概述 9
 - 术语指南 9
- 文档中的术语 9

X

- XML
 - 设置 Tree 组件的格式 482
- XML 属性 336
- XMLConnector
 - 参数 567
 - 方法摘要 568
 - 和架构 567
 - 类 567
 - 事件摘要 568
 - 属性摘要 568
- XMLConnector 组件 567
- XUpdate 574
- XUpdateResolver 组件 574
 - 参数 575
 - 事件 575
 - 使用 575
 - 属性 575
- 系统要求 7
- 行为
 - 视频, 控制视频播放 305

Y

- 颜色
 - 设置样式属性 29
- 样式 25
 - 继承, 跟踪 457
 - 确定优先顺序 28
 - 设置 25, 29
 - 设置全局 26
 - 设置自定义 27
 - 用于自定义组件 598
 - 在实例上设置 26
 - 支持的 30
- 样式声明
 - 创建自定义 27
 - 默认类 28
 - 全局 26
 - 设置类 28
- 样式属性
 - color 29
 - 获取 29
 - 设置 29
- 印刷惯例, 在组件文档中 9

- 用户界面 (UI) 控件 39
- 用于开发组件的代码范例 581
- 语法, 用于元数据语句 587
- 预览组件 17
- 远程过程调用 (RPC), 用于 WebServiceConnector 549
- 远程过程调用组件 407
- 元件
 - 名称, 用于自定义组件 586
 - 所有者, 用于自定义组件 586
- 元件编辑, 针对组件 581
- 元件图层, 为新组件编辑 582
- 元数据 587
 - 标记 588
 - 解释 587
 - 可检查属性 588
 - 事件 591, 597
 - 语法 587
- 元数据的标记 588
- 元数据语句中的可检查属性 588
- 元素 537, 547
- 原型 37

Z

- 侦听器 21
 - 函数 22
- 主题 31
 - 创建 32
 - 应用 32
- 自定义
 - color 25
 - 文本 25
- 自定义样式表 25
- 自定义组件的图标 600
- 子类, 用于替换外观 37
- 资源, 其他 10
- 子组件, 应用外观 29
- 组件
 - DateField 228
 - DepthManager 243
 - Flash Player 支持 12
 - FocusManager 类 248
 - 安装 15
 - 动态添加 19
 - 继承 12
 - 结构 12
 - 类别 39
 - 类别, 已介绍 12
 - 媒体 299
 - 删除 20

- 调整大小 20
- 添加到 Flash 文档 18
 - 在 Flash MX 2004 中可用 8
 - 在 Flash MX Professional 2004 中可用 8
- 组件的版本号 586
- 组件类文件代码范例 584
- 组件类型
 - Accordion 41
 - Alert 53
 - Button 61
 - CellRenderer 70
 - CheckBox 76
 - ComboBox 83
 - DataGrid 137
 - DataHolder 165
 - DataProvider 167
 - DataSet 176
 - DateChooser 218
 - DateField 228
 - Flash Professional 398, 574
 - Label 260
 - List 264
 - Loader 289
 - Menu 334
 - MenuBar 359
 - NumericStepper 53, 218, 228, 368
 - PopUpManager 类 376
 - ProgressBar 377
 - RadioButton 389
 - RDBMSResolver 398
 - ScrollPane 422
 - StyleManager 类 457
 - TextArea 459
 - TextInput 469
 - TransferObject 479
 - Tree 481
 - UI 控件 39
 - WebServiceConnector 549
 - XMLConnector 567
 - XUpdateResolver 574
 - 管理器 41
 - 媒体 41
 - 数据 40
 - 数字步进器 334
 - 远程过程调用 407
- 组件文件, 存储 581
- 组件元件, 创建 582
- 组件源文件 581
- “组件检查器” 面板 16
- “组件” 面板 15