

MX



macromedia®

FLASH™MX
2004

动作脚本参考指南

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，并且可能已经在美国或其它管辖区甚至世界范围内注册。本出版物中提到的其它产品名称、徽标、图案、标题、文字或短语可能是 Macromedia, Inc. 或其它实体的商标、服务标志或商品名称，并且可能已经在特定的管辖区甚至世界范围内注册。

第三方信息

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

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



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

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

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。您还可能有其它权利，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 ZFL70M400X

鸣谢

主编：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

责任编辑：Rosana Francescato

编辑：Linda Adler、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：动作脚本入门	9
面向用户	9
系统要求	9
使用本文档	9
排版印刷约定	10
在本文档中使用的术语	10
附加资源	10

第 I 部分：欢迎使用动作脚本

第 1 章：Flash MX 2004 动作脚本中的新增功能	13
新增的和经改进的语言元素	13
新安全模型和旧 SWF 文件	14
将现有脚本移植到 Flash Player 7	14
动作脚本编辑器更改	19
调试方面的更改	20
新的面向对象编程模型	20
第 2 章：动作脚本基础	23
动作脚本和 JavaScript 之间的差异	23
动作脚本的 Unicode 支持	23
术语	24
语法	26
关于数据类型	30
为元素指定数据类型	33
关于变量	36
使用运算符处理表达式中的值	39
指定对象的路径	44
使用内置函数	45
创建函数	45
第 3 章：编写和调试脚本	49
运行动作脚本时进行控制	49
使用“动作”面板和“脚本”窗口	51

使用动作脚本编辑器	54
调试脚本	61
使用“输出”面板	68
为测试影片更新 Flash Player	71

第 II 部分：处理事件和创建交互操作

第 4 章：处理事件	75
使用事件处理函数方法	75
使用事件侦听器	76
使用按钮和影片剪辑事件处理函数	77
创建具有按钮状态的影片剪辑	78
事件处理函数的范围	79
“this”关键字的范围	80
第 5 章：用动作脚本创建交互操作	81
关于事件和交互	81
控制 SWF 文件回放	81
创造交互性和视觉效果	83
分析范例脚本	96

第 III 部分：使用对象和类

第 6 章：使用内置类	101
关于类和实例	101
内置类概述	102
第 7 章：使用影片剪辑	107
关于通过动作脚本控制影片剪辑	107
在单个影片剪辑上调用多个方法	108
加载和卸载其它 SWF 文件	108
指定加载的 SWF 文件的根时间轴	109
将 JPEG 文件加载到影片剪辑中	110
更改影片剪辑的位置和外观	110
拖动影片剪辑	111
在运行时创建影片剪辑	111
将参数添加到动态创建的影片剪辑	113
管理影片剪辑的深度	114
用动作脚本绘制形状	115
将影片剪辑用作遮罩	115
处理影片剪辑事件	116
将类分配给影片剪辑元件	116
初始化类属性	117

第 8 章：使用文本	119
使用 TextField 类	119
在运行时创建文本字段	120
使用 TextFormat 类	121
使用层叠样式表对文本进行格式设置	122
使用 HTML 格式的文本	130
创建滚动文本	135
第 9 章：使用动作脚本 2.0 创建类	137
面向对象编程的原则	137
使用类：一个简单的示例	138
创建和使用类	141
实例成员和类成员	145
创建和使用接口	147
理解类路径	148
使用包	150
导入类	151
隐式获取 / 设置方法	151
创建动态类	152
类的编译和导出方式	153
 第 IV 部分：使用外部数据和媒体	
 第 10 章：使用外部数据	157
向远程源发送变量和从远程源加载变量	157
向 Flash Player 发送消息以及从 Flash Player 接收消息	164
Flash Player 安全功能	166
 第 11 章：使用外部媒体	171
加载外部媒体的概述	171
加载外部 SWF 和 JPEG 文件	172
加载外部 MP3 文件	173
读取 MP3 文件中的 ID3 标签	174
动态回放外部 FLV 文件	174
预加载外部媒体	175
 第 V 部分：脚本参考	
 第 12 章：动作脚本字典	181
大多数动作脚本元素的范例条目	181
类的范例条目	182
字典内容	182
Accessibility 类	240
Arguments 类	246

Array 类	247
Boolean 类	262
Button 类	265
Camera 类	282
Color 类	300
ContextMenu 类	304
ContextMenuItems 类	310
CustomActions 类	315
Date 类	317
Error 类	345
Function 类	356
Key 类	373
LoadVars 类	390
LocalConnection 类	398
Math 类	408
Microphone 类	424
Mouse 类	438
MovieClip 类	444
MovieClipLoader 类	500
NetConnection 类	511
NetStream 类	513
Number 类	525
Object 类	530
PrintJob 类	550
Selection 类	564
SharedObject 类	573
Sound 类	580
Stage 类	597
String 类	606
System 类	617
System.capabilities 对象	622
System.security 对象	630
TextField 类	634
TextField.StyleSheet 类	658
TextFormat 类	672
TextSnapshot 对象	680
Video 类	700
XML 类	709
XMLNode 类	729
XMLSocket 类	730
 附录 A: 错误消息	 739
 附录 B: 运算符的优先级和结合律	 743
 附录 C: 键盘键和键控代码值	 745
字母 A 到 Z 和标准数字 0 到 9	745
数字键盘上的键	746

功能键	747
其它键	747
附录 D: 为早期的 Flash Player 版本编写脚本	749
关于以早期的 Flash Player 版本作为目标播放器	749
使用 Flash MX 2004 为 Flash Player 4 创建内容	749
附录 E: 使用动作脚本 1 进行面向对象的编程	753
关于动作脚本 1	753
索引	761

引言

动作脚本入门

Macromedia Flash MX 2004 和 Flash MX Professional 2004 是具有专业水准的标准创作工具，使您的产品可以给用户带来更美好、印象更深刻的 Web 体验。动作脚本是您要在 Flash 内开发应用程序时所使用的语言。您不必使用动作脚本就可以使用 Flash，但是，如果您要提供与用户的交互性、使用除内置于 Flash 中的对象之外的其它对象（例如按钮和影片剪辑）或者令您的 SWF 文件更适合于用户使用，可能还是要使用动作脚本。

面向用户

本手册假定您已安装了 Flash MX 2004 或 Flash MX Professional 2004 并了解其用法。您应该知道如果将对象放置于舞台上以及如何在 Flash 创作环境中处理对象。如果您在以前曾编写过程序，将会感到动作脚本对您并不陌生。但即使您以前没有编写过程序，动作脚本也不难学。您可以很容易地从一些非常简单的命令入手，并逐步掌握更复杂的功能。

系统要求

动作脚本没有除 Flash MX 2004 或 Flash MX Professional 2004 之外的任何额外系统要求。不过，本文档假设您使用默认的发布设置来用于 Flash 文件：Flash Player 7 和动作脚本 2.0。如果您更改了这些设置中的任何一个，则本文档中所示的说明和代码示例可能无法正确使用。

使用本文档

本文档提供动作脚本语法的概述、有关在使用不同类型的对象时如何使用动作脚本的信息以及有关每一语言元素的语法和用法的详细信息。您可以从学习本文档的其余部分中使用的术语和基本概念开始（请参见第 23 页的第 2 章“动作脚本基础”）。接下来，学习编写和调试 Flash 脚本的机制（请参见第 49 页的第 3 章“编写和调试脚本”）。

在编写您自己的脚本前，应该首先完成“使用动作脚本编写脚本”和“使用条件逻辑和发送数据创建表单”课程，这两课将教授您如何亲手编写动作脚本。若要找到这些课程，请选择“帮助”>“如何”>“快速任务”。

在您理解了这些基础知识后，就可以利用本文档其余部分中提供的信息，应用于您要获得的特定效果。例如，如果您要学习如何编写在用户单击鼠标时执行某一操作的脚本，请参见第 75 页的第 4 章“处理事件”。

当您查找与要使用的具体命令有关的信息时，可以在第 181 页的第 12 章“动作脚本字典”中查找其条目；在该部分中按字母顺序列出了每一语言元素。

排版印刷约定

在本手册中采用以下排版印刷约定：

- 代码字体指示动作脚本代码。
- *斜体代码字体* 指示元素，例如动作脚本参数或对象名称，您在编写脚本时可以用自己的文本来代替这一字体的内容。

在本文档中使用的术语

在本手册中使用以下术语：

- 您 指的是编写脚本或应用程序的开发人员。
- 用户 指的是将运行您的脚本和应用程序的人士。
- 编译时 是您发布、导出、测试或调试您的文档的时间。
- 运行时 是在 Flash Player 中运行您的脚本的时间。

动作脚本术语（例如方法 和对象）在第 23 页的第 2 章“动作脚本基础”中定义。

附加资源

与 Flash 和相关产品有关的特定文档分别提供。

- 有关在 Flash 创作环境中工作的信息，请参见“使用 Flash 帮助”。有关使用组件的信息，请参见“使用组件帮助”。
- 有关使用 Flash Communication Server 创建通讯应用程序的信息，请参见开发通讯应用程序 和管理 Flash Communication Server。
- 有关使用 Flash 应用程序访问 Web 服务的信息，请参见使用 Flash Remoting。

Macromedia DevNet Web 站点 (www.macromedia.com/go/developer_cn) 定期用有关 Flash 的最新信息进行更新，同时还提供来自专家用户的建议、高级主题、示例、提示和其它更新。请经常查看这一 Web 站点，了解有关 Flash 以及如何充分利用这一程序的最新消息。

Macromedia Flash 支持中心 (www.macromedia.com/go/flash_support_cn) 提供技术说明、文档更新以及指向 Flash 社区中的其它资源的链接。

第 I 部分

欢迎使用动作脚本

这一部分包含有关动作脚本语言的基本信息。

第 1 章包含有关动作脚本和 Flash Player 7 中新增的或经改进的功能的信息。如果您以前使用过动作脚本，应仔细阅读这些信息。

如果您不熟悉动作脚本，请阅读第 2 章和第 3 章，为了解动作脚本的术语和语法以及学习如何编写和调试您自己的脚本打下良好的基础。

第 1 章：Flash MX 2004 动作脚本中的新增功能.....	13
第 2 章：动作脚本基础	23
第 3 章：编写和调试脚本	49

第 1 章

Flash MX 2004 动作脚本中的新增功能

Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 提供了几项改进，使您能够更方便地使用动作脚本语言编写更为坚实的脚本。本章将讨论这些新功能，这些功能包括新的语言元素、改进的编辑和调试工具（请参见第 19 页的“[动作脚本编辑器更改](#)”和第 20 页的“[调试方面的更改](#)”）和更为面向对象的编程模型（请参见第 20 页的“[新的面向对象编程模型](#)”）。

如果您计划将任何现有的 Flash MX 或更早版本的文件发布到 Flash Player 7，则还应仔细阅读本章中包含的增补部分（请参见第 14 页的“[将现有脚本移植到 Flash Player 7](#)”）。

新增的和经改进的语言元素

本节介绍 Flash MX 2004 中新增的或经改进的动作脚本语言元素。若要在脚本中使用这些元素中的任何一种，都必须在发布文档时指定以 Flash Player 7（默认设置）为目标播放器。

- `Array.sort()` 和 `Array.sortOn()` 方法可用于添加参数以指定附加的排序选项，例如升序和降序排序，排序时是否区分大小写，等等。
- `Button.menu`、`MovieClip.menu` 和 `TextField.menu` 属性与新增的 `ContextMenu` 和 `ContextMenuItems` 类一起用于将上下文菜单项与 `Button`、`MovieClip` 或 `TextField` 对象相关联。
- `ContextMenu` 类和 `ContextMenuItems` 类用于自定义上下文菜单，当用户在 Flash Player 中右键单击 (Microsoft Windows) 或按住 `Control` 键并单击 (Macintosh) 时，将显示此菜单。
- `Error` 类以及 `throw` 与 `try..catch..finally` 命令可用于实现更可靠的异常处理。
- `LoadVars.setRequestHeader()` 和 `XML.setRequestHeader()` 方法用于添加或更改用 `POST` 动作发送的 `HTTP` 请求标头（例如，`Content-Type` 或 `SOAPAction`）。
- `MMExecute()` 函数可用于从动作脚本中发出 Flash JavaScript API 命令。
- （仅限于 Windows）当用户使用鼠标滚轮滚动时，将生成 `Mouse.onMouseWheel` 事件侦听器。
- `MovieClip.getNextHighestDepth()` 方法用于在运行时创建 `MovieClip` 实例，并且确保它们的对象在父影片剪辑的 `z` 顺序空间中呈现在其它对象的前面。
`MovieClip.getInstanceAtDepth()` 方法用于以深度作为搜索索引来访问动态创建的 `MovieClip` 实例。
- `MovieClip.getSWFVersion()` 方法用于确定加载的 `SWF` 文件支持哪个版本的 Flash Player。

- `MovieClip.getTextSnapshot()` 方法和 [TextSnapshot](#) 对象用于处理影片剪辑中静态文本字段内的文本。
- `MovieClip._lockroot` 属性用于指定影片剪辑将作为加载到其中的任何影片剪辑的 `_root`，或者指定影片剪辑中 `_root` 的含义在该影片剪辑加载到另一个影片剪辑时将保持不变。
- `MovieClipLoader` 类用于在将文件加载到影片剪辑的过程中监视文件的进度。
- `NetConnection` 类和 `NetStream` 类用于对本地视频文件（FLV 文件）进行流式处理。
- `PrintJob` 类使您（和用户）能够更多地控制 Flash Player 中的打印。
- `Sound.onID3` 事件处理函数可用于访问与包含 MP3 文件的 Sound 对象关联的 ID3 数据。
- `Sound.ID3` 属性可用于访问 MP3 文件中的元数据。
- `System` 类有了新的对象和方法，而且 [System.capabilities](#) 对象有了几个新属性。
- `TextField.condenseWhite` 属性可用于从浏览器中呈现的 HTML 文本字段中删除多余的空白。
- `TextField.mouseWheelEnabled` 属性用于指定当鼠标指针位于文本字段上并且用户滚动鼠标滚轮时，文本字段的内容是否应滚动。
- `TextField.StyleSheet` 类可用于创建包含文本格式设置规则（例如，字体大小、颜色和其它格式样式）的样式表对象。
- `TextField.styleSheet` 属性可用于将样式表对象附加到文本字段上。
- `TextFormat.getTextExtent()` 方法接受一个新参数，它所返回的对象包含一个新成员。
- `XML.setRequestHeader()` 方法可用于添加或更改用 POST 动作发送的 HTTP 请求标头（例如，Content-Type 或 SOAPAction）。

新安全模型和旧 SWF 文件

Flash Player 用于确定两个域是否相同的规则在 Flash Player 7 中有所变化。此外，用于确定 HTTP 域提供的 SWF 文件能否以及如何访问 HTTPS 域提供的 SWF 文件或加载 HTTPS 域提供的数据的规则也有所变化。在大多数情况下，这些改动不会对您造成影响，除非要将现有的 SWF 文件移植到 Flash Player 7。

但是，如果 SWF 文件是以 Flash Player 6 或更低版本为目标播放器发布的，而且这些 SWF 文件从存储在服务器上的文件加载数据，则当调用 SWF 在 Flash Player 7 中播放时，用户可能会看到一个以前从未出现过的对话框，询问是否允许访问。通过在存储数据的站点上实施一个策略文件，可以让该对话框不出现。有关此对话框的更多信息，请参见第 169 页的“关于与以前的 Flash Player 安全模型的兼容性”。

如果您使用的是运行时共享库，可能还需要实现一个策略文件。如果正加载或已加载的 SWF 文件是以 Flash Player 7 为目标播放器发布的，而且这些正加载和已加载的 SWF 文件不是从完全相同的域提供的，则使用一个策略文件来允许访问。有关策略文件的更多信息，请参见第 168 页的“关于允许跨域数据加载”。

将现有脚本移植到 Flash Player 7

与所有新发布的版本相同，Flash Player 7 比该播放器以前的版本支持更多的动作脚本命令；您可以使用这些命令实现更坚实的脚本。（请参见第 13 页的“新增的和经改进的语言元素”。）但是，如果您在现有脚本中使用这些命令中的任何一个，并且该脚本是以 Flash Player 7 为目标播放器发布的，则该脚本不会正常工作。

例如，如果您的脚本具有名为 `Error` 的函数，该脚本的编译可能好像是正确的，但在 `Flash Player 7` 中可能不按照预期的方式运行，因为 `Error` 现在在动作脚本中是内置类（因此是保留字）。您可以通过将该 `Error` 函数重命名为其它名称（例如 `ErrorCondition`），纠正脚本中的这一错误。

此外，在 `Flash Player 7` 中还进行了若干更改，这些更改会影响一个 `SWF` 文件访问其它 `SWF` 文件的方式、加载外部数据的方式以及访问本地设置和数据（例如隐私设置和本地永久共享对象）的方式。最后，一些现有功能的行为也有所变化。

如果现有脚本是针对 `Flash Player 6` 或更低版本编写的，而您希望以 `Flash Player 7` 为目标播放器发布这些脚本，则需要对这些脚本进行修改，使其符合 `Flash Player 7` 的实现方式，并按照设计工作。本节将讨论这些修改。

ECMA-262 版本 4 符合性

为了更加严格地遵从 `ECMA-262` 版本 4 建议，在 `Flash Player 7` 中进行了几项更改（请参见 www.mozilla.org/js/language/es4/index.html）。除了动作脚本 2.0 中提供的基于类的编程技术（请参见第 20 页的“新的面向对象编程模型”）之外，还增加了其它一些功能并更改了某些行为。此外，在以 `Flash Player 7` 为目标播放器进行发布和使用动作脚本 2.0 时，您可以将一种对象类型转换为另一种对象类型。有关更多信息，请参见第 34 页的“转换对象”。这些功能不要求更新现有脚本；但是，如果要将脚本发布到 `Flash Player 7`，然后继续对脚本进行修改和改进，则可能会需要使用这些功能。

与上文中提到的更改不同，下表中列出的更改（某些更改也可以提高 `ECMA` 符合性）可能会导致现有脚本表现出与以前不同的行为。如果在要发布到 `Flash Player 7` 的现有脚本中使用了这些功能，请检查这些更改，以确保代码仍会按照预期的方式工作，或者确定是否需要改写代码。需要特别注意的是，因为在某些情况下对 `undefined` 的计算结果是不同的，所以您应该初始化脚本中移植到 `Flash Player 7` 的所有变量。

以 <code>Flash Player 7</code> 为目标播放器发布的 <code>SWF</code> 文件	以早期 <code>Flash Player</code> 版本为目标播放器发布的 <code>SWF</code> 文件
支持区分大小写（仅大小写不同的变量名被解释为不同的变量）。这一更改还会影响使用 <code>#include</code> 加载的文件和使用 <code>LoadVars.load()</code> 加载的外部变量。有关更多信息，请参见第 26 页的“区分大小写”。	不支持区分大小写（仅大小写不同的变量名被解释为相同的变量）。
在数值上下文中求 <code>undefined</code> 的值，将返回 <code>NaN</code> 。 <pre>myCount +=1; trace(myCount); // NaN</pre>	在数值上下文中求 <code>undefined</code> 的值，将返回 <code>0</code> 。 <pre>myCount +=1; trace(myCount); // 1</pre>
把 <code>undefined</code> 转换为字符串时，结果为 <code>undefined</code> 。 <pre>firstname = "Joan "; lastname = "Flender"; trace(firstname + middlename + lastname); // Joan undefinedFlender</pre>	把 <code>undefined</code> 转换为字符串时，结果为 <code>""</code> （一个空字符串）。 <pre>firstname = "Joan "; lastname = "Flender"; trace(firstname + middlename + lastname); // Joan Flender</pre>

以 Flash Player 7 为目标播放器发布的 SWF 文件	以早期 Flash Player 版本为目标播放器发布的 SWF 文件
将字符串转换为布尔值时，如果字符串的长度大于零，结果为 true；如果是空字符串，则结果为 false。	将字符串转换为布尔值时，首先会将字符串转换为数字；如果数字为非零，结果为 true，否则结果为 false。
在设置数组的长度时，只有有效的数字字符串才设置该长度。例如，“6”正确，而“6”或“6xyz”不正确。 <pre>my_array=new Array(); my_array["6"] = "x"; trace(my_array.length); // 0 my_array["6xyz"] = "x"; trace(my_array.length); // 0 my_array["6"] = "x"; trace(my_array.length); // 7</pre>	在设置数组的长度时，即使是格式错误的数字字符串也能设置长度： <pre>my_array=new Array(); my_array["6"] = "x"; trace(my_array.length); // 7 my_array["6xyz"] = "x"; trace(my_array.length); // 7 my_array["6"] = "x"; trace(my_array.length); // 7</pre>

设置和本地数据的域名规则

在 Flash Player 6 中，在访问本地设置（例如摄像头或麦克风访问权限）或本地永久数据（共享对象）时默认采用超域匹配规则。即，用于在 here.xyz.com、there.xyz.com 和 xyz.com 承载的 SWF 文件的设置和数据都是共享的，并且都存储在 xyz.com 中。

在 Flash Player 7 中，默认采用完全域匹配规则。即，用于在 here.xyz.com 承载的文件的设置和数据存储在 here.xyz.com 中，用于在 there.xyz.com 承载的文件的设置和数据存储在 there.xyz.com 中，依此类推。

新属性 `System.exactSettings` 用于指定要使用的规则。以 Flash Player 6 或更高版本为目标播放器发布的文件支持此属性。对于以 Flash Player 6 为目标播放器发布的文件，默认值是 false，意味着采用超域匹配规则。对于以 Flash Player 7 为目标播放器发布的文件，默认值是 true，意味着采用完全域匹配规则。

如果您使用设置或永久本地数据并想要以 Flash Player 7 为目标播放器发布 Flash Player 6 SWF 文件，则最好在移植文件中将该值设置为 false。

有关更多信息，请参见第 618 页的 `System.exactSettings`。

SWF 文件之间的跨域和子域访问

在开发彼此通讯的一系列 SWF 文件时（例如，在使用 `loadMovie()`、`MovieClip.loadMovie()`、`MovieClipLoader.loadClip()` 或本地连接对象时），可以在不同的域中承载影片，或者在一个超域的不同子域中承载影片。

在以 Flash Player 5 或更低版本为目标播放器发布的文件中，对跨域或子域访问没有限制。

在以 Flash Player 6 为目标播放器发布的文件中，可以使用 `LocalConnection.allowDomain` 处理函数或 `System.security.allowDomain()` 方法指定允许的跨域访问（例如，让 `someOtherSite.com` 上的文件可以访问 `someSite.com` 上的文件），并且不需要执行任何命令就允许子域访问（例如，`store.someSite.com` 上的文件可以访问 `www.someSite.com` 上的文件）。

对于以 Flash Player 7 为目标播放器发布的文件，其实现 SWF 文件间访问的方式有两点与早期版本不同。首先，Flash Player 7 实现完全域匹配规则，而不是超域匹配规则。因此，所访问的文件（即使该文件是以早于 Flash Player 7 的 Player 版本为目标播放器发布的）必须显式允许跨域或子域访问；在下文将讨论这一主题。其次，在某一使用安全协议 (HTTPS) 的站点承载的文件必须显式允许从使用不安全协议 (HTTP 或 FTP) 的站点承载的文件的访问；在下一节中将讨论这一主题（请参见第 18 页的“SWF 文件之间的 HTTP 到 HTTPS 协议访问”）。

因为 Flash Player 7 实现完全域匹配规则，而不是超域匹配规则，所以，如果您希望从以 Flash Player 7 为目标播放器发布的文件访问现有脚本，则可能需要修改这些脚本。（您仍可以 Flash Player 6 为目标播放器发布这些修改后的文件）。如果您在文件中使用了任何 `LocalConnection.allowDomain()` 或 `System.security.allowDomain()` 语句并且指定了允许的超域站点，则必须更改参数以改为指定完全域匹配。以下代码显示您可能要进行更改的类型的示例：

```
// 位于 www.anyOldSite.com 的 SWF 文件中的 Flash Player 6 命令
// 要允许由在 www.someSite.com
// 或在 store.someSite.com 承载的 SWF 文件访问
System.security.allowDomain("someSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="someSite.com");
}
// 允许运行于 Flash Player 7 或更高版本中的 SWF 文件
// 访问的相应命令
System.security.allowDomain("www.someSite.com", "store.someSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="www.someSite.com" ||
        sendingDomain=="store.someSite.com");
}
```

如果您当前没有使用如下语句，可能还需要将这些语句添加到您的文件中。例如，如果您的 SWF 文件承载在 `www.someSite.com` 中并且想要允许由 `store.someSite.com` 上以 Flash Player 7 为目标播放器发布的 SWF 文件访问该文件，则必须将如下语句添加到 `www.someSite.com` 上的文件（您仍可以 Flash Player 6 为目标播放器发布 `www.someSite.com` 上的文件）。

```
System.security.allowDomain("store.someSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="store.someSite.com");
}
```

总之，如果您以 Flash Player 7 为目标播放器发布满足以下条件的文件，则可能需要修改文件以添加或更改 `allowDomain` 语句：

- 您实现了跨 SWF 脚本撰写（使用 `loadMovie()`、`MovieClip.loadMovie()`、`MovieClipLoader.LoadClip()` 或本地连接对象）。
- 被调用的任何版本的 SWF 文件不是承载在使用安全协议 (HTTPS) 的站点中，或者调用和被调用的 SWF 文件都承载在 HTTPS 站点中。（如果只有被调用的 SWF 文件承载在 HTTPS 站点中，请参见第 18 页的“[SWF 文件之间的 HTTP 到 HTTPS 协议访问](#)”。）
- 这些 SWF 文件不位于同一域中（例如，一个文件位于 `www.domain.com` 中，另一个文件位于 `store.domain.com` 中）。

您需要进行以下更改：

- 如果被调用的 SWF 文件是以 Flash Player 7 为目标播放器发布的，则在被调用的 SWF 文件中包括 `System.security.allowDomain` 或 `LocalConnection.allowDomain`，并且使用完全域名匹配。
- 如果被调用的 SWF 文件是以 Flash Player 6 为目标播放器发布的，则修改该被调用的文件以添加或更改 `System.security.allowDomain` 或 `LocalConnection.allowDomain` 语句，并且使用完全域名匹配，如本节前面部分的代码示例所示。您可以以 Flash Player 6 或 Flash Player 7 为目标播放器发布该修改后的文件。
- 如果被调用的 SWF 文件是以 Flash Player 5 或更低版本为目标播放器发布的，则将该被调用的文件移植到 Flash Player 6 或 Flash Player 7 中并添加 `System.security.allowDomain` 语句，并且使用完全域名匹配，如本节前面部分的代码示例所示。（在 Flash Player 5 或更低版本中不支持 `LocalConnection` 对象）。

SWF 文件之间的 HTTP 到 HTTPS 协议访问

如前面一节中所述，在 Flash Player 7 中已更改了跨域和子域访问规则。除了现在正实现的完全域匹配规则外，您必须显示允许在使用不安全协议的站点上承载的文件可以访问在使用安全协议 (HTTPS) 的站点上承载的文件。根据被调用的文件是以 Flash Player 7 还是 Flash Player 6 为目标播放器发布的，您必须或者实现 `allowDomain` 语句之一（请参见第 16 页的“SWF 文件之间的跨域和子域访问”），或者使用新的 `LocalConnection.allowInsecureDomain` 或 `System.security.allowInsecureDomain()` 语句。

警告：实现 `allowInsecureDomain()` 语句将会损及 HTTPS 协议提供的安全性。只有在您无法重新组织您的站点的情况下才应进行这些更改，以便从 HTTPS 协议提供所有 SWF 文件。

以下代码显示您可能要进行更改的类型的示例：

```
// 位于 https://www.someSite.com 的 Flash Player 6 SWF 文件中的命令
// 允许由在
// http://www.someSite.com 或 http://www.someOtherSite.com 承载的 Flash Player 7
// SWF 文件访问
System.security.allowDomain("someOtherSite.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="someOtherSite.com");
}
// Flash Player 7 SWF 文件中的相应命令
// 允许由在
// http://www.someSite.com 或 http://www.someOtherSite.com 承载的 Flash Player 7
// SWF 文件访问
System.security.allowInsecureDomain("www.someSite.com",
    "www.someOtherSite.com");
my_lc.allowInsecureDomain = function(sendingDomain) {
    return(sendingDomain=="www.someSite.com" ||
        sendingDomain=="www.someOtherSite.com");
}
```

如果您当前没有使用如下语句，可能还需要将这些语句添加到您的文件中。即使两个文件位于同一域中（例如，`http://www.domain.com` 中的文件正调用 `https://www.domain.com` 中的文件），也可能需要进行修改。

总之，如果您以 Flash Player 7 为目标播放器发布满足以下条件的文件，则可能需要修改您的文件以添加或更改语句：

- 您实现了跨 SWF 脚本撰写（使用 `loadMovie()`、`MovieClip.loadMovie()`、`MovieClipLoader.LoadClip()` 或本地连接对象）。
- 调用文件不是使用 HTTPS 协议承载的，被调用文件是使用 HTTPS 承载的。

您必须进行以下更改：

- 如果被调用文件是以 Flash Player 7 为目标播放器发布的，则在该被调用文件中包括 `System.security.allowInsecureDomain` 或 `LocalConnection.allowInsecureDomain`，并且使用完全域名匹配，如本节前面部分的代码示例所示。即使调用和被调用的 SWF 文件都位于同一域中，也要求此语句。
- 如果被调用文件是以 Flash Player 6 或更低版本为目标播放器发布的，并且调用和被调用文件都位于同一域中（例如，`http://www.domain.com` 中的文件调用 `https://www.domain.com` 中的文件），则不需要进行任何修改。
- 如果被调用文件是以 Flash Player 6 为目标播放器发布的，这些文件不位于同一域中，并且您不想将该被调用文件移植到 Flash Player 7 中，则修改该被调用文件以添加或更改 `System.security.allowDomain` 或 `LocalConnection.allowDomain`，并且使用完全域名匹配，如本节前面部分的代码示例所示。

- 如果被调用的文件是以 Flash Player 6 为目标播放器发布的并且您要将该被调用的文件移植到 Flash Player 7 中，则在该被调用的文件中包括 `System.security.allowInsecureDomain` 或 `LocalConnection.allowInsecureDomain`，并且使用完全域名匹配，如本节前面部分的代码示例所示。即使两个文件都位于同一域中，也要求此语句。
- 如果被调用文件是以 Flash Player 5 或更低版本为目标播放器发布的，并且两个文件不位于同一域中，您可以执行以下两个事项之一。您或者可以将该被调用文件移植到 Flash Player 6 中，添加或更改 `System.security.allowDomain` 语句，并且使用完全域名匹配，如本节前面部分的代码示例所示；或者可以将该被调用文件移植到 Flash Player 7 中，在该被调用文件中包括 `System.security.allowInsecureDomain` 语句，并且使用完全域名匹配，如本节前面部分的代码示例所示。

允许数据访问的服务器端策略文件

Flash 文档可以使用以下数据加载调用之一从外部源加载数据：`XML.load()`、`XML.sendAndLoad()`、`LoadVars.load()`、`LoadVars.sendAndLoad()`、`loadVariables()`、`loadVariablesNum()`、`MovieClip.loadVariables()`、`XMLSocket.connect()` 和 `Macromedia Flash Remoting (NetServices.createGatewayConnection)`。另外，SWF 文件可以在运行时导入运行时共享库 (RSL) 或另一个 SWF 文件中定义的资源。默认情况下，数据或 RSL 必须与加载该外部数据或媒体的 SWF 文件驻留在同一个域中。

若要使运行时共享库中的数据和资源可用于其它域中的 SWF 文件，应使用跨域策略文件。跨域策略文件是一个 XML 文件，该文件提供的方法可以使服务器指示其数据和文档可用于从某些域或所有域提供的 SWF 文件。服务器的策略文件指定的域所提供的所有 SWF 文件都将被允许访问该服务器中的数据或 RSL。

如果您加载外部数据，即使不想将任何文件移植到 Flash Player 7 中，也应创建策略文件。如果您使用 RSL，并且调用或被调用文件是以 Flash Player 7 为目标播放器发布的，则应创建策略文件。

有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

动作脚本编辑器更改

动作脚本编辑器在许多方面做了更新，从而使得它更加坚实，更便于使用。本节将总结这些更改。

自动换行 现在可以使用“脚本”窗格、“调试器”面板和“输出”面板中的“选项”弹出菜单来启用或禁用自动换行。还可以使用“动作”面板中的弹出菜单来切换自动换行。快捷键为 `Control+Shift+W` (Windows) 或 `Command+Shift+W` (Macintosh)。

查看上下文关联的帮助 当指针位于“动作”工具箱或“脚本”窗格中的某个动作脚本语言元素之上时，可以使用上下文菜单中的“查看帮助”项来显示关于该元素的帮助页。

导入脚本 从“动作”面板上的弹出菜单中选择“导入脚本”时，导入的脚本将被复制到代码文件中插入点所在的位置。在 Flash 的早期版本中，导入脚本将覆盖现有脚本的内容。

单击断点 若要在“调试器”面板中或“动作”面板的“脚本”窗格中在一行代码之前添加调试断点，可以在左边距处单击。在 Flash 的早期版本中，在左边距处单击会选择一行代码。选择一行代码的新方法是按住 `Control` 键并单击 (Windows) 或按住 `Command` 键并单击 (Macintosh)。

“动作”面板中不再有常规和专家模式 在 Flash 的早期版本中，可以在“动作”面板中以常规模式（即通过填写选项和参数来创建代码）工作或以专家模式（即直接在“脚本”窗格中添加命令）工作。在 Flash MX 2004 和 Flash MX Professional 2004，只能通过将命令直接添加到“脚本”窗格来使用“动作”面板。您仍可以从“动作”工具箱将命令拖到“脚本”窗格上，或者使用“脚本”窗格上方的“添加” (+) 按钮向脚本中添加命令。

固定多个脚本 可将一个 FLA 文件中的多个脚本固定在“动作”面板中“脚本”窗格的底部。在以前的 Flash 版本中，一次只能固定一个脚本。

脚本导航器 “动作”面板的左侧现在包含两个窗格：“动作”工具箱和一个新的脚本导航器。脚本导航器是 FLA 文件结构的可视化表示形式；您可以在这里浏览 FLA 文件以查找动作脚本代码。

用于编辑外部文件的集成“脚本”窗口（仅限 Flash Professional） 可以在“脚本”窗口中使用“动作脚本”编辑器（独立于“动作”面板）来编写和编辑外部脚本文件。“脚本”窗口中支持语法颜色、代码提示和其它首选参数，此外还有“动作”工具箱。若要显示“脚本”窗口，请使用“文件” > “新建”，然后选择要编辑的外部文件的类型。可以同时打开多个外部文件；文件名显示在沿“脚本”窗口顶部排列的选项卡上。（这些选项卡仅在 Windows 中出现。）

调试方面的更改

本节介绍可以提高脚本调试能力的一些更改。

“输出”窗口更改为“输出”面板 您现在可以像移动和停放 Flash 中的任何其它面板中一样移动和停放“输出”面板。

改进的编译时错误报告 除了提供更为坚实的异常处理功能外，动作脚本 2.0 还提供了大量新的编译时错误。有关更多信息，请参见第 739 页的附录 A “错误消息”。

改进的异常处理 Error 类和 throw 与 try..catch..finally 命令可用于实现更坚实的异常处理。

新的面向对象编程模型

自从在几年前引入以来，动作脚本语言已经得到了改进和发展。每一次发布 Flash 新版本时，该语言中都会再添加一些关键字、对象、方法和其它语言元素。但是，与 Flash 的早期版本不同，Flash MX 2004 和 Flash MX Professional 2004 引入了几个新的语言元素，这些元素采用比以前更为标准的方式来实现面向对象的编程。由于这些语言元素代表了对核心动作脚本语言的重大改进，因此它们代表了动作脚本的一个新版本：动作脚本 2.0。

动作脚本 2.0 并不是一种新语言，只是它包含的一组核心语言元素简化了面向对象程序的开发。由于引入了 class、interface、extends 和 implements 等关键字，因此现在的动作脚本语法对于熟悉其它语言的程序员来说更易于学习。新的程序员可以学习更为标准的术语，这些术语可以应用于他们将来学习的其它面向对象语言。

动作脚本 2.0 支持动作脚本语言的所有标准元素；它使您能够更加严格地遵守其它面向对象语言（如 Java）所采用的标准来编写脚本。动作脚本 2.0 主要用于满足中级或高级 Flash 程序员的需要，供他们用来创建需要实现类和子类的应用程序。动作脚本 2.0 还使您能够在创建变量时声明变量的对象类型（请参见第 33 页的“严格数据类型指定”），并且提供已大大改进了的编译器错误（请参见第 739 页的附录 A “错误消息”）。

下面列出了动作脚本 2.0 中新增的语言元素。

- `class`
- `extends`
- `implements`
- `interface`
- `dynamic`
- `static`
- `public`
- `private`
- `get`
- `set`
- `import`

动作脚本 2.0 的主要特点包括：

- 使用动作脚本 2.0 定义类或接口的脚本必须存储为外部脚本文件，并且在每个脚本中定义一个类；即，不能在“动作”面板中定义类和接口。
- 您可以隐式导入单个类文件（通过将这些文件存储在全局搜索路径或文档专用搜索路径指定的位置，然后在脚本中使用它们）或显式导入单个类文件（通过使用 `import` 命令）；也可以使用通配符导入包（一个目录中的一组类文件）。
- Flash Player 6 和更高版本支持用动作脚本 2.0 开发的应用程序。

小心：在 Flash MX 2004 中创建的新文件的默认发布设置是动作脚本 2.0。如果您计划修改现有 FLA 文件以使用动作脚本 2.0 语法，请确保该 FLA 文件在其发布设置中指定动作脚本 2.0。如果没有指定，则您的文件将不会正确编译，虽然 Flash 将不生成编译器错误。

有关在 Flash 中使用动作脚本 2.0 编写面向对象程序的更多信息，请参见第 137 页的第 9 章“使用动作脚本 2.0 创建类”。

第 2 章

动作脚本基础

动作脚本具有语法和标点规则，这些规则确定哪些字符和单词可以用于产生含义，并确定它们的撰写顺序。例如，在英语中，句点会结束一个句子。而在动作脚本中，分号会结束一个语句。

下面的一般规则适用于所有动作脚本。大多数动作脚本术语还具有其各自的要求；有关特定术语的规则，请参见第 181 页的第 12 章“动作脚本字典”中的相应条目。

动作脚本和 JavaScript 之间的差异

动作脚本与核心 JavaScript 编程语言类似。虽然您不需要了解 JavaScript 也可以使用和学习动作脚本，但是，如果您了解 JavaScript，会感到动作脚本并不陌生。

本手册并不是一本讲解一般编程的教材。有很多资源提供了有关一般编程概念和 JavaScript 语言的更多信息。

- 欧洲计算机制造商联合会 (European Computers Manufacturers Association, ECMA) 制订的 ECMA-262 规范源自 JavaScript，并作为 JavaScript 语言的国际标准。动作脚本基于 ECMA-262 规范。
- Netscape DevEdge Online 有一个 JavaScript 开发人员中心站点 (<http://developer.netscape.com/tech/javascript/index.html>)，该站点提供了有助于您了解动作脚本的文档和文章。最有价值的资源是核心 JavaScript 指南 (Core JavaScript Guide)。

下面列出了动作脚本和 JavaScript 的一些差异：

- 动作脚本不支持特定于浏览器的对象，例如，Document、Window 和 Anchor。
- 动作脚本不完全支持所有 JavaScript 内置对象。
- 动作脚本不支持某些 JavaScript 语法构造，例语句标签。
- 在动作脚本中，`eval()` 动作只能执行变量引用。

动作脚本的 Unicode 支持

Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 支持动作脚本的 Unicode 文本编码。这意味着您可以在动作脚本文件中包含不同语言的文本。例如，您可以在相同的文件中包含英语、日语和法语的文本。

您可以设置动作脚本首选参数，以指定在导入或导出动作脚本文件时使用的编码类型。您可以选择 UTF-8 编码或默认编码。UTF-8 是一种 8 位 Unicode 格式；默认编码是系统当前使用的语言所支持的编码形式，也称作传统代码页。

通常，如果您导入或导出 UTF-8 格式的动作脚本文件，请使用“UTF-8”首选参数。如果您导入或导出系统上使用的传统代码页中的文件，请使用“默认编码”首选参数。

如果在打开或导入文件时脚本中文本的外观与预期不符，请更改导入编码首选参数。如果在导出动作脚本文件时出现警告消息，可以更改导出编码首选参数或在动作脚本首选参数中关闭此警告。

为导入或导出动作脚本文件选择文本编码选项：

- 1 在“首选参数”对话框（“编辑” > “首选参数”）中，单击“动作脚本”选项卡。
- 2 在“编辑选项”下，执行下面的一项或两项操作：
 - 对于“打开 / 导入”，选择“UTF-8”以使用 Unicode 编码进行打开或导入，或者选择“默认编码”以使用系统当前所用语言的编码形式进行打开或导入。
 - 对于“保存 / 导出”，选择“UTF-8”以使用 Unicode 编码进行保存或导出，或者选择“默认编码”以使用系统当前所用语言的编码形式进行保存或导出。

关闭或打开导出编码警告：

- 1 在“首选参数”对话框（“编辑” > “首选参数”）中，单击“警告”选项卡。
- 2 选择或取消选择“导出 .as 文件过程中编码发生冲突时发出警告”。

小心：如果 SWF 文件路径的任何部分包含不能用 MBCS 编码方案表示的字符，“测试影片”命令（请参见第 61 页的“调试脚本”）将失败。例如，日语路径在英语系统上就不能正常工作。应用程序中所有使用该外部播放器的区域都受此限制。

术语

和任何脚本撰写语言一样，动作脚本使用自己的术语。以下列表提供了对重要动作脚本术语的介绍。

动作是在播放 SWF 文件时指示 SWF 文件执行某些任务的语句。例如，`gotoAndStop()` 将播放头放置到特定的帧或标签。在本手册中，术语*动作*和*语句*是可以互换的。

布尔值是 `true` 或 `false` 值。

类是可以创建以定义新对象类型的数据类型。若要定义类，请在外部脚本文件中（而不是您在“动作”面板上编写的脚本中）使用 `class` 关键字。

常数是不变的元素。例如，常数 `Key.TAB` 的含义始终不变：它代表键盘上的 Tab 键。常数对于比较值很有用。

构造函数是用于定义类的属性和方法的函数。根据定义，构造函数是类定义中与类同名的函数。例如，以下代码定义一个 `Circle` 类并实现一个构造函数：

```
// 文件 Circle.as
class Circle {
    private var radius:Number
    private var circumference:Number
    // 构造函数
    function Circle(radius:Number) {
        circumference = 2 * Math.PI * radius;
    }
}
```

在基于特定的类创建（实例化）对象时，也会使用术语构造函数。以下语句是内置 `Array` 类和自定义 `Circle` 类的构造函数：

```
my_array:Array = new Array();
my_circle:Circle = new Circle();
```


数据类型描述变量或动作脚本元素可以包含的信息的种类。动作脚本数据类型包括字符串、数字、布尔值、对象、影片剪辑、函数、空值和未定义。有关更多信息，请参见第 30 页的“[关于数据类型](#)”。

事件是 SWF 文件播放时发生的动作。例如，在加载影片剪辑，播放头进入帧，用户单击按钮或影片剪辑，或者用户按下键盘键时，会产生不同的事件。

事件处理函数是管理诸如 `mouseDown` 或 `load` 等事件的特殊动作。动作脚本事件处理函数共有两类：事件处理函数方法和事件侦听器。（还有两种事件处理函数 `on()` 和 `onClipEvent()`，您可以将它们直接分配给按钮和影片剪辑。）在“动作”工具箱中，每个具有事件处理函数方法或事件侦听器的动作脚本对象都有一个名为“Events”或“Listeners”的子类别。某些命令既可以用于事件处理函数，也可以用于事件侦听器，并且包括在上述两个子类别中。

表达式是代表值的动作脚本元件的任意合法组合。表达式由运算符和操作数组成。例如，在表达式 `x + 2` 中，`x` 和 `2` 是操作数，而 `+` 是运算符。

函数是可以向其传递参数并能够返回值的可重复使用的代码块。有关更多信息，请参见第 45 页的“[创建函数](#)”。

标识符是用于表示变量、属性、对象、函数或方法的名称。它的第一个字符必须是字母、下划线 (`_`) 或美元记号 (`$`)。其后的字符必须是字母、数字、下划线或美元记号。例如，`firstName` 是一个变量的名称。

实例是属于某个类的对象。类的每个实例均包含该类的所有属性和方法。例如，所有影片剪辑都是 `MovieClip` 类的实例，因此您可以将 `MovieClip` 类的任何方法或属性用于任何影片剪辑实例。

实例名称是脚本中用来表示影片剪辑实例和按钮实例的唯一名称。您可以使用属性检查器为舞台上的实例指定实例名称。例如，库中的主元件可以名为 `counter`，而 SWF 文件中该元件的两个实例可以使用实例名称 `scorePlayer1_mc` 和 `scorePlayer2_mc`。下面的代码用实例名称设置每个影片剪辑实例中名为 `score` 的变量：

```
_root.scorePlayer1_mc.score += 1;
_root.scorePlayer2_mc.score -= 1;
```

可以在命名实例时使用特殊的后缀，以便在键入代码时显示代码提示（请参见第 56 页的“[使用代码提示](#)”）。有关更多信息，请参见第 55 页的“[使用后缀触发代码提示](#)”。

关键字是有特殊含义的保留字。例如，`var` 是用于声明本地变量的关键字。不能使用关键字作为标识符。例如，`var` 不是合法的变量名。要查看关键字的列表，请参见第 30 页的“[关键字](#)”。

方法是与类关联的函数。例如，`getBytesLoaded()` 是与 `MovieClip` 类关联的内置方法。您也可以为基于内置类的对象或为基于您创建的类的对象，创建充当方法的函数。例如，在以下代码中，`clear()` 成为您先前定义的 `controller` 对象的方法：

```
function reset(){
    this.x_pos = 0;
    this.x_pos = 0;
}
controller.clear = reset;
controller.clear();
```

对象是属性和方法的集合；每个对象都有其各自的名称，并且都是特定类的实例。内置对象是在动作脚本语言中预先定义的。例如，内置的 `Date` 对象可以提供系统时钟的信息。

运算符是通过一个或多个值计算新值的术语。例如，加法 (+) 运算符可以将两个或更多个值相加到一起，从而产生一个新值。运算符处理的值称为操作数。

参数（也称作参量）是用于向函数传递值的占位符。例如，下面的 `welcome()` 函数使用它在参数 `firstName` 和 `hobby` 中接收到的两个值：

```
function welcome(firstName, hobby) {  
    welcomeText = "Hello, " + firstName + "I see you enjoy " + hobby;  
}
```

包是位于指定的类路径目录下，包含一个或多个类文件的目录（请参见第 148 页的“理解类路径”）。

属性是定义对象的特性。例如，`_visible` 是定义影片剪辑是否可见的属性，所有影片剪辑都有此属性。

目标路径是 SWF 文件中影片剪辑实例名称、变量和对象的分层结构地址。您可以在影片剪辑属性检查器中对影片剪辑实例进行命名。（主时间轴的名称始终为 `_root`。）您可以使用目标路径引导影片剪辑中的动作，或者获取或设置变量的值。例如，下面的语句是指向影片剪辑 `stereoControl` 内的变量 `volume` 的目标路径：

```
_root.stereoControl.volume
```

有关目标路径的更多信息，请参见“使用 Flash”帮助中的“绝对和相对目标路径”。

变量是包含任何数据类型的值的标识符。可以创建、更改和更新变量。可以检索它们存储的值以在脚本中使用。在下面的示例中，等号左侧的标识符是变量：

```
var x = 5;  
var name = "Lolo";  
var c_color = new Color(mcinstanceName);
```

有关变量的更多信息，请参见第 36 页的“关于变量”。

语法

与任何语言一样，动作脚本具有一定的语法规则，您必须遵守这些语法规则才能创建可正确编译和运行的脚本。本节介绍组成动作脚本语法的元素。

区分大小写

在区分大小写的编程语言中，仅大小写不同的变量名（`book` 和 `Book`）被视为互不相同。因此，最好遵循一致的大小写约定（如本手册中采用的约定），以便于识别动作脚本代码中函数和变量的名称。

为 Flash Player 7 或更高版本发布文件时，无论您使用的是动作脚本 1 还是动作脚本 2.0，Flash 都会实现区分大小写。这意味着关键字、类名、变量、方法名等均区分大小写。例如：

```
// 在面向 Flash Player 7  
// 以及动作脚本 1 或动作脚本 2.0 的文件中  
//  
// 设置两个不同对象的属性  
cat.hilite = true;  
CAT.hilite = true;  
  
// 创建三个不同的变量  
var myVar=10;  
var myvar=10;  
var mYvAr=10;  
// 不生成错误  
var array = new Array();  
var date = new Date();
```

此更改还会影响使用 `LoadVars.load()` 加载的外部变量。

此外，还将为外部脚本（例如，用 `#include` 命令导入的动作脚本 2.0 类文件或脚本）实现区分大小写。如果要为 Flash Player 7 发布文件，并且使用 `#include` 语句在脚本中添加了以前创建的外部文件，则应检查每个文件，确保始终采用了一致的大小写。执行这一任务的方法之一是在“脚本”窗口中打开该文件（仅限 Flash Professional），或者在新的 FLA 文件中将发布设置为 Flash Player 7，并将该文件的内容复制到“动作”面板。然后使用“检查语法”按钮（请参见第 60 页的“检查语法和标点”）或发布该文件；由于命名冲突导致的错误将出现在“输出”面板上。

在启用语法颜色后，大小写正确的语言元素在默认情况下为蓝色。有关更多信息，请参见第 30 页的“关键字”和第 54 页的“语法突出显示”。

点语法

在动作脚本中，点 (.) 用于指示与对象或影片剪辑相关的属性或方法。它还用于标识影片剪辑、变量、函数或对象的目标路径。点语法表达式以对象或影片剪辑的名称开头，后面跟着一个点，最后以要指定的元素结尾。

例如，`_x` 影片剪辑属性指示影片剪辑在舞台上的 *x* 轴位置。表达式 `ballMC._x` 引用影片剪辑实例 `ballMC` 的 `_x` 属性。

又例如，`submit` 是 `form` 影片剪辑中设置的变量，此影片剪辑嵌在影片剪辑 `shoppingCart` 之中。表达式 `shoppingCart.form.submit = true` 将实例 `form` 的 `submit` 变量设置为 `true`。

无论是表达对象的方法还是影片剪辑的方法，均遵循同样的模式。例如，`ball_mc` 影片剪辑实例的 `play()` 方法在 `ball_mc` 的时间轴中移动播放头，如下面的语句所示：

```
ball_mc.play();
```

点语法还使用两个特殊别名：`_root` 和 `_parent`。别名 `_root` 是指主时间轴。您可以使用 `_root` 别名创建一个绝对目标路径。例如，下面的语句调用主时间轴上影片剪辑 `functions` 中的函数 `buildGameBoard()`：

```
_root.functions.buildGameBoard();
```

您可以使用别名 `_parent` 引用当前对象嵌入到的影片剪辑。也可使用 `_parent` 创建相对目标路径。例如，如果影片剪辑 `dog_mc` 嵌入影片剪辑 `animal_mc` 的内部，则实例 `dog_mc` 的如下语句会指示 `animal_mc` 停止：

```
_parent.stop();
```

斜杠语法

斜杠语法在 Flash 3 和 4 中表示影片剪辑或变量的目标路径。虽然 Flash Player 7 仍支持此语法，但不建议使用此语法，而且动作脚本 2.0 中不支持斜杠语法。但是，如果要专门为 Flash Player 4 创建内容，则必须使用斜杠语法。有关更多信息，请参见第 751 页的“使用斜杠语法”。

大括号

动作脚本事件处理函数、类定义和函数用大括号 ({}) 组合在一起形成块。您可以在与声明同一行或下一行上放置一个左大括号，如下面的示例中所示。为使您的代码更具可读性，最好选择一种格式并始终如一地使用它。

```
// 事件处理函数
on (release) {
    myDate = new Date();
    currentMonth = myDate.getMonth();
}
```

```

on(release)
{
    myDate = new Date();
    currentMonth = myDate.getMonth();
}

// 类
class Circle(radius) {
}

class Square(side)
{
}

// 函数
circleArea = function(radius) {
    return radius * radius * MATH.PI;
}
squareArea = function(side)
{
    return side * side;
}

```

您可以检查脚本中的大括号是否匹配；请参见第 60 页的“检查语法和标点”。

分号

动作脚本语句以分号 (;) 结束，如以下示例所示：

```

var column = passedDate.getDay();
var row    = 0;

```

如果省略了结束分号，Flash 仍然能够成功地编译脚本。但是，使用分号是一个很好的脚本撰写习惯。

小括号

在定义函数时，要将所有参数都放在括号中：

```

function myFunction (name, age, reader){
    // 此处是您的代码
}

```

调用函数时，要将传递给该函数的所有参数都包含在括号中，如下所示：

```

myFunction ("Steve", 10, true);

```

您也可使用括号改写动作脚本的优先顺序或增强动作脚本语句的易读性。（请参见第 40 页的“运算符的优先级和结合律”。）

也可使用括号评估点语法中点左侧的表达式。例如，在下面的语句中，括号会使 `new Color(this)` 计算并创建一个 `Color` 对象：

```

onClipEvent (enterFrame) {
    (new Color(this)).setRGB(0xffffffff);
}

```

如果不使用括号，则必须添加一个语句来计算该表达式：

```
onClipEvent (enterFrame) {  
    myColor = new Color(this);  
    myColor.setRGB(0xffffffff);  
}
```

您可以检查脚本中的括号是否匹配；请参见第 60 页的“检查语法和标点”。

注释

极力建议使用注释将备注添加到脚本。注释有助于理解您关注的内容，如果您工作在一个合作的环境中或者要提供范例，注释还有助于向其他开发人员提供信息。如果在创建脚本时进行注释，即使简单的脚本也会更易于理解。

若要指示某一行或一行的某一部分是注释，请在该注释前加两个斜杠（//）：

```
on (release) {  
    // 创建新的 Date 对象  
    myDate = new Date();  
    currentMonth = myDate.getMonth();  
    // 将月份数转换为月份名称  
    monthName = calcMonth(currentMonth);  
    year = myDate.getFullYear();  
    currentDate = myDate.getDate();  
}
```

如果启用了语法颜色（请参见第 54 页的“语法突出显示”），注释在默认情况下为灰色。注释可以具有任意长度，这不会影响导出文件的大小，并且它们不必遵循动作脚本语法或关键字的规则。

如果要“注释掉”脚本的整个部分，请将其放在注释块中，而不是在每行开头添加 //。这种方法更为简单，当您注释掉脚本的大块代码以仅测试脚本的某些部分时，它非常有用。

若要创建注释块，请在命令行开头添加 /*，在末尾添加 */。例如，当以下脚本运行时，将不执行注释块中的任何代码：

```
// 运行以下代码  
var x:Number = 15;  
var y:Number = 20;  
// 不运行以下代码  
/*  
on (release) {  
    // 创建新的 Date 对象  
    myDate = new Date();  
    currentMonth = myDate.getMonth();  
    // 将月份数转换为月份名称  
    monthName = calcMonth(currentMonth);  
    year = myDate.getFullYear();  
    currentDate = myDate.getDate();  
}  
*/  
// 运行以下代码  
var name:String = "My name is";  
var age:Number = 20;
```

关键字

动作脚本保留一些单词用于该语言中的特定用途，因此不能将它们用作标识符，例如变量、函数或标签名称。下表列出了所有动作脚本关键字：

break	case	class	continue
default	delete	dynamic	else
extends	for	function	get
if	implements	import	in
instanceof	interface	intrinsic	new
private	public	return	set
static	switch	this	typeof
var	void	while	with

常数

常数是其值始终不变的属性。

例如，常数 BACKSPACE、ENTER、QUOTE、RETURN、SPACE 和 TAB 是 Key 对象的属性，指代键盘的按键。若要测试用户是否按下了 Enter 键，可以使用下面的语句：

```
if(Key.getCode() == Key.ENTER) {  
    alert = "Are you ready to play?";  
    controlMC.gotoAndStop(5);  
}
```

关于数据类型

数据类型描述变量或动作脚本元素可以包含的信息的种类。Flash 中内置了两种数据类型：原始数据类型和引用数据类型。原始数据类型是指字符串、数字和布尔值，它们都有一个常数值，因此可以包含它们所代表的元素的实际值。引用数据类型是指影片剪辑和对象，它们的值可能发生更改，因此它们包含对该元素的实际值的引用。包含原始数据类型的变量与包含引用类型的变量在某些情况下的行为是不同的。（请参见第 38 页的“在程序中使用变量”。）还有两类特殊的数据类型：空值和未定义。

在 Flash 中，任何不属于原始数据类型或影片剪辑数据类型的内置对象（如 Array 或 Math）均属于对象数据类型。

每种数据类型都有其各自的规则，下面的主题中将对对其进行介绍：

- 第 31 页的“字符串”
- 第 31 页的“数字”
- 第 32 页的“布尔值”
- 第 32 页的“Object”
- 第 32 页的“MovieClip”
- 第 32 页的“Null”
- 第 33 页的“Undefined”

当调试脚本时，可能需要确定表达式或变量的数据类型，以理解其特定行为的原理。可以用 typeof 运算符实现这一目的（请参见第 33 页的“确定项目的数据类型”）。

您可以使用以下转换函数之一将一种数据类型转换为另一种数据类型：`Array()`、`Boolean()`、`Number()`、`Object()`、`String()`。

字符串

字符串是诸如字母、数字和标点符号等字符的序列。在动作脚本语句中输入字符串的方式是将其放在单引号或双引号之间。字符串被当做字符，而不是变量进行处理。例如，在下面的语句中，“L7”是一个字符串：

```
favoriteBand = "L7";
```

可以使用加法 (+) 运算符连接 或合并两个字符串。动作脚本将字符串前面或后面的空格作为该字符串的文本部分。在下面的表达式中，逗号后有一个空格：

```
greeting = "Welcome," + firstName;
```

若要在字符串中包含引号，请在它前面放置一个反斜杠字符 (\)。这就是所谓的将字符转义。在动作脚本中，还有一些只能用特殊的转义序列才能表示的字符。下表提供了所有动作脚本转义符：

转义序列	字符
<code>\b</code>	退格符 (ASCII 8)
<code>\f</code>	换页符 (ASCII 12)
<code>\n</code>	换行符 (ASCII 10)
<code>\r</code>	回车符 (ASCII 13)
<code>\t</code>	制表符 (ASCII 9)
<code>\"</code>	双引号
<code>\'</code>	单引号
<code>\\</code>	反斜杠
<code>\000 - \377</code>	以八进制指定的字节
<code>\x00 - \xFF</code>	以十六进制指定的字节
<code>\u0000 - \uFFFF</code>	以十六进制指定的 16 位 Unicode 字符

数字

数字数据类型是双精度浮点数。您可以使用加 (+)、减 (-)、乘 (*)、除 (/)、求模 (%)、递增 (++) 和递减 (--) 等算术运算符来处理数字。也可使用内置的 `Math` 和 `Number` 类的方法来处理数字。下面的示例使用 `sqrt()`（平方根）方法返回数字 100 的平方根：

```
Math.sqrt(100);
```

有关更多信息，请参见第 40 页的“数值运算符”。

布尔值

布尔值是 `true` 或 `false` 中的一个。动作脚本也会在适当时将值 `true` 和 `false` 转换为 1 和 0。布尔值经常与动作脚本语句中通过比较控制脚本流的逻辑运算符一起使用。例如，在下面的脚本中，如果变量 `password` 为 `true`，则会播放该 SWF 文件：

```
onClipEvent (enterFrame) {  
    if (userName == true && password == true){  
        play();  
    }  
}
```

请参见第 45 页的“使用内置函数”和第 41 页的“逻辑运算符”。

Object

对象是属性的集合。每个属性都有名称和值。属性的值可以是任何的 Flash 数据类型，甚至可以是对象数据类型。这样就可以使对象相互包含（即将其嵌套）。若要指定对象及其属性，可以使用点 (.) 运算符。例如，在下面的代码中，`hoursWorked` 是 `weeklyStats` 的属性，而后者是 `employee` 的属性：

```
employee.weeklyStats.hoursWorked
```

您可以使用内置动作脚本对象来访问和处理特定种类的信息。例如，`Math` 对象具有一些方法，这些方法可以对传递给它们的数字执行数学运算。此示例使用 `sqrt()` 方法：

```
squareRoot = Math.sqrt(100);
```

动作脚本 `MovieClip` 对象具有一些方法，您可以使用这些方法控制舞台上的影片剪辑元件实例。此示例使用 `play()` 和 `nextFrame()` 方法：

```
mcInstanceName.play();  
mc2InstanceName.nextFrame();
```

您也可以创建自定义对象来组织 Flash 应用程序中的信息。若要使用动作脚本向应用程序添加交互操作，将需要许多不同的信息：例如，可能需要用户的姓名、球的速度、购物车中货物的名称、已加载的帧数、用户的邮编或上次所按键。通过创建自定义对象，可以将信息分组，简化您的脚本撰写过程，并且能重新使用您的脚本。

MovieClip

影片剪辑是 Flash 应用程序中可以播放动画的元件。它们是唯一引用图形元素的数据类型。`MovieClip` 数据类型允许您使用 `MovieClip` 类的方法控制影片剪辑元件。可以使用点 (.) 运算符调用这些方法，如下所示：

```
my_mc.startDrag(true);  
parent_mc.getURL("http://www.macromedia.com/support/" + product);
```

Null

空值数据类型只有一个值，即 `null`。此值意味着“没有值”，即缺少数据。`null` 值可以用在各种情况中。下面是一些示例：

- 指示变量尚未接收到值
- 指示变量不再包含值
- 作为函数的返回值，指示函数没有可以返回的值
- 作为函数的参数，指示省略了一个参数

Undefined

未定义的数据类型有一个值，即 `undefined`，它用于尚未分配值的变量。

确定项目的数据类型

当测试和调试程序时，您可能会发现看来与不同项目的数据类型相关的问题。这种情况下，您可能需要确定项目的数据类型。为此，使用 `typeof` 运算符，如以下示例所示：

```
trace(typeof(variableName));
```

有关测试和调试的更多信息，请参见第 49 页的第 3 章 “编写和调试脚本”。

为元素指定数据类型

Flash 自动为以下类型的语言元素指定数据类型，如下一节 “自动数据类型指定” 中所述：

- 变量
- 传递给函数、方法或类的参数
- 函数或方法返回的值
- 创建为现有类的子类的对象

不过，您也可以显式地为项目指定数据类型，这有助于防止或诊断脚本中的某些错误。有关更多信息，请参见第 33 页的 “严格数据类型指定”。

自动数据类型指定

在 Flash 中，不必将项目明确地定义为包含数字、字符串或其它数据类型。Flash 将在指定项目时确定其数据类型：

```
var x = 3;
```

在表达式 `var x = 3` 中，Flash 会评估运算符右侧的元素，然后确定它的数据类型为数字。后面的赋值运算可以更改 `x` 的类型；例如语句 `x = "hello"` 会将 `x` 的类型更改为字符串。尚未赋值的变量的类型为 `undefined`。

动作脚本会在表达式需要时自动转换数据类型。例如，当向 `trace()` 动作传递值时，`trace()` 会自动将该值转换为字符串，并将其发送到 “输出” 面板。在带有运算符的表达式中，动作脚本会根据需要转换数据类型；例如，当用于字符串时，`+` 运算符需要另一个操作数也是字符串：

```
"Next in line, number " + 7
```

动作脚本会将数字 7 转换为字符串 "7"，并将它添加到第一个字符串的结尾，从而产生下面的字符串：

```
"Next in line, number 7"
```

严格数据类型指定

动作脚本 2.0 允许在创建变量时显式声明其对象类型；这称作严格数据类型指定。因为数据类型不匹配会触发编译器错误，所以严格数据类型指定有助于避免为现有变量指定错误的数据类型。若要为某个项目指定特定的数据类型，请使用 `var` 关键字和后冒号语法指定其类型：

```
// 严格指定变量或对象的类型  
var x:Number = 7;  
var birthday>Date = new Date();
```

```
// 参数的严格类型指定
```

```
function welcome(firstName:String, age:Number){
}

// 参数和返回值的严格类型指定
function square(x:Number):Number {
    var squared = x*x;
    return squared;
}
```

由于在严格指定变量的数据类型时必须使用 `var` 关键字，因此不能严格指定全局变量的类型（请参见第 37 页的“确定变量的范围和声明变量”）。

您可以根据内置类（`Button`、`Date`、`MovieClip` 等）以及您创建的类和接口来声明对象的数据类型。例如，如果您在一个名为 `Student.as` 的文件中定义了 `Student` 类，则可以指定您创建的对象属于类型 `Student`：

```
var student:Student = new Student();
```

您也可以指定对象属于类型 `Function` 或 `Void`。

使用严格类型指定有助于确保您不会因为疏忽而为对象指定错误的值类型。Flash 将在编译时检查类型指定不匹配错误。例如，假设您键入以下代码：

```
// 在 Student.as 类文件中
class Student {
    var status:Boolean; // Student 对象的属性
}
```

```
// 在脚本中
var studentMaryLago:Student = new Student();
studentMaryLago.status = "enrolled";
```

当 Flash 编译此脚本时，将生成“类型不匹配”错误。

严格数据类型指定的另一个优点是，对于严格指定类型的内置对象，Flash MX 2004 会自动显示代码提示。有关更多信息，请参见第 54 页的“严格指定对象类型以触发代码提示”。

使用动作脚本 1 发布的文件在编译时不遵守严格数据类型指定。因此，对于严格指定了类型的变量，即使赋给它的值类型不正确，也不会生成编译器错误。

```
var x:String = "abc"
x = 12 ; // 在动作脚本 1 中不会出现任何错误，但在动作脚本 2 中会出现类型不匹配错误
```

其原因是，为动作脚本 1 发布文件时，Flash 以斜杠语法（而不是严格数据类型指定）解释像 `var x:String = "abc"` 这样的语句。（动作脚本 2.0 不支持斜杠语法。）这可能会造成将对象分配给类型错误的变量，导致编译器允许非法的方法调用和未定义的属性引用直接通过，而不报告错误。

因此，如果要实现严格数据类型指定，应确保为动作脚本 2.0 发布文件。

转换对象

动作脚本 2.0 允许将一种数据类型转换为另一种数据类型。Flash 使用的转换运算符采用函数调用的形式，并且符合 ECMA-262 版本 4 建议中指定的显式强制。转换允许您声明一个对象属于某一类型，这样，在执行类型检查时，编译器会将该对象视作具有一组其原始类型不包含的属性。例如，在迭代一组类型可能不同的对象时，这将很有用。

在为 Flash Player 7 或更高版本发布的文件中，在运行时失败的转换语句返回 `null`。在为 Flash Player 6 发布的文件中，未实现对失败转换的任何运行时支持。

转换的语法为 `type(item)`，表示您希望编译器将 `item` 视作数据类型为 `type` 的项目。转换实质上是一个函数调用，如果转换失败，该函数调用将返回 `null`。如果转换成功，该函数调用将返回原对象。但是，将项目转换为在外部类文件中创建的数据类型时，即使转换在运行时失败，编译器也不会生成类型不匹配错误。

```
// 在 Animal.as 中
class Animal { }
```

```
// 在 Dog.as 中
class Dog extends Animal { function bark (){} }
```

```
// 在 Cat.as 中
class Cat extends Animal { function meow (){} }
```

```
// 在 FLA 文件中
var spot:Dog = new Dog();
var temp:Cat = Cat (spot); // 声明一个 Dog 对象的类型为 Cat
temp.meow(); // 不执行任何动作，但也不生成任何编译器错误
```

在此示例中，您向编译器声明了 `temp` 是一个 `Cat` 对象，因此，编译器认为 `temp.meow()` 是一条合法语句。但是，编译器不知道该转换将失败（即，您尝试将一个 `Dog` 对象转换为 `Cat` 类型），因此不会发生任何编译时错误。如果在脚本中加入一条检查语句，检查转换是否成功，在运行时您会发现类型不匹配错误。

```
var spot:Dog = new Dog();
var temp:Cat = Cat (spot);
trace(temp); // 在运行时显示 null
```

可以将表达式的类型转换为接口。如果该表达式是一个实现该接口的对象，或具有实现该接口的基类，则返回该对象。否则，将返回 `null`。

下面的示例显示转换内置对象类型的结果。如 `with(results)` 代码块中的第一行所示，非法转换（在此示例中为将字符串转换为影片剪辑）将返回 `null`。如最后两行所示，转换为 `null` 或 `undefined` 将返回 `undefined`。

```
var mc:MovieClip;
var arr:Array;
var bool:Boolean;
var num3:Number;
var obj:Object;
var str:String;
_root.createTextField("results",2,100,100,300,300);
with(results){
text = "type MovieClip :"+(typeof MovieClip(str)); // 返回 null
text += "\ntype object :"+(typeof Object(str)); // 返回 object
text += "\ntype Array :"+(typeof Array(num3)); // 返回 object
text += "\ntype Boolean :"+(typeof Boolean(mc)); // 返回 boolean
text += "\ntype String :"+(typeof String(mc)); // 返回 string
text += "\ntype Number :"+(typeof Number(obj)); // 返回 number
text += "\ntype Function :"+(typeof Function(mc)); // 返回 object
text += "\ntype null :"+(typeof null(arr)); // 返回 undefined
text += "\ntype undefined :"+(typeof undefined(obj)); // 返回 undefined
}
// “输出” 面板中的结果
type MovieClip :null
type object :object
type Array :object
type Boolean :boolean
type String :string
type Number :number
```

```
type Function :object
type null :undefined
type undefined :undefined
```

原始数据类型（例如，Boolean、Date 和 Number）不能用名称相同的转换运算符覆盖。

关于变量

变量是包含信息的容器。容器本身始终不变，但内容可以更改。通过在 SWF 文件播放时更改变量的值，可以记录和保存用户操作的信息，记录 SWF 文件播放时更改的值，或者计算某个条件是 true 还是 false。

当首次定义变量时，最好为该变量指定一个已知值。这就是所谓的初始化变量，而且通常在 SWF 文件的第一帧中完成。初始化变量有助于在播放 SWF 文件时跟踪和比较变量的值。

变量可以包含任何类型的数据（请参见第 30 页的“关于数据类型”）。变量包含的数据类型影响在脚本中为变量赋值时变量值的变化方式。

变量中可以存储的常见信息类型包括 URL、用户名、数学运算的结果、事件发生的次数，以及是否单击了某个按钮，等等。每个 SWF 文件和影片剪辑实例都有一组变量，每个变量都有其各自的值，与其它 SWF 文件或影片剪辑中的变量无关。

若要测试变量的值，可以使用 trace() 动作向“输出”面板发送值。例如，trace(hoursWorked) 会在测试模式下将变量 hoursWorked 的值发送给“输出”面板。也可在测试模式下，在调试器中检查和设置变量值。有关更多信息，请参见第 70 页的“使用 trace 语句”和第 64 页的“显示和修改变量”。

命名变量

变量名称必须遵守下面的规则：

- 它必须是标识符（请参见第 24 页的“术语”）。
- 它不能是关键字或动作脚本文本，例如 true、false、null 或 undefined。
- 它在其范围内必须是唯一的（请参见第 37 页的“确定变量的范围和声明变量”）。

此外，您不应将动作脚本语言中的任何元素用作变量名称；这样做可能会导致语法错误或意外的结果。例如，如果您将一个变量命名为 String，然后尝试使用 new String() 创建一个 String 对象，则这一新对象是未定义的。

```
hello_str = new String();
trace(hello_str.length); // 返回 0
```

```
String = "hello"; // 为变量提供与内置类相同的名称
hello_str = new String();
trace(hello_str.length); // 返回未定义
```

动作脚本编辑器支持内置类和基于这些类的变量的代码提示。如果您需要 Flash 为指定给变量的特定对象类型提供代码提示，则可以精确键入变量或使用特定的后缀给该变量命名。

例如，假设您键入以下代码：

```
var members:Array = new Array();
members.
```

只要您一键入句点 (.)，Flash 就会显示可用于 Array 对象的方法和属性的列表。有关更多信息，请参见第 54 页的“编写触发代码提示的代码”。

确定变量的范围和声明变量

变量的范围 是指变量在其中已知并且可以引用的区域。在动作脚本中有三种类型的变量范围：

- **本地变量**在声明它们的函数体（由大括号界定）内可用。
- **时间轴变量**可用于该时间轴上的任何脚本。
- **全局变量**和函数对于文档中的每个时间轴和范围均可见。

注意：您创建的动作脚本 2.0 类支持公共、私有和静态变量范围。有关更多信息，请参见第 144 页的“控制成员访问”和第 145 页的“创建类成员”。

本地变量

若要声明本地变量，请在函数体内部使用 `var` 语句。本地变量的使用范围只限于它的代码块，它会在该代码块结束时到期。没有在代码块中声明的本地变量会在它的脚本结束时到期。

例如，变量 `i` 和 `j` 经常用作循环计数器。在下面的示例中，`i` 用作本地变量；它只存在于函数 `makeDays()` 的内部：

```
function makeDays() {  
    var i;  
    for( i = 0; i < monthArray[month]; i++ ) {  
  
        _root.Days.attachMovie( "DayDisplay", i, i + 2000 );  
  
        _root.Days[i].num = i + 1;  
        _root.Days[i]._x = column * _root.Days[i]._width;  
        _root.Days[i]._y = row * _root.Days[i]._height;  
  
        column = column + 1;  
  
        if ( column == 7 ) {  
  
            column = 0;  
            row = row + 1;  
        }  
    }  
}
```

本地变量也可防止出现名称冲突，名称冲突可能会导致应用程序出现错误。例如，如果使用 `name` 作为本地变量，则可以用它在一个上下文中存储用户名，而在另一个上下文中存储影片剪辑实例名称；因为这些变量是在不同的范围中运行的，所以它们不会有冲突。

在函数体中使用本地变量是一个很好的习惯，这样该函数可以充当独立的代码。本地变量只有在它自己的代码块中才是可更改的。如果函数中的表达式使用全局变量，则在该函数以外也可以更改它的值，这样也更改了该函数。

可以在定义本地变量时为其指定数据类型，这有助于防止将类型错误的数据赋给现有变量。有关更多信息，请参见第 33 页的“严格数据类型指定”。

时间轴变量

时间轴变量可用于该时间轴上的任何脚本。要声明时间轴变量，应在该时间轴中的所有帧上都初始化这些变量。应确保首先初始化变量，然后尝试在脚本中访问它。例如，如果将代码 `var x = 10;` 放置在第 20 帧上，则附加到第 20 帧之前的任何帧上的脚本都无法访问该变量。

全局变量

全局变量和函数对于您的文档中的每一时间轴和范围而言都是可见的。若要创建具有全局范围的变量，请在变量名称前使用 `_global` 标识符，并且不使用 `var =` 语法。例如，以下代码创建全局变量 `myName`：

```
var _global.myName = "George"; // 语法错误
_global.myName = "George";
```

但是，如果您使用与全局变量相同的名称初始化一个本地变量，则在处于该本地变量的范围内时对该全局变量不具有访问权限：

```
_global.counter = 100;
counter++;
trace(counter); // 显示 101
function count(){
    for( var counter = 0; counter <= 10 ; counter++ ) {
        trace(counter); // 显示 0 到 10
    }
}
count();
counter++;
trace(counter); // 显示 102
```

在程序中使用变量

您必须在脚本中声明变量，然后才能在表达式中使用它。如果使用未声明的变量（如以下示例所示），该变量的值将是 `NaN` 或 `undefined`，并且您的脚本可能产生意外的结果：

```
var squared = x*x;
trace(squared); // NaN
var x = 6;
```

在下面的示例中，声明变量 `x` 的语句必须排在第一，这样 `squared` 就可以替换为一个值：

```
var x = 6;
var squared = x*x;
trace(squared); // 36
```

当您将未定义的变量传递给方法或函数时，将出现类似的行为：

```
getURL(myWebSite); // 无动作
var myWebSite = "http://www.macromedia.com";

var myWebSite = "http://www.macromedia.com";
getURL(myWebSite); // 浏览器显示 www.macromedia.com
```

在一个脚本中，可以多次更改变量的值。变量包含的数据类型会影响如何以及何时更改变量。原始数据类型（例如字符串和数字）是按值进行传递的。这意味着变量的实际内容会传递给变量。

在下面的示例中，`x` 设置为 15，该值会复制到 `y` 中。在第 3 行中将 `x` 的值更改为 30 后，`y` 的值仍然为 15，这是因为 `y` 并不会参照 `x` 来改变它的值；它的值为在第 2 行中接收到的 `x` 的值。

```
var x = 15;
var y = x;
var x = 30;
```

又例如，变量 `inValue` 包含一个原始值 3，因此实际的值会传递给 `sqrt()` 函数，而返回值为 9：

```
function sqrt(x){
    return x * x;
}
```

```
var inValue = 3;
var out = sqrt(inValue);
```

变量 `inValue` 的值不会更改。

对象数据类型可以包含大量和复杂的信息，所以属于此类型的变量并不包含实际的值；它包含的是对值的引用。这种引用类似于指向变量内容的别名。当变量需要知道它的值时，该引用会查询内容，然后返回答案，而无需将该值传递给变量。

下面是按引用进行传递的示例：

```
var myArray = ["tom", "josie"];
var newArray = myArray;
myArray[1] = "jack";
trace(newArray);
```

上面的代码创建了一个名为 `myArray` 的数组对象，它包含两个元素。然后创建了变量 `newArray`，并向它传递了对 `myArray` 的引用。当 `myArray` 的第二个元素变化时，它将影响引用它的每个变量。`trace()` 动作会向“输出”面板发送 `tom, jack`。

在下面的示例中，`myArray` 包含一个数组对象，因此它会按引用传递给函数 `zeroArray()`。`zeroArray()` 函数会更改 `myArray` 中的数组的内容。

```
function zeroArray (theArray){
    var i;
    for (i=0; i < theArray.length; i++) {
        theArray[i] = 0;
    }
}
```

```
var myArray = new Array();
myArray[0] = 1;
myArray[1] = 2;
myArray[2] = 3;
zeroArray(myArray);
```

函数 `zeroArray()` 会将数组对象作为参数来接受，并将该数组的所有元素设置为 `0`。因为该数组是按引用进行传递的，所以该函数可以修改它。

使用运算符处理表达式中的值

表达式是 Flash 可以计算并返回值的任何语句。可以通过组合运算符和值或者调用函数来创建表达式。

运算符是指定如何组合、比较或修改表达式值的字符。运算符对其执行运算的元素称为操作数。例如，在下面的语句中，`+` 运算符会将数值文本的值添加到变量 `foo` 的值中；`foo` 和 `3` 就是操作数：

```
foo + 3
```

本节介绍关于常见类型的运算符、运算符优先级和运算符结合律的一般规则。有关此处提到的每个运算符的详细信息以及这些类别中未包含的特殊运算符，请参见[第 181 页的第 12 章“动作脚本字典”](#)中的相应条目。

运算符的优先级和结合律

当在同一语句中使用两个或多个运算符时，一些运算符会优先于其它的运算符。动作脚本按照一个精确的层次来确定首先执行哪些运算符。例如，乘法总是先于加法执行；但是，括号中的项目会优先于乘法。因此，如果没有括号，动作脚本会在下面的示例中首先执行乘法：

```
total = 2 + 4 * 3;
```

结果为 14。

但是当加法运算在括号内时，动作脚本会首先执行加法：

```
total = (2 + 4) * 3;
```

结果为 18。

当两个或更多个运算符优先级相同时，它们的结律会确定它们的执行顺序。结合律可以是 从左到右 或者 从右到左。例如，乘法运算符具有从左到右的结合律；因此，下面两个语句是等效的：

```
total = 2 * 3 * 4;  
total = (2 * 3) * 4;
```

要查看包含所有运算符及其优先级和结合律的表，请参见第 743 页的附录 B “运算符的优先级和结合律”。

数值运算符

数值运算符可以执行加法、减法、乘法、除法运算，也可以执行其它算术运算。

增量运算符最常见的用法是 `i++`，而不是比较繁琐的 `i = i+1`。可以在操作数前面或后面使用增量运算符。在下面的示例中，`age` 首先递增，然后再与数字 30 进行比较：

```
if (++age >= 30)
```

在下面的示例中，`age` 在执行比较之后递增：

```
if (age++ >= 30)
```

下表列出了动作脚本数值运算符：

运算符	执行的运算
+	加法
*	乘法
/	除法
%	求模（除后的余数）
-	减法
++	递增
--	递减

比较运算符

比较运算符用于比较表达式的值，然后返回一个布尔值（`true` 或 `false`）。这些运算符最常用于循环语句和条件语句中。在下面的示例中，如果变量 `score` 为 100，则加载特定的 SWF 文件；否则将加载另外一个 SWF 文件：

```
if (score > 100){  
    loadMovieNum("winner.swf", 5);  
}
```



```
} else {  
    loadMovieNum("loser.swf", 5);  
}
```

下表列出了动作脚本比较运算符：

运算符	执行的运算
<	小于
>	大于
<=	小于或等于
>=	大于或等于

字符串运算符

+ 运算符在处理字符串时会有特殊效果：它会将两个字符串操作数连接起来。例如，下面的语句会将 "Congratulations," 连接到 "Donna!"：

```
"Congratulations, " + "Donna!"
```

结果是 "Congratulations, Donna!"。如果 + 运算符的操作数中只有一个是字符串，则 Flash 会将另一个操作数转换为字符串。

比较运算符 >、>=、< 和 <= 在处理字符串时也有特殊的效果。这些运算符会比较两个字符串，以确定哪一个字符串按字母数字顺序排在前面。只有在两个操作数都是字符串时，比较运算符才会执行字符串比较。如果只有一个操作数是字符串，动作脚本会将两个操作数都转换为数字，然后执行数值比较。

逻辑运算符

逻辑运算符对布尔值（true 和 false）进行比较，然后返回第三个布尔值。例如，如果两个操作数都评估为 true，则逻辑“与”运算符（&&）将返回 true。如果其中一个或两个操作数评估为 true，则逻辑“或”运算符（||）将返回 true。逻辑运算符通常与比较运算符结合使用，以确定 if 动作的条件。例如，在下面的脚本中，如果两个表达式都为 true，则将执行 if 动作：

```
if (i > 10 && _framesloaded > 50){  
    play();  
}
```

下表列出了动作脚本逻辑运算符：

运算符	执行的运算
&&	逻辑“与”
	逻辑“或”
!	逻辑“非”

按位运算符

按位运算符在内部处理浮点数，将它们转换为 32 位整型。执行的确切运算取决于运算符，但是所有的按位运算都会分别评估 32 位整型的每个二进制位，从而计算新的值。

下表列出了动作脚本按位运算符：

运算符	执行的运算
&	按位 “与”
	按位 “或”
^	按位 “异或”
~	按位 “非”
<<	左移位
>>	右移位
>>>	右移位填零

等于运算符

可以使用等于 (==) 运算符确定两个操作数的值或标识是否相等。这一比较运算会返回一个布尔值 (true 或 false)。如果操作数为字符串、数字或布尔值，它们会按照值进行比较。如果操作数为对象或数组，它们将按照引用进行比较。

用赋值运算符检查等式是常见的错误。例如，下面的代码会将 x 与 2 进行比较：

```
if (x == 2)
```

在同一示例中，表达式 x = 2 是错误的，因为它不会比较操作数，而是将值 2 赋予变量 x。

严格等于 (===) 运算符与等于运算符相似，但是有一个很重要的差异：严格等于运算符不执行类型转换。如果两个操作数属于不同的类型，严格等于运算符会返回 false。严格不等于 (!==) 运算符会返回严格等于运算符的反值。

下表列出了动作脚本等于运算符：

运算符	执行的运算
==	等于
===	严格等于
!=	不等于
!==	严格不等于

赋值运算符

可以使用赋值 (=) 运算符为变量赋值，如下例所示：

```
var password = "Sk8tEr";
```

还可以使用赋值运算符给同一表达式中的多个变量赋值。在下面的语句中，a 的值会被赋予变量 b、c 和 d：

```
a = b = c = d;
```

也可以使用复合赋值运算符联合多个运算：复合运算符可以对两个操作数都进行运算，然后将新值赋予第一个操作数。例如，下面两个语句是等效的：

```
x += 15;
x = x + 15;
```

赋值运算符也可以用在表达式的中间，如下例所示：

```
// 如果口味不是香草味，则输出消息。
if ((flavor = getIceCreamFlavor()) != "vanilla") {
    trace ("Flavor was " + flavor + ", not vanilla.");
}
```

此代码与下面的稍显繁琐的代码是等效的：

```
flavor = getIceCreamFlavor();
if (flavor != "vanilla") {
    trace ("Flavor was " + flavor + ", not vanilla.");
}
```

下表列出了动作脚本赋值运算符：

运算符	执行的运算
=	赋值
+=	相加并赋值
-=	相减并赋值
*=	相乘并赋值
%=	求模并赋值
/=	相除并赋值
<<=	按位左移位并赋值
>>=	按位右移位并赋值
>>>=	右移位填零并赋值
^=	按位“异或”并赋值
=	按位“或”并赋值
&=	按位“与”并赋值

点运算符和数组访问运算符

可以使用点运算符 (.) 和数组访问运算符 ([]) 来访问内置或自定义的动作脚本对象属性，包括影片剪辑的属性。

点运算符在左侧使用对象名称，而在右侧使用属性或变量的名称。属性或变量名称不能是字符串或评估为字符串的变量；它必须是一个标识符。以下示例使用点运算符：

```
year.month = "June";
year.month.day = 9;
```

点运算符和数组访问运算符执行相同的功能，但是点运算符将标识符作为其属性，而数组访问运算符则会将其内容评估为名称，然后访问该已命名属性的值。例如，以下表达式会访问影片剪辑 `rocket` 中的同一个变量 `velocity`：

```
rocket.velocity;
rocket["velocity"];
```

可以使用数组访问运算符动态设置和检索实例名称和变量。例如，在下面的代码中，将计算 `[]` 运算符中的表达式，计算结果将用作从影片剪辑 `name` 中获取的变量的名称：

```
name["mc" + i]
```

还可以使用 `eval()` 函数，如下所示：

```
eval("mc" + i)
```

数组访问运算符也可以用在赋值语句的左侧。这使您可以动态设置实例、变量和对象的名称，如下例所示：

```
name[index] = "Gary";
```

通过构建一个其中的元素也是数组的数组，可以在动作脚本中创建多维数组。若要访问多维数组的元素，可以将数组访问运算符进行自我嵌套，如下例所示：

```
var chessboard = new Array();
for (var i=0; i<8; i++) {
    chessboard.push(new Array(8));
}
function getContentsOfSquare(row, column){
    chessboard[row][column];
}
```

您可以检查脚本中的 `[]` 运算符是否匹配；请参见第 60 页的“检查语法和标点”。

指定对象的路径

若要使用动作控制影片剪辑或加载的 SWF 文件，则必须指定其名称和地址，这个名称和地址就称作目标路径。

在动作脚本中，可以通过其实例名称标识影片剪辑。例如，在下面的语句中，名为 `star` 的影片剪辑的 `_alpha` 属性设置为 50% 的可见性：

```
star._alpha = 50;
```

为影片剪辑指定实例名称：

- 1 在舞台上选择影片剪辑。
- 2 在属性检查器中输入实例名称。

标识加载的 SWF 文件：

- 使用 `_levelX`，其中 *X* 是在加载 SWF 文件的 `loadMovie()` 动作中所指定的级别号。

例如，加载到第 5 级的 SWF 文件的目标路径为 `_level5`。下面的示例将一个 SWF 文件加载到第 5 级中，并将其可见性设置为 `false`：

```
onClipEvent (load) {
    loadMovieNum("myMovie.swf", 5);
}
onClipEvent (enterFrame) {
    _level5._visible = false;
}
```

输入 SWF 文件的目标路径：

- 在“动作”面板（“窗口” > “开发” > “动作”）中，单击“插入目标路径”按钮，然后从出现的列表中选择影片剪辑。

有关目标路径的更多信息，请参见“使用 Flash”帮助中的“绝对和相对目标路径”。

使用内置函数

函数是可以在 SWF 文件中的任意位置重用的动作脚本代码块。如果您将值当作参数传递给函数，该函数将对这些值执行运算。函数也可以返回值。

Flash 具有一些内置函数，可用于访问特定的信息，执行特定的任务，例如，获取承载 SWF 文件的 Flash Player 的版本号 (`getVersion()`)。属于对象的函数称作方法。不属于对象的函数称作顶级函数，可以在“动作”面板的“Functions”类别中找到这些函数。

每个函数都有其各自的特性，而某些函数需要您传递特定的值。如果传递的参数多于函数的需要，多余的值将被忽略。如果您没有传递必需的参数，则将为空的参数指定 `undefined` 数据类型，这可能会在导出脚本时导致错误。若要调用函数，该函数必须位于播放头到达的帧中。

若要调用函数，只需使用函数名称并传递所有必需的参数：

```
isNaN(someVar);
getTimer();
eval("someVar");
```

有关各个函数的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的相应条目。

创建函数

您可以定义函数，对传递的值执行一系列语句。函数也可以返回值。一旦定义了函数，就可以从任意一个时间轴中调用它，包括加载的 SWF 文件的时间轴。

可以将编写完善的函数看作一个“黑匣子”。如果它的输入、输出和目的都有详细的注释，则该函数的用户就不需要确切地了解该函数的内部工作原理。

定义函数

函数和变量一样，都附加在定义它们的影片剪辑的时间轴上，必须使用目标路径才能调用它们。与处理变量一样，您可以使用 `_global` 标识符声明一个全局函数，该函数无需使用目标路径即可从所有时间轴中进行调用。若要定义全局函数，请在函数名称前面加上标识符 `_global`，如下例所示：

```
_global.myFunction = function (x) {
    return (x*2)+3;
}
```

若要定义时间轴函数，请使用 `function` 动作，后面要带有函数名称、要传递给该函数的所有参数以及指明该函数动作的动作脚本语句。

下面的示例是一个名为 `areaOfCircle` 的函数，它带有参数 `radius`：

```
function areaOfCircle(radius) {
    return Math.PI * radius * radius;
}
```

也可以通过创建函数文本来定义函数。函数文本是一种未命名的函数，它是在表达式中而不是在语句中声明的。可以使用函数文本定义函数，返回它的值，以及将该值赋予一个表达式中的变量，如下例所示：

```
area = (function() {return Math.PI * radius *radius;})(5);
```

当函数被重新定义时，新的定义将替换旧的定义。

将参数传递给函数

参数是函数对其执行代码的元素。（在本手册中，术语参数和参量是可以互换的。）例如，下面的函数接受参数 `initials` 和 `finalScore`：

```
function fillOutScorecard(initials, finalScore) {  
    scorecard.display = initials;  
    scorecard.score = finalScore;  
}
```

在调用函数时，必须将必需的参数传递给函数。函数会用传递的值替换函数定义中的参数。在本示例中，`scorecard` 是影片剪辑的实例名称；`display` 和 `score` 是实例中的输入文本字段。下面的函数调用会将值 "JEB" 赋予变量 `display`，并将值 45000 赋予变量 `score`：

```
fillOutScorecard("JEB", 45000);
```

函数 `fillOutScorecard()` 中的参数 `initials` 与本地变量类似，当调用该函数时，该参数会存在，而当该函数退出时，该参数即会中止退出。如果在函数调用时省略了参数，则省略的参数会以 `undefined` 类型传递。如果在调用函数时提供了函数声明所不需要的多余参数，多余的参数会被忽略。

在函数中使用变量

本地变量是组织代码并使代码易于理解的重要工具。如果函数使用本地变量，则可以在 SWF 文件的其它所有脚本中隐藏其变量；本地变量的范围限定为函数体，在该函数退出后，本地变量也将不再存在。传递给函数的任何参数也作为本地变量来处理。

您还可以在函数中使用全局变量和常规变量。但是，如果要修改全局变量或常规变量，最好使用脚本注释记录这些修改。

从函数中返回值

使用 `return` 语句可以从函数中返回值。`return` 语句会使函数停止运行，然后将其替换为 `return` 动作的值。在函数中使用 `return` 语句时应遵守以下规则：

- 如果为函数指定除 `void` 之外的其它返回类型，则必须在函数中加入一条 `return` 语句。
- 如果指定返回类型为 `void`，则不应加入 `return` 语句。
- 如果不指定返回类型，则可以选择是否加入 `return` 语句。如果不加入该语句，将返回一个空字符串。

例如，下面的函数返回参数 `x` 的平方，并且指定了返回值的类型必须为 `Number`：

```
function sqr(x):Number {  
    return x * x;  
}
```

某些函数会执行一系列的任务，但不返回值。例如，下面的函数会初始化一系列全局变量：

```
function initialize() {  
    boat_x = _global.boat._x;  
    boat_y = _global.boat._y;
```

```
    car_x = _global.car._x;  
    car_y = _global.car._y;  
}
```

调用用户定义的函数

可以使用目标路径从任何时间轴调用任何时间轴中的函数，包括从加载的 SWF 文件的时间轴中进行调用。如果函数是使用 `_global` 标识符声明的，则无需使用目标路径即可调用它。

若要调用函数，请输入函数名称的目标路径，如有必要，在括号内传递所有必需的参数。例如，下面的语句将在主时间轴上调用影片剪辑 `MathLib` 中的函数 `sqr()`，向其传递参数 `3`，然后将结果存储在变量 `temp` 中：

```
var temp = _root.MathLib.sqr(3);
```

下面的示例使用绝对路径调用 `initialize()` 函数，该函数是在主时间轴上定义的，而且不需要参数：

```
_root.initialize();
```

下面的示例使用相对路径调用 `list()` 函数，该函数是在 `functionsClip` 影片剪辑中定义的：

```
_parent.functionsClip.list(6);
```


第 3 章

编写和调试脚本

在 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中，您可以将编写的脚本嵌入到 FLA 文件中，也可以在计算机上存储为外部文件。（如果您正在编写动作脚本 2.0 的类文件，则必须将每一个类存储成与该类同名的外部文件。）要编写嵌入的脚本，请使用“动作”面板并将该动作附加到某一按钮或影片剪辑，或者附加到时间轴中的一个帧（请参见第 49 页的“运行动作脚本时进行控制”）。要编写外部脚本文件，可以使用任何文本编辑器或代码编辑器。在 Flash Professional 中，您还可以使用内置的“脚本”窗口。有关更多信息，请参见第 51 页的“使用“动作”面板和“脚本”窗口”。

在使用动作脚本编辑器时，还可以检查语法错误、自动设置代码格式并用代码提示来帮助您完成语法。此外，标点符号平衡功能可帮助您匹配小括号、大括号或中括号。有关更多信息，请参见第 54 页的“使用动作脚本编辑器”。

当处理文档时，要经常测试它，从而确保它能够尽可能流畅地正常播放。您可以使用“带宽设置”模拟文档在不同连接速度下的情况（请参见“使用 Flash”帮助中的“测试文档下载性能”）。要测试脚本，您可以使用 Flash Player 的专用调试版本来帮助您排查故障。如果在动作脚本中使用合适的创作技术，当出现异常情况时，就会更容易地排除脚本的故障。有关更多信息，请参见第 61 页的“调试脚本”。

运行动作脚本时进行控制

当您撰写脚本时，可以使用“动作”面板将脚本附加到时间轴上的一个帧，或附加到舞台上的一个按钮或影片剪辑。当播放头进入该帧时，附加到该帧的脚本会运行或执行。但是，附加到 SWF 文件的第一帧的脚本的行为可能与附加到后续帧的脚本的行为不同，这是因为 SWF 文件中的第一帧是增量播放的，即对象一边下载到 Flash Player 一边在舞台上绘制，这可能会在动作执行时造成影响。第一帧之后的所有帧都是在帧的每个对象都可以使用之后立即播放的。

附加到影片剪辑或按钮上的脚本在事件发生时执行。事件是 SWF 文件中发生的事情，例如移动鼠标、按下键盘键或加载影片剪辑。您可以使用动作脚本确定事件何时发生并根据事件执行特定的脚本。有关更多信息，请参见第 75 页的第 4 章“处理事件”。

若要根据条件是否成立来执行动作或重复动作，可以使用 `if`、`else`、`else if`、`for`、`while`、`do while`、`for..in` 或 `switch` 语句，在本节的后面将简要介绍这些语句。

检查条件

用于检查一个条件是 `true` 还是 `false` 的语句以术语 `if` 开头。如果条件成立，动作脚本将执行随后的语句。如果条件不成立，动作脚本将跳到此代码块外的下一条语句。

要优化代码的性能，应首先检查最有可能的条件。

下列语句将测试三个条件。术语 `else if` 指定在前面的条件为 `false` 时要执行的替代测试。

```
if (password == null || email == null) {
    gotoAndStop("reject");
} else if (password == userID){
    gotoAndPlay("startMovie");
}
```

如果要检查是否满足若干条件中的一个条件，则可以使用 `switch` 语句，而不必使用多个 `else if` 语句。

重复动作

动作脚本可以将一个动作重复指定的次数，或是在特定的条件成立时重复动作。使用 `while`、`do..while`、`for` 和 `for..in` 动作可以创建循环。

在条件成立时重复动作：

- 使用 `while` 语句。

`While` 循环计算一个表达式的值，如果表达式为 `true`，则会执行循环体中的代码。当循环体中的每个语句都执行完毕后，会再次评估该表达式。在下面的示例中，循环将执行四次：

```
i = 4;
while (var i > 0) {
    my_mc.duplicateMovieClip("newMC" + i, i );
    i--;
}
```

可以使用 `do..while` 语句创建与 `while` 循环同类的循环。`Do..while` 循环中是在代码块结束时计算表达式的值，因此该循环总是至少执行一次，如下例所示：

```
i = 4;
do {
    my_mc.duplicateMovieClip("newMC" + i, i );
    i--;
} while (var i > 0);
```

使用内置计数器重复动作：

- 使用 `for` 语句。

多数循环都会使用某种计数器，以控制循环执行的次数。每执行一次循环就称为一次迭代。可以声明一个变量并编写一条语句，每执行一次循环，该语句都会增加或减小该变量。在 `for` 动作中，计数器和递增计数器的语句都是该动作的一部分。在下面的示例中，第一个表达式 (`var i = 4`) 是在第一次迭代之前计算的初始表达式。第二个表达式 (`i > 0`) 是每次运行循环之前检查的条件。第三个表达式 (`i--`) 称为后表达式，每次运行循环之后会计算该表达式。

```
for (var i = 4; i > 0; i--){
    myMC.duplicateMovieClip("newMC" + i, i + 10);
}
```

遍历影片剪辑或对象的子级：

- 使用 `for..in` 语句。

子级包括其它影片剪辑、函数、对象和变量。下面的示例使用 `trace` 语句在“输出”面板中显示其结果：

```
myObject = { name:'Joe', age:25, city:'San Francisco' };
for (propertyName in myObject) {
```

```

        trace("myObject has the property:" + propertyName + ", with the value:" +
myObject[propertyName]);
    }

```

本示例将在“输出”面板中生成如下结果：

```

myObject has the property:name, with the value:Joe
myObject has the property:age, with the value: 25
myObject has the property:city, with the value:San Francisco

```

您可能想让脚本重复特定的子级类型，例如只重复影片剪辑子级。将 `for..in` 与 `typeof` 运算符配合使用即可实现此目的。

```

for (name in myMovieClip) {
    if (typeof (myMovieClip[name]) == "movieclip") {
        trace("I have a movie clip child named " + name);
    }
}

```

有关各个动作的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中相应的各个条目。

使用“动作”面板和“脚本”窗口

您可以将 Flash 脚本嵌入到 FLA 文件中，也可以将它们存储为外部文件。最好尽量将动作脚本代码存储在外部文件中。这样便于在多个 FLA 文件中重用代码。然后，在 FLA 文件中，创建一个脚本，使用 `#include` 语句来访问存储在外部的代码。使用 `.as` 后缀将您的脚本标识为动作脚本 (AS) 文件。（如果要编写自定义类文件，则必须将这些类文件存储为外部 AS 文件。）

注意：发布、导出、测试或调试 FLA 文件时，外部文件中的动作脚本代码将被编译成 SWF 文件。因此，如果对外部文件进行了任何更改，则必须保存该文件，并重新编译使用它的任何 FLA 文件。

在您将动作脚本代码嵌入到 FLA 文件中时，可以将代码附加到帧和对象。尽可能尝试将嵌入的动作脚本附加到时间轴的第一帧。这样，您就不必搜索 FLA 文件来找到所有代码，代码将集中放置于一个位置上。创建一个名为“动作”的层并将代码放置于该层上。这样，即使您确实将代码放置于其它帧上或将代码附加到对象，也只需在一层上进行查找就可以找到所有代码。

要创建作为您的文档的一部分的脚本，请将动作脚本直接输入到“动作”面板中。要创建外部脚本，请使用您喜爱的文本编辑器；或者，在 Flash Professional 中，可以使用“脚本”窗口。在使用“动作”面板或“脚本”窗口时，您使用相同的动作脚本编辑器，在面板或窗口右侧的“脚本”窗格中键入代码。为了减少不得不做的键入工作量，您还可以从“动作”工具箱中将动作选到或拖到“脚本”窗格中。

要显示“动作”面板，请执行以下操作之一：

- 选择“窗口” > “开发面板” > “动作”。
- 按下 F9 键。

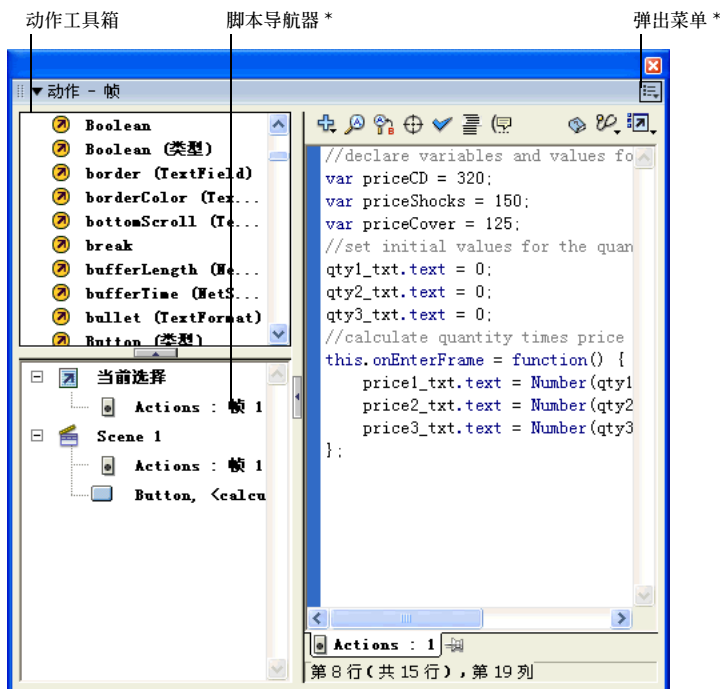
（仅限 Flash Professional）要显示“脚本”窗口，请执行以下操作之一：

- 要开始撰写新脚本，请选择“文件” > “新建” > “动作脚本文件”。
- 要打开现有脚本，请选择“文件” > “打开”，然后打开现有 AS 文件。
- 要编辑已打开的脚本，请单击显示该脚本的名称的文档选项卡。（只有在 Microsoft Windows 中才支持文档选项卡。）

关于动作脚本编辑器环境

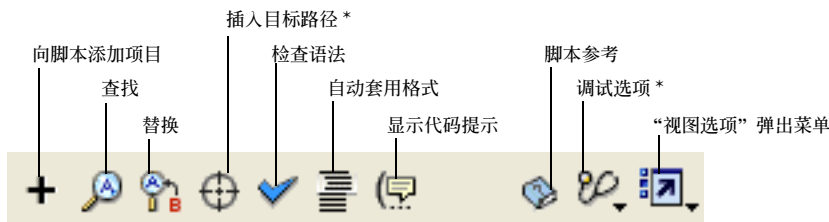
动作脚本编辑器环境由两部分组成。右侧部分是“脚本”窗格，这是键入代码的区域。左侧部分是一个“动作”工具箱，每个动作脚本语言元素在该工具箱中都有一个对应的条目。

在“动作”面板中，“动作”工具箱还包含一个脚本导航器，脚本导航器是 FLA 文件中具有关联动作脚本的位置的可视化表示形式；您可以在这里浏览 FLA 文件以查找动作脚本代码。如果您单击脚本导航器中的某一项目，则与该项目关联的脚本将出现在“脚本”窗格中，并且播放头将移到时间轴上的该位置。如果您双击脚本导航器中的某一项，则该脚本会被固定（请参见第 53 页的“管理 FLA 文件中的脚本”）。



* 仅限“动作”面板

“脚本”窗格的上方还有若干按钮：



* 仅限“动作”面板

您可以直接在“脚本”窗格中编辑动作、输入动作参数或删除动作。您还可以双击“动作”工具箱中的某一项或“脚本”窗格上方的“添加” (+) 按钮，向“脚本”窗格添加动作。

管理 FLA 文件中的脚本

如果您没有将所有代码集中放置于 FLA 文件中的一个位置内，则可以在“动作”面板中固定（就地锁定）多个脚本，以更易于在脚本之中移动。在下图中，与时间轴上当前位置关联的脚本位于名为“Cleanup”层的第 1 帧上。（最左侧的标签始终在时间轴上您的位置后。）该脚本也被固定（作为最右侧的标签显示）。其它两个脚本被固定；一个脚本位于名为“Intro”的层的第 1 帧上，另一个脚本则位于第 15 帧上。您可以通过单击这些标签或通过使用快捷键，在多个固定的脚本之中移动。在固定的脚本之中移动并不更改您在时间轴上的当前位置。



提示：如果“脚本”窗格中显示的内容没有进行相应更改，以反映您在时间轴上选择的位置，则“脚本”窗格可能正显示固定的脚本。单击“脚本”窗格左下角上最左侧的标签可以显示与时间轴上的位置相关联的动作脚本。

固定脚本：

- 1 确定您的指针在时间轴上的位置，以使脚本出现在“动作”面板中“脚本”窗格左下角的标签内。
- 2 执行以下其中一项操作：

- 单击该标签右侧的图钉图标。（如果图钉图标的外观与最左侧上的图标相似，则该脚本已被固定；单击该图标可以取消对该脚本的固定。）
- 右键单击 (Windows) 或按住 Control 键并单击 (Macintosh) 该标签，然后选择“固定脚本”。
- 从“选项”弹出菜单（位于面板的右上角）中选择“固定脚本”。

取消固定一个或多个脚本：

- 执行以下其中一项操作：
- 如果被固定的脚本出现在“动作”面板中“脚本”窗格左下角的标签内，则单击该标签右侧的图钉图标。（如果该图钉图标在外观上与最左侧的图标类似，则该脚本已被取消固定；单击该图标会固定该脚本。）
- 右键单击 (Windows) 或者按住 Control 键并单击 (Macintosh) 一个标签，然后选择“关闭脚本”或“关闭所有脚本”。
- 从“选项”弹出菜单（位于面板的右上角）中选择“关闭脚本”或“关闭所有脚本”。

将快捷键用于固定的脚本：

- 您可以将以下快捷键用于固定的脚本：

动作	Windows 快捷键	Macintosh 快捷键
固定脚本	Control+= （等号）	Command+=
取消固定脚本	Control-- （减号）	Command-=
将焦点移到右侧的标签	Control-Shift., （句点）	Command-Shift.,
将焦点移到左侧的标签	Control-Shift-, （逗号）	Command-Shift-,
取消对所有脚本的固定	Control-Shift-- （减号）	Command-Shift--

使用动作脚本编辑器

Flash MX 2004 和 Flash MX Professional 2004 提供若干工具来帮助您编写语法正确的代码，并且允许您为代码格式和其它选项设置首选参数。本部分将讨论这些功能。

语法突出显示

在动作脚本中，就像在任何语言中一样，语法 是将元素组合在一起产生意义的方式。如果使用了错误的动作脚本语法，脚本将不会运行。

当您在 Flash MX 2004 和 Flash MX Professional 2004 中撰写脚本时，“动作”工具箱中将以黄色显示您的目标播放器版本不支持的命令。例如，如果 Flash Player SWF 版本设置为 Flash 6，仅受 Flash Player 7 支持的动作脚本就在“动作”工具箱中显示为黄色。（有关设置 Flash Player SWF 版本的信息，请参见“使用 Flash”帮助中的“为 Flash SWF 文件格式设置发布选项”。）

您还可以在编写脚本时设置首选参数，使 Flash 能够在您编写脚本时对您的一部分脚本进行“颜色编码”，让您注意到键入错误。例如，假设您设置语法颜色首选参数，以深绿色显示关键字。在您键入代码时，如果键入 `var`，则单词 `var` 以绿色显示。但是，如果您错误地键入了 `vae`，则单词 `vae` 将保持为黑色，使您立即就能注意到键入的单词有误。

要设置键入时的语法颜色首选参数，请执行以下操作之一：

- 选择“编辑”>“首选参数”，并在“动作脚本”选项卡上指定语法颜色设置。
- 在“动作”面板中，从“选项”弹出菜单（位于面板的右上角）中选择“首选参数”，然后在“动作脚本”选项卡上指定语法颜色设置。

编写触发代码提示的代码

使用动作脚本编辑器（在“动作”面板或“脚本”窗口中）时，Flash 可以检测到正在输入的动作并显示代码提示，即包含该动作完整语法的工具提示，或列出可能的方法或属性名称的弹出菜单。当您精确键入或命名对象时，会出现参数、属性和事件的代码提示，这样，动作脚本编辑器就会知道要显示哪些代码提示，此问题将在本部分下文中讨论。有关在代码提示出现时如何使用它们的信息，请参见第 56 页的“使用代码提示”。

注意：代码提示将为那些不要求您为其创建和命名一个对象的本地类（例如 `Math`、`Key`、`Mouse` 等）自动启用。

严格指定对象类型以触发代码提示

在您使用动作脚本 2.0 时，可以精确键入基于内置类（例如 `Button`、`Array` 等）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，假设您键入以下代码：

```
var names:Array = new Array();  
names.
```

只要您键入句点（.），Flash 就会显示可用于 `Array` 对象的方法和属性的列表，因为您已经将该变量的类型指定为数组。有关数据类型指定的更多信息，请参见第 33 页的“严格数据类型指定”。有关在代码提示出现时如何使用它们的信息，请参见第 56 页的“使用代码提示”。

使用后缀触发代码提示

如果您使用的是动作脚本 1，或者在创建对象时未严格指定类型（请参见第 54 页的“严格指定对象类型以触发代码提示”），而又想显示这些对象的代码提示，则必须在创建每个对象时在其名称后添加特殊后缀。例如，触发 Array 类和 Camera 类的代码提示的后缀分别是 `_array` 和 `_cam`。如果您键入以下代码：

```
var my_array = new Array();
var my_cam = Camera.get();
```

然后，键入以下两项中的任何一项（变量名称后面跟有句点），就会分别显示 Array 和 Camera 对象的代码提示。

```
my_array.
my_cam.
```

对于在舞台上出现的对象，请使用属性检查器的“实例名称”文本框中的后缀。例如，要显示 MovieClip 对象的代码提示，请使用属性检查器为所有 MovieClip 对象指定带有 `_mc` 后缀的实例名称。然后，只要您键入实例名称和紧随的句点，就会显示代码提示。

尽管在精确键入对象时不需要后缀来触发代码提示，但一贯地使用它们可以帮助您和其他人理解您的脚本。

下表列出了支持自动代码提示所需的后缀：

对象类型	变量后缀
Array	<code>_array</code>
按钮	<code>_btn</code>
摄像头	<code>_cam</code>
Color	<code>_color</code>
ContextMenu	<code>_cm</code>
ContextMenuItems	<code>_cmi</code>
日期	<code>_date</code>
Error	<code>_err</code>
LoadVars	<code>_lv</code>
LocalConnection	<code>_lc</code>
麦克风	<code>_mic</code>
MovieClip	<code>_mc</code>
MovieClipLoader	<code>_mcl</code>
PrintJob	<code>_pj</code>
NetConnection	<code>_nc</code>
NetStream	<code>_ns</code>
SharedObject	<code>_so</code>
Sound	<code>_sound</code>
字符串	<code>_str</code>

对象类型	变量后缀
TextField	_txt
TextFormat	_fmt
Video	_video
XML	_xml
XMLNode	_xmlnode
XMLSocket	_xmlsocket

有关在代码提示出现时如何使用它们的信息，请参见第 56 页的“使用代码提示”。

使用注释触发代码提示

您也可以使用动作脚本注释来指定对象的类，以用于代码提示。下面的示例将告诉动作脚本，实例 `theObject` 的类是 `Object`，依此类推。如果在这些注释后输入 `mc`，然后再输入一个句点，代码提示将显示 `MovieClip` 的方法和属性列表；如果输入 `theArray`，然后再输入一个句点，代码提示将显示 `Array` 的方法和属性列表，依此类推。


```
// 对象 theObject;
// 数组 theArray;
// 影片剪辑 mc;
```

不过，Macromedia 建议使用严格数据类型指定（请参见第 54 页的“严格指定对象类型以触发代码提示”）或使用后缀（请参见第 55 页的“使用后缀触发代码提示”），而不建议使用此技术，因为前面的两种技术自动启用代码提示，并使代码更易懂。

使用代码提示

默认情况下，启用代码提示。通过设置首选参数，可以禁用代码提示或确定它们出现的速度。如果在首选参数中禁用了代码提示，则仍可为特定命令显示代码提示。

要为自动代码提示指定设置，请执行以下操作之一：

- 选择“编辑” > “首选参数”，然后在“动作脚本”选项卡中启用或禁用“代码提示”。
-  • 在“动作”面板中，从“选项”弹出菜单（位于面板的右上角）中选择“首选参数”，然后在“动作脚本”选项卡上启用或禁用“代码提示”。

如果您启用代码提示，则还可以以秒为单位指定代码提示显示前的延迟。例如，如果您不熟悉动作脚本，则最好选择无延迟，这样，代码提示将始终立即显示。不过，如果您通常知道要键入的内容，只在使用不熟悉的语言元素时才需要代码提示，则可以指定一个延迟，这样，在您不想使用代码提示时这些代码提示不会出现。

使用工具提示样式的代码提示：

- 1 在需要括号的元素（例如方法名称，`if` 或 `do while` 之类的命令等）后键入一个左括号 `[` 以显示代码提示。

代码提示就出现了。

```
if {  
    if { condition } {  
    }  
}  
  
my_array.splice(  
    Array.splice( index, count, elem1, ..., elemN )  
)
```

注意：如果代码提示不出现，请检查您是否在“动作脚本”选项卡上禁用了代码提示。如果您希望为已创建的变量或对象显示代码提示，请确保您正确命名了变量或对象（请参见第 55 页的“使用后缀触发代码提示”），或者精确键入了变量或对象（请参见第 54 页的“严格指定对象类型以触发代码提示”）。

- 2 输入参数的值。如果有多个参数，则用逗号分隔这些值。

像 gotoAndPlay() 或 for 之类的重载命令（即，可以用不同参数集调用的函数或方法）会显示一个指示器，让您选择要设置的参数。单击小箭头按钮或者按下 Control+ 左箭头组合键和 Control+ 右箭头组合键，可以选择参数。

```
for {  
    1 of 2 for { init; condition; next } {  
    }  
}
```

- 3 要使代码提示消失，请执行以下操作之一：

- 键入右括号 [)]。
- 单击该语句之外的地方。
- 按下 Escape 键。

使用菜单样式的代码提示：

- 1 通过在变量或者对象名称后键入句点来显示代码提示。

代码提示菜单就出现了。

```
my_mc.  
_alpha  
_currentframe  
_droptarget  
_focusrect  
_framesloaded  
_height  
_lockroot  
_name
```

注意：如果代码提示不出现，请检查您是否在“动作脚本”选项卡上禁用了代码提示。如果您希望为已创建的变量或对象显示代码提示，请确保您正确命名了变量或对象（请参见第 55 页的“使用后缀触发代码提示”），或者精确键入了变量或对象（请参见第 54 页的“严格指定对象类型以触发代码提示”）。

- 2 要导航代码提示，可用向上和向下箭头键。
- 3 要选择菜单中的某项，请按下 Return 键或 Tab 键，或者双击该项。
- 4 要使代码提示消失，请执行以下操作之一：
 - 选择一个菜单项。
 - 单击该语句之外的地方。
 - 如果已经键入了一个左括号，请键入一个右括号 [)]。
 - 按下 Escape 键。

手动显示代码提示：

1 在可以显示代码提示的代码位置单击。下面是一些示例：

- 在语句或命令之后的点的后面（必须在这里输入属性或方法）
- 在方法名称中的括号之间

2 执行以下其中一项操作：



- 单击“脚本”窗格上方的“显示代码提示”按钮。
- 按下 Control+ 空格键 (Windows) 或者 Command+ 空格键 (Macintosh)。
- 如果您正使用“动作”面板，则打开弹出菜单（在标题栏的右侧），然后选择“显示代码提示”。

使用 Escape 快捷键

您可以通过使用快捷键（按下 Escape 键，再按下两个别的键），向脚本添加许多元素。（这些快捷方式不同于启动某些菜单命令的快捷键。）例如，如果您正在使用“脚本”窗格并键入 Escape+d+o，则以下代码将出现在脚本中，插入点就位于单词 `while` 之后，您可以在此处开始键入条件：

```
do {  
} while ();
```

同样，如果您键入 Escape+c+h，则会将以下代码放置在脚本中，插入点就放置在括号之间，您可以在此处开始键入条件：

```
catch () {  
}
```

如果您要了解哪些命令具有 Escape 快捷键或想得到相关提示，则可以在“动作”面板中的元素旁显示它们。



显示或隐藏 Escape 快捷键：



- 在“视图选项”弹出菜单中，启用或禁用“查看 Escape 快捷键”。

检查语法和标点

要彻底弄清您编写的代码是否能像预期的那样运行，需要发布或测试文件。不过，您可以不必退出 FLA 文件就迅速检查动作脚本代码。语法错误列在“输出”面板中。（在检查语法时，只检查当前的脚本；不检查可能位于 FLA 文件中的其它脚本。）您还可以检查代码块两边的小括号、大括号或中括号（数组访问运算符）是否齐全。

要进行语法检查，请执行以下操作之一：



- 单击“脚本”窗格上方的“检查语法”按钮。
- 在“动作”面板中，显示弹出菜单（位于面板的右上角）并选择“检查语法”。
- 按下 Control+T 组合键 (Windows) 或 Command+T 组合键 (Macintosh)。

检查标点平衡：

- 1 在脚本中的大括号 ({})、数组访问运算符 ([]) 或小括号 (()) 之间单击。
- 2 按下 Control+' 组合键 (Windows) 或 Command+' 组合键 (Macintosh) 来突出显示大括号、中括号或小括号之间的文本。

突出显示有助于检查开始标点符号是否有正确与之对应的结束标点符号。

设置代码格式

您可以指定设置以确定是自动还是手动设置代码格式以及代码的缩进。您还可以选择是否查看行号以及是否让长代码行换行。此外，您还可以选择是否使用动态字体映射。

设置格式选项：

- 1 执行以下操作之一：



- 在“动作”面板中，从“选项”弹出菜单（位于面板的右上角）中选择“自动套用格式选项”。
- （仅限 Flash Professional），在外部脚本文件中，选择“编辑”>“自动套用格式选项”。此时出现“自动套用格式选项”对话框。

- 2 选择任意复选框。要查看每个选择的效果，请看“预览”窗格。

在设置“自动套用格式选项”后，所作设置将自动应用于您编写的代码，但不应用于现有代码。要将所作设置应用于现有代码，您必须手动操作。对于以前已使用不同设置设置了其格式的代码和以前从其它编辑器导入的代码等，您可以使用此过程设置其格式。

要根据“自动套用格式选项”设置来设置代码格式，请执行以下操作之一：



- 单击“脚本”窗格上方的“自动套用格式”按钮。
- 从“动作”面板弹出菜单中选择“自动套用格式”。
- 按下 Control+Shift+F 组合键 (Windows) 或 Command+Shift+F 组合键 (Macintosh)。

使用动态字体映射：

- 由于动态字体映射会在进行脚本撰写时增加运行时间，默认情况下它是关闭的。要打开它，请选择动作脚本首选参数中的“使用动态字体映射”。

例如，如果您正在处理多语言文本，则应打开动态字体映射。

使用自动缩进：

- 在默认情况下自动缩进功能是打开的。要关闭自动缩进，在动作脚本首选参数中取消选择“自动缩进”。

如果打开了自动缩进，在（或{之后键入的文本将按照动作脚本首选参数中的“制表符大小”设置自动缩进。要缩进其它行，请选择该行，然后按 Tab 键。要取消缩进，请按 Shift+Tab 组合键。

启用或禁用行号和自动换行：



- 在“视图选项”弹出菜单中，启用或禁用“查看行号”和“自动换行”。

调试脚本

Flash 提供了几个用于测试 SWF 文件中的动作脚本的工具。本部分下文中介绍的调试器可让您在 SWF 文件在 Flash Player 中运行时查找该 SWF 文件中出现的错误。Flash 还提供以下这些调试工具：

- “输出”面板，可显示错误消息以及变量和对象列表（请参见第 68 页的“使用“输出”面板”）
- trace 语句，可将编程注释和表达式的值发送到“输出”面板（请参见第 70 页的“使用 trace 语句”）
- throw 和 try..catch..finally 语句，可用于从脚本内测试和响应运行时错误
- 提供全面的编译器错误消息，帮助您更快地诊断和解决问题（请参见第 739 页的附录 A “错误消息”）

您必须在 Flash Player 的称作 Flash 调试播放器的特殊版本中查看 SWF 文件。在您安装创作工具时，将自动安装 Flash 调试播放器。因此，如果您安装 Flash 并浏览具有 Flash 内容的 Web 站点，或者执行“测试影片”，则您使用的正是 Flash 调试播放器。您还可以运行 <app_dir>\Players\Debug\ 目录中的安装程序，或者从同一目录中启动独立的 Flash 调试播放器。

在您使用“测试影片”命令测试实现键盘控制（Tab 键切换或使用 Key.addListener() 创建的快捷键等）的影片时，选择“控制”>“禁用快捷键”。选择此选项可以避免创作环境“抢占”键击动作，使这些动作可以传到播放器。例如，在创作环境中，Control+U 组合键将打开“首选参数”对话框。如果您的脚本将 Control+U 组合键分配给为屏幕上的文本加下划线的动作，则在您使用“测试影片”时，按下 Control+U 组合键将打开“首选参数”对话框，而不是运行给文本加下划线的动作。若要令 Control+U 命令传到播放器，您必须选择“控制”>“禁用快捷键”。

小心：如果 SWF 文件路径的任何部分具有不能使用 MBCS 编码方案表示的字符，则“测试影片”命令将失败。例如，在英文系统上使用日文路径将不起作用。使用外部播放器的应用程序的所有方面都受到此限制的约束。

调试器显示了一个当前加载到 Flash Player 中的影片剪辑的分层显示列表。使用调试器，您可在 SWF 文件播放时显示和修改变量和属性的值，并且可以使用断点停止 SWF 文件并逐行跟踪动作脚本代码。

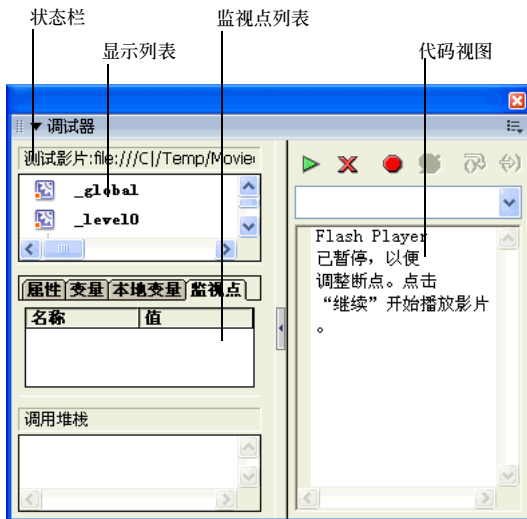
您可以在测试模式下对本地文件使用调试器，或使用调试器测试远程位置的 Web 服务器上的文件。调试器用于在动作脚本中设置断点，断点会在运行时停止 Flash Player，并跟踪代码。然后可以回到脚本中，对它们进行编辑，以便它们产生正确的结果。

在文件被激活后，调试器的状态栏就会显示文件的 URL 或本地路径，表明文件是运行在测试模式下还是从远程位置运行，并且显示影片剪辑显示列表的动态视图。向文件添加影片剪辑或从文件删除影片剪辑时，显示列表会立即反映出这些更改。通过移动水平拆分器，可以调整显示列表的大小。

在测试模式下激活调试器：

- 选择“控制” > “调试影片”。

这样会打开调试器。同时也在测试模式下打开 SWF 文件。



从远程位置调试 SWF 文件

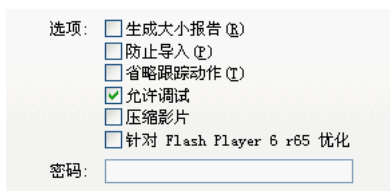
通过使用 Flash Player 的独立版本、ActiveX 版本或者插件版本，可以调试远程 SWF 文件。在导出 SWF 文件时，您可以在文件中启用调试，并且创建调试密码。如果您不启用调试，调试器将不会激活。

要确保只有可信的用户才能在 Flash 调试播放器中运行 SWF 文件，在发布文件时可以使用调试密码。与在 JavaScript 或者 HTML 中一样，用户也可以查看动作脚本中的客户端变量。要安全地存储变量，您必须把它们发送到服务器端应用程序，而不要把它们存储在文件中。然而，作为 Flash 开发人员，您可能有其它的一些不想泄漏出去的商业机密，比如影片剪辑结构。您可以使用调试密码来保护您的工作。

当您导出、发布或者测试影片的时候，Flash 会创建一个包含调试信息的 SWD 文件。要进行远程调试，您必须把 SWD 文件放到服务器上 SWF 文件所在的目录中。

启用 Flash 影片的远程调试：

- 1 选择“文件” > “发布设置”。
- 2 在“发布设置”对话框的“Flash”选项卡上，选择“允许调试”。



- 3 要设置密码，在“密码”框中输入密码。

设置了此密码后，任何人都必须使用该密码才能将信息下载到调试器中。不过，如果将“密码”框留空，则不需要密码。

- 4 关闭“发布设置”对话框，然后选择以下命令之一：

- “控制” > “调试影片”
- “文件” > “导出影片”
- “文件” > “发布设置” > “发布”

Flash 创建以 .swd 为扩展名的调试文件，并把它与 SWF 文件一起保存。SWD 文件包含了允许您使用断点和跟踪代码的信息。

- 5 把 SWD 文件放到服务器上与 SWF 文件相同的目录中。

如果 SWD 文件和 SWF 文件不在同一个目录中，您仍然可以进行远程调试，但是调试器将忽略断点，这样您就不能跟踪代码。

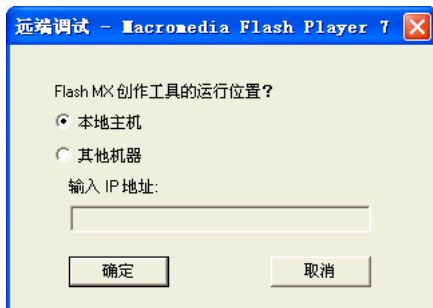
- 6 在 Flash 中，选择“窗口” > “开发面板” > “调试器”。



- 在调试器中，从“选项”弹出菜单（位于面板的右上角）选择“启用远程调试”。

从远程位置激活调试器：

- 1 打开 Flash 创作应用程序。
- 2 在浏览器中或在独立播放器中，从远程位置打开已发布的 SWF 文件。
此时出现“远程调试”对话框。



如果没有出现该对话框，Flash 就不能查找到该 SWD 文件。在这种情况下，在 SWF 文件中以右键单击 (Windows) 或者按住 Control 键并单击 (Macintosh)，以显示上下文菜单，然后选择“调试器”。



- 3 在“远程调试”对话框中选择“本地主机”或“其它机器”：
- 如果 Flash 调试播放器和 Flash 创作应用程序在同一台计算机上，请选择“本地主机”。
 - 如果调试播放器和 Flash 创作应用程序不在同一台计算机上，请选择“其它机器”。输入运行 Flash 创作应用程序的计算机的 IP 地址。
- 4 当连接建立时，就出现一个密码提示。如果设置了调试密码，请输入它。
- 在调试器中会出现 SWF 文件的显示列表。

显示和修改变量

调试器中的“变量”选项卡显示 SWF 文件中的所有全局变量和时间轴变量的名称和值。如果改变了“变量”选项卡上的变量值，当 SWF 文件运行时，您就能看到在 SWF 文件中反映的变化。例如，要测试游戏中的冲突检测，可以输入变量值以便把球定位到墙旁边的正确位置。

调试器的“本地变量”选项卡会显示 SWF 文件在断点处或在用户定义函数内任何位置停止时可用的所有本地变量的名称和值。

显示变量：

- 1 从显示列表中选择包含该变量的影片剪辑。
若要显示全局变量，请选择显示列表中的 `_global` 剪辑。
- 2 单击“变量”选项卡。

在 SWF 文件播放时，显示列表会自动更新。如果在特定的帧从 SWF 文件中删除一段影片剪辑，该影片剪辑以及它的变量、变量名也会从调试器的显示列表中删除。但是，如果将某个变量标记为加入“监视点”列表（请参见第 65 页的“使用“监视点”列表”），则该变量就不会被删除。



修改变量值：

- 双击该值，然后输入新的值。

该值不能是表达式。例如，可以使用 "Hello"、3523 或 "http://www.macromedia.com"，但是不能使用 `x + 2` 或 `eval("name:"+i)`。该值可以是字符串（任何用引号括起的值）、数字或布尔值（true 或 false）。

注意：若要在测试模式下将表达式的值显示到“输出”面板上，请使用 `trace` 语句。请参见第 70 页的“使用 `trace` 语句”。

使用“监视点”列表

要以可管理的方式监视一组关键变量，可以标记这些变量，使之在“监视点”列表中出现。“监视点”列表显示了变量和它的值的绝对路径。和在“变量”选项卡中的方式一样，您也可以可以在“监视点”列表中输入新的变量值。

如果您向“监视点”列表中加入了某个本地变量，只有当 Flash Player 停止在该变量所在范围的动作脚本的某一行时，才会显示该变量的值。所有其它变量会在 SWF 文件播放时显示。如果调试器查找不到该变量的值，则该变量的值将按“未定义”列出。

“监视点”列表只能显示变量，不能显示属性和函数。



为“监视点”列表标记的变量和在“监视点”列表中的变量

要向“监视点”列表中添加变量，请执行以下操作之一：

- 在“变量”或“本地变量”选项卡上以右键单击 (Windows) 或按住 Control 键并单击 (Macintosh) 一个选定的变量，然后从上下文菜单中选择“监视点”。该变量旁边会出现一个蓝点。
- 在“监视点”选项卡上以右键单击 (Windows) 或按住 Control 键并单击 (Macintosh)，然后从上下文菜单中选择“增加”。在字段中输入变量名的目标路径和变量值。

从“监视点”列表中删除变量：

- 在“监视点”选项卡上以右键单击 (Windows) 或者按住 Control 键并单击 (Macintosh)，然后从上下文菜单中选择“删除”。

显示影片剪辑属性和更改可编辑属性

调试器的“属性”选项卡显示舞台中任何影片剪辑的所有属性值。在 SWF 文件运行时，您可以改变一个值，然后查看它对 SWF 文件的影响。某些影片剪辑属性是只读的，不能更改。

在调试器中显示影片剪辑属性：

- 1 从显示列表中选择影片剪辑。

2 在调试器中单击“属性”选项卡。



修改属性值：

- 双击该值，然后输入新的值。

该值不能是表达式。例如，可以输入 50 或 "clearwater"，但是不能输入 $x + 50$ 。该值可以是字符串（任何用引号括起的值）、数字或布尔值（true 或 false）。不能在调试器中输入对象或数组值（例如 {id:"rogue"} 或 [1, 2, 3]）。

有关更多信息，请参见第 41 页的“字符串运算符”和第 39 页的“使用运算符处理表达式中的值”。

注意：若要在测试模式下将表达式的值显示到“输出”面板上，请使用 trace 语句。请参见第 70 页的“使用 trace 语句”。

设置和删除断点

通过使用断点，可以使正在 Flash Player 中运行的 SWF 文件在动作脚本的指定行上停止。您可以使用断点来测试代码中可能的错误点。例如，如果您编写了一组 if..else if 语句，但不能确定哪一个正在执行，则可以在语句前面添加断点，然后在调试器中逐个跟踪这些语句。

您可以在“动作”面板或者调试器中设置断点。（要在外部脚本中设置断点，必须使用调试器。）在“动作”面板中设置的断点会保存在 Flash 文档（FLA 文件）中。在调试器中设置的断点不会保存在 FLA 文件中，并且只在当前的调试会话中有效。



若要在“动作”面板中设置或删除断点，请执行以下操作之一：

- 在左边距中单击。红点指示断点。
- 单击“脚本”窗格上方的“调试选项”按钮。
- 右键单击 (Windows) 或按住 Control 键并单击 (Macintosh)，以显示上下文菜单，然后选择“断点”、“删除断点”或者“删除所有断点”。

- 按下 Control+Shift+B 组合键 (Windows) 或者 Command+Shift+B 组合键 (Macintosh)。

注意：在 Flash 的早期版本中，在“脚本”窗格的左边距处单击会选择该代码行；现在，执行这一操作将添加或删除断点。要选择某一代码行，请按住 Control 键并单击 (Windows) 或按住 Command 键并单击 (Macintosh)。

要在调试器中设置和删除断点，请执行以下操作之一：

- 在左边距中单击。红点指示断点。
 - 单击代码视图上方的“切换断点”或者“删除所有断点”按钮。
 - 右键单击 (Windows) 或按住 Control 键并单击 (Macintosh)，以显示上下文菜单，然后选择“断点”、“删除断点”或者“删除所有断点”。
 - 按下 Control+Shift+B 组合键 (Windows) 或者 Command+Shift+B 组合键 (Macintosh)。
- 一旦 Flash Player 停留在某个断点上，您可以跳入、跳过或者跳出那行代码。如果在“动作”面板中在注释行或者空行上设置断点，该断点就会被忽略。

跟踪代码行

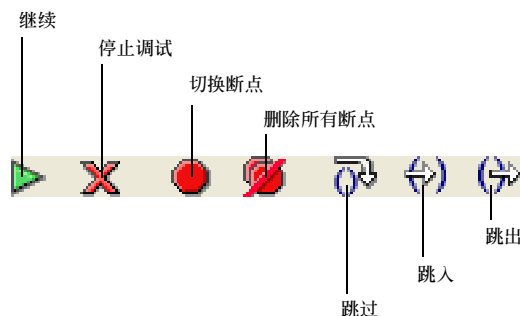
当开始调试会话时，Flash Player 会暂停。如果在“动作”面板中设置断点，只需单击“继续”按钮即可播放 SWF 文件，直到到达断点。例如，在下面的代码中，假定在一个按钮内的 myFunction() 一行上设置了断点：

```
on (press) {
    myFunction();
}
```

当单击按钮时，到达断点 Flash Player 就暂停。无论 myFunction() 函数被定义在文档中的什么位置，您都可以跳入并将调试器移到该函数的第一行。您也可以跟踪或者跳出函数。

如果没有在“动作”面板中设置断点，可以使用调试器中的跳转菜单来选择影片中的任意脚本。一旦选择了一个脚本，就可以向它添加断点。添加断点之后，必须单击“继续”按钮来播放影片。到达断点时，调试器会停止。

当您跟踪代码行时，在“监视点”列表中和“变量”、“本地变量”以及“属性”选项卡中的变量和属性值也跟着改变。沿着调试器代码视图左侧的黄色箭头表明调试器在该行停止。使用代码视图顶部的下列按钮：



跳入使调试器（由黄色箭头指示）进入函数。“跳入”只用于用户定义的函数。

在下面的例子中，如果在第 7 行放置一个断点，然后单击“跳入”，调试器将前进到第 2 行，再单击“跳入”会进入第 3 行。对那些其中没有用户定义函数的行单击“跳入”，调试器会跳过一个代码行。例如，如果在第 2 行上停止，然后选择“跳入”，调试器将前进到第 3 行，如下面的例子中所示：

```
1 function myFunction() {  
2   x = 0;  
3   y = 0;  
4 }  
5  
6 mover = 1;  
7 myFunction();  
8 mover = 0;
```

跳出使调试器跳出函数。只有当前在用户定义函数中停止时，此按钮才能起作用，它把黄色箭头移到调用该函数的那行后面的那一行。在上面的例子中，如果在第 3 行放置一个断点，然后单击“跳出”，调试器就会移动到第 8 行。在不属于用户定义函数的行上单击“跳出”与单击“继续”的作用一样。例如，如果在第 6 行停止，然后单击“跳出”，播放器会继续执行脚本，直到遇到一个断点。

跳过使调试器跳过一行代码。此按钮将黄色箭头移动到脚本的下一行，并且忽略任何用户定义函数。在上面的例子中，如果在第 7 行停止，然后单击“跳过”，将会直接进入第 8 行，函数 `myFunction()` 将被忽略。

继续离开播放器停止处的行并继续播放，直至到达一个断点。

停止调试使调试器处于非活动状态，但是继续在 Flash Player 中播放 SWF 文件。

使用“输出”面板

在测试模式下，“输出”面板可显示相应的信息来帮助您排查 SWF 文件中的故障。一些信息（例如语法错误）可以自动显示出来。通过使用“对象列表”和“变量列表”命令，可以显示其它信息。（请参见第 69 页的“列出 SWF 文件的对象”和第 69 页的“列出 SWF 文件的变量”。）

如果在脚本中使用 `trace` 语句，SWF 文件运行时，可以向“输出”面板发送特定的信息。这些信息可以包括 SWF 文件的状态说明或者表达式的值。（请参见第 70 页的“使用 `trace` 语句”。）

要显示“输出”面板，请选择“窗口” > “开发面板” > “输出”，或者按 F2 键。

注意：如果在脚本中有语法错误，则“输出”面板在您检查语法或测试 SWF 文件时自动出现。



若要处理“输出”面板上的内容，请使用右上角的“选项”弹出菜单。

✓ 自动换行	
复制 (C)	Ctrl+C
清除 (L)	
查找...	Ctrl+F
再次查找	F3
保存到文件 (S)...	
打印...	
过滤级别 (F)	▶
帮助	
最大化面板	
关闭面板	

列出 SWF 文件的对象

在测试模式下，“对象列表”命令会在分层结构列表中显示以下一些信息：级别、帧、对象类型（形状、影片剪辑或者按钮）、目标路径和影片剪辑、按钮以及文本字段的实例名称。这对查找正确的目标路径和实例名称特别有用。与调试器不同，该列表不会在 SWF 文件播放时自动更新；每次要向“输出”面板发送这些信息时，必须选择“对象列表”命令。

“对象列表”命令不会列出所有的动作脚本数据对象。在此上下文中，对象被看成舞台中的形状或者元件。

显示影片中的对象列表：

- 1 如果影片不是在测试模式下运行，请选择“控制” > “测试影片”。
- 2 选择“调试” > “对象列表”。

在“输出”面板中会显示舞台上当前所有对象的列表，如下面的例子中所示：

```
Level #0:Frame=1 Label="Scene_1"
  Button:Target="_level0.myButton"
    Shape:
  Movie Clip:Frame=1 Target="_level0.myMovieClip"
    Shape:
  Edit Text:Target="_level0.myTextField" Text="This is sample text."
```

列出 SWF 文件的变量

在测试模式下，“变量列表”命令会显示 SWF 文件中当前所有变量的列表。这对查找正确的变量目标路径和变量名称特别有用。与调试器不同，该列表不会在 SWF 文件播放时自动更新；每次要向“输出”面板发送这些信息时，必须选择“变量列表”命令。

“变量列表”命令还显示用 `_global` 标识符声明的全局变量。全局变量会显示在“变量列表”命令输出的最上部，列在标题为“全局变量”的部分中，而且每个全局变量都带有 `_global` 前缀。

此外，“变量列表”命令显示 `getter/setter` 属性（这是用 `Object.addProperty()` 方法创建并调用 `get` 或 `set` 方法的属性）。`getter/setter` 属性与其所属的对象中的任何其它属性一起显示。为使这些属性与普通变量相比更易于区别，`getter/setter` 属性的值以 `[getter/setter]` 字符串为前缀。为 `getter/setter` 属性显示的值通过计算该属性的 `get` 函数确定。

显示 SWF 文件中的变量列表：

- 1 如果 SWF 文件不是在测试模式下运行，请选择“控制” > “测试影片”。
- 2 选择“调试” > “变量列表”。

在“输出”面板中会显示 SWF 文件中当前所有变量的列表，如下面的例子中所示：

```
Global Variables:
  Variable _global.MyGlobalArray = [object #1] [
    0:1,
    1:2,
    2:3
  ]
Level #0:
  Variable _level0.$version = "WIN 6.0.0.101"
  Variable _level0.RegularVariable = "Gary"
  Variable _level0.AnObject = [object #1] {
    MyProperty:[getter/setter] 3.14159
  }
```

显示文本字段属性以进行调试

若要获取有关 TextField 对象的调试信息，您可以在测试影片模式中使用“调试” > “列出变量”命令。“输出”面板使用以下惯例显示 TextField 对象：

- 如果对象上未找到属性，则不显示。
- 一行上显示的属性不超过 4 个。
- 具有字符串值的属性显示在单独的行上。
- 如果处理内置属性后又为对象定义了任何其它属性，则将它们按照上面第二项和第三项中的规则添加到显示中。
- 颜色属性显示为十六进制数字 (0x00FF00)。
- 属性按以下顺序显示：variable、text、htmlText、html、textWidth、textHeight、maxChars、borderColor、backgroundColor、textColor、border、background、wordWrap、password、multiline、selectable、scroll、hscroll、maxscroll、maxhscroll、bottomScroll、type、embedFonts、restrict、length、tabIndex、autoSize。

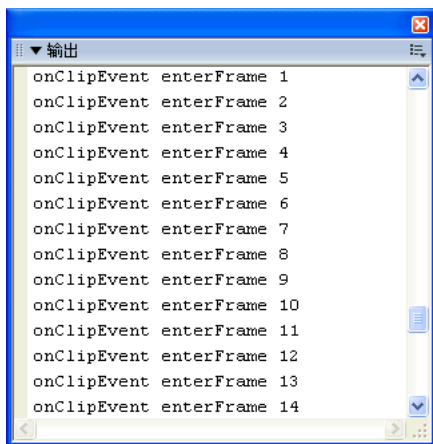
测试模式中的“调试” > “对象列表”命令列出 TextField 对象。如果为文本字段指定了一个实例名称，则将以如下形式在“输出”面板中显示包括该实例名称的完整目标路径：

Target = "target path"

使用 trace 语句

在脚本中使用 trace 语句时，可以将信息发送到“输出”面板。例如，在测试影片或场景时，可以向面板发送具体的编程注释，或者在按下按钮或播放帧时使特定的结果得以显示。trace 语句与 JavaScript 的 alert 语句类似。

在脚本中使用 trace 语句时，可以使用表达式作为参数。在测试模式下，表达式的值会显示在“输出”面板中，如下所示：



trace 语句返回的值将显示在“输出”面板上。

```
onClipEvent (enterFrame) {  
    trace("onClipEvent enterFrame " + enterFrame++)  
}
```

为测试影片更新 Flash Player

您可以从 Macromedia 支持中心 www.macromedia.com/go/flash_support_cn 下载最新的 Flash Player 版本，然后使用这一最新版本的 Flash Player 测试您的 SWF 文件。

第 II 部分

处理事件和创建交互操作

事件既包括由用户生成的事件（例如，单击鼠标或按下键盘键），也包括作为其它过程的结果而发生的事件（例如，通过网络加载 XML 文件）。这一部分中的第一章介绍 Macromedia Flash 中不同类型的事件，以及如何在动作脚本中处理这些事件。第二章介绍如何应用这些原则以创建简单的交互式演示、应用程序和动画。

第 4 章：处理事件 75

第 5 章：用动作脚本创建交互操作..... 81

第 4 章

处理事件

事件是指软件或硬件发生的事情，它要求 Macromedia Flash 应用程序的响应。例如，鼠标单击或按下键盘键之类的事件称作用户事件，因为它是由直接用户交互操作而发生的。Flash Player 自动生成的事件（例如影片剪辑在舞台上第一次出现）称作系统事件，因为它不是由用户直接生成的。

为使应用程序能够对事件做出反应，必须使用事件处理函数。事件处理函数是与特定对象和事件关联的动作脚本代码。例如，当用户单击舞台上的一个按钮时，可以将播放头前进到下一帧。或者，在 XML 文件通过网络加载完毕后，可以在文本字段中显示该文件的内容。

动作脚本提供了以下几种不同的方法来处理事件：事件处理函数方法、事件侦听器以及按钮和影片剪辑事件处理函数。

使用事件处理函数方法

事件处理函数方法是一种类方法，它在事件在该类的实例上发生时调用。例如，Button 类定义 `onPress` 事件处理函数，只要按下鼠标就对 Button 对象调用该处理函数。但是，与一个类的其它方法不同，您没有直接调用事件处理函数；Flash Player 在相应事件发生时自动调用事件处理函数。

默认情况下，事件处理函数方法是未定义的：在发生特定事件时，将调用其相应的事件处理函数，但应用程序不会进一步响应该事件。要让应用程序响应该事件，需要使用 `function` 语句定义一个函数，然后将该函数分配给相应的事件处理函数。然后，只要发生该事件，就自动调用分配给该事件处理函数的函数。

事件处理函数由以下三部分组成：事件所应用的对象、对象的事件处理函数方法的名称和分配给事件处理函数的函数。下例显示事件处理函数的基本结构。

```
object.eventMethod = function () {  
    // 您的代码，对事件作出反应  
}
```

例如，假定您在舞台上具有名为 `next_btn` 的按钮。以下代码将一个函数分配给按钮的 `onPress` 事件处理函数；该函数将播放头向前移动到时间轴上的下一帧。

```
next_btn.onPress = function ()  
    nextFrame();  
}
```

在以上代码中，`nextFrame()` 函数被直接分配给 `onPress`。您也可以将一个函数引用（名称）分配给某一事件处理函数方法，然后在以后定义该函数。

```
// 将一个函数引用分配给按钮的 onPress 事件处理函数方法
```

```
next_btn.onPress = goNextFrame;
```

```
// 定义 doSubmit() 函数
function goNextFrame() {
    nextFrame();
}
```

请注意，您将函数引用（而不是函数的返回值）分配给 `onPress` 事件处理函数。

```
// 错误!
next_btn.onPress = goNextFrame();
// 正确。
next_btn.onPress = goNextFrame;
```

某些事件处理函数接收提供与发生的事件有关的信息的传递参数。例如，在文本字段实例获取键盘焦点时调用 `TextField.onSetFocus` 事件处理函数。此事件处理函数接收对以前具有键盘焦点的文本字段对象的引用。

例如，以下代码将某些文本插入刚失去键盘焦点的文本字段。

```
userName_txt.onSetFocus = function(oldFocus_txt) {
    oldFocus_txt.text = "I just lost keyboard focus";
}
```

以下动作脚本类定义事件处理函数：`Button`、`ContextMenu`、`ContextMenuItems`、`Key`、`LoadVars`、`LocalConnection`、`Mouse`、`MovieClip`、`MovieClipLoader`、`Selection`、`SharedObject`、`Sound`、`Stage`、`TextField`、`XML` 和 `XMLSocket`。有关它们提供的事件处理函数的更多信息，请参见第 181 页的第 12 章“[动作脚本字典](#)”中这些类的相应条目。

还可以将函数分配给在运行时创建的对象的事件处理函数。例如，以下代码创建一个新的影片剪辑实例 (`newclip_mc`)，然后将一个函数分配给该剪辑的 `onPress` 事件处理函数。

```
_root.attachMovie("symbolID", "newclip_mc", 10);
newclip_mc.onPress = function () {
    trace("You pressed me");
}
```

有关更多信息，请参见第 111 页的“[在运行时创建影片剪辑](#)”。

使用事件侦听器

事件侦听器让一个对象（称作侦听器对象）接收由其它对象（称作广播器对象）生成的事件。广播器对象注册侦听器对象以接收由该广播器生成的事件。例如，您可以注册影片剪辑对象以从舞台接收 `onResize` 通知，或者按钮实例可以从文本字段对象接收 `onChanged` 通知。您可以注册多个侦听器对象以从一个广播器接收事件，也可以注册一个侦听器对象以从多个广播器接收事件。

事件侦听器的事件模型类似于事件处理函数的事件模型（请参见第 75 页的“[使用事件处理函数方法](#)”），但有两个主要差别：

- 您向其分配事件处理函数的对象不是发出该事件的对象。
- 您调用广播器对象的特殊方法 `addListener()`，该方法将注册侦听器对象以接收其事件。

要使用事件侦听器，您用具有该广播器对象生成的事件名称的属性创建侦听器对象。然后，将一个函数分配给该事件侦听器（以某种方式响应该事件）。最后，您在正广播该事件的对象上调用 `addListener()`，向它传递侦听器对象的名称。以下代码概要介绍了事件侦听器模型。

```
listenerObject.eventName = function(){
    // 此处是您的代码
};
broadcastObject.addListener(listenerObject);
```

指定的侦听器对象可以是任何对象，例如舞台上的影片剪辑或按钮实例，或者可以是任何动作脚本类的实例。事件名称是在 *broadCastObject* 上发生的事件，然后将该事件广播到 *listenerObject*。可以向一个事件广播器注册多个侦听器。

以下示例显示如何使用 *Selection.onSetFocus* 事件侦听器为输入文本字段组创建简单焦点管理器。在这个例子中，启用（显示）获得键盘焦点的文本字段的边框，并禁用失去焦点的文本字段的边框。

使用事件侦听器创建简单焦点管理器：

- 1 使用文本工具在舞台上创建一个文本字段。
- 2 选择该文本字段，接着在属性检查器的“文本类型”弹出菜单中选择“输入”，然后选择“在文本周围显示边框”选项。
- 3 在第一个文本字段下创建另一个输入文本字段。

确保为该文本字段未选择“在文本周围显示边框”选项。根据需要继续创建输入文本字段。

- 4 在“时间轴”中选择“第 1 帧”，然后打开“动作”面板（“窗口”>“开发面板”>“动作”）。
- 5 要从 *Selection* 类创建侦听焦点通知的对象，请在“动作”面板中输入以下代码：

```
var focusListener = new Object();
focusListener.onSetFocus = function(oldFocus_txt, newFocus_txt) {
    oldFocus_txt.border = false;
    newFocus_txt.border = true;
}
```

此代码创建名为 *focusListener* 的新的（通用）动作脚本对象。该对象为其本身定义 *onSetFocus* 属性，并为该属性分配一个函数。该函数采用两个参数：对失去焦点的文本字段的引用和对获得焦点的文本字段的引用。该函数将失去焦点的文本字段的 *border* 属性设置为 *false*，将获得焦点的文本字段的 *border* 属性设置为 *true*。

- 6 要注册 *focusListener* 对象以从 *Selection* 对象接收事件，请向“动作”面板添加以下代码：
Selection.addListener(focusListener);
- 7 测试影片（“控制”>“测试影片”），在第一个文本字段中单击，然后按下 Tab 键在各字段之间切换焦点。

若要注销侦听器对象以使其不再接收事件，可调用广播器对象的 *removeListener()* 方法，向它传递侦听器对象的名称。

```
broadcastObject.removeListener(listenerObject);
```

事件侦听器可用于以下动作脚本类的对象：*Key*、*Mouse*、*MovieClipLoader*、*Selection*、*TextField* 和 *Stage*。有关可用于各个类的事件侦听器的列表，请参见第 181 页的第 12 章“[动作脚本字典](#)”中这些类的相应条目。

使用按钮和影片剪辑事件处理函数

可以使用 *onClipEvent()* 和 *on()* 处理函数直接将事件处理函数附加到按钮或影片剪辑实例。*onClipEvent()* 处理函数处理影片剪辑事件，而 *on()* 处理函数处理按钮事件。也可以将 *on()* 用于影片剪辑，创建接收按钮事件的影片剪辑。有关更多信息，请参见第 78 页的“[创建具有按钮状态的影片剪辑](#)”。

要使用 *on()* 或 *onClipEvent()* 处理函数，将它直接附加到舞台上按钮或影片剪辑的实例，并且指定您要为该实例处理的事件。例如，只要用户单击处理函数附加到的按钮，就执行以下 *on()* 事件处理函数。

```
on (press) {
```

```
        trace("Thanks for pressing me.");
    }
}
```

您可以为每个 `on()` 处理函数指定两个或多个事件（用逗号分隔）。当该处理函数指定的任何事件之一发生时执行处理函数中的动作脚本。例如，只要鼠标滚过一个按钮，就执行附加到该按钮的以下 `on()` 处理函数。

```
on(rollOver, rollOut) {
    trace("You rolled over, or rolled out");
}
```

如果您想要在发生不同事件时运行不同的脚本，也可以向一个对象附加多个处理函数。例如，可以将以下 `onClipEvent()` 处理函数附加到同一影片剪辑实例。当影片剪辑第一次加载时（或者在舞台上出现时）第一个函数执行；在从舞台卸载该影片剪辑时第二个函数执行。

```
onClipEvent(load) {
    trace("I've loaded");
}
onClipEvent(unload) {
    trace("I've unloaded");
}
```

要查看 `on()` 和 `onClipEvent()` 事件处理函数所支持的事件的完整列表，请参见第 538 页的 `on()` 和第 539 页的 `onClipEvent()`。

通过 `on()` 和 `onClipEvent()` 处理事件与通过您定义的事件处理函数方法处理事件并不冲突。例如，假定 SWF 文件中有一个按钮；该按钮可以具有一个通知影片播放的 `on(press)` 处理函数，并且这个按钮还可以具有一个 `onPress` 方法，用于定义通知舞台上某个对象旋转的函数。单击该按钮后，该 SWF 文件将播放，并且该对象将旋转。根据您的首选参数，可以使用 `on()` 和 `onClipEvent()`、事件处理函数方法或这两种类型的事件处理。但是，`on()` 和 `onClipEvent()` 处理函数中变量和对象的范围不同于事件处理函数和事件侦听器中的范围。（请参见第 79 页的“事件处理函数的范围”。）

只能将 `onClipEvent()` 和 `on()` 附加到创作期间已放置于舞台上的影片剪辑实例。不能将 `onClipEvent()` 或 `on()` 附加到在运行时（例如，使用 `attachMovie()` 方法）创建的影片剪辑实例上。要将事件处理函数附加到运行时创建的对象，请使用事件处理函数方法或事件侦听器。（请参见第 75 页的“使用事件处理函数方法”和第 76 页的“使用事件侦听器”。）

创建具有按钮状态的影片剪辑

在您将 `on()` 处理函数附加到一个影片剪辑时，或者将某一函数附加到用于影片剪辑实例的某一 `MovieClip` 鼠标事件处理函数时，该影片剪辑通过与按钮相同的方式响应鼠标事件。您还可以向影片剪辑的时间轴添加帧标签 `_up`、`_over` 和 `_down`，在影片剪辑中创建自动按钮状态（弹起、指针经过和按下）。

在用户将鼠标移到该影片剪辑之上或者单击它时，将播放头发送到具有适当的帧标签的帧。要指定影片剪辑使用的点击区域，可以使用 `MovieClip` 类的 `hitArea` 属性。

创建影片剪辑中的按钮状态：

- 1 在影片剪辑的时间轴中选择一个帧，作为一个按钮状态（弹起、指针经过或按下）。
- 2 在属性检查器中输入帧标签（`_up`、`_over` 或 `_down`）。
- 3 要添加其它按钮状态，可重复执行步骤 1 和步骤 2。

4 要令影片剪辑响应鼠标事件，请执行以下操作之一：

- 如第 77 页的“使用按钮和影片剪辑事件处理函数”中所述，将一个 `on()` 事件处理函数附加到影片剪辑实例。
- 如第 75 页的“使用事件处理函数方法”中所述，将一个函数分配给影片剪辑对象的某一鼠标事件处理函数（`onPress`、`onRelease` 等）。

事件处理函数的范围

在事件处理函数内声明和执行的变量和命令的范围（即上下文）取决于所使用的事件处理函数的类型：事件处理函数或事件侦听器，或者 `on()` 和 `onClipEvent()` 处理函数。

分配给事件处理函数方法和事件侦听器的函数（类似您编写的所有动作脚本函数）定义本地变量范围，但 `on()` 和 `onClipEvent()` 处理函数却不是这样。

例如，以下面两个事件处理函数为例。第一个处理函数是与名为 `clip_mc` 的影片剪辑关联的 `onPress` 事件处理函数。第二个处理函数是附加到同一影片剪辑实例的 `on()` 处理函数。

```
// 附加到 clip_mc 的父级剪辑时间轴：
clip_mc.onPress = function () {
    var color; // 局部函数变量
    color = "blue";
}
// 附加到 clip_mc 的 on() 处理函数：
on (press) {
    var color; // 无局部变量范围
    color = "blue";
}
```

尽管这两个事件处理函数包含相同的代码，但它们具有不同的结果。在第一个例子中，`color` 变量对于为 `onPress` 定义的函数而言是本地的。在第二个例子中，因为 `on()` 处理函数没有定义本地变量范围，所以变量的范围为影片剪辑 `clip_mc` 的时间轴。

对于附加到按钮（而不是影片剪辑）的 `on()` 事件处理函数，变量（以及函数和方法调用）的范围是包含该按钮实例的时间轴。

例如，以下 `on()` 事件处理函数将产生不同的结果，这取决于它附加到影片剪辑，还是附加到按钮对象。在第一个示例中，`play()` 函数调用启动包含该按钮的时间轴的播放头；而在第二个示例中，`play()` 函数调用启动该处理函数所附加到的那个影片剪辑的时间轴。

```
// 附加到按钮
on (press) {
    play(); // 播放父时间轴
}
// 附加到影片剪辑
on (press) {
    play(); // 播放影片剪辑的时间轴
}
```

也就是说，如果附加到按钮对象，`play()` 方法调用将应用于包含该按钮的时间轴，即该按钮的父时间轴。但如果将同一处理函数附加到影片剪辑对象，`play()` 将应用于包含该处理函数的影片剪辑。

在事件处理函数或事件侦听器函数定义内，相同的 `play()` 函数将应用于包含该函数定义的时间轴。例如，假定以下 `MovieClip.onPress` 事件处理函数是在包含影片剪辑实例 `myMovieClip` 的时间轴上定义的。

```
// 影片剪辑时间轴上定义的函数：
myMovieClip.onPress = function () {
```

```
    play(); // 播放包含该函数定义的时间轴  
}
```

如果要播放定义 `onPress` 事件处理函数的影片剪辑，则必须使用 `this` 关键字显式引用该剪辑，如下所示：

```
myMovieClip.onPress = function () {  
    this.play(); // 播放定义 onPress 处理函数的剪辑的时间轴  
}
```

“this” 关键字的范围

`this` 关键字引用当前正在执行的范围中的对象。`this` 可能会引用不同的对象，这取决于使用的是哪一种事件处理函数机制。

在事件处理函数或事件侦听器函数内，`this` 引用定义该事件处理函数或事件侦听器方法的对象。例如，在下面的代码中，`this` 引用 `myClip` 本身。

```
// 附加到 _level0.myClip 的 onPress() 事件处理函数：  
myClip.onPress = function () {  
    trace(this); // 显示 “_level0.myClip”  
}
```

在附加到影片剪辑的 `on()` 处理函数内，`this` 引用 `on()` 处理函数所附加到的那个影片剪辑。

```
// 附加到名为 “myClip” 的影片剪辑  
on (press) {  
    trace(this); // 显示 “_level0.myClip”  
}
```

在附加到按钮的 `on()` 处理函数内，`this` 引用包含该按钮的时间轴。

```
// 附加到主时间轴上的按钮  
on (press) {  
    trace(this); // 显示 “_level0”  
}
```


第 5 章

用动作脚本创建交互操作

在简单的动画中，Macromedia Flash Player 按顺序播放 SWF 文件中的场景和帧。在交互式 SWF 文件中，观众可以用键盘和鼠标跳到 SWF 文件中的不同部分、移动对象、在表单中输入信息，还可以执行许多其它交互操作。

使用动作脚本可以创建脚本来通知 Flash Player 在发生某个事件时应该执行什么动作。当播放头到达某一帧，或当影片剪辑加载或卸载，或用户单击按钮或按下某个键时，就会发生一些能够触发脚本的事件。

脚本可以由单一命令组成，如指示 SWF 文件停止播放的命令；也可以由一系列命令和语句组成，如先计算条件，再执行动作。许多动作脚本命令都很简单，可用于为 SWF 文件创建一些基本控件。其它一些动作要求创作人员熟悉编程语言，主要用于高级开发。

关于事件和交互

只要用户单击鼠标或按下某个按键，就生成一个事件。这些类型的事件通常称作用户事件，因为这些事件是为响应最终用户的某一操作而生成的。您可以编写动作脚本以响应或处理 这些事件。例如，在用户单击按钮时，您可能想要将播放头发送到 SWF 文件中的其它帧，或者将新网页加载到浏览器。

在 SWF 文件中，按钮、影片剪辑和文本字段都生成可以响应的事件。动作脚本提供三种方法来处理事件：事件处理函数方法、事件侦听器以及 `on()` 和 `onClipEvent()` 处理函数。有关事件和处理事件的更多信息，请参见第 75 页的第 4 章“处理事件”。

控制 SWF 文件回放

下面是一些动作脚本函数，用于控制时间轴上的播放头，以及将新的网页加载到浏览器窗口中：

- `gotoAndPlay()` 和 `gotoAndStop()` 函数将播放头发送到帧或场景。这两个函数是您可以从任何脚本调用的全局函数。您还可以使用 `MovieClip.gotoAndPlay()` 和 `MovieClip.gotoAndStop()` 两个方法来导航特定影片剪辑对象的时间轴。
- `play()` 和 `stop()` 动作用于播放和停止影片。
- `getURL()` 动作用于跳到不同的 URL。

跳到某一帧或场景

要跳到 SWF 文件中的具体帧或场景，您可以使用 `gotoAndPlay()` 和 `gotoAndStop()` 全局函数，或者 `MovieClip` 类的、与这两个函数等效的 `gotoAndPlay()` 和 `gotoAndStop()` 方法。每个函数或方法都可用于在当前场景中指定一个要跳到的帧。如果您的文档中包含多个场景，则可以指定要跳到的场景和帧。

以下示例在按钮对象的 `onRelease` 事件处理函数内使用全局 `gotoAndPlay()` 函数来将包含该按钮的时间轴的播放头发送到第 10 帧。

```
jump_btn.onRelease = function () {  
    gotoAndPlay(10);  
}
```

在下一个示例中，`MovieClip.gotoAndStop()` 方法将名为 `categories_mc` 的影片剪辑的时间轴发送到第 10 帧，然后停止。当使用 `MovieClip` 方法 `gotoAndPlay()` 和 `gotoAndStop()` 时，您必须指定要应用该方法的实例。

```
jump_btn.onPress = function () {  
    categories_mc.gotoAndStop(10);  
}
```

播放和停止影片剪辑

除非另有命令指示，否则在 SWF 文件开始播放后，它将把时间轴上的每一帧从头播放到尾。您可以通过使用 `play()` 和 `stop()` 全局函数或者等效的 `MovieClip` 方法停止或开始播放 SWF 文件。例如，可以使用 `stop()` 在某一场景结束时，在继续播放下一场景之前停止播放 SWF 文件。SWF 文件停止播放后，必须通过调用 `play()` 来明确指示要重新开始播放。

可以使用 `play()` 和 `stop()` 函数或 `MovieClip` 的方法来控制主时间轴，或任何影片剪辑或已加载 SWF 文件的时间轴。您要控制的影片剪辑必须有一个实例名称，而且必须显示在时间轴上。

以下附加到一个按钮的 `on(press)` 处理函数将在包含该按钮对象的 SWF 文件或影片剪辑中启动播放头移动。

```
// 附加到按钮实例  
on (press) {  
    // 播放包含该按钮的时间轴  
    play();  
}
```

这个相同的 `on()` 事件处理函数代码在被附加到影片剪辑对象（而不是按钮）时将产生不同的结果。在附加到按钮对象时，默认情况下，在 `on()` 处理函数内生成的语句将应用于包含该按钮的时间轴。不过，在附加到影片剪辑对象时，在 `on()` 处理函数内生成的语句将应用于附加了 `on()` 处理函数的影片剪辑。

例如，以下 `on()` 处理函数代码将停止附加了处理函数的影片剪辑的时间轴，而不是包含该影片剪辑的时间轴。

```
on (press) {  
    stop();  
}
```

相同的情况适用于附加到影片剪辑对象的 `onClipEvent()` 处理函数。例如，以下代码在影片剪辑第一次加载时或在舞台上出现时，停止那个包含 `onClipEvent()` 处理函数的影片剪辑的时间轴。

```
onClipEvent (load) {  
    stop();  
}
```

跳到不同的 URL

要在浏览器窗口中打开网页，或将数据传递到所定义 URL 处的另一个应用程序，可以使用 `getURL()` 全局函数或 `MovieClip.getURL()` 方法。例如，可以有一个链接到新 Web 站点的按钮，也可以将时间轴变量发送到 CGI 脚本，以便像处理 HTML 表单一样处理数据。您还可以指定目标窗口，就像用 HTML 锚记 (`<a>`) 标签确定目标窗口一样。

例如，以下代码在用户单击名为 `homepage_btn` 的按钮实例时在空浏览器窗口中打开 `macromedia.com` 主页。

```
homepage_btn.onRelease = function () {  
    getURL("http://www.macromedia.com", "_blank");  
}
```

您还可以使用 `GET` 或 `POST` 将变量与 URL 一起发送。如果正从应用程序服务器加载的页面（例如一个 ColdFusion Server (CFM) 页面）预计接收表单变量，则可以使用上述功能。例如，假定您要加载名为 `addUser.cfm` 的、预计接收 `name` 和 `age` 两个表单变量的 CFM 页面。为此，您可以创建一个名为 `variables_mc` 的影片剪辑，它定义如下所示的两个变量。

```
variables_mc.name = "Francois";  
variables_mc.age = 32;
```

随后，以下代码将 `addUser.cfm` 加载到空浏览器窗口中，并传递到 `POST` 标头中的 CFM 页面 `variables_mc.name` 和 `variables_mc.age`。

```
variables_mc.getURL("addUser.cfm", "_blank", "POST");
```

有关更多信息，请参见第 360 页的 `getURL()`。

创造交互性和视觉效果

要创造交互性和其它视觉效果，您需要了解以下技术：

- [创建自定义鼠标指针](#)
- [获取鼠标位置](#)
- [捕获按键](#)
- [设置颜色值](#)
- [创建声音控件](#)
- [检测冲突](#)
- [创建简单的线条绘画工具](#)

创建自定义鼠标指针

标准鼠标指针就是用户的鼠标位置在操作系统屏幕上的表示。通过使用在 Flash 中设计的鼠标指针来代替标准鼠标指针，可以将用户的鼠标移动更紧密地集成到 SWF 文件中。本部分的范例使用的是一个看起来如同大箭头的自定义指针。不过，此功能的强大与否取决于您制作各种形态的自定义指针的能力，例如即将射门的足球，或盖在沙发上用于改变其颜色的织物布样。

要创建自定义指针，可在舞台上设计该指针的影片剪辑。然后在动作脚本中隐藏标准指针，并跟踪自定义指针的移动。要隐藏标准指针，可使用内置 `Mouse` 类的 `Mouse.hide()` 方法。要使用影片剪辑作为自定义指针，可使用 `startDrag()` 动作。

创建自定义指针：

- 1 创建影片剪辑，将其用作自定义指针并将该剪辑的实例放置在舞台上。
- 2 在舞台上选择该影片剪辑实例。

3 如果看不到“动作”面板，则选择“窗口” > “开发面板” > “动作”将其打开。

4 在“动作”面板中键入以下内容：

```
onClipEvent (load) {  
    Mouse.hide();  
    startDrag(this, true);  
}  
onClipEvent(mouseMove) {  
    updateAfterEvent();  
}
```

第一个 onClipEvent() 处理函数在影片剪辑第一次在舞台上出现时隐藏鼠标；第二个处理函数在用户移动鼠标时调用 updateAfterEvent。

updateAfterEvent 函数在发生指定的事件后立即刷新屏幕，而不是在绘制下一帧时刷新，后者是默认行为。（请参见第 699 页的 updateAfterEvent()。）

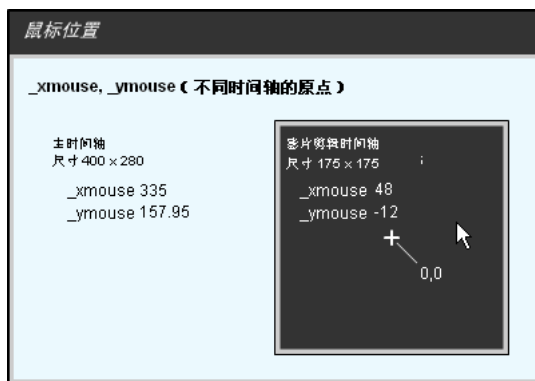
5 选择“控制” > “测试影片”来测试您的自定义指针。

当使用自定义指针时，按钮仍然起作用。将自定义指针放在时间轴的顶层是一个很好的方法，这样当您在 SWF 文件中移动鼠标时，它就可以在按钮和其它对象的前面移动。此外，自定义鼠标指针的“尖端”是您用作自定义指针的影片剪辑的注册点。因此，如果您希望影片剪辑的某一部分作为鼠标尖端，则应将剪辑的注册点坐标设置为该点的坐标。

有关 Mouse 类的方法的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 Mouse 类条目。

获取鼠标位置

可以使用 _xmouse 和 _ymouse 属性找到 SWF 文件中鼠标指针（光标）的位置。每个时间轴都有一个 _xmouse 和 _ymouse 属性，这两个属性返回鼠标在其坐标系统内的位置。该位置始终是相对于注册点而言的。对于主时间轴 (_level0)，注册点是左上角。



主时间轴内的 _xmouse 和 _ymouse 属性以及影片剪辑时间轴

下面的步骤说明了获取鼠标位置的两种方式。

获取主时间轴内的当前鼠标位置：

- 1 创建两个动态文本框，然后将它们命名为 x_pos 和 y_pos。
- 2 如果看不到“动作”面板，则选择“窗口” > “开发面板” > “动作”将其打开。

3 若要返回主时间轴内的鼠标位置，请在 _level0 SWF 文件的任意帧中添加以下代码：

```
x_pos = _root._xmouse;  
y_pos = _root._ymouse;
```

变量 `x_pos` 和 `y_pos` 被用作容器，用于放置鼠标位置值。可以在您的文档的任何脚本中使用这些变量。在下面的 `onClipEvent()` 处理函数中，用户每次移动鼠标时都会更新 `x_pos` 和 `y_pos` 的值。

```
onClipEvent(mouseMove) {  
    x_pos = _root._xmouse;  
    y_pos = _root._ymouse;  
}
```

获取影片剪辑内的当前鼠标位置：

- 1 创建影片剪辑。
- 2 在舞台上选择该影片剪辑实例。使用属性检查器将它命名为 `myMovieClip`。
- 3 如果看不到“动作”面板，则选择“窗口” > “开发面板” > “动作”将其打开。
- 4 使用影片剪辑的实例名称来返回主时间轴内的鼠标位置。

例如，可将下面的语句放置在 _level0 SWF 文件中的任意时间轴上，以返回 `myMovieClip` 实例中的 `_ymouse` 位置：

```
x_pos = _root.myMovieClip._xmouse  
y_pos = _root.myMovieClip._ymouse
```

该代码返回鼠标相对于注册点的 `_xpos` 和 `_ypos` 值。

- 5 选择“控制” > “测试影片”对影片进行测试。

您也可以通过在剪辑事件中使用 `_xmouse` 和 `_ymouse` 属性来确定鼠标在影片剪辑内的位置，如下面的代码所示：

```
onClipEvent (enterFrame) {  
    xmousePosition = this._xmouse;  
    ymousePosition = this._ymouse;  
}
```

有关 `_xmouse` 和 `_ymouse` 属性的更多信息，请参见第 498 页的 `MovieClip._xmouse` 和第 499 页的 `MovieClip._ymouse`。

捕获按键

您可以使用内置的 `Key` 类的方法来检测用户上次按下的键。`Key` 类不需要构造函数；要使用它的方法，您只需调用该类自身上的方法即可，如下面的示例所示：

```
Key.getCode();
```

您可以获得按键的虚拟键控代码或 ASCII（美国信息交换标准码）值：

- 要获得上次所按键的虚拟键控代码，可使用 `getCode()` 方法。
- 要获得上次所按键的 ASCII 值，可使用 `getAscii()` 方法。

键盘上的每一个实际的键都有一个虚拟键控代码。例如，左箭头键的虚拟键控代码为 37。通过使用虚拟键控代码，可以确保 SWF 文件的控制在每个键盘上都相同，而不用考虑语言或平台。

ASCII 值分配给每个字符集的前 127 个字符。ASCII 值提供了有关屏幕上字符的信息。例如，字母“A”和字母“a”有不同的 ASCII 值。

要确定使用哪些键并确定它们的虚拟键控代码，可使用下面其中一种方式：

- 查看第 745 页的附录 C “键盘键和键控代码值” 中的键控代码列表。
- 使用 Key 类常数。（在“动作”工具箱中单击“内置类”类别，然后单击“Movie”，之后单击“Key”，最后单击“Constants”。）
- 将以下 onClipEvent() 处理函数分配给影片剪辑，然后选择“控制” > “测试影片”并按所需键。

```
onClipEvent(keyDown) {  
    trace(Key.getCode());  
}
```

所需键的键控代码将出现在“输出”面板中。

通常是在事件处理函数内使用 Key 类方法。在下面的示例中，用户使用箭头键移动汽车。Key.isDown() 方法表明当前按住的键是左箭头键、右箭头键、上箭头键、还是下箭头键。事件处理函数 onEnterFrame 通过 if 语句确定 Key.isDown(keyCode) 的值。根据该值，处理函数会指示 Flash Player 更新汽车的位置并显示方向。



通过键盘键的输入来移动汽车。

以下步骤介绍如何捕获按键动作来根据当前按下的箭头键（向上、向下、向左或向右）在舞台上向上、向下、向左或向右移动影片剪辑。影片剪辑被限定在一个 400 像素宽、300 像素高的任意区域内。此外，文本字段显示所按下的键的名称。

创建一个能被键盘激活的影片剪辑：

- 1 在舞台上创建一个影片剪辑，该影片剪辑将会随键盘箭头的活动而移动。
在此示例中，影片剪辑实例名称为 `car`。
- 2 在舞台上创建一个随汽车方向更新的动态文本框。使用属性检查器，给它指定以下实例名：
`display_txt`。

注意：不要把变量名和实例名混淆。有关更多信息，请参见第 120 页的“关于文本字段实例和变量名称”。

- 3 在时间轴上选择第 1 帧；如果看不到“动作”面板，则选择“窗口”>“开发面板”>“动作”将其打开。
- 4 要设置每次按键时汽车沿屏幕移动的距离，请定义 `distance` 变量并将它的初始值设置为 10。

```
var distance = 10;
```

- 5 要为汽车影片剪辑创建事件处理函数，以检查当前按下的是哪一个箭头键（向左、向右、向上或向下），请向“动作”面板添加以下代码：

```
car.onEnterFrame = function() {  
  
}
```

- 6 将一个 `with` 语句添加到 `onEnterFrame` 处理函数的主体中，并将 `car` 指定为 `with` 语句的对象。

您的代码应该如下所示：

```
var distance = 10;  
car.onEnterFrame = function() {  
    with (car) {  
    }  
}
```

- 7 若要检查是否已按下右箭头键，并相应地移动汽车影片剪辑，请将以下代码添加到 `with` 语句的主体中。您的代码应该如下所示：

```
distance = 10;  
car.onEnterFrame = function() {  
    with (car) {  
        if (Key.isDown(Key.RIGHT)) {  
            _x += distance;  
            if (_x >= 400) {  
                _x = 400;  
            }  
            _root.display_txt.text = "Right";  
        }  
    }  
}
```

如果按下了右箭头键，则汽车的 `_x` 属性将按 `distance` 变量指定的量增加值。下一个 `if` 语句测试剪辑的 `_x` 属性的值是否大于或等于 400 (`if(_x >=400)`)；如果是，则其位置将固定在 400。另外，`Right` 一词应显示在 SWF 文件中。

- 8 使用类似的代码检查是否按下了左箭头键、上箭头键或下箭头键。您的代码应该如下所示：

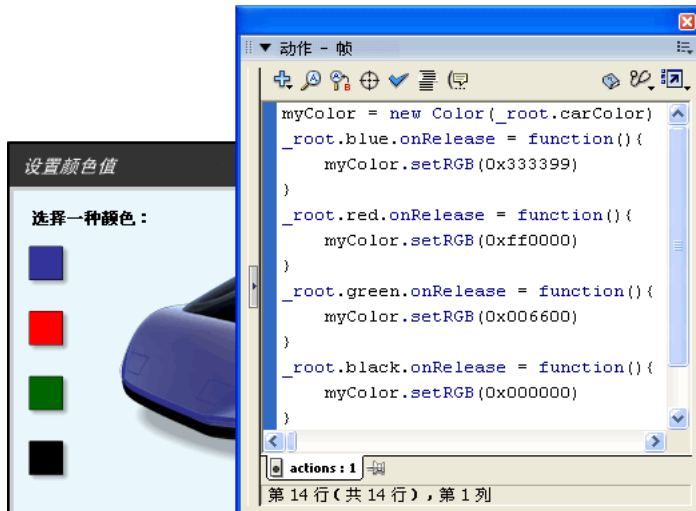
```
var distance = 10;
car.onEnterFrame = function() {
    with (car) {
        if (Key.isDown(Key.RIGHT)) {
            _x += distance;
            if (_x >= 400) {
                _x = 400;
            }
            _root.display_txt.text = "Right";
        } else if (Key.isDown(Key.LEFT)) {
            _x -= distance;
            if (_x < 0) {
                _x = 0;
            }
            _root.display_txt.text = "Left";
        } else if (Key.isDown(Key.UP)) {
            _y -= distance;
            if (_y < 0) {
                _y = 0 ;
            }
            _root.display_txt.text = "Up";
        } else if (Key.isDown(Key.DOWN)) {
            _y += distance;
            if (_y > 300) {
                _y = 300;
            }
            _root.display_txt.text = "Down";
        }
    }
}
```

- 9 选择“控制” > “测试影片”对文件进行测试。

有关 Key 类的方法的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 Key 类条目。

设置颜色值

可以使用内置 `Color` 类的方法来调节影片剪辑的颜色。`setRGB()` 方法会为影片剪辑指定十六进制 RGB（红、绿、蓝）值。下面的示例就使用 `setRGB()` 来根据用户输入来更改对象的颜色。



该按钮动作创建了一个 `Color` 对象，并根据用户的输入改变汽车的颜色。

设置影片剪辑的颜色值：

- 1 选择舞台上的一个影片剪辑。
- 2 在属性检查器中，输入 `carColor` 作为实例名称。
- 3 创建一个名为 `color chip` 的按钮，接着在舞台上放置该按钮的四个实例，并将它们分别命名为 `red`、`green`、`blue` 和 `black`。
- 4 在主时间轴上选择第 1 帧，然后选择“窗口” > “开发面板” > “动作”。
- 5 若要为 `carColor` 影片剪辑创建一个 `Color` 对象，请在“动作”面板中添加以下代码：

```
myColor = new Color(_root.carColor);
```

- 6 要使蓝色按钮将 `carColor` 影片剪辑的颜色更改为蓝色，请向“动作”面板添加以下代码：

```
_root.blue.onRelease = function(){
    myColor.setRGB(0x0000ff)
}
```

十六进制值 `0x0000ff` 为蓝色。下表显示您将使用的其它颜色及其十六进制值：

- 7 为其它按钮（红色、绿色和黑色）重复执行第 6 步，以便将影片剪辑的颜色更改为相应的颜色。您的代码现在应该如下所示：

```
myColor = new Color(_root.carColor)
_root.blue.onRelease = function(){
    myColor.setRGB(0x0000ff)
}
_root.red.onRelease = function(){
    myColor.setRGB(0xff0000)
}
_root.green.onRelease = function(){
    myColor.setRGB(0x00ff00)
}
```

```

_root.black.onRelease = function(){
    myColor.setRGB(0x000000)
}

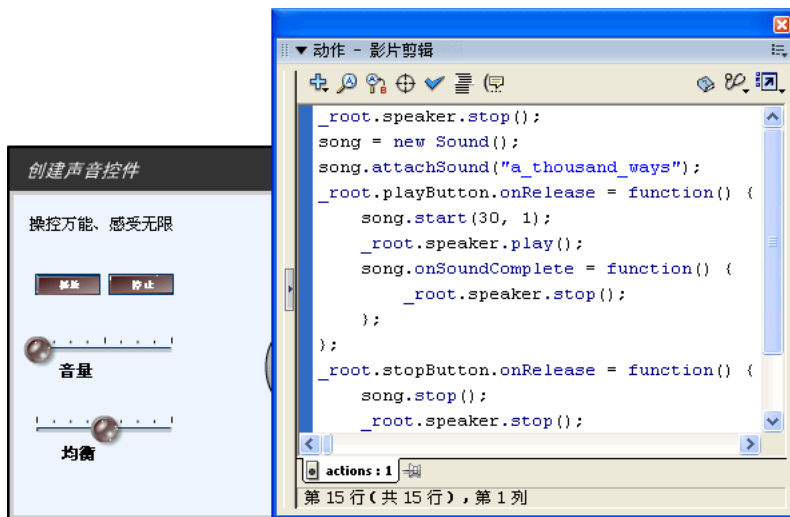
```

8 选择“控制”>“测试影片”来改变影片剪辑的颜色。

有关 Color 类的方法的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 Color 类条目。

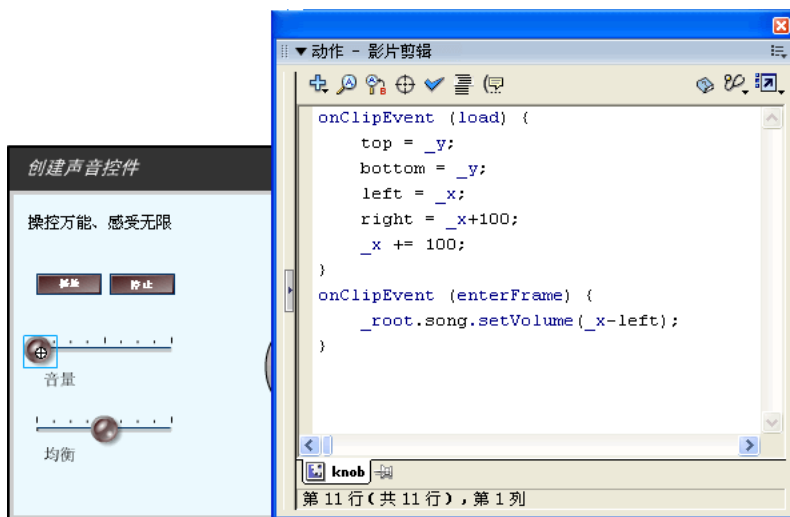
创建声音控件

可以使用内置 Sound 类控制 SWF 文件中的声音。若要使用 Sound 类的方法，必须先创建一个 Sound 对象。然后可以使用 attachSound() 方法在 SWF 文件运行时将库中的声音插入该 SWF 文件。



当用户松开“播放”按钮时，扬声器就会播放出一首歌曲。

Sound 类的 `setVolume()` 方法控制着音量，而 `setPan()` 方法则调节声音的左右平衡。



当用户拖动音量滑块时，就会调用 `setVolume()` 方法。

下面的步骤介绍如何创建如上所示的声音控件。

将声音附加到时间轴上：

- 1 选择“文件” > “导入”来导入一种声音。
- 2 在库中选择声音，右键单击 (Windows) 或按住 Control 键单击 (Macintosh)，然后选择“链接”。
- 3 选择“为动作脚本导出”和“在第一帧导出”，然后为其指定标识符 `a_thousand_ways`。
- 4 在舞台上添加一个按钮，然后将它命名为 `playButton`。
- 5 在舞台上添加一个按钮，然后将它命名为 `stopButton`。
- 6 在舞台上添加一个影片剪辑，然后将它命名为 `speaker`。
- 7 在主时间轴上选择第 1 帧，然后选择“窗口” > “开发面板” > “动作”。将以下代码添加到“动作”面板中：

```
speaker.stop();
song = new Sound();
song.onSoundComplete = function() {
    speaker.stop();
};
song.attachSound("a_thousand_ways");
playButton.onRelease = function() {
    song.start();
    speaker.play();
};
stopButton.onRelease = function () {
    song.stop();
    speaker.stop();
}
```

该代码首先停止扬声器影片剪辑。然后创建一个新的 Sound 对象 (song)，并向该对象附加链接标识符为 a_thousand_ways 的声音。接下来，它为 song 对象定义 onSoundComplete 处理函数，该函数将在声音结束后停止 speaker 影片剪辑。最后，与 playButton 和 stopButton 对象关联的 onRelease 处理函数使用 Sound.start() 和 Sound.stop() 方法开始和停止该声音，并且还播放和停止 speaker 影片剪辑。

- 8 选择“控制” > “测试影片”来试听声音。

创建滑动的音量控件：

- 1 将某个按钮拖到舞台上。
- 2 选择该按钮，然后选择“修改” > “转换为符号”。选择影片剪辑行为时要小心。这创建了一个在第一帧中带有按钮的影片剪辑。

- 3 选择该影片剪辑，然后选择“编辑” > “编辑所选项目”。
- 4 选择该按钮，然后选择“窗口” > “开发面板” > “动作”。
- 5 输入下列动作：

```
on (press) {  
    startDrag(this, false, left, top, right, bottom);  
}  
on (release) {  
    stopDrag();  
}
```

startDrag() 参数 left、top、right 和 bottom 是在剪辑动作中设置的变量。

- 6 选择“编辑” > “编辑文档”返回到主时间轴。
- 7 在舞台上选择影片剪辑。
- 8 输入下列动作：

```
onClipEvent (load) {  
    top = _y;  
    bottom = _y;  
    left = _x;  
    right = _x+100;  
    _x += 100;  
}  
onClipEvent (enterFrame) {  
    _parent.song.setVolume(_x-left);  
}
```

- 9 选择“控制” > “测试影片”来使用音量滑块。

创建滑动的平衡控件：

- 1 将某个按钮拖到舞台上。
- 2 选择该按钮，然后选择“插入” > “转换为符号”。选择该影片剪辑属性。
- 3 选择该影片剪辑，然后选择“编辑” > “编辑符号”。
- 4 选择该按钮，然后选择“窗口” > “开发面板” > “动作”。
- 5 输入下列动作：

```
on (press) {  
    startDrag ("", false, left, top, right, bottom);  
    dragging = true;  
}  
on (release, releaseOutside) {  
    stopDrag();  
}
```

```
        dragging = false;
    }
```

startDrag() 参数 left、top、right 和 bottom 是在剪辑动作中设置的变量。

6 选择 “编辑” > “编辑文档” 返回到主时间轴。

7 在舞台上选择影片剪辑。

8 输入下列动作：

```
onClipEvent (load) {
    top=_y;
    bottom=_y;
    left=_x-50;
    right=_x+50;
    center=_x;
}

onClipEvent (enterFrame) {
    if (dragging==true){
        _parent.setPan((_x-center)*2);
    }
}
```

9 选择 “控制” > “测试影片” 来使用平衡滑块。

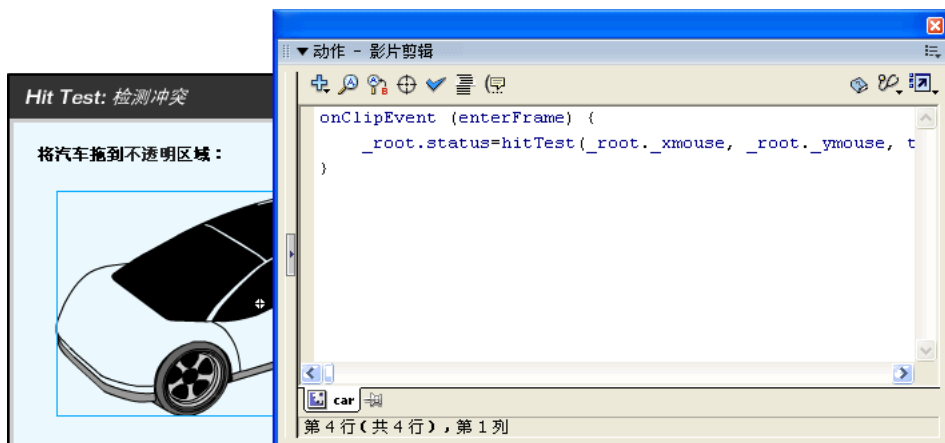
有关 Sound 类的方法的更多信息，请参见第 181 页的第 12 章 “动作脚本字典” 中的 [Sound 类](#) 条目。

检测冲突

MovieClip 类的 hitTest() 方法可以检测 SWF 文件中的冲突。它检查某个对象是否与影片剪辑有冲突，然后返回一个布尔值（true 或 false）。

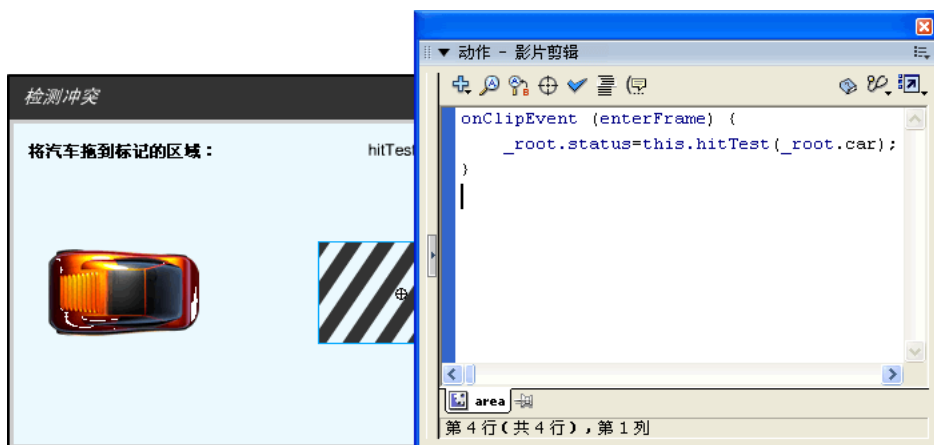
在两种情况下，您想知道是否已经发生了冲突：一种情况是测试用户是否已经到达舞台上的某个特定静态区域，另一种情况是确定影片剪辑何时接触到另一个影片剪辑。利用 hitTest() 方法，可以判定这些结果。

可以使用 `hitTest()` 的参数来指定舞台上某个点击区域的 x 和 y 坐标，或者使用另一个影片剪辑的目标路径作为点击区域。在指定 x 和 y 时，如果由 (x, y) 标识的点是非透明点，那么 `hitTest()` 返回 `true`。当目标传递给 `hitTest()` 时，会对两个影片剪辑的边框进行比较。如果它们重叠，则 `hitTest()` 返回 `true`。如果两个边框没有相交，则 `hitTest()` 返回 `false`。



只要鼠标指针位于汽车车身上方，文本字段中就会出现 “True” 字样。

您也可以使用 `hitTest()` 来测试两个影片剪辑之间的冲突。



只要一个影片剪辑与其它影片剪辑相接触，文本字段中就会出现 “True” 字样。

下面的步骤通过使用汽车示例介绍如何检测冲突。

在一个影片剪辑和舞台上的某一点之间执行冲突检测：

- 1 在舞台上创建一个新的影片剪辑，然后在属性检查器中输入 `box` 作为实例名称。
- 2 在舞台上创建一个动态文本框，然后在属性检查器中输入 `status` 作为实例名称。
- 3 在时间轴的图层 1 上选择第一个帧。
- 4 如果看不到 “动作” 面板，选择 “窗口” > “开发面板” > “动作” 将其打开。

- 5 在“动作”面板中添加以下代码：

```
box.onEnterFrame = function () {  
    status.text = this.hitTest(_xmouse, _ymouse, true);  
}
```

- 6 选择“控制” > “测试影片”，然后让鼠标滑过影片剪辑来测试冲突。
只要鼠标位于不透明的像素上方，就会显示值 true。

执行两个影片剪辑之间的冲突检测：

- 1 把两个影片剪辑拖到舞台上，然后分别为它们指定实例名 car 和 area。
- 2 在舞台上创建一个动态文本框，然后在属性检查器中输入 status 作为实例名称。
- 3 在时间轴的图层 1 上选择第一个帧。
- 4 如果看不到“动作”面板，选择“窗口” > “开发面板” > “动作”将其打开。
- 5 在“动作”面板中输入下面的代码：

```
area.onEnterFrame = function () {  
    status.text=this.hitTest(car);  
}  
car.onPress = function () {  
    this.startDrag(false);  
    updateAfterEvent();  
}  
car.onRelease = function () {  
    this.stopDrag();  
}
```

- 6 选择“控制” > “测试影片”，然后拖动影片剪辑以测试冲突检测。
只要汽车边框与停车区域的边框发生相交，该状态就会变为 true。

有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `MovieClip.hitTest()`。

创建简单的线条绘画工具

您可以使用 `MovieClip` 类的方法在 SWF 文件播放时在舞台上绘制线条和填充。这样您就可以为用户创建绘画工具，并且可以通过响应事件在 SWF 文件中绘制形状。绘画方法包括 `beginFill()`、`beginGradientFill()`、`clear()`、`curveTo()`、`endFill()`、`lineTo()`、`lineStyle()` 和 `moveTo()`。可以将这些方法应用于任何影片剪辑实例（例如 `myClip.lineTo()`），也可应用于某一级别（`_root.curveTo()`）。

`lineTo()` 和 `curveTo()` 方法分别用于绘制线条和曲线。您可以使用 `lineStyle()` 方法指定线条或曲线的线条颜色、粗细和 `alpha` 设置。`moveTo()` 绘画方法将当前绘画位置设置为您指定的 `x` 和 `y` 舞台坐标。

`beginFill()` 和 `beginGradientFill()` 方法分别用纯色填充或渐变填充来填充闭合路径，`endFill()` 将在最后的调用中指定的填充应用于 `beginFill()` 或 `beginGradientFill()`。`clear()` 方法擦除已在指定的影片剪辑对象中绘制的内容。

有关更多信息，请参见第 450 页的 `MovieClip.beginFill()`、第 450 页的 `MovieClip.beginGradientFill()`、第 453 页的 `MovieClip.clear()`、第 456 页的 `MovieClip.curveTo()`、第 459 页的 `MovieClip.endFill()`、第 470 页的 `MovieClip.lineTo()`、第 469 页的 `MovieClip.lineStyle()` 和第 474 页的 `MovieClip.moveTo()`。

创建简单线条绘画工具：

- 1 在一个新文档中，在舞台上创建一个按钮，然后在属性检查器中输入 clear_btn 作为其实例名称。
- 2 在时间轴中选择第 1 帧，如果没有看到“动作”面板，则选择“窗口” > “开发面板” > “动作”将其打开。

- 3 在“动作”面板中输入以下代码：

```
_root.onMouseDown = function() {  
    _root.lineStyle(5, 0xFF0000, 100);  
    _root.moveTo(_root._xmouse, _root._ymouse);  
    isDrawing = true;  
};  
_root.onMouseMove = function() {  
    if (isDrawing == true) {  
        _root.lineTo(_root._xmouse, _root._ymouse);  
        updateAfterEvent();  
    }  
};  
_root.onMouseUp = function() {  
    isDrawing = false;  
};  
clear_btn.onRelease = function() {  
    _root.clear();  
};
```

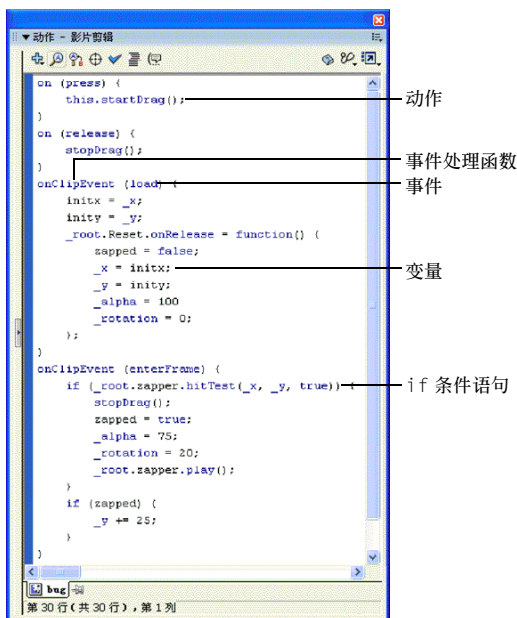
- 4 选择“控制” > “测试影片”对影片进行测试。单击并拖动鼠标来在舞台上绘制线条。单击该按钮可以擦除所绘制的内容。

分析范例脚本

在范例 SWF 文件 zipper.swf（可以在“使用 Flash”帮助中观看该文件）中，当用户将瓢虫拖到电源插座时，瓢虫会下落，并且插座会抖动。主时间轴只有一个帧，包含三个对象：瓢虫、插座和重置按钮。这些对象中的每一个都是一个影片剪辑实例。



SWF 文件中有一个脚本；它被附加到 bug 实例，如下面的“动作”面板中所示：



带有附加到 bug 实例的脚本的“动作”面板

瓢虫的实例名称为 bug，插座的实例名称为 zapper。在脚本中，this 即指瓢虫，因为脚本就附加到瓢虫上，而保留字 this 引用包含脚本的对象。

有两个 onClipEvent() 处理函数，它们各有一个不同的事件：load 和 enterFrame。onClipEvent(load) 语句中的动作仅在 SWF 文件加载时执行一次。onClipEvent(enterFrame) 语句中的动作在每次播放头进入帧时都执行。即使在只有一个帧的 SWF 文件中，播放头仍然会反复进入该帧，而且脚本会重复执行。下面的动作会在每个 onClipEvent() 处理函数中发生：

onClipEvent(load) initx 和 inity 这两个变量被指定用于存储 bug 影片剪辑实例的初始 x 和 y 位置。为 Reset 实例的 onRelease 事件定义并分配了一个函数。每次用鼠标按钮按下和松开“重置”按钮时都会调用此函数。该函数将瓢虫重放回到它在舞台上的开始位置，重置它的旋转和 alpha 值，并且将 zapped 变量重置为 false。

onClipEvent(enterFrame) if 条件语句使用 hitTest() 方法检查瓢虫实例是否触到了插座实例 (_root.zapper)。该计算具有两种可能的输出结果，即 true 或 false：

```
onClipEvent (load) {  
    initx = _x;  
    inity = _y;  
    _root.Reset.onRelease = function() {  
        zapped = false;  
        _x = initx;  
        _y = inity;  
        _alpha = 100;  
        _rotation = 0;  
    };  
}
```

如果 `hitTest()` 方法返回 `true`，则调用 `stopDrag()` 方法，`zapper` 变量设置为 `true`，`alpha` 和旋转属性会被更改，并会通知 `zapped` 实例进行播放。

如果 `hitTest()` 方法返回 `false`，则不会运行紧跟在 `if` 语句后面大括号 (`{}`) 内的任何代码。

有两个 `on()` 处理函数附加到 `bug` 实例，它们具有两个不同的事件：`press` 和 `release`。在 `bug` 实例上按下鼠标按钮时，会执行 `on(press)` 语句中的动作。在 `bug` 实例上松开鼠标按钮时，则会执行 `on(release)` 语句中的动作。下面的动作会在每个 `onClipEvent()` 处理函数中发生：

on(press) 一个 `startDrag()` 动作，使瓢虫可以拖动。因为脚本被附加到 `bug` 实例，所以关键字 `this` 表明可拖动的是 `bug` 实例：

```
on (press) {  
    this.startDrag();  
}
```

on(release) 一个 `stopDrag()` 动作，可以停止拖动动作：

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

若要观看 SWF 文件播放，请参见“动作脚本参考指南”帮助。

第 III 部分

使用对象和类

这一部分讨论 Macromedia Flash 运行时对象模型及其功能，主要介绍了如何使用影片剪辑和文本。这一部分还介绍了如何使用动作脚本 2.0 创建您自己的类和接口。

第 6 章：使用内置类.....	101
第 7 章：使用影片剪辑.....	107
第 8 章：使用文本.....	119
第 9 章：使用动作脚本 2.0 创建类.....	137

第 6 章

使用内置类

除了动作脚本的核心语言元素和构造（例如 `for` 和 `while` 循环）以及上文中所述的原始数据类型（数字、字符串和数组，请参见第 23 页的“动作脚本基础”）之外，动作脚本还提供了大量内置类，或称作复杂数据类型。这些类为您提供了多种脚本撰写功能。

这些类中有一些基于 ECMAScript 规范，称为核心动作脚本类。这些类包括 `Array`、`Boolean`、`Date` 和 `Math` 类。有关更多信息，请参见第 102 页的“核心类”。

其余的内置动作脚本类是 Macromedia Flash 和 Flash Player 对象模型专用的。要理解核心动作脚本类与 Flash 专用类之间的区别，可以从核心 JavaScript 与客户端 JavaScript 之间的区别入手：与客户端 JavaScript 类提供对客户端环境（Web 浏览器和 Web 页内容）的控制相似，Flash 专用类提供对 Flash 应用程序外观和行为的运行时控制。

本章介绍内置动作脚本类，以及可以用这些类执行的常见任务，并提供代码示例。有关这些类的概述，请参见第 102 页的“内置类概述”。有关在面向对象的编程中使用类和对象的概述，请参见第 101 页的“关于类和实例”。

关于类和实例

在面向对象编程中，类定义对象的类别。类描述对象的属性（数据）和行为（方法），这与描述建筑物特性的建筑蓝图非常相似。若要使用类所定义的属性和方法，必须先创建该类的实例。实例与它的类之间的关系类似于房子与它的建筑蓝图之间的关系。

创建新对象

若要创建动作脚本类的实例，请使用 `new` 运算符调用该类的构造函数。构造函数与类始终具有相同的名称，并返回该类的实例，而您通常会将该实例分配给一个变量。

例如，下面的代码创建一个新的 `Sound` 对象。

```
var song:Sound= new Sound();
```

某些情况下，您不需要创建类的实例即可使用它。有关更多信息，请参见第 102 页的“关于类（静态）成员”。

访问对象属性

使用点（`.`）运算符可以访问对象中的属性的值。对象名称在点的左边，而属性名称在点的右边。例如，在下面的语句中，`myObject` 是对象，而 `name` 是属性：

```
myObject.name
```

下面的代码创建一个新的 `TextField` 对象，然后将它的 `autoSize` 属性设置为 `true`。

```
var my_text = new TextField();
my_text.autoSize = true;
```

也可以使用数组访问运算符 (`[]`) 来访问对象的属性。请参见第 43 页的“点运算符和数组访问运算符”。

调用对象方法

可以通过在点 (`.`) 运算符后面加上方法来调用对象的方法。例如，下面的代码将创建一个新的声音对象，并调用其 `setVolume()` 方法：

```
mySound = new Sound(this);
mySound.setVolume(50);
```

关于类（静态）成员

有些内置动作脚本类具有称作类成员（或静态成员）的成员。类成员（属性和方法）是通过类名本身来访问或调用的，而不是通过类的实例。也就是说，不用创建类的实例即可使用这些属性和方法。

例如，`Math` 类的所有属性都是静态的。以下代码调用 `Math` 类的 `max()` 方法来确定两个数字中较大的数字。

```
var largerNumber = Math.max(10, 20);
```

内置类概述

本节列出了所有动作脚本类，其中包括对各个类的简单说明，以及对本文档中其它相关章节的交叉引用。

核心类

核心动作脚本类是从 ECMAScript 直接借用的类。在“动作”工具箱中，这些类位于“内置类” > “核心”子文件夹下。

类	说明
Arguments	包含作为参数传递给任何函数的值的数组。请参见第 181 页的第 12 章“动作脚本字典”中的 Arguments 类条目。
Array	<code>Array</code> 类提供用于处理数组对象的方法和属性。请参见第 181 页的第 12 章“动作脚本字典”中的 Array 类条目。
布尔值	<code>Boolean</code> 类是布尔值 (<code>true</code> 或 <code>false</code>) 的包装。请参见第 181 页的第 12 章“动作脚本字典”中的 Boolean 类条目。
按钮	<code>Button</code> 类提供用于处理按钮对象的方法和属性。请参见第 181 页的第 12 章“动作脚本字典”中的 Button 类条目。
日期	使用 <code>Date</code> 类可以访问相对于通用时间（格林尼治标准时间），或相对于运行 Flash Player 的操作系统的日期和时间值。请参见第 181 页的第 12 章“动作脚本字典”中的 Date 类条目。
Error	<code>Error</code> 类包含关于脚本中出现的错误的信息。通常使用 <code>throw</code> 语句来生成错误条件，然后可以使用 <code>try..catch..finally</code> 语句来处理该错误条件。请参见第 181 页的第 12 章“动作脚本字典”中的 <code>try..catch..finally</code> 和 Error 类条目。

类	说明
函数	Function 类是所有动作脚本函数（包括动作脚本的固有函数和您定义的函数）的类表示形式。请参见第 181 页的第 12 章“动作脚本字典”中的 Function 类条目。
Math	Math 类用于访问和处理数学常量和函数。Math 类的所有属性和方法都是静态的，必须使用以下语法来调用： <code>Math.method(parameter)</code> 或 <code>Math.constant</code> 。请参见第 181 页的第 12 章“动作脚本字典”中的 Math 类条目。
数字	Number 类是原始数字数据类型的包装。请参见第 181 页的第 12 章“动作脚本字典”中的 Number 类条目。
对象	Object 类位于动作脚本类层次结构的根部；其它所有类均继承其方法和属性。请参见第 181 页的第 12 章“动作脚本字典”中的 Object 类条目。
字符串	String 类是字符串原始数据类型的包装，它使您能够使用 String 对象的方法和属性处理原始字符串值类型。请参见第 181 页的第 12 章“动作脚本字典”中的 String 类条目。

Flash Player 专用类

下表列出了 Flash Player 和 Flash 运行时模型的专用类。这些类通常分为四个类别：影片类（提供对 SWF 文件和 Flash Player 的总体控制）、媒体类（用于处理声音和视频）、客户端 / 服务器类（用于处理 XML 和其它外部数据源）和创作类（提供对 Flash 创作环境的控制）。

注意：这样分类只影响这些类在“动作”工具箱中的位置，而不影响其使用方式。

影片类

影片类提供对 SWF 文件中大多数可视元素的控制，包括影片剪辑、文本字段和按钮。影片类位于“动作”工具箱中的“内置类”>“影片”子文件夹下。

类	说明
Accessibility	Accessibility 类管理 SWF 文件与屏幕读取应用程序之间的通讯。将此类的方法与全局 <code>_accProps</code> 属性结合使用，可以在运行时控制影片剪辑、按钮和文本字段的可访问属性。请参见第 181 页的第 12 章“动作脚本字典”中的 <code>_accProps</code> 和 Accessibility 类条目。
按钮	SWF 文件中的所有按钮都是 Button 类的实例。Button 类提供用于处理按钮的方法、属性和事件处理函数。请参见第 181 页的第 12 章“动作脚本字典”中的 Button 类条目。
Color	Color 类可用于获取和设置影片剪辑对象的 RGB 颜色值。有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 Color 类条目。有关使用 Color 类更改影片剪辑颜色的示例，请参见第 89 页的“设置颜色值”。
ContextMenu	ContextMenu 类可用于控制 Flash Player 上下文菜单的内容。可以将单独的 ContextMenu 对象与 MovieClip、Button 或 TextField 对象相关联（使用这些类的 <code>menu</code> 属性）。还可以通过使用 ContextMenuItem 类向 ContextMenu 对象中添加自定义菜单项。请参见第 181 页的第 12 章“动作脚本字典”中的 ContextMenu 类和 ContextMenuItem 类条目。
ContextMenuItem	ContextMenuItem 类可用于创建出现在 Flash Player 上下文菜单中的新菜单项。通过 ContextMenu 类可将使用此类创建的新菜单项添加到 Flash Player 上下文菜单中。请参见第 181 页的第 12 章“动作脚本字典”中的 ContextMenu 类和 ContextMenuItem 类条目。

类	说明
Key	Key 类提供用于获取有关键盘和按键的信息的方法和属性。有关更多信息，请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 Key 类条目。有关通过捕获按键来创建交互式 SWF 文件的示例，请参见第 85 页的 “ 捕获按键 ”。
LocalConnection	LocalConnection 类可用于在同一台计算机上运行的两个 SWF 文件之间进行通讯。请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 LocalConnection 类条目。
Mouse	Mouse 类在 SWF 文件中提供对鼠标的控制（例如，此类可用于隐藏或显示鼠标指针）。有关更多信息，请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 Mouse 类条目。有关使用 Mouse 类的示例，请参见第 83 页的 “ 创建自定义鼠标指针 ”。
MovieClip	Flash 影片中的每个影片剪辑均是 MovieClip 类的实例。您可以使用此类的方法和属性控制影片剪辑对象。请参见第 107 页的第 7 章 “ 使用影片剪辑 ” 和第 181 页的第 12 章 “ 动作脚本字典 ” 中的 MovieClip 类条目。
MovieClipLoader	MovieClipLoader 类可用于通过事件侦听器机制跟踪 SWF 和 JPEG 文件的下载进度。请参见第 175 页的 “ 预加载 SWF 和 JPEG 文件 ” 和第 181 页的第 12 章 “ 动作脚本字典 ” 中的 MovieClipLoader 类条目。
PrintJob	PrintJob 类可用于打印动态呈现的内容和多页文档。请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 PrintJob 类条目和 “使用 Flash” 帮助中的 “使用动作脚本 PrintJob 类”。
Selection	Selection 类可用于获取和设置文本字段焦点、文本字段选择范围和文本字段插入点。请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 Selection 类条目。
SharedObject	SharedObject 类在客户端计算机上提供本地数据存储。请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 SharedObject 类条目。
Stage	Stage 类提供有关 SWF 文件的尺寸、对齐方式和缩放模式的信息，并报告舞台大小调整事件。请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 Stage 类条目。
System	System 类提供有关 Flash Player 和运行 Flash Player 的系统的信息（例如，屏幕分辨率和当前系统语言）。它还可用于显示或隐藏 Flash Player 设置面板和修改 SWF 文件的安全设置。请参见第 181 页的第 12 章 “ 动作脚本字典 ” 中的 System 类条目。
TextField	TextField 类提供对动态文本字段和输入文本字段的控制。请参见第 119 页的第 8 章 “ 使用文本 ” 和第 181 页的第 12 章 “ 动作脚本字典 ” 中的 TextField 类条目。
TextField.StyleSheet	TextField.StyleSheet 类（TextField 类的 “内部类”）可用于创建 CSS 文本样式并将其应用到 HTML 或 XML 格式的文本。请参见第 122 页的 “ 使用层叠样式表对文本进行格式设置 ” 和第 181 页的第 12 章 “ 动作脚本字典 ” 中的 TextField.StyleSheet 类条目。
TextFormat	TextFormat 类可用于将格式样式应用于 TextField 对象中的字符或段落。请参见第 121 页的 “ 使用 TextFormat 类 ” 和第 181 页的第 12 章 “ 动作脚本字典 ” 中的 TextFormat 类条目。

媒体类

媒体类提供对 SWF 文件中的声音和视频的回放控制，并且可用于访问用户的麦克风和摄像机（如果安装了这些设备）。这些类位于“动作”工具箱中的“内置类”>“媒体”子文件夹下。

类	说明
摄像头	Camera 类用于访问用户的摄像机（如果安装了此设备）。与 Flash Communication Server MX 一起使用时，SWF 文件可以捕获、广播和录制来自用户摄像机的图像和视频。请参见第 181 页的第 12 章“动作脚本字典”中的 Camera 类 条目。
麦克风	Microphone 类用于访问用户的麦克风（如果安装了此设备）。与 Flash Communication Server MX 一起使用时，SWF 文件可以广播和录制来自用户麦克风的音频。请参见第 181 页的第 12 章“动作脚本字典”中的 Microphone 类 条目。
NetConnection	NetConnection 类用于建立本地流连接，以便从 HTTP 地址或本地文件系统播放 Flash 视频 (FLV) 文件。有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 NetConnection 类 条目。有关通过 Internet 播放 FLV 文件的更多信息，请参见第 174 页的“动态回放外部 FLV 文件”。
NetStream	NetStream 类用于控制 FLV 文件的回放。有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 NetStream 类 条目。有关通过 Internet 播放 FLV 文件的更多信息，请参见第 174 页的“动态回放外部 FLV 文件”。
Sound	Sound 类提供对 SWF 文件中的声音的控制。有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 Sound 类 条目。有关使用 Sound 类创建音量 and 均衡控制器的示例，请参见第 90 页的“创建声音控件”。
Video	Video 类用于播放 SWF 文件中的视频对象。请参见第 181 页的第 12 章“动作脚本字典”中的 Video 类 条目。

客户端 / 服务器类

下表列出的类可用于发送和接收来自外部数据源的数据，或者通过 FTP、HTTP 或 HTTPS 与应用程序服务器进行通讯。

注意：在 Flash Player 7 中，SWF 文件只能加载提供该 SWF 文件的域中的数据。有关更多信息，请参见第 166 页的“Flash Player 安全功能”和第 168 页的“关于允许跨域数据加载”。

这些类位于“动作”面板中的“内置类”>“客户端 / 服务器”子文件夹下。

类	说明
LoadVars	LoadVars 类是 <code>loadVariables()</code> 动作的另一种形式，用于在 SWF 文件和服务器之间以名称 / 值对的形式传输变量。请参见第 159 页的“使用 LoadVars 类”和第 181 页的第 12 章“动作脚本字典”中的 LoadVars 类 条目。
XML	XML 类扩展 XMLNode 类并提供用于处理 XML 格式数据（包括加载和分析外部 XML、新建 XML 文档和导航 XML 文档树）的方法、属性和事件处理函数。请参见第 160 页的“使用 XML 类”和第 181 页的第 12 章“动作脚本字典”中的 XML 类 条目。

类	说明
XMLNode	XMLNode 类表示 XML 文档树中的单个节点。它是 XML 类的超类。请参见第 181 页的第 12 章 “动作脚本字典” 中的 XMLNode 类条目。
XMLSocket	XMLSocket 类可用于创建与其它计算机的永久套接字连接，以实现等待时间较短的数据传输，例如，实时聊天应用程序所需要的快速数据传输。请参见第 163 页的 “使用 XMLSocket 类” 和第 181 页的第 12 章 “动作脚本字典” 中的 XMLSocket 类条目。

创作类

创作类只能在 Flash 创作环境中使用。这些类位于 “动作” 工具箱中的 “内置类” > “创作” 子文件夹下。

类	说明
CustomActionsg	CustomActions 类可用于管理任何用该创作工具注册的自定义动作。请参见第 181 页的第 12 章 “动作脚本字典” 中的 CustomActions 类条目。
实时预览	实时预览功能（虽然不是类，但列在 “动作” 工具箱的 “内置类” 下）提供一个名为 onUpdate 的函数，供组件开发人员使用。请参见第 181 页的第 12 章 “动作脚本字典” 中的 onUpdate。

第 7 章

使用影片剪辑

影片剪辑是自成一体的小型 SWF 文件，这些 SWF 文件互相独立运行且与包含它们的时间轴无关。例如，假设主时间轴只有一个帧，而该帧中的影片剪辑有十个帧，则播放主 SWF 文件时，将播放影片剪辑中的每个帧。影片剪辑还可以包含其它影片剪辑，即嵌套剪辑。以这种方式嵌套的影片剪辑具有层次结构关系，其中父级剪辑 包含一个或多个子级剪辑。

每个影片剪辑实例都具有一个名称，即实例名称，实例名称将实例唯一地标识为可由动作脚本控制的对象。具体而言，实例名称将其标识为 MovieClip 类类型的一个对象。您可以使用 MovieClip 类的属性和方法控制影片剪辑运行时的外观和行为。

您可以将影片剪辑看作自治对象，它可以响应事件、向其它影片剪辑对象发送消息、保持自身状态并管理子级剪辑。通过这种方式，影片剪辑提供了 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中基于组件的体系结构 的基础。事实上，“组件”面板（“窗口” > “开发面板” > “组件”）中提供的组件都是一些复杂的影片剪辑，这些剪辑经过设计和编程从而具有了某种特定外观和行为。有关创建组件的信息，请参见使用组件。

关于通过动作脚本控制影片剪辑

您可以使用全局动作脚本函数或 MovieClip 类的方法对影片剪辑执行任务。某些 MovieClip 方法执行与同名函数相同的任务；而有些 MovieClip 方法，例如 hitTest() 和 swapDepths()，则没有相应的函数名称。

下面的示例说明了使用方法和使用函数之间的差异。这两条语句都重制实例 my_mc，将新剪辑命名为 newClip，然后将它放在第 5 层。

```
my_mc.duplicateMovieClip("newClip", 5);  
duplicateMovieClip("my_mc", "newClip", 5);
```

当函数和方法提供相似的行为时，您可以选择其中任意一个来控制影片剪辑。这种选择取决于您的喜好以及对在动作脚本中撰写脚本的熟悉程度。无论使用函数还是方法，调用函数或方法时，必须将目标时间轴加载到 Flash Player 中。

要使用方法，可通过使用实例名称的目标路径、点和紧跟其后的方法名称和参数来调用它，如下面的语句所示：

```
myMovieClip.play();  
parentClip.childClip.gotoAndPlay(3);
```

在第一条语句中，play() 移动 myMovieClip 实例中的播放头。在第二条语句中，gotoAndPlay() 将 childClip（它是实例 parentClip 的子级实例）中的播放头发送到第 3 帧，然后继续移动播放头。

控制时间轴的全局函数有一个 *target* 参数，可用于指定到所要控制实例的目标路径。例如，在下面的脚本中，`startDrag()` 将 `customCursor` 实例作为目标并使该实例可拖动：

```
on (press) {  
    startDrag("customCursor");  
}
```

以下函数以影片剪辑为操作对象：`loadMovie()`、`unloadMovie()`、`loadVariables()`、`setProperty()`、`startDrag()`、`duplicateMovieClip()` 和 `removeMovieClip()`。要使用这些函数，必须为函数的 *target* 参数输入一个目标路径以指示函数的目标。

下列 `MovieClip` 方法可以控制影片剪辑或加载的级别，它们没有等价的函数：

```
MovieClip.attachMovie(), MovieClip.createEmptyMovieClip(),  
MovieClip.createTextField(), MovieClip.getBounds(), MovieClip.getBytesLoaded(),  
MovieClip.getBytesTotal(), MovieClip.getDepth(), MovieClip.getInstanceAtDepth(),  
MovieClip.getNextHighestDepth(), MovieClip.globalToLocal(),  
MovieClip.localToGlobal(), MovieClip.hitTest(), MovieClip.setMask(),  
MovieClip.swapDepths().
```

有关这些函数和方法的更多信息，请参见第 181 页的第 12 章 “动作脚本字典”。

在单个影片剪辑上调用多个方法

可以使用 `with` 语句只指定一次影片剪辑的地址，然后对该剪辑执行一系列方法。`with` 语句可以用于所有动作脚本对象（如 `Array`、`Color` 和 `Sound`），而不仅仅是影片剪辑。

`with` 语句使用对象作为参数。您指定的对象会添加到当前目标路径的末尾。嵌套在 `with` 语句内的所有动作都将在新目标路径（即范围）内执行。例如，在下面的脚本中，为 `with` 语句传递对象 `donut.hole` 以更改 `hole` 的属性：

```
with (donut.hole){  
    _alpha = 20;  
    _xscale = 150;  
    _yscale = 150;  
}
```

此脚本的行为方式好像 `with` 语句中的语句是从 `hole` 实例的时间轴调用的。上面的代码等同于如下代码：

```
donut.hole._alpha = 20;  
donut.hole._xscale = 150;  
donut.hole._yscale = 150;
```

上面的代码也等同于如下代码：

```
with (donut){  
    hole._alpha = 20;  
    hole._xscale = 150;  
    hole._yscale = 150;  
}
```

加载和卸载其它 SWF 文件

若要在不关闭 Flash Player 的情况下播放其它 SWF 文件或在不加载另一 HTML 页的情况下切换 SWF 文件，您可以使用全局 `loadMovie()` 函数或 `MovieClip` 类的 `loadMovie()` 方法。也可以使用 `loadMovie()` 将变量发送到 CGI 脚本，该脚本会生成一个 SWF 文件作为它的 CGI 输出。当加载 SWF 文件时，可以指定向其中加载 SWF 文件的级别或影片剪辑目标。如果将 SWF 文件加载入目标，加载的 SWF 文件会继承目标影片剪辑的属性。一旦影片加载完毕，就可以更改这些属性。

`unloadMovie()` 方法删除以前由 `loadMovie()` 加载的 SWF 文件。用 `unloadMovie()` 方法明确卸载 SWF 文件可以确保 SWF 文件之间的平滑过渡，并可以减少 Flash Player 所需的内存。

使用 `loadMovie()` 可以执行以下的任何任务：

- 播放一系列格式为 SWF 文件的条幅广告，方法是将 `loadMovie()` 函数放置在每个 SWF 文件的末尾来加载下一个 SWF 文件。
- 开发一个分支界面，使用户可以在多个不同的 SWF 文件中作出选择。
- 构建一个导航界面，其导航控件位于级别 0，用于加载其它级别。加载级别所产生的过渡比在浏览器中加载新 HTML 页面更为平滑。

有关加载影片的更多信息，请参见第 172 页的“加载外部 SWF 和 JPEG 文件”。

指定加载的 SWF 文件的根时间轴

`_root` 动作脚本属性指定或返回对 SWF 文件根时间轴的引用。如果 SWF 文件有多个级别，则根时间轴位于包含当前正在执行的脚本的级别上。例如，如果级别 1 中的脚本计算 `_root`，则返回 `_level1`。但是，`_root` 所指定的时间轴可能有所变化，具体取决于 SWF 文件是独立运行（在它自己的级别中）还是已由 `loadMovie()` 调用加载到影片剪辑实例中。

例如，假设名为 `container.swf` 的文件在其主时间轴上有一个名为 `target_mc` 的影片剪辑实例。`container.swf` 文件在其主时间轴上声明了一个名为 `userName` 的变量；然后相同的脚本将名为 `contents.swf` 的另一个文件加载到影片剪辑 `target_mc` 中。

```
// 在 container.swf 中：
_root.userName = "Tim";
target_mc.loadMovie("contents.swf");
```

加载的 SWF 文件 `contents.swf` 还会在其根时间轴上声明一个名为 `userName` 的变量。

```
// 在 content.swf 中：
_root.userName = "Mary";
```

当 `contents.swf` 加载到 `container.swf` 中的影片剪辑中时，附加到宿主 SWF 文件 (`container.swf`) 的根时间轴上的 `userName` 的值将设置为 "Mary"。这将导致 `container.swf`（以及 `contents.swf`）中的代码出现问题。

若要强制 `_root` 始终指定所加载的 SWF 文件的时间轴，而不是实际的根时间轴，请使用 `_lockroot` 属性。正在实施加载的 SWF 文件或正被加载的 SWF 文件都可以设置此属性。当 `_lockroot` 在影片剪辑实例上设置为 `true` 时，该影片剪辑将充当加载到其中的任何 SWF 文件的 `_root`。当 `_lockroot` 在 SWF 文件中设置为 `true` 时，该 SWF 文件将充当它自己的根，而不管加载它的其它 SWF 文件。任何影片剪辑以及任意数目的影片剪辑都可以将 `_lockroot` 设置为 `true`。默认情况下，此属性为 `false`。

例如，`container.swf` 的创作者可以将以下代码附加到 `target_mc` 影片剪辑：

```
// 附加到 target_mc 影片剪辑：
onClipEvent (load) {
    this._lockroot = true;
}
```

这可以确保在 `contents.swf`（或加载到 `target_mc` 的任何 SWF 文件）中对 `_root` 的引用将引用自己的时间轴，而不是 `container.swf` 的实际根时间轴。

同样，`contents.swf` 的创作者可以将以下代码添加到主时间轴中。

```
// 在 contents.swf 中：
this._lockroot = true;
```

这可以确保无论将 `contents.swf` 加载到什么位置，它对 `_root` 的任何引用都将引用它自己的主时间轴，而不是宿主 SWF 文件的时间轴。

有关更多信息，请参见[第 473 页的 `MovieClip._lockroot`](#)。

将 JPEG 文件加载到影片剪辑中

您可以使用 `loadMovie()` 函数或同名的 `MovieClip` 方法将 JPEG 图像文件加载到影片剪辑实例中。还可以使用 `loadMovieNum()` 函数将 JPEG 文件加载到某个级别。

当您图像加载到影片剪辑中时，图像的左上角将放置在影片剪辑的注册点上。因此此注册点通常位于影片剪辑的中心，所以加载的图像可能不会出现在中心。同时，当将图像加载到根时间轴上时，图像的左上角会被放置在舞台的左上角上。加载的图像会继承影片剪辑的旋转和缩放特性，但会删除影片剪辑的原始内容。

有关更多信息，请参见[第 172 页的“加载外部 SWF 和 JPEG 文件”](#)、[第 385 页的 `loadMovie\(\)`](#)、[第 471 页的 `MovieClip.loadMovie\(\)`](#) 和 [第 386 页的 `loadMovieNum\(\)`](#)。

更改影片剪辑的位置和外观

若要在影片剪辑播放时更改它的属性，可编写一条为属性赋值的语句或使用 `setProperty()` 函数。例如，下面的代码将实例 `mc` 的旋转设置为 45 度：

```
mc._rotation = 45;
```

上面的语句等同于下面使用 `setProperty()` 函数的代码：

```
setProperty("mc", _rotation, 45);
```

有些属性是只读属性，只可以读取但不能设置它们的值。（这些属性在它们的动作脚本字典条目中被指定为只读。）下面是一些只读属性：`_currentframe`、`_droptarget`、`_framesloaded`、`_parent`、`_target`、`_totalframes`、`_url`、`_xmouse` 和 `_ymouse`。

可以编写语句来设置任何非只读的属性。下面的语句设置影片剪辑实例 `wheel` 的 `_alpha` 属性，该实例是 `car` 实例的子级：

```
car.wheel._alpha = 50;
```

此外，可以编写获取影片剪辑的属性值的语句。例如，下面的语句获取当前级别的时间轴上的 `_xmouse` 属性的值，并将 `customCursor` 实例的 `_x` 属性设置为该值：

```
onClipEvent (enterFrame) {
    customCursor._x = _root._xmouse;
}
```

上面的语句等同于下面使用 `getProperty()` 函数的代码：

```
onClipEvent (enterFrame) {  
    customCursor._x = getProperty(_root, _xmouse);  
}
```

`_x`、`_y`、`_rotation`、`_xscale`、`_yscale`、`_height`、`_width`、`_alpha` 和 `_visible` 属性都受影片剪辑的父级变形的影响，并对影片剪辑和该剪辑的所有子级加以变形。`_focusrect`、`_highquality`、`_quality` 和 `_soundbuftime` 属性是全局属性；它们只属于级别 0 主时间轴。所有其它属性都属于每个影片剪辑或加载的级别。

有关影片剪辑属性的列表，请参见第 445 页的“[MovieClip 类的属性概要](#)”。

拖动影片剪辑

可以使用全局 `startDrag()` 函数或 `MovieClip.startDrag()` 方法使影片剪辑可拖动。例如，可以为游戏、拖放功能、自定义界面、滚动条和滑块制作可拖动影片剪辑。

除非用 `stopDrag()` 明确停止或用 `startDrag()` 将另一个影片剪辑作为目标，否则影片剪辑一直是可拖动的。一次只能拖动一个影片剪辑。

若要创建更复杂的拖放行为，可以评估正被拖动的影片剪辑的 `_droptarget` 属性。例如，可以检查 `_droptarget` 属性以查看影片剪辑是否被拖到一个特定的影片剪辑（如“trash can”影片剪辑），然后触发另一个动作。有关详细信息，请参见第 602 页的 `startDrag()` 或第 491 页的 `MovieClip.startDrag()`。

在运行时创建影片剪辑

您不仅可以在 Flash 创作环境中创建影片剪辑实例，还可以在运行时创建它们。动作脚本提供了三种在运行时创建新影片剪辑的方法：

- 通过创建新的空白影片剪辑实例
- 通过重制现有的影片剪辑实例
- 通过将影片剪辑库中元件的实例附加到舞台

运行时创建的每个影片剪辑实例必须具有实例名称和深度（堆叠或 z 顺序）值。您指定的深度将决定新剪辑与同一时间轴上的其它剪辑重叠的方式。（请参见第 114 页的“[管理影片剪辑的深度](#)”。）

创建空白影片剪辑

若要在舞台上创建一个空白影片剪辑，可使用 `MovieClip` 类的 `createEmptyMovieClip()` 方法。该方法将创建一个影片剪辑，作为调用该方法的剪辑的子级。新创建的空白影片剪辑的注册点为左上角。

例如，下面的代码在名为 `parent_mc` 的影片剪辑中创建一个名为 `new_mc` 的新的子级影片剪辑，新剪辑的深度为 10。

```
parent_mc.createEmptyMovieClip("new_mc", 10);
```

下面的代码在运行脚本的 SWF 文件的根时间轴上创建一个名为 `canvas_mc` 的新影片剪辑，然后调用 `loadMovie()` 将一个外部 JPEG 文件加载到自身中。

```
_root.createEmptyMovieClip("canvas_mc", 10);  
canvas_mc.loadMovie("flowers.jpg");
```

有关更多信息，请参见第 454 页的 `MovieClip.createEmptyMovieClip()`。

重制或删除影片剪辑

若要重制或删除影片剪辑实例，可使用 `duplicateMovieClip()` 或 `removeMovieClip()` 全局函数或同名的 `MovieClip` 类方法。`duplicateMovieClip()` 方法创建现有影片剪辑实例的新实例，为其指定一个新实例名称并指定一个深度（即 z 顺序）。重制的影片剪辑始终从第 1 帧开始，即使原始影片剪辑在重制时位于其它帧也如此，并且，重制的影片剪辑始终位于时间轴上所有以前定义的影片剪辑的前面。

若要删除使用 `duplicateMovieClip()` 创建的影片剪辑，可使用 `removeMovieClip()`。如果删除了父级影片剪辑，那么重制的影片剪辑也会被删除。

有关更多信息，请参见第 341 页的 `duplicateMovieClip()` 和第 561 页的 `removeMovieClip()`。

将影片剪辑元件附加到舞台

在运行时创建影片剪辑实例的最后一种方法是使用 `attachMovie()`。`attachMovie()` 方法将 SWF 文件库中影片剪辑元件的实例附加到舞台上。新剪辑将成为附加它的剪辑的子级剪辑。

若要使用动作脚本从库中附加一个影片剪辑元件，必须为动作脚本导出该元件并为其指定一个唯一的链接标识符。为此，可以使用“链接属性”对话框。

默认情况下，为用于动作脚本而导出的所有影片剪辑都将在包含它们的 SWF 文件的第一帧之前加载。这可能会造成在第一帧播放之前出现延迟。当为某个元素指定链接标识符时，也可以指定是否在第一帧之前加载该内容。如果没有将该元素加载到第一帧中，则必须将它的实例包含在 SWF 文件的其它某个帧中；如果不包含它，则无法将该元素导出到 SWF 文件中。

将链接标识符分配给影片剪辑：

- 1 选择“窗口” > “库”以打开“库”面板。
- 2 在“库”面板中选择一个影片剪辑。
- 3 在“库”面板中，从“库”面板选项菜单中选择“链接”。
即可出现“链接属性”对话框。
- 4 对于“链接”，选择“为动作脚本导出”。
- 5 对于“标识符”，输入影片剪辑的 ID。
默认情况下，标识符与元件名称相同。
- 6 您还可以将动作脚本 2.0 的类分配给影片剪辑元件。（请参见第 116 页的“将类分配给影片剪辑元件”。）
- 7 如果不想在第一帧之前加载影片剪辑，则取消选择“在第一帧导出”选项。
如果取消选择此选项，则将影片剪辑的实例放置在时间轴帧上的所需位置。例如，如果您编写的脚本直到第 10 帧才引用该影片剪辑，则将该元件的实例放置在时间轴的该帧上或该帧之前。
- 8 单击“确定”。

将链接标识符分配给影片剪辑后，可以使用 `attachMovie()` 在运行时将元件的实例附加到舞台上。

将影片剪辑附加到另一个影片剪辑：

- 1 如上所述将链接标识符分配给影片剪辑库元件。
- 2 在“动作”面板（“窗口” > “开发面板” > “动作”）打开时，在时间轴中选择一个帧。
- 3 在“动作”面板的“脚本”窗格中，键入影片剪辑的名称或要向其附加新影片剪辑的级别。
例如，若要将影片剪辑附加到根时间轴，则键入 `_root`。

- 4 在“动作”工具箱（在“动作”面板的左侧）中，依次单击“内置类”类别、“影片”类别和“影片剪辑”类别，然后双击 `attachMovie()`。
- 5 参照出现的代码提示，输入以下参数的值：
 - 对于 `idName`，指定在“链接属性”对话框中输入的标识符。
 - 对于 `newName`，输入附加剪辑的实例名称，以便您能够将它作为目标。
 - 对于 `depth`，输入重制的影片剪辑附加到影片剪辑的级别。每个附加的影片剪辑都有它自己的堆叠顺序，其中级别 0 是起源影片剪辑所在的级别。附加的影片剪辑始终位于原始影片剪辑的上面。下面是一个示例：

```
myMovieClip.attachMovie("calif", "california", 10);
```

有关更多信息，请参见第 449 页的 `MovieClip.attachMovie()`。

将参数添加到动态创建的影片剪辑

使用 `MovieClip.attachMovie()` 和 `MovieClip.duplicateMovieClip()` 动态创建或重制影片剪辑时，您可以用另一对象中的参数填充影片剪辑。`attachMovie()` 和 `duplicateMovieClip()` 的 `initObject` 参数允许动态创建的影片剪辑接收剪辑参数。`initObject` 参数是可选的。

有关更多信息，请参见第 449 页的 `MovieClip.attachMovie()` 和第 458 页的 `MovieClip.duplicateMovieClip()`。

若要用指定对象中的参数填充动态创建的影片剪辑，请执行以下操作之一：

- 使用以下包含 `attachMovie()` 的语法：

```
myMovieClip.attachMovie(idName, newName, depth [, initObject])
```
 - 使用以下包含 `duplicateMovieClip()` 的语法：

```
myMovieClip.duplicateMovieClip(idName, newName, depth [, initObject])
```
- `initObject` 参数指定您要其参数填充动态创建的影片剪辑的对象的名称。

使用 `attachMovie()` 用参数填充影片剪辑：

- 1 在新的 Flash 文档中，通过选择“插入”>“新建元件”创建一个影片剪辑元件。在“元件名称”文本框中键入 `dynamic`，然后选择影片剪辑行为。
- 2 在该元件内部，在舞台上创建一个实例名称为 `name_txt` 的动态文本字段。
- 3 选择影片剪辑的时间轴的第一帧，然后打开“动作”面板（“窗口”>“开发面板”>“动作”）。
- 4 创建一个名为 `name` 的新变量，然后将其值分配给 `name_txt` 的 `text` 属性，如下所示：

```
var name:String;
name_txt.text = name;
```
- 5 选择“编辑”>“编辑文档”返回到主时间轴。
- 6 在库中选择影片剪辑元件，然后从“库”面板选项菜单中选择“链接属性”。即可出现“链接属性”对话框。
- 7 选择“为动作脚本导出”选项，然后单击“确定”。
- 8 选择主时间轴的第一帧并将以下代码添加到“动作”面板的“脚本”窗格中：

```
_root.attachMovie("dynamic", "newClipName", 10, {name:"Erick"});
```
- 9 测试影片（“控制”>“测试影片”）。您在 `attachMovie()` 调用中指定的名称将出现在新影片剪辑的文本字段中。

管理影片剪辑的深度

每个影片剪辑都具有自身的 z 顺序空间，z 顺序空间确定各对象在其父级 SWF 文件或影片剪辑中的重叠方式。每个影片剪辑都有一个关联的深度值，该值用来确定它是显示在同一个影片剪辑时间轴中其它影片剪辑的前面还是后面。当使用 `MovieClip.attachMovie()`、`MovieClip.duplicateMovieClip()` 或 `MovieClip.createEmptyMovieClip()` 在运行时创建影片剪辑时，总是要以方法参数的形式为新剪辑指定一个深度。例如，以下代码将一个新影片剪辑附加到名为 `container_mc` 的影片剪辑的时间轴，其深度值为 10。

```
container_mc.attachMovie("symbolID", "clip_1", 10);
```

这将在 `container_mc` 的 z 顺序空间中创建一个深度为 10 的新影片剪辑。

例如，以下代码将两个新影片剪辑附加到 `container_mc`。第一个剪辑（名为 `clip_1`）将显示在 `clip_2` 的后面，这是因为向第一个剪辑分配了一个较小的深度值。

```
container_mc.attachMovie("symbolID", "clip_1", 10);
container_mc.attachMovie("symbolID", "clip_2", 15);
```

影片剪辑的深度值的范围是从 -16384 到 1048575。

`MovieClip` 类提供了多种管理影片剪辑深度的方法：请参见第 463 页的 `MovieClip.getNextHighestDepth()`、第 462 页的 `MovieClip.getInstanceAtDepth()`、第 462 页的 `MovieClip.getDepth()` 和第 493 页的 `MovieClip.swapDepths()`。

确定下一个最大的可用深度

若要确定影片剪辑中下一个最大的可用深度，请使用 `MovieClip.getNextHighestDepth()`。此方法返回一个整数值指示下一个可用的深度，该深度的对象将显示在影片剪辑中所有其它对象的前面。

以下代码在名为 `menus_mc` 的影片剪辑的时间轴上创建一个新影片剪辑，其深度值为 10。然后，它确定同一影片剪辑中下一个最大的可用深度，并在该深度创建一个新的影片剪辑。

```
menus_mc.attachMovie("menuClip", "file_menu", 10);
var nextDepth = menus_mc.getNextHighestDepth();
menus_mc.attachMovie("menuClip", "edit_menu", nextDepth);
```

在本例中，名为 `nextDepth` 的变量包含值 11，因为这是影片剪辑 `menus_mc` 的下一个最大的可用深度。

若要获得当前占用的最大深度，从 `getNextHighestDepth()` 返回的值中减 1 即可得到，如下一节所示（请参见第 114 页的“确定处于特定深度的实例”）。

确定处于特定深度的实例

若要确定处于特定深度的实例，请使用 `MovieClip.getInstanceAtDepth()`。此方法返回对处于指定深度的 `MovieClip` 实例的引用。

以下代码结合使用 `getNextHighestDepth()` 和 `getInstanceAtDepth()`，确定处于根时间轴上已占用的最大（当前）深度的影片剪辑。

```
var highestOccupiedDepth = _root.getNextHighestDepth() - 1;
var instanceAtHighestDepth = _root.getInstanceAtDepth(highestOccupiedDepth);
```

有关更多信息，请参见第 462 页的 `MovieClip.getInstanceAtDepth()`。

确定实例的深度

若要确定影片剪辑实例的深度，请使用 `MovieClip.getDepth()`。

以下代码对 SWF 文件主时间轴上的所有影片剪辑进行迭代处理，并在“输出”面板中显示每个剪辑的实例名称和深度值。

```
for each in _root {  
    var obj = _root[each];  
    if (obj instanceof MovieClip) {  
        var objDepth = obj.getDepth();  
        trace(obj._name + ":" + objDepth)  
    }  
}
```

有关更多信息，请参见[第 462 页的 MovieClip.getDepth\(\)](#)。

交换影片剪辑的深度

若要交换同一个时间轴上两个影片剪辑的深度，请使用 `MovieClip.swapDepths()`。有关更多信息，请参见[第 493 页的 MovieClip.swapDepths\(\)](#)。

用动作脚本绘制形状

您可以使用 `MovieClip` 类的方法在舞台上绘制线条和填充。这样您就可以为用户创建绘画工具，并且可以在影片中绘制响应事件的形状。绘画方法包括 `beginFill()`、`beginGradientFill()`、`clear()`、`curveTo()`、`endFill()`、`lineTo()`、`lineStyle()` 和 `moveTo()`。

可以在任何一个影片剪辑中使用绘画方法。不过，如果对在创作模式下创建的影片剪辑使用绘画方法，则该绘画方法会在绘制该剪辑之前执行。换句话说，在创作模式下创建的内容会绘制在用绘画方法绘制的内容上面。

您可以将带有绘画方法的影片剪辑作为遮罩使用，不过，与所有影片剪辑遮罩一样，都会忽略笔触。

绘制形状：

- 1 使用 `createEmptyMovieClip()` 可以在舞台上创建一个空的影片剪辑。

新影片剪辑是现有影片剪辑或主时间轴的子级，如下面的示例所示：

```
_root.createEmptyMovieClip( "triangle", 1 );
```

- 2 使用空影片剪辑调用绘画方法。

下面的示例将绘制一个轮廓为 5 磅粗细的洋红色线条但没有填充的三角形：

```
with (_root.triangle) {  
    lineStyle (5, 0xff00ff, 100);  
    moveTo (200, 200);  
    lineTo (300, 300);  
    lineTo (100, 300);  
    lineTo (200, 200);  
}
```

有关这些方法的详细信息，请参见[第 181 页的第 12 章“动作脚本字典”](#)中的相应条目。

将影片剪辑用作遮罩

可以将影片剪辑用作遮罩，创建一个孔洞，透过它可以看到另一个影片剪辑的内容。遮罩影片剪辑和普通的影片剪辑一样，播放时间轴中的所有帧。您可以使遮罩影片剪辑成为可拖动的、让它沿着运动引导层运动、在单个遮罩内使用单独的形状，也可以动态调整遮罩的大小。您还可以使用动作脚本打开和关闭遮罩。

不能使用遮罩遮蔽另一个遮罩。不能设置遮罩影片剪辑的 `_alpha` 属性。只有填充可以用在作为遮罩的影片剪辑中；笔触都会被忽略。

创建遮罩：

- 1 在舞台上，选择一个要被遮蔽的影片剪辑。
- 2 在属性检查器中，输入该影片剪辑的实例名称，例如 `image`。
- 3 创建一个要用作遮罩的影片剪辑。在属性检查器中，为其指定一个实例名称，如 `mask`。
在充当遮罩的影片剪辑的所有不透明区域下，被遮蔽的影片剪辑将会显示出来。
- 4 在时间轴中选择第 1 帧。
- 5 如果“动作”面板（“窗口” > “开发面板” > “动作”）尚未打开，请打开该面板。
- 6 在“动作”面板中输入以下代码：

```
image.setMask(mask);
```

有关详细信息，请参见第 490 页的 `MovieClip.setMask()`。

关于遮蔽设备字体

您可以使用影片剪辑遮蔽设置为设备字体的文本。为了使用于设备字体的影片剪辑遮罩正常工作，用户必须具有 Flash Player 6 版本 40 或更高版本。

当使用影片剪辑遮蔽设置为设备字体的文本时，遮罩的矩形边框将被用作遮罩形状。也就是说，如果您在 Flash 创作环境中为设备字体文本创建非矩形的影片剪辑遮罩，显示在 SWF 文件中的遮罩形状将是遮罩的矩形边框的形状，而不是遮罩本身的形状。

您只能通过将影片剪辑用作遮罩来遮蔽设备字体。不能通过在舞台上使用遮罩层来遮蔽设备字体。

处理影片剪辑事件

影片剪辑可以对用户事件（例如鼠标单击和按键）及系统级事件（例如在舞台上初次加载影片剪辑）进行响应。动作脚本提供两种处理影片剪辑事件的方法：一是通过事件处理函数方法，二是通过 `onClipEvent()` 和 `on()` 事件处理函数。有关更多信息，请参见第 75 页的第 4 章“处理事件”。

将类分配给影片剪辑元件

使用动作脚本 2.0，您可以创建自己的类来扩展内置 `MovieClip` 类的行为，然后使用“链接属性”对话框将所创建的类分配给一个影片剪辑库元件。在您创建分配了类的影片剪辑的实例后，该剪辑实例将使用由分配给它的类所定义的属性和行为。（有关动作脚本 2.0 的更多信息，请参见第 137 页的第 9 章“使用动作脚本 2.0 创建类”。）

在 `MovieClip` 类的子类中，您可以为内置的 `MovieClip` 方法和事件处理函数提供方法定义，如 `onEnterFrame` 和 `onRelease`。在下面的过程中，您将创建一个名为 `MoveRight` 的类，该类扩展 `MovieClip` 类并定义一个 `onPress` 处理函数（该函数在用户单击影片剪辑时将剪辑向右移动 20 个像素）。在第二个过程中，您将在新的 Flash (FLA) 文档中创建一个影片剪辑元件，并将 `MoveRight` 类分配给该元件。

创建影片剪辑子类：

- 1 创建一个名为 BallTest 的新目录。
- 2 执行下列操作之一，创建一个新的动作脚本文件：
 - (Flash MX Professional 2004) 选择 “文件” > “新建”，然后从文档类型列表中选择 “动作脚本文件”。
 - (Flash MX 2004) 使用自己常用的文本编辑器创建一个文本文件。
- 3 在脚本中输入以下代码：

```
// MoveRight 类 -- 将剪辑的每个帧向右移动 5 个像素
class MoveRight extends MovieClip {
    function onPress() {
        this._x += 20;
    }
}
```

- 4 在 BallTest 目录中将该文档保存为 MoveRight.as。

将类分配给影片剪辑元件：

- 1 在 Flash 中，选择 “文件” > “新建”，从文件类型列表中选择 “Flash 文档”，然后单击 “确定”。
- 2 使用椭圆工具在舞台上绘制一个圆形。
- 3 选择所绘圆形，然后选择 “修改” > “转换为元件”。在 “转换为元件” 对话框中，选择 “影片剪辑” 作为元件的行为，并在 “名称” 文本框中输入 Ball。
- 4 打开 “库” 面板 (“窗口” > “库”)，然后选择 Ball 元件。
- 5 在 “库” 面板选项菜单中选择 “链接” 以打开 “链接属性” 对话框。
- 6 在 “链接属性” 对话框中，选择 “为动作脚本导出” 选项，然后在 “AS 2.0 类” 文本框中键入 MoveRight。单击 “确定”。
- 7 在 BallTest 目录 (包含 MoveRight.as 文件的同一个目录) 中将该文件保存为 Ball fla。
- 8 测试影片 (“控制” > “测试影片”)。

每次单击球形影片剪辑时，它都会向右移动 20 个像素。

初始化类属性

在前面介绍的示例中，您将 Ball 元件的实例手动添加到舞台上 (即在创作过程中完成此操作)。正如以前所讨论的 (请参见第 113 页的 [“将参数添加到动态创建的影片剪辑”](#))，可以使用 attachMovie() 和 duplicateMovie() 的 *initObject* 参数在运行时为创建的剪辑指定参数。您可以使用此功能初始化分配到影片剪辑的类的属性。

例如，下面这个名为 MoveRightDistance 的类是以前讨论的 MoveRight 类 (请参见第 116 页的 [“将类分配给影片剪辑元件”](#)) 的变体。两者的区别在于 MoveRightDistance 类有一个名为 distance 的新属性，该属性值确定每次单击影片剪辑时该影片剪辑移动多少像素。

```
// MoveRightDistance 类 -- 将剪辑的每个帧向右移动 5 个像素
class MoveRightDistance extends MovieClip {
    // distance 属性确定每次
    // 按下鼠标按钮时将剪辑移动多少像素
    var distance:Number;
    function onPress() {
        this._x += distance;
    }
}
```

假设将此类分配给具有链接标识符 Ball 的元件，以下代码将在 SWF 文件的根时间轴上创建该元件的两个新实例。第一个实例（名为 ball_50）每次单击时移动 50 个像素；第二个实例（名为 ball_125）每次单击时移动 125 个像素。

```
_root.attachMovie("Ball", "ball_50", 10, {distance:50});  
_root.attachMovie("Ball", "ball_125", 20, {distance:125});
```

第 8 章

使用文本

动态或输入文本字段是一个 TextField 对象（TextField 类的实例）。当您创建文本字段时，可以在属性检查器中给它指定一个实例名称。您可以在动作脚本语句中使用该实例名称通过 TextField 和 TextFormat 类来设置、更改该文本字段及其内容并设置其格式。

TextField 类的方法允许您设置、选择并操控在创作过程中或运行时创建的动态或输入文本字段中的文本。有关更多信息，请参见第 119 页的“使用 TextField 类”。有关在运行时调试文本字段的信息，请参见第 70 页的“显示文本字段属性以进行调试”。

动作脚本还提供了多种在运行时对文本进行格式设置的方法。TextFormat 类允许您设置 TextField 对象的字符和段落格式（请参见第 121 页的“使用 TextFormat 类”）。Flash Player 还支持部分 HTML 标签，可以使用这些 HTML 标签对文本进行格式设置（请参见第 130 页的“使用 HTML 格式的文本”）。Flash Player 7 和更高版本支持 HTML 标签，该标签不仅允许您嵌入外部图像，还可嵌入外部 SWF 文件以及驻留在库中的影片剪辑（请参见第 131 页的“图像标签 ()”）。

在 Flash Player 7 和更高版本中，您可以使用 TextField.StyleSheet 类将层叠样式表 (CSS) 样式应用于文本字段。您可以使用 CSS 设置内置 HTML 标签的样式、定义新的格式设置标签或应用样式。有关使用 CSS 的更多信息，请参见第 122 页的“使用层叠样式表对文本进行格式设置”。

您还可以将 HTML 格式化文本（该文本可以选择使用 CSS 样式）直接分配给文本字段。在 Flash Player 7 和更高版本中，分配给文本字段的 HTML 文本可以包含嵌入的媒体（影片剪辑、SWF 文件和 JPEG 文件）。文本将在嵌入的媒体旁换行，就像 Web 浏览器让文本在 HTML 文档中嵌入的媒体旁换行一样。有关更多信息，请参见第 131 页的“图像标签 ()”。

使用 TextField 类

TextField 类表示您使用 Flash 中的“文本”工具创建的任何动态或可选择（可编辑）文本字段。使用此类的方法和属性在运行时控制文本字段。TextField 对象支持与 MovieClip 对象相同的属性，_currentframe、_droptarget、_framesloaded 和 _totalframes 属性除外。您可以动态地获取并设置属性及调用文本字段的方法。

若要使用动作脚本控制动态或输入文本字段，您必须在属性检查器中为其指定一个实例名称。然后，您可以使用该实例名称引用文本字段，并使用 TextField 类的方法和属性控制文本字段的内容或基本外观。您还可以使用 MovieClip.createTextField() 方法在运行时创建 TextField 对象并为它们指定实例名称。有关更多信息，请参见第 120 页的“在运行时创建文本字段”。

在运行时将文本分配到文本字段

若要将文本分配到文本字段，请使用 `TextField.text` 属性。

在运行时将文本分配到文本字段：

- 1 使用文本工具在舞台上创建一个文本字段。
- 2 选中文本字段的同时，在属性检查器（“窗口” > “属性”）中，在检查器左侧“文本类型”弹出式菜单正下方的“实例名称”文本框中输入 `headline_txt`。
实例名称只能包含字母、下划线（`_`）和美元符号（`$`）。
- 3 在时间轴中，选择图层 1 中的第 1 帧并打开“动作”面板（“窗口” > “开发面板” > “动作”）。
- 4 在“动作”面板中键入以下代码：

```
headline_txt.text = "Brazil wins World Cup";
```
- 5 选择“控制” > “测试影片”对影片进行测试。

关于文本字段实例和变量名称

在属性检查器中，除实例名称外，您还可以为动态或输入文本字段指定一个变量名称。随后可以在动作脚本中引用文本字段的变量名称，其值确定文本字段的内容。但是，不能混淆文本字段的实例名称和变量名称。

使用为文本字段指定的实例名称调用方法、在该文本字段上获取并设置属性。文本字段的变量名称只是对该文本字段所包含文本的变量引用，它不是对对象的引用。

例如，如果您为某个文本字段指定变量名称 `myTextVar`，则随后可以使用以下代码设置该文本字段的内容：

```
var myTextVar = "This is what will appear in the text field";
```

但是，您不能使用 `myTextVar` 变量将同一个文本字段的文本属性设置为某文本。

```
// 这将不起作用  
myTextVar.text = "A text field variable is not an object reference";
```

通常情况下，使用 `TextField.text` 属性控制文本字段的内容，除非面向的是不支持 `TextField` 类的 Flash Player 版本。这将减小变量名称冲突的可能性，变量名称冲突可能导致运行时出现意外情况。

在运行时创建文本字段

可以使用 `MovieClip` 类的 `createTextField()` 方法于运行时在舞台上创建一个空文本字段。新文本字段会被附加到调用该方法的影片剪辑的时间轴上。`createTextField()` 方法使用以下语法：

```
movieClip.createTextField(instanceName, depth, x, y, width, height)
```

例如，以下代码在点 (0,0) 且深度 (z 顺序) 为 10 的位置创建一个名为 `test_txt` 的 300 x 100 像素的文本字段。

```
_root.createTextField("test_txt", 10, 0, 0, 300, 100);
```

您可以使用 `createTextField()` 调用中指定的实例名称访问 `TextField` 类的方法和属性。例如，以下代码创建一个名为 `test_txt` 的新文本字段，然后修改其属性使之成为一个多行、自动换行的文本字段，该文本字段在插入文本时可以进行扩展。最后，该代码将一些文本分配到文本字段的 `text` 属性。


```
_root.createTextField("test_txt", 10, 0, 0, 100, 50);
test_txt.multiline = true;
test_txt.wordWrap = true;
test_txt.autoSize = true;
test_txt.text = "Create new text fields with the MovieClip.createTextField
method.";
```

您可以使用 `TextField.removeTextField()` 方法删除用 `createTextField()` 创建的文本字段。`removeTextField()` 方法对创作过程中由时间轴放置的文本字段不起作用。

有关更多信息，请参见[第 454 页的 `MovieClip.createTextField\(\)`](#) 和[第 653 页的 `TextField.removeTextField\(\)`](#)。

使用 TextFormat 类

您可以使用动作脚本的 `TextFormat` 类来设置文本字段的格式设置属性。`TextFormat` 类包含有关字符格式和段落格式的信息。字符格式信息描述单个字符的外观：字体名称、磅值、颜色和关联的 URL。段落格式信息描述段落的外观：左边距、右边距、首行缩进、左对齐、右对齐或居中对齐。

若要使用 `TextFormat` 类，首先创建一个 `TextFormat` 对象并设置其字符和段落格式样式。然后使用 `TextField.setTextFormat()` 或 `TextField.setNewTextFormat()` 方法将 `TextFormat` 对象应用于文本字段。

`setTextFormat()` 方法更改应用于文本字段中单个字符、字符组或整体文本的文本格式。但是，新插入的文本（例如用户输入的文本或通过动作脚本插入的文本）不采用 `setTextFormat()` 调用指定的格式设置。若要指定新插入文本的默认格式设置，请使用 `TextField.setNewTextFormat()`。有关更多信息，请参见[第 656 页的 `TextField.setTextFormat\(\)`](#) 和[第 656 页的 `TextField.setNewTextFormat\(\)`](#)。

通过 `TextFormat` 类对文本字段进行格式设置：

- 1 在新的 Flash 文档中，使用“文本”工具在舞台上创建一个文本字段。在舞台上文本字段中键入一些文本，例如“Bold、italic、24 point text”。
- 2 在属性检查器中，在“实例名称”文本框中键入 `myText_txt`，从“文本类型”弹出式菜单中选择“动态”，然后从“线条类型”弹出式菜单中选择“多行”。
- 3 在时间轴中，选择图层 1 中的第 1 帧并打开“动作”面板（“窗口”>“开发面板”>“动作”）。
- 4 在“动作”面板中输入以下代码创建一个 `TextFormat` 对象，然后将其 `bold` 和 `italic` 属性设置为 `true`，将其 `size` 属性设置为 24。

```
// 创建 TextFormat 对象
var txtfmt_fmt = new TextFormat();
// 指定段落格式和字符格式
txtfmt_fmt.bold = "true";
txtfmt_fmt.italic = "true";
txtfmt_fmt.size = "24"
```

- 5 使用 `TextField.setTextFormat()` 将 `TextFormat` 对象应用于您在第 1 步中创建的文本字段。`myText_txt.setTextFormat(txtfmt_fmt)`;

此版本的 `setTextFormat()` 将指定的格式设置应用于整个文本字段。此方法还有另外两个版本，用于将格式设置应用于单个字符或字符组。例如，以下代码将粗体、斜体和 24 磅格式设置应用于您在文本字段中输入的前四个字符。

```
myText_txt.setTextFormat(txtfmt_fmt, 0, 3);
```

有关更多信息，请参见[第 656 页的 `TextField.setTextFormat\(\)`](#)。

- 6 选择“控制”>“测试影片”对影片进行测试。

新文本字段的默认属性

运行时使用 `createTextField()` 创建的文本字段接收具有以下属性的默认 `TextFormat` 对象：

```
font = "Times New Roman"
size = 12
textColor = 0x000000
bold = false
italic = false
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
bullet = false
tabStops = [] (empty array)
```

有关 `TextFormat` 方法及其说明的完整列表，请参见第 181 页的第 12 章“动作脚本字典”中的 `TextFormat` 类条目。

获取文本测量信息

您可以使用 `TextFormat.getTextExtent()` 方法获得应用了特定格式设置的文本字符串的详细文本测量信息。例如，假设您需要在运行时创建一个包含任意数目文本的新 `TextField` 对象，所包含文本的格式设置为 24 磅、粗体、Arial 字体以及 5 个像素的缩进。您需要确定新 `TextField` 对象必须有多宽或多高才能显示所有文本。`getTextExtent()` 方法提供测量信息，例如上升、下降、宽度和高度。

有关更多信息，请参见第 675 页的 `TextFormat.getTextExtent()`。

使用层叠样式表对文本进行格式设置

层叠样式表是创建可应用于 HTML 或 XML 文档的文本样式的机制。样式表是格式设置规则的集合，这些规则指定如何对 HTML 或 XML 元素进行格式设置。每个规则都将一个样式名称（即选择器）与一个或多个样式属性及其值关联起来。例如，以下样式定义名为 `bodyText` 的选择器。

```
bodyText { text-align:left }
```

您可以创建重新定义 Flash Player 所使用的内置 HTML 格式标签（例如 `<p>` 和 ``）的样式、使用 `<p>` 或 `` 标签的 `class` 属性创建可应用于特定 HTML 元素的样式“类”，或者定义新标签。

您可以使用 `TextField.StyleSheet` 类处理文本样式表。您可以从外部 CSS 文件加载样式或使用动作脚本直接创建样式。若要将样式表应用于包含 HTML 或 XML 格式文本的文本字段，可以使用 `TextField.styleSheet` 属性。在样式表中定义的样式将被自动映射到 HTML 或 XML 文档中定义的标签。

使用样式表涉及三个基本步骤：

- 从 `TextField.StyleSheet` 类创建样式表对象。请参见第 123 页的“创建样式表对象”。
- 通过从外部 CSS 文件导入样式或使用动作脚本定义样式，将样式添加到样式表对象。请参见第 124 页的“加载外部 CSS 文件”和第 125 页的“使用动作脚本创建新样式”。

- 将样式表对象分配到包含 XML 或 HTML 格式文本的文本字段。请参见第 125 页的“将样式应用于 TextField 对象”、第 127 页的“将样式用于 HTML 的示例”和第 129 页的“将样式用于 XML 的示例”。

支持的 CSS 属性

Flash Player 支持原 CSS1 规范 (www.w3.org/TR/REC-CSS1) 中的一部分属性。下表显示支持的 CSS 属性和值，以及它们对应的动作脚本属性名称。（每个动作脚本属性名称都是从对应的 CSS 属性名称产生的；省略连字符并将连字符后的字符变成大写。）

CSS 属性	动作脚本属性	用法和-supported 值
text-align	textAlign	可以识别的值包括 left、center 和 right。
font-size	fontSize	只使用该值的数字部分；不分析单位（px、pt）；像素和磅是等价的。
text-decoration	textDecoration	可以识别的值包括 none 和 underline。
margin-left	marginLeft	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
margin-right	marginRight	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
font-weight	fontWeight	可以识别的值包括 normal 和 bold。
font-style	fontStyle	可以识别的值包括 normal 和 italic。
text-indent	textIndent	只使用该值的数字部分。不分析单位（px、pt）；像素和磅是等价的。
font-family	fontFamily	用逗号分隔的供使用字体的列表，按需求降序排列。可以使用任何字体系列名称。如果您指定通用字体名称，则它将转换为相应的设备字体。支持以下字体转换：mono 转换为 _typewriter、sans-serif 转换为 _sans 以及 serif 转换为 _serif。
color	color	只支持十六进制颜色值。不支持命名的颜色（例如 blue）。
display	display	支持的值包括 inline、block 和 none。

创建样式表对象

在动作脚本中，CSS 样式表由 TextField.StyleSheet 类表示。此类只用于面向 Flash Player 7 或更高版本的 SWF 文件。若要创建样式表对象，可以调用 TextField.StyleSheet 类的构造函数。

```
var newStyle = new TextField.StyleSheet();
```

若要将样式添加到样式表对象中，您可以将外部 CSS 文件加载到对象中，也可以在动作脚本中定义样式。请参见第 124 页的“加载外部 CSS 文件”和第 125 页的“使用动作脚本创建新样式”。

加载外部 CSS 文件

您可以在外部 CSS 文件中定义样式，然后将该文件加载到样式表对象中。在 CSS 文件中定义的样式会被添加到样式表对象中。若要加载外部 CSS 文件，请使用 `TextField.StyleSheet` 类的 `load()` 方法。若要确定 CSS 文件何时完成加载，请使用样式表对象的 `onLoad` 事件处理函数。

在下面的示例中，您将创建并加载一个外部 CSS 文件并使用 `TextField.StyleSheet.getStyleNames()` 方法检索所加载样式的名称。

加载外部样式表：

- 1 在喜欢使用的文本或 XML 编辑器中，创建一个文件。
- 2 将以下样式定义添加到该文件中：

```
// 文件名：styles.css
bodyText {
    font-family:Arial,Helvetica,sans-serif;
    font-size:12px;
}

headline {
    font-family:Arial,Helvetica,sans-serif;
    font-size:24px;
}
```

- 3 将该 CSS 文件另存为 `styles.css`。
- 4 在 Flash 中，创建一个 FLA 文档。
- 5 在时间轴（“窗口” > “时间轴”）中，选择“图层 1”。
- 6 打开“动作”面板（“窗口” > “开发面板” > “动作”）。
- 7 将以下代码添加到“动作”面板中：

```
var css_styles = new TextField.StyleSheet();
css_styles.load("styles.css");
css_styles.onLoad = function(ok) {
    if (ok) {
        // 显示样式名称
        trace(this.getStyleNames());
    } else {
        trace("Error loading CSS file.");
    }
}
```

- 8 将该文件保存到包含 `styles.css` 的同一个目录中。
- 9 测试影片（“控制” > “测试影片”）。

您应该看到两个样式的名称显示在“输出”面板中：

```
body
headLine
```

如果看到“输出”面板中显示“加载 CSS 文件出错”，则确保 FLA 文件和 CSS 文件在同一个目录中并且您正确键入了 CSS 文件的名称。

对于通过网络加载数据的所有其它动作脚本方法，CSS 文件必须与加载文件的 SWF 文件驻留在同一个域中。（请参见第 168 页的“关于允许跨域数据加载”。）

使用动作脚本创建新样式

您可以通过动作脚本使用 `TextField.StyleSheet` 类的 `setStyle()` 方法创建新的文本样式。此方法采用两个参数：样式的名称以及定义该样式属性的对象。

例如，以下代码创建名为 `styles` 的样式表对象，该对象定义两个样式，这两个样式与您先前导入的样式相同（请参见第 124 页的“加载外部 CSS 文件”）。

```
var styles = new TextField.StyleSheet();
styles.setStyle("bodyText",
    {fontFamily:'Arial,Helvetica,sans-serif',
      fontSize:'12px'}
);
styles.setStyle("headline",
    {fontFamily:'Arial,Helvetica,sans-serif',
      fontSize:'24px'}
);
```

将样式应用于 TextField 对象

若要将样式表对象应用于文本字段，请将样式表对象分配到文本字段的 `styleSheet` 属性。

```
textObj_txt.styleSheet = styleSheetObj;
```

注意：请注意，不要混淆 `TextField.styleSheet` 属性和 `TextField.StyleSheet` 类。大写字母指示了它们的区别。

当将样式表对象分配到 `TextField` 对象时，文本字段的正常行为将发生以下更改：

- 文本字段的 `text` 和 `htmlText` 属性以及与文本字段关联的任何变量总是包含相同的值并具有相同的行为方式。
- 文本字段变为只读，用户不能对其进行编辑。
- `TextField` 类的 `setTextFormat()` 和 `replaceSel()` 方法对文本字段不再有效。更改字段的唯一方法是更改文本字段的 `text` 或 `htmlText` 属性，或者更改文本字段的关联变量。
- 分配到文本字段的 `text` 属性、`htmlText` 属性或关联变量的任何文本都是逐字存储的；写入这些属性之一的任何内容都可以通过文本的原始格式进行检索。

合并样式

Flash Player 中的 CSS 样式是添加式的；也就是说，当样式嵌套时，每层嵌套都可以贡献附加样式信息，这些信息加在一起形成最终的格式设置。

例如，这里是一些分配给文本字段的 XML 数据：

```
<sectionHeading>This is a section</sectionHeading>
<mainBody>This is some main body text, with one
<emphatic>emphatic</emphatic> word.</mainBody>
```

对于上文中的 `emphatic` 一词，`emphasized` 样式嵌套在 `mainBody` 样式中。`mainBody` 样式提供 `color`、`font-size` 和 `decoration` 规则。`emphasized` 样式将 `font-weight` 规则添加到这些规则中。将使用 `mainBody` 和 `emphasized` 所指定规则的组合对 `emphatic` 一词进行格式设置。

使用样式类

您可以创建样式“类”，可以使用 <p> 或 标签的 class 属性将样式类应用于这两个标签。当应用于 <p> 标签时，该样式会影响整个段落。您还可以通过使用 标签的样式类设置文本范围的样式。

例如，以下样式表定义两个样式类：mainBody 和 emphasis。

```
.mainBody {
  font-family:Arial,Helvetica,sans-serif;
  font-size:24px;
}
.emphasis {
  color: #666666;
  font-style:italic;
}
```

在分配给文本字段的 HTML 文本中，您可以将这些样式应用于 <p> 和 标签，如下所示。

```
<p class="mainBody">This is <span class="emphasis">really exciting!</span></p>
```

设置内置 HTML 标签的样式

Flash Player 支持部分 HTML 标签。（有关更多信息，请参见第 130 页的“使用 HTML 格式的文本”。）您可以将 CSS 样式分配到文本字段中显示的内置 HTML 标签的每个实例。例如，以下内容定义内置 <p> HTML 标签的样式。将按照样式规则指定的方式对该标签的所有实例进行样式设置。

```
p {
  font-family:Arial,Helvetica,sans-serif;
  font-size:12px;
  display:inline;
}
```

下表显示可设置样式的内置 HTML 标签以及每种样式的应用方式。

样式名称	样式应用方式
p	影响所有 <p> 标签。
body	影响所有 <body> 标签。如果指定了 p 样式，则该样式优先于 body 样式。
li	影响所有 项目符号标签。
a	影响所有 <a> 锚标签。
a:link	影响所有 <a> 锚标签。此样式在所有 a 样式之后应用。
a:hover	当鼠标从链接上方划过时应用于 <a> 锚标签。此样式在所有 a 和 a:link 样式之后应用。 一旦鼠标从链接上方移开，a:hover 样式也会从该链接中删除。
a:active	当用户单击链接时应用于 <a> 锚标签。此样式在所有 a 和 a:link 样式之后应用。 一旦鼠标按钮松开，a:active 样式也会从该链接中删除。

将样式用于 HTML 的示例

本节介绍将样式用于 HTML 标签的示例。您将创建对某些内置标签进行样式设置并定义某些样式类的样式表。随后将该样式表应用于包含 HTML 格式文本的 TextField 对象。

若要使用样式表设置 HTML 的格式，请执行以下操作：

- 1 在您喜欢使用的文本编辑器中，创建一个文件。
- 2 将以下样式表定义添加到该文件中：

```
p {
    color: #000000;
    font-family:Arial,Helvetica,sans-serif;
    font-size:12px;
    display:inline;
}

a:link {
    color:#FF0000;
}

a:hover{
    text-decoration:underline;
}

.headline {
    color: #000000;
    font-family:Arial,Helvetica,sans-serif;
    font-size:18px;
    font-weight:bold;
    display:block;
}

.byline {
    color: #666600;
    font-style:italic;
    font-weight:bold;
    display:inline;
}
```

此样式表定义两个内置 HTML 标签（<p> 和 <a>）的样式，这些样式将应用于这些标签的所有实例。它还定义两个样式类（.headline 和 .byline），样式类将应用于特定段落和文本范围。

- 3 将该文件另存为 html_styles.css。
- 4 在 Flash 中，创建一个 FLA 文件。
- 5 使用“文本”工具，创建一个大约 400 像素宽、300 像素高的文本字段。
- 6 打开属性检查器（“窗口” > “属性”）并选择该文本字段。
- 7 在属性检查器中，从“文本类型”菜单中选择“动态文本”，从“线条类型”菜单中选择“多行”，选择“将文本呈现为 HTML”选项，然后在“实例名称”文本框中键入 news_txt。
- 8 在时间轴（“窗口” > “时间轴”）中选择图层 1 中的第 1 帧。
- 9 打开“动作”面板（“窗口” > “开发面板” > “动作”），将以下代码添加到“动作”面板中：

```
// 创建新样式表对象
var style_sheet = new TextField.StyleSheet();
// 定义样式的 CSS 文件的位置
var css_url = "html_styles.css";
// 创建一些要显示的 HTML 文本
```

```

var storyText:String = "<p class='headline'>Flash Player 现在支持层叠样式表 !</p><p><span class='byline'>San Francisco, CA</span>ó Macromedia Inc. 今天宣布 Flash Player 的新版本支持层叠样式表 (CSS) 文本样式。有关更多信息, 请访问 <a href='http://www.macromedia.com'>Macromedia Flash Web 站点。</a></p>";
// 加载 CSS 文件并定义 onLoad 处理函数 :
style_sheet.load(css_url);
style_sheet.onLoad = function(ok) {
    if (ok) {
        // 如果样式表加载没有错误,
        // 则将其分配到文本对象,
        // 然后将 HTML 文本分配到文本字段。
        news_txt.styleSheet = style_sheet;
        news_txt.text = storyText;
    }
};

```

注意：为简便起见，设置了样式的 HTML 文本被“硬编码”到脚本中；在实际应用程序中，您可能要从外部文件加载文本。有关加载外部数据的信息，请参见第 157 页的第 10 章“使用外部数据”。

- 10 将该文件（使用文件名 news_html fla）保存到包含以前创建的 CSS 文件的同一个目录中。
- 11 运行影片（“控制” > “测试影片”）自动查看应用于 HTML 文本的样式。

使用样式定义新标签

如果在样式表中定义新样式，则该样式可用作标签，就像使用内置 HTML 标签一样。例如，如果样式表定义名为 sectionHeading 的 CSS 样式，则可以将 <sectionHeading> 用作与该样式表关联的任何文本字段中的元素。此功能允许您将任何 XML 格式文本直接分配到文本字段，以便使用样式表中的规则自动设置文本的格式。

例如，以下样式表创建新样式 sectionHeading、mainBody 和 emphasized。

```

sectionHeading {
    font-family:Verdana, Arial, Helvetica, sans-serif;
    font-size:18px; display:block
}
mainBody {
    color: #000099;
    text-decoration:underline;
    font-size:12px; display:block
}
emphasized {
    font-weight:bold; display:inline
}

```

您随后应该用以下 XML 格式文本填充与该样式表关联的文本字段：

```

<sectionHeading>This is a section</sectionHeading>
<mainBody>This is some main body text,
with one <emphasized>emphatic</emphasized> word.
</mainBody>

```


将样式用于 XML 的示例

在本节中，您将创建与先前创建的 FLA 文件（请参见第 127 页的“将样式用于 HTML 的示例”）相同的文件，但这次使用 XML 格式文本。在本例中，您将使用动作脚本创建样式表，而不是从 CSS 文件导入样式。

使用样式表设置 XML 的格式：

- 1 在 Flash 中，创建一个 FLA 文件。
- 2 使用“文本”工具，创建一个大约 400 像素宽、300 像素高的文本字段。
- 3 打开属性检查器（“窗口” > “属性”）并选择该文本字段。
- 4 在属性检查器中，从“文本类型”菜单中选择“动态文本”，从“线条类型”菜单中选择“多行”，选择“将文本呈现为 HTML”选项，然后在“实例名称”文本框中键入 news_txt。
- 5 在时间轴（“窗口” > “时间轴”）的图层 1 上，选择第 1 帧。
- 6 若要创建样式表对象，打开“动作”面板（“窗口” > “开发面板” > “动作”）并将以下代码添加到“动作”面板中：

```
var xml_styles = new TextField.StyleSheet();
xml_styles.setStyle("mainBody", {
    color:'#000000',
    fontFamily:'Arial,Helvetica,sans-serif',
    fontSize:'12',
    display:'block'
});
xml_styles.setStyle("title", {
    color:'#000000',
    fontFamily:'Arial,Helvetica,sans-serif',
    fontSize:'18',
    display:'block',
    fontWeight:'bold'
});
xml_styles.setStyle("byline", {
    color:'#666666',
    fontWeight:'bold',
    fontStyle:'italic',
    display:'inline'
});
xml_styles.setStyle("a:link", {
    color:'#FF0000'
});
xml_styles.setStyle("a:hover", {
    textDecoration:'underline'
});
```

此代码创建一个名为 xml_styles 的新样式表对象，该对象使用 setStyle() 方法定义样式。这些样式与本章前面部分中在外部 CSS 文件中创建的样式完全匹配。

- 7 若要创建将分配到文本字段的 XML 文本，请将以下代码添加到“操作”面板上：

```
var storyText = "<title>Flash Player 现在支持 CSS</title><mainBody><byline>San Francisco, CA</byline>-- Macromedia Inc. 今天宣布 Flash Player 的新版本支持层叠样式表 (CSS) 文本样式。有关更多信息，请访问 <a href='\"http://www.macromedia.com\"'>Macromedia Flash Web 站点</a></mainBody>";
```

- 8 最后，添加以下代码将样式表对象应用于文本字段的 styleSheet 属性并将 XML 文本分配到该文本字段。

```
news_txt.styleSheet = xml_styles;
news_txt.text = storyText;
```

9 将该文件另存为 news_xml.fla。

10 运行影片（“控制” > “测试影片”）自动查看应用于字段中文本的样式。

使用 HTML 格式的文本

Flash Player 支持部分标准的 HTML 标签（例如 <p> 和 ），您可以使用这些标签对任何动态或输入文本字段中的文本进行样式设置。Flash Player 7 和更高版本中的文本字段还支持 标签，该标签允许您在文本字段中嵌入 JPEG 文件、SWF 文件和影片剪辑。Flash Player 让文本在文本字段中嵌入的图像旁自动换行，这非常类似于 Web 浏览器让 HTML 页中的文本在嵌入的图像旁自动换行。有关更多信息，请参见第 134 页的“在文本字段中嵌入图像、SWF 文件和影片剪辑”。

Flash Player 还支持 <textformat> 标签，该标签允许您将 TextFormat 类的段落格式样式应用于启用 HTML 的文本字段。有关更多信息，请参见第 121 页的“使用 TextFormat 类”。

使用 HTML 格式文本的概述

若要在文本字段中使用 HTML，您必须启用文本字段的 HTML 格式，方法是在属性检查器中选择“将文本呈现为 HTML”选项或将文本字段的 html 属性设置为 true。若要将 HTML 插入文本字段，请使用 TextField.htmlText 属性。

例如，以下代码为名为 headline_txt 的文本字段启用 HTML 格式，然后将一些 HTML 分配到该文本字段。

```
headline_txt.html = true;
headline_txt.htmlText = "<font face='Times New Roman' size='24'>This is how you
    assign HTML text to a text field.</font>";
```

HTML 标签的属性必须括在双引号或单引号中。不带引号的属性值可能产生无法预料的结果，例如不正确的文本呈现。例如，Flash Player 将无法正确呈现以下 HTML 代码片断，这是因为分配到 align 属性的值 (left) 未括在引号中：

```
textField.htmlText = "<p align=left>This is left-aligned text</p>";
```

如果将属性值括在双引号中，则必须对引号进行转义处理 (\")。例如，以下代码都是可接受的：

```
textField.htmlText = "<p align='left'>This uses single quotes</p>";
textField.htmlText = "<p align=\"left\">This uses escaped double quotes</p>";
```

如果从外部文件加载文本，则无需对双引号进行转义处理；只有在动作脚本中分配文本字符串才需执行此操作。

支持的 HTML 标签

本节列出 Flash Player 支持的内置 HTML 标签。您还可以使用层叠样式表创建新的样式和标签；请参见第 122 页的“使用层叠样式表对文本进行格式设置”。

锚标签 (<a>)

<a> 标签创建超级链接并支持以下属性：

- href 指定浏览器中要加载页的 URL。该 URL 可以是绝对路径或相对路径（相对于加载页的 SWF 文件的位置）。
- target 指定要将页加载到的目标窗口的名称。

例如，以下 HTML 代码片断创建链接 “Go home”，该链接将在新浏览器窗口中打开 [www.macromedia.com](#)。

```
<a href="../../../home.htm" target="_blank">Go home</a>
```

您还可以使用样式表为锚标签定义 `a:link`、`a:hover` 和 `a:active` 样式。请参见 [第 126 页的“设置内置 HTML 标签的样式”](#)。

粗体标签 ()

`` 标签将文本呈现为粗体。粗体字体必须可用于显示文本的字体。

```
<b>This is bold text.</b>
```

换行标签 (
)

`
` 标签在文本字段中创建换行符，如以下示例所示：

```
One line of text<br>Another line of text<br>
```

字体标签 ()

`` 标签指定用于显示文本的字体或字体列表。

字体标签支持以下属性：

- `color` 只支持十六进制颜色值 (`#FFFFFF`)。例如，以下 HTML 代码创建红色文本。
`This is red text`
- `face` 指定要使用字体的名称。您还可以指定用逗号分隔的字体名称列表，在这种情况下，Flash Player 选择第一个可用的字体。如果指定的字体未安装在回放系统上或者未嵌入在 SWF 文件中，则 Flash Player 将选择替代字体。

示例：

```
<font face="Times, Times New Roman">This is either Times or Times New Roman.</font>
```

有关在 Flash 应用程序中嵌入字体的更多信息，请参见[第 641 页的 `TextField.embedFonts`](#)和“使用 Flash”帮助中的“设置动态和输入文本选项”。

- `size` 指定字体的大小，以像素为单位。您还可以使用相对磅值（+2 或 -4）。
`This is green, 24-point text`

图像标签 ()

`` 标签允许您将外部 JPEG 文件、SWF 文件和影片剪辑嵌入文本字段中。在文本字段中，文本在嵌入的图像旁自动换行。只有多行并且文本换行的动态和输入文本字段中才支持此标签。

若要创建文本自动换行的多行文本字段，请执行以下操作之一：

- 在 Flash 创作环境中，在舞台上选择文本字段，然后在属性检查器中，从“文本类型”弹出式菜单中选择“多行”。
- 对于在运行时使用 `MovieClip.createTextField()` 创建的文本字段，将新文本字段实例的 `TextField.multiline` 和 `TextField.wordWrap` 属性设置为 `true`。

`` 标签具有一个必需的属性 `src`，该属性指定到 JPEG 文件、SWF 文件的路径或影片剪辑元件的链接标识符。所有其它属性都是可选的。

 标签支持以下属性：

- **src** 指定到 JPEG 或 SWF 文件的 URL，或库中影片剪辑元件的链接标识符。此属性是必需的，所有其它属性都是可选的。外部文件（JPEG 和 SWF 文件）在完全下载完之后才显示。

注意：Flash Player 不支持渐进 JPEG 文件。

- **id** 指定包含嵌入 JPEG 文件、SWF 文件或影片剪辑的影片剪辑实例（由 Flash Player 创建）的名称。该属性可用于使用动作脚本控制嵌入的内容。
- **width** 图像、SWF 文件或影片剪辑的宽度（以像素为单位）。
- **height** 所插入的图像、SWF 文件或影片剪辑的高度（以像素为单位）。
- **align** 指定文本字段中嵌入图像的水平对齐。有效值是 `left` 和 `right`。默认值是 `left`。
- **hspace** 指定图像周围的水平空间量，此空间内将不显示任何文本。默认值为 8。
- **vspace** 指定图像周围的垂直空间量，此空间内将不显示任何文本。默认值是 8。

有关使用 标签的更多信息和示例，请参见第 134 页的“在文本字段中嵌入图像、SWF 文件和影片剪辑”。

斜体标签 (<i>)

<i> 标签以斜体显示括在该标签中的文本。斜体字体必需可用于所使用的字体。

That is very <i>interesting</i>.

上面的代码将按如下方式呈现：

That is very interesting.

列表项标签 ()

 标签在括在该标签的文本前放置一个项目符号。

```
Grocery list:  
<li>Apples</li>  
<li>Oranges</li>  
<li>Lemons</li>
```

上面的代码将按如下方式呈现：

Grocery list:

- Apples
- Oranges
- Lemons

段落标签 (<p>)

<p> 标签创建一个新段落。它支持以下属性：

- **align** 指定段落中的文本对齐方式；有效值为 `left`、`right` 和 `center`。
- **class** 指定 TextField.StyleSheet 对象定义的 CSS 样式类。（有关更多信息，请参见第 126 页的“使用样式类”。）

下面的示例使用 align 属性让文本在文本字段的右侧对齐。

```
textField.htmlText = "<p align='right'>This text is aligned on the right side  
of the text field</p>";
```

下面的示例使用 class 属性将文本样式类分配到 <p> 标签。

```
var myStyleSheet = new TextField.StyleSheet();  
myStyleSheet.secreateTextField("test", 10, 0,0, 300,100);  
createTextField("test", 10, 0,0, 300,100);  
test.styleSheet = myStyleSheet;  
test.htmlText = "<p class='body'>This is some body-styled text.</p>.";
```

范围标签 ()

 标签只可用于 CSS 文本样式。（有关更多信息，请参见第 122 页的“使用层叠样式表对文本进行格式设置”。）它支持以下属性：

- class 指定 TextField.StyleSheet 对象定义的 CSS 样式类。有关创建文本样式类的更多信息，请参见第 126 页的“使用样式类”。

文本样式标签 (<textformat>)

<textformat> 标签允许在 HTML 文本字段中使用 TextFormat 类的部分段落格式设置属性，其中包括行距、缩进、边距和 Tab 键停靠位。您可以将 <textformat> 标签与内置 HTML 标签结合起来。

<textformat> 标签具有以下属性：

- blockindent 指定块缩进（以磅值为单位）；对应于 TextFormat.blockIndent。（请参见第 674 页的 TextFormat.blockIndent。）
- indent 指定从左边缘到段落中第一个字符的缩进；对应于 TextFormat.indent。（请参见第 677 页的 TextFormat.indent。）
- leading 指定行与行之间的前导垂直间距量；对应于 TextFormat.leading。（请参见第 678 页的 TextFormat.leading。）
- leftmargin 指定段落的左边距（以磅值为单位）；对应于 TextFormat.leftMargin。（请参见第 678 页的 TextFormat.leftMargin。）
- rightmargin 指定段落的右边距（以磅值为单位）；对应于 TextFormat.rightMargin。（请参见第 678 页的 TextFormat.rightMargin。）
- tabstops 将自定义 Tab 键停靠位指定为非负整数数组；对应于 TextFormat.tabStops。（请参见第 679 页的 TextFormat.tabStops。）

下面的代码示例使用 <textformat> 标签的 tabstops 属性创建一个数据表，其行标题为粗体，如下所示：

Name	Age	Department
Tim	32	IMD
Edwin	46	Engineering

使用 Tab 键停靠位创建具有格式的数据表：

- 1 使用“文本”工具，创建一个大约 300 像素宽、100 像素高的动态文本字段。
- 2 在属性检查器中，在“实例名称”文本框中输入 table_txt，从“线条类型”菜单中选择“多行”，然后选择“将文本呈现为 HTML”选项。

3 在时间轴中，选择图层 1 上的第 1 帧。

4 打开“动作”面板（“窗口” > “开发面板” > “动作”），然后在“动作”面板中输入以下代码：

```
var rowHeaders = "<b>Name\t</b><b>Age\t</b><b>Department";  
var row_1 = "Tim\t31\tIMD";  
var row_2 = "Edwin\t42\tQA";  
table_txt.htmlText = "<textformat tabstops='[100, 200]'\t";  
table_txt.htmlText += rowHeaders;  
table_txt.htmlText += row_1;  
table_txt.htmlText += row_2;  
table_txt.htmlText += "</textformat>";
```

请注意，使用 Tab 字符转义序列 (\t) 在表中的每“列”间添加 Tab 字符。

5 选择“控制” > “测试影片”对影片进行测试。

下划线标签 (<u>)

<u> 标签为括在该标签中的文本添加下划线。

This text is <u>underlined</u>.

上面的代码将按如下方式呈现：

This text is underlined.

在文本字段中嵌入图像、SWF 文件和影片剪辑

在 Flash Player 7 和更高版本中，您可以使用 标签在动态和输入文本字段中嵌入 JPEG 文件、SWF 文件和影片剪辑。（有关 标签属性的完整列表，请参见第 131 页的“[图像标签 \(\)](#)”。）

默认情况下，Flash 以完全大小显示文本字段中嵌入的媒体。若要指定嵌入媒体的尺寸，请使用 标签的 height 和 width 属性。（请参见第 135 页的“[指定高度和宽度值](#)”。）

通常情况下，文本字段中嵌入的图像显示在 标签后的行上。但是，如果 标签是文本字段中的第一个字符，则该图像显示在文本字段的第一行上。

嵌入 SWF 和 JPEG 文件

若要将 JPEG 或 SWF 文件嵌入文本字段中，请在 标签的 src 属性中指定到 JPEG 或 SWF 文件的绝对或相对路径。例如，以下代码插入一个 JPEG 文件，该文件与 SWF 文件位于同一个目录中。

```
textField_txt.htmlText = "<p>Here a picture from my last vacation:<img  
src='beach.jpg'>";
```

嵌入影片剪辑元件

若要在文本字段中嵌入影片剪辑元件，您必须为 标签的 src 属性指定元件的链接标识符。（有关定义链接标识符的信息，请参见第 112 页的“[将影片剪辑元件附加到舞台](#)”。）

例如，以下代码插入具有链接标识符 symbol_ID 的影片剪辑元件。

```
textField_txt.htmlText = "<p>Here a movie clip symbol:<img src='symbol_ID'>";
```

为了使嵌入的影片剪辑正确完整地显示，其元件的注册点应该位于点 (0,0)。

指定高度和宽度值

如果为 标签指定了 width 和 height 属性，则将在文本字段中为 JPEG 文件、SWF 文件或影片剪辑保留空间。JPEG 或 SWF 文件完全下载完之后，它将显示在该保留空间中。Flash 将根据 height 和 width 值按比例增大或缩小媒体。

如果未指定 height 和 width 值，则不为嵌入的媒体保留空间。JPEG 或 SWF 文件完全下载完之后，Flash 将其以完整大小插入文本字段中并重新排列周围的文本。

通过动作脚本控制嵌入的媒体

Flash Player 为每个 标签创建一个新影片剪辑并将该影片剪辑嵌入 TextField 对象中。 标签的 id 属性允许您将实例名称分配到创建的影片剪辑。这允许您通过动作脚本控制该影片剪辑。

Flash Player 创建的影片剪辑作为子级影片剪辑添加到包含图像的文本字段中。

例如，以下代码在级别 0 上名为 textField_txt 的文本字段中嵌入名为 animation.swf 的 SWF 文件，然后将实例名称 animation_mc 分配给包含 SWF 文件的影片剪辑。

```
_level0.textField_txt.htmlText = "Here an interesting animation:<img  
src='animation.swf' id='animation_mc'>
```

在本例中，到新创建的影片剪辑的全限定路径是 _level0.textField_txt.animation_mc。例如，您可以将以下代码附加到一个按钮（与 textField_txt 在同一个时间轴上），该按钮可以停止嵌入 SWF 文件的播放头。

```
on (press) {  
    textField_txt.animation_mc.stop();  
}
```

通过嵌入的媒体创建超级链接

若要通过嵌入的 JPEG 文件、SWF 文件或影片剪辑创建超级链接，请将 标签括在 <a> 标签中：

```
textField.htmlText = "Click the image to return home<a href='home.htm'><img  
src='home.jpg'></a>";
```

当鼠标位于括在 <a> 标签中的图像、SWF 文件或影片剪辑上方时，鼠标指针将变为“手形”图标，就像标准的超级链接一样。互动操作（例如鼠标单击和按键）不在括在 <a> 标签中的 SWF 文件和影片剪辑中注册。

创建滚动文本

在 Flash 中创建滚动文本有多种方法。选择“文本”菜单或上下文菜单中的“可滚动模式”选项，或按住 Shift 键双击文本块手柄可以将动态和输入文本字段设置为可滚动模式。

可以使用 TextField 对象的 scroll 和 maxscroll 属性在文本块中控制垂直滚动，使用 hscroll 和 maxhscroll 属性在文本块中控制水平滚动。scroll 和 hscroll 属性分别指定当前垂直和水平滚动位置；您可以读写这些属性。maxscroll 和 maxhscroll 属性分别指定最大垂直和水平滚动位置；您只能读取这些属性。

Flash MX 2004 中的 TextArea 组件提供一种简单的方法通过最少的脚本撰写工作创建滚动文本字段。有关更多信息，请参见“使用组件”帮助中的“TextArea 组件条目”。

要创建可滚动动态文本块，请执行以下操作之一：

- 按住 Shift 键双击动态文本块上的手柄。
- 用箭头工具选择动态文本块，然后选择 “文本” > “可滚动”。
- 用箭头工具选择动态文本块。右击 (Windows) 或按住 Control 键单击 (Macintosh) 动态文本块，然后选择 “文本” > “可滚动”。

使用滚动属性创建滚动文本：

1 执行以下其中一项操作：

- 使用 “文本” 工具在舞台上拖出一个文本字段。在属性检查器中为文本字段指定一个实例名称 `textField`。
- 使用动作脚本，通过 `MovieClip.createTextField()` 方法动态创建一个文本字段。为该文本字段指定一个实例名称 `textField` 作为该方法的参数。

2 创建一个向上按钮和一个向下按钮或选择 “窗口” > “其它面板” > “公用库” > “按钮”，然后将按钮拖到舞台上。

您将使用这些按钮来上下滚动文本。

3 从舞台中选择向下按钮。

4 在 “动作” 面板中（“窗口” > “开发面板” > “动作”），输入以下代码在文本字段中向下滚动文本：

```
on (press) {  
    textField.scroll += 1;  
}
```

5 从舞台中选择向上按钮。

6 在 “动作” 面板中，输入以下代码向上滚动文本：

```
on (press) {  
    textField.scroll -= 1;  
}
```


第 9 章

使用动作脚本 2.0 创建类

动作脚本 2.0 在动作脚本语言的基础上进行了调整，新提供了一些其它编程语言（例如 Java）所具有的强大编程功能。动作脚本 2.0 提倡可复用、可伸缩、可维护的可靠程序结构。它还通过为用户提供全程的编码辅助和调试信息，从而缩短了开发时间。动作脚本 2.0 符合现有标准，且基于 ECMAScript 4 建议 (www.mozilla.org/js/language/es4/)。Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中提供动作脚本 2.0。

下面内容对动作脚本 2.0 的功能进行了介绍。

熟悉的面向对象编程 (OOP) 模型 动作脚本 2.0 的主要功能是一个大家熟悉的用于创建面向对象程序的模型。动作脚本 2.0 中引入了几个新的面向对象的概念和关键字，例如，类、接口和包。如果您使用过 Java 编程，那么您对这些概念一定很熟悉。

动作脚本 2.0 提供的 OOP 模型是以前的 Macromedia Flash 版本中用于创建对象和建立继承的原型链方法的“句法定式”。

严格数据类型指定 动作脚本 2.0 还允许您为变量、函数参数和函数返回类型明确指定数据类型。例如，以下代码声明一个名为 `userName`、类型为 `String` 的变量，`String` 是一种内置的活动脚本数据类型（即类）。

```
var userName:String = "";
```

编译器警告和错误 以上两个功能使创作工具和编译器能够提供编译器警告和错误消息，帮助您更快地找到应用程序中的错误，而以前在 Flash 中要慢得多。

小心：如果要使用动作脚本 2.0 语法，请确保 FLA 文件的“发布设置”指定为“动作脚本 2.0”。对于在 Flash MX 2004 中创建的文件，这是默认设置。但是，如果您打开以前的使用动作脚本 1 的文件并用动作脚本 2.0 对其进行改写，则请将 FLA 文件的“发布设置”更改为“动作脚本 2.0”。如果不执行此操作，则您的 FLA 文件将不能正确编译，但不生成任何错误。

面向对象编程的原则

本节简要介绍开发面向对象的程序需要遵循的原则。本章的其余部分将进一步深入讨论这些原则，并详细介绍如何在 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中实现这些原则。

对象

考虑现实世界中的一个对象 — 例如，一只猫。我们可以说一只猫具有属性（或状态），例如猫名、猫龄和颜色；猫还具有各种行为，例如睡觉、吃食和发出呜呜声。在面向对象编程的世界里，对象也具有属性和行为。使用面向对象的技术，您可以为现实世界中的对象（例如一只猫）或更为抽象的对象（例如一个化学过程）建立模型。

类和类成员

我们继续探讨现实世界中猫的例子，猫的颜色、猫龄和猫名可能不同，它们吃食和发出呜呜声的方式也可能不同。但所有猫都属于某一个对象类，它们都是类型为“猫”的对象。每只猫（现实世界中）都是猫这个类中的一个实例。

与此类似，在面向对象的编程中，类定义一类对象的蓝图。属于某个类的特性和行为称作类的成员。特性（在猫的例子中，特性包括猫名、猫龄和颜色）称作类的属性，用变量表示；行为（吃食、睡觉）称作类的方法，用函数表示。

例如，您可以创建一个 Person 类，然后创建该类的一个实例（一个人），也称为一个 Person 对象。该 Person 对象将包含 Person 类的所有属性和方法。

在动作脚本中，使用 class 语句来定义类（请参见第 141 页的“创建和使用类”）。动作脚本中有许多内置类，例如，MovieClip、TextField 和 String 类。有关更多信息，请参见第 101 页的第 6 章“使用内置类”。

继承

面向对象编程的主要优点之一就是可以创建类的子类；子类可以继承超类的所有属性和方法。子类通常会定义额外的方法和属性；这也称作扩展超类。子类还可以重写（提供自己的定义）从超类继承的方法。

例如，您可以创建一个 Mammal 类，定义所有哺乳动物共有的某些属性和行为。然后，您可以创建一个扩展 Mammal 类的 Cat 类。通过这种方式，继承可以提高代码的可复用性；您不必重新创建两个类共有的任何代码，而只对现有类加以扩展即可。同样，另一个子类也可以再次扩展 Cat 类，依此类推。在复杂的应用程序中，确定类的层次结构是设计过程中的大部头工作。

在活动脚本中，可以使用 extends 关键字在一个类和它的超类之间建立继承关系。有关更多信息，请参见第 143 页的“创建子类”。

接口

在面向对象的编程中，可将接口看作是未实现（定义）方法的类。另一个类可以实现该接口所声明的方法。

也可将接口看作是用于将两个不相关的类（如果没有接口，这两个类之间没有任何关系）关联起来的“编程约定”。例如，假设您和一个程序员小组一起工作，每个程序员开发同一个应用程序的不同部分（类）。设计这个应用程序时，大家约定不同的类将使用一组方法来进行通信。因此，您创建了一个接口，用以声明这些方法、方法的参数及其返回类型。任何实现该接口的类都必须提供这些方法的定义，否则将出现编译器错误。

您还可以使用接口来提供有限的“多重继承”。动作脚本 2.0 中不允许使用多重继承。所谓“多重继承”是指一个类扩展多个类。例如，在 C++ 中，Cat 类可以扩展 Mammal 类，也可以扩展具有 ChaseTail 和 EatCatNip 方法的 Playful 类。动作脚本 2.0 与 Java 类似，不允许一个类直接扩展多个类。但是，您可以创建一个 Playful 接口，在其中声明 ChaseTail 和 EatCatNip 方法。这样，Cat 类或任何其它类就可以实现该接口并提供这些方法的定义了。

有关更多信息，请参见第 147 页的“创建接口”。

使用类：一个简单的示例

本节向那些刚刚开始学习面向对象编程的读者概要介绍在 Flash 中创建和使用类的工作流程。这个工作流程至少包括以下步骤：

- 1 在外部动作脚本类文件中定义一个类。

- 2 将类文件保存到一个指定的类路径目录（Flash 在此位置查找类）中。
- 3 在另一个脚本中创建类的一个实例（既可以是在一个 Flash (FLA) 文档中，也可以是在一个外部脚本文件中），或基于原始类创建一个子类。

本节还将讨论动作脚本 2.0 中一个称为严格数据类型指定的新功能。该功能允许您为变量、函数参数或函数返回类型指定数据类型。

虽然本节只讨论类，但使用接口的工作流程与此基本相同。有关更多信息，请参见[第 147 页的“创建和使用接口”](#)。

创建类文件

若要创建类，首先必须创建一个外部动作脚本 (AS) 文件。类（和接口）只能在外部脚本文件中定义。例如，您不能在附加到 Flash 文档 (FLA) 中的帧或按钮的脚本中定义类。若要创建一个外部 AS 文件，请使用 Flash 附带的动作脚本编辑器，或您常用的代码或文本编辑器。

注意：发布、导出、测试或调试 FLA 文件时，外部文件中的动作脚本代码将被编译成 SWF 文件。因此，如果对外部文件进行了任何更改，则必须保存该文件，并重新编译使用它的任何 FLA 文件。

在下面的步骤中，您将创建一个名为 `Person` 的类。这个类包含两个属性（`age` 和 `name`）和一个方法（`showInfo()`），调用该方法将在“输出”面板中显示这两个属性的值。

创建类文件：

- 1 在您的硬盘上新建一个目录，并将其命名为“PersonFiles”。该目录将包含此项目的所有文件。
- 2 执行以下其中一项操作：
 - 使用您常用的文本或代码编辑器新建一个文件。
 - （仅限 Flash Professional）选择“文件”>“新建”，打开“新建文档”对话框；从文件类型列表中选择“动作脚本文件”，然后单击“确定”。“脚本”窗口将打开并显示一个空文件。
- 3 将该文件保存到 PersonFiles 目录中，文件名为 `Person.as`。
- 4 在脚本窗口中，输入以下代码：

```
class Person {  
}
```

这称作类声明。最基本的类声明应包括：一个 `class` 关键字，后面跟有类名（本例中为 `Person`），然后跟有一对大括号（`{}`）。大括号内的所有内容都称作类体，类的属性和方法都在其中定义。

注意：类的名称 (`Person`) 应与包含该类的 AS 文件 (`Person.as`) 相匹配。这非常重要；如果这两个名称不匹配，将无法编译类。

- 5 若要创建 `Person` 类的属性，请使用 `var` 关键字定义两个名称分别为 `age` 和 `name` 的变量，如下所示。

```
class Person {  
  
    var age:Number;  
    var name:String;  
  
}
```

提示：按照约定，应在类体的最前面定义类属性（这可以使代码更易让人理解），但不是非要这样做。

请注意变量声明中使用的冒语法号 (`var age:Number` 和 `var name:String`)。这就是严格数据类型指定的一个示例。按这种方式键入变量时 (`var variableName:variableType`)，动作脚本 2.0 编译器将确保赋给该变量的任何值都与指定的类型匹配。虽然不是非要按照此语法定义变量，但这是一个好习惯，并且可以使脚本调试更容易。(有关更多信息，请参见第 33 页的“严格数据类型指定”。)

- 6 下一步将创建 `showInfo()` 方法。该方法返回一个预先设置好格式的字符串，其中包含 `age` 和 `name` 属性的值。在类体中添加 `showInfo()` 函数定义，如下所示。

```
class Person {  
  
    var age:Number;  
    var name:String;  
  
    // 用于返回属性值的方法  
    function showInfo():String {  
        return("Hello, my name is " + name + " and I'm " + age + " years old.");  
    }  
}
```

请注意函数定义中数据类型指定的使用（可选，但建议使用）。

```
function showInfo():String {...}
```

在本例中，我们指定了 `showInfo()` 函数的返回值的数据类型（字符串）。

- 7 本节中您将添加的最后一段代码是一个称作构造函数 的特殊函数。在面向对象的编程中，构造函数初始化类的每个新实例。

构造函数的名称必须类的名称相同。若要创建该类的构造函数，请添加以下代码：

```
class Person {  
  
    var age:Number;  
    var name:String;  
  
    // 用于返回属性值的方法  
    function showInfo():String {  
        return("Hello, my name is " + name + " and I " + age + " years old.");  
    }  
  
    // 构造函数  
    function Person (myName:String, myAge:Number) {  
        name = myName;  
        age = myAge;  
    }  
}
```

`Person()` 构造函数有两个参数 `myName` 和 `myAge`，并将这两个参数的值赋给 `name` 和 `age` 属性。这两个参数的数据类型分别被严格指定为 `String` 和 `Number`。有关构造函数的更多信息，请参见第 144 页的“构造函数”。

注意：如果不创建构造函数，编译过程中将自动创建一个空构造函数。

- 8 将该文件保存到您在第 1 步中创建的 `PersonFiles` 目录中，文件名为 `Person.as`。
如果您正在使用 Flash MX 2004（而不是 Flash Professional），请转到下一节。
- 9（仅限 Flash Professional）选择“工具”>“语法检查”，或按下 `Control+T` 组合键 (Windows) 或 `Command+T` 组合键 (Macintosh)，对该类文件进行语法检查。
如果“输出”面板中报告有错误，请对您脚本中的代码与上面第 7 步中的最终代码进行比较。
如果您无法修复代码中的错误，请从“帮助”面板复制第 7 步中的完整代码。

创建 Person 类的实例

下一步是在另一个脚本（例如，Flash (FLA) 文档中的一个帧脚本或另一个 AS 脚本）中创建 Person 类的一个实例，并将其赋给一个变量。若要创建自定义类的实例，应使用 new 运算符，就像创建内置动作脚本类（例如，XML 或 TextField 类）的实例一样。

例如，以下代码创建 Person 类的一个实例，并将其赋给变量 newPerson。

```
var newPerson:Person = new Person("Nate", 32);
```

此代码调用 Person 类的构造函数，并将参数值 "Nate" 和 32 传递给该函数。

newPerson 变量被指定为一个 Person 类型的对象。以这种方式指定对象类型使编译器能够确保您不会尝试访问该类中未定义的属性或方法。（但如果您用 dynamic 关键字将类声明为动态类，则是个例外。请参见第 152 页的“创建动态类”。）

在 Flash 文档中创建 Person 类的实例：

- 1 在 Flash 中，选择“文件”>“新建”，从文档类型列表中选择“Flash 文档”，然后单击“确定”。
- 2 将该文件保存到您以前创建的 PersonFiles 目录中，文件名为 createPerson.fla。
- 3 在“时间轴”中选择“图层 1”，然后打开“动作”面板（“窗口”>“开发面板”>“动作”）。
- 4 在“动作”面板中输入以下代码：

```
var person_1:Person = new Person("Nate", 32);  
var person_2:Person = new Person("Jane", 28);  
trace(person_1.showInfo());  
trace(person_2.showInfo());
```

以上代码创建 Person 类的两个实例 person_1 和 person_2，然后对每个实例调用 showInfo() 方法。

- 5 保存您的工作，然后选择“控制”>“测试影片”。您应在“输出”面板中看到以下内容：
Hello, my name is Nate and I'm 32 years old.
Hello, my name is Jane and I'm 28 years old.

通过调用类的构造函数来创建该类的实例时，Flash 将在一组预定的目录位置中查找与该构造函数同名的动作脚本文件。这组目录位置统称为类路径（请参见第 148 页的“理解类路径”）。

您现在对如何在 Flash 文档中创建和使用类应该已经有了一个总体概念。本章后面的部分将更详细地研究类和接口。

创建和使用类

如上文所述，类由两部分组成：类声明和类体。类声明至少应包含一条 class 语句，后面跟有一个类名标识符，然后跟有一对大括号。大括号内的所有内容都是类体。

```
class className {  
    // 类体  
}
```

只能在动作脚本 (AS) 文件中定义类。例如，不能在 FLA 文件中的帧脚本中定义类。此外，指定的类名必须与包含该类的 AS 文件的文件名相匹配。例如，如果创建一个名为 Shape 的类，那么包含类定义的 AS 文件必须名为 Shape.as。

```
// 在 Shape.as 文件中  
class Shape {  
    // Shape 类体  
}
```

您创建的所有动作脚本类文件都必须保存在指定的类路径目录（Flash 在编译脚本时将在这些目录中查找类定义）中。（请参见第 148 页的“理解类路径”。）

类名必须是标识符；也就是说，第一个字符必须是字母、下划线（_）或美元符号（\$），后面的每个字符都必须是字母、数字、下划线或美元符号。另外，在声明类名的文件中，该类名必须是全限定的；也就是说，它必须反映存储的目录。例如，要创建存储在 myClasses/education/curriculum 目录中的名为 RequiredClass 的类，您必须按如下所示在 RequiredClass.as 文件中声明该类：

```
class myClasses.education.curriculum.RequiredClass {  
}
```

因此，最好在开始创建类之前就计划好您的目录结构。否则，如果您在创建类文件之后决定移动它们，就将不得不修改类声明语句以反映其新位置。

创建属性和方法

类的成员包括属性（变量声明）和方法（函数声明）。所有属性和方法都必须在类体（大括号）内声明；否则，编译时将出现错误。

在类内（但在函数外）声明的任何函数都是该类的属性。例如，上文中讨论的 Person 类有两个属性 age 和 name，其类型分别为 Number 和 String。

```
class Person {  
    var age:Number;  
    var name:String;  
}
```

同样，在类内声明的任何函数都将被视为该类的方法。在 Person 类的示例中，您创建了一个名为 showInfo() 的方法。

```
class Person {  
    var age:Number;  
    var name:String;  
    function showInfo() {  
        // showInfo() 方法定义  
    }  
}
```

以内联方式初始化属性

您可以使用默认值以内联方式（即，在声明属性时）初始化属性，如下所示：

```
class Person {  
    var age:Number = 50;  
    var name:String = "John Doe";  
}
```

以内联方式初始化属性时，赋值语句右侧的表达式必须为编译时常量。即，该表达式不能引用任何在运行时设置或定义的变量。编译时常量包括文本字符串、数字、布尔值、null 和 undefined，以及以下内置类的构造函数：Array、Boolean、Number、Object 和 String。

例如，以下类定义以内联方式初始化几个属性：

```
class CompileTimeTest {  
    var foo:String = "my foo"; // 没问题  
    var bar:Number = 5; // 没问题  
    var bool:Boolean = true; // 没问题  
    var name:String = new String("Jane"); // 没问题  
    var who:String = foo; // 没问题，因为 "foo" 是常量
```



```

var whee:String = myFunc(); // 错误! 不是编译时常量表达式
var lala:Number = whee; // 错误! 不是编译时常量表达式
var star:Number = bar + 25; // 没问题, “bar” 和 “25” 都是常量

function myFunc() {
    return "Hello world";
}
}

```

此规则仅适用于实例变量（被复制到类的各个实例中的变量），而不适用于类变量（属于类本身的变量）。有关这些变量类型的更多信息，请参见第 145 页的“实例成员和类成员”。

创建子类

在面向对象的编程中，子类可以继承另一个类（称作超类）的属性和方法。若要在两个类中创建这种关系，应使用 `class` 语句的 `extends` 子句。若要指定超类，请使用以下语法：

```
class SubClass extends SuperClass {}
```

在 `SubClass` 中指定的类将继承超类定义的所有属性和方法。例如，可以创建一个 `Mammal` 类，定义所有哺乳动物所共有的属性和方法。若要创建 `Mammal` 类的一个变体，例如一个 `Marsupial` 类，则应扩展 `Mammal` 类（即，创建 `Mammal` 类的一个子类）。

```
class Marsupial extends Mammal {}
```

子类将继承超类的所有属性和方法，包括使用 `private` 关键字声明的任何私有属性或方法。（有关私有变量的更多信息，请参见第 144 页的“控制成员访问”。）

您可以扩展您自己的自定义类，也可以扩展任何内置动作脚本类，例如，`XML`、`Sound` 或 `MovieClip` 类。扩展内置动作脚本类时，您的自定义类将继承该内置类的所有方法和属性。

例如，以下代码定义 `JukeBox` 类，它扩展内置的 `Sound` 类。它定义一个名为 `songList` 的数组和一个名为 `playSong()` 的方法。此方法播放歌曲并调用 `loadSound()` 方法（继承自 `Sound` 类）。

```

class JukeBox extends Sound {
    var songList:Array = new Array("beethoven.mp3", "bach.mp3", "mozart.mp3");
    function playSong(songID:Number) {
        this.loadSound(songList[songID]);
    }
}

```

如果您没有将对 `super()` 的调用放入子类的构造函数中，则编译器将自动生成对其不含参数的直接超类的构造函数的调用作为该函数的第一个语句。如果超类没有构造函数，则编译器将创建一个空构造函数，然后生成从子类对该构造函数的调用。但是，如果超类采用其定义中的参数，则必须在子类中创建构造函数，并用必需的参数调用超类。

多重继承（即，从多个类继承）是不允许的。但是，如果使用单个 `extends` 语句，则类可以有效地从多个类继承：

```

// 不允许
class C extends A, B {}
// 允许
class B extends A {}
class C extends B {}

```

您还可以使用 `extends` 关键字创建接口的子类：

```
interface iA extends interface iB {}
```

构造函数

类的构造函数 是一个特殊的函数。使用 `new` 运算符创建类的实例时将自动调用该函数。构造函数的名称与包含它的类的名称相同。例如，您在上文中创建的 `Person` 类包含以下构造函数：

```
// Person 类构造函数
function Person (myName:String, myAge:Number) {
    name = myName;
    age = myAge;
}
```

如果未明确声明任何构造函数（也就是说，如果未创建名称与类名相同的函数），编译器将自动创建一个空构造函数。

一个类只能包含一个构造函数；动作脚本 2.0 中不支持重载构造函数。

控制成员访问

默认情况下，类的所有属性和方法都可被任何其它类访问：类的所有成员默认都是公共成员。但在某些情况下，您可能需要保护类的数据和方法，不让其它类访问。您需要将这些成员指定为私有成员 - 只有声明或定义这些成员的类才能访问它们。

可以使用 `public` 或 `private` 成员属性来指定公共或私有成员。例如，以下代码声明一个私有变量（一个属性）和一个私有方法（一个函数）。

例如，下面的类 (`LoginClass`) 定义一个名为 `userName` 的私有属性，和一个名为 `getUserName()` 的私有方法。

```
class LoginClass {
    private var userName:String;
    private function getUserName() {
        return this.userName;
    }
    // 构造函数
    function LoginClass(user:String) {
        this.userName = user;
    }
}
```

私有成员（属性和方法）只能由定义这些成员的类和该原始类的子类访问。原始类的实例，或该类的子类的实例，不能访问声明为私有成员的属性和方法；也就是说，私有成员只能在类定义中访问，而不能在实例级别上访问。

例如，可以创建 `LoginClass` 的一个子类（名为 `NewLoginClass`）。这个子类可以访问 `LoginClass` 定义的私有属性 (`userName`) 和私有方法 (`getUserName()`)。

```
class NewLoginClass extends LoginClass {
    // 可以访问 userName 和 getUserName()
}
```

但是，`LoginClass` 或 `NewLoginClass` 的实例不能访问这些私有成员。例如，如果将以下代码添加到某个 FLA 文件内的帧脚本中，将出现一个编译器错误，提示您 `getUserName()` 是私有成员，不能访问。

```
var loginObject:LoginClass = new LoginClass("Maxwell");
var user = loginObject.getUserName();
```

还应注意成员访问控制是一个仅限于编译时的功能；在运行时，Flash Player 不区分私有或公共成员。

实例成员和类成员

在面向对象的编程中，类的成员（属性或方法）可以是实例成员，也可以是类成员。对于类的各个实例，都要分别创建实例成员，并将其复制到各个实例中；相反，对于每个类，仅创建一次类成员。（类成员也称作静态成员。）

若要调用实例方法或访问实例属性，应引用该类的一个实例。例如，以下代码调用 `MovieClip` 类的一个名为 `clip_mc` 的实例的 `showInfo()` 方法：

```
clip_mc.showInfo();
```

但是，类（静态）成员分配给类本身，而不分配给该类的任何实例。若要调用类方法或访问类属性，应引用类名本身，而不是该类的某个特定实例：

```
ClassName.classMember;
```

例如，活动脚本的 `Math` 类只包含静态方法和属性。若要调用它的任何一个方法，不需要创建 `Math` 类的实例，而只需通过 `Math` 类本身调用这些方法。以下代码调用 `Math` 类的 `sqrt()` 方法：

```
var square_root:Number = Math.sqrt(4);
```

实例成员可以读取静态成员，但不能对它们进行写入操作。实例成员在 `for` 或 `for..in` 循环中不是可枚举的。

创建类成员

若要将类的属性指定为静态属性，应使用 `static` 限定符，如下所示。

```
static var variableName;
```

也可以将类的方法声明为静态方法。

```
static function functionName() {  
    // 函数体  
}
```

类（静态）方法只能访问类（静态）属性，而不能访问实例属性。例如，以下代码将导致出现编辑器错误，因为类方法 `getName()` 引用了实例变量 `name`。

```
class StaticTest {  
    var name="Ted";  
  
    static function getName() {  
        var local_name = name;  
        // 错误！不能在静态函数中访问实例变量。  
    }  
}
```

若要解决此问题，可将该方法声明为实例方法，或将该变量声明为类变量。

使用类成员：一个简单的示例

类（静态）成员的一种用途是保留有关类及其实例的状态信息。例如，假设您要跟踪从某个类创建的实例的数目。简单的实现方法就是使用一个类属性，每当创建一个新实例时这个属性的值递增。

在以下示例中，您将创建一个名为 `Widget` 的类，在其中定义一个名为 `widgetCount` 的静态实例计数器。每次创建该类的一个新实例时，`widgetCount` 的值都递增 1，并在“输出”面板中显示 `widgetCount` 的当前值。

使用类变量创建实例计数器：

- 1 新建一个动作脚本 (AS) 文件。
- 2 在文件中添加以下代码：

```
class Widget {
    static var widgetCount:Number = 0; // 初始化类变量
    function Widget() {
        trace("Creating widget #" + widgetCount);
        widgetCount++;
    }
}
```

由于 widgetCount 变量声明为静态变量，因此仅初始化为 0 一次。每次调用 Widget 类的构造函数时，它都将 widgetCount 的值加 1，然后显示当前创建的实例的编号。

- 3 将文件保存为 Widget.as。
- 4 新建一个 Flash (FLA) 文档，并将其保存到与 Widget.as 相同的目录中，文件名称为 createWidget.fla。

在这个文件中，您将创建 Widget 类的新实例。

- 5 在 createWidget.fla 文件中，在“时间轴”中选择“图层 1”，然后打开“动作”面板（“窗口” > “开发面板” > “动作”）。
- 6 在“动作”面板中添加以下代码。

```
// 在创建类的任何实例之前，
// widgetCount 为零 (0)
trace("Widget count at start:" + Widget.widgetCount);
var widget_1 = new Widget();
var widget_2 = new Widget();
var widget_3 = new Widget();
```

- 7 保存文件，然后进行测试（“控制” > “测试影片”）。

您应在“输出”面板中看到以下内容：

```
Widget count at start: 0
Creating widget # 0
Creating widget # 1
Creating widget # 2
```

类成员和子类

类成员将传播到定义这些成员的超类的子类。在上例中（请参见第 145 页的“使用类成员：一个简单的示例”），您使用了一个类属性来跟踪所创建的该类的实例数。您可以创建 Widget 类的一个子类，如下所示。

```
class SubWidget extends Widget {
    function SubWidget() {
        trace("Creating subwidget #" + Widget.widgetCount);
    }
}
```

创建和使用接口

在面向对象的编程中，接口类似于只声明了方法、除此之外不起任何作用的类。也就是说，接口由“空”方法组成。

接口的一个用途是在两个不相关的类（如果没有接口，这两个类没有任何关系）之间实现一个协议。例如，假设您是一个程序员团队中的一员，团队中的每个程序员分别负责一个大型应用程序中不同的部分（也就是说，不同的类）。在这些类中，大多数都是不相关的，但您仍需要一种机制，使不同的类之间能够相互通信。也就是说，您需要定义一个接口，或者称作通信协议，所有的类都必须遵守这个协议。

一种实现方式是创建一个 `Communication` 类，定义所有这些方法，然后让每个类都扩展（或继承自）这个超类。但是，由于应用程序由不相关的类组成，因此，将这些类都硬生生地放置到一个公共的类层次结构中是没有意义的。一个更好的解决方法是创建一个接口，声明这些类将用于通信的方法，然后让每个类都实现这些方法（为这些方法提供其自己的定义）。

通常，不使用接口也可以成功编写程序。但是，如果恰当地使用接口，可以使应用程序设计更完美、更具可伸缩性和可维护性。

创建接口

创建接口的过程与创建类相同。与类相同，只能在外部动作脚本 (AS) 文件中定义接口。接口的声明使用 `interface` 关键字，后面跟有接口名，然后跟有一对大括号，在括号中定义接口体。

```
interface interfaceName {  
    // 接口方法声明  
}
```

接口只能包含方法（函数）声明，包括参数、参数类型和函数返回类型。

例如，以下代码声明一个名为 `MyInterface` 的接口，其中包含两个方法 `method_1()` 和 `method_2()`。第一个方法没有任何参数，而且没有返回类型（指定为 `Void`）。第二个方法声明具有一个类型为 `String` 的参数，其返回类型指定为 `Boolean`。

```
interface MyInterface {  
    function method_1():Void;  
    function method_2(param:String):Boolean;  
}
```

接口不能包含任何变量声明或赋值。在接口中声明的函数不能包含大括号。例如，以下接口无法编译。

```
interface BadInterface{  
    // 编译器错误。不允许在接口中声明变量。  
    var illegalVar;  
  
    // 编译器错误。接口中不允许有函数体。  
    function illegalMethod(){  
    }  
}
```

接口的命名和将它们存储在包中的规则与类的相应规则相同；请参见第 141 页的“创建和使用类”和第 150 页的“使用包”。

接口作为数据类型

与类相似，接口也定义一种新的数据类型。任何实现某个接口的类都可被看作属于该接口所定义的类型。这有助于确定给定的对象是否实现了给定的接口。例如，考虑一下以下接口。

```
interface Movable {  
    function moveUp();  
    function moveDown();  
}
```

现在，考虑一下实现 Movable 接口的 Box 类。

```
class Box implements Movable {  
    var x_pos, y_pos;  
  
    function moveUp() {  
        // 方法定义  
    }  
    function moveDown() {  
        // 方法定义  
    }  
}
```

然后，在另一个创建了 Box 类的实例的脚本中，您可以声明一个 Movable 类型的变量。

```
var newBox:Movable = new Box();
```

在 Flash Player 7 和更高版本中，可在运行时为表达式指定接口类型。如果该表达式是一个实现该接口的对象，或具有实现该接口的超类，则返回该对象。否则，将返回 null。这可用于判断特定对象是否实现了某个接口。

例如，以下代码首先检查对象名 someObject 是否实现了 Movable 接口，然后对该对象调用 moveUp() 方法。

```
if(Movable(someObject) != null) {  
    someObject.moveUp();  
}
```

理解类路径

要使用您定义类或接口，Flash 必须能够找到包含类或接口定义的外部 AS 文件。Flash 在其中搜索类和接口定义的那一组目录称为类路径。

创建动作脚本类文件时，需要将该文件保存到类路径中指定的目录之一，或其子目录中。（您可以修改类路径，加入所需的目录路径；请参见第 149 页的“[修改类路径](#)”。）否则，Flash 将无法解析或找到脚本中指定的类或接口。您在类路径目录中创建的子目录称作包，使用包可以对类进行分类整理。（有关更多信息，请参见第 150 页的“[使用包](#)”。）

全局类路径和文档级类路径

Flash 有两个类路径设置：全局类路径和文档级类路径。全局类路径应用于外部 AS 和 FLA 文件，在“首选参数”对话框（“编辑” > “首选参数”）中设置。文档级类路径仅应用于 FLA 文件，在特定 FLA 文件的“发布设置”对话框（“文件” > “发布设置”）中设置。

全局类路径默认包含两个目录路径：一个指向包含当前文档的目录的相对路径，和位于随 Flash 安装的用户配置目录中的 Classes 目录。该目录的位置如下所示：

- Windows 2000 或 Windows XP : C:\Documents and Settings\<user>\Local Settings\Application Data\Macromedia\Flash MX2004\<language>\Configuration\

- Windows 98 : C:\Windows\Application Data\Macromedia\Flash MX 2004\<language>\Configuration\
- Macintosh OS X : Hard Drive/Users/Library/Application Support/Macromedia/Flash MX 2004/<language>/Configuration/

文档级类路径默认为空。

编辑器如何解析类引用

Flash 在尝试解析 FLA 脚本中的类引用时，首先会在为该 FLA 指定的文档级类路径中进行搜索。如果在该类路径中未找到该类，或如果该类路径为空，Flash 将在全局类路径中进行搜索。如果在全局类路径中也未找到该类，将出现一个编译器错误。

Flash 在尝试解析 AS 脚本中的类引用时，只在全局类路径目录中进行搜索，因为 AS 文件没有相关联的文档类路径。

修改类路径

您可以使用“首选参数”对话框来修改全局类路径。若要修改文档级类路径设置，应使用该 FLA 文档的“发布设置”对话框。您可以添加绝对目录路径（例如，C:/my_classes）和相对目录路径（例如，../my_classes 或 “.”）。

全局类路径默认包含一个绝对路径（用户配置目录中的 Classes 目录）和一个相对类路径。这个相对类路径用一个圆点 (.) 表示，指向当前文档目录。请注意相对类路径可能指向不同的目录，这取决于正在编译或发布的文档的位置。有关更多信息，请参见第 148 页的“全局类路径和文档级类路径”。

修改全局类路径：

- 1 选择“编辑” > “首选参数”打开“首选参数”对话框。
- 2 单击“动作脚本”选项卡，然后单击“动作脚本 2.0 设置”按钮。
- 3 执行以下任意一个操作：
 - 若要将某个目录添加到类路径中，单击“浏览到路径”按钮，浏览到要添加的目录，然后单击“确定”。
您也可以单击“添加新路径” (+) 按钮，在“类路径”列表中添加新的一行。双击新添加的行，键入一个相对路径或绝对路径，然后单击“确定”。
 - 若要编辑现有的类路径目录，在“类路径”列表中选中该路径，单击“浏览到路径”按钮，浏览到要添加的目录，然后单击“确定”。
您也可以在“类路径”列表中双击该路径，键入所需的路径，然后单击“确定”。
 - 若要删除类路径中的某个目录，在“类路径”列表中选中该路径，然后单击“从路径删除”按钮。

修改文档级类路径：

- 1 选择“文件” > “发布设置”，打开“发布设置”对话框。
- 2 单击“Flash”选项卡。
- 3 单击“动作脚本版本”弹出菜单旁边的“设置”按钮。

4 执行以下任意一个操作：

- 若要将某个目录添加到类路径中，单击“浏览到路径”按钮，浏览到要添加的目录，然后单击“确定”。

您也可以单击“添加新路径” (+) 按钮，在“类路径”列表中添加新的一行。双击新添加的行，键入一个相对路径或绝对路径，然后单击“确定”。

- 若要编辑现有的类路径目录，在“类路径”列表中选中该路径，单击“浏览到路径”按钮，浏览到要添加的目录，然后单击“确定”。

您也可以在“类路径”列表中双击该路径，键入所需的路径，然后单击“确定”。

- 若要从类路径中删除某个目录，在“类路径”列表中选中该路径，然后单击“从路径删除”按钮。

使用包

您可以使用包 对动作脚本类文件进行分类整理。包是一个位于指定的类路径目录下、包含一个或多个类文件的目录。（请参见第 148 页的“理解类路径”。）包也可以包含其它的包，这些包称作子包，每个都可以具有其自己的类文件。

包名必须是标识符；也就是说，第一个字符必须是字母、下划线 (_) 或美元符号 (\$)，后面的每个字符都必须是字母、数字、下划线或美元符号。

包通常用于将相关的类整理在一起。例如，您可能有三个相关的类 Square、Circle 和 Triangle。这三个类在 Square.as、Circle.as 和 Triangle.as 文件中定义。假设您已经将这些 AS 文件保存到了类路径中指定的某个目录下。

```
// 在 Square.as 中：  
class Square {}
```

```
// 在 Circle.as 中：  
class Circle {}
```

```
// 在 Triangle.as 中：  
class Triangle {}
```

由于这三个类文件是相关的，因此您可能会决定将它们放入一个名为 Shapes 的包（目录）中。如果是这样，全限定类名将包括包路径和简单类名。包路径用点语法表示，其中每个点表示一个子目录。

例如，如果将各个定义形状的 AS 文件放在了 Shapes 目录下，则需要更改各个类文件中的类名，以反映其新位置，如下所示：

```
// 在 Shapes/Square.as 中：  
class Shapes.Square {}
```

```
// 在 Shapes/Circle.as 中：  
class Shapes.Circle {}
```

```
// 在 Shapes/Triangle.as 中：  
class Shapes.Triangle {}
```

若要引用位于包目录中的类，您可以指定其全限定类名，或使用 import 语句导入该包（请参见下面的内容）。

导入类

若要在另一个脚本中引用类，必须在类名的前面加上该类的包路径。类的名称与其包路径的组合称作类的全限定类名。如果类位于顶级类路径目录中（而不是在该类路径目录的子目录中），则其全限定类名就是其类名。

若要指定包路径，请使用圆点分隔包目录名称。包路径采用分层结构，其中每个圆点代表一个嵌套的目录。例如，假设您创建了一个名为 `Data` 的类，它位于您的类路径下的 `com/network/` 包中。若要创建该类的一个实例，您可以指定全限定类名，如下所示：

```
var dataInstance = new com.network.Data();
```

您也可以使用全限定类名来指定变量的数据类型：

```
var dataInstance:com.network.Data = new Data();
```

可以使用 `import` 语句将包导入到脚本中，这样就可以使用类的缩写名，而不必使用其全限定名。您还可以使用通配符 (*) 导入包中的所有类。

例如，假设您创建了一个名为 `UserClass` 的类，该类位于 `macr/util/users` 包目录路径下：

```
// 在 macr/util/users/UserClass.as 文件中  
class macr.util.users.UserClass { ... }
```

假设在另一个脚本中，您使用 `import` 语句导入了该类，如下所示：

```
import macr.util.users.UserClass;
```

之后，您可以在同一脚本中使用该类的缩写名称引用该类：

```
var myUser:UserClass = new UserClass();
```

您可以使用通配符 (*) 导入某个给定包中的所有类。例如，假设您有一个名为 `macr.util` 的包，其中包含两个动作脚本类文件 `foo.as` 和 `bar.as`。在另一个脚本中，您可以使用通配符同时导入该包中的两个类，如下所示：

```
import macr.util.*;
```

然后，在同一脚本中您就可以直接引用 `foo` 和 `bar` 类了。

```
var myFoo:foo = new foo();  
var myBar:bar = new bar();
```

`import` 语句仅应用于调用该语句的当前脚本（帧或对象）。如果脚本中未使用某个导入的类，则生成的 SWF 文件字节码中将不包含该类，并且该类对包含 `import` 语句的 FLA 文件可能调用的任何 SWF 文件都不可用。有关更多信息，请参见[第 367 页的 import](#)。

隐式获取 / 设置方法

面向对象的编程惯例不提倡直接访问类内的属性。类通常会定义“获取”方法和“设置”方法来分别提供对给定属性的读访问和写访问。例如，假设某个类包含一个名为 `userName` 的属性：

```
var userName:String;
```

该类不允许其实例直接访问这个属性（例如 `obj.userName = "Jody"`），而改为提供两个方法 `getUserName` 和 `setUserName`。这两个方法的实现代码如下所示：

```
function getUserName:String() {  
    return userName;  
}  
  
function setUserName(name:String): {  
    userName = name;  
}
```

您可以看到，`getUserName` 返回 `userName` 的当前值，而 `setUserName` 将 `userName` 的值设置为传递给该方法的字符串参数。这样，该类的实例将使用以下语法来获取或设置 `userName` 属性。

```
// 调用“获取”方法
var name = obj.getUserName();
// 调用“设置”方法
obj.setUserName("Jody");
```

但是，如果您想使用更简明的语法，可以使用隐式获取 / 设置方法。利用隐式获取 / 设置方法能够以直接访问的方式访问类属性，同时还可以保持良好的面向对象编程惯例。

若要定义这些方法，应使用 `get` 和 `set` 方法属性。首先应创建用于获取或设置属性值的方法，然后在方法名前面加上 `get` 或 `set` 关键字。

```
function get user():String {
    return userName;
}

function set user(name:String):Void {
    userName = name;
}
```

获取方法不能有任何参数。而设置方法必须有一个必要参数。设置方法与同一范围内的获取方法可以具有相同的名称。但获取 / 设置方法的名称不能与其它属性的名称相同。例如，在上面的示例代码中，因为定义了名为 `user` 的获取 / 设置方法，所以在同一个类中就不能再有名称也为 `user` 的属性了。

与普通方法不同，调用获取 / 设置方法时，方法名后面不带任何括号或参量。例如，现在可以按照以下语法使用上面定义的获取 / 设置方法来访问或修改 `userName` 的值。

```
var name = obj.user;
obj.user = "Jack";
```

注意：隐式获取 / 设置方法是对动作脚本 1 中 `Object.addProperty()` 方法的句法简化。

创建动态类

默认情况下，类的属性和方法是固定的。也就是说，类的实例不能创建或访问该类原来未声明或定义的属性或方法。例如，考虑一个 `Person` 类，该类定义了两个属性 `name` 和 `age`：

```
class Person {
    var name:String;
    var age:Number;
}
```

如果您在另一个脚本中创建了 `Person` 类的一个实例，并且尝试访问某个该类中不存在的属性，编辑器将生成一个错误。例如，以下代码创建 `Person` 类的一个新实例 (`a_person`)，然后尝试给名为 `hairColor` 的属性赋值，但该属性实际不存在。

```
var a_person:Person = new Person();
a_person.hairColor = "blue"; // 编译器错误
```

此代码将导致一个编译器错误，因为 `Person` 类未声明名为 `hairColor` 的属性。在大多数情况下，这与您的需要正好相符。

但在某些情况下，您可能需要在运行时添加和访问原类定义中未定义的属性或方法。这可以通过 `dynamic` 类限定符来实现。例如，以下代码在上面讨论的 `Person` 类中添加了 `dynamic` 限定符：

```
dynamic class Person {
    var name:String;
    var age:Number;
}
```


现在，Person 类的实例就可以添加和访问原类定义中未定义的属性和方法了。

```
var a_person:Person = new Person();  
a_person.hairColor = "blue"; // 由于是动态类，因此不会出现编译器错误
```

动态类的子类也是动态的。

类的编译和导出方式

默认情况下，SWF 文件使用的类在该 SWF 的第一帧上打包并导出。您也可以指定在哪一帧上打包并导出类。例如，如果 SWF 文件使用了许多类，需要很长时间来下载，这将很有用。如果类在第一帧上导出，则用户必须等到所有类代码都下载完毕后，才能看到那一帧。如果指定在时间轴上后面的某一帧打包并导出类，下载该帧中的类代码时，可在时间轴的前几帧中显示一段简短的加载动画。

为 Flash 文档指定类的导出帧：

- 1 打开 FLA 文件后，选择“文件” > “发布设置”。
- 2 在“发布设置”对话框中，单击“Flash”选项卡。
- 3 单击“动作脚本版本”弹出菜单旁边的“设置”按钮，打开“动作脚本设置”对话框。
- 4 在“导出用于类的帧”文本框中，输入要在第几帧上导出类代码。
如果时间轴上没有指定的帧，发布 SWF 时将出现一条错误消息。
- 5 单击“确定”关闭“动作脚本设置”对话框，然后单击“确定”关闭“发布设置”对话框。

第 IV 部分

使用外部数据和媒体

这一部分介绍如何将外部数据和媒体加入到 Macromedia Flash 应用程序中。

第 10 章：使用外部数据	157
第 11 章：使用外部媒体	171

第 10 章

使用外部数据

在 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中，您可以使用动作脚本将外部源中的数据加载到 SWF 文件中。您还可以发送 SWF 文件中的数据让应用程序服务器（例如 Macromedia ColdFusion MX 或 Macromedia JRun）或另一种类型的服务器端脚本（例如 PHP 或 Perl）来处理这些数据。Flash Player 可以通过 HTTP 或 HTTPS 发送和加载数据，也可以发送和加载本地文本文件中的数据。您还可以为需要较少反应时间的应用程序（例如聊天应用程序或股票报价服务）创建永久性的 TCP/IP 套接字连接。

加载到 SWF 文件或从 SWF 文件发送的数据可格式化为 XML（可扩展标记语言）或名称 / 值对。

Flash Player 还可以与其主机环境（例如，Web 浏览器）或同一台计算机上 Flash Player 的另一个实例互相收发数据。

默认情况下，SWF 文件只能访问驻留在产生 Flash 影片的同个域（例如，www.macromedia.com）中的数据。（有关更多信息，请参见第 166 页的“Flash Player 安全功能”。）

向远程源发送变量和从远程源加载变量

与 HTML 页面类似，SWF 文件是用于捕获和显示信息的窗口。然而，SWF 文件可以在浏览器中保持加载状态，同时用新信息持续更新而不必重新加载整个页面。使用动作脚本函数和方法，可以向服务器端脚本、文本文件以及 XML 文件发送信息，也可从它们那里接收信息。

此外，服务器端脚本可从数据库中请求特定信息，然后将其转发给 SWF 文件。可用多种不同的语言撰写服务器端脚本：其中最常用的是 CFML、Perl、ASP (Microsoft Active Server Pages) 和 PHP。通过在数据库中存储信息和从其中检索信息，您可为 SWF 文件创建动态的和个性化的内容。例如，您可创建消息板、用户的个人配置或者购物车，购物车能够跟踪记录用户购物信息从而确定用户偏好。

许多动作脚本函数和方法可用于将信息传入 SWF 文件和从 SWF 文件传出信息。每个函数或方法使用一个协议来传输信息，并要求信息以一定的方式格式化。

- 使用 HTTP 或 HTTPS 协议以 URL 编码格式发送信息的函数和 MovieClip 方法是 `getURL()`、`loadVariables()`、`loadVariablesNum()`、`loadMovie()` 和 `loadMovieNum()`。
- 使用 HTTP 或 HTTPS 协议以 URL 编码格式发送和加载信息的 LoadVars 方法是 `load()`、`send()` 和 `sendAndLoad()`。
- 使用 HTTP 或 HTTPS 协议以 XML 格式发送和加载信息的方法是 `XML.send()`、`XML.load()` 和 `XML.sendAndLoad()`。

- 创建和使用 TCP/IP 套接字连接以 XML 格式发送和加载信息的方法是 `XMLSocket.connect()` 和 `XMLSocket.send()`。

检查已加载的数据

向 SWF 文件加载数据的每个函数或方法（除了 `XMLSocket.send()` 之外）都是异步的：动作结果返回的时间不确定。

在 SWF 文件中使用已加载数据之前，必须检查它是否已被加载。例如，您不能在同一脚本中加载变量并操作它们的值。在下面的脚本中，只有确保已从文件 `myData.txt` 加载了变量 `lastFrameVisited`，才能使用该变量：

```
loadVariables("myData.txt", 0);
gotoAndPlay(lastFrameVisited);
```

每个函数或方法都有一种特定的技术，您可以使用该技术检查数据是否已被加载。如果使用 `loadVariables()` 或 `loadMovie()`，则可以向影片剪辑目标中加载信息，并使用 `onClipEvent()` 处理函数的 `data` 事件来执行脚本。如果使用 `loadVariables()` 加载数据，则在最后一个变量加载后将执行 `onClipEvent(data)` 处理函数。如果使用 `loadMovie()` 加载数据，则每次 SWF 文件的片断进入 Flash Player 时都会执行 `onClipEvent(data)` 处理函数。

例如，下面的按钮动作从文件 `myData.txt` 中向影片剪辑 `loadTargetMC` 加载变量：

```
on (release) {
    loadVariables("myData.txt", _root.loadTargetMC);
}
```

为 `loadTargetMC` 实例指定的 `onClipEvent()` 处理函数使用变量 `lastFrameVisited`，它是从文件 `myData.txt` 中加载的。下面的处理函数只有在加载了所有变量（包括 `lastFrameVisited`）之后才执行：

```
onClipEvent(data) {
    gotoAndPlay(lastFrameVisited);
}
```

如果使用 `XML.load()`、`XML.sendAndLoad()` 和 `XMLSocket.connect()` 方法，则应该定义一个在数据到达时对其进行处理的处理函数。这个处理函数是 `XML` 或 `XMLSocket` 对象的一个属性，您要为该对象指定一个已定义的函数。当接收到信息时自动调用这些处理函数。对于 `XML` 对象，使用 `XML.onLoad()` 或 `XML.onData()`。对于 `XMLSocket` 对象，使用 `XMLSocket.onConnect()`。

有关更多信息，请参见第 160 页的“使用 XML 类”和第 163 页的“使用 XMLSocket 类”。

使用 HTTP 连接到服务器端脚本

`loadVariables()`、`loadVariablesNum()`、`getURL()`、`loadMovie()` 和 `loadMovieNum()` 函数以及 `MovieClip.loadVariables()`、`MovieClip.loadMovie()` 和 `MovieClip.getURL()` 方法都可以通过 HTTP 或 HTTPS 协议与服务器端脚本进行通讯。这些函数发送函数附加到的时间轴中的所有变量。当用作 `MovieClip` 对象的方法时，`loadVariables()`、`getURL()` 和 `loadMovie()` 发送指定影片剪辑的所有变量；每个函数（或方法）按以下方式处理其响应：

- `getURL()` 将所有信息返回到浏览器窗口，而不是 Flash Player。
- `loadVariables()` 将变量加载到 Flash Player 的指定时间轴或级别中。
- `loadMovie()` 将 SWF 文件加载到 Flash Player 的指定级别或影片剪辑中。

当使用 `loadVariables()`、`getURL()` 或 `loadMovie()` 时，您可以指定几个参数：

- *URL* 是远程变量所在的文件。
- *Location* 是 SWF 文件中接收变量的级别或目标。（`getURL()` 函数不采用此参数。）有关级别和目标的更多信息，请参见“使用 Flash”帮助中的“关于多个时间轴和级别”。
- *Variables* 设置用来发送变量的 HTTP 方法，该方法为 GET 或 POST。当省略时，Flash Player 默认使用 GET，但不发送任何变量。

例如，如果要记录游戏的高分，可在服务器上存储这些得分，并且每次有人玩这个游戏时都用 `loadVariables()` 将它们加载到 SWF 文件中。该函数调用可以是：

```
loadVariables("http://www.mySite.com/scripts/high_score.php", _root.scoreClip, GET);
```

这样会将变量从名为 `high_score.php` 的 PHP 脚本加载到影片剪辑实例 `scoreClip` 中，使用的是 GET HTTP 方法。

使用 `loadVariables()` 函数加载的任何变量必须是标准的 MIME 格式 `application/x-www-form-urlencoded`（CGI 脚本使用的标准格式）。在 `loadVariables()` 的 *URL* 参数中指定的文件必须以这种格式写出变量和值对，这样 Flash 才可以读取它们。此文件可以指定任意数量的变量；变量和值对之间必须用“and”符（&）分隔，并且值中的词之间必须用加号（+）分隔。例如，此短语定义了几个变量：

```
highScore1=54000&playerName1=rockin+good&highScore2=53455&playerName2=bonehelmet&highScore3=42885&playerName3=soda+pop
```

有关更多信息，请参见第 387 页的 `loadVariables()`、第 360 页的 `getURL()`、第 385 页的 `loadMovie()` 和第 181 页的第 12 章“动作脚本字典”中的 **LoadVars** 类条目。

使用 LoadVars 类

可以使用 **LoadVars** 类（而不是 `loadVariables()`）在 SWF 文件和服务器之间传输变量。**LoadVars** 类使您可以将对象中的所有变量发送到指定的 URL 中，并且可以将指定 URL 中的所有变量加载到某个对象中。来自服务器的响应会触发 `LoadVars.onLoad()` 方法并设置目标中的变量。可以使用 **LoadVars** 获取错误信息和进度指示信息，并且可用它在下载数据的过程中同时播放数据。

LoadVars 类与 XML 类类似；它使用方法 `load()`、`send()` 和 `sendAndLoad()` 启动与服务器的通讯。**LoadVars** 和 XML 类之间的主要差别是 **LoadVars** 数据是 **LoadVars** 对象的属性，而不是存储在 XML 对象中的 XML DOM（文档对象模型）树。

您必须创建一个 **LoadVars** 对象来调用其方法。这个对象是用来保存已加载数据的一个容器。

以下过程说明如何使用 **LoadVars** 对象加载文本文件中的变量并在文本字段中显示这些变量。

用 **LoadVars** 对象加载数据：

- 1 在“记事本”或 SimpleText 等文本编辑器中，创建一个文本文件，然后将以下文本添加到该文本文件中：
`day=11&month=July&year=2003`
- 2 将该文件另存为 `date.txt`。
- 3 在 Flash 中，创建一个文档。
- 4 在舞台上创建一个动态文本字段，为其指定实例名称 `date_txt`。
- 5 在时间轴中选择第 1 帧，如果“动作”面板（“窗口” > “开发面板” > “动作”）尚未打开，则将其打开。

6 在“动作”面板中输入下面的代码：

```
var dateVars = new LoadVars();
dateVars.onLoad = function(ok) {
    if (ok) {
        date_txt.text = dateVars.day+"/"+dateVars.month+"/"+dateVars.year;
    }
};
dateVars.load("date.txt");
```

此代码加载 data.txt (day, month, year) 中的变量，然后对它们进行格式化并在文本字段 date_txt 中显示它们。

7 将该文档保存到包含 date.txt（您在第 3 步中保存的文本文件）的同一个目录，文件名为 dateReader fla。

8 选择“控制”>“测试影片”对该文档进行测试。

有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 LoadVars 类条目。

关于 XML

XML（可扩展标记语言）正成为 Internet 应用程序中交换结构化数据的标准。您可把 Flash 中的数据与使用 XML 技术的服务器集成在一起，从而构建高级的应用程序，例如聊天系统或者代理系统。

在 XML 中，和 HTML 一样，使用标签来标记（或者说指定）文本正文。在 HTML 中，使用预定义标签指示文本在 Web 浏览器中的显示方式（例如 标签指示文本应该为粗体）。在 XML 中，您定义用来标识数据类型的标签（例如 <password>VerySecret</password>）。XML 把信息的结构与它的显示方式分离开，这样相同的 XML 文档可在不同的环境中使用和重用。

每个 XML 标签被称作一个节点 或一个元素。每个节点有一个类型（1，表明是一个 XML 元素；或 3，表明是一个文本节点），并且元素也可有多个属性。嵌套在节点中的节点称作子节点。这种节点的分层树结构被称为 XML 文档对象模型 (DOM)，与 JavaScript DOM 很类似，JavaScript DOM 是 Web 浏览器中元素的结构。

在下面的示例中，<PORTFOLIO> 是父节点；它没有属性并包含子节点 <HOLDING>，该子节点具有属性 SYMBOL、QTY、PRICE 和 VALUE：

```
<PORTFOLIO>
  <HOLDING SYMBOL="RICH"
    QTY="75"
    PRICE="245.50"
    VALUE="18412.50" />
</PORTFOLIO>
```

使用 XML 类

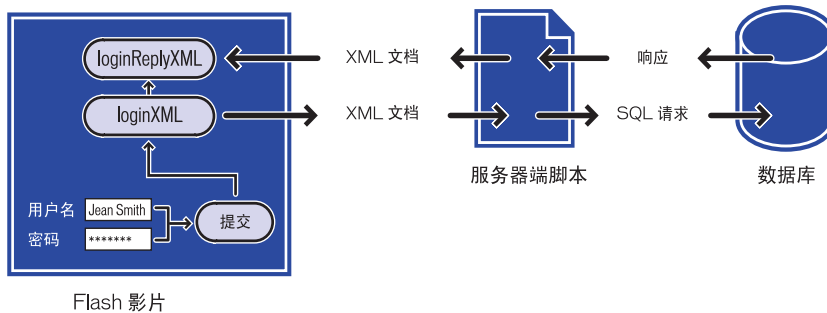
动作脚本 XML 类的方法（例如 appendChild()、removeNode() 和 insertBefore()）使您可以在 Flash 中组织要发送到服务器的 XML 数据，并且它们还可以操作和解释下载的 XML 数据。

下面的 XML 类方法使用 HTTP POST 方法将 XML 数据发送并加载到服务器。

- load() 方法从某个 URL 下载 XML，并将其放在一个动作脚本 XML 对象中。
- send() 方法向 URL 传递 XML 对象。返回的任何信息都被发送到另外一个浏览器窗口。
- sendAndLoad() 方法向 URL 发送 XML 对象。返回的任何信息都放在一个动作脚本 XML 对象中。

例如，您可创建一个代理系统，它在数据库中存储它的所有信息（用户名、密码、会话 ID、公文包以及事务信息）。

在 Flash 和数据库之间传递信息的服务器端脚本以 XML 格式读出和写入数据。可用动作脚本将 SWF 文件中收集到的信息（例如，用户名和密码）转换成一个 XML 对象，然后将数据作为 XML 文档发送到服务器端脚本。您还可以使用动作脚本加载服务器返回到 XML 对象中的 XML 文档，该文档将在 SWF 文件中使用。



Flash 影片、服务器端脚本和数据库之间的数据流以及数据转换

代理系统的密码验证需要两个脚本：一个在第 1 帧中定义的函数，一个脚本，该脚本创建和发送附加到窗体中“Submit”按钮的 XML 对象。

当用户用变量 `username` 和 `password` 将他们的信息输入到 SWF 文件中的文本字段时，变量必须先转换成 XML 再发送到服务器。脚本的第一部分将变量加载到新创建的名为 `loginXML` 的 XML 对象中。当用户单击“Submit”按钮时，`loginXML` 对象先被转换成 XML 字符串，然后发送到服务器。

下面的脚本附加到“Submit”按钮。为了理解此脚本，请阅读注释行（用字符 `//` 表示）：

```
on (release) {  
    // A. Construct an XML document with a LOGIN element  
    loginXML = new XML();  
    loginElement = loginXML.createElement("LOGIN");  
    loginElement.attributes.username = username;  
    loginElement.attributes.password = password;  
    loginXML.appendChild(loginElement);  
  
    // B. Construct an XML object to hold the server's reply  
    loginReplyXML = new XML();  
    loginReplyXML.onLoad = onLoginReply;  
  
    // C. Send the LOGIN element to the server,  
    // place the reply in loginReplyXML  
    loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi",  
        loginReplyXML);  
}
```

当用户单击“Submit”按钮时，脚本的第一部分生成下面的 XML：

```
<LOGIN USERNAME="JeanSmith" PASSWORD="VerySecret" />
```

服务器接收到这个 XML，生成一个 XML 响应，然后将它回送给 SWF 文件。如果接受了该密码，服务器会通过下面的代码进行响应：

```
<LOGINREPLY STATUS="OK" SESSION="rnr6f7vkj2oe14m7jkkycilb" />
```

这个 XML 包括一个 SESSION 属性，该属性包含一个唯一的随机生成的会话 ID，它将被用在该会话其余部分客户端和服务端之间的所有通讯中。如果拒绝了该密码，服务器会通过下面的消息进行响应：

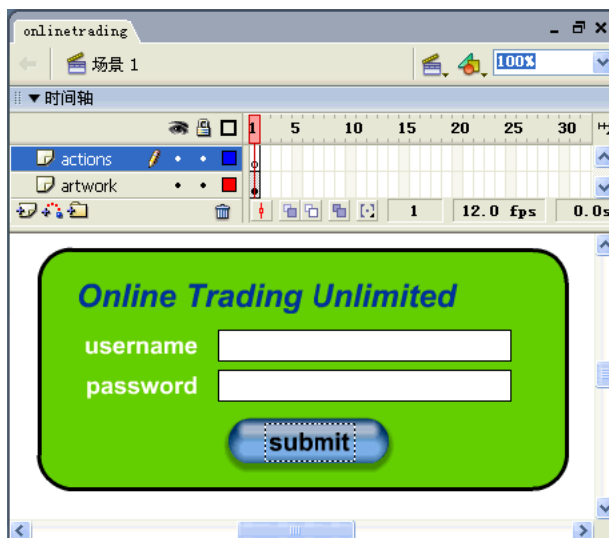
```
<LOGINREPLY STATUS="FAILED" />
```

LOGINREPLY XML 节点必须加载到 SWF 文件的空白 XML 对象中。下面的语句创建 XML 对象 loginReplyXML 来接收 XML 节点：

```
// B. Construct an XML object to hold the server's reply
loginReplyXML = new XML();
loginReplyXML.onLoad = onLoginReply;
```

第二个语句将 onLoginReply() 函数指定给 loginReplyXML.onLoad 处理函数。

LOGINREPLY XML 元素异步到达（与从 loadVariables() 函数得出的数据类似），然后加载到 loginReplyXML 对象中。当数据到达时，即会调用 loginReplyXML 对象的 onLoad 处理函数。您必须定义 onLoginReply() 函数，然后将其指定给 loginReplyXML.onLoad 处理函数，以便它可以处理 LOGINREPLY 元素。您还必须将 onLoginReply() 函数指定给包含“Submit”按钮的帧。



onLoginReply() 函数是在 SWF 文件的第 1 帧中定义的。（为了理解此脚本，请阅读注释行。）

```
function onLoginReply() {
    // Get the first XML element
    var e = this.firstChild;
    // If the first XML element is a LOGINREPLY element with
    // status OK, go to the portfolio screen. Otherwise,
    // go to the login failure screen and let the user try again.
    if (e.nodeName == "LOGINREPLY" && e.attributes.status == "OK") {
        // Save the session ID for future communications with server
        sessionId = e.attributes.session;
        // Go to the portfolio viewing screen
        gotoAndStop("portfolioView");
    } else {
        // Login failed! Go to the login failure screen.
        gotoAndStop("loginFailed");
    }
}
```

该函数的第一行 `var e = this.firstChild` 使用关键字 `this` 来引用刚刚从服务器通过 XML 加载的 XML 对象 `loginReplyXML`。您可以使用 `this`，原因是 `onLoginReply()` 已作为 `loginReplyXML.onLoad` 进行调用，所以尽管 `onLoginReply()` 看上去是一个标准函数，实际上它充当 `loginReplyXML` 的一个方法。

要以 XML 的形式将用户名和密码发送到服务器，然后将 XML 响应加载到 SWF 文件中，您可以使用 `sendAndLoad()` 方法，如下所示：

```
// C. Send the LOGIN element to the server,
//    place the reply in loginReplyXML
loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi", loginReplyXML);
```

注意：这个设计方案只是一个例子，Macromedia 不保证它的安全性级别。如果您正实施一个安全的密码保护系统，请确保对网络安全有很好的理解。

有关更多信息，请参见“在 Web 应用程序中集成 XML 和 Flash”（网址为 www.macromedia.com/support/flash/interactivity/xml/）和第 181 页的第 12 章“动作脚本字典”中的 XML 类条目。

使用 XMLSocket 类

动作脚本提供了一个内置的 XMLSocket 类，它允许您打开与服务器的持续连接。套接字连接允许服务器在信息可用时就向客户端发布（或“推送”）信息。如果没有持续连接，服务器必须等待一个 HTTP 请求。这种打开的连接消除了等待时间问题，它通常用于实时的应用程序，比如聊天。数据以字符串的形式在套接字连接上发送，它应该是 XML 格式的。可以使用 XML 类来组织数据。

要创建套接字连接，必须创建服务器端应用程序来等待套接字连接请求，然后向 SWF 文件发送响应。这种类型的服务器端应用程序可用编程语言（例如 Java）来编写。

您可以使用 XMLSocket 类的 `connect()` 和 `send()` 方法通过套接字连接与服务器互相收发 XML。`connect()` 方法建立与 Web 服务器端口的套接字连接。`send()` 方法将 XML 对象传递给套接字连接中指定的服务器。

当调用 `connect()` 方法时，Flash Player 打开与服务器的 TCP/IP 连接，并保持该连接处于打开状态，直到发生以下情况之一：

- XMLSocket 类的 `close()` 方法被调用。
- 对 XMLSocket 对象的引用不再存在。
- Flash Player 退出。
- 连接中断（例如，调制解调器断开连接）。

下面的示例创建一个 XML 套接字连接，并从 XML 对象 `myXML` 发送数据。为了理解此脚本，请阅读注释行（用字符 `//` 表示）：

```
// 创建新的 XMLSocket 对象
sock = new XMLSocket();
// 调用 connect() 方法来建立到 URL 定义的服务器
// 端口 1024 的连接
sock.connect("http://www.myserver.com", 1024);
// 定义一个函数，分配给处理服务器相应的
// sock 对象如果连接成功，发送
// myXML 对象。如果失败，则在文本字段中提供一条错误
// 消息。
function onSockConnect(success){
    if (success){
        sock.send(myXML);
    } else {
```

```
        msg="There has been an error connecting to "+serverName;
    }
}
// 将 onSockConnect() 函数分配给 onConnect 属性
sock.onConnect = onSockConnect;
```

有关更多信息，请参见第 181 页的第 12 章 “动作脚本字典” 中的 [XMLSocket](#) 类条目。

向 Flash Player 发送消息以及从 Flash Player 接收消息

要将消息从 SWF 文件发送到其宿主环境（例如，Web 浏览器、Macromedia Director 影片或独立的 Flash Player），可以使用 `fscommand()` 函数。此函数允许通过使用宿主的功能来扩展您的 SWF 文件。例如，您可以将 `fscommand()` 函数传递到 HTML 页面中的 JavaScript 函数，该函数会打开一个具有特定属性的新浏览器窗口。

要从 Web 浏览器脚本撰写语言（例如 JavaScript、VBScript 和 Microsoft Jscript）控制 Flash Player 中的 SWF，可使用 Flash Player 的方法，即从宿主环境向 SWF 发送消息的函数。例如，您可在 HTML 页面中有一个链接，它将 SWF 文件发送到特定的帧。

使用 fscommand()

使用 `fscommand()` 函数将消息发送到承载 Flash Player 的那个程序。`fscommand()` 函数有两个参数：*command* 和 *arguments*。要把消息发送到独立的 Flash Player，必须使用预定义的命令和参量（参数）。例如，下面的动作设置独立播放器在按钮释放时将 SWF 文件缩放至整个显示器屏幕大小：

```
on (release) {
    fscommand("fullscreen", "true");
}
```

下表显示为了控制 SWF 文件在独立播放器（包括放映文件）中的播放，可为 `fscommand()` 的 *command* 和 *arguments* 参数指定的值：

命令	Arguments	目的
quit	无	关闭播放器。
fullscreen	true 或 false	指定 true 将 Flash Player 设置为全屏模式。指定 false 使播放器返回标准菜单视图。
allowscale	true 或 false	指定 false 设置播放器始终按 SWF 文件的原始大小绘制 SWF 文件，从不进行缩放。指定 true 强制 SWF 文件缩放到播放器的 100% 大小。
showmenu	true 或 false	指定 true 启用整个上下文菜单项集合。指定 false 使除“设置”和“关于 Flash Player”外的所有上下文菜单项变暗。
exec	应用程序的路径	在播放器内执行应用程序。

若要使用 `fscommand()` 向 Web 浏览器中的脚本撰写语言（例如 JavaScript）发送消息，您可以在 *command* 和 *arguments* 参数中传递任意两个参数。这些参数可以是字符串或表达式，它们将在用来“捕获”或处理 `fscommand()` 函数的 JavaScript 函数中使用。

`fscommand()` 函数在嵌入 SWF 文件的 HTML 页面中调用 JavaScript 函数 `movienamename_DoFSCommand`，其中 *movienamename* 是由 `EMBED` 标签的 `NAME` 属性或由 `OBJECT` 标签的 `ID` 属性指定的 Flash Player 的名称。如果为 Flash Player 指定的名称为 `myMovie`，则调用的 JavaScript 函数就是 `myMovie_DoFSCommand`。

使用 `fscommand()` 从 HTML 页面的 SWF 文件中通过 JavaScript 打开消息框：

- 1 在嵌入 SWF 文件的 HTML 页面中，添加下面的 JavaScript 代码：

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

如果在 HTML “发布设置”对话框中使用具有 FSCOMMAND 模板的 Flash 发布 SWF 文件，则将自动插入此代码。该 SWF 文件的 NAME 和 ID 属性将是文件名。例如，对于文件 `myMovie fla`，属性将设置为 `myMovie`。（有关发布的更多信息，请参见“使用 Flash”帮助中的“发布”。）

或者，对于 Microsoft Internet Explorer 应用程序，可直接在 `<SCRIPT>` 标签中附加事件处理函数，如此例中所示：

```
<Script Language = "JavaScript" event="FSCommand (command, args)" for=  
    "theMovie">  
...  
</Script>
```

- 2 在 Flash 文档中，向按钮添加 `fscommand()` 函数，如此例中所示：

```
on (press) {  
    fscommand("messagebox", "This is a message box invoked from within  
    Flash.");  
}
```

您也可以对 `fscommand()` 和参数使用表达式，如此例中所示：

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

- 3 选择“文件”>“发布预览”>“HTML”来测试文档。

`fscommand()` 函数可将消息发送给 Macromedia Director，Lingo 将消息解释为字符串、事件或可执行的 Lingo 代码。如果消息是字符串或事件，则必须撰写 Lingo 代码以便从 `fscommand()` 函数进行接收，然后在 Director 中执行动作。有关更多信息，请访问 Director 支持中心，网址为：www.macromedia.com/support/director。

在 Visual Basic、Visual C++ 和可承载 ActiveX 控件的其它程序中，`fscommand()` 利用可在环境的编程语言中处理的两个字符串发送 VB 事件。有关更多信息，请使用关键字 Flash method 搜索 Flash 技术支持中心，网址为：www.macromedia.com/go/flash_support_cn。

关于 Flash Player 方法

您可以在 Web 浏览器脚本撰写语言（例如 JavaScript 和 VBScript）中使用 Flash Player 方法控制 Flash Player 中的 SWF 文件。和使用其它方法一样，您可用 Flash Player 方法从脚本撰写环境（而不是从动作脚本）中向 SWF 文件发送调用。每个方法都有一个名字，大多数方法都带参数。参数指定方法要操作的值。某些方法执行的计算返回的值可由脚本撰写环境使用。

有两种不同的技术可以启用浏览器和 Flash Player 之间的通讯：LiveConnect（Windows 95/98/2000/NT 或 Power Macintosh 上的 Netscape Navigator 3.0 或更高版本）和 ActiveX（Windows 95/98/2000/NT 上的 Internet Explorer 3.0 以及更高版本）。虽然所有浏览器和语言的脚本撰写技术都相似，但 ActiveX 控件还可以利用其它一些属性和事件。

有关更多信息，包括 Flash Player 脚本撰写方法的完整列表，请使用关键字 Flash method 搜索 Flash 技术支持中心，网址为：www.macromedia.com/go/flash_support_cn。

关于将 Flash JavaScript 方法用于 Flash Player

Flash Player 6 版本 40 和更高版本支持在 Netscape 6.2 和更高版本中使用 Flash JavaScript 方法和 FSCommand。Flash Player 6 版本 40 以前的版本不支持在 Netscape 6.2 或更高版本中使用 FlashJavaScript 方法和 FSCommand。

对于 Netscape 6.2 和更高版本，无需将 swLiveConnect 设置为 true。但是，将 swLiveConnect 设置为 true 没有负面影响。

Flash Player 安全功能

默认情况下，Flash Player 7 和更高版本防止从一个域提供的 SWF 文件访问从另一个域提供的数据、对象或变量。从不同的域提供的 SWF 文件不能互相访问对方的对象和变量。另外，通过不安全的（非 HTTPS）协议加载的内容不能访问通过安全的 (HTTPS) 协议加载的内容，即使这些内容都在完全相同的域中。例如，如果未得到明确许可，位于

<http://www.macromedia.com/main.swf> 的 SWF 文件不能加载

<https://www.macromedia.com/data.txt> 中的数据。从一个域提供的 SWF 文件也不能从另一个域加载数据（例如使用 loadVariables()）。

相同的数字 IP 地址兼容。但是，域名与 IP 地址不兼容，即使该域名解析为相同的 IP 地址。

下表显示兼容域的示例：

www.macromedia.com	www.macromedia.com
data.macromedia.com	data.macromedia.com
65.57.83.12	65.57.83.12

下表显示不兼容域的示例：

www.macromedia.com	data.macromedia.com
macromedia.com	www.macromedia.com
www.macromedia.com	macromedia.com
65.57.83.12	www.macromedia.com（即使此域可解析为 65.57.83.12）
www.macromedia.com	65.57.83.12（即使 www.macromedia.com 可解析为此 IP）

有关如何允许从一个域提供的 SWF 文件访问从另一个域提供的 SWF 文件中数据、对象或变量的信息，请参见第 167 页的“关于允许跨域 SWF 文件间的数据访问”。有关如何允许通过安全的 (HTTPS) 协议提供的 SWF 文件访问通过不安全的协议提供的 SWF 文件中的数据、对象或变量的信息，请参见第 167 页的“关于允许 SWF 文件间 HTTP 对 HTTPS 协议的访问”。有关如何允许从一个域提供的 SWF 文件加载另一个域中数据（例如使用 loadVariables()）的信息，请参见第 168 页的“关于允许跨域数据加载”。

有关这些安全更改如何影响在 Flash MX 和更早版本中创作的内容的信息，请参见第 169 页的“关于与以前的 Flash Player 安全模型的兼容性”。

关于允许跨域 SWF 文件间的数据访问

一个 SWF 文件可以从 Internet 上的任何位置加载另一个 SWF 文件。但是，为了使两个 SWF 文件能够访问彼此的数据（变量和对象），这两个文件必须源于同一个域。默认情况下，在 Flash Player 7 和更高版本中，这两个域必须完全匹配才能实现两个文件共享数据。但是，SWF 文件可以通过调用 `LocalConnection.allowDomain` 或 `System.security.allowDomain()` 向从特定域提供的 SWF 文件授予访问权限。

例如，假设 `main.swf` 是从 `www.macromedia.com` 提供的，随后该 SWF 文件将来自 `data.macromedia.com` 的另一个 SWF 文件 (`data.swf`) 加载到影片剪辑实例 (`target_mc`) 中。

```
// 在 macromedia.swf 中
target_mc.loadMovie("http://data.macromedia.com/data.swf");
```

此外，假设 `data.swf` 在其主时间轴上定义名为 `getData()` 的方法。默认情况下，一旦 `data.swf` 进行了加载，`main.swf` 就不能调用该文件中定义的 `getData()` 方法。这是因为两个 SWF 文件不在相同的域中。例如，在 `data.swf` 完成加载之后，`main.swf` 中的以下方法调用将失败。

```
// 在 macromedia.swf 中，在加载了 data.swf 之后：
target_mc.getData(); // 此方法调用将失败
```

但是，`data.swf` 可以使用 `LocalConnection.allowDomain` 处理函数或 `System.security.allowDomain()` 方法向从 `www.macromedia.com` 提供的 SWF 文件授予访问权限，具体取决于所需的访问类型。以下代码在被添加到 `data.swf` 之后将允许从 `www.macromedia.com` 提供的 SWF 文件访问其变量和方法：

```
// 在 data.swf 中
System.security.allowDomain("www.macromedia.com");
my_lc.allowDomain = function(sendingDomain) {
    return(sendingDomain=="www.macromedia.com");
}
```

请注意，`allowDomain` 允许被授予权限的域中的任何 SWF 文件撰写允许该权限的域中的任何其它 SWF 文件的脚本，除非被访问的 SWF 文件位于使用安全 (HTTPS) 协议的站点。在此情况下，您必须使用 `allowInsecureDomain`，而不是 `allowDomain`；请参见下面的“[关于允许 SWF 文件间 HTTP 对 HTTPS 协议的访问](#)”。

有关域名匹配的更多信息，请参见第 166 页的“[Flash Player 安全功能](#)”。

关于允许 SWF 文件间 HTTP 对 HTTPS 协议的访问

如上一节所述，您必须使用 `allowDomain` 处理函数或方法允许一个域中的 SWF 文件可以被另一个域中的 SWF 文件访问。但是，如果被访问的 SWF 文件位于使用安全协议 (HTTPS) 的站点上，则 `allowDomain` 处理函数或方法不允许位于使用不安全协议的站点上的 SWF 文件进行访问。若要允许这种访问，您必须使用 `LocalConnection.allowInsecureDomain()` 或 `System.security.allowInsecureDomain()` 语句。

例如，如果位于 `https://www.someSite.com/data.swf` 的 SWF 文件必须允许位于 `http://www.someSite.com` 的 SWF 文件的访问，则添加到 `data.swf` 中的以下代码将允许此访问：

```
// 在 data.swf 中
System.security.allowInsecureDomain("www.someSite.com");
my_lc.allowInsecureDomain = function(sendingDomain) {
    return(sendingDomain=="www.someSite.com");
}
```


关于允许跨域数据加载

Flash 文档可以使用以下数据加载调用之一从外部源加载数据：XML.load()、XML.sendAndLoad()、LoadVars.load()、LoadVars.sendAndLoad()、loadVariables()、loadVariablesNum()。另外，SWF 文件可以在运行时导入运行时共享库或另一个 SWF 文件中定义的资源。默认情况下，数据或 SWF 媒体（运行时共享库的情况）必须与加载该外部数据或媒体的 SWF 驻留在同一个域中。

若要使运行时共享库中的数据和资源可用于其它域中的 SWF 文件，请使用跨域策略文件。跨域策略文件是一个 XML 文件，该文件提供的方法可以使服务器指示其数据和文档可用于从某些域或所有域提供的 SWF 文件。服务器的策略文件指定的域所提供的所有 SWF 文件都将被允许访问该服务器中的数据或资源。

当 Flash 文档试图访问另一个域中的数据时，Flash Player 自动尝试从该域加载策略文件。如果尝试访问数据的 Flash 文档所在的域包括在该策略文件中，则数据将自动成为可访问数据。

策略文件必须具有名称 crossdomain.xml 并驻留在提供数据的服务器的根目录中。只有在通过 HTTP、HTTPS 或 FTP 进行通讯的服务器上，策略文件才起作用。策略文件特定于所在服务器的端口和协议。

例如，某个策略文件位于 <https://www.macromedia.com:8080/crossdomain.xml>，它只适用于在端口 8080 通过 HTTPS 对 www.macromedia.com 进行的数据加载调用。

当您使用 XMLSocket 对象连接到另一个域中的套接字服务器时，此规则有例外情况。如果是这种情况，运行于与套接字服务器所在的同一个域中端口 80 上的 HTTP 服务器必须提供该方法调用的策略文件。

XML 策略文件包含单个 <cross-domain-policy> 标签，该标签又包含零个或多个 <allow-access-from> 标签。每个 <allow-access-from> 标签包含一个属性 domain，该属性指定确切的 IP 地址、确切的域或通配符域（任何域）。通配符域由单个星号（*）（匹配所有域和所有 IP 地址）或后接后缀的星号（只匹配那些以指定后缀结尾的域）表示。后缀必须以点开头。但是，带有后缀的通配符域可以匹配那些只包含后缀但不包含前导点的域。例如，foo.com 可以看作是 *.foo.com 的一部分。IP 域规范中不允许使用通配符。

如果您指定了一个 IP 地址，则将只向使用 IP 语法从该 IP 地址（例如 <http://65.57.83.12/flashmovie.swf>）加载的 SWF 文件授予访问权限，而不向使用域名语法加载的那些 SWF 文件授予访问权限。Flash Player 不执行 DNS 解析。

以下是一个策略文件的示例，该策略文件允许从 foo.com 上的 Flash 文档访问来自 foo.com、friendOfFoo.com、*.foo.com 和 105.216.0.40 的 Flash 文档：

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="www.friendOfFoo.com" />
  <allow-access-from domain="*.foo.com" />
  <allow-access-from domain="105.216.0.40" />
</cross-domain-policy>
```

不包含任何 <allow-access-from> 标签的策略文件等同于服务器上没有策略。

关于与以前的 Flash Player 安全模型的兼容性

作为 Flash Player 中安全功能更改的结果（请参见第 166 页的“Flash Player 安全功能”），在 Flash Player 6 或更早版本中可以正常运行的内容在 Flash Player 7 或更高版本中可能无法正常运行。

例如，在 Flash Player 6 中，驻留在 `www.macromedia.com` 中的 SWF 文件可以访问位于 `data.macromedia.com` 的服务器上的数据。也就是说，Flash Player 6 允许一个域中的 SWF 文件加载“类似的”域中的数据。

在 Flash Player 7 和更高版本中，如果版本 6（或更早版本）的 SWF 文件试图从驻留在另一个域中的服务器加载数据，并且该服务器未提供允许从该 SWF 文件的域进行访问的策略文件，则将出现 Macromedia Flash Player “设置”对话框。该对话框询问用户是允许还是拒绝跨域数据访问。



如果用户单击“允许”，则允许该 SWF 文件访问请求的数据；如果用户单击“拒绝”，则将不允许该 SWF 文件访问请求的数据。

若不想让该对话框出现，请在提供数据的服务器上创建一个安全策略文件。有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

第 11 章

使用外部媒体

如果在 Macromedia Flash MX 2004 或 Macromedia Flash MX Professional 2004 中创作文档时导入图像或声音，则当发布文档时该图像和声音将被打包并存储在 SWF 文件中。除了在创作时导入媒体之外，您还可以在运行时导入外部媒体。出于种种原因，您可能要将媒体保存在 Flash 文档之外。

缩小文件大小 通过将较大的媒体文件保存在 Flash 文档之外并在运行时加载它们，您可以缩短应用程序和演示文稿的初次下载时间，特别是在使用较慢的 Internet 连接进行下载时。

模块化较大的演示文稿 您可以将较大的演示文稿或应用程序分解为多个独立的 SWF 文件，随后在运行时按需要加载这些独立的文件。这不仅可以缩短初始下载时间，还使维护和更新演示文稿的内容变得更加简单。

将内容与演示文稿分开 这是应用程序（特别是数据驱动的应用程序）开发中的一个常见主题。例如，购物车应用程序将显示每个产品的 JPEG 图像。通过在运行时加载每个图像的 JPEG 文件，您可以方便地更新产品的图像而无需修改原始 FLA 文件。

利用仅限运行时功能 某些功能（例如流 FLV 和 MP3 回放）只有在运行时才能通过动作脚本使用。

加载外部媒体的概述

运行时，可以将四种类型的媒体文件加载到 Flash 应用程序中：SWF、MP3、JPEG 和 FLV 文件。Flash Player 可以从任何 HTTP 或 FTP 地址加载外部媒体，也可以使用相对路径从本地磁盘加载或使用 `file://` 协议加载。

若要加载外部 SWF 和 JPEG 文件，您可以使用 `loadMovie()` 或 `loadMovieNum()` 函数，或 `MovieClip.loadMovie()` 方法。当加载 SWF 或 JPEG 文件时，指定影片剪辑或影片级别作为该媒体的目标。有关加载 SWF 和 JPEG 文件的更多信息，请参见第 172 页的“加载外部 SWF 和 JPEG 文件”。

若要回放外部 MP3 (MPEG Layer 3) 文件，请使用 `Sound` 类的 `loadSound()` 方法。此方法允许您指定在开始播放 MP3 文件前是进行流处理还是完全下载。您还可以阅读嵌入 MP3 文件的 ID3 信息（如果可用）。有关更多信息，请参见第 174 页的“读取 MP3 文件中的 ID3 标签”。

Flash Video (FLV) 是 Flash Player 使用的本机视频格式。您可以通过 HTTP 或在本地文件系统中回放 FLV 文件。与在 Flash 文档中嵌入视频相比，播放外部 FLV 文件有多个好处，例如更好的性能和内存管理以及独立的视频和 Flash 帧频。有关更多信息，请参见第 174 页的“动态回放外部 FLV 文件”。

您还可以预加载外部媒体或跟踪外部媒体的下载进度。Flash Player 7 引入了 `MovieClipLoader` 类，可以使用该类跟踪 SWF 或 JPEG 文件的下载进度。若要预加载 MP3 和 FLV 文件，您可以使用 `Sound` 类的 `getBytesLoaded()` 方法和 `NetStream` 类的 `bytesLoaded` 属性。有关更多信息，请参见第 175 页的“预加载外部媒体”。

加载外部 SWF 和 JPEG 文件

若要加载 SWF 或 JPEG 文件，请使用 `loadMovie()` 或 `loadMovieNum()` 全局函数，或 `MovieClip` 类的 `loadMovie()` 方法。若要在 Flash Player 中将 SWF 或 JPEG 文件加载到某个级别，请使用 `loadMovieNum()`。若要将 SWF 或 JPEG 文件加载到影片剪辑目标，请使用 `loadMovie()` 函数或方法。不管是哪种情况，所加载的内容都会替换指定级别或目标影片剪辑的内容。

当您将 SWF 或 JPEG 文件加载到影片剪辑目标中时，SWF 文件或 JPEG 图像的左上角被放置在影片剪辑的注册点上。因为此注册点通常位于影片剪辑的中心，所以加载的内容可能不会出现在中心。另外，当将 SWF 文件或 JPEG 图像加载到根时间轴上时，图像的左上角会位于舞台的左上角。加载的内容会继承影片剪辑的旋转和缩放，只是删除了影片剪辑的原始内容。

您可以选择通过 `loadMovie()` 或 `loadMovieNum()` 调用发送动作脚本变量。这十分有用，例如以下这种情况：您在方法调用中指定的 URL 是一个服务器端脚本，该脚本根据从 Flash 应用程序传递的数据返回 JPEG 或 SWF 文件。

对于图像文件，Flash 仅支持标准的 JPEG 图像文件类型，不支持渐进式 JPEG 文件。

当使用全局 `loadMovie()` 或 `loadMovieNum()` 函数时，指定目标级别或剪辑作为参数。例如，以下代码将 Flash 应用程序 `contents.swf` 加载到名为 `target_mc` 的影片剪辑实例中：

```
loadMovieNum("contents.swf", target_mc);
```

同样，您可以使用 `MovieClip.loadMovie()` 实现相同的结果：

```
target_mc.loadMovie("contents.swf");
```

以下代码将 JPEG 图像 `flowers.jpg` 加载到影片剪辑实例 `image_clip` 中：

```
image_clip.loadMovie("flowers.jpg");
```

有关 `loadMovie()`、`loadMovieNum()` 和 `MovieClip.loadMovie()` 的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中各自的条目。

关于加载的 SWF 文件和根时间轴

动作脚本属性 `_root` 指定或返回对 SWF 文件根时间轴的引用。如果将 SWF 文件加载到另一个 SWF 文件的影片剪辑中，则对加载的 SWF 文件中 `_root` 的任何引用都将解析到宿主 SWF 文件中的根时间轴，而不是加载的 SWF 文件的根时间轴。有时，这可能会导致运行时出现意外情况，例如，当宿主 SWF 文件和加载的 SWF 文件都使用 `_root` 指定变量时可能出现意外情况。

在 Flash Player 7 和更高版本中，您可以使用 `MovieClip._lockroot` 属性强制影片剪辑对 `_root` 的引用解析到它自己的时间轴，而不是包含该影片剪辑的 SWF 文件的时间轴。有关更多信息，请参见第 109 页的“指定加载的 SWF 文件的根时间轴”。

关于访问加载的 SWF 文件中的数据

一个 SWF 文件可以从 Internet 上的任何位置加载另一个 SWF 文件。但是，若要让一个 SWF 文件访问其它 SWF 文件中定义的数据（变量、方法等），这两个文件必须源于同一个域。在 Flash Player 7 和更高版本中，除非加载的 SWF 文件通过调用 `System.security.allowDomain()` 另行指定，否则禁用跨域脚本。

有关更多信息，请参见第 166 页的“Flash Player 安全功能”和第 181 页的第 12 章“动作脚本字典”中的 `System.security.allowDomain()`。

加载外部 MP3 文件

若要在运行时加载 MP3 文件，请使用 `Sound` 类的 `loadSound()` 方法。首先，创建一个 `Sound` 对象：

```
var song_1_sound = new Sound();
```

然后使用这个新对象调用 `loadSound()` 来加载事件或声音流。事件声音在完全加载之后播放；而声音流在它们下载的过程中播放。您可以设置 `loadSound()` 的 `isStreaming` 参数来指定声音是事件声音还是声音流。在加载了事件声音之后，必须调用 `Sound` 类的 `start()` 方法来播放声音。当足够的数据加载到 SWF 文件中之后，声音流就会开始播放；不必使用 `start()`。

例如，以下代码创建一个名为 `classical` 的 `Sound` 对象，然后加载一个名为 `beethoven.mp3` 的 MP3 文件：

```
var classical:Sound = new Sound();
classical.loadSound("http://server.com/mp3s/beethoven.mp3", true);
```

在大多数情况下，将 `isStreaming` 参数设置为 `true`，特别是当加载较大的需要及早开始播放的声音文件时（例如创建 MP3 “自动唱片点唱机”应用程序时）。但是，如果要下载较短的声音剪辑并需要在特定时间（例如，当用户单击某个按钮时）播放它们，则将 `isStreaming` 设置为 `false`。

若要确定声音何时下载完毕，请使用 `Sound.onLoad` 事件处理函数。此事件处理函数自动接收一个布尔值（`true` 或 `false`），该值指示文件是否成功下载。

例如，假设您正在创建一个联机游戏，该游戏根据用户到达游戏中的不同级别使用不同的声音。以下代码将 MP3 文件 (`blastoff.mp3`) 加载到名为 `gameSound` 的 `Sound` 对象中，然后在完成下载时播放该声音：

```
var gameSound = new Sound();
gameSound.onLoad = function (loadedOK) {
    if(loadedOK) {
        gameSound.start();
    }
}
gameSound.loadSound("http://server.com/sounds/blastoff.mp3", false);
```

对于声音文件，Flash Player 仅支持 MP3 声音文件类型。

有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `Sound.loadSound()`、`Sound.start()` 和 `Sound.onLoad`。

读取 MP3 文件中的 ID3 标签

ID3 标签是添加到 MP3 文件的数据字段，这些数据字段包含有关该文件的信息，例如歌曲名称、唱片名称和歌手姓名。

若要读取 MP3 文件中的 ID3 标签，请使用 `Sound.ID3` 属性，其属性对应于正在加载的 MP3 文件中包含的 ID3 标签的名称。若要确定正在下载的 MP3 文件的 ID3 标签何时可用，请使用 `Sound.onID3` 事件处理函数。Flash Player 7 支持版本 1.0、1.1、2.3 和 2.4 标签；不支持版本 2.2 标签。

例如，以下代码将名为 `favoriteSong.mp3` 的 MP3 文件加载到名为 `song` 的 `Sound` 对象中。当该文件的 ID3 标签可用时，名为 `display_txt` 的文本字段显示歌手姓名和歌曲名称。

```
var song = new Sound();
song.onID3 = function () {
    display_txt.text = "Artist:" + song.id3.TCOM + newline;
    display_txt.text += "Song:" + song.id3.TIT2);
}
song.loadSound("mp3s/favoriteSong.mp3, true");
```

因为 ID3 2.0 标签位于 MP3 文件的开始处（在声音数据之前），所以当文件刚开始下载时便可以得到这些标签。但是，ID3 1.0 标签位于文件的末尾（在声音数据之后），这些标签直到整个 MP3 文件完成下载后才可用。

每次有新的 ID3 数据可用时都会调用 `onID3` 事件处理函数。这意味着如果 MP3 文件包含 ID3 2.0 标签和 ID3 1.0 标签，`onID3` 处理函数将被调用两次，这是因为这些标签位于文件中的不同部分。

有关支持的 ID3 标签的列表，请参见第 586 页的 `Sound.ID3`。

动态回放外部 FLV 文件

作为将视频导入 Flash 创作环境的替代方法，您可以在 Flash Player 中使用动作脚本动态回放外部 FLV 文件。您可以通过 HTTP 地址或从本地文件系统回放 FLV 文件。若要回放 FLV 文件，您可以使用 `NetConnection` 和 `NetStream` 类以及 `Video` 类的 `attachVideo()` 方法。（有关完整的信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `NetConnection` 类、`NetStream` 类和 `Video.attachVideo()` 条目。）

您可以通过将视频导入 Flash 创作工具然后将其以 FLV 文件的形式导出来创建 FLV 文件。（请参见“使用 Flash”帮助中的“Macromedia Flash 视频”。）如果您具有 Flash Professional，则可以使用 FLV Export 插件从支持的视频编辑应用程序导出 FLV 文件。（请参见“使用 Flash”帮助中的“从视频编辑应用程序导出 FLV 文件（仅限 Flash Professional）”。）

使用外部 FLV 文件可以提供使用导入的视频时不可用的某些功能：

- 您可以在 Flash 文档中使用较长的视频剪辑而无需放慢回放速度。外部 FLV 文件是使用缓存的内存播放的。这意味着将以较小的块存储并动态访问较大的文件，不需要嵌入视频文件那么多内存。
- 外部 FLV 文件可以和它在其中播放的 Flash 文档具有不同的帧频。例如，可以将 Flash 文档帧频设置为 30 fps 并将视频帧频设置为 21 fps。这使您在确保平滑的视频回放方面具有更大的控制权。
- 通过外部 FLV 文件，加载视频文件时不需要中断 Flash 文档回放。导入的视频文件有时可能需要中断文档回放来执行某些功能；例如访问 CD-ROM 驱动器。FLV 文件可以独立于 Flash 文档执行功能，因此不会中断回放。

- 对于外部 FLV 文件，为视频内容加字幕更加简单，这是因为您可以使用事件处理函数访问视频的元数据。

以下步骤介绍如何回放名为 videoFile.flv 的文件，该文件与您的 SWF 文件存储在同一个位置。

在 Flash 文档中回放外部 FLV 文件：

- 1 当在 Flash 创作工具中打开文档时，在“库”面板（“窗口” > “库”）中选择“库选项”菜单中的“新建视频”以创建一个视频对象。
- 2 将视频对象从“库”面板拖放到舞台上。这将创建一个视频对象实例。
- 3 当在舞台上选中视频对象时，在属性检查器（“窗口” > “属性”）中的“实例名称”文本框中输入 my_video。
- 4 打开“组件”面板（“窗口” > “开发面板” > “组件”）并将 TextArea 组件拖到舞台上。
- 5 当在舞台上选中 TextArea 对象时，在属性检查器的“实例名称”文本框中输入 status。
- 6 在时间轴中选择第 1 帧，然后打开“动作”面板（“窗口” > “开发面板” > “动作”）。
- 7 将以下代码添加到“动作”面板中：

```
// 创建一个 NetConnection 对象：
var netConn:NetConnection = new NetConnection();
// 创建一个本地流连接：
netConn.connect(null);
// 创建一个 NetStream 对象并定义一个 onStatus() 函数：
var netStream:NetStream = new NetStream(netConn);
netStream.onStatus = function(infoObject) {
    status.text += "Status (NetStream)" + newline;
    status.text += "Level:"+infoObject.level + newline;
    status.text += "Code:"+infoObject.code + newline;
};
// 将 NetStream 视频输入信号附加到 Video 对象：
my_video.attachVideo(netStream);
// 设置缓冲时间：
netStream.setBufferTime(5);
// 播放 FLV 文件：
netStream.play("videoFile.flv");
```

预加载外部媒体

动作脚本提供多种方式预加载外部媒体或跟踪外部媒体的下载进度。若要预加载 SWF 和 JPEG 文件，请使用 MovieClipLoader 类，该类提供事件侦听器机制用于检查下载进度。此类是 Flash Player 7 中新引入的。有关更多信息，请参见第 175 页的“预加载 SWF 和 JPEG 文件”。

若要跟踪 MP3 文件的下载进度，可以使用 Sound.getBytesLoaded() 和 Sound.getBytesTotal() 方法；若要跟踪 FLV 文件的下载进度，可以使用 NetStream.bytesLoaded 和 NetStream.bytesTotal 属性。有关更多信息，请参见第 177 页的“预加载 MP3 和 FLV 文件”。

预加载 SWF 和 JPEG 文件

若要将 SWF 和 JPEG 文件预加载到影片剪辑实例中，您可以使用 MovieClipLoader 类。此类提供事件侦听器机制，可以提供有关将文件下载到影片剪辑的状态的通知。使用 MovieClipLoader 对象预加载 SWF 和 JPEG 文件包含以下步骤：

创建新的 MovieClipLoader 对象 您可以使用单个 MovieClipLoader 对象跟踪多个文件的下载进度，或为每个文件的进度创建一个单独的对象。

```
var loader:MovieClipLoader = new MovieClipLoader();
```

创建侦听器对象和创建事件处理函数 侦听器对象可以是任何动作脚本对象，例如通用 Object 对象、影片剪辑或自定义组件。

例如，以下代码创建一个名为 loadListener 的通用侦听器对象，然后为自身定义 onLoadStart、onLoadProgress 和 onLoadComplete 函数。

```
// 创建侦听器对象：
var loadListener:Object = new Object();
loadListener.onLoadStart = function (loadTarget) {
    trace("Loading into " + loadTarget + " has started.");
}
loadListener.onLoadProgress = function(loadTarget, bytesLoaded, bytesTotal) {
    var percentLoaded = bytesLoaded/bytesTotal * 100;
    trace("%" + percentLoaded + " into target " + loadTarget);
}
loadListener.onLoadComplete = function(loadTarget) {
    trace("Load completed into:" + loadTarget);
}
```

向 MovieClipLoader 对象注册侦听器对象 为了让侦听器对象接收加载事件，您必须将其向 MovieClipLoader 对象进行注册。

```
loader.addListener(loadListener);
```

开始将文件（JPEG 或 SWF）加载到目标剪辑 若要开始 JPEG 或 SWF 文件的下载，可以使用 MovieClipLoader.loadClip() 方法。

```
loader.loadClip("scene_2.swf");
```

注意：您只能使用 MovieClipLoader 方法跟踪用 MovieClipLoader.loadClip() 方法加载的文件的下载进度。不能使用 loadMovie() 函数或 MovieClip.loadMovie() 方法。

以下示例使用 ProgressBar 组件的 setProgress() 方法显示 SWF 文件的下载进度。（请参见“使用组件”帮助中的“ProgressBar 组件”。）

使用 ProgressBar 组件显示下载进度：

- 1 在新的 Flash 文档中，在舞台上创建一个影片剪辑并将其命名为 target_mc。
- 2 打开“组件”面板（“窗口” > “开发面板” > “组件”）。
- 3 将 ProgressBar 组件从“组件”面板拖到舞台上。
- 4 在属性检查器中，将 ProgressBar 组件命名为 pBar，然后在“参数”选项卡上，从“模式”弹出式菜单中选择“手动”。
- 5 选择时间轴中的第 1 帧，然后打开“动作”面板（“窗口” > “开发面板” > “动作”）。
- 6 将以下代码添加到“动作”面板中：

```
// 创建一个 MovieClipLoader 对象和一个侦听器对象
myLoader = new MovieClipLoader();
myListener = new Object();
// 将 MovieClipLoader 回调函数添加到您的侦听器对象中
myListener.onLoadStart = function(clip) {
    // 此事件会在加载开始时触发一次
    pBar.label = "Now loading:" + clip;
};
myListener.onLoadProgress = function(clip, bytesLoaded, bytesTotal) {
    var percentLoaded = int (100*(bytesLoaded/bytesTotal));
    pBar.setProgress(bytesLoaded, bytesTotal);
};myLoader.addListener(myListener);
myLoader.loadClip("veryLargeFile.swf", target_mc);
```


7 通过选择 “控制” > “测试影片” 对文档进行测试。

有关更多信息，请参见第 181 页的第 12 章 “动作脚本字典” 中的 [MovieClipLoader](#) 类条目。

预加载 MP3 和 FLV 文件

若要预加载 MP3 和 FLV 文件，可以使用 `setInterval()` 函数创建 “轮询” 机制，该机制以预先确定的时间间隔检查为 `Sound` 或 `NetStream` 对象加载的字节数。若要跟踪 MP3 文件的下载进度，可以使用 `Sound.getBytesLoaded()` 和 `Sound.getBytesTotal()` 方法；若要跟踪 FLV 文件的下载进度，可以使用 `NetStream.bytesLoaded` 和 `NetStream.bytesTotal` 属性。

以下代码使用 `setInterval()` 以预先确定的时间间隔检查为 `Sound` 或 `NetStream` 对象加载的字节数。

```
// 创建一个新的 Sound 对象来播放声音。
var songTrack = new Sound();
// 创建一个跟踪下载进度的轮询函数。
// 这是进行轮询的函数。它检查
// 作为引用传递的 Sound 对象的下载进度。
checkProgress = function (soundObj) {
    var bytesLoaded = soundObj.getBytesLoaded();
    var bytesTotal = soundObj.getBytesTotal();
    var percentLoaded = Math.floor(bytesLoaded/bytesTotal * 100);
    trace("%" + percentLoaded + " loaded.");
}
// 当文件完成加载之后，清除间隔轮询。
songTrack.onLoad = function () {
    clearInterval(poll);
}
// 加载 MP3 流文件并开始调用 checkProgress()
songTrack.loadSound("beethoven.mp3", true);
var poll = setInterval(checkProgress, 1000, songTrack);
```

您可以使用同类轮询方法预加载外部 FLV 文件。若要获取总字节数和当前为 FLV 文件加载的字节数，请使用 `NetStream.bytesLoaded` 和 `NetStream.bytesTotal` 属性。

预加载 FLV 文件的另一种途径是使用 `NetStream.setBufferTime()` 方法。此方法采用单个参数，该参数指示回放开始前 FLV 流进行下载的秒数。

有关更多信息，请参见第 181 页的第 12 章 “动作脚本字典” 中的 [MovieClip.getBytesLoaded\(\)](#)、[MovieClip.getBytesTotal\(\)](#)、[NetStream.bytesLoaded](#)、[NetStream.bytesTotal](#)、[NetStream.setBufferTime\(\)](#)、[setInterval\(\)](#)、[Sound.getBytesLoaded\(\)](#) 和 [Sound.getBytesTotal\(\)](#)。

第 V 部分

脚本参考

这一部分包含“动作脚本字典”，它提供了动作脚本语言中每个元素的语法和用法信息。这一部分还包含附录，这些附录提供了编写脚本时可能需要查阅的参考资料。

第 12 章：动作脚本字典	181
附录 A：错误消息	739
附录 B：运算符的优先级和结合律	743
附录 C：键盘键和键控代码值	745
附录 D：为早期的 Flash Player 版本编写脚本	749
附录 E：使用动作脚本 1 进行面向对象的编程	753

第 12 章

动作脚本字典

本字典描述 Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 中的动作脚本元素的语法和用法。若要使用脚本中的示例，请复制本字典中的示例代码，然后将其粘贴到“脚本”窗格或外部脚本文件中。本字典列出了所有动作脚本元素，包括运算符、关键字、语句、动作、属性、函数、类和方法。有关所有字典条目的概述，请参见第 182 页的“字典内容”；若要查找其类未知的运算符或方法，可以从本节中的表开始入手。有关组件的信息，请参见使用组件。

本字典中有两种类型的条目：

- 表示运算符、关键字、函数、变量、属性、方法和语句的单独条目
- 提供与内置类有关的一般信息的类条目

范例条目中的信息用于解释在这些类型的条目中使用的结构和约定。

大多数动作脚本元素的范例条目

下面的范例字典条目解释用于除类之外的所有动作脚本元素的约定。

条目标题

所有条目均按字母顺序列出。这种字母顺序不考虑大写、前导下划线等等。

可用性

除非另有说明，否则，“可用性”部分中将指出哪些版本的 Flash Player 支持该元素。这里所说的版本与创作该内容所用的 Flash 的版本不同。例如，如果您使用 Macromedia Flash MX 2004 或 Macromedia Flash MX Professional 2004 为 Flash Player 6 创建内容，则只能使用可用于 Flash Player 6 的动作脚本元素。

在某些情况下，该部分还指出哪一版本的创作工具支持某一元素。有关示例，请参见 `System.setClipboard()`。

最后，如果某一元素只在动作脚本 2.0 中得到支持，则在该部分中还将指出这一信息。

用法

这一部分提供在代码中使用该动作脚本元素的正确语法。语法的必需部分以代码字体表示，用户提供的代码以斜体代码字体表示。括号 ([]) 表示可选参数。

参数

这一部分描述在语法中列出的所有参数。

返回

这一部分说明该元素返回什么值（如果有的话）。

说明

这一部分标识元素类型（例如运算符、方法、函数等等），然后描述如何使用该元素。

示例

这一部分提供代码范例，演示如何使用该元素。

另请参见

这一部分列出相关的动作脚本字典条目。

类的范例条目

下面的范例字典条目解释用于内置动作脚本类的约定。类以及所有其它元素按字母顺序在字典中列出。

条目标题

条目标题提供类的名称。类名后面是一般性描述信息。

方法和属性摘要表

每个类条目都包含一个列出所有相关方法的表。如果该类具有属性（通常为常数）、事件处理函数或事件侦听器，则这些元素在另外的表中摘要列出。这些表中列出的所有元素还有其自己的字典条目，这些条目跟在该类条目的后面。

构造函数

如果某个类要求使用构造函数来访问其方法和属性，则在每个类条目中描述该构造函数。这种描述中包含其它字典条目所具有的全部标准元素（语法、说明等等）。

方法和属性列表

类的方法和属性在该类条目后面按字母顺序列出。

字典内容

所有字典条目均按字母顺序列出。但是，有些运算符是符号，所以按 ASCII 码顺序列出。另外，与某个类关联的方法将与该类名称一起列出，例如，Math 类的 `abs()` 方法以 `Math.abs()` 的形式列出。

下面的两个表可帮助您找到这些元素。第一个表按照运算符号在字典中的出现顺序列举它们。第二个表列出所有其它的动作脚本元素。

运算符号	参见条目
--	-- (递减)
++	++ (递增)
!	! (逻辑 NOT)
!=	!= (不等于)
!==	!== (不全等)
%	% (模)
%=	%= (模赋值)
&	& (按位 AND 运算符)
&&	&& (逻辑 AND)
&=	&= (按位 AND 赋值)
()	() (括号)
-	- (减号)
*	* (乘号)
*=	*= (相乘赋值)
,	, (逗号)
.	. (点)
:	: (类型)
?:	?: (条件)
/	/ (除号)
//	// (注释分隔符)
/*	/* (注释分隔符)
/=	/= (除法赋值)
[]	[] (数组访问)
^	^ (按位 XOR)
^=	^= (按位 XOR 赋值)
{ }	{ } (对象初始值设定项)
	(按位 OR)
	(逻辑 OR)
=	= (按位 OR 赋值)
~	~ (按位 NOT)
+	+ (加号)

运算符号	参见条目
+=	+= （加法赋值）
<	< （小于）
<<	<< （按位向左移位）
<<=	<<= （按位向左移位并赋值）
<=	<= （小于等于）
<>	<> （不等于）
=	= （赋值）
-=	-= （减法赋值）
==	== （等于）
===	=== （全等）
>	> （大于）
>=	>= （大于等于）
>>	>> （按位向右移位）
>>=	>>= （按位向右移位并赋值）
>>>	>>> （按位无符号向右移位）
>>>=	>>>= （按位无符号向右移位并赋值）

下表列出所有不是运算符的动作脚本元素。

动作脚本元素	参见条目
<code>#endinitclip</code>	#endinitclip
<code>#include</code>	#include
<code>#initclip</code>	#initclip
<code>__proto__</code>	Object.__proto__
<code>_accProps</code>	_accProps
<code>_alpha</code>	MovieClip._alpha , Button._alpha , TextField._alpha
<code>_currentframe</code>	MovieClip._currentframe
<code>_droptarget</code>	MovieClip._droptarget
<code>_focusrect</code>	_focusrect , Button._focusrect , MovieClip._focusrect
<code>_framesloaded</code>	MovieClip._framesloaded
<code>_global</code>	_global 对象
<code>_height</code>	Button._height , MovieClip._height , TextField._height
<code>_highquality</code>	_highquality , Button._highquality , MovieClip._highquality , TextField._highquality
<code>_lockroot</code>	MovieClip._lockroot

动作脚本元素	参见条目
<code>_name</code>	Button._name , MovieClip._name , TextField._name
<code>_parent</code>	_parent , Button._parent , MovieClip._parent , TextField._parent
<code>_quality</code>	_quality , Button._quality , TextField._quality
<code>_root</code>	_root
<code>_rotation</code>	Button._rotation , MovieClip._rotation , TextField._rotation
<code>_soundbuftime</code>	_soundbuftime , Button._soundbuftime , MovieClip._soundbuftime , TextField._soundbuftime
<code>_target</code>	Button._target , MovieClip._target , TextField._target
<code>_totalframes</code>	MovieClip._totalframes
<code>_url</code>	Button._url , MovieClip._url , TextField._url
<code>_visible</code>	Button._visible , MovieClip._visible , TextField._visible
<code>_width</code>	Button._width , MovieClip._width , TextField._width
<code>_x</code>	Button._x , MovieClip._x , TextField._x
<code>_xmouse</code>	Button._xmouse , MovieClip._xmouse , TextField._xmouse
<code>_xscale</code>	Button._xscale , MovieClip._xscale , TextField._xscale
<code>_y</code>	Button._y , MovieClip._y , TextField._y
<code>_ymouse</code>	Button._ymouse , MovieClip._ymouse , TextField._ymouse
<code>_yscale</code>	Button._yscale , MovieClip._yscale , TextField._yscale
<code>abs</code>	Math.abs()
Accessibility	Accessibility 类
<code>acos</code>	Math.acos()
<code>activityLevel</code>	Camera.activityLevel , Microphone.activityLevel
<code>add</code>	add
<code>addListener</code>	Key.addListener() , Mouse.addListener() , MovieClipLoader.addListener() , Selection.addListener() , Stage.addListener() , TextField.addListener()
<code>addPage</code>	PrintJob.addPage()
<code>addProperty</code>	Object.addProperty()
<code>addRequestHeader</code>	LoadVars.addRequestHeader() , XML.addRequestHeader()
<code>align</code>	Stage.align , TextFormat.align
<code>allowDomain</code>	LocalConnection.allowDomain , System.security.allowDomain()
<code>allowInsecureDomain</code>	LocalConnection.allowInsecureDomain , System.security.allowInsecureDomain()
<code>and</code>	and

动作脚本元素	参见条目
appendChild	XML.appendChild()
apply	Function.apply()
Arguments	Arguments 类
Array	Array 类 , Array()
asfunction	asfunction
asin	Math.asin()
atan	Math.atan()
atan2	Math.atan2()
attachAudio	MovieClip.attachAudio()
attachMovie	MovieClip.attachMovie()
attachSound	Sound.attachSound()
attachVideo	Video.attachVideo()
attributes	XML.attributes
autosize	TextField.autoSize
avHardwareDisable	System.capabilities.avHardwareDisable
background	TextField.background
backgroundColor	TextField.backgroundColor
BACKSPACE	Key.BACKSPACE
bandwidth	Camera.bandwidth
beginFill	MovieClip.beginFill()
beginGradientFill	MovieClip.beginGradientFill()
blockIndent	TextFormat.blockIndent
bold	TextFormat.bold
布尔值	Boolean() , Boolean 类
border	TextField.border
borderColor	TextField.borderColor
bottomScroll	TextField.bottomScroll
break	break
bufferLength	NetStream.bufferLength
bufferTime	NetStream.bufferTime
builtInItems	ContextMenu.builtInItems
bullet	TextFormat.bullet
按钮	Button 类

动作脚本元素	参见条目
bytesLoaded	NetStream.bytesLoaded
bytesTotal	NetStream.bytesTotal
call	call() , Function.call()
callee	arguments.callee
caller	arguments.caller
摄像头	Camera 类
capabilities	System.capabilities 对象
CAPSLock	Key.CAPSLock
caption	ContextMenuItem.caption
case	case
catch	try..catch..finally
ceil	Math.ceil()
charAt	String.charAt()
charCodeAt	String.charCodeAt()
childNodes	XML.childNodes
chr	chr
class	class
clear	MovieClip.clear() , SharedObject.clear() , Video.clear()
clearInterval	clearInterval()
cloneNode	XML.cloneNode()
close	LocalConnection.close() , NetStream.close() , XMLSocket.close()
Color	Color 类 , TextFormat.color
concat	Array.concat() , String.concat()
connect	LocalConnection.connect() , NetConnection.connect() , XMLSocket.connect()
condenseWhite	TextField.condenseWhite
构造函数	Array 类 , Boolean 类 , Camera 类 , Color 类 , ContextMenu 类 , ContextMenuItem 类 , Date 类 , Error 类 , LoadVars 类 , LocalConnection 类 , Microphone 类 , NetConnection 类 , NetStream 类 , Number 类 , Object 类 , PrintJob 类 , SharedObject 类 , Sound 类 , String 类 , TextField.StyleSheet 类 , TextFormat 类 , XML 类 , XMLSocket 类
contentType	LoadVars.contentType , XML.contentType
ContextMenu	ContextMenu 类
ContextMenuItem	ContextMenuItem 类

动作脚本元素	参见条目
continue	continue
CONTROL	Key.CONTROL
copy	ContextMenu.copy() , ContextMenuItem.copy()
cos	Math.cos()
createElement	XML.createElement()
createEmptyMovieClip	MovieClip.createEmptyMovieClip()
createTextField	MovieClip.createTextField()
createTextNode	XML.createTextNode()
currentFps	Camera.currentFps , NetStream.currentFps
curveTo	MovieClip.curveTo()
CustomActions	CustomActions 类
customItems	ContextMenu.customItems
data	SharedObject.data
日期	Date 类
deblocking	Video.deblocking
default	default
delete	delete
DELETEKEY	Key.DELETEKEY
do while	do while
docTypeDecl	XML.docTypeDecl
domain	LocalConnection.domain()
DOWN	Key.DOWN
duplicateMovieClip	duplicateMovieClip() , MovieClip.duplicateMovieClip()
duration	Sound.duration
dynamic	dynamic
E	Math.E
else	else
else if	else if
embedFonts	TextField.embedFonts
enabled	Button.enabled , ContextMenuItem.enabled , MovieClip.enabled
END	Key.END
endFill	MovieClip.endFill()
ENTER	Key.ENTER

动作脚本元素	参见条目
eq	eq (等于 - 字符串专用)
Error	Error 类
ESCAPE (常数)	Key.ESCAPE
escape (函数)	escape
eval	eval()
exactSettings	System.exactSettings
exp	Math.exp()
extends	extends
false	false
finally	try..catch..finally
findText	TextSnapshot.findText()
firstChild	XML.firstChild
floor	Math.floor()
flush	SharedObject.flush()
focusEnabled	MovieClip.focusEnabled
font	TextFormat.font
for	for
for..in	for..in
fps	Camera.fps
fromCharCode	String.fromCharCode()
fscommand	fscommand()
function	function , Function 类
gain	Microphone.gain
ge	ge (大于或等于 - 字符串专用)
get	Camera.get() , CustomActions.get() , get , Microphone.get()
getAscii	Key.getAscii()
getBeginIndex	Selection.getBeginIndex()
getBounds	MovieClip.getBounds()
getBytesLoaded	LoadVars.getBytesLoaded() , MovieClip.getBytesLoaded() , Sound.getBytesLoaded() , XML.getBytesLoaded()
getBytesTotal	LoadVars.getBytesTotal() , MovieClip.getBytesTotal() , Sound.getBytesTotal() , XML.getBytesTotal()
getCaretIndex	Selection.getCaretIndex()
getCode	Key.getCode()

动作脚本元素	参见条目
getCount	TextSnapshot.getCount()
getDate	Date.getDate()
getDay	Date.getDay()
getDepth	Button.getDepth() , MovieClip.getDepth() , TextField.getDepth()
getEndIndex	Selection.getEndIndex()
getFocus	Selection.getFocus()
getFontList	TextField.getFontList()
getFullYear	Date.getFullYear()
getHours	Date.getHours()
getInstanceAtDepth	MovieClip.getInstanceAtDepth()
getLocal	SharedObject.getLocal()
getMilliseconds	Date.getMilliseconds()
getMinutes	Date.getMinutes()
getMonth	Date.getMonth()
getNewTextFormat	TextField.getNewTextFormat()
getNextHighestDepth	MovieClip.getNextHighestDepth()
getPan	Sound.getPan()
getProgress	MovieClipLoader.getProgress()
getProperty	getProperty
getRGB	Color.getRGB()
getSeconds	Date.getSeconds()
getSelected	TextSnapshot.getSelected()
getSelectedText	TextSnapshot.getSelectedText()
getSize	SharedObject.getSize()
getStyle	TextField.StyleSheet.getStyle()
getStyleNames	TextField.StyleSheet.getStyleNames()
getSWFVersion	MovieClip.getSWFVersion()
getText	TextSnapshot.getText()
getTextExtent	TextFormat.getTextExtent()
getTextFormat	TextField.getTextFormat()
getTextSnapshot	MovieClip.getTextSnapshot()
getTime	Date.getTime()
getTimer	getTimer

动作脚本元素	参见条目
getTimezoneOffset	Date.getTimezoneOffset()
getTransform	Color.getTransform() , Sound.getTransform()
getURL	getURL() , MovieClip.getURL()
getUTCDate	Date.getUTCDate()
getUTCDay	Date.getUTCDay()
getUTCFullYear	Date.getUTCFullYear()
getUTCHours	Date.getUTCHours()
getUTCMilliseconds	Date.getUTCMilliseconds()
getUTCMinutes	Date.getUTCMinutes()
getUTCMonth	Date.getUTCMonth()
getUTCSeconds	Date.getUTCSeconds()
getVersion	getVersion
getVolume	Sound.getVolume()
getYear	Date.getYear()
globalToLocal	MovieClip.globalToLocal()
goto	gotoAndPlay() , gotoAndStop()
gotoAndPlay	gotoAndPlay() , MovieClip.gotoAndPlay()
gotoAndStop	gotoAndStop() , MovieClip.gotoAndStop()
gt	gt (大于 - 字符串专用)
hasAccessibility	System.capabilities.hasAccessibility
hasAudio	System.capabilities.hasAudio
hasAudioEncoder	System.capabilities.hasAudioEncoder
hasChildNodes	XML.hasChildNodes()
hasEmbeddedVideo	System.capabilities.hasEmbeddedVideo
hasMP3	System.capabilities.hasMP3
hasPrinting	System.capabilities.hasPrinting
hasScreenBroadcast	System.capabilities.hasScreenBroadcast
hasScreenPlayback	System.capabilities.hasScreenPlayback
hasStreamingAudio	System.capabilities.hasStreamingAudio
hasStreamingVideo	System.capabilities.hasStreamingVideo
hasVideoEncoder	System.capabilities.hasVideoEncoder
height	Camera.height , Stage.height , Video.height
hide	Mouse.hide()

动作脚本元素	参见条目
hideBuiltInItems	ContextMenu.hideBuiltInItems()
hitArea	MovieClip.hitArea
hitTest	MovieClip.hitTest()
hitTestTextNearPos	TextSnapshot.hitTestTextNearPos()
HOME	Key.HOME
hscroll	TextField.hscroll
html	TextField.html
htmlText	TextField.htmlText
ID3	Sound.ID3
if	if
ifFrameLoaded	ifFrameLoaded
ignoreWhite	XML.ignoreWhite
implements	implements
import	import
indent	TextFormat.indent
index	Camera.index , Microphone.index
indexOf	String.indexOf()
Infinity	Infinity
-Infinity	-Infinity
INSERT	Key.INSERT
insertBefore	XML.insertBefore()
install	CustomActions.install()
instanceof	instanceof
int	int
interface	interface
isActive	Accessibility.isActive()
isDebugger	System.capabilities.isDebugger
isDown	Key.isDown()
isFinite	isFinite
isNaN	isNaN()
isToggled	Key.isToggled()
italic	TextFormat.italic
join	Array.join()

动作脚本元素	参见条目
Key	Key 类
language	System.capabilities.language
lastChild	XML.lastChild
lastIndexOf	String.lastIndexOf()
le	le (小于或等于 - 字符串专用)
leading	TextFormat.leading
LEFT	Key.LEFT
leftMargin	TextFormat.leftMargin
length	length , arguments.length , Array.length , String.length , TextField.length
level	_level
lineStyle	MovieClip.lineStyle()
lineTo	MovieClip.lineTo()
list	CustomActions.list()
LN10	Math.LN10
LN2	Math.LN2
load	LoadVars.load() , TextField.StyleSheet.load() , XML.load() ,
loadClip	MovieClipLoader.loadClip()
loaded	LoadVars.loaded , XML.loaded
loadMovie	loadMovie() , MovieClip.loadMovie()
loadMovieNum	loadMovieNum()
loadSound	Sound.loadSound()
loadVariables	loadVariables() , MovieClip.loadVariables()
loadVariablesNum	loadVariablesNum()
LoadVars	LoadVars 类
LocalConnection	LocalConnection 类
localFileReadDisable	System.capabilities.localFileReadDisable
localToGlobal	MovieClip.localToGlobal()
log	Math.log()
LOG10E	Math.LOG10E
LOG2E	Math.LOG2E
lt	lt (小于 - 字符串专用)
manufacturer	System.capabilities.manufacturer

动作脚本元素	参见条目
Math	Math 类
max	Math.max()
MAX_VALUE	Number.MAX_VALUE
maxChars	TextField.maxChars
maxhscroll	TextField.maxhscroll
maxscroll	maxscroll , TextField.maxscroll
mbchr	mbchr
mblength	mblength
mbord	mbord
mbsubstring	mbsubstring
menu	Button.menu , MovieClip.menu , TextField.menu
message	Error.message
麦克风	Microphone 类
min	Math.min()
MIN_VALUE	Number.MIN_VALUE
MMExecute	MMExecute()
motionLevel	Camera.motionLevel
motionTimeOut	Camera.motionTimeOut
Mouse	Mouse 类
mouseWheelEnabled	TextField.mouseWheelEnabled
moveTo	MovieClip.moveTo()
MovieClip	MovieClip 类
MovieClipLoader	MovieClipLoader 类
multiline	TextField.multiline
muted	Camera.muted , Microphone.muted
name	Error.name , Microphone.name
names	Camera.names , Microphone.names
NaN	NaN , Number.NaN
ne	ne (不等于 - 字符串专用)
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
NetConnection	NetConnection 类
NetStream	NetStream 类
new (运算符)	new

动作脚本元素	参见条目
newline	newline
nextFrame	nextFrame() , MovieClip.nextFrame()
nextScene	nextScene()
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
not	not
null	null
数字	Number() , Number 类
对象	Object 类 , Object()
on	on()
onActivity	Camera.onActivity , Microphone.onActivity
onChanged	TextField.onChanged
onClipEvent	onClipEvent()
onClose	XMLSocket.onClose()
onConnect	XMLSocket.onConnect()
onData	LoadVars.onData , MovieClip.onData , XML.onData , XMLSocket.onData()
onDragOut	Button.onDragOut , MovieClip.onDragOut
onDragOver	Button.onDragOver , MovieClip.onDragOver
onEnterFrame	MovieClip.onEnterFrame
onID3	Sound.onID3
onKeyDown	Button.onKeyDown , Key.onKeyDown , MovieClip.onKeyDown
onKeyUp	Button.onKeyUp , Key.onKeyUp , MovieClip.onKeyUp
onKillFocus	Button.onKillFocus , MovieClip.onKillFocus , TextField.onKillFocus
onLoad	LoadVars.onLoad , MovieClip.onLoad , Sound.onLoad , TextField.StyleSheet.onLoad , XML.onLoad()
onLoadComplete	MovieClipLoader.onLoadComplete()
onLoadError	MovieClipLoader.onLoadError()
onLoadInit	MovieClipLoader.onLoadInit()
onLoadProgress	MovieClipLoader.onLoadProgress()
onLoadStart	MovieClipLoader.onLoadStart()
onMouseDown	Mouse.onMouseDown , MovieClip.onMouseDown

动作脚本元素	参见条目
onMouseMove	Mouse.onMouseMove , MovieClip.onMouseMove
onMouseUp	Mouse.onMouseUp , MovieClip.onMouseUp
onMouseWheel	Mouse.onMouseWheel
onPress	Button.onPress , MovieClip.onPress
onRelease	Button.onRelease , MovieClip.onRelease
onReleaseOutside	Button.onReleaseOutside , MovieClip.onReleaseOutside
onResize	Stage.onResize
onRollOut	Button.onRollOut , MovieClip.onRollOut
onRollOver	Button.onRollOver , MovieClip.onRollOver
onScroller	TextField.onScroller
onSelect	ContextMenu.onSelect , ContextMenu.onSelect
onSetFocus	Button.onSetFocus , MovieClip.onSetFocus , Selection.onSetFocus , TextField.onSetFocus
onSoundComplete	Sound.onSoundComplete
onStatus	Camera.onStatus , LocalConnection.onStatus , Microphone.onStatus , NetStream.onStatus , SharedObject.onStatus , System.onStatus
onUnload	MovieClip.onUnload
onUpdate	onUpdate
onXML	XMLSocket.onXML()
or (逻辑 OR)	or
ord	ord
os	System.capabilities.os
parentNode	XML.parentNode
parseCSS	TextField.StyleSheet.parseCSS()
parseFloat	parseFloat()
parseInt	parseInt
parseXML	XML.parseXML()
password	TextField.password
pause	NetStream.pause()
PGDN	Key.PGDN
PGUP	Key.PGUP
PI	Math.PI
pixelAspectRatio	System.capabilities.pixelAspectRatio
play	play() , MovieClip.play() , NetStream.play()

动作脚本元素	参见条目
playerType	System.capabilities.playerType
pop	Array.pop()
position	Sound.position
POSITIVE_INFINITY	Number.POSITIVE_INFINITY
pow	Math.pow()
prevFrame	prevFrame() , MovieClip.prevFrame()
previousSibling	XML.previousSibling
prevScene	prevScene()
print	print()
printAsBitmap	printAsBitmap()
printAsBitmapNum	printAsBitmapNum()
PrintJob	PrintJob 类
printNum	printNum()
private	private
prototype	Function.prototype
public	public
push	Array.push()
quality	Camera.quality
random	random , Math.random()
rate	Microphone.rate
registerClass	Object.registerClass()
removeListener	Key.removeListener() , Mouse.removeListener() , MovieClipLoader.removeListener() , Selection.removeListener() , Stage.removeListener() , TextField.removeListener()
removeMovieClip	removeMovieClip() , MovieClip.removeMovieClip()
removeNode	XML.removeNode()
removeTextField	TextField.removeTextField()
replaceSel	TextField.replaceSel()
replaceText	TextField.replaceText()
resolutionX	System.capabilities.screenResolutionX
resolutionY	System.capabilities.screenResolutionY
restrict	TextField.restrict
return	return
reverse	Array.reverse()

动作脚本元素	参见条目
RIGHT	Key.RIGHT
rightMargin	TextFormat.rightMargin
round	Math.round()
scaleMode	Stage.scaleMode
screenColor	System.capabilities.screenColor
screenDPI	System.capabilities.screenDPI
screenResolutionX	System.capabilities.screenResolutionX
screenResolutionY	System.capabilities.screenResolutionY
scroll	scroll , TextField.scroll
seek	NetStream.seek()
selectable	TextField.selectable
Selection	Selection 类
send	LoadVars.send() , LocalConnection.send() , PrintJob.send() , XML.send() , XMLSocket.send()
sendAndLoad	LoadVars.sendAndLoad() , XML.sendAndLoad()
separatorBefore	ContextMenuItem.separatorBefore
serverString	System.capabilities.serverString
set	set
set variable	set variable
setBufferTime	NetStream.setBufferTime()
setClipboard	System.setClipboard()
setDate	Date.setDate()
setFocus	Selection.setFocus()
setFullYear	Date.setFullYear()
setGain	Microphone.setGain()
setHours	Date.setHours()
setInterval	setInterval()
setMask	MovieClip.setMask()
setMilliseconds	Date.setMilliseconds()
setMinutes	Date.setMinutes()
setMode	Camera.setMode()
setMonth	Date.setMonth()
setMotionLevel	Camera.setMotionLevel()

动作脚本元素	参见条目
setNewTextFormat	TextField.setNewTextFormat()
setPan	Sound.setPan()
setProperty	setProperty()
setQuality	Camera.setQuality()
setRate	Microphone.setRate()
setRGB	Color.setRGB()
setSeconds	Date.setSeconds()
setSelectColor	TextSnapshot.setSelectColor()
setSelected	TextSnapshot.setSelected()
setSelection	Selection.setSelection()
setSilenceLevel	Microphone.setSilenceLevel()
setStyle	TextField.StyleSheet.setStyle()
setTextFormat	TextField.setTextFormat()
setTime	Date.setTime()
setTransform	Color.setTransform() , Sound.setTransform()
setUseEchoSuppression	Microphone.setUseEchoSuppression()
setUTCDate	Date.setUTCDate()
setUTCFullYear	Date.setUTCFullYear()
setUTCHours	Date.setUTCHours()
setUTCMilliseconds	Date.setUTCMilliseconds()
setUTCMinutes	Date.setUTCMinutes()
setUTCMonth	Date.setUTCMonth()
setUTCSeconds	Date.setUTCSeconds()
setVolume	Sound.setVolume()
setYear	Date.setYear()
SharedObject	SharedObject 类
SHIFT (常数)	Key.SHIFT
shift (方法)	Array.shift()
show	Mouse.show()
showMenu	Stage.showMenu
showSettings	System.showSettings()
silenceLevel	Microphone.silenceLevel()

动作脚本元素	参见条目
silenceTimeout	Microphone.silenceTimeout()
sin	Math.sin()
size	TextFormat.size
slice	Array.slice() , String.slice()
smoothing	Video.smoothing
sort	Array.sort()
sortOn	Array.sortOn()
Sound	Sound 类
SPACE	Key.SPACE
splice	Array.splice()
split	String.split()
sqrt	Math.sqrt()
SQRT1_2	Math.SQRT1_2
SQRT2	Math.SQRT2
Stage	Stage 类
start	PrintJob.start() , Sound.start()
startDrag	startDrag() , MovieClip.startDrag()
static	static
status	XML.status
stop	stop() , MovieClip.stop() , Sound.stop()
stopAllSounds	stopAllSounds()
stopDrag	stopDrag() , MovieClip.stopDrag()
字符串	String 类 , String()
StyleSheet (类)	TextField.StyleSheet 类
styleSheet (属性)	TextField.styleSheet
substr	String.substr()
substring	substring , String.substring()
super	super
swapDepths	MovieClip.swapDepths()
switch	switch
System	System 类
TAB	Key.TAB
tabChildren	MovieClip.tabChildren

动作脚本元素	参见条目
tabEnabled	Button.tabEnabled , MovieClip.tabEnabled , TextField.tabEnabled
tabIndex	Button.tabIndex , MovieClip.tabIndex , TextField.tabIndex
tabStops	TextFormat.tabStops
tan	Math.tan()
target	TextFormat.target
targetPath	targetPath
tellTarget	tellTarget
text	TextField.text
textColor	TextField.textColor
TextField	TextField 类
TextFormat	TextFormat 类
textHeight	TextField.textHeight
TextSnapshot	TextSnapshot 对象
textWidth	TextField.textWidth
this	this
throw	throw
time	NetStream.time
toggleHighQuality	toggleHighQuality()
toLowerCase	String.toLowerCase()
toString	Array.toString() , Boolean.toString() , Date.toString() , Error.toString() , LoadVars.toString() , Number.toString() , Object.toString() , XML.toString()
toUpperCase	String.toUpperCase()
trace	trace()
trackAsMenu	Button.trackAsMenu , MovieClip.trackAsMenu
true	true
try	try...catch...finally
type	TextField.type
typeof	typeof
undefined	undefined
underline	TextFormat.underline
unescape	unescape
uninstall	CustomActions.uninstall()
unloadClip	MovieClipLoader.unloadClip()

动作脚本元素	参见条目
unloadMovie	unloadMovie() , MovieClip.unloadMovie()
unLoadMovieNum	unloadMovieNum()
unshift	Array.unshift()
unwatch	Object.unwatch()
UP	Key.UP
updateAfterEvent	updateAfterEvent()
updateProperties	Accessibility.updateProperties()
url	TextFormat.url
useCodePage	System.useCodepage
useEchoSuppression	Microphone.useEchoSuppression()
useHandCursor	Button.useHandCursor , MovieClip.useHandCursor
UTC	Date.UTC()
valueOf	Boolean.valueOf() , Number.valueOf() , Object.valueOf()
var	var
variable	TextField.variable
version	System.capabilities.version
Video	Video 类
visible	ContextMenuItems.visible
void	void
watch	Object.watch()
while	while
width	Camera.width , Stage.width , Video.width
with	with
wordwrap	TextField.wordWrap
XML	XML 类
xmlDecl	XML.xmlDecl
XMLNode	XMLNode 类
XMLSocket	XMLSocket 类

--（递减）

可用性

Flash Player 4。

用法

行 *expression*
expression *行*

参数

无。

返回

一个数字。

说明

运算符（算术）；从 *expression* 中减 1 的预先递减和滞后递减一元运算符。此运算符的预先递减格式 (*--expression*) 从 *expression* 中减去 1，然后返回结果。此运算符的滞后递减格式 (*expression--*) 从 *expression* 中减去 1，然后返回 *expression* 的初始值（即减去 1 之前的值）。

示例

此运算符的预先递减格式将 *x* 递减为 2 (*x* - 1 = 2)，并将结果返回给 *y*：

```
x = 3;  
y = --x;  
//y 等于 2
```

此运算符的滞后递减格式将 *x* 递减为 2 (*x* - 1 = 2)，并将 *x* 的初始值作为结果返回给 *y*：

```
x = 3;  
y = x--  
//y 等于 3
```

++（递增）

可用性

Flash Player 4。

用法

```
++expression  
expression++
```

参数

无。

返回

一个数字。

说明

运算符（算术）；将 *expression* 加 1 的预先递增和滞后递增一元运算符。*expression* 可以是变量、数组中的元素或对象属性。此运算符的预先递增格式 (*++expression*) 将 *expression* 加 1，然后返回结果。此运算符的滞后递增格式 (*expression++*) 将 *expression* 加 1 并返回 *expression* 的初始值（即加上 1 之前的值）。

此运算符的预先递增格式将 *x* 递增为 2 ($x + 1 = 2$)，并将结果返回给 *y*：

```
x = 1;  
y = ++x  
//y 等于 2
```

此运算符的滞后递增格式将 *x* 递增为 2 ($x + 1 = 2$)，并将 *x* 的初始值作为结果返回给 *y*：

```
x = 1;  
y = x++;  
//y 等于 1
```

示例

下面的示例将 ++ 用作滞后递增运算符，以使 while 循环运行 5 次。

```
i = 0;  
while(i++ < 5){  
    trace("this is execution " + i);  
}
```

此示例将 ++ 用作预先递增运算符。

```
var a = [];  
var i = 0;  
while (i < 10) {  
    a.push(++i);  
}  
trace(a.join());
```

此脚本在“输出”面板中显示以下结果：

1,2,3,4,5,6,7,8,9,10

下面的示例将 ++ 用作滞后递增运算符。

```
var a = [];  
var i = 0;  
while (i < 10) {
```

```
a.push(i++);  
}  
trace(a.join());
```

此脚本在“输出”面板中显示以下结果：

0,1,2,3,4,5,6,7,8,9

！（逻辑 NOT）

可用性

Flash Player 4。

用法

`!expression`

参数

无。

返回

一个布尔值。

说明

运算符（逻辑）；对变量或表达式的布尔值取反。如果 *expression* 是绝对值或转换值为 `true` 的变量，则 `!expression` 的值为 `false`。如果表达式 `x && y` 的计算结果为 `false`，则表达式 `!(x && y)` 的计算结果为 `true`。

下面的表达式说明使用！运算符的结果：

`!true` 返回 `false`

`!false` 返回 `true`

示例

在下面的示例中，将变量 `happy` 设置为 `false`。if 条件对 `!happy` 条件进行计算，如果该条件为 `true`，则 `trace()` 动作向“输出”面板发送一个字符串。

```
happy = false;  
if (!happy) {  
    trace("don't worry, be happy");  
}
```

!= (不等于)

可用性

Flash Player 5。

用法

expression1 != *expression2*

参数

无。

返回

一个布尔值。

说明

运算符 (不等于) ；测试结果与 == 运算符正好相反。如果 *expression1* 与 *expression2* 相等，则结果为 false。与 == 运算符一样，*相等*的定义取决于所比较的数据类型。

- 数字、字符串和布尔值按值进行比较。
- 变量、对象、数组和函数按引用进行比较。

示例

下面的示例举例说明 != 运算符的结果：

5 != 8 返回 true

5 != 5 返回 false

此示例举例说明 != 运算符在 if 语句中的用法。

```
a = "David";  
b = "Fool"  
if (a != b){  
    trace("David is not a fool");  
}
```

另请参见

[!= \(不全等\)](#) , [== \(等于\)](#) , [=== \(全等\)](#)

!== (不全等)

可用性

Flash Player 6。

用法

expression1 !== *expression2*

说明

运算符；测试结果与 === 运算符正好相反。除了不转换数据类型外，不全等运算符执行的运算与不等于运算符相同。如果 *expression1* 等于 *expression2*，而且它们的数据类型也相同，则结果为 false。与 === 运算符一样，*相等*的定义取决于所比较的数据类型。

- 数字、字符串和布尔值按值进行比较。
- 变量、对象、数组和函数按引用进行比较。

示例

下面的代码显示使用等于、全等和不全等运算符的运算所返回的值。

```
s1 = new String("5");
s2 = new String("5");
s3 = new String("Hello");
n  = new Number(5);
b  = new Boolean(true);
```

```
s1 == s2; // true
s1 == s3; // false
s1 == n;  // true
s1 == b;  // false
```

```
s1 === s2; // true
s1 === s3; // false
s1 === n;  // false
s1 === b;  // false
```

```
s1 !== s2; // false
s1 !== s3; // true
s1 !== n;  // true
s1 !== b;  // true
```

另请参见

[!= \(不等于\)](#) , [== \(等于\)](#) , [=== \(全等\)](#)

%（模）

可用性

Flash Player 4。在 Flash 4 文件中，% 运算符在 SWF 文件中展开为 $x - \text{int}(x/y) * y$ ，而且可能不如在 Flash Player 的更高版本中那样快速和精确。

用法

expression1 % *expression2*

参数

无。

返回

无。

说明

运算符（算术）；计算 *expression1* 除以 *expression2* 的余数。如果两个 *expression* 参数中有一个是非数字值，则模运算符尝试将它们转换为数字。*expression* 可以是数字或转换为数值的字符串。

示例

下面是一个使用模 (%) 运算符的数字示例。

```
trace (12 % 5);  
// 返回 2  
trace (4.3 % 2.1);  
// 返回值约等于 0.1
```

%=（模赋值）

可用性

Flash Player 4。

用法

expression1 %= *expression2*

参数

无。

返回

无。

说明

运算符（算术复合赋值）；将 *expression1* % *expression2* 的值赋予 *expression1*。例如，下面两个表达式是相同的：

```
x %= y  
x = x % y
```


示例

下面的示例将值 4 赋予变量 x。

```
x = 14;  
y = 5;  
trace(x %= y);  
// 返回 4
```

另请参见

[% \(模\)](#)

&（按位 AND 运算符）

可用性

Flash Player 5。在 Flash 4 中，& 运算符用于连接字符串。在 Flash 5 和更高版本中，& 运算符是一个按位 AND 运算符，因此必须使用 add 和 + 运算符来连接字符串。使用 & 运算符的 Flash 4 文件在引入 Flash 5 或更高版本创作环境时自动更新为使用 add。

用法

expression1 & *expression2*

参数

无。

返回

无。

说明

运算符（按位）；将 *expression1* 和 *expression2* 转换为 32 位无符号整数，并对这些整数参数的每一位执行布尔 AND 运算。结果是一个新的 32 位无符号整数。

&&（逻辑 AND）

可用性

Flash Player 4。

用法

expression1 && *expression2*

参数

无。

返回

一个布尔值。

说明

运算符（逻辑）；对一个或两个表达式的值执行布尔运算。计算 *expression1*（该运算符左侧的表达式），当此表达式的计算结果为 `false` 时返回 `false`。如果 *expression1* 的计算结果为 `true`，则计算 *expression2*（该运算符右侧的表达式）。如果 *expression2* 的计算结果为 `true`，则最终结果为 `true`；否则，为 `false`。

示例

此示例用 `&&` 运算符执行一个测试，确定游戏者是否已经赢得游戏。在游戏过程中，当游戏者赢得一轮或者得到积分点时，就会对 `turns` 和 `score` 变量进行更新。在 3 轮之内游戏者的得分达到或超过 75 时，下面的脚本就会在“输出”面板中显示“You Win the Game!”。

```
turns=2;
score=77;
winner = (turns <= 3) && (score >= 75);
if (winner) {
    trace("You Win the Game!");
} else {
    trace("Try Again!");
}
```

&=（按位 AND 赋值）

可用性

Flash Player 5。

用法

expression1 &= *expression2*

参数

无。

返回

无。

说明

运算符；将 *expression1* & *expression2* 的值赋予 *expression1*。例如，下面两个表达式是相同的。

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

示例

下面的示例将值 9 赋予 `x`。

```
x = 15;
y = 9;
trace(x &= y);
// 返回 9
```

另请参见

[&（按位 AND 运算符）](#)

() (括号)

可用性

Flash Player 4。

用法

```
(expression1, expression2)  
function(parameter1, ..., parameterN)
```

参数

expression1、*expression2* 数字、字符串、变量或文本。
function 要对括号中的内容执行的函数。
parameter1...*parameterN* 一系列参数，在将其结果作为参数传递给括号外的函数之前执行这些参数。

返回

无。

说明

运算符；对一个或多个参数执行分组运算，或者括住一个或多个参数并将它们作为参数传递给括号外的函数。

用法 1：控制表达式中运算符的执行顺序。括号覆盖正常的优先级顺序，从而导致先计算括号内的表达式。如果括号是嵌套的，则先计算最里面括号中的内容，然后计算较靠外括号中的内容。

用法 2：括住一个或多个参数并将它们作为参数传递给括号外的函数。

示例

用法 1：下面的语句举例说明用括号控制表达式执行顺序的方法。每个表达式的值显示在每行的下面，如下所示：

```
trace((2 + 3) * (4 + 5));  
// 显示 45  
  
trace(2 + (3 * (4 + 5)));  
// 显示 29  
  
trace(2 + (3 * 4) + 5);  
// 显示 19
```

用法 2：下面的示例举例说明将括号与函数结合使用的方法。

```
getDate();  
  
invoice(item, amount);  
  
function traceParameter(param){  
    trace(param);  
}  
traceParameter(2*2);
```

另请参见

[with](#)

-（减号）

可用性

Flash Player 4。

用法

（符号反转） $-expression$

（减法） $expression1 - expression2$

参数

无。

返回

无。

说明

运算符（算术）；用于符号反转或减法运算。

用法 1：用于符号反转时，它将数值 *expression* 的符号反转。

用法 2：用于减法时，它对两个数值表达式执行算术减法运算，从 *expression1* 中减去 *expression2*。两个表达式都为整数时，差为整数。其中任何一个或两个表达式为浮点数时，差为浮点数。

示例

用法 1：下面的语句将表达式 $2 + 3$ 的符号反转。

$-(2 + 3)$

结果为 -5。

用法 2：下面的语句从整数 5 中减去整数 2。

$5 - 2$

结果为 3，是一个整数。

用法 2：下面的语句从浮点数 3.25 中减去浮点数 1.5。

$3.25 - 1.5$

结果为 1.75，是一个浮点数。

*（乘号）

可用性

Flash Player 4。

用法

expression1 * *expression2*

参数

无。

返回

无。

说明

运算符（算术）；将两个数值表达式相乘。如果两个表达式都是整数，则积为整数。如果其中任何一个或两个表达式是浮点数，则积为浮点数。

示例

用法 1：下面的语句将整数 2 与 3 相乘。

```
2 * 3
```

结果为 6，是一个整数。

用法 2：下面的语句将浮点数 2.0 与 3.1416 相乘。

```
2.0 * 3.1416
```

结果为 6.2832，是一个浮点数。

*=（相乘赋值）

可用性

Flash Player 4。

用法

expression1 *= *expression2*

参数

无。

返回

无。

说明

运算符（算术组合赋值）；将 *expression1* * *expression2* 的值赋予 *expression1*。例如，下面两个表达式是相同的：

```
x *= y  
x = x * y
```

示例

用法 1：下面的示例将值 50 赋予变量 x。

```
x = 5;  
y = 10;  
trace(x *= y);  
// 返回 50
```

用法 2：以下示例的第二行和第三行计算等号右侧的表达式，然后将结果赋予 x 和 y。

```
i = 5;  
x = 4 - 6;  
y = i + 2;  
trace(x *= y);  
// 返回 -14
```

另请参见

[* \(乘号\)](#)

, (逗号)

可用性

Flash Player 4。

用法

expression1, expression2

参数

无。

返回

无。

说明

运算符；先计算 *expression1*，再计算 *expression2*，然后返回 *expression2* 的值。此运算符主要与 for 循环语句一起使用。

示例

下面的代码范例使用了逗号运算符：

```
var a=1, b=2, c=3;
```

这相当于编写了如下代码：

```
var a=1;  
var b=2;  
var c=3;
```

. (点)

可用性

Flash Player 4。

用法

```
object.property_or_method  
instancename.variable  
instancename.childinstance.variable
```

参数

object 类的一个实例。此对象可以是任意内置动作脚本类或自定义类的实例。此参数总是在点 (.) 运算符的左侧。

property_or_method 与对象相关联的属性或方法的名称。内置类的所有有效方法和属性都在该类的方法和属性摘要表中列出。此参数总是在点 (.) 运算符的右侧。

instancename 影片剪辑的实例名称。

childinstance 从属于或嵌套于另一个影片剪辑的影片剪辑实例。

variable 影片剪辑实例时间轴上的变量，该影片剪辑实例的名称在点 (.) 运算符的左侧。

返回

无。

说明

运算符；用于定位影片剪辑的层次结构，以便访问嵌套的（子级）影片剪辑、变量或属性。点运算符也用于测试或设置对象的属性、执行对象的方法或创建数据结构。

示例

下面的语句标识影片剪辑 `person_mc` 中变量 `hairColor` 的当前值。

```
person_mc.hairColor
```

这与下面的 Flash 4 语法等效：

```
/person_mc:hairColor
```

: (类型)

可用性

Flash Player 6。

用法

```
[modifiers] [var] variableName:[type]  
function functionName():[type] { ... }  
function functionName(parameter1:[type], ..., parameterN:[type]) { ... }
```

参数

variableName 变量的标识符。

type 本机数据类型、您已定义的类名称或接口名称。

functionName 函数的标识符。

parameter 函数参数的标识符。

说明

运算符；指定变量类型、函数返回类型或函数参数类型。在变量声明或赋值中使用时，此运算符指定变量的类型；在函数声明或定义中使用时，此运算符指定函数的返回类型；在函数定义中与函数参数一起使用时，此运算符指定该参数应使用的变量类型。

类型是仅限编译时的功能。所有类型都在编译时检查，并在出现不匹配的情况时生成错误。（有关更多信息，请参见第 739 页的附录 A “错误消息”。）在使用点 (.) 运算符进行赋值运算、函数调用和类成员解除引用的过程中，可能会出现不匹配情况。若要避免类型不匹配错误，可使用显式类型指定（请参见第 33 页的“严格数据类型指定”）。

您可以使用的类型包括全部本机对象类型、您定义的类和接口以及 Void 和 Function（仅作为类型存在，而不能作为对象存在）。识别的本机类型有 Array、Boolean、Button、Color、CustomActions、Date、Function、LoadVars、LocalConnection、Microphone、MovieClip、NetConnection、NetStream、Number、Object、SharedObject、Sound、String、TextField、TextFormat、Video、Void、XML、XMLNode 和 XMLSocket。

示例

用法 1：下面的示例声明一个类型为 String 且名为 userName 的公共变量，并且给该变量分配一个空字符串。

```
public var userName:String = "";
```

用法 2：该示例举例说明如何指定函数的参数类型。下面的代码定义一个名为 setDate() 的函数，该函数使用类型为 Date 且名为 currentDate 的参数。

```
function setDate(currentDate:Date) {  
    this.date = currentDate;  
}
```

用法 3：下面的代码定义一个名为 squareRoot() 的函数，它使用类型为 Number 且名为 val 的参数，并且返回 val 的平方根（也为 Number 类型）。

```
function squareRoot(val:Number):Number {  
    return Math.sqrt(val);  
}
```


?:(条件)

可用性

Flash Player 4。

用法

expression1 ? *expression2* : *expression3*

参数

expression1 计算结果为布尔值的表达式，通常为像 *x* < 5 这样的比较表达式。

expression2、*expression3* 任何类型的值。

返回

无。

说明

运算符；指示 Flash 计算 *expression1*，如果 *expression1* 的值为 true，则它返回 *expression2* 的值；否则，它返回 *expression3* 的值。

示例

下面的语句因为 *expression1* 的计算结果为 true，所以将变量 *x* 的值赋予变量 *z*：

```
x = 5;  
y = 10;  
z = (x < 6) ? x:y;  
trace (z);  
// 返回 5
```

/ (除号)

可用性

Flash Player 4。

用法

expression1 / *expression2*

参数

expression 数字或计算结果为数字的变量。

返回

无。

说明

运算符（算术）；将 *expression1* 除以 *expression2*。除法运算的结果为双精度浮点数。

示例

下面的语句将浮点数 22.0 除以 7.0，然后在“输出”面板中显示结果。

```
trace(22.0 / 7.0);
```

结果为 3.1429，是一个浮点数。

//（注释分隔符）

可用性

Flash 1。

用法

// 注释

参数

comment 任何字符。

返回

无。

说明

注释；指示脚本注释的开始。任何出现在注释分隔符 *//* 和行结束符之间的字符都被动作脚本解释程序解释为注释并忽略。

示例

此脚本使用注释分隔符将第一、第三、第五和第七行标识为注释。

```
// 记录 ball 影片剪辑的 X 位置
ballX = ball._x;
// 记录 ball 影片剪辑的 Y 位置
ballY = ball._y;
// 记录 bat 影片剪辑的 X 位置
batX = bat._x;
// 记录 bat 影片剪辑的 Y 位置
batY = bat._y;
```

另请参见

[/*（注释分隔符）](#)

/*（注释分隔符）

可用性

Flash Player 5。

用法

```
/* comment */

/*
comment
comment
*/
```

参数

comment 任何字符。

返回

无。

说明

注释；指示一行或多行脚本注释。任何出现在注释开始标签 `/*` 和注释结束标签 `*/` 之间的字符都被动作脚本解释程序解释为注释并忽略。第一种语法类型用于标识单行注释。第二种语法类型用于标识连续多行注释。在使用注释分隔符的这种格式时，如果不使用结束标签 `*/`，就会返回一个错误信息。

示例

此脚本在脚本的开头使用注释分隔符。

```
/* 记录 ball 和 bat 影片剪辑的  
   X 和 Y 位置  
*/  
  
ballX = ball._x;  
ballY = ball._y;  
batX = bat._x;  
batY = bat._y;
```

另请参见

[//](#)（注释分隔符）

`/=`（除法赋值）

可用性

Flash Player 4。

用法

expression1 `/=` *expression2*

参数

expression1、*expression2* 数字或计算结果为数字的变量。

返回

无。

说明

运算符（算术组合赋值）；将 *expression1* / *expression2* 的值赋予 *expression1*。例如，下面两个语句是相同的：

```
x /= y  
x = x / y
```

示例

下面的代码举例说明如何将 `/=` 运算符与变量和数字结合使用。

```
x = 10;  
y = 2;  
x /= y;  
// x 现在包含值 5
```

[]（数组访问）

可用性

Flash Player 4。

用法

```
my_array = ["a0", a1,...aN]
myMultiDimensional_array = [[ "a0",...aN],...[ "a0",...aN]]
my_array[E] = value
myMultiDimensional_array[E][E] = value
object["value"]
```

参数

my_array 数组的名称。

a0, a1,...aN 数组中的元素。

myMultiDimensional_array 模拟多维数组的名称。

E 数组中元素的编号（或索引）。

object 对象的名称。

value 字符串或计算结果为字符串的表达式，用来命名对象的属性。

返回

无。

说明

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

用法 1：数组是一种对象，其属性称作“元素”，这些元素由称作“索引”的数字逐一标识。创建数组时，需用数组访问运算符（或“括号”）括住元素。一个数组可以包含各种类型的元素。例如，下面这个名为 `employee` 的数组包含三个元素；第一个元素是数字，另外两个元素是字符串（在引号内）。

```
employee = [15, "Barbara", "Erick"];
```

用法 2：可以通过嵌套括号来模拟多维数组。下面的代码创建一个名为 `ticTacToe` 且含有三个元素的数组；而每个元素也是一个具有三个元素的数组。

```
ticTacToe = [[1,2,3],[4,5,6],[7,8,9]];
```

```
// 在测试影片模式下，选择“调试” > “变量列表”
// 来查看数组元素的列表
```

用法 3：用括号括住每个元素的索引可直接对其进行访问；这样可以向数组添加新元素以及更改或获取现有元素的值。数组中第一个元素的索引永远是 0：

```
my_array[0] = 15;
my_array[1] = "Hello";
my_array[2] = true;
```

可以使用括号来添加第四个元素，如下所示：

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

用法 4：括号可用于访问多维数组中的元素。第一对括号标识原始数组中的元素，第二对括号标识嵌套数组中的元素。下面的几行代码将数字 6 发送到“输出”面板。

```
ticTacToe = [[1,2,3],[4,5,6],[7,8,9]];
trace(ticTacToe[1][2]);
```

```
// 返回 6
```

用法 5：您可以用数组访问运算符代替 eval 函数，从而动态地设置并获取影片剪辑名称的值或一个对象的任何属性：

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

示例

用法 1：下面的代码范例说明新建空 Array 对象的两种不同方式；第一行使用括号。

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

用法 1 和 2：下面的示例创建一个名为 employee_array 的数组，并使用 trace() 动作将这些元素发送到“输出”面板。在第四行中，更改数组中的一个元素，而第五行将刚修改过的数组发送到“输出”面板：

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

用法 3：在下面的示例中，计算括号中的表达式 ("piece" + i)，并将结果用作要从影片剪辑 my_mc 中获取的变量名。在此示例中，变量 i 与所释放的按钮必须在同一个时间轴上。例如，如果变量 i 等于 5，影片剪辑 my_mc 中的变量 piece5 的值将显示在“输出”面板中：

```
on (release) {
    x = my_mc["piece"+i];
    trace(x);
}
```

用法 3：在下面的代码中，计算括号中的表达式，并将结果用作要从影片剪辑 name_mc 中获取的变量名：

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

如果您熟悉 Flash 4 动作脚本的斜杠语法，可以使用 eval 函数来达到同样的结果：

```
eval("name.A" & i);
```

用法 3：您还可以在赋值语句的左侧使用数组访问运算符，从而动态地设置实例、变量和对象的名称：

```
name[index] = "Gary";
```

另请参见

[Array 类](#), [Object 类](#), [eval\(\)](#)

^（按位 XOR）

可用性

Flash Player 5。

用法

expression1 ^ *expression2*

参数

expression1、*expression2* 数字。

返回

无。

说明

运算符（按位）；将 *expression1* 和 *expression2* 转换为 32 位无符号整数，然后对于 *expression1* 或者 *expression2* 中相应位为 1 且不同时为 1 的每一位，返回 1。

示例

下面的示例对十进制的 15 和 9 使用按位 XOR 运算符，然后将结果赋予变量 x。

```
// 十进制的 15 = 二进制的 1111
// 十进制的 9 = 二进制的 1001
x = 15 ^ 9
trace (x)
// 1111 ^ 1001 = 0110
// 返回十进制的 6 (= 二进制的 0110)
```

^=（按位 XOR 赋值）

可用性

Flash Player 5。

用法

expression1 ^= *expression2*

参数

expression1、*expression2* 整数和变量。

返回

无。

说明

运算符（按位组合赋值）；将 *expression1* ^ *expression2* 的值赋予 *expression1*。例如，下面两个语句是相同的：

```
x ^= y
x = x ^ y
```

示例

下面是一个 ^= 运算的示例。

```
// 十进制的 15 = 二进制的 1111
x = 15;
// 十进制的 9 = 二进制的 1001
y = 9;
trace(x ^= y);
// 返回十进制的 6 (= 二进制的 0110)
```

另请参见

[^ \(按位 XOR\)](#)

{ } (对象初始值设定项)

可用性

Flash Player 5。

用法

```
object = {name1:value1, name2:value2,...nameN:valueN}
```

参数

object 要创建的对象。

name1,2,...N 属性名。

value1,2,...N 每个 *name* 属性对应的值。

返回

无。

说明

运算符；创建一个新对象并用指定的 *name* 和 *value* 属性对其进行初始化。使用此运算符的效果与使用 `new Object` 语法然后使用赋值运算符填充属性对的效果相同。通常，将新建对象的原型命名为 `Object` 对象。

示例

下面代码的第一行用对象初始值设定项运算符创建一个空对象；第二行用构造函数创建一个新对象。

```
object = {};  
object = new Object();
```

下面的示例创建对象 `account`，然后用附带的值初始化属性 `name`、`address`、`city`、`state`、`zip` 和 `balance`。

```
account = { name:"Betty Skate",  
            address:"123 Main Street",  
            city:"Blossomville",  
            state:"California",  
            zip: "12345",  
            balance: "1000" };
```

下面的示例说明数组和对象初始值设定项可以如何相互嵌套。

```
person = { name:"Gina Vechio",
  children:[ "Ruby", "Chickie", "Puppa" ] };
```

下面的示例使用上述示例中的信息，并使用构造函数得到相同的结果。

```
person = new Object();
person.name = 'Gina Vechio';
person.children = new Array();
person.children[0] = 'Ruby';
person.children[1] = 'Chickie';
person.children[2] = 'Puppa';
```

另请参见

[\[\]](#)（数组访问），[new](#)，[Object](#) 类

|（按位 OR）

可用性

Flash Player 5。

用法

expression1 | *expression2*

参数

expression1、*expression2* 数字。

返回

无。

说明

运算符（按位）；将 *expression1* 和 *expression2* 转换为 32 位无符号整数，然后对于 *expression1* 或 *expression2* 的相应位为 1 的每一位返回 1。

示例

下面是一个按位 OR 运算的示例。

```
// 十进制的 15 = 二进制的 1111
x = 15;
// 十进制的 9 = 二进制的 1001
y = 9;
trace(x | y);
// 1111 | 0011 = 1111
// 返回十进制的 15 (= 二进制的 1111)
```


||（逻辑 OR）

可用性

Flash Player 4。

用法

expression1 || *expression2*

参数

expression1、*expression2* 布尔值或可转换为布尔值的表达式。

返回

一个布尔值。

说明

运算符（逻辑）；计算 *expression1* 和 *expression2*。如果其中任何一个或者两个表达式的计算结果为 `true`，则结果为 `true`；只有当两个表达式的计算结果都为 `false` 时，结果才为 `false`。逻辑 OR 运算符可用于任意多个操作数；只要任意一个操作数的计算结果为 `true`，则结果为 `true`。

对于非布尔表达式，逻辑 OR 运算符使得 Flash 对左侧的表达式进行计算；如果左侧的表达式可以转换为 `true`，则结果为 `true`。否则，计算右侧的表达式，而且结果就是该表达式的值。

示例

用法 1：下面的示例在 `if` 语句中使用 `||` 运算符。第二个表达式的计算结果为 `true`，因此最终结果为 `true`：

```
x = 10
y = 250
start = false
if(x > 25 || y > 200 || start){
    trace('the logical OR test passed');
}
```

用法 2：此示例演示非布尔表达式是如何导致意外结果的。如果左侧的表达式转换为 `true`，则返回该结果，而不再转换右侧的表达式。

```
function fx1(){
    trace ("fx1 called");
    returns true;
}
function fx2(){
    trace ("fx2 called");
    return true;
}
if (fx1() || fx2()){
    trace ("IF statement entered");
}
// 将以下内容发送到“输出”面板：
// fx1 called
// IF statement entered
```

|= (按位 OR 赋值)

可用性

Flash Player 5。

用法

expression1 |= *expression2*

参数

expression1、*expression2* 数字或变量。

返回

无。

说明

运算符（按位组合赋值）；将 *expression1* | *expression2* 的值赋予 *expression1*。例如，下面两个语句是相同的：

```
x |= y;  
x = x | y;
```

示例

下面的示例使用了 |= 运算符：

```
// 十进制的 15 = 二进制的 1111  
x = 15;  
// 十进制的 9 = 二进制的 1001  
y = 9;  
trace(x |= y);  
// 1111 |= 1001  
// 返回十进制的 15 (= 二进制的 1111)
```

另请参见

| (按位 OR)

~ (按位 NOT)

可用性

Flash Player 5。

用法

~ *expression*

参数

expression 数字。

返回

无。

说明

运算符（按位）；将 *expression* 转换为 32 位无符号整数，然后对这些位进行求反。按位 NOT 运算更改数字的符号然后减 1。

示例

下面的示例说明对一个变量执行按位 NOT 运算。

```
a = 0;
trace ("when a = 0, ~a = "+~a);
// 当 a = 0, ~a = -1
a = 1;
trace ("when a = 1, ~a = "+~a);
// 当 a = 0, ~a = -2
// 因此, ~0=-1, ~1=-2
```

+（加号）

可用性

Flash Player 4 ； Flash Player 5。在 Flash 5 和更高版本中，+ 既可以是数值运算符，也可以是字符串连接符，这要取决于参数的数据类型。在 Flash 4 中，+ 只是数值运算符。引入到 Flash 5 或更高版本创作环境中的 Flash 4 文件经历了一个转换过程，以保持数据类型的完整性。下面的示例举例说明对包含数值比较的 Flash 4 文件的转换过程：

Flash 4 文件：

```
x + y
```

转换的 Flash 5 或更高版本的文件：

```
Number(x) + Number(y)
```

用法

```
expression1 + expression2
```

参数

expression1、*expression2* 数字或字符串。

返回

无。

说明

运算符；将数值表达式相加或者连接（合并）字符串。如果其中一个表达式为字符串，则所有其它表达式都被转换为字符串，然后连接起来。

两个表达式都为整数时，和为整数；其中一个或两个表达式为浮点数时，和为浮点数。

示例

用法 1：下面的示例连接两个字符串，然后在“输出”面板中显示结果。

```
name = "Cola";
instrument = "Drums";
trace (name + " plays " + instrument);
```

用法 2：与动态和输入文本字段相关联的变量是字符串数据类型。在下面的示例中，变量 deposit 是舞台上的一个输入文本字段。在用户输入存款数目后，该脚本尝试将 deposit 加到 oldBalance 上。然而，由于 deposit 是字符串数据类型，因此脚本连接（合并成一个字符串）变量的值，而不是对它们求和。

```
oldBalance = 1345.23;
```

```
currentBalance = deposit + oldBalance;  
trace (currentBalance);
```

例如，如果用户在 `deposit` 文本字段中输入 475，则 `trace()` 动作将值 4751345.23 发送到“输出”面板。

若要更正这一点，可使用 `Number()` 函数将字符串转换为数字，如下所示：

```
currentBalance = Number(deposit) + oldBalance;
```

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

```
trace (2 +3);
```

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

```
trace (2.5 + 3.25);
```

另请参见

[_accProps](#)

+=（加法赋值）

可用性

Flash Player 4。

用法

```
expression1 += expression2
```

参数

expression1、*expression2* 数字或字符串。

返回

无。

说明

运算符（算术组合赋值）：将 *expression1* + *expression2* 的值赋予 *expression1*。例如，下面两个语句的结果是相同的：

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

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

示例

下面的示例显示 += 运算符的数值用法。

```
x = 5;  
y = 10;  
x += y;  
trace(x);  
//x 返回 15
```

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

```
x = "My name is "  
x += "Gilbert"  
trace(x)  
// 返回 “My name is Gilbert”
```

另请参见

[+ \(加号\)](#)

< (小于)

可用性

Flash Player 4 ； Flash Player 5。在 Flash 5 和更高版本中，< (小于) 运算符是能够处理各种数据类型的比较运算符。而在 Flash 4 中，< 是数值运算符。引入到 Flash 5 或更高版本创作环境中的 Flash 4 文件经历了一个转换过程，以保持数据类型的完整性。下面举例说明对包含数值比较的 Flash 4 文件的转换过程。

Flash 4 文件：

```
x < y
```

转换的 Flash 5 或更高版本的文件：

```
Number(x) < Number(y)
```

用法

```
expression1 < expression2
```

参数

expression1、*expression2* 数字或字符串。

说明

运算符 (比较) ； 比较两个表达式以确定 *expression1* 是否小于 *expression2* ；如果是，则该运算符返回 true。如果 *expression1* 大于或等于 *expression2*，则该运算符返回 false。使用字母顺序计算字符串表达式 ；所有的大写字母排在小写字母的前面。

示例

下面的示例举例说明数值比较和字符串比较所返回的 true 和 false。

```
3 < 10;  
// true  
  
10 < 3;  
//false  
  
"Allen" < "Jack";  
// true  
  
"Jack" < "Allen";  
//false  
  
"11" < "3";  
// true
```

```
"11" < 3;  
// 数值比较  
//false  
  
"C" < "abc";  
//false  
  
"A" < "a";  
// true
```

<<（按位向左移位）

可用性

Flash Player 5。

用法

expression1 << *expression2*

参数

expression1 要向右移位的数字或表达式。

expression2 将转换为 0 到 31 之间整数的数字或表达式。

返回

无。

说明

运算符（按位）；将 *expression1* 和 *expression2* 转换为 32 位整数，然后将 *expression1* 中的所有位向左移一定的位数，该位数由 *expression2* 转换而来的整数指定。由于此运算的结果而被清空的位上则填补 0。将一个值向左移一位等效于将它乘以 2。

示例

在下面的示例中，将整数 1 向左移 10 位。

```
x = 1 << 10
```

此运算的结果为 $x = 1024$ 。这是因为十进制的 1 等于二进制的 1，二进制的 1 向左移 10 位是二进制的 1000000000，而二进制的 1000000000 就是十进制的 1024。

在下面的示例中，将整数 7 向左移 8 位。

```
x = 7 << 8
```

此运算的结果为 $x = 1792$ 。这是因为十进制的 7 等于二进制的 111，二进制的 111 向左移 8 位是二进制的 1110000000，而二进制的 1110000000 就是十进制的 1792。

另请参见

>>（按位向右移位并赋值），>>（按位向右移位），<<=（按位向左移位并赋值）

<<= （按位向左移位并赋值）

可用性

Flash Player 5。

用法

expression1 <<= *expression2*

参数

expression1 要向右移位的数字或表达式。

expression2 将转换为 0 到 31 之间整数的数字或表达式。

返回

无。

说明

运算符（按位组合赋值）；此运算符执行按位向左移位运算，然后将该内容作为结果存储在 *expression1* 中。下面的两个表达式是等效的。

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

另请参见

[<<（按位向左移位）](#)，[>>=（按位向右移位并赋值）](#)，[>>（按位向右移位）](#)

<= （小于等于）

可用性

Flash Player 4。

Flash 4 文件：

```
x <= y
```

转换的 Flash 5 或更高版本的文件：

```
Number(x) <= Number(y)
```

用法

expression1 <= *expression2*

参数

expression1、*expression2* 数字或字符串。

返回

一个布尔值。

说明

运算符（比较）；比较两个表达式以确定 *expression1* 是否小于等于 *expression2*；如果是，则该运算符返回 `true`。如果 *expression1* 大于 *expression2*，则该运算符返回 `false`。使用字母顺序计算字符串表达式；所有的大写字母排在小写字母的前面。

在 Flash 5 或更高版本中，小于等于 (<=) 运算符是能够处理各种数据类型的比较运算符。而在 Flash 4 中，<= 是一个数值运算符。引入到 Flash 5 或更高版本创作环境中的 Flash 4 文件经历了一个转换过程，以保持数据类型的完整性。下面举例说明对包含数值比较的 Flash 4 文件的转换过程。

示例

下面的示例举例说明用于数值和字符串比较的 true 和 false 结果：

```
5 <= 10;  
// true  
  
2 <= 2;  
// true  
  
10 <= 3;  
//false  
  
"Allen" <= "Jack";  
// true  
  
"Jack" <= "Allen";  
//false  
  
"11" <= "3";  
// true  
  
"11" <= 3;  
// 数值比较  
//false  
  
"C" <= "abc";  
//false  
  
"A" <= "a";  
// true
```

<> (不等于)

可用性

Flash 2。

用法

expression1 <> *expression2*

参数

expression1、*expression2* 数字、字符串、布尔值、变量、对象、数组或函数。

返回

一个布尔值。

说明

运算符 (不等于)；测试结果与 == 运算符正好相反。如果 *expression1* 与 *expression2* 相等，则结果为 false。与 == 运算符一样，*相等* 的定义取决于所比较的数据类型：

- 数字、字符串和布尔值按值进行比较。

- 变量、对象、数组和函数按引用进行比较。

在 Flash 5 中已不鼓励使用此运算符，Macromedia 建议您使用 `!=` 运算符。

另请参见

[!= \(不等于\)](#)

= (赋值)

可用性

Flash Player 4。

Flash 4 文件：

```
x = y
```

转换的 Flash 5 或更高版本的文件：

```
Number(x) == Number(y)
```

用法

```
expression1 = expression2
```

参数

expression1 变量、数组元素或对象属性。

expression2 任何类型的值。

返回

无。

说明

运算符；将 *expression2*（位于右侧的参数）的类型赋予 *expression1* 中的变量、数组元素或属性。

在 Flash 5 或更高版本中，`=` 为赋值运算符，而 `==` 运算符用于计算相等性。在 Flash 4 中，`=` 为数值等于运算符。引入到 Flash 5 或更高版本创作环境中的 Flash 4 文件经历了一个转换过程，以保持数据类型的完整性。

示例

下面的示例使用赋值运算符将数字数据类型赋予变量 `x`。

```
x = 5
```

下面的示例使用赋值运算符将字符串数据类型赋予变量 `x`。

```
x = "hello"
```

另请参见

[== \(等于\)](#)

-=（减法赋值）

可用性

Flash Player 4。

用法

expression1 -= expression2

参数

expression1、*expression2* 数字或计算结果为数字的表达式。

返回

无。

说明

运算符（算术组合赋值）；将 *expression1 - expression2* 的值赋予 *expression1*。例如，下面两个语句是相同的：

```
x -= y;  
x = x - y;
```

必须将字符串表达式转换为数字；否则返回 NaN。

示例

用法 1：下面的示例使用 -= 运算符从 5 中减去 10，然后将结果赋予变量 x。

```
x = 5;  
y = 10;  
x -= y  
trace(x);  
// 返回 -5
```

用法 2：下面的示例说明如何将字符串转换为数字。

```
x = "5";  
y = "10";  
x -= y;  
trace(x);  
// 返回 -5
```

==（等于）

可用性

Flash Player 5。

用法

expression1 == expression2

参数

expression1、*expression2* 数字、字符串、布尔值、变量、对象、数组或函数。

返回

一个布尔值。

说明

运算符（等于）；测试两个表达式是否相等。如果表达式相等，则结果为 `true`。

*相等*的定义取决于参数的数据类型：

- 数字和布尔值按值进行比较，如果它们具有相同的值，则视为相等。
- 对于字符串表达式，如果它们具有相同的字符数，而且这些字符都相同，则这些字符串表达式相等。
- 变量、对象、数组和函数按引用进行比较。对于变量，如果两个变量引用相同的对象、数组或函数，则它们相等。而两个单独的数组即使具有相同数量的元素，也永远不会被视为相等。

示例

用法 1：下面的示例将 `==` 运算符与 `if` 语句结合使用：

```
a = "David" , b = "David";
if (a == b){
    trace("David is David");
}
```

用法 2：这些示例显示比较混合类型的运算的结果。

```
x = "5"; y = "5";
trace (x == y);
// true
```

```
x = "5"; y = "66";
trace (x == y);
//false
```

```
x = "chris"; y = "steve";
trace (x == y);
//false
```

另请参见

`!=`（不等于），`===`（全等），`!==`（不全等）

===（全等）

可用性

Flash Player 6。

用法

```
expression1 === expression2
```

返回

一个布尔值。

说明

运算符；测试两个表达式是否相等；除了不转换数据类型外，全等运算符执行的运算与等于运算符相同。如果两个表达式完全相等（包括它们的数据类型都相等），则结果为 `true`。

*相等*的定义取决于参数的数据类型：

- 数字和布尔值按值进行比较，如果它们具有相同的值，则视为相等。

- 对于字符串表达式，如果它们具有相同的字符数，而且这些字符都相同，则这些字符串表达式相等。
- 变量、对象、数组和函数按引用进行比较。对于变量，如果两个变量引用相同的对象、数组或函数，则它们相等。而两个单独的数组即使具有相同数量的元素，也永远不会被视为相等。

示例

下面的代码显示使用等于、全等和不全等运算符的运算所返回的值。

```
s1 = new String("5");
s2 = new String("5");
s3 = new String("Hello");
n  = new Number(5);
b  = new Boolean(true);
```

```
s1 == s2; // true
s1 == s3; // false
s1 == n;  // true
s1 == b;  // false
```

```
s1 === s2; // true
s1 === s3; // false
s1 === n;  // false
s1 === b;  // false
```

```
s1 !== s2; // false
s1 !== s3; // true
s1 !== n;  // true
s1 !== b;  // true
```

另请参见

[==](#)（等于），[!=](#)（不等于），[===](#)（全等）

>（大于）

可用性

Flash Player 4。

Flash 4 文件：

`x > y`

转换的 Flash 5 或更高版本的文件：

`Number(x) > Number(y)`

用法

expression1 > *expression2*

参数

expression1、*expression2* 数字或字符串。

返回

一个布尔值。

说明

运算符（比较）；比较两个表达式以确定 *expression1* 是否大于 *expression2*；如果是，则该运算符返回 `true`。如果 *expression1* 小于或等于 *expression2*，则该运算符返回 `false`。使用字母顺序计算字符串表达式；所有的大写字母排在小写字母的前面。

在 Flash 5 或更高版本中，大于 (>) 是能够处理各种数据类型的比较运算符。而在 Flash 4 中，> 是数值运算符。引入到 Flash 5 或更高版本创作环境中的 Flash 4 文件经历了一个转换过程，以保持数据类型的完整性。

>= （大于等于）

可用性

Flash Player 4。

Flash 4 文件：

```
x > y
```

转换的 Flash 5 或更高版本的文件：

```
Number(x) > Number(y)
```

用法

```
expression1 >= expression2
```

参数

expression1、*expression2* 字符串、整数或浮点数。

返回

一个布尔值。

说明

运算符（比较）；比较两个表达式以确定 *expression1* 是大于等于 *expression2* (`true`)，还是 *expression1* 小于 *expression2* (`false`)。

在 Flash 5 或更高版本中，大于等于 (>=) 运算符是能够处理各种数据类型的比较运算符。而在 Flash 4 中，>= 是一个数值运算符。引入到 Flash 5 或更高版本创作环境中的 Flash 4 文件经历了一个转换过程，以保持数据类型的完整性。

>> （按位向右移位）

可用性

Flash Player 5。

用法

```
expression1 >> expression2
```

参数

expression1 要向右移位的数字或表达式。

expression2 将转换为 0 到 31 之间整数的数字或表达式。

返回

无。

说明

运算符（按位）；将 *expression1* 和 *expression2* 转换为 32 位整数，然后将 *expression1* 中的所有位向右移一定位数，该位数由 *expression2* 转换而来的整数指定。移到右侧的位被去掉。若要保留原 *expression* 的符号，则如果 *expression1* 的最高有效位（最左端的位）为 0，那么左侧的位都填补 0；如果最高有效位为 1，那么左侧的位都填补 1。将一个值右移一位等效于将它除以 2 并去掉余数。

示例

下面的示例将 65535 转换为 32 位整数，然后右移 8 位。

```
x = 65535 >> 8
```

上述运算的结果如下所示：

```
x = 255
```

这是因为十进制的 65535 等于二进制的 1111111111111111（16 个 1），二进制的 1111111111111111 向右移 8 位是二进制的 11111111，而二进制的 11111111 是十进制的 255。因为这个整数是 32 位的，最高有效位为 0，所以用 0 来填补位。

下面的示例将 -1 转换为 32 位整数，然后右移 1 位。

```
x = -1 >> 1
```

上述运算的结果如下所示：

```
x = -1
```

这是因为十进制的 -1 等于二进制的 11111111111111111111111111111111（32 个 1），向右移一位使得最低有效位（最右侧的位）被去掉而最高有效位被填补上 1。结果为二进制的 11111111111111111111111111111111（32 个 1），表示 32 位整数 -1。

另请参见

[>>=（按位向右移位并赋值）](#)

>>=（按位向右移位并赋值）

可用性

Flash Player 5。

用法

```
expression1 >>=expression2
```

参数

expression1 要向右移位的数字或表达式。

expression2 将转换为 0 到 31 之间整数的数字或表达式。

返回

无。

说明

运算符（按位组合赋值）；此运算符执行按位向右移位运算，然后将该内容作为结果存储在 *expression1* 中。

示例

下面的两个表达式是等效的。

```
A >>= B
```

```
A = (A >> B)
```

下面加有注释的代码使用按位 (>>=) 运算符。这也是使用所有按位运算符的示例。

```
function convertToBinary(number){
    var result = "";
    for (var i=0; i<32; i++) {
        // 使用按位 AND 提取最低有效位
        var lsb = number & 1;
        // 将此位添加到结果字符串中
        result = (lsb ? "1" : "0") + result;
        // 将数字向右移一位以显示下一位
        number >>= 1;}
    return result;
}
trace(convertToBinary(479));
// 返回字符串 0000000000000000000000000111011111
// 上述字符串是十进制
// 数字 479 的二进制表示形式
```

另请参见

[<<（按位向左移位）](#)

>>>（按位无符号向右移位）

可用性

Flash Player 5。

用法

```
expression1 >>> expression2
```

参数

expression1 要向右移位的数字或表达式。

expression2 将转换为 0 到 31 之间整数的数字或表达式。

返回

无。

说明

运算符（按位）；除了由于左侧的位总是填补 0 而不保留原 *expression* 的符号外，该运算符与按位向右移位 (>>) 运算符相同。

示例

下面的示例将 -1 转换为 32 位整数，然后右移 1 位。

```
x = -1 >>> 1
```

上述运算的结果如下所示：

```
x = 2147483647
```

这是因为十进制的 -1 是二进制的 11111111111111111111111111111111 (32 个 1)，当向右 (无符号) 移 1 位时，最低有效位 (最右端的位) 被去掉，而最高有效位 (最左端的位) 被填补上 0。结果为二进制的 01111111111111111111111111111111，表示 32 位整数 2147483647。

另请参见

[>>= \(按位向右移位并赋值\)](#)

>>>= (按位无符号向右移位并赋值)

可用性

Flash Player 5。

用法

```
expression1 >>>= expression2
```

参数

expression1 要向右移位的数字或表达式。

expression2 将转换为 0 到 31 之间整数的数字或表达式。

返回

无。

说明

运算符 (按位组合赋值)；执行无符号按位向右移位运算，然后将该内容作为结果存储在 *expression1* 中。下面的两个表达式是等效的：

```
A >>>= B  
A = (A >>> B)
```

另请参见

[>>> \(按位无符号向右移位\)](#)，[>>= \(按位向右移位并赋值\)](#)

Accessibility 类

可用性

Flash Player 6 版本 65。

说明

Accessibility 类管理与屏幕读取器之间的通讯。Accessibility 类的方法是静态的，即，您不必创建该类的实例即可使用其方法。

若要获取和设置特定对象（例如按钮、影片剪辑或文本字段）的可访问属性，请使用 `_accProps` 属性。若要确定播放器是否运行在支持辅助功能的环境中，请使用 `System.capabilities.hasAccessibility`。

Accessibility 类的方法摘要

方法	说明
<code>Accessibility.isActive()</code>	指示屏幕读取器程序是否处于活动状态。
<code>Accessibility.updateProperties()</code>	为屏幕读取器更新屏幕上对象的说明。

Accessibility.isActive()

可用性

Flash Player 6 版本 65。

用法

`Accessibility.isActive()`

参数

无。

返回

如果有活动的 Microsoft Active Accessibility (MSAA) 客户端，并且播放器正在支持在 Flash Player 和辅助功能之间进行通讯的环境中运行，则返回布尔值 `true`；否则返回布尔值 `false`。

说明

方法；指示 MSAA 屏幕读取器程序是否当前处于活动状态并且播放器是否正在支持在 Flash Player 和辅助功能之间进行通讯的环境中运行。当希望应用程序在有屏幕读取器的情况下行为方式不同时，可使用此方法。

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

注意：如果您在播放文档的 Flash 窗口第一次出现的大约 1 秒或 2 秒内调用此方法，则可能获得返回值 `false`，即使有活动的 MSAA 客户端也是如此。这是由于 Flash 和 MSAA 客户端之间的异步通讯机制造成的。您可以通过确保在加载您的文档后和调用此方法前有 1 秒到 2 秒的延迟，来解决这一限制问题。

另请参见

`Accessibility.updateProperties()`, `_accProps`, `System.capabilities.hasAccessibility`

Accessibility.updateProperties()

可用性

Flash Player 6 版本 65。

用法

Accessibility.updateProperties()

参数

无。

返回

无。

说明

方法；导致 Flash Player 重新检查所有辅助功能属性，为屏幕读取器更新其对象说明，并且根据需要将事件发送到屏幕读取器以指示已发生更改。有关设置辅助功能属性的信息，请参见 [_accProps](#)。

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

如果您修改多个对象的辅助功能属性，则只需调用 Accessibility.updateProperties() 一次；多次调用可能导致性能降低以及屏幕读取器的结果无法理解。

示例

以下动作脚本代码利用动态辅助功能属性。此示例出自可以更改它显示的图标的非文本按钮。

```
function setIcon( newIconNum, newTextEquivalent )
{
    this.iconImage = this.iconImages[ newIconNum ];
    if ( newTextEquivalent != undefined )
    {
        if ( this._accProps == undefined )
            this._accProps = new Object();
        this._accProps.name = newTextEquivalent;
        Accessibility.updateProperties();
    }
}
```

另请参见

[Accessibility.isActive\(\)](#), [_accProps](#), [System.capabilities.hasAccessibility](#)

_accProps

可用性

Flash Player 6 版本 65。

用法

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

参数

- propertyName* 辅助功能属性名（请参见以下有关有效名称的说明）。
- instanceName* 分配给影片剪辑、按钮、动态文本字段或输入文本字段的实例的实例名称。

说明

属性；用于在运行时控制 SWF 文件、影片剪辑、按钮、动态文本字段或输入文本字段的屏幕读取器辅助功能选项。这些属性覆盖创作时在“辅助功能”面板中提供的相应设置。为使对这些属性的更改生效，您必须调用 `Accessibility.updateProperties()`。有关“辅助功能”面板的信息，请参见“使用 Flash”帮助中的“Flash 辅助功能面板介绍”。

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

下表列出了每一 `_accProps` 属性的名称和数据类型、它在“辅助功能”面板中的等效设置以及可以应用该属性的对象类型。术语反转逻辑是指该属性设置与“辅助功能”面板中的相应设置相反。例如，将 `silent` 属性设置为 `true` 等效于取消选择“使影片可访问”或“使对象可访问”选项。

属性	数据类型	“辅助功能”面板中的等效设置	应用于
<code>silent</code>	布尔值	使影片可访问 / 使对象可访问 (反转逻辑)	整个影片 影片剪辑 按钮 动态文本 输入文本
<code>forceSimple</code>	布尔值	使子对象可访问（反转逻辑）	整个影片 影片剪辑
<code>name</code>	字符串	Name	整个影片 影片剪辑 按钮 输入文本
<code>description</code>	字符串	说明	整个影片 影片剪辑 按钮 动态文本 输入文本
<code>shortcut</code>	字符串	快捷键*	影片剪辑 按钮 输入文本

* 有关向可访问对象分配快捷键的信息，请参见 `Key.addListener()`。

若要在“辅助功能”面板中指定与 Tab 键索引设置相对应的设置，请使用 `Button.tabIndex`、`MovieClip.tabIndex` 或 `TextField.tabIndex` 属性。

无法在运行时指定“自动标签”设置。

如果使用时未采用 `instanceName` 参数，则对 `_accProps` 属性的更改将应用于整个影片。例如，以下代码将整个影片的辅助功能 `name` 属性设置为字符串 "Pet Store"，然后调用 `Accessibility.updateProperties()` 来使所作更改生效。

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

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

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

如果您指定若干辅助功能属性，则应在调用 `Accessibility.updateProperties()` 前进行尽可能多的更改，而不是在每一属性语句后调用它：

```
_accProps.name = "Pet Store";
animal_mc._accProps.name = "Animal";
animal_mc._accProps.description = "Cat, dog, fish, etc.";
price_mc._accProps.name = "Price";
price_mc._accProps.description = "Cost of a single item";
Accessibility.updateProperties();
```

如果您没有指定影片或对象的辅助功能属性，则实现在“辅助功能”面板中设置的任何值。

在指定一个辅助功能属性后，就不能将它的值还原为在“辅助功能”面板中设置的值。但是，可以通过删除 `_accProps` 对象，将该属性设置为它的默认值（对于布尔值，其默认值是 `false`；对于字符串值，其默认值是空字符串）：

```
my_mc._accProps.silent = true; // 设置属性
// 此处为其它代码
delete my_mc._accProps.silent; // 还原为默认值
```

若要将一个对象的所有辅助功能值都还原为默认值，可以删除 `instanceName._accProps` 对象：

```
deletemy_btn._accProps;
```

若要将所有对象的辅助功能值都还原为默认值，可以删除全局 `_accProps` 对象：

```
delete _accProps;
```

如果您把某一属性指定给不支持该属性的对象类型，那么该属性指定将被忽略，不会引发任何错误。例如，按钮不支持 `forceSimple` 属性，因此忽略如下行：

```
my_btn._accProps.forceSimple = false; //ignored
```

示例

下面是某些利用动态辅助功能属性的示例动作脚本代码。您应将该代码分配给可以更改其显示的图标而非文本图标按钮组件。

```
function setIcon( newIconNum, newTextEquivalent )
{
    this.iconImage = this.iconImages[ newIconNum ];
    if ( newTextEquivalent != undefined )
    {
        if ( this._accProps == undefined )
            this._accProps = new Object();
    }
}
```

```
        this._accProps.name = newTextEquivalent;  
        Accessibility.updateProperties();  
    }  
}
```

另请参见

[Accessibility.isActive\(\)](#), [Accessibility.updateProperties\(\)](#),
[System.capabilities.hasAccessibility](#)

add

可用性

Flash Player 4。

用法

string1 add *string2*

参数

string1、*string2* 字符串。

返回

无。

说明

运算符；连接（合并）两个或多个字符串。add 运算符替换了 Flash 4 的加法 (&) 运算符；当将使用 & 运算符的 Flash Player 4 文件引入 Flash 5 或更高版本的创作环境中时，就会自动对其进行转换以使用 add 运算符进行字符串连接。不过，在 Flash Player 5 中已不鼓励使用 add 运算符，Macromedia 建议您在为 Flash Player 5 或更高版本创建内容时使用 + 运算符。如果您为 Flash Player 4 或更早版本的播放器创建内容，则使用 add 运算符来连接字符串。

另请参见

[+（加号）](#)

and

可用性

Flash Player 4。

用法

condition1 and *condition2*

参数

condition1、*condition2* 计算结果为 true 或 false 的条件或表达式。

返回

无。

说明

运算符；在 Flash Player 4 中执行逻辑 AND 运算。如果两个表达式的计算结果均为 true，则整个表达式为 true。在 Flash 5 中已不鼓励使用该运算符，Macromedia 建议您使用 && 运算符。

另请参见

[&& \(逻辑 AND\)](#)

Arguments 类

可用性

Flash Player 5 ；属性已添加到 Flash Player 6。

说明

Arguments 类是一个数组，其中包含作为参数传递给任何函数的值。每次在动作脚本中调用函数时，都会为该函数自动创建 Arguments 对象。同时还会创建一个局部变量 arguments，使您可以引用 Arguments 对象。

Arguments 类的属性摘要

属性	说明
arguments.callee	指被调用的函数。
arguments.caller	指进行调用的函数。
arguments.length	传递给函数的参数数量。

arguments.callee

可用性

Flash Player 5。

用法

`arguments.callee`

说明

属性；指当前被调用的函数。

示例

可以使用 `arguments.callee` 属性来生成递归的匿名函数，如下所示：

```
factorial = function (x) {
    if (x <= 1) {
        return 1;
    } else {
        return x * arguments.callee(x-1);
    }
};
```

下面是一个命名的递归函数：

```
function factorial (x) {  
    if (x <= 1) {  
        return 1;  
    } else {  
        return x * factorial(x-1);  
    }  
}
```

arguments.caller

可用性

Flash Player 6。

用法

```
arguments.caller
```

说明

属性；指进行调用的函数。

arguments.length

可用性

Flash Player 5。

用法

```
arguments.length
```

说明

属性；实际传递给函数的参数数量。

Array 类

可用性

Flash Player 5（已成为 Flash Player 6 本身的对象，Flash Player 6 大大提高了性能）。

说明

Array 类使您可以访问和操纵数组。数组是一个对象，其属性由表示该属性在数组中位置的数字来标识。此数字称为“索引”。所有数组都从零开始，这意味着数组中的第一个元素为 [0]，第二个元素为 [1]，依次类推。在下面的示例中，my_array 包含了一年中的月份。

```
my_array[0] = "January"  
my_array[1] = "February"  
my_array[2] = "March"  
my_array[3] = "April"
```

若要创建 Array 对象，请使用构造函数 `new Array()` 或数组访问运算符 (`[]`)。若要访问数组中的元素，请使用数组访问运算符 (`[]`)。

Array 类的方法摘要

方法	说明
<code>Array.concat()</code>	连接参数，并将其作为新数组返回。
<code>Array.join()</code>	将数组内的所有元素联接为一个字符串。
<code>Array.pop()</code>	删除数组中最后一个元素，并返回该元素的值。
<code>Array.push()</code>	将一个或多个元素添加到数组的结尾，并返回该数组的新长度。
<code>Array.reverse()</code>	倒转数组的方向。
<code>Array.shift()</code>	删除数组中第一个元素，并返回该元素的值。
<code>Array.slice()</code>	提取数组中的一部分，并将该部分作为新数组返回。
<code>Array.sort()</code>	就地对数组进行排序。
<code>Array.sortOn()</code>	基于数组中的某个字段对数组进行排序。
<code>Array.splice()</code>	在数组中添加元素和删除元素。
<code>Array.toString()</code>	返回表示 Array 对象中元素的字符串值。
<code>Array.unshift()</code>	将一个或多个元素添加到数组的开头，并返回该数组的新长度。

Array 类的属性摘要

属性	说明
<code>Array.length</code>	指定数组中元素数量的非从零开始的整数。

Array 类的构造函数

可用性

Flash Player 5。

用法

```
new Array()  
new Array(length)  
new Array(element0, element1, element2,...elementN)
```

参数

length 一个指定数组中元素数量的整数。在元素不连续的情况下，*length* 参数指定的是数组中最后一个元素的索引号加 1。

element0...elementN 一个包含两个或多个任意值的列表。这些值可以是数字、字符串、对象或其它数组。数组中第一个元素的索引或位置始终为 0。

返回

无。

说明

构造函数；它使您可以创建数组。您可使用构造函数来创建不同类型的数组：空数组、具有特定长度但其中元素没有值的数组或其中元素具有特定值的数组。

用法 1：如果不指定任何参数，则创建长度为 0 的数组。

用法 2：如果仅指定长度，则创建元素数等于 *length* 的数组，但这些元素没有值。

用法 3：如果使用 *element* 参数指定值，则创建具有特定值的数组。

示例

用法 1：下面的示例创建初始长度为 0 的新 Array 对象。

```
my_array = new Array();  
trace(my_array.length); // 返回 0
```

用法 2：下面的示例创建初始长度为 4 的新 Array 对象。

```
my_array = new Array(4);  
trace(my_array.length); // 返回 4
```

用法 3：下面的示例创建初始长度为 5 的新 Array 对象 *go_gos_array*。

```
go_gos_array = new Array("Belinda", "Gina", "Kathy", "Charlotte", "Jane");  
trace(my_array.length); // 返回 5  
trace(go_gos_array.join(", ")); // 显示元素
```

go_gos 数组的初始元素标识如下：

```
go_gos_array[0] = "Belinda";  
go_gos_array[1] = "Gina";  
go_gos_array[2] = "Kathy";  
go_gos_array[3] = "Charlotte";  
go_gos_array[4] = "Jane";
```

下面的代码将第 6 个元素添加到 *go_gos_array* 数组中并更改第二个元素：

```
go_gos_array[5] = "Donna";  
go_gos_array[1] = "Nina";  
trace(go_gos_array.join(" + "));
```

另请参见

[Array.length, \[\]](#) (数组访问)

Array.concat()

可用性

Flash Player 5。

用法

```
my_array.concat( [ value0, value1,...valueN ])
```

参数

value0...valueN 要在新数组中连接的数字、元素或字符串。如果您没有传递任何值，则创建 *my_array* 的一个副本。

返回

无。

说明

方法；将参数中指定的元素与 *my_array* 中的元素连接，并创建新的数组。如果 *value* 参数指定的是数组，则连接该数组的元素而不是该数组本身。数组 *my_array* 保持不变。

示例

下面的代码连接两个数组。

```
alpha_array = new Array("a","b","c");
numeric_array = new Array(1,2,3);
alphaNumeric_array=alpha_array.concat(numeric_array);
trace(alphaNumeric_array);
// 创建数组 ["a","b","c",1,2,3]
```

下面的代码连接三个数组。

```
num1_array = [1,3,5];
num2_array = [2,4,6];
num3_array = [7,8,9];
nums_array=num1_array.concat(num2_array,num3_array)
trace(nums_array);
// 创建数组 [1,3,5,2,4,6,7,8,9]
```

嵌套数组不能像普通数组那样展开。嵌套数组中的元素不会分解为数组 *x_array* 中的独立元素，如下面的示例所示。

```
a_array = new Array ("a","b","c");

// 2 和 3 是嵌套数组中的元素
n_array = new Array(1, [2, 3], 4);

x_array = a_array.concat(n_array);
trace(x_array[0]); // "a"
trace(x_array[1]); // "b"
trace(x_array[2]); // "c"
trace(x_array[3]); // 1
trace(x_array[4]); // 2, 3
trace(x_array[5]); // 4
```

Array.join()

可用性

Flash Player 5。

用法

```
my_array.join([separator])
```

参数

separator 在返回字符串中分隔数组元素的字符或字符串。如果省略此参数，则使用逗号作为默认分隔符。

返回

字符串。

说明

方法；将数组中的元素转换为字符串、在元素间插入指定的分隔符、连接这些元素然后返回结果字符串。嵌套数组总是以逗号分隔，而不使用传递给 *join()* 方法的分隔符分隔。

示例

下面的示例创建一个具有三个元素的数组：Earth、Moon 和 Sun。该示例然后联接这一数组三次：第一次使用默认的分隔符（逗号和空格），然后使用破折号，最后使用加号（+）；并且在“输出”面板中显示这些数组：

```
a_array = new Array("Earth","Moon","Sun")
trace(a_array.join());
// 返回 Earth, Moon, Sun
trace(a_array.join(" - "));
// 返回 Earth - Moon - Sun
trace(a_array.join(" + "));
// 返回 Earth + Moon + Sun
```

Array.length

可用性

Flash Player 5。

用法

```
my_array.length
```

说明

属性；指定数组中元素数量的非从零开始的整数。当将新元素添加到数组时，此属性会自动更新。当您向某一数组元素赋值时（例如 `my_array[index] = value`），如果 `index` 是数字，并且 `index+1` 大于 `length` 属性，则将 `length` 属性更新为 `index+1`。

示例

下面的代码解释 `length` 属性是如何更新的。

```
my_array = new Array();
trace(my_array.length); // 初始长度为 0
my_array[0] = 'a';
trace(my_array.length); // 将 my_array.length 更新为 1
my_array[1] = 'b';
trace(my_array.length); // 将 my_array.length 更新为 2
my_array[9] = 'c';
trace(my_array.length); // 将 my_array.length 更新为 10
```

Array.pop()

可用性

Flash Player 5。

用法

```
my_array.pop()
```

参数

无。

返回

指定的数组中最后一个元素的值。

说明

方法；删除数组中最后一个元素，并返回该元素的值。

示例

下面的代码创建包含四个元素的 `myPets` 数组，然后删除其最后一个元素。

```
myPets = ["cat", "dog", "bird", "fish"];
popped = myPets.pop();
trace(popped);
// 返回 fish
```

Array.push()

可用性

Flash Player 5。

用法

```
my_array.push(value,...)
```

参数

value 要追加到数组中的一个或多个值。

返回

新数组的长度。

说明

方法；将一个或多个元素添加到数组的结尾，并返回该数组的新长度。

示例

下面的示例创建具有两个元素（`cat` 和 `dog`）的数组 `myPets`。第二行将两个元素添加到数组。在调用 `push()` 方法后，变量 `pushed` 包含四个元素。因为 `push()` 方法返回数组的新长度，所以最后一行中的 `trace()` 动作将 `myPets` 的新长度 (4) 发送到“输出”面板：

```
myPets = ["cat", "dog"];
pushed = myPets.push("bird", "fish");
trace(pushes);
```

Array.reverse()

可用性

Flash Player 5。

用法

```
my_array.reverse()
```

参数

无。

返回

无。

说明

方法；就地倒转数组。

示例

下面是一个使用此方法的示例。

```
var numbers_array = [1, 2, 3, 4, 5, 6];
trace(numbers_array.join()); // 1.2.3.4.5.6
numbers_array.reverse();
trace(numbers_array.join()); // 6,5,4,3,2,1
```

Array.shift()

可用性

Flash Player 5。

用法

```
my_array.shift()
```

参数

无。

返回

数组中的第一个元素。

说明

方法；删除数组中第一个元素，并返回该元素。

示例

下面的代码创建数组 `myPets`，然后删除该数组中的第一个元素，并将其分配给变量 `shifted`。

```
var myPets_array = ["cat", "dog", "bird", "fish"];
shifted = myPets_array.shift();
trace(shifted); // 返回 "cat"
```

另请参见

[Array.pop\(\)](#)

Array.slice()

可用性

Flash Player 5。

用法

```
my_array.slice( [ start [ , end ] ] )
```

参数

start 指定片段起始点索引的数字。如果 *start* 是负数，则起点从数组的结尾开始，其中 -1 指的是最后一个元素。

end 指定片段终点索引的数字。如果省略此参数，则片段包括数组中从开头到结尾的所有元素。如果 *end* 是负数，则终点从数组的结尾指定，其中 -1 指的是最后一个元素。

返回

一个数组。

说明

方法；提取数组中的一个片段或子字符串，并将其作为新数组返回，而不修改原始数组。返回的数组包括 *start* 元素以及从其开始到 *end* 元素（但不包括该元素）的所有元素。

如果您没有传递任何参数，则创建 *my_array* 的一个副本。

Array.sort()

可用性

Flash Player 5；附加功能已在 Flash Player 7 中添加。

用法

```
my_array.sort()  
my_array.sort(compareFunction)  
my_array.sort(option | option |... )  
my_array.sort(compareFunction, option | option |... )
```

参数

compareFunction 一个用来确定数组中元素排序顺序的可选比较函数。给定元素 A 和 B，*compareFunction* 的结果可具有以下三个值之一：

- -1，如果 A 应在排序后序列中出现在 B 之前
- 0，如果 A = B
- 1，如果 A 应在排序后序列中出现在 B 之后

option 由 | (按位 OR) 运算符分隔的一个或多个数字或字符串，这些数字或字符串更改根据默认值排序的行为。下面是 *option* 可以接受的值：

- 1 或 Array.CASEINSENSITIVE
- 2 或 Array.DESENDING
- 4 或 Array.UNIQUE
- 8 或 Array.RETURNINDEXEDARRAY
- 16 或 Array.NUMERIC

有关此参数的信息，请参见 [Array.sortOn\(\)](#)。

返回

返回值取决于是否传递任何参数：

- 如果您为 *option* 指定值 4 或 Array.UNIQUE，并且所排序的两个或多个元素具有相同的排序字段，则 Flash 返回值 0 并且不修改该数组。
- 如果您为 *option* 指定值 8 或 Array.RETURNINDEXEDARRAY，则 Flash 将返回反映排序结果的数组并且不修改该数组。

- 否则，Flash 不返回任何内容并修改该数组以反映排序顺序。

说明

方法；对数组中的元素进行排序。Flash 根据 ASCII (Unicode) 值排序。如果所比较的两个元素中的任何一个不包含在 *fieldName* 参数中指定的字段，则认为该字段是 *undefined*，并且在排序后的数组中不按任何特定顺序连续放置这些元素。

默认情况下，`Array.sort()` 按如下方式工作：

- 排序区分大小写（Z 位于 a 之前）。
- 排序是升序（a 位于 b 之前）。
- 修改该数组以反映排序顺序；在排序后的数组中不按任何特定顺序连续放置具有相同排序字段的多个元素。
- 数值字段就像是字符串一样排序，因此 100 位于 99 之前，因为“1”是比“9”低的字符串值。
- 不返回任何内容。

如果您要以其它方式排序，则创建一个进行所需排序的函数并将其名称传递给 *compareFunction* 参数。例如，如果您要按姓氏以字母顺序的升序排序，然后按邮编以降序排序，则可以创建相应的函数并传递其名称。

如果您要指定作为排序依据的一个或多个字段，并且使用默认排序或 *options* 参数，则使用 `Array.sortOn()`。

示例

用法 1：以下示例显示如何使用 `Array.sort()`，以具有和不具有为 *option* 传递的值这两种情况为例：

```
var fruits_array = ["oranges", "apples", "strawberries", "pineapples",
    "cherries"];
trace(fruits_array.join());
fruits_array.sort();
trace(fruits_array.join());
fruits_array.sort(Array.DESENDING);
trace(fruits_array.join());
```

“输出”面板将显示以下结果：

```
oranges,apples,strawberries,pineapples,cherries// 原始数组
apples,cherries,oranges,pineapples,strawberries// 默认排序
strawberries,pineapples,oranges,cherries,apples// 降序
```

用法 2：以下示例将 `Array.sort()` 与比较函数一起使用。

```
var passwords = ["mom:glam","ana:ring","jay:mag","anne:home","regina:silly"];
function order(a,b){
    // 要排序的条目采用 name:password 的形式
    // 只将条目的名称部分作为关键字排序。
    var name1 =a.split(":")[0 ];
    var name2 =b.split(":")[0 ];
    if (name1 <name2){
        return -1;
    }
    else if (name1 >name2){
        return 1;
    }
    else {
        return 0;
    }
}
```

```

    }
}
trace ("Unsorted:");
trace (passwords.join());

passwords.sort(order);
trace ("Sorted:");
trace (passwords.join());

```

“输出” 面板将显示以下结果：

```

Unsorted:
mom:glam,ana:ring,jay:mag,anne:home,regina:silly
Sorted:
ana:ring,anne:home,jay:mag,mom:glam,regina:silly

```

另请参见

| (按位 OR) , [Array.sortOn\(\)](#)

Array.sortOn()

可用性

Flash Player 6 ；附加功能已在 Flash Player 7 中添加。

用法

```

my_array.sortOn("fieldName" )
my_array.sortOn("fieldName", option | option |... )
my_array.sortOn( [ "fieldName" , "fieldName" , ... ] )
my_array.sortOn( [ "fieldName" , "fieldName" , ...] , option | option |... )

```

注意：在显示中括号 ([]) 的地方，您必须将它们包括在代码中；即，中括号不表示可选参数。

参数

fieldName 一个字符串，标识 Array 的某元素中用作排序值的字段。

option 由 | (按位 OR) 运算符分隔的一个或多个数字或字符串，这些数字或字符串更改根据默认值排序的行为。下面是 *option* 可以接受的值：

- 1 或 Array.CASEINSENSITIVE
- 2 或 Array.DESENDING
- 4 或 Array.UNIQUE
- 8 或 Array.RETURNINDEXEDARRAY
- 16 或 Array.NUMERIC

上述各个选项在下面的“说明”部分中将予以详细讨论。

返回

返回值取决于是否传递任何参数：

- 如果您为 *option* 指定值 4 或 Array.UNIQUE，并且所排序的两个或多个元素具有相同的排序字段，则 Flash 返回值 0 并且不修改该数组。
- 如果您为 *option* 指定值 8 或 Array.RETURNINDEXEDARRAY，则 Flash 将返回反映排序结果的数组并且不修改该数组。

- 否则，Flash 不返回任何内容并修改该数组以反映排序顺序。

说明

方法；基于数组中的一个或多个字段对数组中的元素进行排序。如果您传递多个 *fieldName* 参数，则第一个字段表示主排序字段，第二个字段表示下一个排序字段，依此类推。Flash 根据 ASCII (Unicode) 值排序。如果所比较的两个元素中的任何一个不包含在 *fieldName* 参数中指定的字段，则认为该字段是 *undefined*，并且在排序后的数组中不按任何特定顺序连续放置这些元素。

默认情况下，`Array.sortOn()` 按如下方式工作：

- 排序区分大小写（Z 位于 a 之前）。
- 排序是升序（a 位于 b 之前）。
- 修改该数组以反映排序顺序；在排序后的数组中不按任何特定顺序连续放置具有相同排序字段的多个元素。
- 数值字段就像是字符串一样排序，因此 100 位于 99 之前，因为“1”是比“9”低的字符串值。
- 不返回任何内容。

您可以使用 *option* 标志覆盖这些默认值。以下示例出于阐释目的使用 *option* 标志的不同形式。如果您要对简单数组（例如，只有一个字段的数组）进行排序，或者如果要指定 *options* 参数不支持的排序顺序，则使用 `Array.sort()`。

若要用数值格式传递多个标志，则用 |（按位 OR）运算符分隔它们或将这些标志值加在一起。以下代码显示三种不同的方式来指定数值降序排序：

```
my_Array.sortOn(someFieldName, 2 | 16);
my_Array.sortOn(someFieldName, 18);
my_Array.sortOn(someFieldName, Array.DESENDING | Array.NUMERIC);
```

如果您使用标志的字符串形式（例如，`DESCENDING`），而不是数值形式 (2)，则启用代码提示（请参见第 56 页的“使用代码提示”）。

考虑以下数组：

```
var my_array:Array = new Array();
my_array.push({password:"Bob", age:29});
my_array.push({password:"abcd", age:3});
my_array.push({password:"barb", age:35});
my_array.push({password:"catchy", age:4});
```

对密码字段执行默认排序将产生以下结果：

```
my_array.sortOn("password")
// Bob
// abcd
// barb
// catchy
```

对密码字段执行不区分大小写的排序将产生以下结果：

```
my_array.sortOn("password", Array.CASEINSENSITIVE)
// abcd
// barb
// Bob
// catchy
```

对密码字段执行不区分大小写的降序排序将产生以下结果：

```
my_array.sortOn("password", 1|2)
// catchy
// Bob
// barb
// abcd
```

对年龄字段执行默认排序将产生以下结果：

```
my_array.sortOn("age")
// 29
// 3
// 35
// 4
```

对年龄字段执行数值排序将产生以下结果：

```
my_array.sortOn("age", 16)
// 3
// 4
// 29
// 35
```

对年龄字段执行降序数值排序将产生以下结果：

```
my_array.sortOn("age", 18)
// 35
// 29
// 4
// 3
```

执行排序时按如下所示更改数组中的元素：

```
// 在排序前
// my_array[0].age = 29;
// my_array[1].age = 3;
// my_array[2].age = 35;
// my_array[3].age = 4;

// 在不为 option 传递值 8 的任何排序后
my_array.sortOn("age", Array.NUMERIC);
// my_array[0].age = 3;
// my_array[1].age = 4;
// my_array[2].age = 29;
// my_array[3].age = 35;
```

执行返回索引数组的排序时不更改该数组中的元素：

```
// 在排序前
// my_array[0].age = 29;
// my_array[1].age = 3;
// my_array[2].age = 35;
// my_array[3].age = 4;

// 在返回包含索引值的数组的排序后
// 请注意，原始数组未更改。
// 然后，可以使用返回的数组显示排序的信息，
// 而不必修改原始数组。
var indexArray:Array = my_array.sortOn("age", Array.RETURNINDEXEDARRAY);
// my_array[0].age = 29;
// my_array[1].age = 3;
// my_array[2].age = 35;
// my_array[3].age = 4;
```

示例

此示例创建一个新数组，并且按照字段 `name` 和 `city` 对该新数组排序：第一次排序使用 `name` 作为第一个排序值，使用 `city` 作为第二个排序值。第二次排序使用 `city` 作为第一个排序值，使用 `name` 作为第二个排序值。

```
var rec_array = new Array();
rec_array.push( { name:"john", city:"omaha", zip: 68144 } );
rec_array.push( { name:"john", city:"kansas city", zip: 72345 } );
rec_array.push( { name:"bob", city:"omaha", zip: 94010 } );
for(i=0; i<rec_array.length; i++) {
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// 结果为
// john, omaha
// john, kansas city
// bob, omaha

rec_array.sortOn( [ "name", "city" ] );
for(i=0; i<rec_array.length; i++) {
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// 结果为
// bob, omaha
// john, kansas city
// john, omaha

rec_array.sortOn( [ "city", "name" ] );
for(i=0; i<rec_array.length; i++) {
    trace(rec_array[i].name + ", " + rec_array[i].city);
}
// 结果为
// john, kansas city
// bob, omaha
// john, omaha
```

另请参见

| (按位 OR) , [Array.sort\(\)](#)

Array.splice()

可用性

Flash Player 5。

用法

```
my_array.splice(start, deleteCount [, value0, value1...valueN])
```

参数

start 插入或删除动作开始处的数组元素的索引。

deleteCount 要删除的元素数量。这一数量包含 *start* 参数中指定的元素。如果没有为 *deleteCount* 指定值，则该方法将删除从 *start* 元素到数组中最后一个元素之间的所有值。如果该参数的值为 0，则不删除任何元素。

value 一个可选参数，指定在 *start* 参数中指定的插入点插入到数组中的值。

返回

无。

说明

方法；向数组中添加元素或删除数组中元素。此方法会修改数组但不制作副本。

Array.toString()

可用性

Flash Player 5。

用法

```
my_array.toString()
```

参数

无。

返回

字符串。

说明

方法；返回表示指定 Array 对象中的元素的字符串值。数组中的每一个元素（从索引 0 开始到索引 *my_array.length-1* 结束）均会转换为一个连接字符串，并以逗号分隔。

示例

下面的示例创建 *my_array*，将其转换为字符串，然后在“输出”面板中显示 1,2,3,4,5。

```
my_array = new Array();  
my_array[0] = 1;  
my_array[1] = 2;  
my_array[2] = 3;  
my_array[3] = 4;  
my_array[4] = 5;  
trace(my_array.toString());
```

Array.unshift()

可用性

Flash Player 5。

用法

```
my_array.unshift(value1,value2,...valueN)
```

参数

value1...valueN 一个或多个要插入到数组开头的数字、元素或变量。

返回

数组的新长度。

说明

方法；将一个或多个元素添加到数组的开头，并返回该数组的新长度。

Array()

可用性

Flash Player 6。

用法

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

参数

element 要放置于数组中的一个或多个元素。

返回

一个数组。

说明

转换函数；创建新的空数组，或者将指定元素转换为数组。此函数的用法类似于使用 `Array` 构造函数创建数组（请参见第 248 页的“[Array 类的构造函数](#)”）。

asfunction

可用性

Flash Player 5。

用法

```
asfunction:function,"parameter"
```

参数

function 函数的标识符。

parameter 要传递给 *function* 参数中指定的函数的字符串。

返回

无。

说明

协议；一个用于 HTML 文本字段中 URL 的特殊协议。在 HTML 文本字段中，可使用 HTML `A` 标签超链接文本。`A` 标签的 `HREF` 属性包含可用于类似 HTTP、HTTPS 或 FTP 等标准协议的 URL。`asfunction` 协议是特定于 Flash 的一个附加协议，可使链接调用动作脚本函数。

示例

在此示例中，代码的前三行中定义了 `MyFunc()` 函数。`TextField` 对象 `myTextField` 与 HTML 文本字段相关联。文本“Click Me!”是文本字段中的一个超链接。当用户单击该超链接时，将调用 `MyFunc()` 函数：

```
function MyFunc(arg){
    trace ("You clicked me!Argument was "+arg);
}
myTextField.htmlText ="<A HREF=\"asfunction:MyFunc,Foo \">Click Me!</A>";

在单击该超链接后，下面的结果将显示在 “输出” 面板中：
You clicked me!Parameter was Foo
```

Boolean 类

可用性

Flash Player 5 （已成为 Flash Player 6 本身的对象， Flash Player 6 大大提高了性能）。

说明

Boolean 类是一种包装对象，它具有与标准 JavaScript Boolean 对象一样的功能。使用 Boolean 类可获取 Boolean 对象的原始数据类型或字符串表示形式。
必须使用构造函数 new Boolean() 创建 Boolean 对象后，然后才能调用其方法。

Boolean 类的方法摘要

方法	说明
<code>Boolean.toString()</code>	返回 Boolean 对象的字符串表示形式 ("true" 或 "false")。
<code>Boolean.valueOf()</code>	返回指定 Boolean 对象的原始值类型。

Boolean 类的构造函数

可用性

Flash Player 5。

用法

```
new Boolean([x])
```

参数

x 任何表达式。此参数是可选的。

返回

无。

说明

构造函数；创建 Boolean 对象。如果省略 x 参数，则将 Boolean 对象初始化为具有值 false。如果为 x 参数指定值，则该方法会计算它，并根据 Boolean() 函数中的规则以布尔值返回结果。

示例

下面的代码创建一个称为 myBoolean 的新的空 Boolean 对象。
myBoolean = new Boolean();

Boolean.toString()

可用性

Flash Player 5。

用法

```
myBoolean.toString()
```

参数

无。

返回

一个布尔值。

说明

方法；返回 Boolean 对象的字符串表示形式 ("true" 或 "false")。

Boolean.valueOf()

可用性

Flash Player 5。

用法

```
myBoolean.valueOf()
```

参数

无。

返回

一个布尔值。

说明

方法；如果指定 Boolean 对象的原始值类型为 true，则返回 true，如果该类型为 false，则返回 false。

示例

```
var x:Boolean = new Boolean();  
trace(x.valueOf());    // false  
x = (6==3+3);  
trace(x.valueOf());    // true
```

Boolean()

可用性

Flash Player 5 ；行为在 Flash Player 7 中发生了变化。

用法

`Boolean(expression)`

参数

expression 一个可转换为布尔值的表达式。

返回

一个布尔值或 *expression* 值，如下所述。

说明

函数；将参数 *expression* 转换为布尔值，并以如下形式返回值：

如果 *expression* 是布尔值，则返回值为 *expression*。

如果 *expression* 是数字，则在该数字不为零时返回值为 `true`，否则，返回值为 `false`。

如果 *expression* 为字符串，则返回值如下所示：

- 在以 Flash Player 6 或更低版本为目标播放器发布的文件中，字符串首先被转换为一个数字，如果该数字是非零数字，则该值为 `true`，否则为 `false`。
- 在以 Flash Player 7 或更高版本为目标播放器发布的文件中，如果字符串的长度大于零，则结果为 `true`；如果字符串是空字符串，则该值是 `false`。

如果 *expression* 未定义，则返回值为 `false`。

如果 *expression* 是影片剪辑或对象，则返回值为 `true`。

另请参见

[Boolean 类](#)

break

可用性

Flash Player 4。

用法

`break`

参数

无。

返回

无。

说明

语句；出现在一个循环（for、for..in、do while 或 while 循环）中，或者出现在与 switch 动作内特定 case 语句相关联的语句块中。break 动作可命令 Flash 跳过循环体的其余部分，停止循环动作，并执行循环语句之后的语句。当使用 break 动作时，Flash 解释程序会跳过该 case 块中的其余语句，转到包含它的 switch 动作后的第一个语句。使用 break 动作可跳出一系列嵌套的循环。

示例

下面的示例使用 break 动作退出一个循环，如果没有该动作，则该循环为无限循环。

```
i = 0;
while (true) {
    if (i >= 100) {
        break;
    }
    i++;
}
```

另请参见

[break](#), [for](#), [for..in](#), [do while](#), [while](#), [switch](#), [case](#)

Button 类

可用性

Flash Player 6。

说明

SWF 文件中的所有按钮元件都是 Button 对象的实例。您可在属性检查器中指定按钮实例名称，并通过动作脚本使用 Button 类的方法和属性来操纵按钮。按钮实例名称显示在“影片浏览器”中和“动作”面板的“插入目标路径”对话框中。

Button 类继承自 [Object](#) 类。

Button 类的方法摘要

方法	说明
Button.getDepth()	返回按钮实例的深度。

Button 类的属性摘要

属性	说明
Button._alpha	按钮实例的透明度值。
Button.enabled	指示按钮是否处于活动状态。
Button._focusrect	指示具有焦点的按钮四周是否有黄色矩形。
Button._height	按钮实例的高度，以像素为单位。
Button._highquality	应用于当前 SWF 文件的消除锯齿功能的级别。
Button.menu	将 ContextMenu 对象与 button 对象相关联。

属性	说明
<code>Button._name</code>	按钮实例的实例名称。
<code>Button._parent</code>	指向包含当前影片剪辑或对象的影片剪辑或对象的引用。
<code>Button._quality</code>	指示 SWF 文件的呈现品质。
<code>Button._rotation</code>	按钮实例的旋转度数。
<code>Button._soundbuftime</code>	预加载的声音的秒数。
<code>Button.tabEnabled</code>	指示某按钮是否包括在 Tab 键的自动排序中。
<code>Button.tabIndex</code>	指示对象的 Tab 键顺序。
<code>Button._target</code>	按钮实例的目标路径。
<code>Button.trackAsMenu</code>	指示其它按钮是否可接收鼠标按钮释放事件。
<code>Button._url</code>	创建按钮实例的 SWF 文件的 URL。
<code>Button.useHandCursor</code>	指示当鼠标指针经过按钮上方时是否显示手指形光标。
<code>Button._visible</code>	指示按钮实例是隐藏还是可见的布尔值。
<code>Button._width</code>	按钮实例的宽度，以像素为单位。
<code>Button._x</code>	按钮实例的 x 坐标。
<code>Button._xmouse</code>	鼠标指针相对于按钮实例的 x 坐标。
<code>Button._xscale</code>	指定按钮实例水平缩放百分比的值。
<code>Button._y</code>	按钮实例的 y 坐标。
<code>Button._ymouse</code>	鼠标指针相对于按钮实例的 y 坐标。
<code>Button._yscale</code>	指定按钮实例垂直缩放百分比的值。

Button 类的事件处理函数摘要

事件处理函数	说明
<code>Button.onDragOut</code>	当在按钮上按下鼠标按钮，然后将鼠标指针滑出按钮时调用。
<code>Button.onDragOver</code>	当用户在按钮外部按下鼠标按钮，然后将鼠标指针拖动到按钮之上时调用。
<code>Button.onKeyUp</code>	当释放按键时调用。
<code>Button.onKillFocus</code>	当从按钮移除焦点时调用。
<code>Button.onPress</code>	在鼠标指针位于按钮上方的情况下，按下鼠标按钮时调用。
<code>Button.onRelease</code>	在鼠标指针位于按钮上方的情况下，释放鼠标按钮时调用。
<code>Button.onReleaseOutside</code>	在这样的情况下调用：在鼠标指针位于按钮内部的情况下按下按钮，然后将鼠标指针移到该按钮外部并释放鼠标按钮。
<code>Button.onRollOut</code>	当鼠标指针滚动到按钮区域之外时调用。
<code>Button.onRollOver</code>	当鼠标指针滚过按钮时调用。
<code>Button.onSetFocus</code>	当按钮具有输入焦点而且释放某按键时调用。

Button._alpha

可用性

Flash Player 6。

用法

`my_btn._alpha`

说明

属性；由 `my_btn` 指定的按钮的 `alpha` 透明度的值。有效值为 0（完全透明）到 100（完全不透明）。默认值为 100。按钮的 `_alpha` 设置为 0 时，其中的对象处于活动状态，即使这些对象不可见。

示例

下面的代码在按钮被单击时将名为 `star_btn` 的按钮的 `_alpha` 属性设置为 30%：

```
on (release) {  
    star_btn._alpha = 30;  
}
```

另请参见

[MovieClip._alpha](#), [TextField._alpha](#)

Button.enabled

可用性

Flash Player 6。

用法

`my_btn.enabled`

说明

属性；指定按钮是否处于启用状态的布尔值。默认值为 `true`。

Button._focusrect

可用性

Flash Player 6。

用法

`my_btn._focusrect`

说明

属性；一个布尔值，指定当按钮具有键盘焦点时，其四周是否有黄色矩形。此属性可以覆盖全局 `_focusrect` 属性。

Button.getDepth()

可用性

Flash Player 6。

用法

```
my_btn.getDepth()
```

返回

一个整数。

说明

方法；返回按钮实例的深度。

Button._height

可用性

Flash Player 6。

用法

```
my_btn._height
```

说明

属性；以像素为单位的按钮的高度。

示例

下面的代码示例在用户单击鼠标时，设置按钮的高度和宽度：

```
my_btn._width = 200;  
my_btn._height = 200;
```

Button._highquality

可用性

Flash Player 6。

用法

```
my_btn._highquality
```

说明

属性（全局）；指定应用于当前 SWF 文件的锯齿消除级别。指定 2（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 1（高品质），则应用消除锯齿功能；如果 SWF 文件不包含动画，这将对位图进行平滑处理。指定 0（低品质），则不消除锯齿。

另请参见

[_quality](#)

Button.menu

可用性

Flash Player 7。

用法

```
my_button.menu = contextMenu
```

参数

contextMenu 一个 ContextMenu 对象。

说明

属性；将 ContextMenu 对象 *contextMenu* 与按钮对象 *my_button* 相关联。ContextMenu 类用于修改当用户在 Flash Player 中右击 (Windows) 或按住 Control 键单击 (Macintosh) 时显示的上下文菜单。

示例

下面的示例将 ContextMenu 对象分配给一个名为 *save_btn* 的 Button 对象。ContextMenu 对象包含单个菜单项（标记有“Save...”），该菜单项具有名为 *doSave* 的关联回调处理函数（未显示）。

```
var menu_cm = new ContextMenu();  
menu_cm.customItems.push(new ContextMenuItem("Save...", doSave));  
function doSave(menu, obj) {  
    // "Save" 代码  
}  
save_btn.menu = menu_cm;
```

另请参见

[ContextMenu 类](#), [ContextMenuItem 类](#), [MovieClip.menu](#), [TextField.menu](#)

Button._name

可用性

Flash Player 6。

用法

```
my_btn._name
```

说明

属性；由 *my_btn* 指定的按钮的实例名称。

Button.onDragOut

可用性

Flash Player 6。

用法

```
my_btn.onDragOut = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当在按钮上按下鼠标按钮，然后将鼠标指针滑出按钮时调用。

必须定义一个在调用事件处理函数时执行的函数。

Button.onDragOver

可用性

Flash Player 6。

用法

```
my_btn.onDragOver = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当用户在按钮外部按下鼠标按钮，然后将鼠标指针拖动到按钮之上时调用。

必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onKeyDown` 处理函数定义一个函数，该函数将 `trace()` 动作发送到“输出”面板：

```
my_btn.onDragOver = function () {  
    trace ("onDragOver called");  
};
```

另请参见

[Button.onKeyUp](#)

Button.onKeyDown

可用性

Flash Player 6。

用法

```
my_btn.onKeyDown = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当按钮具有键盘焦点并按下某按键时调用。调用 `onKeyDown` 事件处理函数时没有参数。可以使用 `Key.getAscii()` 和 `Key.getCode()` 来确定按下了哪个键。

必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onKeyDown` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onKeyDown = function () {  
    trace ("onKeyDown called");  
};
```

另请参见

[Button.onKeyUp](#)

Button.onKeyUp

可用性

Flash Player 6。

用法

```
my_btn.onKeyUp = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当按钮具有输入焦点而且释放某按键时调用。调用 `onKeyUp` 事件处理函数时没有参数。可以使用 `Key.getAscii()` 和 `Key.getCode()` 来确定按下了哪个键。

必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onKeyPress` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onKeyUp = function () {  
    trace ("onKeyUp called");  
};
```

Button.onKillFocus

可用性

Flash Player 6。

用法

```
my_btn.onKillFocus = function (newFocus) {  
    // 此处是您的语句  
}
```

参数

newFocus 要接收焦点的对象。

返回

无。

说明

事件处理函数；当按钮失去键盘焦点时调用。`onKillFocus` 方法接收一个参数 *newFocus*，该参数是一个对象，表示要接收焦点的新对象。如果没有对象接收焦点，则 *newFocus* 将包含值 `null`。

Button.onPress

可用性

Flash Player 6。

用法

```
my_btn.onPress = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当按下按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onPress` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onPress = function () {  
    trace ("onPress called");  
};
```

Button.onRelease

可用性

Flash Player 6。

用法

```
my_btn.onRelease = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当释放按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onRelease` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onRelease = function () {  
    trace ("onRelease called");  
};
```

Button.onReleaseOutside

可用性

Flash Player 6。

用法

```
my_btn.onReleaseOutside = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；在鼠标指针位于按钮内部时按下按钮后，将鼠标指针移到该按钮外部然后释放鼠标按钮，在这种情况下进行调用。

必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onReleaseOutside` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onReleaseOutside = function () {  
    trace ("onReleaseOutside called");  
};
```

Button.onRollOut

可用性

Flash Player 6。

用法

```
my_btn.onRollOut = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当指针移出按钮区域时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onRollOut` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onRollOut = function () {  
    trace ("onRollOut called");  
};
```

Button.onRollOver

可用性

Flash Player 6。

用法

```
my_btn.onRollOver = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当指针移过按钮区域时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

在下面的示例中，为 `onRollOver` 处理函数定义将 `trace()` 动作发送到“输出”面板的函数。

```
my_btn.onRollOver = function () {  
    trace ("onRollOver called");  
};
```

Button.onSetFocus

可用性

Flash Player 6。

用法

```
my_btn.onSetFocus = function(oldFocus){  
    // 此处是您的语句  
}
```

参数

oldFocus 将失去键盘焦点的对象。

返回

无。

说明

事件处理函数；当按钮接收键盘焦点时调用。*oldFocus* 参数是失去焦点的对象。例如，如果用户按下 Tab 键将输入焦点从文本字段移到按钮，则 *oldFocus* 包含该文本字段实例。

如果以前没有具有焦点的对象，则 *oldFocus* 包含 `null` 值。

Button._parent

可用性

Flash Player 6。

用法

```
my_btn._parent.property  
_parent.property
```

说明

属性；指向包含当前影片剪辑或对象的影片剪辑或对象的引用。当前对象是包含引用 `_parent` 的动作脚本代码的对象。

使用 `_parent` 可以指定一个相对路径，该路径指向当前影片剪辑或对象之上的影片剪辑或对象。可以使用 `_parent` 在显示列表中攀升多个级别，如下所示：

```
_parent._parent._alpha = 20;
```

另请参见

[MovieClip._parent](#), [_root](#), [targetPath](#)

Button._quality

可用性

Flash Player 6。

用法

```
my_btn._quality
```

说明

属性（全局）；设置或获取用于 SWF 文件的呈现品质。设备字体始终是带有锯齿的，因此不受 `_quality` 属性的影响。

注意：尽管您可以为 Button 对象指定该属性，但它实际上是一个全局属性，因此您只能将它的值指定为 `_quality`。有关更多信息，请参见 [_quality](#)。

Button._rotation

可用性

Flash Player 6。

用法

```
my_btn._rotation
```

说明

属性；以度为单位的按钮距其原始方向的旋转程度。从 0 到 180 的值表示顺时针方向的旋转；从 0 到 -180 的值表示逆时针方向的旋转。对于此范围之外的值，可以通过加上或减去 360 获得该范围内的值。例如，语句 `my_btn._rotation = 450` 与 `my_btn._rotation = 90` 相同。

另请参见

[MovieClip._rotation](#), [TextField._rotation](#)

Button._soundbuftime

可用性

Flash Player 6。

用法

```
myButton._soundbuftime
```

说明

属性（全局）；一个整数，指定在声音开始进入流之前，预先缓冲的秒数。

注意：尽管您可以为 Button 对象指定该属性，但它实际上是一个全局属性，因此您只能将它的值指定为 `_soundbuftime`。有关更多信息，请参见 [_soundbuftime](#)。

Button.tabEnabled

可用性

Flash Player 6。

用法

```
my_btn.tabEnabled
```

说明

属性；指定 *my_btn* 是否包含在 Tab 键的自动排序中。默认情况下它是 `undefined`。

如果 `tabEnabled` 属性为 `undefined` 或 `true`，则该对象包括在 Tab 键的自动排序中。如果 `tabIndex` 属性也设置为某个值，则该对象也包括在 Tab 键的自定义排序中。如果 `tabEnabled` 为 `false`，则即使设置了 `tabIndex` 属性，该对象也不包括在 Tab 键的自动或自定义排序中。

另请参见

[Button.tabIndex](#), [MovieClip.tabEnabled](#), [TextField.tabEnabled](#)

Button.tabIndex

可用性

Flash Player 6。

用法

```
my_btn.tabIndex
```

说明

属性；可用于自定义 SWF 文件中对象的 Tab 键排序。可以设置按钮、影片剪辑或文本字段实例的 `tabIndex` 属性；默认情况下，它为 `undefined`。

如果 SWF 文件中当前显示的任意对象都包含 `tabIndex` 属性，则禁用 Tab 键的自动排序，而使用该 SWF 文件中对象的 `tabIndex` 属性来计算 Tab 键排序。Tab 键的自定义排序仅包括那些具有 `tabIndex` 属性的对象。

`tabIndex` 属性可以是非负整数。将根据这些对象的 `tabIndex` 属性按升序对这些对象进行排序。`tabIndex` 值为 1 的对象位于 `tabIndex` 值为 2 的对象之前。如果两个对象具有相同的 `tabIndex` 值，则在 Tab 键排序中哪个对象在前面是未定义的。

由 `tabIndex` 属性定义的 Tab 键的自定义排序是平构的。这意味着不考虑 SWF 文件中对象的层次结构关系。SWF 文件中具有 `tabIndex` 属性的所有对象都排入 Tab 键顺序中，而 Tab 键顺序由 `tabIndex` 值的顺序确定。如果两个对象具有相同的 `tabIndex` 值，则哪个对象在前面是不确定的。多个对象不应使用相同的 `tabIndex` 值。

另请参见

[Button.tabEnabled](#), [MovieClip.tabChildren](#), [MovieClip.tabEnabled](#),
[MovieClip.tabIndex](#), [TextField.tabIndex](#)

Button._target

可用性

Flash Player 6。

用法

myButton._target

说明

属性（只读）；返回 *my_btn* 指定的按钮实例的目标路径。

另请参见

[targetPath](#)

Button.trackAsMenu

可用性

Flash Player 6。

用法

my_btn.trackAsMenu

说明

属性；指示其它按钮或影片剪辑是否可接收鼠标按钮释放事件的布尔值。这将允许您创建菜单。您可以对任何按钮或影片剪辑对象设置 `trackAsMenu` 属性。如果 `trackAsMenu` 属性未定义，则默认行为是 `false`。

可以在任何时间更改 `trackAsMenu` 属性；修改后的按钮会立即采用新的行为。

另请参见

[MovieClip.trackAsMenu](#)

Button._url

可用性

Flash Player 6。

用法

```
my_btn._url
```

说明

属性（只读）；获取创建按钮的 SWF 文件的 URL。

Button.useHandCursor

可用性

Flash Player 6。

用法

```
my_btn.useHandCursor
```

说明

属性；一个布尔值，当设置为 `true`（默认值）时，指示在用户用鼠标指针滑过按钮上方时是否显示手形光标（手指形）。如果将该属性设置为 `false`，则将改用箭头光标。

可以在任何时间更改 `useHandCursor` 属性；修改后的按钮会立即采用新的光标行为。可以从原型对象读出 `useHandCursor` 属性。

Button._visible

可用性

Flash Player 6。

用法

```
my_btn._visible
```

说明

属性；一个布尔值，指示由 `my_btn` 指定的按钮是否可见。不可见的按钮（将 `_visible` 属性设置为 `false`）被禁用。

另请参见

[MovieClip._visible](#), [TextField._visible](#)

Button._width

可用性

Flash Player 6。

用法

my_btn._width

说明

属性；以像素为单位的按钮的宽度。

示例

下面的示例设置一个按钮的高度和宽度属性。

```
my_btn._width=200;  
my_btn._height=200;
```

另请参见

[MovieClip._width](#)

Button._x

可用性

Flash Player 6。

用法

my_btn._x

说明

属性；一个设置按钮 *x* 坐标（该坐标相对于父级影片剪辑的本地坐标）的整数。如果按钮在主时间轴上，则其坐标系统将舞台的左上角作为 (0, 0)。如果按钮在具有变形的影片剪辑内，则该按钮位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 90 度的影片剪辑，其中的按钮将继续继承逆时针旋转 90 度的坐标系统。按钮的坐标指的是注册点的位置。

另请参见

[Button._xscale](#), [Button._y](#), [Button._yscale](#)

Button._xmouse

可用性

Flash Player 6。

用法

my_btn._xmouse

说明

属性（只读）；返回鼠标位置的 *x* 坐标（相对于按钮）。

另请参见

[Button._ymouse](#)

Button._xscale

可用性

Flash Player 6。

用法

my_btn._xscale

说明

属性；从按钮注册点开始应用的按钮水平缩放比例（以百分比表示）。默认注册点为 (0,0)。

缩放本地坐标系将影响 `_x` 和 `_y` 属性的设置，这些设置是以像素为单位定义的。例如，如果父级影片剪辑缩放到 50%，则设置 `_x` 属性将移动按钮中的对象，距离为在 SWF 文件为 100% 时其像素数的一半。

另请参见

[Button._x](#), [Button._y](#), [Button._yscale](#)

Button._y

可用性

Flash Player 6。

用法

my_btn._y

说明

属性；按钮的 `y` 坐标，该坐标相对于父级影片剪辑的本地坐标。如果按钮在主时间轴上，则其坐标系将舞台的左上角作为 (0, 0)。如果按钮在具有变形的另一个影片剪辑内，则该按钮将位于包含它的影片剪辑的本地坐标系中。因此，对于逆时针旋转 90 度的影片剪辑，其中的按钮将继承逆时针旋转 90 度的坐标系。按钮的坐标指的是注册点的位置。

另请参见

[Button._x](#), [Button._xscale](#), [Button._yscale](#)

Button._ymouse

可用性

Flash Player 6。

用法

my_btn._ymouse

说明

属性（只读）；指示鼠标位置的 `y` 坐标（相对于按钮）。

另请参见

[Button._xmouse](#)

Button._yscale

可用性

Flash Player 6。

用法

```
my_btn._yscale
```

说明

属性；从按钮注册点开始应用的按钮垂直缩放比例（以百分比表示）。默认注册点为 (0,0)。

另请参见

[Button._y](#), [Button._x](#), [Button._xscale](#)

call()

可用性

Flash Player 4。在 Flash 5 中已不鼓励使用此动作，Macromedia 建议您改用 `function` 动作。

用法

```
call(frame)
```

参数

frame 时间轴中帧的标签或编号。

返回

无。

说明

不鼓励使用的动作；执行被调用帧中的脚本，而不将播放头移动到该帧。在脚本执行后，局部变量将消失。

另请参见

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

Camera 类

可用性

Flash Player 6。

说明

Camera 类主要用于 Macromedia Flash Communication Server，但也可以用在其它地方，只是在使用上受到限制。

可通过 Camera 类捕获来自连接到计算机（该计算机应运行 Macromedia Flash Player）的摄像机的视频，例如监视来自连接到本地系统的 Web 摄像机的视频输入信号。（Flash 提供类似的音频功能；有关更多信息，请参见 [Microphone](#) 类条目。）

若要创建或引用 Camera 对象，请使用 [Camera.get\(\)](#)。

Camera 类的方法摘要

方法	说明
Camera.get()	返回默认或指定的 Camera 对象；如果没有可用的摄像机，则返回 null。
Camera.setMode()	设置摄像机捕获模式的各个方面，包括高度、宽度和每秒播放的帧数。
Camera.setMotionLevel()	指定调用 Camera.onActivity(true) 所需的运动量，并且指定在没有运动多长时间后才会调用 Camera.onActivity(false) 。
Camera.setQuality()	一个整数，指定当前输出视频输入信号可以使用的最大带宽量，以每秒字节数为单位。

Camera 类的属性摘要

属性（只读）	说明
Camera.activityLevel	所检测的摄像机的运动量。
Camera.bandwidth	当前输出视频输入信号可以使用的最大带宽量，以字节为单位。
Camera.currentFps	摄像机捕获数据的速率，以每秒帧数为单位。
Camera.fps	您希望摄像机捕获数据的速率，以每秒帧数为单位。
Camera.height	当前捕获高度，以像素为单位。
Camera.index	在 Camera.names 返回的数组中反映的摄像机的索引。
Camera.motionLevel	调用 Camera.onActivity(true) 所需的运动量。
Camera.motionTimeOut	摄像机停止检测运动的时刻和调用 Camera.onActivity(false) 的时刻之间相差的毫秒数。
Camera.muted	一个布尔值，指定用户是允许还是拒绝访问摄像机。
Camera.name	摄像机硬件指定的摄像机的名称。
Camera.names	Class 属性；一个字符串数组，反映所有可用视频捕获设备（包括显卡和摄像机在内）的名称。
Camera.quality	一个整数，指定所需的图片品质级别，该级别由应用于每一视频帧的压缩量确定。
Camera.width	当前捕获宽度，以像素为单位。

Camera 类的事件处理函数摘要

事件处理函数	说明
Camera.onActivity	在摄像机开始或停止检测运动时调用。
Camera.onStatus	当用户允许或拒绝访问摄像机时调用。

Camera 类的构造函数

请参见 [Camera.get\(\)](#)。

Camera.activityLevel

可用性

Flash Player 6。

用法

```
active_cam.activityLevel
```

说明

只读属性；指定摄像机检测的运动量的数值。该数值的范围是从 0（检测到没有运动）到 100（检测到大范围运动）。该属性的值可以帮助您确定是否需要将设置传递到 [Camera.setMotionLevel\(\)](#)。

如果摄像机可用，但却因为尚未调用 [Video.attachVideo\(\)](#) 而未被使用，则此属性设置为 -1。

如果只传送未压缩的本地视频，则只有在为 [Camera.onActivity](#) 事件处理函数分配函数后才能设置此属性。否则，它是未定义的。

另请参见

[Camera.motionLevel](#), [Camera.setMotionLevel\(\)](#)

Camera.bandwidth

可用性

Flash Player 6。

用法

```
active_cam.bandwidth
```

说明

只读属性；一个整数，指定当前输出视频输入信号可以使用的最大带宽量，以字节为单位。值 0 意味着，Flash 视频可以使用所需数量的带宽来保持指定的帧品质。

若要设置此属性，请使用 [Camera.setQuality\(\)](#)。

示例

下面的示例在摄像机带宽为 32 KB 或更高的情况下加载另一个 SWF 文件。

```
if(myCam.bandwidth >= 32768){  
    loadMovie("splat.swf",_root.hiddenvar);  
}
```

另请参见

[Camera.setQuality\(\)](#)

Camera.currentFps

可用性

Flash Player 6。

用法

active_cam.currentFps

说明

只读属性；摄像机捕获数据的速率，以每秒帧数为单位。无法设置此属性；不过，您可以使用 [Camera.setMode\(\)](#) 方法设置相关属性 [Camera.fps](#)，该属性指定您希望摄像机捕获数据的最大帧速率。

另请参见

[Camera.fps](#), [Camera.setMode\(\)](#)

Camera.fps

可用性

Flash Player 6。

用法

active_cam.fps

说明

只读属性；您希望摄像机捕获数据的最大速率，以每秒帧数为单位。可能的最大速率取决于摄像机的性能；即，如果摄像机不支持您在此处设置的值，则不会达到该帧速率。

- 若要为该属性设置所需值，请使用 [Camera.setMode\(\)](#)。
- 若要确定摄像机当前捕获数据的速率，请使用 [Camera.currentFps](#) 属性。

示例

下面的示例将活动摄像机的 `fps` 速率 `myCam.fps` 设置为用户的文本框所提供的值

```
this.config.txt_fps。
```

```
if (this.config.txt_fps != undefined) {  
    myCam.setMode(myCam.width, myCam.height, this.config.txt_fps, false);  
}
```

注意： `setMode` 函数不保证满足所要求的 `fps` 设置；该函数设置所要求的 `fps` 或可用的最快 `fps`。

另请参见

[Camera.currentFps](#), [Camera.setMode\(\)](#)

Camera.get()

可用性

Flash Player 6。

用法

```
Camera.get([index])
```

注意：正确的语法为 `Camera.get()`。若要将 `Camera` 对象分配给一个变量，请使用类似 `active_cam = Camera.get()` 的语法。

参数

index 一个从零开始的可选整数，指定要获取的摄像机，该整数根据 `Camera.names` 属性返回的数组确定。若要获取默认的摄像机（建议用于大多数应用程序），请省略此参数。

返回

- 如果未指定 *index*，则此方法返回对默认摄像机的引用；或者，如果默认摄像机正由其它应用程序使用，则此方法返回对第一个可用摄像机的引用。（如果安装了多个摄像机，则用户可以在 Flash Player 的“摄像机设置”面板中指定默认的摄像机。）如果没有可用摄像机或没有安装任何摄像机，则该方法返回 `null`。
- 如果指定 *index*，则该方法返回对请求的摄像机的引用；如果请求的摄像机不可用，则返回 `null`。

说明

方法；为捕获视频返回对 `Camera` 对象的引用。若要实际开始捕获视频，必须将 `Camera` 对象附加到 `Video` 对象（请参见 `Video.attachVideo()`）。

与使用 `new` 构造函数创建的对象不同，对 `Camera.get()` 的多个调用引用同一摄像机。因此，如果您的脚本包含 `first_cam = Camera.get()` 和 `second_cam = Camera.get()` 这两行，则 `first_cam` 和 `second_cam` 引用同一（默认）摄像机。

通常，不应传递 *index* 的值；只使用 `Camera.get()` 来返回对默认摄像机的引用。通过“摄像机设置”面板（将在本节的后面进行讨论），用户可以指定 Flash 应使用的默认摄像机。如果您传递 *index* 的值，则您可能会尝试引用非用户指定的其它摄像机。您可能在极少的情况下使用 *index*；例如，您的应用程序同时从两个摄像机捕获视频。

在一个 SWF 文件尝试访问 `Camera.get()` 返回的摄像机时，Flash Player 显示“隐私”对话框，用户可从中选择是允许还是拒绝对摄像机的访问。（请确保舞台大小至少为 215 x 138 像素；这是 Flash 显示该对话框所需的最小大小。）



当用户对此对话框做出响应时，`Camera.onStatus` 事件处理函数将返回指示用户响应的信息对象。若要不处理此事件处理函数就确定用户是拒绝还是允许对摄像机的访问，请使用 `Camera.muted` 属性。

用户也可以为特定域指定永久隐私设置，方法是在 SWF 文件播放过程中右击 (Windows) 或按住 Control 键单击 (Macintosh)，选择 “设置”，打开 “隐私” 面板，然后选择 “记住”。



您不能使用动作脚本来设置用户的 Allow 或 Deny 值，但可以通过使用 `System.showSettings(0)` 来为用户显示 “隐私” 面板。如果用户选择 “记住”，则 Flash Player 不再为该域的视频显示 “隐私” 对话框。

如果 `Camera.get` 返回 `null`，则摄像机正由其它应用程序使用，或者在系统上没有安装任何摄像机。若要确定是否安装了任何摄像机，请使用 `Camera.names.length`。若要显示 Flash Player 的 “摄像机设置” 面板（让用户选择由 `Camera.get()` 引用的摄像机），请使用 `System.showSettings(3)`。



扫描硬件以找到摄像机需要花一些时间。在 Flash 找到至少一个摄像机后，在该播放器实例的生存期内不再扫描硬件。不过，如果 Flash 没有找到任何摄像机，则它在每次调用 `Camera.get` 时都进行扫描。这在用户忘记连接摄像机时非常有用；如果您的 SWF 文件提供调用 `Camera.get` 的 “重试” 按钮，则用户不必重新启动该 SWF 文件，Flash 即可找到摄像机。

示例

下面的示例在舞台上名为 `my_video` 的 Video 对象内本地捕获和显示视频。

```
var my_cam = Camera.get();
my_video.attachVideo(myCam);
```

另请参见

[Camera.index](#), [Camera.muted](#), [Camera.names](#), [Camera.onStatus](#), [Camera.setMode\(\)](#), [System.showSettings\(\)](#), [Video.attachVideo\(\)](#)

Camera.height

可用性

Flash Player 6。

用法

`active_cam.height`

说明

只读属性；当前的捕获高度，以像素为单位。若要设置此属性的值，请使用 [Camera.setMode\(\)](#)。

示例

以下代码行用当前高度值更新用户界面中的文本框。

```
my_txt._height = myCam.height;
```

另请参见有关 [Camera.setMode\(\)](#) 的示例。

另请参见

[Camera.setMode\(\)](#), [Camera.width](#)

Camera.index

可用性

Flash Player 6。

用法

`active_cam.index`

说明

只读属性；一个从零开始的整数，指定 [Camera.names](#) 返回的数组中反映的摄像机的索引。

示例

下面的示例获具有值 `index` 的摄像机。

```
my_cam = Camera.get(index);
```

另请参见

[Camera.get\(\)](#), [Camera.names](#)

Camera.motionLevel

可用性

Flash Player 6。

用法

```
active_cam.motionLevel
```

说明

只读属性；指定调用 `Camera.onActivity(true)` 所需的运动量的数值。可接受的值的范围是从 0 到 100。默认值为 50。

无论 `motionLevel` 属性的值是多少都能显示视频。有关更多信息，请参见 [Camera.setMotionLevel\(\)](#)。

另请参见

[Camera.activityLevel](#), [Camera.onActivity](#), [Camera.onStatus](#), [Camera.setMotionLevel\(\)](#)

Camera.motionTimeOut

可用性

Flash Player 6。

用法

```
active_cam.motionTimeOut
```

说明

只读属性；摄像机停止检测运动的时刻和调用 `Camera.onActivity(false)` 的时刻之间相差的毫秒数。默认值为 2000（2 秒）。

若要设置此值，请使用 [Camera.setMotionLevel\(\)](#)。

示例

下面的示例将摄像机停止检测运动的时刻和调用 `Camera.onActivity(false)` 的时刻之间相差的毫秒数设置为 1000 毫秒，即 1 秒。

```
if(my_cam.motionTimeOut >= 1000){  
    my_cam.setMotionLevel(myCam.motionLevel, 1000);  
}
```

另请参见

[Camera.onActivity](#), [Camera.setMotionLevel\(\)](#)

Camera.muted

可用性

Flash Player 6。

用法

```
active_cam.muted
```

说明

只读属性；一个布尔值，指定用户在 Flash Player 的“隐私设置”面板中是拒绝对摄像机的访问 (`true`) 还是允许访问 (`false`)。当此值出现更改时，将调用 `Camera.onStatus`。有关更多信息，请参见 `Camera.get()`。

另请参见

`Camera.get()`, `Camera.onStatus`

Camera.name

可用性

Flash Player 6。

用法

```
active_cam.name
```

说明

只读属性；一个字符串，指定由摄像机硬件返回的当前摄像机的名称。

示例

下面的示例在“输出”面板中显示默认摄像机的名称。在 Windows 中，该名称与“扫描仪和摄像机”属性表中列出的设备名称相同。

```
my_cam = Camera.get();  
trace("The camera name is:" + my_cam.name);
```

另请参见

`Camera.get()`, `Camera.names`

Camera.names

可用性

Flash Player 6。

用法

```
Camera.names
```

注意：正确的语法为 `Camera.names`。若要向变量分配一个返回值，请使用类似 `cam_array = Camera.names` 的语法。若要确定当前摄像机的名称，请使用 `active_cam.name`。

说明

只读类属性；获取反映所有可用摄像机名称而不显示 Flash Player 的“隐私设置”面板的字符串数组。此数组的行为与其它任何动作脚本数组的行为相同，隐式提供每一摄像机从零开始的索引以及系统上摄像机的数目（通过 `Camera.names.length`）。有关更多信息，请参见 [Array 类](#) 条目。

调用 `Camera.names` 属性要求全面检查硬件，并且它可能花几秒钟的时间来生成数组。在大多数情况下，您只能使用默认的摄像机。

示例

下面的示例在没有多个可用摄像机的情况下使用默认的摄像机；如果有多个可用摄像机，用户可以选择将哪个摄像机设置为默认摄像机。

```
cam_array = Camera.names;
if (cam_array.length == 1){
    my_cam = Camera.get();
}
else {
    System.showSettings(3);
    my_cam = Camera.get();
}
```

另请参见

[Camera.get\(\)](#), [Camera.index](#), [Camera.name](#)

Camera.onActivity

可用性

Flash Player 6。

用法

```
active_cam.onActivity = function(activity) {
    // 此处是您的语句
}
```

参数

activity 一个布尔值，在摄像机开始检测运动时设置为 `true`，在摄像机停止检测运动时设置为 `false`。

返回

无。

说明

事件处理函数；在摄像机开始或停止检测运动时调用。如果要对此事件处理函数做出响应，则必须创建一个函数来处理其 *activity* 值。

若要指定调用 `Camera.onActivity(true)` 所需的运动量以及必须经过多少时间没有活动才调用 `Camera.onActivity(false)`，请使用 [Camera.setMotionLevel\(\)](#)。

示例

下面的示例在摄像机开始或停止检测运动时在“输出”面板中显示 `true` 或 `false`。

```
// 假设名为“myVideoObject”的 Video 对象位于舞台上
my_cam = Camera.get();
myVideoObject.attachVideo(my_cam);
my_cam.setMotionLevel(10, 500);
my_cam.onActivity = function(mode)
{
    trace(mode);
}
```

另请参见

[Camera.onActivity](#), [Camera.setMotionLevel\(\)](#)

Camera.onStatus

可用性

Flash Player 6。

用法

```
active_cam.onStatus = function(infoObject) {
    // 此处是您的语句
}
```

参数

infoObject 按照状态消息定义的参数。

返回

无。

说明

事件处理函数；在用户允许或拒绝对摄像机的访问时调用。如果您要响应此事件处理函数，必须创建一个函数来处理摄像机生成的信息对象。

当一个 SWF 文件尝试访问摄像机时，Flash Player 显示“隐私”对话框，用户可以选择是允许还是拒绝访问。

- 如果用户允许访问，则 [Camera.muted](#) 属性设置为 `false`，并且用 `code` 属性为“Camera.Unmuted”、`level` 属性为“Status”的信息对象调用此处理函数。
- 如果用户拒绝访问，则 [Camera.muted](#) 属性设置为 `true`，并且用 `code` 属性为“Camera.Muted”、`level` 属性为“Status”的信息对象调用此处理函数。

若要不处理此事件处理函数就确定用户是拒绝还是允许对摄像机的访问，请使用 [Camera.muted](#) 属性。

注意：如果用户选择永久允许或拒绝对来自指定域的所有 SWF 文件的访问，则对于来自该域的 SWF 文件不调用此处理函数，除非用户以后更改该隐私设置。有关更多信息，请参见 [Camera.get\(\)](#)。

示例

下面的事件处理函数无论用户拒绝还是允许对摄像机的访问都显示一个消息。

```
myCam = Camera.get();
myVideoObject.attachVideo(myCam);
```

```
myCam.onStatus = function(infoMsg) {  
    if(infoMsg.code == "Camera.Muted"){  
        trace("User denies access to the camera");  
    }  
    else  
        trace("User allows access to the camera");  
}  
// 更改 Allow 或 Deny 值以调用该函数  
System.showSettings(0);
```

另请参见

[Camera.get\(\)](#), [Camera.muted](#)

Camera.quality

可用性

Flash Player 6。

用法

active_cam.quality

说明

只读属性；一个指定所需图片品质等级的整数，图片品质等级由应用于每一视频帧的压缩量确定。可接受的品质值范围是从 1（最低品质，最大压缩）到 100（最高品质，无压缩）。默认值为 0，这意味着图片品质可以根据需要进行变化，以避免超出可用带宽。

另请参见

[Camera.setQuality\(\)](#)

Camera.setMode()

可用性

Flash Player 6。

用法

active_cam.setMode(*width*, *height*, *fps* [,*favorSize*])

参数

width 以像素为单位的请求的捕获宽度。默认值为 160。

height 以像素为单位的请求的捕获高度。默认值为 120。

fps 摄像机捕获数据所用的请求的速率，以每秒帧数为单位。默认值为 15。

favorSize 可选：一个布尔值，指定当摄像机不具有满足指定要求的本机模式时如何控制宽度、高度和帧速率。默认值为 true，这意味着支持保持捕获大小；使用此参数选择与 *width* 和 *height* 值最匹配的模式，即使这样做会由于降低了帧速率而对性能有不利影响。若要在对摄像机高度和宽度的影响最小的情况下最大限度地提高帧速率，则为 *favorSize* 参数传递 false。

返回

无。

说明

方法；将摄像机的捕获模式设置为最符合指定要求的本机模式。如果摄像机不具有匹配您传递的所有参数的本机模式，则 Flash 选择最接近于请求模式的合成的捕获模式。此操作可能涉及裁切图像和删除帧。

默认情况下，Flash 根据需要删除帧，以保持图像大小。若要将删除的帧的数目降至最低，即使这样做意味着减小图像的大小，为 *favorSize* 参数传递 *false*。

在选择本机模式时，Flash 尝试尽可能保持请求的纵横比。例如，如果您发出命令 *active_cam.setMode(400, 400, 30)*，并且摄像机上可用的最大宽度和高度值分别为 320 和 288，Flash 会将宽度和高度值都设置为 288；通过将这些属性设置为相同的值，Flash 保持您请求的 1:1 纵横比。

若要确定 Flash 在选择与您请求的值最匹配的模式后分配给这些属性的值，请使用 *Camera.width*、*Camera.height* 和 *Camera.fps*。

示例

下面的示例基于用户输入（如果用户单击该按钮）设置宽度、高度和 *fps*。示例中不包括可选参数 *favorSize*，因为默认值 *true* 将提供在不降低图片品质的情况下与用户的首选参数最接近的设置，尽管可能会影响 *fps*。然后用新设置更新用户界面。

```
on (press)
{
    // 根据用户输入设置宽度、高度和 fps。
    _root.myCam.setMode(txt_width, my_txt._height, txt_fps);

    // 使用新设置更新用户的文本字段。
    _root.txt_width = myCam.width;
    _root.txt_height = myCam.height;
    _root.txt_fps = myCam.fps;
}
```

另请参见

Camera.currentFps, *Camera.fps*, *Camera.height*, *Camera.width*

Camera.setMotionLevel()

可用性

Flash Player 6。

用法

```
active_cam.setMotionLevel(sensitivity [, timeout])
```

参数

sensitivity 一个数值，指定调用 *Camera.onActivity(true)* 所需的运动量。可接受值的范围是 0 到 100。默认值为 50。

timeout 一个可选的数值参数，指定必须经过多少没有任何活动的时间（按毫秒数计算），Flash 才会认为活动已停止并调用 *Camera.onActivity(false)* 事件处理函数。默认值为 2000（2 秒）。

返回

无。

说明

方法；指定调用 `Camera.onActivity(true)` 所需的运动量。或者，设置必须经过多少没有任何活动的时间（按毫秒数计算），Flash 才会认为运动已停止并调用 `Camera.onActivity(false)`。

注意：不论 `sensitivity` 参数的值为如何都可以显示视频。该参数只确定在何时以及哪些情况下调用 `Camera.onActivity`，而与实际上是捕获还是显示视频无关。

- 若要完全禁止摄像机检测运动，则为 `sensitivity` 传递值 100；`Camera.onActivity` 永远不被调用。（您可能只能将该值用作测试目的，例如暂时禁用在使用 `Camera.onActivity` 时发生的任何动作集。）
- 若要确定摄像机当前检测到的运动量，请使用 `Camera.activityLevel` 属性。

运动敏感度值直接对应于活动值。完全不运动为活动值 0。持续运动为活动值 100。当您未移动时，活动值小于运动敏感度值；当您正移动时，活动值会经常超过运动敏感度值。

此方法的用途与 `Microphone.setSilenceLevel()` 相同；这两种方法都用于指定应该在何时调用 `onActivity` 事件处理函数。但是这些方法对发布流具有非常不同的影响：

- `Microphone.setSilenceLevel()` 用于优化带宽。在认为音频流被静音时，不发送任何音频数据。所发送的是一个指示静音已启动的消息。
- `Camera.setMotionLevel()` 用于检测运动，它不影响带宽用量。即使视频流未检测到运动，仍将发送视频。

示例

下面的示例在视频活动开始或结束时将消息发送到“输出”面板。将值为 30 的运动敏感度值更改为更高或更低的数值，看一下不同的值是如何影响运动检测的。

```
// 假设名为“myVideoObject”的 Video 对象位于舞台上
c = Camera.get();
x = 0;
function motion(mode)
{
    trace(x + "：模式
    x++;
}
c.onActivity = function(mode) {motion(mode);};
c.setMotionLevel(30, 500);
myVideoObject.attachVideo(c);
```

另请参见

[Camera.activityLevel](#), [Camera.motionLevel](#), [Camera.motionTimeout](#), [Camera.onActivity](#)

Camera.setQuality()

可用性

Flash Player 6。

用法

```
active_cam.setQuality(bandwidth, frameQuality)
```

参数

bandwidth 一个整数，指定当前输出视频输入信号可以使用的最大带宽量，以每秒字节数为单位。若要指定 Flash 视频可以使用所需的任何数量的带宽来保持 *frameQuality* 的值，则为 *bandwidth* 传递 0。默认值为 16384。

frameQuality 一个指定所需的图片品质级别的整数，该图片品质级别由应用于每一视频帧的压缩量确定。可接受的值范围从 1（最低品质，最大压缩）到 100（最高品质，无压缩）。若要指定图片品质可以根据需要进行变化，以避免超出带宽，则为 *frameQuality* 传递 0。默认值为 0。

返回

无。

说明

方法；设置每秒带宽的最大量或者当前输出视频输入信号的所需图片品质。此方法通常只在您使用 Flash Communication Server 传输视频时适用。

使用此方法指定输出视频输入信号的哪一方面对于您的应用程序更重要：是带宽使用率还是图片品质。

- 若要指示带宽使用率更为重要，则为 *bandwidth* 传递一个值并且为 *frameQuality* 传递值 0。Flash 将在指定的带宽内以可能的最高品质传输视频。如有必要，Flash 将降低图片品质以避免超出指定的带宽。通常，随着运动的增加，品质将降低。
- 若要指示品质最重要，则为 *bandwidth* 传递值 0 并为 *frameQuality* 传递一个数值。Flash 将根据需要使用尽量多的带宽来保持指定的品质。如有必要，Flash 将降低帧速率以保持图片品质。通常，随着运动的增加，带宽的使用率也将增加。
- 若要指定带宽和品质同等重要，则为这两个参数都传递数值。Flash 将传输达到指定品质并且不超过指定带宽的视频。如有必要，Flash 将降低帧速率以在不超出指定的带宽的情况下保持图片品质。

示例

下面的示例举例说明如何使用此方法来控制带宽使用率和图片品质。

```
// 确保发送视频的速率不超过 8192（8K/秒），
active_cam.setQuality(8192.0);

// 确保发送视频的速率不超过 8192（8K/秒），
// 而品质最低为 50
active_cam.setQuality(8192.50);

// 确保最低品质为 50，无论采用多大的带宽
active_cam.setQuality(0,50);
```

另请参见

[Camera.bandwidth](#), [Camera.quality](#)

Camera.width

可用性

Flash Player 6。

用法

```
active_cam.width
```

说明

只读属性；以像素为单位的当前捕获宽度。若要为该属性设置所需值，请使用 [Camera.setMode\(\)](#)。

示例

下面的代码行用当前宽度值更新用户界面中的文本框。

```
myTextField.text=myCam.width;
```

另请参见有关 `Camera.setMode()` 的示例。

另请参见

`Camera.height`

case

可用性

Flash Player 4。

用法

```
case expression:statements
```

参数

expression 任何表达式。

statements 任何语句。

返回

无。

说明

语句；定义用于 `switch` 动作的条件。如果 `case` 关键字后的 *expression* 参数在使用全等 (`===`) 的情况下等于 `switch` 动作的 *expression* 参数，则执行 *statements* 参数中的语句。

如果在 `switch` 语句外部使用 `case` 动作，则将产生错误，脚本不能编译。

另请参见

`break`, `default`, `===` (全等), `switch`

chr

可用性

Flash Player 4。不鼓励在 Flash 5 中使用该函数，而推荐使用 `String.fromCharCode()`。

用法

```
chr(number)
```

参数

number ASCII 码数字。

返回

无。

说明

字符串函数；将 ASCII 码数字转换为字符。

示例

下面的示例将数字 65 转换为字母 A，并将其分配给变量 myVar。

```
myVar = chr(65);
```

另请参见

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

class

可用性

Flash Player 6。

用法

```
[dynamic] class className [ extends superClass ]  
    [ implements interfaceName [, interfaceName... ] ]  
{  
    // 此处是类定义  
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

className 类的全限定名。

superClass 可选；*className* 扩展（继承自）的类的名称。

interfaceName 可选；其 *className* 方法必须实现的接口的名称。

说明

语句；定义一个自定义类，通过该自定义类可以实例化共享您定义的方法和属性的对象。例如，如果您正开发一个发票跟踪系统，则可以创建一个发票类，它定义每一发票应具有的所有方法和属性。然后您可以使用 `new invoice()` 命令来创建发票对象。

该类的名称必须与包含该类的外部文件的名称相同。例如，如果您将一个类命名为 `Student`，则定义该类的文件必须被命名为 `Student.as`。

该类名称在声明该类的文件内必须是全限定的；即，该类名称必须反映存储它的目录。例如，要创建存储在 `myClasses/education/curriculum` 目录中的名为 `RequiredClass` 的类，您必须按如下所示在 `RequiredClass.as` 文件中声明该类：

```
class myClasses.education.curriculum.RequiredClass {  
}
```

因此，最好在开始创建类之前就计划好您的目录结构。否则，如果您在创建类文件之后决定移动它们，就将不得不修改类声明语句以反映其新位置。

您不能嵌套类定义；即，不能在一个类定义内定义其它类。

若要指示对象可在运行时添加和访问动态属性，应在类语句前面放置 `dynamic` 关键字。若要基于接口创建类，请使用 `implements` 关键字。若要创建一个类的子类，请使用 `extends` 关键字。（某一类只能扩展一个类，但可以实现多个接口。）您可以在一个语句中使用 `implements` 和 `extends`。

```
class C implements Interface_i, Interface_j // 可以
```

```
class C extends Class_d implements Interface_i, Interface_j    // 可以
class C extends Class_d, Class_e    // 不可以
```

有关更多信息，请参见第 141 页的“创建和使用类”。

示例

下面的示例创建一个名为 Plant 的类。其构造函数采用两个参数。

```
// 文件名 Plant.as
class Plant {
    // 定义属性名称和类型
    var leafType:String;
    var bloomSeason:String;
    // 以下行是构造函数
    // 因为它具有与类相同的名称
    function Plant (param_leafType:String, param_bloomSeason:String) {
        // 在创建新的 Plant 对象时向属性分配传递的值
        leafType = param_leafType;
        bloomSeason = param_bloomSeason;
    }
    // 创建返回属性值的方法，因为最佳实践
    // 建议直接引用一个类的属性
    function getLeafType():String {return leafType;};
    function getBloomSeason():String {return bloomSeason;};
}
```

在外部脚本文件或“动作”面板中，使用 new 运算符创建一个 Plant 对象。

```
var pineTree:Plant = new Plant("Evergreen","N/A");
// 确认参数已正确传递
trace(pineTree.getLeafType());
trace(pineTree.getBloomSeason());
```

另请参见

[dynamic](#), [extends](#), [implements](#), [import](#), [interface](#), [new](#), [Object.registerClass\(\)](#)

clearInterval()

可用性

Flash Player 6。

用法

```
clearInterval( intervalID )
```

参数

intervalID 从对 [setInterval\(\)](#) 的调用返回的对象。

返回

无。

说明

函数；清除对 [setInterval\(\)](#) 的调用。

示例

下面的示例首先设置一个间隔调用，然后将其清除：

```
function callback() {
    trace("interval called");
}
var intervalID;
intervalID = setInterval( callback, 1000 );

// 有时稍后
clearInterval( intervalID );
```

另请参见

[setInterval\(\)](#)

Color 类

可用性

Flash Player 5。

说明

通过 Color 类，您可以设置影片剪辑的 RGB 颜色值和颜色转换，并可以在设置后获取这些值。必须使用构造函数 `new Color()` 创建 Color 对象后，才可调用其方法。

Color 类的方法摘要

方法	说明
Color.getRGB()	返回由最后一次 <code>setRGB()</code> 调用设置的 RGB 数值。
Color.getTransform()	返回由最后一次 <code>setTransform()</code> 调用设置的转换信息。
Color.setRGB()	为 Color 对象设置 RGB 值的十六进制表示形式。
Color.setTransform()	为 Color 对象设置颜色转换。

Color 类的构造函数

可用性

Flash Player 5。

用法

```
new Color(target)
```

参数

target 影片剪辑的实例名称。

返回

无。

说明

构造函数；为由 *target* 参数指定的影片剪辑创建 Color 对象。然后可使用该 Color 对象的方法来更改整个目标影片剪辑的颜色。

示例

以下示例为影片剪辑 `my_mc` 创建一个名为 `my_color` 的 Color 对象并设置其 RGB 值：

```
my_color = new Color(my_mc);  
my_color.setRGB(0xff9933);
```

Color.getRGB()

可用性

Flash Player 5。

用法

```
my_color.getRGB()
```

参数

无。

返回

表示指定颜色的 RGB 数字值的数字。

说明

方法；返回由最后一次 `setRGB()` 调用设置的数字值。

示例

下面的代码获取 Color 对象 `my_color` 的 RGB 值，将其转换为十六进制字符串，并将其分配给 `value` 变量。

```
value = my_color.getRGB().toString(16);
```

另请参见

[Color.setRGB\(\)](#)

Color.getTransform()

可用性

Flash Player 5。

用法

```
my_color.getTransform()
```

参数

无。

返回

一个对象，其属性包含指定颜色的当前偏移量和百分比值。

说明

方法；返回由最后一次 `Color.setTransform()` 调用设置的转换值。

另请参见

[Color.setTransform\(\)](#)

Color.setRGB()

可用性

Flash Player 5。

用法

```
my_color.setRGB(0xRRGGBB)
```

参数

`0xRRGGBB` 要设置的十六进制或 RGB 颜色。对于 *RR*、*GG* 和 *BB*，每种代码由两个十六进制数字组成，这些数字指定每种颜色成分的偏移量。0x 告知动作脚本编译器该数字是十六进制数值。

说明

方法；指定 `Color` 对象的 RGB 颜色。调用此方法将覆盖任何先前的 `Color.setTransform()` 设置。

返回

无。

示例

此示例设置影片剪辑 `my_mc` 的 RGB 颜色值。若要查看此代码的工作情况，请将影片剪辑置于实例名称为 `my_mc` 的舞台上。然后将下面的代码置于主时间轴中的“第 1 帧”上，再选择“控制”>“测试影片”。

```
my_color = new Color(my_mc);  
my_color.setRGB(0x993366);
```

另请参见

[Color.setTransform\(\)](#)

Color.setTransform()

可用性

Flash Player 5。

用法

```
my_color.setTransform(colorTransformObject)
```

参数

`colorTransformObject` 使用 `new Object` 构造函数创建的对象。`Object` 类的这一实例必须具有以下指定颜色转换值的属性：`ra`、`rb`、`ga`、`gb`、`ba`、`bb`、`aa`、`ab`。这些属性在下面进行了解释。

返回

无。

说明

方法；为 Color 对象设置颜色转换信息。*colorTransformObject* 参数是通过 *new Object* 构造函数创建的通用对象。它具有指定颜色的红、绿、蓝和 alpha（透明度）成分百分比和偏移量值的参数，以 *0xRRGGBBAA* 的格式输入。

颜色转换对象的参数与“高级效果”对话框中的设置相对应，定义如下：

- *ra* 是红色成分的百分比（-100 到 100）。
- *rb* 是红色成分的偏移量（-255 到 255）。
- *ga* 是绿色成分的百分比（-100 到 100）。
- *gb* 是绿色成分的偏移量（-255 到 255）。
- *ba* 是蓝色成分的百分比（-100 到 100）。
- *bb* 是蓝色成分的偏移量（-255 到 255）。
- *aa* 是 alpha 的百分比（-100 到 100）。
- *ab* 是 alpha 的偏移量（-255 到 255）。

您可按如下形式创建 *colorTransformObject* 参数：

```
myColorTransform = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

您也可使用以下语法来创建 *colorTransformObject* 参数：

```
myColorTransform = { ra:?50? rb:?244? ga:?40? gb:?112? ba:?12? bb:?90? aa:?40?
ab:?70?
```

示例

此示例为目标 SWF 文件创建新 Color 对象，使用上面定义的属性创建名为 *myColorTransform* 的通用对象，然后使用 *setTransform()* 方法将 *colorTransformObject* 传递给一个 Color 对象。若要在 Flash (FLA) 文档中使用此代码，请将其放在主时间轴中的第 1 帧上，然后将影片剪辑置于实例名称为 *my_mc* 的舞台上，如以下代码所示：

```
// 为目标 my_mc 创建名为 my_color 的 Color 对象
my_color = new Color(my_mc);
// 使用通用 Object 对象创建名为 myColorTransform
// 的 Color Transform 对象
myColorTransform = new Object();
// 设置 myColorTransform 的值
myColorTransform = { ra:'50', rb:'244', ga:'40', gb:'112', ba:'12', bb:'90',
aa:'40', ab:'70'};
// 将 Color Transform 对象与 Color 对象相关联
// 为 my_mc 创建
my_color.setTransform(myColorTransform);
```

ContextMenu 类

可用性

Flash Player 7。

说明

ContextMenu 类提供对 Flash Player 上下文菜单项的运行时控制，当用户在 Flash Player 中右击 (Windows) 或按住 Control 键单击 (Macintosh) 时，上下文菜单将出现。您可以使用 ContextMenu 类的方法和属性添加自定义菜单项，控制内置菜单项的显示（例如，“放大”和“打印”），或者创建菜单的副本。

您可以将 ContextMenu 对象附加到特定的按钮、影片剪辑或文本字段，也可以附加到整个影片级别。这可以使用 Button、MovieClip 或 TextField 类的 menu 属性来实现。有关 menu 属性的更多信息，请参见 Button.menu、MovieClip.menu 和 TextField.menu。

若要向 ContextMenu 对象添加新的菜单项，请创建一个 ContextMenuItem 对象，然后将该对象添加到 ContextMenu.customItems 数组。有关创建上下文菜单项的更多信息，请参见 ContextMenuItem 类条目。

Flash Player 具有三种类型的上下文菜单：标准菜单（当您在 Flash Player 中右击时出现）、编辑菜单（当您在可选择或可编辑的文本字段上右击时出现）和错误菜单（当 SWF 文件未能加载到 Flash Player 中时出现）。只有标准菜单和编辑菜单才能使用 ContextMenu 类进行修改。

自定义菜单项始终出现在 Flash Player 上下文菜单的顶部，并位于所有可见内置菜单项之上；内置菜单项和自定义菜单项之间由一个分隔条加以分隔。一个上下文菜单包含的自定义菜单项不能超过 15 个。

必须在使用构造函数 new ContextMenu() 创建 ContextMenu 对象之后，才能调用它的方法。

ContextMenu 类的方法概要

方法	说明
<code>ContextMenu.copy()</code>	返回指定 ContextMenu 对象的副本。
<code>ContextMenu.hideBuiltInItems()</code>	在 Flash Player 上下文菜单中隐藏大多数内置项。

ContextMenu 类的属性概要

属性	说明
<code>ContextMenu.builtInItems</code>	其成员与内置上下文菜单项相对应的对象。
<code>ContextMenu.customItems</code>	默认情况下未定义的数组，它包含 ContextMenuItem 对象。

ContextMenu 类的事件处理函数概要

属性	说明
<code>ContextMenu.onSelect</code>	在显示菜单之前调用。

ContextMenu 类的构造函数

可用性

Flash Player 7。

用法

```
new ContextMenu ([callbackFunction])
```

参数

callbackFunction 对函数的引用，该函数在用户右击或按住 **Control** 键单击时，但在菜单显示之前被调用。此参数是可选的。

返回

无。

说明

构造函数；创建新的 **ContextMenu** 对象。或者，您可以在创建该对象时指定事件处理函数的标识符。指定的函数在用户调用上下文菜单时，但在菜单实际显示之前被调用。在根据应用程序状态或者根据用户右击或按住 **Control** 键单击的对象类型（影片剪辑、文本字段或按钮）自定义菜单内容时，此函数非常有用。（有关创建事件处理函数的示例，请参见 [ContextMenu.onSelect](#)。）

示例

以下示例隐藏上下文菜单中的所有内置对象。（但是，由于不能禁用“设置”和“关于”菜单项，所以它们仍会出现。）

```
var newMenu = new ContextMenu();
newMenu.hideBuiltInItems();
_root.menu = newMenu;
```

在此示例中，指定的事件处理函数 `menuHandler` 根据名为 `showItem` 的布尔型变量的值启用或禁用自定义菜单项（使用 `ContextMenu.customItems` 数组）。如果为 `false`，则禁用自定义菜单项；否则，将其启用。

```
var showItem = false; // 将其更改为 true 以查看其效果
my_cm = new ContextMenu(menuHandler);
my_cm.customItems.push(new ContextMenuItem("Hello", itemHandler));
function menuHandler(obj, menuObj) {
    if (showItem == false) {
        menuObj.customItems[0].enabled = false;
    } else {
        menuObj.customItems[0].enabled = true;
    }
}
function itemHandler(obj, item) {
}
_root.menu = my_cm;
```

另请参见

[Button.menu](#), [ContextMenu.onSelect](#), [ContextMenu.customItems](#),
[ContextMenu.hideBuiltInItems\(\)](#), [MovieClip.menu](#), [TextField.menu](#)

ContextMenu.builtInItems

可用性

Flash Player 7。

用法

my_cm.builtInItems

说明

属性；具有以下布尔型属性的对象：`save`、`zoom`、`quality`、`play`、`loop`、`rewind`、`forward_back` 和 `print`。如果将这些变量设置为 `false`，则会删除指定 `ContextMenu` 对象中的相应菜单项。这些属性是可枚举的，并在默认情况下设置为 `true`。

示例

在此示例中，对于附加到 SWF 文件的根时间轴的 `ContextMenu` 对象 `my_cm`，内置“品质”和“打印”菜单项被禁用。

```
var my_cm = new ContextMenu();
my_cm.builtInItems.quality=false;
my_cm.builtInItems.print=false;
_root.menu = my_cm;
```

在下一个示例中，`for..in` 循环枚举通过 `ContextMenu` 对象 `my_cm` 的内置菜单项的所有名称和值。

```
my_cm = new ContextMenu();
for(eachProp in my_cm.builtInItems) {
    var propName = eachProp;
    var propValue = my_cm.builtInItems[propName];
    trace(propName + " : " + propValue);
}
```

ContextMenu.copy()

可用性

Flash Player 7。

用法

my_cm.copy()

参数

无。

返回

`ContextMenu` 对象。

说明

方法；创建指定 `ContextMenu` 对象的副本。该副本继承初始菜单对象的所有属性。

示例

此示例创建其内置菜单项已隐藏的 `ContextMenu` 对象 `my_cm` 的副本，并添加带文本“保存...”的菜单项。然后，它创建 `my_cm` 的副本，并将其分配给变量 `clone_cm`，该变量继承初始菜单的所有属性。

```
my_cm = new ContextMenu();
my_cm.hideBuiltInItems();
my_cm.customItems.push(new ContextMenuItem("Save...", saveHandler);
function saveHandler (obj, menuItem) {
    saveDocument(); // 自定义函数（未显示）
}
clone_cm = my_cm.copy();
```

ContextMenu.customItems

可用性

Flash Player 7。

用法

```
my_cm.customItems
```

说明

属性； `ContextMenuItem` 对象的数组。数组中的每个对象表示您已经定义的上下文菜单项。使用此属性可添加、删除或修改这些自定义菜单项。

若要添加新的菜单项，请首先创建一个新的 `ContextMenuItem` 对象，然后将其添加到 `menu_mc.customItems` 数组（例如，使用 `Array.push()`）。有关创建新菜单项的更多信息，请参见 [ContextMenuItem](#) 类条目。

示例

以下示例新建一个名为 `menuItem_cm` 的自定义菜单项（其标题为“发送电子邮件”）和一个名为 `emailHandler`（未显示）的回调处理函数。然后，使用 `customItems` 数组将新的菜单项添加到 `ContextMenu` 对象 `my_cm`。最后，将新菜单附加到名为 `email_mc` 的影片剪辑。

```
var my_cm = new ContextMenu();
var menuItem_cm = new ContextMenuItem("Send e-mail", emailHandler);
my_cm.customItems.push(menuItem_cm);
email_mc.menu = my_cm;
```

另请参见

`Button.menu`, [ContextMenu](#) 类, `MovieClip.menu`, `TextField.menu`

ContextMenu.hideBuiltInItems()

可用性

Flash Player 7。

用法

```
my_cm.hideBuiltInItems()
```

参数

无。

返回

无。

说明

方法；隐藏指定 ContextMenu 对象中的所有内置菜单项（“设置”除外）。如果 Flash 调试播放器正在运行，则显示“调试”菜单项，但如果 SWF 文件未启用远程调试功能，则该菜单项会变暗。

此方法仅隐藏标准上下文菜单中显示的菜单项；它不影响编辑菜单或错误菜单中显示的菜单项。有关不同菜单类型的更多信息，请参见 [ContextMenu](#) 类条目。

此方法通过将 *my_cm.builtInItems* 的所有布尔型成员设置为 *false* 来实现其功能。您可以有选择地显示内置菜单项，方法是将其在 *my_cm.builtInItems* 中的相应成员设置为 *true*（如下示例所示）。

示例

以下示例创建其内置菜单项已隐藏（“打印”除外）的新 ContextMenu 对象 *my_cm*。然后将该菜单对象附加到根时间轴。

```
my_cm = new ContextMenu();  
my_cm.hideBuiltInItems();  
my_cm.builtInItems.print = true;  
_root.menu = my_cm;
```

ContextMenu.onSelect

可用性

Flash Player 7。

用法

```
my_cm.onSelect = function (item:Object, item_menu:ContextMenu) {  
    // 此处是您的代码  
}
```

参数

item 对特定对象（影片剪辑、按钮或可选择的文本字段）的引用，当调用 Flash Player 上下文菜单时，该对象位于鼠标指针下，并且该对象的 *menu* 属性设置为有效的 *ContextMenu* 对象。

item_menu 对 *ContextMenu* 对象的引用，该对象分配到 *object* 的 *menu* 属性。

返回

无。

说明

事件处理函数；在用户调用 Flash Player 上下文菜单时，但在该菜单实际显示之前调用。这使您可以根据当前的应用程序状态自定义上下文菜单的内容。

您也可以在构造新的 *ContextMenu* 对象时指定 *ContextMenu* 对象的回调处理函数。有关更多信息，请参见 [ContextMenu](#) 类条目。

示例

以下示例确定对哪种类型的对象调用了上下文菜单。

```
my_cm = new ContextMenu();  
menuHandler = function (obj:Object, menu:ContextMenu) {  
    if(obj instanceof MovieClip) {  
        trace("Movie clip:" + obj);  
    }  
    if(obj instanceof TextField) {  
        trace("Text field:" + obj);  
    }  
    if(obj instanceof Button) {  
        trace("Button:" + obj);  
    }  
}  
my_cm.onSelect = menuHandler;
```

ContextMenuItem 类

可用性

Flash Player 7。

说明

使用 ContextMenuItem 类可创建在 Flash Player 上下文菜单中显示的自定义菜单项。每个 ContextMenuItem 对象均具有在上下文菜单中显示的标题（文本）以及在选择菜单项时调用的回调处理函数（函数）。若要向上下文菜单添加新的上下文菜单项，请将其添加到 ContextMenu 对象的 customItems 数组。

您可以启用或禁用特定菜单项，显示或隐藏菜单项，或者更改与菜单项关联的标题或回调处理函数。

自定义菜单项出现在上下文菜单的顶部，并位于所有内置菜单项之上。自定义菜单项与内置菜单项始终由分隔条隔开。向 Flash Player 上下文菜单添加的自定义菜单项不得超过 15 个。每个菜单项必须至少包含一个可见字符，控制字符、换行符和其它空白字符将被忽略。所有菜单项的长度不得超过 100 个字符。如果菜单项与任何内置菜单项或其它自定义菜单项相同，则无论匹配菜单项是否可见，均会忽略该菜单项。对菜单项进行比较时将忽略大小写、标点和空格。

自定义菜单项中不能出现以下字词：Macromedia、Flash Player、或 设置。

ContextMenuItem 类的方法概要

方法	说明
<code>ContextMenuItem.copy()</code>	返回指定 ContextMenuItem 对象的副本。

ContextMenuItem 类的属性概要

属性	说明
<code>ContextMenuItem.caption</code>	指定菜单项中显示的文本。
<code>ContextMenuItem.enabled</code>	指定是启用还是禁用菜单项。
<code>ContextMenuItem.separatorBefore</code>	指定菜单项之上是否应该出现分隔条。
<code>ContextMenuItem.visible</code>	指定菜单项是否可见。

ContextMenuItem 类的事件处理函数概要

事件处理函数	说明
<code>ContextMenuItem.onSelect</code>	在选择菜单项时调用。

ContextMenuItem 类的构造函数

可用性

Flash Player 7。

用法

```
new ContextMenuItem(caption, callbackFunction, [ separatorBefore, [ enabled,  
[ visible ] ] ] )
```

参数

caption 指定与菜单项关联的文本的字符串。

callbackFunction 您定义的函数，它在选择菜单项时被调用。

separatorBefore 布尔值，指示分隔条是否应该出现在上下文菜单中的特定菜单项之上。此参数是可选的；其默认值为 `false`。

enabled 布尔值，指示是启用还是禁用上下文菜单中的特定菜单项。此参数是可选的；其默认值为 `true`。

visible 布尔值，指示菜单项是否可见。此参数是可选的；其默认值为 `true`。

返回

无。

说明

构造函数；创建可添加到 `ContextMenu.customItems` 数组的新 `ContextMenuItem` 对象。

示例

此示例将“开始”和“停止”菜单项（由分隔条隔开）添加到 `ContextMenu` 对象 `my_cm`。当从上下文菜单中选择“开始”时，将调用 `startHandler()` 函数；当选择“停止”时，将调用 `stopHandler()`。`ContextMenu` 对象会应用到根时间轴。

```
my_cm = new ContextMenu();  
my_cm.customItems.push(new ContextMenuItem("Start", startHandler));  
my_cm.customItems.push(new ContextMenuItem("Stop", stopHandler, true));  
function stopHandler(obj, item) {  
    trace("Stopping...");  
}  
function startHandler(obj, item) {  
    trace("Starting...");  
}  
_root.menu = my_cm;
```

ContextMenuItem.caption

可用性

Flash Player 7。

用法

```
menuItem_cmi.caption
```

说明

属性；指定上下文菜单中显示的菜单项标题（文本）的字符串。

示例

此示例显示“输出”面板中选定菜单项（“暂停游戏”）的标题。

```
my_cm = new ContextMenu();
menuItem_cmi = new ContextMenuItem("Pause Game", onPause);
my_cm.customItems.
function onPause(obj, menuItem) {
    trace("You chose:" + menuItem.caption);
}
```

ContextMenu.copy()

可用性

Flash Player 7。

用法

```
menuItem_cmi.copy();
```

返回

ContextMenuItem 对象。

说明

方法；创建并返回指定 ContextMenuItem 对象的副本。该副本包含初始对象的所有属性。

示例

此示例新建名为 original_cmi 的 ContextMenuItem 对象，其标题文本为“暂停”，回调处理函数设置为函数 onPause。然后，此示例创建 ContextMenuItem 对象的副本，并将其分配给变量 copy_cmi。

```
original_cmi = new ContextMenuItem("Pause", onPause);
function onPause(obj, menu) {
    _root.stop();
}
original_cmi.visible = false;
copy_cmi = orig_cmi.copy();
```

ContextMenu.enabled

可用性

Flash Player 7。

用法

```
menuItem_cmi.enabled
```

说明

属性；指示是启用还是禁用指定菜单项的布尔值。默认情况下，此属性为 true。

示例

以下示例创建新的上下文菜单项，然后禁用该菜单项。

```
var saveMenuItem = new ContextMenuItem("Save...", doSave);
saveMenuItem.enabled = false;
```


ContextMenuItem.onSelect

可用性

Flash Player 7。

用法

```
menuItem_cmi.onSelect = function (obj, menuItem) {  
    // 此处是您的语句  
}
```

参数

obj 对用户右击或按住 Control 键单击的影片剪辑（或时间轴）、按钮或可选择（可编辑）文本字段的引用。

menuItem 对选定 ContextMenuItem 对象的引用。

返回

无。

说明

事件处理函数；当从 Flash Player 上下文菜单中选择指定菜单项时调用。指定的回调处理函数会接收两个参数：*obj*，对在用户调用 Flash Player 上下文菜单时位于鼠标下的对象的调用；*menuItem*，对表示选定菜单项的 ContextMenuItem 对象的调用。

示例

以下示例将一个函数分配给名为 start_cmi 的 ContextMenuItem 对象的 onSelect 处理函数。该函数显示选定菜单项的标题。

```
start_cmi.onSelect = function (obj, item) {  
    trace("You choose:" + item.caption);  
}
```

另请参见

[ContextMenu.onSelect](#)

ContextMenuItem.separatorBefore

可用性

Flash Player 7。

用法

```
menuItem_cmi.separatorBefore
```

说明

属性；指定分隔条是否应该在指定菜单项之上出现的布尔值。默认情况下，此属性为 false。

注意：任何自定义菜单项和内置菜单项之间始终会出现分隔条。

示例

此示例创建三个菜单项，其标签为“打开”、“保存”和“打印”。“保存”和“打印”菜单项由分隔条隔开。然后，将这些菜单项添加到 ContextMenu 对象的 customItems 数组。最后，该菜单被附加到 SWF 文件的根时间轴上。

```

my_cm = new ContextMenu();
open_cmi = new ContextMenuItem("Open", itemHandler);
save_cmi = new ContextMenuItem("Save", itemHandler);
print_cmi = new ContextMenuItem("Print", itemHandler);
print_cmi.separatorBefore = true;
my_cm.customItems.push(open_cmi, save_cmi, print_cmi);
function itemHandler(obj, menuItem) {
    trace("You chose:" + menuItem.caption);
};
_root.menu = my_cm;

```

另请参见

[ContextMenu.onSelect](#)

ContextMenuItem.visible

可用性

Flash Player 7。

用法

menuItem_cmi.visible

说明

属性；指示当显示 Flash Player 上下文菜单时指定菜单项是否可见的布尔值。默认情况下，此属性为 `true`。

continue

可用性

Flash Player 4。

用法

`continue`

参数

无。

返回

无。

说明

语句；出现在几种类型的循环语句中；它在每种类型的循环中的行为方式各不相同。

在 `while` 循环中，`continue` 可使 Flash 解释程序跳过循环体的其余部分，并转到循环的顶端（在该处进行条件测试）。

在 `do while` 循环中，`continue` 可使 Flash 解释程序跳过循环体的其余部分，并转到循环的底端（在该处进行条件测试）。

在 `for` 循环中，`continue` 可使 Flash 解释程序跳过循环体的其余部分，并转而计算 `for` 循环的后表达式（`post-expression`）。

在 `for..in` 循环中，`continue` 可使 Flash 解释程序跳过循环体的其余部分，并跳回循环的顶端（在这里处理下一个枚举值）。

另请参见

`do while`, `for`, `for..in`, `while`

CustomActions 类

可用性

Flash Player 6。

说明

`CustomActions` 类的方法使得在 Flash 创作工具中播放的 SWF 文件可以管理任何用该创作工具注册的自定义动作。SWF 文件可安装和卸载自定义动作，获取自定义动作的 XML 定义，以及获取已注册自定义动作的列表。

您可使用这些方法来生成属于 Flash 创作工具扩展功能的 SWF 文件。例如，这样的扩展功能可使用“Flash 应用程序协议”定位 UDDI 储备库，并将 Web 服务下载到“动作”工具箱。

CustomActions 类的方法摘要

方法	说明
<code>CustomActions.get()</code>	读取自定义动作 XML 定义文件的内容。
<code>CustomActions.install()</code>	安装新的自定义动作 XML 定义文件。
<code>CustomActions.list()</code>	返回所有已注册自定义动作的列表。
<code>CustomActions.uninstall()</code>	删除自定义动作 XML 定义文件。

CustomActions.get()

可用性

Flash Player 6。

用法

`CustomActions.get(customActionsName)`

参数

customActionsName 要获取的自定义动作定义的名称。

返回

如果找到自定义动作的 XML 定义，则返回字符串，否则，返回 `undefined`。

说明

方法；读取名为 *customActionsName* 的自定义动作 XML 定义文件的内容。

定义文件的名称必须是简单文件名，没有 `.xml` 文件扩展名，也没有任何目录分隔符（`“:”`、`“/”` 或 `“\”`）。

如果找不到由 *customActionsName* 指定的定义文件，则返回一个 `undefined` 值。如果找到了由 *customActionsName* 参数指定的自定义动作 XML 定义，则该定义将被完整地读取，并以字符串形式返回。

CustomActions.install()

可用性

Flash Player 6。

用法

```
CustomActions.install(customActionsName, customXMLDefinition)
```

参数

customActionsName 要安装的自定义动作定义的名称。

customXMLDefinition 要安装的 XML 定义的文本。

返回

如果安装过程中发生错误，则返回布尔值 `false`；否则，返回布尔值 `true`，指示该自定义动作已成功安装。

说明

方法；安装由 *customActionsName* 参数指示的新自定义动作 XML 定义文件。该文件的内容由字符串 *customXMLDefinition* 指定。

定义文件的名称必须是简单文件名，没有 `.xml` 文件扩展名，也没有任何目录分隔符（`“:”`、`“/”` 或 `“\”`）。

如果已存在名为 *customActionsName* 的自定义动作文件，则该文件将被覆盖。

如果调用此方法时不存在目录 `Configuration/ActionsPanel/CustomActions`，则将创建该目录。

CustomActions.list()

可用性

Flash Player 6。

用法

```
CustomActions.list()
```

参数

无。

返回

一个数组。

说明

方法；返回一个 Array 对象，该对象包含 Flash 创作工具中所有已注册自定义动作的名称。该数组的元素均是简单名称，没有 .xml 文件扩展名，并且没有任何目录分隔符（例如，“:”、“/”或“\”）。如果没有已注册的自定义动作，则 list() 将返回一个长度为零的数组。如果发生错误，list() 将返回值 undefined。

CustomActions.uninstall()

可用性

Flash Player 6。

用法

```
CustomActions.uninstall(customActionsName)
```

参数

customActionsName 要卸载的自定义动作定义的名称。

返回

如果没有找到名为 *customActionsName* 的任何自定义动作，则返回布尔值 false。如果自定义动作被成功删除，则返回值 true。

说明

方法；删除名为 *customActionsName* 的自定义动作 XML 定义文件。

定义文件的名称必须是简单文件名，没有 .xml 文件扩展名，也没有任何目录分隔符（“:”、“/”或“\”）。

Date 类

可用性

Flash Player 5。

说明

Date 类用于获取相对于通用时间（格林尼治平均时，现在叫做通用时间或 UTC）或相对于运行 Flash Player 的操作系统的日期和时间值。Date 类的方法不是静态的，但仅应用于调用方法时指定的单个 Date 对象。Date.UTC() 方法属于异常，它是一个静态方法。

Date 类以不同的方式处理夏时制，具体方式取决于操作系统和 Flash Player 的版本。在下面的操作系统中，Flash Player 6 及更高版本以这些方式处理夏时制：

- Windows Date 对象为夏时制自动调整其输出。Date 对象检测在当前区域设置内是否应用了夏时制，如果是，它将检测标准的夏时制转换日期和时间。然而，当前有效的转换日期会应用到以前和将来的日期，所以如果该区域设置具有不同的转换日期，则对于以前的日期，夏时制偏差的计算结果可能会不正确。
- Mac OS X Date 对象为夏时制自动调整其输出。Mac OS X 中的时区信息数据库用于确定现在或以前的任何日期或时间是否应该应用夏时制偏差。

在下面的操作系统中，Flash Player 5 处理夏时制方式如下所示：

- Windows 始终采用美国的夏时制规则，在欧洲和采用夏时制但转换时间与美国不同的其它区域中，这将造成不正确的转换结果。Flash 能够正确检测出在当前区域设置中是否应用了 DST。

若要调用 Date 类的方法，必须按照本节后面部分所述使用 Date 类的构造函数创建一个 Date 对象。

Date 类的方法摘要

方法	说明
Date.getDate()	按照本地时间返回某天是当月的第几天。
Date.getDay()	按照本地时间返回某天是周几。
Date.getFullYear()	按照本地时间返回 4 位数字的年份数。
Date.getHours()	按照本地时间返回小时值。
Date.getMilliseconds()	按照本地时间返回毫秒值。
Date.getMinutes()	按照本地时间返回分钟值。
Date.getMonth()	按照本地时间返回月份数。
Date.getSeconds()	按照本地时间返回秒数。
Date.getTime()	返回自 1970 年 1 月 1 日午夜（通用时间）以来的毫秒数。
Date.getTimezoneOffset()	以分钟为单位，返回计算机的本地时间和通用时间的差值。
Date.getUTCDate()	按照通用时间返回某天（日期）是当月的第几天。
Date.getUTCDay()	按照通用时间返回某天是周几。
Date.getUTCFullYear()	按照通用时间返回 4 位数字的年份数。
Date.getUTCHours()	按照通用时间返回小时值。
Date.getUTCMilliseconds()	按照通用时间返回毫秒值。
Date.getUTCMinutes()	按照通用时间返回分钟值。
Date.getUTCMonth()	按照通用时间返回月份数。
Date.getUTCSeconds()	按照通用时间返回秒数。
Date.getYear()	按照本地时间返回年份数。
Date.setDate()	按照本地时间设置某天是当月的第几天。返回以毫秒为单位的新时间。
Date.setFullYear()	按照本地时间设置完整的年份数。返回以毫秒为单位的新时间。
Date.setHours()	按照本地时间设置小时值。返回以毫秒为单位的新时间。
Date.setMilliseconds()	按照本地时间设置毫秒值。返回以毫秒为单位的新时间。
Date.setMinutes()	按照本地时间设置分钟值。返回以毫秒为单位的新时间。
Date.setMonth()	按照本地时间设置月份数。返回以毫秒为单位的新时间。
Date.setSeconds()	按照本地时间设置秒数。返回以毫秒为单位的新时间。

方法	说明
<code>Date.setTime()</code>	以毫秒为单位设置日期。返回以毫秒为单位的新时间。
<code>Date.setUTCDate()</code>	按照通用时间设置日期。返回以毫秒为单位的新时间。
<code>Date.setUTCFullYear()</code>	按照通用时间设置年份数。返回以毫秒为单位的新时间。
<code>Date.setUTCHours()</code>	按照通用时间设置小时值。返回以毫秒为单位的新时间。
<code>Date.setUTCMilliseconds()</code>	按照通用时间设置毫秒值。返回以毫秒为单位的新时间。
<code>Date.setUTCMinutes()</code>	按照通用时间设置分钟值。返回以毫秒为单位的新时间。
<code>Date.setUTCMonth()</code>	按照通用时间设置月份数。返回以毫秒为单位的新时间。
<code>Date.setUTCSeconds()</code>	按照通用时间设置秒数。返回以毫秒为单位的新时间。
<code>Date.setYear()</code>	按照本地时间设置年份数。
<code>Date.toString()</code>	返回一个表示存储在指定 <code>Date</code> 对象中的日期和时间的字符串值。
<code>Date.UTC()</code>	返回 1970 年 1 月 1 日午夜（通用时间）和指定时间之间的毫秒数。

Date 类的构造函数

可用性

Flash Player 5。

用法

```
new Date()  
new Date(year, month [, date [, hour [, minute [, second [, millisecond ]]]])
```

参数

year 一个 0 至 99 之间的值，表示 1900 年至 1999 年；如果年份不在上述范围内，则必须指定表示年份的所有 4 位数字。

month 从 0（一月）到 11（十二月）之间的整数。

date 从 1 到 31 之间的整数。此参数是可选的。

hour 从 0（午夜）到 23（深夜 11 点）之间的整数。

minute 从 0 到 59 之间的整数。此参数是可选的。

second 从 0 到 59 之间的整数。此参数是可选的。

millisecond 从 0 到 999 之间的整数。此参数是可选的。

返回

无。

说明

对象；构造一个新的 `Date` 对象，该对象保存当前日期和时间或指定的日期。

示例

下面的示例获取当前日期和时间。

```
now_date = new Date();
```

以下示例为 Gary 的生日（1974 年 8 月 12 日）创建一个新的 Date 对象。（由于月份参数从零开始，所以此示例使用 7 而不是 8 作为月份。）

```
garyBirthday_date = new Date (74, 7, 12);
```

以下示例创建一个新的 Date 对象，连接 `Date.getMonth()`、`Date.getDate()` 和 `Date.getFullYear()` 的返回值，并在变量 `date_str` 所指定的文本字段中显示这些值。

```
today_date = new Date();  
date_str = ((today_date.getMonth() + 1) + "/" + today_date.getDate() + "/" +  
    today_date.getFullYear());
```

Date.getDate()

可用性

Flash Player 5。

用法

```
my_date.getDate()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中表示日期的值（一个 1 至 31 之间的整数）。本地时间由运行 Flash Player 的操作系统确定。

Date.getDay()

可用性

Flash Player 5。

用法

```
my_date.getDay()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中表示周几的值（0 代表星期日，1 代表星期一，依此类推）。本地时间由运行 Flash Player 的操作系统确定。

Date.getFullYear()

可用性

Flash Player 5。

用法

```
my_date.getFullYear()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中的完整年份数（一个 4 位数，例如 2000）。本地时间由运行 Flash Player 的操作系统确定。

示例

下面的示例使用构造函数创建一个新的 Date 对象，并将 `getFullYear()` 方法返回的值发送到“输出”面板：

```
my_date = new Date();  
trace(my_date.getFullYear());
```

Date.getHours()

可用性

Flash Player 5。

用法

```
my_date.getHours()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象的小时值（0 至 23 之间的整数）。本地时间由运行 Flash Player 的操作系统确定。

Date.getMilliseconds()

可用性

Flash Player 5。

用法

```
my_date.getMilliseconds()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中的毫秒值（一个 0 至 999 之间的整数）。本地时间由运行 Flash Player 的操作系统确定。

Date.getMinutes()

可用性

Flash Player 5。

用法

```
my_date.getMinutes()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中的分钟数（0 至 59 之间的整数）。本地时间由运行 Flash Player 的操作系统确定。

Date.getMonth()

可用性

Flash Player 5。

用法

```
my_date.getMonth()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中的月份数（0 代表一月，1 代表二月，依此类推）。本地时间由运行 Flash Player 的操作系统确定。

Date.getSeconds()

可用性

Flash Player 5。

用法

```
my_date.getSeconds()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中的秒钟数（0 至 59 之间的整数）。本地时间由运行 Flash Player 的操作系统确定。

Date.getTime()

可用性

Flash Player 5。

用法

```
my_date.getTime()
```

参数

无。

返回

一个整数。

说明

方法；返回指定 Date 对象自 1970 年 1 月 1 日午夜（通用时间）以来的毫秒数。当比较两个或更多个 Date 对象时，使用这种方法表示特定实时时刻。

Date.getTimezoneOffset()

可用性

Flash Player 5。

用法

```
my_date.getTimezoneOffset()
```

参数

无。

返回

一个整数。

说明

方法；以分钟为单位，返回计算机的本地时间和通用时间之间的差值。

示例

下面的示例返回 San Francisco 本地夏时制时间和通用时间之间的差值。只有当 Date 对象中定义的日期在夏时制期间到达时，才考虑夏时制因素，将其计入返回结果。

```
trace(new Date().getTimezoneOffset());  
  
// “输出” 面板中显示 420  
// (7 小时 * 60 分钟 / 小时 = 420 分钟)  
// 此示例为太平洋夏令时 (PDT, GMT-0700)。  
// 根据区域设置和一年中时间的不同，结果将有所不同。
```

Date.getUTCDate()

可用性

Flash Player 5。

用法

```
my_date.getUTCDate()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象中表示某天是当月第几天的值（1 至 31 之间的整数）。

Date.getUTCDay()

可用性

Flash Player 5。

用法

```
my_date.getUTCDay()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象中表示周几的值（0 代表星期日，1 代表星期一，依此类推）。

Date.getUTCFullYear()

可用性

Flash Player 5。

用法

```
my_date.getUTCFullYear()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象中用 4 位数字表示的年份数。

Date.getUTCHours()

可用性

Flash Player 5。

用法

```
my_date.getUTCHours()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象的小时值。

Date.getUTCMilliseconds()

可用性

Flash Player 5。

用法

```
my_date.getUTCMilliseconds()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象中的毫秒值。

Date.getUTCMinutes()

可用性

Flash Player 5。

用法

```
my_date.getUTCMinutes()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象的分钟值。

Date.getUTCMonth()

可用性

Flash Player 5。

用法

```
my_date.getUTCMonth()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象中的月份数（0 代表一月，1 代表二月，依此类推）。

Date.getUTCSeconds()

可用性

Flash Player 5。

用法

```
my_date.getUTCSeconds()
```

参数

无。

返回

一个整数。

说明

方法；按照通用时间返回指定 Date 对象中的秒数。

Date.getYear()

可用性

Flash Player 5。

用法

```
my_date.getYear()
```

参数

无。

返回

一个整数。

说明

方法；按照本地时间返回指定 Date 对象中的年份数。本地时间由运行 Flash Player 的操作系统确定。这里的年份是指完整的年份数减去 1900。例如，2000 年表示为 100。

另请参见

[Date.getFullYear\(\)](#)

Date.setDate()

可用性

Flash Player 5。

用法

```
my_date.setDate(date)
```

参数

date 1 至 31 之间的整数。

返回

一个整数。

说明

方法；按照本地时间为指定的 Date 对象设置表示日期的值，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.setFullYear()

可用性

Flash Player 5。

用法

```
my_date.setFullYear(year [, month [, date]] )
```

参数

year 指定年份的一个 4 位数。两位数字的数不代表完整年份数的简写；例如，99 不代表 1999 年，而是表示 99 年。

month 从 0（一月）到 11（十二月）之间的整数。此参数是可选的。

date 从 1 到 31 之间的数字。此参数是可选的。

返回

一个整数。

说明

方法；按照本地时间设置指定 Date 对象的年份数，并以毫秒为单位返回新时间。如果指定了 *month* 和 *date* 参数，它们也将被设置为本地时间。本地时间由运行 Flash Player 的操作系统确定。

调用此方法并不修改指定 Date 对象的其它字段，但如果因调用该方法而导致表示周几的值发生了变化，则 [Date.getUTCDay\(\)](#) 和 [Date.getDay\(\)](#) 可能报告一个新值。

Date.setHours()

可用性

Flash Player 5。

用法

```
my_date.setHours(hour)
```

参数

hour 从 0（午夜）到 23（深夜 11 点）之间的整数。

返回

一个整数。

说明

方法；按照本地时间设置指定 Date 对象中的小时值，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.setMilliseconds()

可用性

Flash Player 5。

用法

```
my_date.setMilliseconds(milliseconds)
```

参数

milliseconds 从 0 到 999 之间的整数。

返回

一个整数。

说明

方法；按照本地时间设置指定 Date 对象中的毫秒值，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.setMinutes()

可用性

Flash Player 5。

用法

```
my_date.setMinutes(minute)
```

参数

minute 从 0 到 59 之间的整数。

返回

一个整数。

说明

方法；按照本地时间设置指定 Date 对象中的分钟值，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.setMonth()

可用性

Flash Player 5。

用法

```
my_date.setMonth(month [, date ])
```

参数

month 从 0（一月）到 11（十二月）之间的整数。

date 从 1 到 31 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；用本地时间为指定的 Date 对象设置月份数，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.setSeconds()

可用性

Flash Player 5。

用法

```
my_date.setSeconds(second)
```

参数

second 从 0 到 59 之间的整数。

返回

一个整数。

说明

方法；用本地时间为指定的 Date 对象设置表示秒的值，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.setTime()

可用性

Flash Player 5。

用法

```
my_date.setTime(milliseconds)
```

参数

milliseconds 整数值，其中 0 表示 GMT 1970 年 1 月 1 日 0:00。

返回

一个整数。

说明

方法；使用自 1970 年 1 月 1 日午夜起的毫秒数为指定的 Date 对象设置日期值，并以毫秒为单位返回新时间。

Date.setUTCDate()

可用性

Flash Player 5。

用法

```
my_date.setUTCDate(date)
```

参数

date 1 至 31 之间的整数。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象设置日期值，并以毫秒为单位返回新时间。调用此方法并不修改指定 Date 对象的其它字段，但如果因调用该方法导致表示周几的值发生了变化，则 [Date.getUTCDay\(\)](#) 和 [Date.getDay\(\)](#) 可能会报告一个新值。

Date.setUTCFullYear()

可用性

Flash Player 5。

用法

```
my_date.setUTCFullYear(year [, month [, date]])
```

参数

year 指定一个完整 4 位数年份的年份数，例如，2000。

month 从 0（一月）到 11（十二月）之间的整数。此参数是可选的。

date 从 1 到 31 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象 (*my_date*) 设置年份数，并以毫秒为单位返回新时间。

还可选用这种方法设置指定 Date 对象所表示的月份和日期值。调用此方法并不修改指定 Date 对象的其它字段，但如果因调用该方法导致表示周几的值发生了变化，则 [Date.getUTCDay\(\)](#) 和 [Date.getDay\(\)](#) 可能会报告一个新值。

Date.setUTCHours()

可用性

Flash Player 5。

用法

```
my_date.setUTCHours(hour [, minute [, second [, millisecond]]])
```

参数

hour 从 0（午夜）到 23（深夜 11 点）之间的整数。

minute 从 0 到 59 之间的整数。此参数是可选的。

second 从 0 到 59 之间的整数。此参数是可选的。

millisecond 从 0 到 999 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象设置小时值，并以毫秒为单位返回新时间。

Date.setUTCMilliseconds()

可用性

Flash Player 5。

用法

```
my_date.setUTCMilliseconds(millisecond)
```

参数

millisecond 从 0 到 999 之间的整数。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象设置毫秒值，并以毫秒为单位返回新时间。

Date.setUTCMinutes()

可用性

Flash Player 5。

用法

```
my_date.setUTCMinutes(minute [, second [, millisecond]])
```

参数

minute 从 0 到 59 之间的整数。

second 从 0 到 59 之间的整数。此参数是可选的。

millisecond 从 0 到 999 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象设置分钟值，并以毫秒为单位返回新时间。

Date.setUTCMonth()

可用性

Flash Player 5。

用法

```
my_date.setUTCMonth(month [, date])
```

参数

month 从 0（一月）到 11（十二月）之间的整数。

date 从 1 到 31 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象设置月份数，还可以选择设置表示某天 (*date*) 的值，并以毫秒为单位返回新时间。调用此方法并不修改指定 Date 对象的其它字段，但如果因指定 *date* 参数的值而导致表示周几的值发生了变化，则 [Date.getUTCDay\(\)](#) 和 [Date.getDay\(\)](#) 可能会报告一个新值。

Date.setUTCSeconds()

可用性

Flash Player 5。

用法

```
my_date.setUTCSeconds(second [, millisecond])
```

参数

second 从 0 到 59 之间的整数。

millisecond 从 0 到 999 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；用通用时间为指定的 Date 对象设置表示秒的值，并以毫秒为单位返回新时间。

Date.setYear()

可用性

Flash Player 5。

用法

```
my_date.setYear(year)
```

参数

year 如果 *year* 是 0 至 99 之间的整数，则 `setYear` 将年份数设置为 `1900 + year`；否则，年份数就是 *year* 参数的值。

返回

一个整数。

说明

方法；用本地时间为指定的 Date 对象设置年份数，并以毫秒为单位返回新时间。本地时间由运行 Flash Player 的操作系统确定。

Date.toString()

可用性

Flash Player 5。

用法

```
my_date.toString()
```

参数

无。

返回

字符串。

说明

方法；以可读格式返回指定 Date 对象的字符串值，并以毫秒为单位返回新时间。

示例

下面的示例将 `dateOfBirth_date` Date 对象中的信息作为字符串返回。

```
var dateOfBirth_date = new Date(74, 7, 12, 18, 15);  
trace (dateOfBirth_date.toString());
```

输出（用太平洋标准时间表示）：

```
Mon Aug 12 18:15:00 GMT-0700 1974
```

Date.UTC()

可用性

Flash Player 5。

用法

```
Date.UTC(year, month [, date [, hour [, minute [, second [, millisecond ]]]]])
```

参数

year 一个 4 位数，例如 2000。

month 从 0（一月）到 11（十二月）之间的整数。

date 从 1 到 31 之间的整数。此参数是可选的。

hour 从 0（午夜）到 23（深夜 11 点）之间的整数。

minute 从 0 到 59 之间的整数。此参数是可选的。

second 从 0 到 59 之间的整数。此参数是可选的。

millisecond 从 0 到 999 之间的整数。此参数是可选的。

返回

一个整数。

说明

方法；返回 1970 年 1 月 1 日午夜（通用时间）与参数中指定的时间之间的毫秒数。这是一个静态方法，它通过 `Date` 对象的构造函数进行调用，而不是通过特定的 `Date` 对象进行调用。这种方法使您可以创建一个采用通用时间的 `Date` 对象，而 `Date` 构造函数采用本地时间。

示例

下面的示例创建一个以通用时间定义的新的 `Date` 对象 `garyBirthday_date`。这是用于 `new Date` 构造函数方法的示例的通用时间变体：

```
garyBirthday_date = new Date(Date.UTC(1974, 7, 12));
```

default

可用性

Flash Player 6。

用法

```
default: statements
```

参数

statements 任何语句。

返回

无。

说明

语句；定义 `switch` 动作的默认情况。对于一个给定的 `switch` 动作，如果该 `switch` 动作的 *expression* 参数与 `case` 关键字后面的任何一个 *expression* 参数都不相等（使用全等），则执行这些语句。

`switch` 不必一定有 `default` 情况。`default` 情况也不必出现在列表的最后。在 `switch` 动作外部使用 `default` 动作是错误的，脚本将不编译。

示例

在下面的示例中，表达式 `A` 与表达式 `B` 或 `D` 不相等，所以运行 `default` 关键字后面的语句，并将 `trace()` 动作发送到“输出”面板。

```
switch ( A ) {  
    case B:  
        C;  
        break;  
    case D:  
        E;  
        break;  
    default:  
        trace ("no specific case was encountered");  
}
```

另请参见

[switch](#), [case](#), [break](#)

delete

可用性

Flash Player 5。

用法

```
delete reference
```

参数

reference 要消除的变量或对象的名称。

返回

一个布尔值。

说明

运算符；销毁由 *reference* 参数指定的对象或变量，如果该对象被成功删除，则返回 `true`；否则返回 `false` 值。该运算符对于释放脚本所用的内存非常有用。虽然 `delete` 是一个运算符，但它通常作为语句使用，如下所示：

```
delete x;
```

如果 *reference* 参数不存在，或者不能被删除，则 `delete` 运算符可能失败并返回 `false`。预定义的对象和属性以及用 `var` 声明的变量不能删除。不能使用 `delete` 运算符删除影片剪辑。

示例

用法 1：下面的示例创建一个对象并使用它，然后在不再需要时删除它。

```
account = new Object();
account.name = 'Jon';
account.balance = 10000;
```

```
delete account;
```

用法 2：下面的示例删除对象的一个属性。

```
// 创建新对象 "account"
account = new Object();
// 将属性名称分配给 account
account.name = 'Jon';
// 删除该属性
delete account.name;
```

用法 3：下面是删除对象属性的另一个示例。

```
// 创建长度为 0 的 Array 对象
my_array = new Array();
// 向数组中添加一个元素。Array.length 现在是 1
my_array[0] = "abc";
// 向数组中添加又一个元素。Array.length 现在是 2
my_array[1] = "def";
// 向数组中添加又一个元素。Array.length 现在是 3
my_array[2] = "ghi";
// my_array[2] 已被删除，但 Array.length 未被更改
delete array[2];
trace(my_array.length);
```

用法 4：下面的示例说明 `delete` 对于对象引用的行为方式：

```
// 创建一个新对象，并分配变量 ref1
// 来引用该对象
ref1 = new Object();
ref1.name = "Jody";
// 将引用变量复制到新变量中
// 并删除 ref1
ref2 = ref1;
delete ref1;
```

如果 `ref1` 尚未复制到 `ref2` 中，则当删除 `ref1` 时，该对象将被删除，因为将没有指向它的引用。如果删除 `ref2`，则不再有指向该对象的任何引用；将销毁该对象，同时释放它所使用的内存。

另请参见

`var`

do while

可用性

Flash Player 4。

用法

```
do {
    statement(s)
} while (condition)
```

参数

condition 要计算的条件。

statement(s) 只要 *condition* 参数计算结果为 `true` 就会执行的语句。

返回

无。

说明

语句；执行语句，然后只要条件为 `true`，就计算循环中的条件。

另请参见

`break`, `continue`

duplicateMovieClip()

可用性

Flash Player 4。

用法

```
duplicateMovieClip(target, newname, depth)
```

参数

target 要重制的影片剪辑的目标路径。

newname 已重制的影片剪辑的唯一标识符。

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

返回

对重制的影片剪辑的引用。

说明

函数；当 SWF 文件正在播放时，创建一个影片剪辑的实例。无论播放头在原始影片剪辑（或“父级”）中处于什么位置，重制的影片剪辑的播放头始终从第 1 帧开始。父级影片剪辑中的变量不复制到重制的影片剪辑中。如果删除父级影片剪辑，则重制的影片剪辑也会被删除。使用 `removeMovieClip()` 动作或方法可以删除用 `duplicateMovieClip()` 创建的影片剪辑实例。

另请参见

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

dynamic

可用性

Flash Player 6。

用法

```
dynamic class className [ extends superClass ]  
                        [ implements interfaceName [, interfaceName... ] ]  
{  
    // 此处是类定义  
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

说明

关键字；指定基于指定的类的对象可以在运行时添加和访问动态属性。

对于动态类的类型检查与对于非动态类的类型检查相比更为宽松，因为在类定义内访问的成员和在类实例上访问的成员不与在类范围内定义的那些成员进行比较。但是，仍可能对类成员函数进行类型检查，以确定返回类型和参数类型。当处理 `MovieClip` 对象时，此行为特别有用，因为这种情况下可以通过多种不同的方法动态地将属性和对象添加到影片剪辑，如 `MovieClip.createEmptyMovieClip()` 和 `MovieClip.createTextField()`。

动态类的子类也是动态的。

有关更多信息，请参见第 152 页的“创建动态类”。

示例

在以下示例中，类 B 已标记为动态的，因此对它调用某个未声明的函数将不会在编译时引发错误。

```
// 在 B.as 中
dynamic class B extends class_A {
    function B() {
        /*this is the constructor*/
    }
    function m():Number {return 25;}
    function o(s:String):Void {trace(s);}
}

// 在 C.as 中
class C extends class_A {
    function C() {
        /*this is the constructor*/
    }
    function m():Number {return 25;}
    function o(s:String):Void {trace(s);}
}

// 在其它脚本中
var var1 = B.n();    // 无错误
var var2 = C.n()     // 错误，因为在 C.as 中没有函数 n
```

另请参见

[class](#), [extends](#)

else

可用性

Flash Player 4。

用法

```
if (condition){
    statement(s);
} else (condition){
    statement(s);
}
```

参数

condition 计算结果为 `true` 或 `false` 的表达式。

statement(s) 当 `if` 语句中指定的条件为 `false` 时运行的替代语句系列。

返回

无。

说明

语句；指定当 if 语句中的条件返回 false 时要运行的语句。

另请参见

[if](#)

else if

可用性

Flash Player 4。

用法

```
if (condition){
    statement(s);
} else if (condition){
    statement(s);
}
```

参数

condition 计算结果为 true 或 false 的表达式。

statement(s) 当 if 语句中指定的条件为 false 时运行的替代语句系列。

返回

无。

说明

语句；计算条件，并指定当初始 if 语句中的条件返回 false 时要运行的语句。如果 else if 条件返回 true，则 Flash 解释程序运行该条件后面花括号 ({}) 中的语句。如果 else if 条件为 false，则 Flash 将跳过花括号中的语句，而运行花括号之后的语句。使用 else if 动作可在脚本中创建分支逻辑。

示例

下面的示例使用 else if 动作检查对象的每一边是否都在特定的边界内：

```
// 如果对象超出边界，
// 则将其发回并倒转其行进路径
if (this._x>rightBound) {
    this._x = rightBound;
    xInc = -xInc;
} else if (this._x<leftBound) {
    this._x = leftBound;
    xInc = -xInc;
} else if (this._y>bottomBound) {
    this._y = bottomBound;
    yInc = -yInc;
} else if (this._y<topBound) {
    this._y = topBound;
    yInc = -yInc;
}
```

另请参见

[if](#)

#endinitclip

可用性

Flash Player 6。

用法

```
#endinitclip
```

参数

无。

返回

无。

说明

编译器指令；指示初始化动作块的结尾。

示例

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

另请参见

[#initclip](#)

eq（等于 - 字符串专用）

可用性

Flash Player 4。不鼓励在 Flash 5 中使用此运算符，而推荐使用 [==（等于）](#) 运算符。

用法

```
expression1 eq expression2
```

参数

expression1、*expression2* 数字、字符串或变量。

返回

无。

说明

比较运算符；比较两个表达式是否相等。如果 *expression1* 的字符串表示形式与 *expression2* 的字符串表示形式相等，则返回值 true；否则，该运算返回值 false。

另请参见

[==（等于）](#)

Error 类

可用性

Flash Player 7。

说明

包含关于脚本中出现的错误的信息。Error 对象使用 `Error` 构造函数来创建。通常，新的 Error 对象从 `try` 代码块中“抛出”，然后由 `catch` 或 `finally` 代码块“捕获”。

您也可以创建 Error 类的子类，然后引发该子类的实例。

Error 类的方法概要

方法	说明
<code>Error.toString()</code>	返回 Error 对象的字符串表示形式。

Error 类的属性概要

属性	说明
<code>Error.message</code>	包含与错误关联的错误消息的字符串。
<code>Error.name</code>	包含 Error 对象名称的字符串。

Error 类的构造函数

可用性

Flash Player 7。

用法

```
new Error([message])
```

参数

message 与 Error 对象关联的字符串；此参数是可选的。

返回

无。

说明

构造函数；创建新的 Error 对象。如果指定 *message*，其值将赋予对象的 `Error.message` 属性。

示例

在以下示例中，如果给函数传递的两个字符串不相同，则该函数将引发错误（并显示指定的消息）。

```
function compareStrings(string_1, string_2) {  
    if(string_1 != string_2) {  
        throw new Error("Strings do not match.");  
    }  
}  
try {
```

```
        compareStrings("Dog","dog");
    } catch (e) {
        trace(e.toString());
    }
}
```

另请参见

[throw](#), [try..catch..finally](#)

Error.message

可用性

Flash Player 7。

用法

myError.message

说明

属性；包含与 Error 对象关联的消息。默认情况下，此属性的值为 "Error"。当通过将错误字符串传递给 Error 构造函数来创建新的 Error 对象时，可以指定 message 属性。

另请参见

[throw](#), [try..catch..finally](#)

Error.name

可用性

Flash Player 7。

用法

myError.name

说明

属性；包含 Error 对象的名称。默认情况下，此属性的值为 "Error"。

另请参见

[throw](#), [try..catch..finally](#)

Error.toString()

可用性

Flash Player 7。

用法

```
my_err.toString()
```

返回

字符串。

说明

方法；在默认情况下返回字符串 "Error"，如果已定义，则返回 `Error.message` 中包含的值。

另请参见

[Error.message](#), [throw](#), [try..catch..finally](#)

escape

可用性

Flash Player 5。

用法

```
escape(expression)
```

参数

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

返回

无。

说明

函数；将参数转换为字符串，并以 URL 编码格式进行编码，在这种格式中，将所有非字母数字的字符都转义为 % 十六进制序列。

示例

运行下面的代码将得出结果 `Hello%7B%5BWorld%5D%7D`。

```
escape("Hello{[World]}");
```

另请参见

[unescape](#)

eval()

可用性

Flash Player 5 或更高版本支持其完全功能。在向 Flash Player 4 导出时，可以使用 `eval()` 函数，但必须使用斜杠记号，并且只能访问变量，不能访问属性或对象。

用法

```
eval(expression)
```

参数

expression 包含要获取的变量、属性、对象或影片剪辑的名称的字符串。

返回

一个值（对对象或影片剪辑的引用）或 `undefined`。

说明

函数；按照名称访问变量、属性、对象或影片剪辑。如果 *expression* 是变量或属性，则返回该变量或属性的值。如果 *expression* 是对象或影片剪辑，则返回指向该对象或影片剪辑的引用。如果无法找到 *expression* 中指定的元素，则返回 `undefined`。

在 Flash 4 中，使用 `eval()` 模拟数组；在 Flash 5 或更高版本中，建议您使用 `Array` 类模拟数组。

在 Flash 4 中，您还可以使用 `eval()` 动态地设置和获取变量或实例名称的值。然而，也可以使用数组访问运算符 (`[]`) 来实现这一点。

在 Flash 5 或更高版本中，不能使用 `eval()` 动态设置和获取变量的值或实例名称，因为不能在等式的左侧使用 `eval()`。例如，将代码

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

替换为：

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

或者替换为：

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

示例

下面的示例使用 `eval()` 确定表达式 `"piece" + x` 的值。因为该结果是一个变量名 `piece3`，所以 `eval()` 返回该变量的值并将其赋予 `y`：

```
piece3 = "dangerous";  
x = 3;  
  
y = eval("piece" + x);  
trace(y);  
  
// 输出: dangerous
```

另请参见

[Array 类](#)

extends

可用性

Flash Player 6。

用法

```
class className extends otherClassName {}  
interface interfaceName extends otherInterfaceName {}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

className 您所定义的类的名称。
otherClassName *className* 所基于的类的名称。
interfaceName 您所定义的接口的名称。
otherInterfaceName *interfaceName* 所基于的接口的名称。

说明

关键字；定义作为另一个类或接口的子类的类或接口；后者为超类。子类继承超类中定义的所有方法、属性、函数等。

有关更多信息，请参见第 143 页的“创建子类”。

示例

在下面定义的类 B 中，对类 A 的构造函数的调用将自动作为 B 构造函数的第一个语句插入，因为这里尚不存在调用。（即，它在该示例中被注释掉。）

```
class B extends class A  
{  
    function B() { // 它是构造函数  
        //    super(); // 可选；如果省略，则在编译过程中插入  
    }  
    function m():Number {return 25;}  
    function o(s:String):Void {trace(s);}  
}
```

另请参见

[class](#), [implements](#), [interface](#)

false

可用性

Flash Player 5。

用法

`false`

说明

常数；表示与 `true` 相反的唯一的布尔值。

另请参见

[true](#)

_focusrect

可用性

Flash Player 4。

用法

`_focusrect = Boolean;`

说明

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

另请参见

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

for

可用性

Flash Player 5。

用法

```
for(init; condition; next) {  
    statement(s);  
}
```

参数

init 一个在开始循环序列前要计算的表达式，通常为赋值表达式。此参数还允许使用 `var` 语句。

condition 计算结果为 `true` 或 `false` 的表达式。在每次循环迭代前计算该条件；当条件的计算结果为 `false` 时退出循环。

next 在每次循环迭代后要计算的表达式；通常为使用 `++`（递增）或 `--`（递减）运算符的赋值表达式。

statement(s) 要在循环体内执行的指令。

说明

语句；一种循环结构，它首先计算一次 `init`（初始化）表达式，然后按照以下顺序开始循环序列：只要 `condition` 的计算结果为 `true`，就执行 `statement`，然后计算下一个表达式。

一些属性无法用 `for` 或 `for..in` 动作进行枚举。例如，`Array` 类的内置方法（例如 `Array.sort()` 和 `Array.reverse()`）不包括在 `Array` 对象的枚举中，而影片剪辑属性（如 `_x` 和 `_y`）也不能枚举。在外部类文件中，实例成员是不可枚举的；只有动态和静态成员是可枚举的。

示例

下面的示例使用 `for` 在数组中添加元素：

```
my_array=new Array();
for(i=0; i<10; i++) {
    my_array [i] = (i + 5)*10;
    trace(my_array[i]);
}
```

“输出”面板中将显示以下结果：

```
50
60
70
80
90
100
110
120
130
140
```

下面是使用 `for` 重复执行同一动作的示例。在下面的代码中，`for` 循环将从 1 到 100 的数字相加：

```
var sum = 0;
for (var i=1; i<=100; i++) {
    sum = sum + i;
}
```

另请参见

[++（递增）](#)，[--（递减）](#)，[for..in](#)，[var](#)

for..in

可用性

Flash Player 5。

用法

```
for(variableIterant in object){  
    statement(s);  
}
```

参数

variableIterant 作为迭代变量的变量的名称，迭代变量引用数组中对象或元素的每个属性。

object 要重复的对象的名称。

statement(s) 要为每次迭代执行的指令。

返回

无。

说明

语句；循环通过数组中对象或元素的属性，并为对象的每个属性执行 *statement*。

一些属性无法用 `for` 或 `for..in` 动作进行枚举。例如，`Array` 类的内置方法（例如 `Array.sort()` 和 `Array.reverse()`）不包括在 `Array` 对象的枚举中，而影片剪辑属性（如 `_x` 和 `_y`）也不能枚举。在外部类文件中，实例成员是不可枚举的；只有动态和静态成员是可枚举的。

`for..in` 语句迭代所迭代对象的原型链中对象的属性。如果 `child` 的原型为 `parent`，则如果用 `for..in` 迭代 `child` 的属性，也将迭代 `parent` 的属性。

`for..in` 动作枚举对象原型链中的所有对象。首先枚举该对象的属性，接着枚举其直接原型的属性，然后枚举该原型的原型的属性，依次类推。`for..in` 动作不会将相同的属性名枚举两次。如果对象 `child` 具有原型 `parent`，而这两个对象都包含属性 `prop`，则对 `child` 调用的 `for..in` 动作将枚举来自 `child` 的 `prop`，而忽略 `parent` 中的该属性。

示例

下面的示例使用 `for..in` 迭代某对象的属性：

```
myObject = { name:'Tara', age:27, city:'San Francisco' };  
for (name in myObject) {  
    trace ("myObject."+ name + " = " + myObject[name]);  
}
```

此示例的输出如下所示：

```
myObject.name = Tara  
myObject.age = 27  
myObject.city = San Francisco
```

下面的示例将 `typeof` 运算符与 `for..in` 一起使用来迭代特定类型的 `child`：

```
for (name in my_mc) {  
    if (typeof (my_mc[name]) = "movieclip") {  
        trace ("I have a movie clip child named " + name);  
    }  
}
```


下面的示例枚举影片剪辑的子级，并将每个子级发送到其各自时间轴的第 2 帧。

RadioButtonGroup 影片剪辑是一个父级，它具有多个子级：_RedRadioButton_、_GreenRadioButton_ 和 _BlueRadioButton。

```
for (var name in RadioButtonGroup) {
    RadioButtonGroup[name].gotoAndStop(2);
}
```

fscommand()

可用性

Flash Player 3。

用法

fscommand("command", "parameters")

参数

command 一个传递给宿主应用程序用于任何用途的字符串；或者一个传递给 Flash Player 的命令。

parameters 一个传递给宿主应用程序用于任何用途的字符串；或者一个传递给 Flash Player 的值。

返回

无。

说明

函数；使 SWF 文件能够与 Flash Player 或承载 Flash Player 的程序（如 Web 浏览器）进行通讯。还可使用 fscommand 动作将消息传递给 Macromedia Director，或者传递给 Visual Basic、Visual C++ 和其它可承载 ActiveX 控件的程序。

用法 1：若要将消息发送给 Flash Player，必须使用预定义的命令和参数。下表显示可为 fscommand 动作的 *command* 和 *parameters* 参数指定的值，这些值用于控制在 Flash player（包括播放器）中播放的 SWF 文件：

命令	参数	目的
quit	无	关闭播放器。
fullscreen	true 或 false	指定 true 将 Flash Player 设置为全屏模式。如果指定 false，播放器会返回到常规菜单视图。
allowscale	true 或 false	如果指定 false，则设置播放器以始终按 SWF 文件的原始大小绘制 SWF 文件，从不进行缩放。如果指定 true，则强制 SWF 文件缩放到播放器的 100%。
showmenu	true 或 false	如果指定 true，则启用整个上下文菜单项集合。如果指定 false，则使得除 “关于 Flash Player” 外的所有上下文菜单项变暗。
exec	应用程序的路径	在播放器内执行应用程序。
trapallkeys	true 或 false	如果指定 true，则将所有按键事件（包括快捷键事件）发送到 Flash Player 中的 onClipEvent(keyDown/keyUp) 处理函数。

exec 命令只能包含字符 A 至 Z、a 至 z、0 至 9、句点 (.) 和下划线 (_)。exec 仅在子目录 fscommand 中运行。也就是说,如果您使用 fscommand exec 命令调用应用程序,该应用程序必须位于名为 fscommand 的子目录中。

用法 2: 若要在 Web 浏览器中使用 fscommand 动作将消息发送到脚本撰写语言 (如 JavaScript), 可以在 *command* 和 *parameters* 参数中传递任意两个参数。这些参数可以是字符串或表达式, 它们在“捕获”或处理 fscommand 动作的 JavaScript 函数中使用。

在 Web 浏览器中, fscommand 动作在包含 SWF 文件的 HTML 页中调用 JavaScript 函数 `movieName_DoFSCommand`。movieName 是 Flash Player 影片的名称, 该名称由 EMBED 标签的 NAME 属性指定, 或者由 OBJECT 标签的 ID 属性指定。如果您为 Flash Player 影片指定名称 myDocument, 所调用的 JavaScript 函数就是 `myDocument_DoFSCommand`。

用法 3: fscommand 动作可以将消息发送给 Macromedia Director, Lingo 将消息解释为字符串、事件或可执行的 Lingo 代码。如果该消息为字符串或事件, 则必须编写 Lingo 代码以便从 fscommand 动作接收该消息, 并在 Director 中执行动作。有关更多信息, 请参见 Director 支持中心, 网址为 www.macromedia.com/support/director。

用法 4: 在 Visual Basic、Visual C++ 和可承载 ActiveX 控件的其它程序中, fscommand 利用可在环境的编程语言中处理的两个字符串发送 VB 事件。有关更多信息, 请使用 Flash method (Flash 方法) 关键字搜索 Flash 支持中心, 网址为 www.macromedia.com/go/flash_support_cn。

示例

用法 1: 在下面的示例中, fscommand 动作设置 Flash Player, 以便在释放按钮时, 将 SWF 文件放大到整个显示器屏幕大小。

```
on (release) {  
    fscommand("fullscreen", true);  
}
```

用法 2: 下面的示例使用应用到 Flash 中按钮的 fscommand 动作打开 HTML 页中的 JavaScript 消息框。消息本身作为 fscommand 参数发送到 JavaScript。

必须将一个函数添加到包含 SWF 文件的 HTML 页。此函数 (`myDocument_DoFSCommand`) 位于 HTML 页中, 等待 Flash 中的 fscommand 动作。当在 Flash 中触发 fscommand 后 (例如, 当用户按下按钮时), *command* 和 *parameter* 字符串被传递到 `myDocument_DoFSCommand` 函数。可以在 JavaScript 或 VBScript 代码中以任何需要的方式使用所传递的字符串。在此示例中, 该函数包含一个条件 if 语句, 该语句检查命令字符串是否为 "messagebox"。如果是, 则 JavaScript 警告框 (或“消息框”) 打开并显示 *parameters* 字符串的内容。

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

在 Flash 文档中, 将 fscommand 动作添加到按钮:

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

也可以为 fscommand 动作和参数使用表达式, 如下面的示例所示:

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

若要测试影片, 请选择“文件” > “发布预览” > “HTML”。

注意：如果在 HTML “发布设置” 中使用具有 FSCommand 模板的 Flash 发布 SWF 文件，则将自动插入 `myDocument_DoFSCommand` 函数。该 SWF 文件的 NAME 和 ID 属性将是文件名。例如，对于文件 `myDocument fla`，这些属性将设置为 `myDocument`。

function

可用性

Flash Player 5。

用法

```
function functionname ([parameter0, parameter1,...parameterN]){  
    statement(s)  
}  
function ([parameter0, parameter1,...parameterN]){  
    statement(s)  
}
```

参数

functionname 新函数的名称。

parameter 一个标识符，表示要传递给函数的参数。这些参数是可选的。

statement(s) 为 `function` 的函数体定义的任何动作脚本指令。

返回

无。

说明

语句；您定义的用来执行特定任务的一组语句。可以在 SWF 文件的一个地方声明 或定义函数，然后从 SWF 文件的其它脚本中调用它。定义函数时，还可以为其指定参数。参数是函数要对其进行操作的值的占位符。每次调用函数时，可以向其传递不同的参数。这使您可以在不同场合重复使用一个函数。

在函数的 *statement(s)* 中使用 `return` 动作可使函数返回或生成一个值。

用法 1：用指定的 *functionname*、*parameters* 和 *statement(s)* 声明一个 `function`。当调用函数时，则调用函数声明。允许提前引用；在同一“动作”列表中，函数可以先调用后声明。一个函数的声明会替换同一函数以前的任何声明。只要是允许使用语句的地方就可使用此语法。

用法 2：创建一个匿名函数并返回它。此语法用于表达式中，对于在对象中安装方法尤其有用。

示例

用法 1：下面的示例定义函数 `sqr`，该函数接受一个参数并返回该参数的 `square(x*x)`。如果在同一脚本中声明和使用该函数，则可以先使用该函数，后声明它。

```
y=sqr(3);
```

```
function sqr(x) {  
    return x*x;  
}
```

用法 2：下面的函数定义一个 `Circle` 对象：

```
function Circle(radius) {  
    this.radius = radius;  
}
```

下面的语句定义一个匿名函数，该函数计算圆形的面积，并将其作为方法附加到对象 `Circle`：

```
Circle.prototype.area = function () {return Math.PI * this.radius * this.radius}
```

Function 类

可用性

Flash Player 6。

Function 类的方法概要

方法	说明
<code>Function.apply()</code>	使动作脚本代码能够调用函数。
<code>Function.call()</code>	调用 <code>Function</code> 对象表示的函数。

Function 类的属性概要

属性	说明
<code>Function.prototype</code>	引用一个作为类的原型的对象。

Function.apply()

可用性

Flash Player 6。

用法

```
myFunction.apply(thisObject, argumentsObject)
```

参数

thisObject *myFunction* 应用到的对象。

argumentsObject 一个数组，其元素作为参数传递给 *myFunction*。

返回

被调用函数指定的任何值。

说明

方法；指定将在动作脚本调用的任何函数内使用的 `this` 的值。此方法还指定要传递给任何被调用函数的参数。因为 `apply()` 是 `Function` 类的方法，所以它也是动作脚本中每个函数对象的方法。

参数被指定为 `Array` 对象。如果在脚本实际执行前，无法知道要传递的参数的数量，那么这种方法通常很有用。

示例

下面的函数调用是等效的：

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

您可以构造一个包含输入入口字段的 SWF 文件，该字段允许用户输入要调用的函数名，以及要传递给该函数的零个或多个参数。如果按“调用”按钮，则将使用 `apply` 方法调用该函数，并指定相应的参数。

在此示例中，用户在名为 `functionName` 的输入文本字段中指定函数名。在名为 `numParameters` 的输入文本字段中指定参数数量。在文本字段中最多可指定 10 个参数，这些文本字段名为 `parameter1`、`parameter2`，直到 `parameter10`。

```
on (release) {
    callTheFunction();
}
...
function callTheFunction()
{
    var theFunction = eval(functionName.text);
    var n = Number(numParameters);
    var parameters = [];
    for (var i = 0; i < n; i++) {
        parameters.push(eval("parameter" + i));
    }
    theFunction.apply(null, parameters);
}
```

Function.call()

可用性

Flash Player 6。

用法

```
myFunction.call(thisObject, parameter1, ..., parameterN)
```

参数

thisObject 指定函数体内 `this` 的值。

parameter1 要传递给 *myFunction* 的参数。可以指定零个或多个参数。

parameterN

返回

无。

说明

方法；调用 `Function` 对象表示的函数。动作脚本中的每个函数都由一个 `Function` 对象来表示，所以所有的函数都支持此方法。

几乎在所有的情形下，函数调用运算符 (`()`) 都可以代替此方法来使用。函数调用运算符使代码简明易读。此方法主要用于需要显式控制函数调用的 `this` 参数时。通常，如果作为对象的方法调用函数，则在函数体内，`this` 设置为 `myObject`，如下所示：

```
myObject.myMethod(1, 2, 3);
```

在某些情况下，您可能希望 `this` 指向其它地方；例如这种情况：函数必须作为对象的方法进行调用，但该函数实际上不是作为该对象的方法存储的。

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

您可以将值 `null` 传递给 `thisObject` 参数，以便作为常规函数而不是作为对象的方法来调用函数。例如，下面的函数调用是等效的：

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

示例

此示例使用 `Function.call()` 使函数表现得像另一个对象的方法，而不将此函数存储在对象中。

```
function MyObject() {
}
function MyMethod(obj) {
    trace("this == obj?" + (this == obj));
}
var obj = new MyObject();
MyMethod.call(obj, obj);
```

`trace()` 动作将以下代码发送到“输出”面板：

```
this == obj? true
```

Function.prototype

可用性

Flash Player 5。如果您使用的是动作脚本 2.0，则不需要使用此属性；该属性反映动作脚本 1 中继承的实现方式。

用法

```
myFunction.prototype
```

说明

属性；在动作脚本 1 的构造函数中，`prototype` 属性引用一个对象，该对象是构造类的原型。由构造函数创建的类的每个实例都继承该原型对象的所有属性和方法。

ge（大于或等于 - 字符串专用）

可用性

Flash Player 4。不鼓励在 Flash 5 中使用此运算符，而推荐使用 `>=`（大于等于）运算符。

用法

```
expression1 ge expression2
```

参数

expression1、*expression2* 数字、字符串或变量。

返回

无。

说明

运算符（比较）；比较 *expression1* 和 *expression2* 的字符串表示形式，如果 *expression1* 大于或等于 *expression2*，则返回 `true`；否则，返回 `false`。

另请参见

[>= \(大于等于\)](#)

get

可用性

Flash Player 6。

用法

```
function get property() {  
    // 此处是您的语句  
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

property 您要用来表示 get 访问的属性的单词；该值必须与在相应 set 命令中使用的值相同。

返回

由 *propertyName* 指定的属性的值。

说明

关键字；允许基于您在外部类文件中定义的类隐式“获取”与对象关联的属性。通过使用隐式获取方法，您可以不必直接访问对象就访问其属性。隐式获取 / 设置方法是对动作脚本 1 中 `Object.addProperty()` 方法的句法简化。

有关更多信息，请参见第 151 页的“隐式获取 / 设置方法”。

另请参见

[Object.addProperty\(\)](#), [set](#)

getProperty

可用性

Flash Player 4。

用法

```
getProperty(my_mc, property)
```

参数

my_mc 要获取其属性的影片剪辑的实例名称。

property 影片剪辑的属性。

返回

指定属性的值。

说明

函数；返回影片剪辑 *my_mc* 的指定属性的值。

示例

下面的示例获取影片剪辑 *my_mc* 的水平轴坐标 (*_x*)，并将其分配给变量 *my_mc_x*：

```
my_mc_x = getProperty(_root.my_mc, _x);
```

getTimer

可用性

Flash Player 4。

用法

```
getTimer()
```

参数

无。

返回

自 SWF 文件开始播放时起已经过的毫秒数。

说明

函数；返回自 SWF 文件开始播放时起已经过的毫秒数。

getURL()

可用性

Flash 2。GET 和 POST 选项仅适用于 Flash Player 4 和 Player 的更高版本。

用法

```
getURL(url [, window [, "variables"]])
```

参数

url 可从该处获取文档的 URL。

window 可选参数，指定文档应加载到其中的窗口或 HTML 框架。您可输入特定窗口的名称，或从下面的保留目标名称中选择：

- *_self* 指定当前窗口中的当前框架。
- *_blank* 指定一个新窗口。
- *_parent* 指定当前框架的父级。
- *_top* 指定当前窗口中的顶级框架。

variables 用于发送变量的 GET 或 POST 方法。如果没有变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，该方法用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，该方法用于发送大量的变量。

返回

无。

说明

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

示例

此示例将一个新 URL 加载到空浏览器窗口中。getURL() 动作将变量 incomingAd 作为 *url* 参数的目标，这样您无需编辑 SWF 文件即可更改加载的 URL。在这之前，在 SWF 文件中使用 loadVariables() 动作将 incomingAd 变量的值传递到 Flash 中。

```
on (release) {  
    getURL(incomingAd, "_blank");  
}
```

另请参见

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

getVersion

可用性

Flash Player 5。

用法

```
getVersion()
```

参数

无。

返回

包含 Flash Player 版本和平台信息的字符串。

说明

函数；返回包含 Flash Player 版本和平台信息的字符串。

getVersion 函数只能返回 Flash Player 5 或 Player 更高版本的信息。

示例

以下示例是 getVersion 函数返回的字符串。

```
WIN 5,0,17,0
```

这指示出平台是 Microsoft Windows，Flash Player 的主要版本号为 5，次要版本号为 17 (5.0r17)。

另请参见

[System.capabilities.os](#), [System.capabilities.version](#)

_global 对象

可用性

Flash Player 6。

用法

`_global.identifier`

参数

无。

返回

对包含核心动作脚本类的全局对象（例如 String、Object、Math 和 Array）的引用。

说明

标识符；创建全局变量、对象或类。例如，您可以创建公开为全局动作脚本对象的库，此库非常类似于 Math 或 Date 对象。与时间轴声明或局部声明的变量和函数不一样，全局变量和函数只要未被内部范围中具有相同名称的标识符遮蔽，则它们对于 SWF 文件中的每个时间轴和范围均是可见的。

示例

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

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

另请参见

[var](#), [set variable](#)

gotoAndPlay()

可用性

Flash 2。

用法

```
gotoAndPlay([scene,] frame)
```

参数

scene 可选字符串，指定播放头要转到的场景的名称。

frame 表示将播放头转到的帧编号的数字，或者表示将播放头转到的帧的标签的字符串。

返回

无。

说明

函数；将播放头转到场景中指定的帧并从该帧开始播放。如果未指定场景，则播放头将转到当前场景中的指定帧。

示例

当用户单击 `gotoAndPlay()` 动作所分配到的按钮时，播放头将转到当前场景中的第 16 帧并开始播放。

```
on (release) {  
    gotoAndPlay(16);  
}
```

另请参见

[MovieClip.gotoAndPlay\(\)](#)

gotoAndStop()

可用性

Flash 2。

用法

```
gotoAndStop([scene,] frame)
```

参数

scene 可选字符串，指定播放头要转到的场景的名称。

frame 表示将播放头转到的帧编号的数字，或者表示将播放头转到的帧的标签的字符串。

返回

无。

说明

函数；将播放头转到场景中指定的帧并停止播放。如果未指定场景，则播放头将转到当前场景中的帧。

示例

当用户单击 `gotoAndStop()` 动作所分配到的按钮时，播放头将转到当前场景中的第 5 帧并且 SWF 文件停止播放。

```
on (release) {  
    gotoAndStop(5);  
}
```

另请参见

[stop\(\)](#)

gt（大于 - 字符串专用）

可用性

Flash Player 4。不鼓励在 Flash 5 中使用此运算符，而推荐使用新的 [>（大于）](#) 运算符。

用法

expression1 gt *expression2*

参数

expression1、*expression2* 数字、字符串或变量。

说明

运算符（比较）；比较 *expression1* 和 *expression2* 的字符串表示形式，如果 *expression1* 大于 *expression2*，则返回 `true`；否则，返回 `false`。

另请参见

[>（大于）](#)

_highquality

可用性

Flash Player 4；不鼓励使用该属性，而推荐使用 [_quality](#)。

用法

`_highquality`

说明

不鼓励使用的属性（全局）；指定应用于当前 SWF 文件的锯齿消除级别。指定 2（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 1（高品质），则应用消除锯齿功能；如果 SWF 文件不包含动画，这将对位图进行平滑处理。指定 0（低品质），则不消除锯齿。

示例

```
_highquality = 1;
```

另请参见

[_quality](#), [toggleHighQuality\(\)](#)

if

可用性

Flash Player 4。

用法

```
if(condition) {  
    statement(s);  
}
```

参数

condition 计算结果为 true 或 false 的表达式。

statement(s) 在条件计算结果为 true 的情况下执行的指令。

返回

无。

说明

语句；对条件进行计算以确定 SWF 文件中的下一步动作。如果条件为 true，则 Flash 将运行条件后面花括号 ({}) 内的语句。如果条件为 false，则 Flash 将跳过花括号内的语句，而运行花括号后面的语句。使用 if 动作可在脚本中创建分支逻辑。

示例

在下面的示例中，括号内的条件对变量 name 进行计算以查看其是否具有文本值 "Erica"。如果有，则运行花括号内的 play() 动作。

```
if(name == "Erica"){  
    play();  
}
```

下面的示例使用 if 动作来确定用户何时释放 SWF 文件中的可拖动对象。如果对象在拖动后 300 毫秒之内被释放，则条件的计算结果为 true，并运行花括号内的语句。这些语句设置变量以存储对象的新位置、拖放对象的难易程度和拖放对象时的速度。timePressed 变量也将被重置。如果对象在拖动后超过 300 毫秒之后被释放，则条件的计算结果为 false，并且不运行任何语句。

```
if (getTimer()<timePressed+300) {  
    // 如果条件为 true，  
    // 则抛出此对象。  
    // 此对象的新位置在哪里？  
    xNewLoc = this._x;  
    yNewLoc = this._y;  
    // 将它抛出多远？  
    xTravel = xNewLoc-xLoc;  
    yTravel = yNewLoc-yLoc;  
    // 根据它们随该对象行进的距离设置  
    // 该对象的速度  
    xInc = xTravel/2;  
    yInc = yTravel/2;  
    timePressed = 0;  
}
```

另请参见

[else](#)

ifFrameLoaded

可用性

Flash Player 3。不鼓励在 Flash 5 中使用 `ifFrameLoaded` 动作；Macromedia 建议使用 `MovieClip._framesloaded` 属性。

用法

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

参数

scene 可选字符串，指定必须加载的场景的名称。

frame 在执行下一条语句之前必须加载的帧编号或帧标签。

statement(s) 在加载了指定的场景（或场景及帧）时所执行的指令。

返回

无。

说明

不鼓励使用的动作；检查特定帧的内容是否为本地可用。使用 `ifFrameLoaded` 可在 SWF 文件的其余部分下载到本地计算机时开始播放简单的动画。使用 `_framesloaded` 和使用 `ifFrameLoaded` 之间的差别在于 `_framesloaded` 允许您添加自己的 `if` 或 `else` 语句。

另请参见

[MovieClip._framesloaded](#)

implements

可用性

Flash Player 6。

用法

```
myClass implements interface01 [, interface02, ...]
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

说明

关键字；定义必须为所实现的接口中定义的所有方法提供实现的类。有关更多信息，请参见[第 148 页的“接口作为数据类型”](#)。

示例

请参见 [interface](#)。

另请参见

[class](#), [extends](#), [interface](#)

import

可用性

Flash Player 6。

用法

```
import className  
import packageName.*
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。在“动作”面板以及外部类文件中支持该语句。

参数

className 您在外部类文件中定义的类的全限定名。

packageName 存储相关类文件的目录。

说明

关键字；使您不必指定类的全限定名就可以访问类。例如，如果您要在脚本中使用类 `macr.util.users.UserClass.as`，则必须通过其全限定名引用它或导入它；如果您导入该类，则可以通过类名称引用它：

```
// 在导入前  
var myUser:UserClass = new macr.util.users.UserClass();  
// 在导入后  
import macr.util.users.UserClass;  
var myUser:UserClass = new UserClass();
```

如果在您要访问的目录中有若干类文件，则可以通过一个语句全部导入它们：

```
import macr.util.users.*;
```

您必须首先发出 `import` 语句，然后才能在不完全指定导入的类的名称的情况下访问该类。

如果您导入一个类，但没有在脚本中使用该类，则该类不作为 SWF 文件的一部分导出。这意味着您导入大型包时不必担心 SWF 文件的大小；只有在实际使用某一类的情况下，才会在 SWF 文件中包括与该类关联的字节码。

`import` 语句仅应用于调用该语句的当前脚本（帧或对象）。例如，假设您在某个 Flash 文档的第 1 帧上导入了 `macr.util` 包中的所有类。那么，在那一帧上则可以使用简单类名来引用该包中的类。

```
// 在某个 FLA 的第 1 帧上：  
import macr.util.*;
```

```
var myFoo:foo = new foo();
```

但在其它帧脚本中，仍需要使用其全限定类名来引用该包中的类 (`var myFoo:foo = new macr.util.foo();`)，或在其它帧上也添加一条 `import` 语句，导入该包中的类。

有关导入的更多信息，请参见第 151 页的“导入类”和第 150 页的“使用包”。

#include

可用性

Flash Player 4。

用法

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

注意：不要将分号 (;) 放在包含 #include 语句的行的末尾。

参数

[path] filename.as 要添加到“动作”面板的脚本的文件名和可选路径；.as 是推荐使用的文件扩展名。

返回

无。

说明

编译器指令：包含指定文件的内容，就像该文件中的命令属于调用脚本本身的一部分。#include 指令在编译时调用。因此，如果对外部文件进行了任何更改，则必须保存该文件，并重新编译使用它的任何 FLA 文件。

如果对包含 #include 语句的脚本使用“语法检查”按钮，则该包含文件中的语法也将被检查。

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

您可以为要包含的文件指定无路径、相对路径或绝对路径。

- 如果您没有指定路径，则 AS 文件必须与 FLA 文件或包含 #include 语句的脚本位于相同的目录中。
- 若要为 AS 文件指定相对于 FLA 文件或脚本的路径，请使用一个圆点 (.) 来指示当前目录，使用两个圆点 (..) 来指示父目录，并且使用斜杠 (/)。请参见下面的示例。
- 若要为 AS 文件指定绝对路径，请使用您的平台（Macintosh 或 Windows）所支持的格式。请参见下面的示例。但是，由于此用法要求目录结构在用来编译脚本的任何计算机上均相同，因此不建议采用此用法。

示例

以下示例显示多种为要包含在脚本中的文件指定路径的方法。

```
// 请注意，#include 语句未以分号 (;) 结尾
// AS 文件与 FLA 文件或脚本位于同一个目录中
#include "init_script.as"

// AS 文件位于包含 FLA 文件或脚本
// 的目录的子目录中
// 该子目录名为“FLA_includes”
#include "FLA_includes/init_script.as"

// AS 文件所在的目录与 FLA 文件或脚本所在的目录位于同一级
// 该目录名为“ALL_includes”
#include "../ALL_includes/init_script.as"

// AS 文件在 Windows 上由绝对路径来指定
// 请注意应使用斜杠，而不是反斜杠
#include "C:/Flash_scripts/init_script.as"
```



```
// AS 文件在 Macintosh 上由绝对路径来指定
#include "Mac HD:Flash_scripts:init_script.as"
```

另请参见

[import](#)

Infinity

可用性

Flash Player 5。

用法

`Infinity`

说明

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

-Infinity

可用性

Flash Player 5。

用法

`-Infinity`

说明

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

#initclip

可用性

Flash Player 6。

用法

`#initclip order`

参数

order 指定 `#initclip` 代码块执行顺序的整数。这是一个可选参数。

说明

编译器指令；指示初始化动作块的开始。当同时初始化多个剪辑时，您可以使用 *order* 参数来指定先执行哪个初始化动作。在定义影片剪辑元件时，将执行初始化动作。如果影片剪辑是导出的元件，则初始化动作将在执行 SWF 文件第 1 帧上的动作之前执行。否则，组件初始化动作将在一特定帧（该帧包含关联影片剪辑元件的第一个实例）的帧动作脚本之前立即执行。

初始化动作在 SWF 文件播放过程中仅执行一次，您应该将这些动作用于一次性的初始化动作，例如类定义和注册。

另请参见

[#endinitclip](#)

instanceof

可用性

Flash Player 6。

用法

object instanceof *class*

参数

object 动作脚本对象。

class 对动作脚本构造函数（例如 String 或 Date）的引用。

返回

如果 *object* 是 *class* 的实例，则 instanceof 返回 true ；否则，instanceof 返回 false。此外，_global instanceof Object 返回 false。

说明

运算符；确定某对象是否属于指定的类。测试 *object* 是否为 *class* 的实例。

instanceof 运算符不会将原始类型转换为包装对象。例如，下面的代码返回 true：

```
new String("Hello") instanceof String;
```

而下面的代码则返回 false：

```
"Hello" instanceof String;
```

另请参见

[typeof](#)

int

可用性

Flash Player 4。不鼓励在 Flash 5 中使用该函数，而推荐使用 [Math.round\(\)](#)。

用法

int(*value*)

参数

value 要舍入为整数的数字。

返回

无。

说明

函数；将十进制数字转换为最相近的整数值。

另请参见

[Math.floor\(\)](#)

interface

可用性

Flash Player 6。

用法

```
interface InterfaceName {}  
interface InterfaceName [extends InterfaceName [, InterfaceName ...]] {}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

说明

关键字；定义接口。接口与类相似，但也具有以下重要差异：

- 接口仅包含方法的声明，而不包含其实现。也就是说，实现接口的每个类必须为该接口中声明的每个方法提供实现。
- 接口定义中只允许有公共成员。此外，不允许实例和类成员。
- 在接口定义中不允许 `get` 和 `set` 语句。

有关更多信息，请参见第 147 页的“创建和使用接口”。

示例

以下示例显示用于定义和实现接口的若干方法。

(在顶层包 `.as` 文件 `Ia`、`B`、`C`、`Ib`、`D`、`Ic`、`E` 中)

```
// 文件名 Ia.as  
interface Ia  
{  
    function k():Number;           // 仅限方法声明  
    function n(x:Number):Number; // 不实现  
}  
// 文件名 B.as  
class B implements Ia  
{  
    function k():Number {return 25;}  
    function n(x:Number):Number {return x+5;}  
}  
// 外部脚本或“动作”面板  
var mvar:B = new B();  
trace(mvar.k()); // 25  
trace(mvar.n(7)); // 12  
  
// 文件名 c.as  
class C implements Ia  
{  
    function k():Number {return 25;}  
} // 错误：类必须实现所有接口方法  
  
// 文件名 Ib.as  
interface Ib  
{  
    function o():Void;  
}  
class D implements Ia, Ib
```

```

{
    function k():Number {return 15;}
    function n(x:Number):Number {return x*x;}
    function o():Void {trace("o");}
}

// 外部脚本或“动作”面板
mvar = new D();
trace(D.k());    // 15
trace(D.n(7));   // 49
trace(D.o());    // "o"

interface Ic extends Ia
{
    function p():Void;
}
class E implements Ib, Ic
{
    function k():Number {return 25;}
    function n(x:Number):Number {return x+5;}
    function o():Void {trace("o");}
    function p():Void {trace("p");}
}

```

另请参见

[class](#), [extends](#), [implements](#)

isFinite

可用性

Flash Player 5。

用法

`isFinite(expression)`

参数

expression 要计算的布尔值表达式、变量表达式或其它表达式。

返回

一个布尔值。

说明

函数 `isFinite` 对 *expression* 进行计算，如果为有限数，则返回 `true`，如果为无穷大或负无穷大，则返回 `false`。无穷大或负无穷大的出现指示有错误的数学条件，例如被 0 除。

示例

下面是 `isFinite` 返回值的示例：

```

isFinite(56)
// 返回 true

isFinite(Number.POSITIVE_INFINITY)
// 返回 false

```

isNaN()

可用性

Flash Player 5。

用法

`isNaN(expression)`

参数

expression 要计算的布尔表达式、变量表达式或其它表达式。

返回

一个布尔值。

说明

函数；对参数进行计算，如果值不是数字 (NaN)，则返回 `true`，指示存在数学错误。

示例

下面的代码显示了 `isNaN` 函数的返回值。

```
isNaN("Tree")
// 返回 true

isNaN(56)
// 返回 false

isNaN(Number.POSITIVE_INFINITY)
// 返回 false
```

另请参见

[NaN](#), [Number.NaN](#)

Key 类

可用性

Flash Player 6。

说明

`Key` 类是不通过构造函数即可使用其方法和属性的顶级类。使用 `Key` 类的方法可生成用户能够通过标准键盘控制的界面。`Key` 类的属性是常量，表示控制游戏时最经常使用的键。有关键控代码值的完整列表，请参见第 745 页的附录 C “[键盘键和键控代码值](#)”。

Key 类的方法概要

方法	说明
Key.addListener()	注册一个对象，以便在调用 <code>onKeyDown</code> 和 <code>onKeyUp</code> 方法时接收通知。
Key.getAscii()	返回按下的最后一个键的 ASCII 值。
Key.getCode()	返回按下的最后一个键的虚拟键控代码。

方法	说明
<code>Key.isDown()</code>	当按下参数中指定的键时返回 <code>true</code> 。
<code>Key.isToggled()</code>	当激活 Num Lock 键或 Caps Lock 键时返回 <code>true</code> 。
<code>Key.removeListener()</code>	删除以前用 <code>Key.addListener()</code> 注册的对象。

Key 类的属性概要

Key 类的所有属性都是常量。

属性	说明
<code>Key.BACKSPACE</code>	与 Backspace 键的键控代码值 (8) 关联的常量。
<code>Key.CAPSLOCK</code>	与 Caps Lock 键的键控代码值 (20) 关联的常量。
<code>Key.CONTROL</code>	与 Control 键的键控代码值 (17) 关联的常量。
<code>Key.DELETEKEY</code>	与 Delete 键的键控代码值 (46) 关联的常量。
<code>Key.DOWN</code>	与向下箭头键的键控代码值 (40) 关联的常量。
<code>Key.END</code>	与 End 键的键控代码值 (35) 关联的常量。
<code>Key.ENTER</code>	与 Enter 键的键控代码值 (13) 关联的常量。
<code>Key.ESCAPE</code>	与 Escape 键的键控代码值 (27) 关联的常量。
<code>Key.HOME</code>	与 Home 键的键控代码值 (36) 关联的常量。
<code>Key.INSERT</code>	与 Insert 键的键控代码值 (45) 关联的常量。
<code>Key.LEFT</code>	与左箭头键的键控代码值 (37) 关联的常量。
<code>Key.PGDN</code>	与 Page Down 键的键控代码值 (34) 关联的常量。
<code>Key.PGUP</code>	与 Page Up 键的键控代码值 (33) 关联的常量。
<code>Key.RIGHT</code>	与右箭头键的键控代码值 (39) 关联的常量。
<code>Key.SHIFT</code>	与 Shift 键的键控代码值 (16) 关联的常量。
<code>Key.SPACE</code>	与空格键的键控代码值 (32) 关联的常量。
<code>Key.TAB</code>	与 Tab 键的键控代码值 (9) 关联的常量。
<code>Key.UP</code>	与向上箭头键的键控代码值 (38) 关联的常量。

Key 类的侦听器概要

方法	说明
<code>Key.onKeyDown</code>	当按下某按键时获得通知。
<code>Key.onKeyUp</code>	当释放某按键时获得通知。

Key.addListener()

可用性

Flash Player 6。

用法

```
Key.addListener (newListener)
```

参数

newListener 具有方法 `onKeyDown` 和 `onKeyUp` 的对象。

返回

无。

说明

方法；注册一个对象以接收 `onKeyDown` 和 `onKeyUp` 通知。当按下或释放按键时，不管输入焦点情况如何，所有用 `addListener()` 注册的侦听对象都将调用其 `onKeyDown` 方法或 `onKeyUp` 方法。可以有多个对象侦听键盘通知。如果已经注册了侦听器 *newListener*，则不会发生任何更改。

示例

以下示例创建一个新的侦听器对象，并为 `onKeyDown` 和 `onKeyUp` 定义一个函数。最后一行使用 `addListener()` 向 `Key` 对象注册该侦听器，以使该对象可接收按下和释放按键事件的通知。

```
myListener = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
myListener.onKeyUp = function () {
    trace ("You released a key.");
}
Key.addListener(myListener);
```

以下示例将快捷键 `Control+7` 分配给实例名称为 `myButton` 的按钮，并将与该快捷键有关的信息提供给屏幕读取器（请参见 `_accProps`）。在此示例中，在您按下 `Control+7` 组合键时，`myOnPress` 函数在“输出”面板中显示文本“hello”；您可以在自己的文件中创建更具意义的函数。

```
function myOnPress() {
    trace( "hello" );
}

function myOnKeyDown() {
    if (Key.isDown(Key.CONTROL) && Key.getCode() == 55) // 55 是 7 的键控代码
    {
        Selection.setFocus( myButton );
        myButton.onPress();
    }
}

var myListener = new Object();
myListener.onKeyDown = myOnKeyDown;
Key.addListener(myListener);

myButton.onPress = myOnPress;
```

```
myButton._accProps.shortcut = "Ctrl+F"  
Accessibility.updateProperties();
```

另请参见

[Key.getCode\(\)](#), [Key.isDown\(\)](#), [Key.onKeyDown](#), [Key.onKeyUp](#), [Key.removeListener\(\)](#)

Key.BACKSPACE

可用性

Flash Player 5。

用法

Key.BACKSPACE

说明

属性；与 Backspace 键的键控代码值 (8) 关联的常量。

Key.CAPSLOCK

可用性

Flash Player 5。

用法

Key.CAPSLOCK

说明

属性；与 Caps Lock 键的键控代码值 (20) 关联的常量。

Key.CONTROL

可用性

Flash Player 5。

用法

Key.CONTROL

说明

属性；与 Control 键的键控代码值 (17) 关联的常量。

Key.DELETEKEY

可用性

Flash Player 5。

用法

Key.DELETEKEY

说明

属性；与 Delete 键的键控代码值 (46) 关联的常量。

Key.DOWN

可用性

Flash Player 5。

用法

`Key.DOWN`

说明

属性；与向下箭头键的键控代码值 (40) 关联的常量。

Key.END

可用性

Flash Player 5。

用法

`Key.END`

说明

属性；与 End 键的键控代码值 (35) 关联的常量。

Key.ENTER

可用性

Flash Player 5。

用法

`Key.ENTER`

说明

属性；与 Enter 键的键控代码值 (13) 关联的常量。

Key.ESCAPE

可用性

Flash Player 5。

用法

`Key.ESCAPE`

说明

属性；与 Escape 键的键控代码值 (27) 关联的常量。

Key.getAscii()

可用性

Flash Player 5。

用法

```
Key.getAscii();
```

参数

无。

返回

表示上次所按键的 ASCII 值的整数。

说明

方法；返回按下或释放的最后一个键的 ASCII 码。返回的 ASCII 值为英文键盘值。例如，如果您按下 Shift+2，则在日文键盘上 `Key.getAscii()` 将返回 `@`，就如同该方法在英文键盘上的返回结果一样。

Key.getCode()

可用性

Flash Player 5。

用法

```
Key.getCode();
```

参数

无。

返回

表示上次所按键的键控代码的整数。

说明

方法；返回按下的最后一个键的键控代码值。若要将返回的键控代码值与标准键盘上的键相匹配，请参见[第 745 页的附录 C “键盘键和键控代码值”](#)。

Key.HOME

可用性

Flash Player 5。

用法

```
Key.HOME
```

说明

属性；与 Home 键的键控代码值 (36) 关联的常量。

Key.INSERT

可用性

Flash Player 5。

用法

`Key.INSERT`

说明

属性；与 Insert 键的键控代码值 (45) 关联的常量。

Key.isDown()

可用性

Flash Player 5。

用法

`Key.isDown(keycode)`

参数

keycode 分配给特定键的键控代码值，或与特定键相关联的 Key 类属性。若要将返回的键控代码值与标准键盘上的键相匹配，请参见[第 745 页的附录 C “键盘键和键控代码值”](#)。

返回

一个布尔值。

说明

方法；如果按 *keycode* 中指定的键，则返回 `true`；否则，返回 `false`。在 Macintosh 上，Caps Lock 键和 Num Lock 键的键控代码值相同。

示例

以下脚本使用户可以控制影片剪辑的位置。

```
onClipEvent (enterFrame) {  
    if(Key.isDown(Key.RIGHT)) {  
        this._x=_x+10;  
    } else if (Key.isDown(Key.DOWN)) {  
        this._y=_y+10;  
    }  
}
```

Key.isToggled()

可用性

Flash Player 5。

用法

`Key.isToggled(keycode)`

参数

keycode Caps Lock 的键控代码 (20) 或 Num Lock 的键控代码 (144)。

返回

一个布尔值。

说明

方法；在 Caps Lock 键或 Num Lock 键处于激活状态（被切换）时返回 `true`，否则返回 `false`。在 Macintosh 上，Caps Lock 键和 Num Lock 键的键控代码值相同。

Key.LEFT

可用性

Flash Player 5。

用法

`Key.LEFT`

说明

属性；与左箭头键的键控代码值 (37) 关联的常量。

Key.onKeyDown

可用性

Flash Player 6。

用法

`someListener.onKeyDown`

说明

侦听器；当按下某按键时获得通知。若要使用 `onKeyDown`，必须创建侦听器对象。然后可以为 `onKeyDown` 定义一个函数，并使用 `addListener()` 向 `Key` 对象注册该侦听器，如下所示：

```
someListener = new Object();
someListener.onKeyDown = function () { ... };
Key.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

另请参见

[Key.addListener\(\)](#)

Key.onKeyUp

可用性

Flash Player 6。

用法

someListener.onKeyUp

说明

侦听器；当释放某按键时获得通知。若要使用 onKeyUp，必须创建侦听器对象。然后可以为 onKeyUp 定义一个函数，并使用 addListener() 向 Key 对象注册该侦听器，如下所示：

```
someListener = new Object();
someListener.onKeyUp = function () { ... };
Key.addListener(someListener);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

另请参见

[Key.addListener\(\)](#)

Key.PGDN

可用性

Flash Player 5。

用法

Key.PGDN

说明

属性；与 Page Down 键的键控代码值 (34) 关联的常量。

Key.PGUP

可用性

Flash Player 5。

用法

Key.PGUP

说明

属性；与 Page Up 键的键控代码值 (33) 关联的常量。

Key.removeListener()

可用性

Flash Player 6。

用法

```
Key.removeListener (listener)
```

参数

listener 对象。

返回

如果成功删除了 *listener*，则该方法返回 `true`。如果未能成功删除 *listener*（例如，如果 *listener* 不在 `Key` 对象的侦听器列表上），则该方法返回 `false`。

说明

方法；删除以前用 `Key.addListener()` 注册的对象。

Key.RIGHT

可用性

Flash Player 5。

用法

```
Key.RIGHT
```

说明

属性；与右箭头键的键控代码值 (39) 关联的常量。

Key.SHIFT

可用性

Flash Player 5。

用法

```
Key.SHIFT
```

说明

属性；与 Shift 键的键控代码值 (16) 关联的常量。

Key.SPACE

可用性

Flash Player 5。

用法

Key.SPACE

说明

属性；与空格键的键控代码值 (32) 关联的常量。

Key.TAB

可用性

Flash Player 5。

用法

Key.TAB

说明

属性；与 Tab 键的键控代码值 (9) 关联的常量。

Key.UP

可用性

Flash Player 5。

用法

Key.UP

说明

属性；与向上箭头键的键控代码值 (38) 关联的常量。

le（小于或等于 - 字符串专用）

可用性

Flash Player 4。不鼓励在 Flash 5 中使用此运算符，而推荐使用 `<=`（小于等于）运算符。

用法

expression1 le *expression2*

参数

expression1、*expression2* 数字、字符串或变量。

返回

无。

说明

运算符（比较）；比较 *expression1* 与 *expression2*，如果 *expression1* 小于或等于 *expression2*，则返回值 `true`；否则，返回 `false` 值。

另请参见

[<=（小于等于）](#)

length

可用性

Flash Player 4。不鼓励在 Flash 5 中使用此函数及所有字符串函数。Macromedia 建议使用 `String` 类的方法以及 `String.length` 属性来执行相同的操作。

用法

`length(expression)`

`length(variable)`

参数

expression 字符串。

variable 变量的名称。

返回

指定字符串或变量名称的长度。

说明

字符串函数；返回指定字符串或变量名称的长度。

示例

下面的示例返回字符串 "Hello" 的值。

```
length("Hello");
```

结果为 5。

另请参见

[" "](#)（字符串分隔符），[String](#) 类，[String.length](#)

`_level`

可用性

Flash Player 4。

用法

`_levelN`

说明

标识符；对 `_levelN` 的根时间轴的引用。必须在使用 `loadMovieNum()` 将 SWF 文件加载到 Flash Player 中以后，才可使用 `_level` 属性来定位这些 SWF。还可使用 `_levelN` 来定位由 *N* 所指定级别处的已加载 SWF 文件。

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

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

示例

下面的示例将播放头停止在位于 `_level9` 中的 SWF 文件的主时间轴中。

```
_level9.stop();
```

下面的示例将位于 `_level4` 中的 SWF 文件的主时间轴中的播放头转到第 5 帧。在此之前，处于 `_level4` 中的 SWF 文件必须已经用 `loadMovieNum()` 动作加载。

```
_level4.gotoAndStop(5);
```

另请参见

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

`loadMovie()`

可用性

Flash Player 3。

用法

```
loadMovie("url",target [, method])
```

参数

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

target 指向目标影片剪辑的路径。目标影片剪辑将替换为加载的 SWF 文件或图像。

method 可选参数，指定用于发送变量的 HTTP 方法。该参数必须是字符串 `GET` 或 `POST`。如果没有要发送的变量，则省略此参数。`GET` 方法将变量追加到 URL 的末尾，它用于发送少量的变量。`POST` 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

返回

无。

说明

函数；在播放原始 SWF 文件的同时将 SWF 文件或 JPEG 文件加载到 Flash Player 中。

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

使用 `loadMovie()` 函数可以一次显示几个 SWF 文件并且无需加载另一个 HTML 文档即可在 SWF 文件间进行切换。如果不使用 `loadMovie()` 函数，则 Flash Player 显示单个 SWF 文件，然后关闭。

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

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

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

示例

下面的 `loadMovie()` 语句附加到标签为 `Products` 的导航按钮。在舞台上有一个实例名称为 `dropZone` 的不可见影片剪辑。`loadMovie()` 函数使用此影片剪辑作为目标参数将 SWF 文件中的产品加载到舞台上的正确位置。

```
on (release) {  
    loadMovie("products.swf",_root.dropZone);  
}
```

下面的示例从特定目录中加载一个 JPEG 图像，该目录与调用 `loadMovie()` 函数的 SWF 文件的目录相同：

```
loadMovie("image45.jpeg", "ourMovieClip");
```

另请参见

`_level`, `loadMovieNum()`, `MovieClipLoader.loadClip()`, `unloadMovie()`

loadMovieNum()

可用性

Flash Player 4。用 Flash 5 或更高版本打开的 Flash 4 文件将进行转换，以使用正确的语法。

用法

```
loadMovieNum("url",level [, variables])
```

参数

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

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

variables 可选参数，指定发送变量所使用的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

返回

无。

说明

函数；在播放原来加载的 SWF 文件的同时将 SWF 文件或 JPEG 文件加载到 Flash Player 中的某个级别。

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

一般情况下，Flash Player 显示单个 SWF 文件，然后关闭。`loadMovieNum()` 动作使您可以同时显示几个 SWF 文件，并且无需加载另一个 HTML 文档就可在 SWF 文件之间切换。

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

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

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

`loadMovieNum()` 动作也允许您在播放 SWF 文件时将 JPEG 文件加载到 SWF 文件中。对于图像和 SWF 文件，在文件加载时，图像的左上角均与舞台的左上角对齐。另外，在这两种情况下，加载的文件均继承旋转和缩放设置，并且初始内容将被覆盖。

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

示例

此示例将 JPEG 图像 “image45.jpg” 加载到 Flash Player 的级别 2 中。

```
loadMovieNum("http://www.blag.com/image45.jpg", 2);
```

另请参见

`loadMovie()`, `unloadMovieNum()`, `_level`

loadVariables()

可用性

Flash Player 4；行为在 Flash Player 7 中发生了变化。

用法

```
loadVariables ( "url" , target [, variables] )
```

参数

url 变量所处位置的绝对或相对 URL。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

target 指向接收所加载变量的影片剪辑的目标路径。

variables 可选参数，指定发送变量所使用的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

返回

无。

说明

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

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

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

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 www.someDomain.com 的 SWF 文件可以从位于 store.someDomain.com 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 someDomain.com 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“[Flash Player 安全功能](#)”）。例如，位于 www.someDomain.com 的 SWF 文件只能从同样位于 www.someDomain.com 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“[关于允许跨域数据加载](#)”。

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

示例

此示例将文本文件中的信息加载到主时间轴上 `varTarget` 影片剪辑的文本字段中。文本字段的变量名必须与 `data.txt` 文件中的变量名匹配。

```
on (release) {  
    loadVariables("data.txt", "_root.varTarget");  
}
```

另请参见

[loadVariablesNum\(\)](#), [loadMovie\(\)](#), [loadMovieNum\(\)](#), [getURL\(\)](#), [MovieClip.loadMovie\(\)](#), [MovieClip.loadVariables\(\)](#)

loadVariablesNum()

可用性

Flash Player 4。用 Flash 5 或更高版本打开的 Flash 4 文件将进行转换，以使用正确的语法。行为在 Flash Player 7 中已更改。

用法

```
loadVariablesNum ("url" ,level [, variables])
```

参数

url 变量所处位置的绝对或相对 URL。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

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

variables 可选参数，指定发送变量所使用的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

返回

无。

说明

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

指定 URL 处的文本必须为标准的 MIME 格式 `application/x-www-form-urlencoded`（CGI 脚本所使用的一种标准格式）。可以指定任意数量的变量。例如，下面的语句定义了几个变量：

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

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 `someDomain.com` 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“[Flash Player 安全功能](#)”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“[关于允许跨域数据加载](#)”。

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

示例

此示例将来自文本文件中的信息加载到 Flash Player 中级别 0 处的 SWF 文件的主时间轴上的文本字段中。文本字段的变量名必须与 `data.txt` 文件中的变量名匹配。

```
on (release) {  
    loadVariablesNum("data.txt", 0);  
}
```

另请参见

```
getUrl(), loadMovie(), loadMovieNum(), loadVariables(), MovieClip.loadMovie(),  
MovieClip.loadVariables()
```

LoadVars 类

可用性

Flash Player 6。

说明

LoadVars 类是 `loadVariables()` 函数的替代方法，用于在 Flash 应用程序和服务器之间传输变量。

您可以使用 LoadVars 类在下载时获取成功数据加载、进度指示和流数据的验证信息。LoadVars 类的工作原理非常类似于 [XML 类](#)；它使用方法 `load()`、`send()` 和 `sendAndLoad()` 与服务器进行通讯。LoadVars 类和 XML 类之间的主要差别在于 LoadVars 传输动作脚本的名称和值对，而不是 XML 对象中存储的 XML DOM 树。

LoadVars 类与 XML 类遵循相同的安全限制。

LoadVars 类的方法概要

方法	说明
<code>LoadVars.setRequestHeader()</code>	添加或更改 POST 操作的 HTTP 标头。
<code>LoadVars.getBytesLoaded()</code>	返回 <code>LoadVars.load()</code> 或 <code>LoadVars.sendAndLoad()</code> 所下载的字节数。
<code>LoadVars.getBytesTotal()</code>	返回将通过 <code>load</code> 或 <code>sendAndLoad</code> 方法下载的总字节数。
<code>LoadVars.load()</code>	从指定的 URL 下载变量。
<code>LoadVars.send()</code>	将变量从 LoadVars 对象发布到 URL。
<code>LoadVars.sendAndLoad()</code>	将变量从 LoadVars 对象发布到 URL，并将服务器的响应下载到目标对象。
<code>LoadVars.toString()</code>	返回包含 LoadVars 对象中所有可枚举变量的 URL 编码字符串。

LoadVars 类的属性概要

属性	说明
<code>LoadVars.contentType</code>	指示数据的 MIME 类型。
<code>LoadVars.loaded</code>	指示 <code>load</code> 或 <code>sendAndLoad</code> 操作是否已完成的布尔值。

LoadVars 类的事件处理函数概要

事件处理函数	说明
<code>LoadVars.onData</code>	当数据从服务器上完全下载时，或者当从服务器下载数据的过程中出现错误时调用。
<code>LoadVars.onLoad</code>	当完成 <code>load</code> 或 <code>sendAndLoad</code> 操作时调用。

LoadVars 类的构造函数

可用性

Flash Player 6。

用法

```
new LoadVars()
```

参数

无。

返回

无。

说明

构造函数；创建 LoadVars 对象。然后您可使用该 LoadVars 对象的方法来发送和加载数据。

示例

下面的示例创建一个名为 my_lv 的 LoadVars 的对象：

```
var my_lv = new LoadVars();
```

LoadVars.setRequestHeader()

可用性

Flash Player 6。

用法

```
my_lv.setRequestHeader(headerName, headerValue)  
my_lv.setRequestHeader(["headerName_1", "headerValue_1" ... "headerName_n",  
    "headerValue_n"])
```

参数

headerName HTTP 请求标头名称。

headerValue 与 *headerName* 关联的值。

返回

无。

说明

方法，添加或更改用 POST 动作发送的 HTTP 请求标头（如 Content-Type 或 SOAPAction）。在第一种用法中，向该方法传递了两个字符串：*headerName* 和 *headerValue*。在第二种用法中，传递了字符串、替代标头名称和标头值的数组。

如果通过多次调用来设置相同的标头名称，则每个后继值将替换在上一次调用中设置的值。

不能 使用此方法添加或更改以下标准 HTTP 标头 : Accept-Ranges、Age、Allow、Allowed、Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、Vary、Via、Warning 和 WWW-Authenticate。

示例

以下示例将值为 Foo 的自定义 HTTP 标头 SOAPAction 添加到 my_lv 对象。

```
my_lv.setRequestHeader("SOAPAction", "'Foo'");
```

下一个示例创建名为 headers 的数组, 它包含两个替代 HTTP 标头及其关联值。该数组作为参量传递给 addRequestHeader()。

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];  
my_lv.setRequestHeader(headers);
```

另请参见

[XML.setRequestHeader\(\)](#)

LoadVars.contentType

可用性

Flash Player 6。

用法

```
my_lv.contentType
```

说明

属性 ; 当您调用 [LoadVars.send\(\)](#) 或 [LoadVars.sendAndLoad\(\)](#) 时发送到服务器的 MIME 类型。默认值为 application/x-www-form-urlencoded。

另请参见

[LoadVars.send\(\)](#), [LoadVars.sendAndLoad\(\)](#)

LoadVars.getBytesLoaded()

可用性

Flash Player 6。

用法

```
my_lv.getBytesLoaded()
```

参数

无。

返回

一个整数。

说明

方法；返回 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 所下载的字节数。如果当前没有执行加载操作或者尚未开始加载操作，此方法将返回 `undefined`。

LoadVars.getBytesTotal()

可用性

Flash Player 6。

用法

```
my_lv.getBytesTotal()
```

参数

无。

返回

一个整数。

说明

方法；返回 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 所下载的总字节数。如果当前没有执行加载操作或者尚未开始加载操作，此方法将返回 `undefined`。如果无法确定总字节数（例如，如果下载已开始但服务器尚未传输 HTTP Content-Length 标头），则此方法也会返回 `undefined`。

LoadVars.load()

可用性

Flash Player 6 ；行为在 Flash Player 7 中发生了变化。

用法

```
my_lv.load(url)
```

参数

url 要从中下载变量的 URL。如果发出此调用的 SWF 文件正在 Web 浏览器中运行，则 *url* 必须与该 SWF 文件位于同一域中；有关详细信息，请参见下面的“说明”。

返回

字符串。

说明

方法；从指定的 URL 下载变量，分析变量数据，并将结果变量放在 *my_lv* 中。*my_lv* 中任何与下载变量同名的属性都将被覆盖。*my_lv* 中任何与下载变量不同名的属性都不会被删除。这是一个异步动作。

下载的数据必须是 MIME 内容类型 `application/x-www-form-urlencoded`。这种格式与 `loadVariables()` 所用的格式相同。

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 `someDomain.com` 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“Flash Player 安全功能”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

此外，在以 Flash Player 7 为目标播放器发布的文件中，使用 `LoadVars.load()` 加载的外部变量支持区分大小写（请参见第 26 页的“区分大小写”）。

此方法与 `XML.load()` 相似。

LoadVars.loaded

可用性

Flash Player 6。

用法

```
my_lv.loaded
```

说明

属性；默认情况下为未定义。当 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 操作开始时，`loaded` 属性设置为 `false`；当该操作完成时，`loaded` 属性设置为 `true`。如果该操作尚未完成或由于错误而失败，则 `loaded` 属性仍保持设置为 `false`。

此属性与 `XML.loaded` 属性相似。

LoadVars.onData

可用性

Flash Player 6。

用法

```
my_lv.onData = function(src) {  
    // 此处是您的语句  
}
```

参数

`src` 来自 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 方法调用的原始（未分析）数据。

返回

无。

说明

事件处理函数；当数据从服务器上完全下载时，或者当从服务器下载数据的过程中出现错误时调用。此处理函数在分析数据之前调用，因此它可用于调用自定义分析例程，而不必调用 Flash Player 中的内置分析例程。对于分配给 `LoadVars.onData` 的函数，传递给它的 `src` 参数的值可以是 `undefined`，或者包含从服务器下载的 URL 编码名称 / 值的字符串。如果返回值为 `undefined`，则说明从服务器下载数据时出现了错误。

`LoadVars.onData` 的默认实现调用 `LoadVars.onLoad`。您可以通过将自定义函数分配给 `LoadVars.onData` 来覆盖此默认实现，但 `LoadVars.onLoad` 将不再被调用，除非您在 `LoadVars.onData` 的实现中调用它。

LoadVars.onLoad

可用性

Flash Player 6。

用法

```
my_lv.onLoad = function(success) {  
    // 此处是您的语句  
}
```

参数

success 该参数指示加载操作是成功完成 (true) 还是以失败结束 (false)。

返回

一个布尔值。

说明

事件处理函数；当 `LoadVars.load()` 或 `LoadVars.sendAndLoad()` 操作结束时调用。如果该操作成功，`my_lv` 将填充为该操作所下载的变量，而这些变量将在调用此处理函数时变为可用。

默认情况下该处理函数未定义。

此方法与 `XML.onLoad()` 相似。

LoadVars.send()

可用性

Flash Player 6。

用法

```
my_lv.send(url [,target,method])
```

参数

url 要将变量上传到的 URL。

target 浏览器框架窗口，所有响应都将在其中显示。

method HTTP 协议的 GET 或 POST 方法。

返回

字符串。

说明

方法；将 `my_lv` 对象中的变量发送到指定的 URL。默认情况下，`my_lv` 中的所有可枚举变量都将连接为一个 `application/x-www-form-urlencoded` 格式的字符串，该字符串将通过 HTTP POST 方法被发布到 URL。这与 `loadVariables()` 动作所用的格式相同。在 HTTP 请求标头中发送的 MIME 内容类型是 `my_lv.contentType` 的值或默认的 `application/x-www-form-urlencoded`。除非指定了 GET，否则将使用 POST 方法。

如果指定了 *target* 参数，则服务器的响应将显示在指定的目标浏览器框架窗口中。如果省略 *target* 参数，则服务器响应将被丢弃。

此方法与 `XML.send()` 相似。

LoadVars.sendAndLoad()

可用性

Flash Player 6 ；行为在 Flash Player 7 中发生了变化。

用法

```
my_lv.sendAndLoad(url, targetObject[, method])
```

参数

url 要将变量上传到的 URL。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

targetObject 接收下载变量的 LoadVars 对象。

method HTTP 协议的 GET 或 POST 方法。

返回

字符串。

说明

方法：将 *my_lv* 对象中的变量发布到指定的 URL。将下载服务器响应，并将其作为变量数据进行分析，然后将结果变量放在 *targetObject* 对象中。

变量发布的方式与 `LoadVars.send()` 相同。变量下载到 *targetObject* 中的方式与 `LoadVars.load()` 相同。

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 `someDomain.com` 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“[Flash Player 安全功能](#)”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“[关于允许跨域数据加载](#)”。

此方法与 `XML.sendAndLoad()` 相似。

LoadVars.toString()

可用性

Flash Player 6。

用法

```
my_lv.toString()
```

参数

无。

返回

字符串。

说明

方法；以 MIME 内容编码格式 application/x-www-form-urlencoded 返回包含 `my_lv` 对象中所有可枚举变量的字符串。

示例

```
var myVars = new LoadVars();
myVars.name =   ary?
myVars.age = 26;
trace (myVars.toString());
// 将输出
//name=Gary&age=26
```

LocalConnection 类

可用性

Flash Player 6。

说明

`LocalConnection` 类用于开发 SWF 文件，这些文件无需使用 `fscommand()` 或 JavaScript 即可相互发送指令。`LocalConnection` 对象只能在运行于同一台客户机上的 SWF 文件之间通讯，但这些 SWF 文件可以在两个不同的应用程序中运行。例如，一个 SWF 文件在浏览器中运行，而一个 SWF 文件在播放器中运行。您可以使用 `LocalConnection` 对象在单个 SWF 文件中发送和接收数据，但这不是标准的实现；本节中的所有示例均说明不同 SWF 文件之间的通讯。

用来发送和接收数据的主要方法是 `LocalConnection.send()` 和 `LocalConnection.connect()`。在最基础的层次，您的代码将实现以下命令；请注意 `LocalConnection.send()` 和 `LocalConnection.connect()` 命令均指定相同的连接名称 `lc_name`：

```
// 接收方影片中的代码
receiving_lc = new LocalConnection();
receiving_lc.methodToExecute = function(param1, param2)
{
    // 要执行的代码
}
receiving_lc.connect("lc_name");
// 发送方影片中的代码
sending_lc = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", dataItem1, dataItem2)
```

`LocalConnection` 对象最简单的使用方法是仅允许在位于同一个域中的 `LocalConnection` 对象之间进行通讯，因为您将不需要处理与安全相关的问题。但是，如果您需要允许域间通讯，则可以通过多种方法来实现安全措施。有关更多信息，请参见 `LocalConnection.send()` 中对 `connectionName` 参数的讨论，以及 `LocalConnection.allowDomain` 和 `LocalConnection.domain()` 条目。

LocalConnection 类的方法概述

方法	说明
<code>LocalConnection.close()</code>	关闭（断开连接） <code>LocalConnection</code> 对象。
<code>LocalConnection.connect()</code>	准备 <code>LocalConnection</code> 对象，以从 <code>LocalConnection.send()</code> 命令中接收命令。

方法	说明
<code>LocalConnection.domain()</code>	返回一个字符串，表示当前 SWF 文件位置的超域。
<code>LocalConnection.send()</code>	对指定的 LocalConnection 对象调用方法。

LocalConnection 类的事件处理函数概要

事件处理函数	说明
<code>LocalConnection.allowDomain</code>	每当当前（接收方）LocalConnection 对象从发送方 LocalConnection 对象收到调用方法的请求时调用。
<code>LocalConnection.allowInsecureDomain</code>	每当当前（接收方）LocalConnection 对象（位于使用安全协议 HTTPS 的域承载的 SWF 文件中）从发送方 LocalConnection 对象（位于使用非安全协议的域承载的 SWF 文件中）收到调用方法的请求时调用。
<code>LocalConnection.onStatus</code>	当发送方 LocalConnection 对象尝试将命令发送到接收方 LocalConnection 对象之后调用。

LocalConnection 类的构造函数

可用性

Flash Player 6。

用法

`new LocalConnection()`

参数

无。

返回

无。

说明

构造函数；创建 LocalConnection 对象。

示例

以下示例显示接收方 SWF 文件和发送方 SWF 文件如何创建 LocalConnection 对象。请注意，这两个 SWF 文件可以为各自的 LocalConnection 对象使用相同的名称或不同的名称。在此示例中，它们使用相同的名称 `my_lc`。

```
// 接收方 SWF 中的代码
my_lc = new LocalConnection();
my_lc.someMethod = function() {
    // 此处是您的语句
}
my_lc.connect("connectionName");

// 发送方 SWF 中的代码
my_lc = new LocalConnection();
my_lc.send("connectionName", "someMethod");
```

另请参见

[LocalConnection.connect\(\)](#), [LocalConnection.send\(\)](#)

LocalConnection.allowDomain

可用性

Flash Player 6 ; 行为在 Flash Player 7 中进行了更改。

用法

```
receiving_lc.allowDomain = function([sendingDomain]) {  
    // 您在此处输入的语句返回 true 或 false  
}
```

参数

sendingDomain 可选参数, 指定包含发送方 LocalConnection 对象的 SWF 文件的域。

返回

无。

说明

事件处理函数 ; 每当 *receiving_lc* 从发送方 LocalConnection 对象收到调用方法的请求时调用。Flash 需要您在此处理函数中实现的代码返回布尔值 true 或 false。如果此处理函数没有返回 true, 则将忽略发送方对象的请求, 并且不调用方法。

使用此命令可从指定域或任何域中显式允许 LocalConnection 对象执行接收方 LocalConnection 对象的方法。如果没有声明 *sendingDomain* 参数, 则可能需要接受任何域中的命令, 并且处理函数中的代码将只为 return true。如果您声明了 *sendingDomain*, 则可能需要将 *sendingDomain* 的值与要从中接受命令的域进行比较。以下示例将说明这两种实现方式。

在 Flash Player 6 中运行的文件中, *sendingDomain* 参数包含调用方的超域。在 Flash Player 7 或更高版本中运行的文件中, *sendingDomain* 参数包含该调用方的确切的域。在后一种情况中, 若要允许由在 [www.domain.com](#) 或 [store.domain.com](#) 承载的 SWF 文件访问, 您必须显式允许从这两个域的访问。

```
// 对于 Flash Player 6  
receiving_lc.allowDomain = function(sendingDomain) {  
    return(sendingDomain=="domain.com");  
}  
// 允许运行于 Flash Player 7 或更高版本中的 SWF 文件  
// 进行访问的对应命令  
receiving_lc.allowDomain = function(sendingDomain) {  
    return(sendingDomain=="www.domain.com" ||  
           sendingDomain=="store.domain.com");  
}
```

此外, 对于在 Flash Player 7 或更高版本中运行的文件, 您不能使用此方法允许使用安全协议 (HTTPS) 承载的 SWF 文件可以从使用非安全协议承载的 SWF 文件访问 ; 必须改用 [LocalConnection.allowInsecureDomain](#) 事件处理函数。

示例

以下示例显示接收方 SWF 文件中的 `LocalConnection` 对象如何允许任何域中的 SWF 文件调用其方法。将其与 `LocalConnection.connect()` 中的示例进行比较，在该示例中，只有来自同一个域中的 SWF 文件才能调用接收方 SWF 文件中的 `Trace` 方法。有关下划线 () 在连接名称中的用法的讨论，请参见 `LocalConnection.send()`。

```
var aLocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString)
{
    aTextField = aTextField + aString + newline;
}
```

```
aLocalConnection.allowDomain = function() {
    // 任何域均可对此 LocalConnection 对象调用方法
    return true;
}
```

```
aLocalConnection.connect("_trace");
```

在以下示例中，接收方 SWF 文件只接受来自位于 `thisDomain.com` 或 `thatDomain.com` 中的 SWF 文件的命令。

```
var aLocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString)
{
    aTextField = aTextField + aString + newline;
}

aLocalConnection.allowDomain = function(sendingDomain)
{
    return(sendingDomain=="thisDomain.com" || sendingDomain=="thatDomain.com");
}

aLocalConnection.connect("_trace");
```

另请参见

[LocalConnection.connect\(\)](#), [LocalConnection.domain\(\)](#), [LocalConnection.send\(\)](#)

LocalConnection.allowInsecureDomain

可用性

Flash Player 7。

用法

```
receiving_lc.allowInsecureDomain = function([sendingDomain]) {
    // 您在此处输入的语句返回 true 或 false
}
```

参数

sendingDomain 可选参数，指定包含发送方 `LocalConnection` 对象的 SWF 文件的域。

返回

无。

说明

事件处理函数；每当 `receiving_lc`（位于使用安全协议 HTTPS 的域承载的 SWF 文件中）从发送方 `LocalConnection` 对象（位于使用非安全协议的域承载的 SWF 文件中）收到调用方法的请求时调用。Flash 需要您在此处理函数中实现的代码返回布尔值 `true` 或 `false`。如果此处理函数没有返回 `true`，则将忽略发送方对象的请求，并且不调用方法。

默认情况下，使用 HTTPS 协议承载的 SWF 文件只能被其它使用 HTTPS 协议承载的 SWF 文件访问。这种实现保持了 HTTPS 协议所提供的完整性。

不建议使用此方法覆盖默认行为，因为这样做会损及 HTTPS 安全。但在某些情况下您可能需要这样做；例如，您可能需要允许从以 Flash Player 6 为目标播放器发布的 HTTP 文件访问以 Flash Player 7 或更高版本为目标播放器发布的 HTTPS 文件。

以 Flash Player 6 为目标播放器发布的 SWF 文件可以使用 `LocalConnection.allowDomain` 事件处理函数允许 HTTP 到 HTTPS 访问。但是，因为 Flash Player 7 中实现安全的方式是不同的，所以，您必须使用 `LocalConnection.allowInsecureDomain()` 方法在以 Flash Player 7 或更高版本为目标播放器发布的 SWF 文件中允许此类访问。

另请参见

[LocalConnection.allowDomain](#), [LocalConnection.connect\(\)](#)

LocalConnection.close()

可用性

Flash Player 6。

用法

```
receiving_lc.close
```

参数

无。

返回

无。

说明

方法；关闭（断开连接）`LocalConnection` 对象。在您不再想让该对象接受命令时（例如，当您使用另一个 SWF 文件中的相同 `connectionName` 参数发出 `LocalConnection.connect()` 命令时）发出此命令。

另请参见

[LocalConnection.connect\(\)](#)

LocalConnection.connect()

可用性

Flash Player 6。

用法

```
receiving_lc.connect(connectionName)
```

参数

connectionName 一个字符串，对应于要与 *receiving_lc* 通讯的 [LocalConnection.send\(\)](#) 命令中所指定的连接名称。

返回

如果在同一台客户机上运行的其它进程均未使用 *connectionName* 参数的相同值发出此命令，则返回布尔值 `true`；否则返回 `false`。

说明

方法；准备 `LocalConnection` 对象，以从 [LocalConnection.send\(\)](#) 命令（称作“发送方 `LocalConnection` 对象”）中接收命令。与此命令一起使用的对象称作“接收方 `LocalConnection` 对象”。接收方对象和发送方对象必须在同一台客户机上运行。

务必要先定义附加到 *receiving_lc* 的方法，然后再调用此方法，如本节的所有示例所示。

默认情况下，Flash Player 将 *connectionName* 解析为 "*superdomain:connectionName*" 格式的值，其中 *superdomain* 是包含 [LocalConnection.connect\(\)](#) 命令的 SWF 文件的超域。例如，如果包含接收方 `LocalConnection` 对象的 SWF 文件位于 `www.someDomain.com`，则 *connectionName* 解析为 "`someDomain.com:connectionName`"。（如果 SWF 文件位于客户机上，则赋予 *superdomain* 的值为 "`localhost`"。）

此外，在默认情况下，Flash Player 只允许接收方 `LocalConnection` 对象从其连接名称也解析为 "*superdomain:connectionName*" 的值的发送方 `LocalConnection` 对象中接受命令。这样，Flash 就使得位于同一个域中的 SWF 文件可以很容易地相互通讯。

如果您仅在同一个域中的 SWF 文件之间实现通讯，请为 *connectionName* 指定一个不以下划线 (`_`) 开头且不指定域名的字符串（例如 "`myDomain:connectionName`"）。在 [LocalConnection.connect\(connectionName\)](#) 命令中使用相同的字符串。

如果您在位于不同域的 SWF 文件之间实现通讯，请参见 [LocalConnection.send\(\)](#) 中对 *connectionName* 的讨论，以及 [LocalConnection.allowDomain](#) 和 [LocalConnection.domain\(\)](#) 条目。

示例

以下示例显示特定域中的 SWF 文件如何调用同一域的接收方 SWF 文件中名为 `Trace` 的方法。接收方 SWF 文件充当发送方 SWF 文件的跟踪窗口；它包含其它 SWF 文件可以调用的两个方法：`Trace` 和 `Clear`。在发送方 SWF 文件中所按的按钮以指定的参数调用这些方法。

```
// 接收方 SWF
var aLocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString)
{
    aTextField = aTextField + aString + newline;
}
aLocalConnection.Clear = function()
{

```

```

    aTextField = "";
}
aLocalConnection.connect("trace");
stop();

```

SWF 1 包含附加到带 PushMe 标签的按钮的以下代码。当您按该按钮时，将在接收方 SWF 文件中看到句子“已按下该按钮”。

```

on (press)
{
    var lc = new LocalConnection();
    lc.send("trace", "Trace", " 已按下该按钮。");
    delete lc;
}

```

SWF 2 包含变量名为 myText 的输入文本框以及附加到带标签 Copy 的按钮的以下代码。在您键入一些文本并按该按钮后，将在接收方 SWF 文件中看到您键入的文本。

```

on (press)
{
    _parent.lc.send("trace", "Trace", _parent.myText);
    _parent.myText = "";
}

```

SWF 3 包含附加到带标签 Clear 的按钮的以下代码。当您按该按钮时，将清除（擦除）接收方 SWF 文件的跟踪窗口中的内容。

```

on (press)
{
    var lc = new LocalConnection();
    lc.send("trace", "Clear");
    delete lc;
}

```

另请参见

[LocalConnection.send\(\)](#)

LocalConnection.domain()

可用性

Flash Player 6 ；行为在 Flash Player 7 中进行了更改。

用法

```
my_lc.domain()
```

参数

无。

返回

一个字符串，表示当前 SWF 文件的位置的域；有关详细信息，请参见下面的“说明”。

说明

方法；返回表示当前 SWF 文件所在位置的域的字符串。

在以 Flash Player 6 为目标播放器发布的 SWF 文件中，返回的字符串是当前 SWF 文件的超域。例如，如果 SWF 文件位于 www.macromedia.com，则此命令将返回 "macromedia.com"。

在以 Flash Player 7 或更高版本为目标播放器发布的 SWF 文件中，返回的字符串是当前 SWF 文件的确切域。例如，如果 SWF 文件位于 www.macromedia.com，则此命令将返回 "www.macromedia.com"。

如果当前 SWF 文件是驻留在客户机上的本地文件，此命令将返回 "localhost"。

此命令最常见的用法是包含发送方 `LocalConnection` 对象的域名作为要在接收方 `LocalConnection` 对象中调用的方法的参数，或者与 `LocalConnection.allowDomain` 一起使用来接受来自指定域中的命令。如果您仅启用位于同一个域的 `LocalConnection` 对象之间的通讯，则可能不需要使用此命令。

示例

在以下示例中，接收方 SWF 文件仅接受来自位于同一个域或位于 macromedia.com 的 SWF 文件的命令。

```
my_lc = new LocalConnection();
my_lc.allowDomain = function(sendingDomain)
{
    return (sendingDomain==this.domain() || sendingDomain=="macromedia.com");
}
```

在下面的示例中，位于 yourdomain.com 的发送方 SWF 文件调用位于 mydomain.com 的接收方 SWF 文件中的方法。发送方 SWF 文件包含其域名作为它所调用的方法的参数，使接收方 SWF 文件能够返回对正确域中的 `LocalConnection` 对象的应答值。发送方 SWF 文件还指定它将只接受来自位于 mydomain.com 的 SWF 文件的命令。

为了便于参考，代码中包含了行号。事件的顺序如下：

- 接收方 SWF 文件准备在名为 "sum" 的连接（第 11 行）上接收命令。Flash Player 将此连接的名称解析为 "mydomain.com:sum"（请参见 `LocalConnection.connect()`）。
- 发送方 SWF 文件准备在名为 "result" 的 `LocalConnection` 对象（第 58 行）上接收应答。它还指定它将只接受来自位于 mydomain.com 的 SWF 文件的命令（第 51 行到第 53 行）。
- 发送方 SWF 文件调用名为 "mydomain.com:sum" 的连接的 `aSum` 方法（第 59 行），然后传递以下参数：它的域 (`lc.domain()`)、接收应答的连接的名称 ("result") 和 `aSum` 所使用的值 (123 和 456)。
- 使用以下值调用 `aSum` 方法（第 6 行）：`sender = "mydomain.com:result"`、`replyMethod = "aResult"`、`n1 = 123` 和 `n2 = 456`。因此，它执行以下代码行：
`this.send("mydomain.com:result", "aResult", (123 + 456));`
- `aResult` 方法（第 54 行）显示 `aSum` 返回的值 (579)。

```
// 位于 http://www.mydomain.com/folder/movie.swf 的接收方 SWF
// 包含以下代码
```

```
1  var aLocalConnection = new LocalConnection();
2  aLocalConnection.allowDomain = function()
3  {
4      // 允许任何域中的连接
5      return true;
6  }
7  aLocalConnection.aSum = function(sender, replyMethod, n1, n2)
8  {
9      this.send(sender, replyMethod, (n1 + n2));
10 }
11 aLocalConnection.connect("sum");
```

```
// 位于 http://www.yourdomain.com/folder/movie.swf 的发送方 SWF
// 包含以下代码

50 var lc = new LocalConnection();
51 lc.allowDomain = function(aDomain) {
    // 只允许 mydomain.com 中的连接
52     return (aDomain == "mydomain.com");
53 }
54 lc.aResult = function(aParam) {
55     trace("The sum is " + aParam);
56 }
57
58 lc.connect("result");
59 lc.send("mydomain.com:sum", "aSum", lc.domain() + ':' + "result",
    "aResult", 123, 456);
```

另请参见

[LocalConnection.allowDomain](#)

LocalConnection.onStatus

可用性

Flash Player 6。

用法

```
sending_lc.onStatus = function(infoObject) {
    // 此处是您的语句
}
```

参数

infoObject 按照状态消息定义的参数。有关此参数的详细信息，请参见下面的“说明”。

返回

无。

说明

事件处理函数；在发送方 LocalConnection 对象尝试向接收方 LocalConnection 对象发送命令之后调用。如果要对此事件处理函数做出响应，则必须创建一个函数来处理 LocalConnection 对象所发送的信息对象。

如果此事件处理函数返回的信息对象包含 "Status" 的 level 值，则表明 Flash 已将该命令成功发送到接收方 LocalConnection 对象。这并不意味着 Flash 已成功调用接收方 LocalConnection 对象的指定方法，而只表示 Flash 能够发送该命令。例如，如果接收方 LocalConnection 对象不允许从发送方域建立连接，或者该方法不存在，则不调用该方法。确知是否已调用该方法的唯一方式是让接收方对象向发送方对象发送应答。

如果此事件处理函数返回的信息对象包含 "Error" 的 level 值，则表明 Flash 无法将该命令发送到接收方 LocalConnection 对象，这很可能是因为没有连接这样的接收方 LocalConnection 对象：其名称对应于调用此处理函数的 *sending_lc.send()* 命令中所指定的名称。

除了此 onStatus 处理函数外，Flash 还提供称作 [System.onStatus](#) 的“超级”函数。如果为特定对象调用了 onStatus 但未分配任何函数对其进行响应，则 Flash 将处理分配到 System.onStatus 的函数（如果存在）。

大多数情况下，实现此处理函数只是为了对错误条件做出响应，如以下示例所示。

示例

以下示例显示关于“输出”面板中的失败连接的信息：

```
sending_lc = new LocalConnection();
sending_lc.onStatus = function(infoObject)
{
    if (infoObject.level == "Error")
    {
        trace("Connection failed.");
    }
}
sending_lc.send("receiving_lc", "methodName");
```

另请参见

[LocalConnection.send\(\)](#), [System.onStatus](#)

LocalConnection.send()

可用性

Flash Player 6。

用法

```
sending_lc.send (connectionName, method [, p1,...,pN])
```

参数

connectionName 一个字符串，对应于要与 *sending_lc* 通讯的 [LocalConnection.connect\(\)](#) 命令中所指定的连接名称。

method 一个字符串，指定要在接收方 [LocalConnection](#) 对象中调用的方法的名称。以下方法名称会导致该命令失败：[send](#)、[connect](#)、[close](#)、[domain](#)、[onStatus](#) 和 [allowDomain](#)。

p1,...,pN 要传递给指定方法的可选参数。

返回

如果 Flash 可以执行请求，则返回布尔值 `true`；否则，返回 `false`。

注意：返回值为 `true` 并不一定表示 Flash 已成功连接到接收方 [LocalConnection](#) 对象，而只表示该命令在语句构成上正确。若要确定连接是否成功，请参见 [LocalConnection.onStatus](#)。

说明

方法；在用 [LocalConnection.connect\(connectionName\)](#) 命令（称作“接收方 [LocalConnection](#) 对象”）打开的连接上调用名为 *method* 的方法。与此命令一起使用的对象称作“发送方 [LocalConnection](#) 对象”。包含发送方对象的 SWF 文件和包含接收方对象的 SWF 文件必须在同一台客户机上运行。

您能够以参数形式传递给此命令的数据量是有限的。如果该命令返回 `false` 但您的语法却是正确的，请尝试将 [LocalConnection.send\(\)](#) 请求拆分成多个命令。

如 [LocalConnection.connect\(\)](#) 条目所述，Flash 在默认情况下会将当前超域添加到 *connectionName*。如果您在不同的域之间实现通讯，则需要在发送方 [LocalConnection](#) 对象中以及接收方 [LocalConnection](#) 对象中定义 *connectionName*，使 Flash 不会将当前超域添加到 *connectionName*。这可以通过两种方法来实现：

- 在发送方 LocalConnection 对象和接收方 LocalConnection 对象中 `connectionName` 的开头使用下划线 (`_`)。在包含接收方对象的 SWF 文件中，使用 `LocalConnection.allowDomain` 指定将接受来自任何域的连接。这一实现使您可以存储任何域中的发送方 SWF 文件和接收方 SWF 文件。
- 将 `connectionName` 中的超域包含在发送方 LocalConnection 对象中，例如 `myDomain.com:myConnectionName`。在接收方对象中，使用 `LocalConnection.allowDomain` 指定将接受来自指定超域的连接（本例中为 `myDomain.com`），或者将接受来自任何域的连接。

注意：不能在接收方 LocalConnection 对象（只能在发送方 LocalConnection 对象）中指定 `connectionName` 中的超域。

示例

有关位于同一个域中的 LocalConnection 对象之间的通讯的示例，请参见 `LocalConnection.connect()`。有关位于任何域的 LocalConnection 对象之间的通讯的示例，请参见 `LocalConnection.allowDomain`。有关位于指定域的 LocalConnection 对象之间的通讯的示例，请参见 `LocalConnection.allowDomain` 和 `LocalConnection.domain()`。

另请参见

`LocalConnection.allowDomain`, `LocalConnection.connect()`, `LocalConnection.domain()`, `LocalConnection.onStatus`

lt（小于 - 字符串专用）

可用性

Flash Player 4。不鼓励在 Flash 5 中使用此运算符，而推荐使用新的 `<`（小于）运算符。

用法

`expression1 lt expression2`

参数

`expression1`、`expression2` 数字、字符串或变量。

说明

运算符（比较）；比较 `expression1` 与 `expression2`，如果 `expression1` 小于 `expression2`，则返回 `true`；否则，返回 `false`。

另请参见

`<`（小于）

Math 类

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

说明

Math 类是一个顶级类，不必使用构造函数即可使用其方法和属性。

使用该类的方法和属性可以访问和处理数学常数和函数。Math 类的所有属性和方法都是静态的，并且必须使用语法 `Math.method(parameter)` 或 `Math.constant` 来调用。在动作脚本中，使用双精度 IEEE-754 浮点数的最高精度定义常数。

有几个 Math 类的方法使用角的弧度作为参数。可以用下面的等式来计算弧度值，或者只需为弧度参数传递该等式（输入度数值）。

若要计算弧度值，请使用该公式：

```
radian = Math.PI/180 * degree
```

下面的示例将等式作为参数来传递，以计算一个 45 度角的正弦值：

`Math.SIN(Math.PI/180 * 45)` 与 `Math.SIN(.7854)` 相同

Flash Player 5 完全支持 Math 类。在 Flash Player 4 中，可以使用 Math 类的方法，但是这些方法是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

Math 类的方法概要

方法	说明
<code>Math.abs()</code>	计算绝对值。
<code>Math.acos()</code>	计算反余弦值。
<code>Math.asin()</code>	计算反正弦值。
<code>Math.atan()</code>	计算反正切值。
<code>Math.atan2()</code>	计算从 x 坐标轴到点的角度。
<code>Math.ceil()</code>	将数字向上舍入为最接近的整数。
<code>Math.cos()</code>	计算余弦值。
<code>Math.exp()</code>	计算指数值。
<code>Math.floor()</code>	将数字向下舍入为最接近的整数。
<code>Math.log()</code>	计算自然对数。
<code>Math.max()</code>	返回两个整数中较大的一个。
<code>Math.min()</code>	返回两个整数中较小的一个。
<code>Math.pow()</code>	计算 x 的 y 次方。
<code>Math.random()</code>	返回一个 0.0 与 1.0 之间的伪随机数。
<code>Math.round()</code>	四舍五入为最接近的整数。
<code>Math.sin()</code>	计算正弦值。
<code>Math.sqrt()</code>	计算平方根。
<code>Math.tan()</code>	计算正切值。

Math 类的属性概要

Math 类的所有属性都是常数。

属性	说明
Math.E	欧拉 (Euler) 常数，自然对数的底（大约为 2.718）。
Math.LN2	2 的自然对数（大约为 0.693）。
Math.LOG2E	e 的以 2 为底的对数（大约为 1.442）。
Math.LN2	10 的自然对数（大约为 2.302）。
Math.LOG10E	e 的以 10 为底的对数（大约为 0.434）。
Math.PI	一个圆的周长与其直径的比值（大约为 3.14159）。
Math.SQRT1_2	1/2 的平方根的倒数（大约为 0.707）。
Math.SQRT2	2 的平方根（大约为 1.414）。

Math.abs()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.abs(x)`

参数

x 数字。

返回

一个数字。

说明

方法；计算并返回由参数 *x* 指定的数字的绝对值。

Math.acos()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.acos(x)`

参数

x 介于 -1.0 到 1.0 之间的数字。

返回

无。

说明

方法；以弧度为单位计算并返回参数 x 中指定的数字的反余弦值。

Math.asin()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.asin(x);
```

参数

x 介于 -1.0 到 1.0 之间的数字。

返回

一个数字。

说明

方法；以弧度为单位计算并返回参数 x 中指定的数字的反正弦值。

Math.atan()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.atan(x)
```

参数

x 表示角的正切值的数字。

返回

一个数字。

说明

方法；以弧度为单位计算并返回在参数 x 中指定的角的正切值。返回值介于负二分之 π 与正二分之 π 之间。

另请参见

[Math.atan2\(\)](#), [Math.tan\(\)](#)

Math.atan2()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.atan2(y, x)
```

参数

y 指定点的 *y* 坐标的数字。

x 指定点的 *x* 坐标的数字。

返回

一个数字。

说明

方法；当从圆的 *x* 轴逆时针测量时，以弧度为单位计算并返回点 *y/x* 的角（其中 0,0 表示圆心）。返回值介于正 pi 和负 pi 之间。

另请参见

[Math.atan\(\)](#), [Math.tan\(\)](#)

Math.ceil()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.ceil(x)
```

参数

x 数字或表达式。

返回

一个数字。

说明

方法；返回指定数字或表达式的上限值。数字的上限值是大干等于该数字的最接近的整数。

Math.cos()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.cos(x)`

参数

`x` 以弧度为单位的角度值。

返回

一个数字。

说明

方法；返回由参数 `x` 指定的角的余弦值（介于 -1.0 与 1.0 之间的值）。必须以弧度为单位指定角度 `x`。使用 [Math 类](#) 条目中概述的信息来计算弧度。

Math.E

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.E`

参数

无。

返回

无。

说明

常数；代表自然对数的底的数学常数，表示为 `e`。`e` 的近似值为 2.71828。

Math.exp()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.exp(x)
```

参数

x 指数；数字或表达式。

返回

一个数字。

说明

方法；返回 *e* 的 *x* 次方的值，其中 *e* 为自然对数的底 (*e*)，*x* 为参数 *x* 中指定的指数。常数 `Math.E` 可以提供 *e* 的值。

Math.floor()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.floor(x)
```

参数

x 数字或表达式。

返回

一个数字。

说明

方法；返回参数 *x* 中指定的数字或表达式的下限值。下限值是小于等于指定数字或表达式的最接近的整数。

示例

下面的代码返回一个值 12：

```
Math.floor(12.5);
```

Math.log()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.log(x)`

参数

x 其值大于 0 的数字或表达式。

返回

一个数字。

说明

方法；返回参数 x 的对数。

Math.LN2

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.LN2`

参数

无。

返回

无。

说明

常数；代表 2 的自然对数的数学常数，表示为 $\log_e 2$ ，其值大约为 0.69314718055994528623。

Math.LN10

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.LN10`

参数

无。

返回

无。

说明

常数；代表 10 的自然对数的数学常数，表示为 $\log_e 10$ ，其值大约为 2.3025850929940459011。

Math.LOG2E

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

Math.LOG2E

参数

无。

返回

无。

说明

常数；代表常数 e (Math.E) 的以 2 为底的对数的数学常数，表示为 $\log_2 e$ ，其值大约为 1.442695040888963387。

Math.LOG10E

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

Math.LOG10E

参数

无。

返回

无。

说明

常数；代表常数 e (Math.E) 的以 10 为底的对数的数学常数，表示为 $\log_{10} e$ ，其值大约为 0.43429448190325181667。

Math.max()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.max(x , y)
```

参数

x 数字或表达式。

y 数字或表达式。

返回

一个数字。

说明

方法；计算 *x* 和 *y* 并返回较大的值。

Math.min()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.min(x , y)
```

参数

x 数字或表达式。

y 数字或表达式。

返回

一个数字。

说明

方法；计算 *x* 和 *y* 并返回较小的值。

Math.PI

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

Math.PI

参数

无。

返回

无。

说明

常数；代表一个圆的周长与其直径的比值的数学常数，表示为 pi，其值大约为 3.14159265358979。

Math.pow()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

Math.pow(*x* , *y*)

参数

x 将自乘的数字。

y 指定参数 *x* 自乘幂的数字。

返回

一个数字。

说明

方法；计算并返回 *x* 的 *y* 次方： x^y 。

Math.random()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.random()
```

参数

无。

返回

一个数字。

说明

方法；返回 n ，其中 $0 \leq n < 1$ 。

另请参见

[random](#)

Math.round()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.round(x)
```

参数

x 数字。

返回

一个数字。

说明

方法；将参数 x 的值向上或向下舍入为最接近的整数并返回值。

Math.sin()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.sin(x)
```

参数

x 以弧度为单位的角度值。

返回

数字；指定角度的正弦（-1.0 和 1.0 之间）。

说明

方法；计算并返回以弧度为单位指定的角度的正弦值。使用 [Math 类](#) 条目中概述的信息来计算弧度。

Math.sqrt()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.sqrt(x)
```

参数

x 一个大于等于 0 的数字或表达式。

返回

一个数字。

说明

方法；计算并返回指定数字的平方根。

Math.SQRT1_2

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

```
Math.SQRT1_2
```

参数

无。

返回

无。

说明

常数；代表 $1/2$ 的平方根的数学常数。

Math.SQRT2

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.SQRT2`

参数

无。

返回

无。

说明

常数；代表 2 的平方根的数学常数，其值大约为 1.414213562373。

Math.tan()

可用性

Flash Player 5。在 Flash Player 4 中，Math 类的方法和属性是使用近似值模拟的，可能不如 Flash Player 5 支持的非模拟数学函数那样精确。

用法

`Math.tan(x)`

参数

x 以弧度为单位的角度值。

返回

一个数字。

说明

方法；计算并返回指定角度的正切值。若要计算弧度，请使用 [Math](#) 类简介中所概述的信息。

另请参见

[Math.atan\(\)](#)

maxscroll

可用性

Flash Player 4。不鼓励使用该函数，而推荐使用 `TextField.maxscroll` 属性。

用法

```
variable_name.maxscroll
```

说明

属性（只读）；一个不鼓励使用的属性，指示当字段中最底行也可见时，文本字段中最高文本可见行的行号。`maxscroll` 属性与 `scroll` 属性一起使用，来控制文本字段中信息的显示。该属性可被获取，但不能进行修改。

另请参见

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

mbchr

可用性

Flash Player 4。不鼓励使用该函数，而推荐使用 `String.fromCharCode()` 方法。

用法

```
mbchr(number)
```

参数

number 要转换为多字节字符的数字。

返回

字符串。

说明

字符串函数；将 ASCII 码数字转换为多字节字符。

另请参见

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

mblength

可用性

Flash Player 4。不鼓励使用该函数，而推荐使用 `String` 类。

用法

```
mblength(string)
```

参数

string 字符串。

返回

一个数字。

说明

字符串函数；返回多字节字符串的长度。

mbord

可用性

Flash Player 4。不鼓励在 Flash 5 中使用该函数，而推荐使用 [String.charCodeAt\(\)](#)。

用法

```
mbord(character)
```

参数

character 要转换为多字节数字的字符。

返回

一个数字。

说明

字符串函数；将指定的字符转换为多字节数字。

另请参见

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

mbsubstring

可用性

Flash Player 4。不鼓励在 Flash 5 中使用该函数，而推荐使用 [String.substr\(\)](#)。

用法

```
mbsubstring(value, index, count)
```

参数

value 要从其中提取新的多字节字符串的多字节字符串。

index 要提取的第一个字符的编号。

count 要包含在所提取字符串中的字符数，其中不包括 *index* 字符。

返回

字符串。

说明

字符串函数；从多字节字符串中提取一个新的多字节字符串。

另请参见

[String.substr\(\)](#)

Microphone 类

可用性

Flash Player 6。

说明

Microphone 类用于从运行 Flash Player 的计算机上所连接的麦克风中捕获音频。

Microphone 类主要与 Flash Communication Server 一起使用，但在没有该服务器的情况下也能以有限的方式使用，例如通过本地系统上的扬声器传送麦克风中的声音。

若要创建或引用 Microphone 对象，请使用 `Microphone.get()` 方法。

Microphone 类的方法概要

方法	说明
<code>Microphone.get()</code>	返回默认或指定的 Microphone 对象，如果麦克风不可用，则返回 <code>null</code> 。
<code>Microphone.setGain()</code>	指定麦克风信号的提升量。
<code>Microphone.setRate()</code>	以 kHz 为单位指定麦克风的声音捕获频率。
<code>Microphone.setSilenceLevel()</code>	指定激活麦克风所需的音量。
<code>Microphone.setUseEchoSuppression()</code>	指定是否使用音频编解码器的回声抑制功能。

Microphone 类的属性概要

属性（只读）	说明
<code>Microphone.activityLevel</code>	麦克风所检测的音量。
<code>Microphone.gain</code>	麦克风在传送信号前将信号提升的增益量。
<code>Microphone.index</code>	当前麦克风的索引。
<code>Microphone.muted</code>	布尔值，指定用户是允许还是拒绝对麦克风的访问。
<code>Microphone.name</code>	当前声音捕获设备的名称，它由声音捕获硬件返回。
<code>Microphone.names</code>	类属性：字符串的数组，反映所有可用的声音捕获设备（包括声卡和麦克风）的名称。
<code>Microphone.rate</code>	声音捕获频率，单位为 kHz。
<code>Microphone.silenceLevel()</code>	激活麦克风所需的音量。
<code>Microphone.silenceTimeout()</code>	麦克风停止检测声音和调用 <code>Microphone.onActivity(false)</code> 这两个时间之间的毫秒数。
<code>Microphone.useEchoSuppression()</code>	布尔值，指示是否使用回声抑制。

Microphone 类的事件处理函数概要

事件处理函数	说明
Microphone.onActivity	在麦克风开始或停止检测声音时调用。
Microphone.onStatus	在用户允许或拒绝对麦克风的访问时调用。

Microphone 类的构造函数

请参见 [Microphone.get\(\)](#)。

Microphone.activityLevel

可用性

Flash Player 6。

用法

activeMicrophone.activityLevel

说明

只读属性；指定麦克风所检测的音量的数字值。值的范围从 0（未检测到声音）到 100（检测到非常大的声音）。此属性的值有助于确定向 [Microphone.setSilenceLevel\(\)](#) 方法传递的适当值。

如果麦克风可用，但却因为尚未调用 [Microphone.get\(\)](#) 而未被使用，则此属性设置为 -1。

示例

以下示例将变量 `level` 设置为当前麦克风的级别 `myMic.activityLevel`。

```
var level = myMic.activityLevel;
```

另请参见

[Microphone.setGain\(\)](#)

Microphone.gain

可用性

Flash Player 6。

用法

activeMicrophone.gain

说明

只读属性；麦克风信号的提升量。有效值的范围是 0 到 100。默认值为 50。

示例

以下示例附加到滑动条的尖端。当加载此剪辑时，Flash 将检查值 `myMic.gain`，如果该值未定义，则提供默认值。然后，使用 `_x` 位置将麦克风的增益设置为用户的首选项。

```
onClipEvent (load) {  
    if (_root.myMic.gain == undefined) {  
        _root.myMic.setGain = 75;  
    }  
}
```

```

    }

    this._x = _root.myMic.gain;
    _root.txt_micgain = this._x;

    left = this._x;
    right = left+50;
    top = this._y;
    bottom = top;
}

on (press) {
    startDrag(this, false, left, top, right, bottom);
    this._xscale = 100;
    this._yscale = 100;
}

on (release, releaseOutside) {
    stopDrag();
    g = (this._x-50)*2;
    _root.myMic.setGain(g);
    _root.txt_micgain = g;
    this._xscale = 100;
    this._yscale = 100;
}

```

另请参见

[Microphone.setGain\(\)](#)

Microphone.get()

可用性

Flash Player 6。

用法

`Microphone.get([index])`

注意：正确的语法为 `Microphone.get()`。若要将在 `Microphone` 对象分配给变量，请使用类似于 `active_mic = Microphone.get()` 的语法。

参数

index 一个从零开始的可选整数，指定要获取的麦克风，该整数根据 `Microphone.names` 包含的数组确定。若要获取默认的麦克风（建议大多数应用程序采用此设置），请省略此参数。

返回

- 如果未指定 *index*，此方法将返回对默认麦克风的引用，如果默认麦克风不可用，则返回对第一个可用麦克风的引用。如果没有可用的麦克风或者未安装麦克风，该方法将返回 `null`。
- 如果已指定 *index*，此方法将返回对所请求麦克风的引用，如果该麦克风不可用，则返回 `null`。

说明

方法；返回用于捕获音频的 `Microphone` 对象的引用。若要实际开始捕获音频，必须将 `Microphone` 对象附加到 `MovieClip` 对象（请参见 `MovieClip.attachAudio()`）。

与使用 `new` 构造函数创建的对象不同，对 `Microphone.get()` 的多次调用会引用相同的麦克风。因此，如果您的脚本包含代码行 `mic1 = Microphone.get()` 和 `mic2 = Microphone.get()`，`mic1` 和 `mic2` 均引用相同的（默认）麦克风。

通常情况下，不应该为 `index` 传递值，而应使用 `Microphone.get()` 方法返回对默认麦克风的引用。通过“麦克风设置”面板（将在本节的后面部分讨论），用户可以指定 Flash 应该使用的默认麦克风。如果为 `index` 传递值，则可能会试图引用用户首选项之外的麦克风。您可能会在极少见的情况下（即您的应用程序同时从两个麦克风捕获音频）使用 `index`。

当 SWF 文件尝试访问 `Microphone.get()` 方法返回的麦克风时（例如，当您发出 `MovieClip.attachAudio()` 时），Flash Player 将显示一个“隐私”对话框，让用户选择是允许还是拒绝对麦克风的访问。（确保舞台大小至少为 215 x 138 像素；这是 Flash 显示该对话框所需的最小大小。）



当用户对此对话框做出响应时，`Microphone.onStatus` 事件处理函数将返回指示用户响应的信息对象。若要在不处理此事件处理函数的情况下确定用户是拒绝还是允许对摄像机的访问，请使用 `Microphone.muted`。

用户也可以为特定域指定永久隐私设置，方法是在 SWF 文件播放过程中右击 (Windows) 或按住 `Control` 键单击 (Macintosh)，选择“设置”，打开“隐私”面板，然后选择“记住”。



如果 `Microphone.get()` 返回 `null`，则表明麦克风正在由另一个应用程序使用，或者系统上没有安装麦克风。若要确定是否已安装麦克风，请使用 `Microphones.names.length`。若要显示 Flash Player “麦克风设置”面板（让用户选择 `Microphone.get()` 所引用的麦克风），请使用 `System.showSettings(2)`。



示例

以下示例让用户指定默认麦克风，然后捕获音频并在本地回放。若要避免回馈，可能需要带着耳机测试此代码。

```
System.showSettings(2);
```

```
myMic = Microphone.get();  
_root.attachAudio(myMic);
```

另请参见

[Microphone.index](#), [Microphone.muted](#), [Microphone.names](#), [Microphone.onStatus](#),
[MovieClip.attachAudio\(\)](#)

Microphone.index

可用性

Flash Player 6。

用法

activeMicrophone.index

说明

只读属性；从零开始的整数，指定麦克风的索引，它反映在 [Microphone.names](#) 返回的数组中。

另请参见

[Microphone.get\(\)](#), [Microphone.names](#)

Microphone.muted

可用性

Flash Player 6。

用法

activeMicrophone.muted

说明

只读属性；布尔值，指定用户是已经拒绝对麦克风的访问 (*true*) 还是已经允许访问 (*false*)。当此值出现更改时，将调用 [Microphone.onStatus](#)。有关更多信息，请参见 [Microphone.get\(\)](#)。

示例

在以下示例中，当用户单击该按钮时，如果麦克风不处于静音状态，Flash 将发布并播放实时流。

```
on (press)
{
    // 如果用户将麦克风设置为静音，则显示脱机通知。
    // 否则，从麦克风中发布并播放实时流。
    if(myMic.muted) {
        _root.debugWindow+="Microphone offline."newline
    } else {

        // 通过调用根函数 pubLive()
        // 发布麦克风数据。
        _root.pubLive();

        // 播放通过调用根函数 pubLive()
        // 所发布的内容。
        _root.playLive();
    }
}
```

另请参见

[Microphone.get\(\)](#), [Microphone.onStatus](#)

Microphone.name

可用性

Flash Player 6。

用法

activeMicrophone.name

说明

只读属性；指定当前声音捕获设备的名称的字符串，它由声音捕获硬件返回。

示例

以下示例显示“输出”面板中默认麦克风的名称。

```
myMic = Microphone.get();
trace("The microphone name is:" + myMic.name);
```

另请参见

[Microphone.get\(\)](#), [Microphone.names](#)

Microphone.names

可用性

Flash Player 6。

用法

`Microphone.names`

注意：正确的语法是 `Microphone.names`。若要将返回值赋予变量，请使用类似于 `mic_array = Microphone.names` 的语法。若要确定当前麦克风的名称，请使用 `activeMicrophone.name`。

说明

只读类属性；获取反映所有可用声音捕获设备名称的字符串数组，而不显示 Flash Player “隐私设置” 面板。此数组的行为与其它任何动作脚本数组均相同，即隐式提供每个声音捕获设备的从零开始索引以及系统上的声音捕获设备数量（通过 `Microphone.names.length`）。有关更多信息，请参见 [Array 类](#) 条目。

若要调用 `Microphone.names`，需要广泛地检查硬件，而建立数组可能需要几秒钟的时间。大多数情况下，使用默认麦克风即可。

示例

以下代码返回关于音频设备数组的信息。

```
allMicNames_array = Microphone.names;
_root.debugWindow += "Microphone.names located these device(s):" + newline;
for(i=0; i < allMicNames_array.length; i++){
    debugWindow += "[" + i + "]: " + allMicNames[i] + newline;
}
```

例如，可能会显示以下信息。

```
Microphone.names located these device(s):
[0]: Crystal SoundFusion(tm)
[1]: USB Audio Device
```

另请参见

[Array 类](#), [Microphone.name](#)

Microphone.onActivity

可用性

Flash Player 6。

用法

```
activeMicrophone.onActivity = function(activity) {  
    // 此处是您的语句  
}
```

参数

activity 布尔值，在麦克风开始检测声音时设置为 `true`，在停止时设置为 `false`。

返回

无。

说明

事件处理函数；在麦克风开始或停止检测声音时调用。如果要对此事件处理函数做出响应，则必须创建一个函数来处理其 *activity* 值。

若要指定调用 `Microphone.onActivity(true)` 所需的音量以及在调用 `Microphone.onActivity(false)` 之前必须在无声状态下经过的时间，请使用 [Microphone.setSilenceLevel\(\)](#)。

示例

当麦克风开始或停止检测声音时，以下示例显示在“输出”面板中显示 `true` 或 `false`。

```
m = Microphone.get();  
_root.attachAudio(m);  
m.onActivity = function(mode)  
{  
    trace(mode);  
};
```

另请参见

[Microphone.onActivity](#), [Microphone.setSilenceLevel\(\)](#)

Microphone.onStatus

可用性

Flash Player 6。

用法

```
activeMicrophone.onStatus = function(infoObject) {  
    // 此处是您的语句  
}
```

参数

infoObject 按照状态消息定义的参数。

返回

无。

说明

事件处理函数；在用户允许或拒绝对麦克风的访问时调用。如果要对此事件处理函数做出响应，必须创建一个函数来处理麦克风生成的信息对象。

当 SWF 文件尝试访问麦克风时，Flash Player 将显示一个“隐私”对话框，用户可以在其中选择是允许还是拒绝访问。

- 如果用户允许访问，则 `Microphone.muted` 属性设置为 `false`，并且用 `code` 属性为 "Microphone.Unmuted"、`level` 属性为 "Status" 的信息对象调用此事件处理函数。
- 如果用户拒绝访问，则 `Microphone.muted` 属性设置为 `true`，并且用 `code` 属性为 "Microphone.Muted"、`level` 属性为 "Status" 的信息对象调用此事件处理函数。

若要在不处理此事件处理函数的情况下确定用户是拒绝还是允许对麦克风的访问，请使用 `Microphone.muted`。

注意：如果用户选择永久允许或拒绝对来自指定域的所有 SWF 文件的访问，则对于来自该域的 SWF 文件不调用此方法，除非用户以后更改该隐私设置。有关更多信息，请参见 `Microphone.get()`。

示例

请参见 `Camera.onStatus` 的示例。

另请参见

`Microphone.get()`、`Microphone.muted`

Microphone.rate

可用性

Flash Player 6。

用法

`activeMicrophone.rate`

说明

只读属性；麦克风的声​​音捕获频率，单位为 kHz。如果您的声音捕获设备支持，则默认值为 8 kHz。否则，默认值为您的声音捕获设置支持且高于 8 kHz 的下一个可用捕获级别，这通常是 11 kHz。

若要设置此值，请使用 `Microphone.setRate()`。

示例

以下示例将当前频率保存到变量 `original`。

```
original = myMic.rate;
```

另请参见

`Microphone.setRate()`

Microphone.setGain()

可用性

Flash Player 6。

用法

```
activeMicrophone.setGain(gain)
```

参数

gain 指定麦克风信号提升量的整数。有效值的范围是 0 到 100。默认值为 50 ；但是，用户可以在 Flash Player “麦克风设置” 面板中更改该值。

返回

无。

说明

方法 ；设置麦克风增益，即麦克风在传送信号之前应该将信号加倍的数量。值为 0 将指示 Flash 乘以 0 ；即，麦克风不传送任何声音。

您可以将此设置视为立体声音响上的音量旋钮：0 表示无音量，50 表示正常音量 ；小于 50 的数字指定低于正常音量，而大于 50 的数字指定高于正常音量。

示例

以下示例确保麦克风增益设置小于或等于 55。

```
var myMic = Microphone.get();
if (myMic.gain > 55){
    myMic.setGain(55);
}
```

另请参见

[Microphone.gain](#), [Microphone.setUseEchoSuppression\(\)](#)

Microphone.setRate()

可用性

Flash Player 6。

用法

```
activeMicrophone.setRate(kHz)
```

参数

kHz 麦克风捕获声音的频率，单位是 kHz。可接受的值为 5、8、11、22 和 44。如果您的声音捕获设备支持，则默认值为 8 kHz。否则，默认值为您的声音捕获设置支持且高于 8 kHz 的下一个可用捕获级别，这通常是 11 kHz。

返回

无。

说明

方法 ；设置麦克风捕获声音的频率，单位是 kHz。

示例

以下示例将麦克风频率设置为用户的首选项（已经赋予 `userRate` 变量的值），前提是它是以下值之一：5、8、11、22 或 44。否则，该值将舍入到最近似的且声音捕获设备支持的可接受值。

```
myMic.setRate(userRate);
```

另请参见

[Microphone.rate](#)

Microphone.setSilenceLevel()

可用性

Flash Player 6。

用法

```
activeMicrophone.setSilenceLevel(level [, timeout])
```

参数

level 整数，指定激活麦克风和调用 `Microphone.onActivity(true)` 所需的音量。可接受值的范围是 0 到 100。默认值为 10。

timeout 可选的整数参数，指定必须经过多少毫秒的不活动时间，Flash 才能认为声音已停止并调用 `Microphone.onActivity(false)`。默认值为 2000（2 秒）。

返回

无。

说明

方法；设置应该被视为有声的最小输入级别以及（可选）指示静音已实际启用的静音时间。

- 若要防止麦克风检测到任何声音，请为 *level* 传递值 100；这样就决不会调用 `Microphone.onActivity`。
- 若要确定麦克风当前所检测的音量，请使用 `Microphone.activityLevel`。

活动检测是检测声音级别在何时表示某人正在谈话的功能。当某人没有谈话时，由于不需要发送关联的音频流，因此可以节约带宽。此信息也可用于视频反馈，以便让用户知道他们（或其他人）没有谈话。

静音值与活动值直接对应。完全静音对应于活动值 0。持续噪音（可以根据当前增益设置检测到的噪音）对应于活动值 100。当增益得到适当调整之后，活动值将在您未谈话时小于静音值，而在您谈话时大于静音值。

此方法的用途与 `Camera.setMotionLevel()` 相同；这两种方法都用于指定应该在何时调用 `onActivity` 事件处理函数。但是这些方法对发布流具有非常不同的影响：

- `Camera.setMotionLevel()` 用于检测运动，它不影响带宽用量。即使视频流未检测到运动，仍将发送视频。
- `Microphone.setSilenceLevel()` 用于优化带宽。当音频流被视为静音时，将不发送任何音频数据。所发送的是一个指示静音已启动的消息。

示例

以下代码根据用户的输入更改静音级别。该按钮附加了以下代码：

```
on (press)
{
    this.makeSilenceLevel(this.silenceLevel);
}
```

该按钮调用的 `makeSilenceLevel()` 函数将继续：

```
function makeSilenceLevel(s)
{
    this.obj.setSilenceLevel(s);
    this.SyncMode();
    this.silenceLevel= s;
}
```

有关更多信息，请参见 `Camera.setMotionLevel()` 的示例。

另请参见

`Microphone.activityLevel`, `Microphone.onActivity`, `Microphone.setGain()`,
`Microphone.silenceLevel()`, `Microphone.silenceTimeout()`

Microphone.setUseEchoSuppression()

可用性

Flash Player 6。

用法

```
activeMicrophone.setUseEchoSuppression(suppress)
```

参数

suppress 布尔值，指定是 (`true`) 否 (`false`) 应该使用回声抑制。

返回

无。

说明

方法；指定是否使用音频编解码器的回声抑制功能。默认值为 `false`，除非用户已经在 Flash Player “麦克风设置” 面板中选择 “降低回声”。

回声抑制是指降低音频回馈效果，当麦克风发出的声音由同一台计算机上的麦克风拾取时，将导致音频回馈。（这不同于回声消除，后者完全消除回馈。）

通常情况下，当所捕获的声音通过同一台计算机上的扬声器（而不是耳机）播放时，建议使用回声抑制。如果您的 SWF 文件允许用户指定声音输出设备，则当他们指出他们正在使用扬声器并且还将使用麦克风时，可能需要调用 `Microphone.setUseEchoSuppression(true)`。

用户也可以在 Flash Player “麦克风设置” 面板中调整这些设置。

示例

以下示例启用回声抑制。

```
my_mic.setUseEchoSuppression(true);
```

另请参见

[Microphone.setGain\(\)](#), [Microphone.useEchoSuppression\(\)](#)

Microphone.silenceLevel()

可用性

Flash Player 6。

用法

activeMicrophone.silenceLevel

说明

只读属性；指定激活麦克风和调用 [Microphone.onActivity\(true\)](#) 所需的音量的整数。默认值为 10。

示例

请参见 [Microphone.silenceTimeout\(\)](#) 的示例。

另请参见

[Microphone.gain](#), [Microphone.setSilenceLevel\(\)](#)

Microphone.silenceTimeout()

可用性

Flash Player 6。

用法

activeMicrophone.silenceTimeout

说明

只读属性；表示麦克风停止检测声音和调用 [Microphone.onActivity\(false\)](#) 这两个事件之间的毫秒数的数字值。默认值为 2000（2 秒）。

若要设置此值，请使用 [Microphone.setSilenceLevel\(\)](#)。

示例

以下示例将超时设置为其当前值的两倍。

```
myMic.setSilenceLevel(myMic.silenceLevel, myMic.silenceTimeOut * 2);
```

另请参见

[Microphone.setSilenceLevel\(\)](#)

Microphone.useEchoSuppression()

可用性

Flash Player 6。

用法

activeMicrophone.useEchoSuppression

说明

只读属性；如果启用回声抑制，则为布尔值 `true`；否则为 `false`。默认值为 `false`，除非用户已经在 Flash Player “麦克风设置” 面板上选择 “降低回声”。

示例

以下示例检查回声抑制，如果它尚未启用，则将其启用。

```
_root.myMic.onActivity = function(active) {  
    if (active == true) {  
        if (_root.myMic.useEchoSuppression == false) {  
            _root.myMic.setUseEchoSuppression(true);  
        }  
    }  
}
```

另请参见

[Microphone.setUseEchoSuppression\(\)](#)

MMExecute()

可用性

Flash Player 7。

用法

MMExecute(*"Flash JavaScript API command;"*)

参数

Flash JavaScript API command 您可以在 Flash JavaScript (JSFL) 文件中使用的任何命令。

返回

JavaScript 语句发送的结果（如果有）。

说明

函数；可用于从动作脚本中发出 Flash JavaScript API 命令。

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

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

有关 JSAPI 的更多信息，请参见 www.macromedia.com/go/jsapi_info_zh_cn。

示例

下面的命令返回库中对象的数组：

```
var lib:Array = MMExecute("fl.getDocumentDOM().library.items;");
trace(lib.length + " items in library");
```

Mouse 类

可用性

Flash Player 5。

说明

Mouse 类是不通过构造函数即可访问其属性和方法的顶级类。您可以使用 Mouse 类的方法来隐藏和显示 SWF 文件中的鼠标指针（光标）。默认情况下鼠标指针是可见的，但是您可以将其隐藏并实现用影片剪辑创建的自定义指针（请参见第 83 页的“创建自定义鼠标指针”）。

Mouse 类的方法概要

方法	说明
Mouse.addListener()	注册一个对象以接收 <code>onMouseDown</code> 、 <code>onMouseMove</code> 和 <code>onMouseUp</code> 通知。
Mouse.hide()	隐藏 SWF 文件中的鼠标指针。
Mouse.removeListener()	删除用 <code>addListener()</code> 注册的对象。
Mouse.show()	在 SWF 文件中显示鼠标指针。

Mouse 类的侦听器概要

方法	说明
Mouse.onMouseDown	按下鼠标按钮时获得通知。
Mouse.onMouseMove	移动鼠标按钮时获得通知。
Mouse.onMouseUp	释放鼠标按钮时获得通知。
Mouse.onMouseWheel	当用户滚动鼠标滚轮时获得通知。

Mouse.addListener()

可用性

Flash Player 6。

用法

```
Mouse.addListener (newListener)
```

参数

newListener 一个对象。

返回

无。

说明

方法；注册一个对象以接收 `onMouseDown`、`onMouseMove` 和 `onMouseUp` 侦听器的通知。

newListener 参数应该包含一个带有为 `onMouseDown`、`onMouseMove` 和 `onMouseUp` 侦听器定义的方法的对象。

当按下、移动或释放鼠标时，不论输入焦点位于何处，用此方法注册的所有侦听对象都调用其 `onMouseDown`、`onMouseMove` 或 `onMouseUp` 方法。可以有多个对象侦听鼠标通知。如果已经注册了侦听器 *newListener*，则不会发生任何更改。

另请参见

[Mouse.onMouseDown](#), [Mouse.onMouseMove](#), [Mouse.onMouseUp](#)

Mouse.hide()

可用性

Flash Player 5。

用法

```
Mouse.hide()
```

参数

无。

返回

布尔值：如果指针可见，则为 `true`；如果指针不可见，则为 `false`。

说明

方法；隐藏 SWF 文件中的指针。指针在默认情况下可见。

示例

附加到主时间轴上影片剪辑的以下代码会隐藏标准指针，并将 `customPointer_mc` 影片剪辑实例的 `x` 和 `y` 位置设置为主时间轴中的 `x` 和 `y` 鼠标位置。

```
onClipEvent (enterFrame) {  
    Mouse.hide();  
    customPointer_mc._x = _root._xmouse;
```

```
        customPointer_mc._y = _root._ymouse;  
    }  
}
```

另请参见

[Mouse.show\(\)](#), [MovieClip._xmouse](#), [MovieClip._ymouse](#)

Mouse.onMouseDown

可用性

Flash Player 6。

用法

someListener.onMouseDown

参数

无。

返回

无。

说明

侦听器；按下鼠标时获得通知。若要使用 `onMouseDown` 侦听器，必须先创建一个侦听器对象。然后可以为 `onMouseDown` 定义一个函数并使用 `addListener()` 向 `Mouse` 对象注册该侦听器，如下代码所示：

```
somelister = new Object();  
somelister.onMouseDown = function () { ... };  
Mouse.addListener(somelister);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

另请参见

[Mouse.addListener\(\)](#)

Mouse.onMouseMove

可用性

Flash Player 6。

用法

someListener.onMouseMove

参数

无。

返回

无。

说明

侦听器；鼠标移动时获得通知。若要使用 `onMouseMove` 侦听器，必须先创建一个侦听器对象。然后可以为 `onMouseMove` 定义一个函数并使用 `addListener()` 向 `Mouse` 对象注册该侦听器，如下代码所示：

```
somelister = new Object();
somelister.onMouseMove = function () { ... };
Mouse.addListener(somelister);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

另请参见

[Mouse.addListener\(\)](#)

Mouse.onMouseUp

可用性

Flash Player 6。

用法

somelister.onMouseUp

参数

无。

返回

无。

说明

侦听器；释放鼠标时获得通知。若要使用 `onMouseUp` 侦听器，必须先创建一个侦听器对象。然后可以为 `onMouseUp` 定义一个函数并使用 `addListener()` 方法向 `Mouse` 对象注册该侦听器，如下代码所示：

```
somelister = new Object();
somelister.onMouseUp = function () { ... };
Mouse.addListener(somelister);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

另请参见

[Mouse.addListener\(\)](#)

Mouse.onMouseWheel

可用性

Flash Player 7（仅限 Windows）。

用法

```
someListener.onMouseWheel = function ( [ delta , scrollTarget ] ) {  
    // 此处是您的语句  
}
```

参数

delta 一个可选数字，指示对于用户将鼠标滚轮滚动的每个刻度，屏幕上将滚动多少行。正 *delta* 值指示向上的滚动；而负值指示向下的滚动。常用值为 1 到 3，而较快的滚动可能产生较大的值。

如果您不想为 *delta* 指定值，但想为 *scrollTarget* 指定值，则为 *delta* 传递 `null`。

scrollTarget 当滚动鼠标滚轮时位于鼠标下的最顶部的影片剪辑实例。

返回

无。

说明

侦听器；在用户滚动鼠标滚轮时获得通知。若要使用 `onMouseWheel` 侦听器，必须先创建一个侦听器对象。然后可以为 `onMouseWheel` 定义一个函数并使用 `addListener()` 向 `Mouse` 对象注册该侦听器。

注意：鼠标滚轮事件侦听器只有在 Windows 版的 Flash Player 上才可用。

示例

以下示例显示如何创建对鼠标滚轮事件做出响应的侦听器对象。在此示例中，每当用户滚动鼠标滚轮时，名为 `clip_mc`（未显示）的影片剪辑对象的 `x` 坐标将发生更改。

```
mouseListener = new Object();  
mouseListener.onMouseWheel = function(delta) {  
    clip_mc._x += delta;  
}  
Mouse.addListener(mouseListener);
```

另请参见

[Mouse.addListener\(\)](#), [TextField.mouseWheelEnabled](#)

Mouse.removeListener()

可用性

Flash Player 6。

用法

```
Mouse.removeListener (listener)
```

参数

listener 一个对象。

返回

如果成功删除了 *listener* 对象，则该方法返回 `true` ；如果未成功删除 *listener*（例如，如果 *listener* 不在 `Mouse` 对象的侦听器列表中），则该方法返回 `false`。

说明

方法；删除以前用 `addListener()` 注册的对象。

Mouse.show()

可用性

Flash Player 5。

用法

```
Mouse.show()
```

参数

无。

返回

无。

说明

方法；在 SWF 文件中显示鼠标指针。指针在默认情况下可见。

另请参见

[Mouse.show\(\)](#), [MovieClip._xmouse](#), [MovieClip._ymouse](#)

MovieClip 类

可用性

Flash Player 3。

说明

MovieClip 类的方法提供的功能与定位影片剪辑的动作所提供的功能相同。还有一些其它方法在“动作”面板中的“动作”工具箱中没有等效动作。

无需使用构造函数方法即可调用 MovieClip 类的方法；只需使用以下语法按名称引用影片剪辑实例即可：

```
my_mc.play();
my_mc.gotoAndPlay(3);
```

MovieClip 类的方法概要

方法	说明
<code>MovieClip.attachAudio()</code>	从麦克风硬件中捕获并播放本地音频。
<code>MovieClip.attachMovie()</code>	附加库中的 SWF 文件。
<code>MovieClip.createEmptyMovieClip()</code>	创建一个空的影片剪辑。
<code>MovieClip.createTextField()</code>	创建一个空的文本字段。
<code>MovieClip.duplicateMovieClip()</code>	重制指定的影片剪辑。
<code>MovieClip.getBounds()</code>	返回 SWF 文件在指定坐标空间中的最小和最大的 x、y 坐标。
<code>MovieClip.getBytesLoaded()</code>	返回为指定影片剪辑加载的字节数。
<code>MovieClip.getBytesTotal()</code>	以字节为单位返回影片剪辑的大小。
<code>MovieClip.getDepth()</code>	返回影片剪辑的深度。
<code>MovieClip.getInstanceAtDepth()</code>	指定某特定深度是否已被影片剪辑所占用。
<code>MovieClip.getNextHighestDepth()</code>	指定一个深度值，您可以将该值传递给其它方法，以确保 Flash 将该影片剪辑呈现在当前影片剪辑中所有其它对象的前面。
<code>MovieClip.getSWFVersion()</code>	返回一个整数，该整数指示所发布的影片剪辑的 Flash Player 版本
<code>MovieClip.getTextSnapshot()</code>	返回包含指定影片剪辑的静态文本字段中的文本的 TextSnapshot 对象。
<code>MovieClip.getURL()</code>	从一个 URL 获取文档。
<code>MovieClip.globalToLocal()</code>	将 point 对象的坐标从舞台坐标转换为指定影片剪辑的本地坐标。
<code>MovieClip.gotoAndPlay()</code>	将播放头转到影片剪辑中的特定帧并播放 SWF 文件。
<code>MovieClip.gotoAndStop()</code>	将播放头转到影片剪辑中的特定帧并停止 SWF 文件。
<code>MovieClip.hitTest()</code>	如果指定影片剪辑的边框与目标影片剪辑的边框交叉，则返回 true。
<code>MovieClip.loadMovie()</code>	将指定的 SWF 文件加载到影片剪辑中。
<code>MovieClip.loadVariables()</code>	将来自 URL 或其它位置的变量加载到影片剪辑中。

方法	说明
<code>MovieClip.localToGlobal()</code>	将 <code>Point</code> 对象的坐标从影片剪辑的本地坐标转换为全局舞台坐标。
<code>MovieClip.nextFrame()</code>	将播放头转到影片剪辑的下一帧。
<code>MovieClip.play()</code>	播放指定的影片剪辑。
<code>MovieClip.prevFrame()</code>	将播放头转到影片剪辑的上一帧。
<code>MovieClip.removeMovieClip()</code>	如果影片剪辑是用 <code>duplicateMovieClip()</code> 、 <code>MovieClip.duplicateMovieClip()</code> 或 <code>MovieClip.attachMovie()</code> 创建的，则将其从时间轴中删除。
<code>MovieClip.setMask()</code>	将影片剪辑指定为另一个影片剪辑的遮罩。
<code>MovieClip.startDrag()</code>	将影片剪辑指定为可拖动的并开始拖动该影片剪辑。
<code>MovieClip.stop()</code>	停止当前播放的 SWF 文件。
<code>MovieClip.stopDrag()</code>	停止拖动任何正在拖动的影片剪辑。
<code>MovieClip.swapDepths()</code>	交换两个 SWF 文件的深度级别。
<code>MovieClip.unloadMovie()</code>	删除用 <code>loadMovie()</code> 加载的 SWF 文件。

MovieClip 类的绘制方法概要

方法	说明
<code>MovieClip.beginFill()</code>	开始在舞台上绘制填充。
<code>MovieClip.beginGradientFill()</code>	开始在舞台上绘制渐变填充。
<code>MovieClip.clear()</code>	删除与影片剪辑实例相关联的所有绘画命令。
<code>MovieClip.curveTo()</code>	使用最新的线条样式绘制曲线。
<code>MovieClip.endFill()</code>	终止由 <code>beginFill()</code> 或 <code>beginGradientFill()</code> 指定的填充。
<code>MovieClip.lineStyle()</code>	定义用 <code>lineTo()</code> 和 <code>curveTo()</code> 方法创建的线条的笔触。
<code>MovieClip.lineTo()</code>	使用当前的线条样式绘制线条。
<code>MovieClip.moveTo()</code>	将当前的绘画位置移到指定的坐标。

MovieClip 类的属性概要

属性	说明
<code>MovieClip._alpha</code>	影片剪辑实例的透明度值。
<code>MovieClip._currentframe</code>	播放头当前所处的帧的编号。
<code>MovieClip._droptarget</code>	以斜杠语法记号表示的影片剪辑实例（放置可拖动影片剪辑的影片剪辑实例）的绝对路径。
<code>MovieClip.enabled</code>	指示按钮影片剪辑是否处于启用状态。
<code>MovieClip.focusEnabled</code>	使影片剪辑能够接收焦点。
<code>MovieClip._focusrect</code>	指示具有焦点的影片剪辑周围是否有黄色矩形。

属性	说明
<code>MovieClip._framesloaded</code>	从 SWF 文件流中已经加载的帧数。
<code>MovieClip._height</code>	影片剪辑实例的高度，以像素为单位。
<code>MovieClip.hitArea</code>	将另一个影片剪辑指定为按钮影片剪辑的点击区域。
<code>MovieClip._highquality</code>	设置 SWF 文件的呈现品质。
<code>MovieClip.menu</code>	将 ContextMenu 对象与影片剪辑关联。
<code>MovieClip._name</code>	影片剪辑实例的实例名称。
<code>MovieClip._parent</code>	对包含有该影片剪辑的影片剪辑的引用。
<code>MovieClip._rotation</code>	影片剪辑实例的旋转角度。
<code>MovieClip._soundbuftime</code>	声音开始进入流之前存储的秒数。
<code>MovieClip.tabChildren</code>	指示影片剪辑的子级是否包含在 Tab 键的自动排序中。
<code>MovieClip.tabEnabled</code>	指示某影片剪辑是否包含在 Tab 键排序中。
<code>MovieClip.tabIndex</code>	指示对象的 Tab 键顺序。
<code>MovieClip._target</code>	影片剪辑实例的目标路径。
<code>MovieClip._totalframes</code>	影片剪辑实例中的总帧数。
<code>MovieClip.trackAsMenu</code>	指示其它按钮是否可接收鼠标按钮释放事件。
<code>MovieClip._url</code>	从中下载影片剪辑的 SWF 文件的 URL。
<code>MovieClip.useHandCursor</code>	确定当用户滑过按钮影片剪辑时是否显示手形光标。
<code>MovieClip._visible</code>	一个布尔值，确定影片剪辑实例是隐藏的还是可见的。
<code>MovieClip._width</code>	影片剪辑实例的宽度，以像素为单位。
<code>MovieClip._x</code>	影片剪辑实例的 x 坐标。
<code>MovieClip._xmouse</code>	影片剪辑实例中鼠标指针的 x 坐标。
<code>MovieClip._xscale</code>	指定用于水平缩放影片剪辑的百分比的值。
<code>MovieClip._y</code>	影片剪辑实例的 y 坐标。
<code>MovieClip._ymouse</code>	影片剪辑实例中鼠标指针的 y 坐标。
<code>MovieClip._yscale</code>	指定用于垂直缩放影片剪辑的百分比的值。

MovieClip 类的事件处理函数概要

事件处理函数	说明
<code>MovieClip.onData</code>	当所有数据都加载到影片剪辑中时调用。
<code>MovieClip.onDragOut</code>	鼠标指针位于按钮内时按下鼠标按钮，然后滑出该按钮区域，在此条件下，当鼠标指针位于该按钮外时进行调用。
<code>MovieClip.onDragOver</code>	鼠标指针位于按钮内时按下鼠标按钮，然后滑出该按钮区域，接着滑回到该按钮上，在此条件下，当鼠标指针位于该按钮上时进行调用。

事件处理函数	说明
<code>MovieClip.onEnterFrame</code>	以 SWF 文件的帧频持续调用。首先处理与 <code>enterFrame</code> 剪辑事件关联的动作，然后才处理附加到受影响帧的所有帧动作脚本。
<code>MovieClip.onKeyDown</code>	当按下按键时调用。使用 <code>Key.getCode()</code> 和 <code>Key.getAscii()</code> 方法可获取关于最后所按键的信息。
<code>MovieClip.onKeyUp</code>	当释放按键时调用。
<code>MovieClip.onKillFocus</code>	当从按钮移除焦点时调用。
<code>MovieClip.onLoad</code>	当影片剪辑被实例化并显示在时间轴上时调用。
<code>MovieClip.onMouseDown</code>	当按下鼠标左键时调用。
<code>MovieClip.onMouseMove</code>	每次移动鼠标时调用。
<code>MovieClip.onMouseUp</code>	当释放鼠标左键时调用。
<code>MovieClip.onPress</code>	在鼠标指针位于按钮上方的情况下，按下鼠标按钮时调用。
<code>MovieClip.onRelease</code>	在鼠标指针位于按钮上方的情况下，释放鼠标按钮时调用。
<code>MovieClip.onReleaseOutside</code>	在这样的情况下调用：在鼠标指针位于按钮内部的情况下按下按钮，然后将鼠标指针移到该按钮外部并释放鼠标按钮。
<code>MovieClip.onRollOut</code>	当鼠标指针滚动到按钮区域之外时调用。
<code>MovieClip.onRollOver</code>	当鼠标指针滚过按钮时调用。
<code>MovieClip.onSetFocus</code>	当按钮具有输入焦点而且释放某按键时调用。
<code>MovieClip.onUnload</code>	从时间轴删除影片剪辑后，在第 1 帧中调用。处理与 <code>Unload</code> 影片剪辑事件关联的动作之前，不将任何动作附加到受影响的帧。

MovieClip._alpha

可用性

Flash Player 4。

用法

`my_mc._alpha`

说明

属性；`my_mc` 指定的影片剪辑的 `alpha` 透明度值。有效值为 0（完全透明）到 100（完全不透明）。默认值为 100。如果影片剪辑的 `_alpha` 设置为 0，即使其中的对象不可见，但也是活动的。例如，您仍可以单击一个 `_alpha` 属性设置为 0 的影片剪辑中的按钮。

示例

下面的代码完成以下任务：当单击按钮时，将名为 `star_mc` 的影片剪辑的 `_alpha` 属性设置为 30%：

```
on (release) {
    star_mc._alpha = 30;
}
```

另请参见

[Button._alpha](#), [TextField._alpha](#)

MovieClip.attachAudio()

可用性

Flash Player 6 ；能够附加来自 Flash Player 7 中添加的 Flash 视频 (FLV) 文件的音频。

用法

```
my_mc.attachAudio(source)
```

参数

source 包含要播放的音频的对象。有效值是 Microphone 对象、播放 FLV 文件的 NetStream 对象以及 false（停止播放音频）。

返回

无。

说明

方法；指定要播放的音频源。若要停止播放音频源，请为 *source* 传递 false。

示例

以下代码将一个麦克风附加到影片剪辑。

```
my_mic = Microphone.get();  
this.attachAudio(my_mic);
```

下面的示例显示如何使用 Sound 对象控制与 FLV 文件关联的声音。

```
// Clip 是包含  
// 视频对象 “my_video” 的影片剪辑的实例名称。  
_root.Clip.my_video.attachVideo(_root.myNetStream);  
_root.Clip.attachAudio(_root.myNetStream);  
var snd = new Sound("_root.Clip");  
// 调整音频：  
_root.snd.setVolume(100);
```

另请参见

[Microphone 类](#), [NetStream.play\(\)](#), [Sound 类](#), [Video.attachVideo\(\)](#)

MovieClip.attachMovie()

可用性

Flash Player 5。

用法

```
my_mc.attachMovie(idName, newName, depth [, initObject])
```

参数

idName 库中要附加到舞台上某影片剪辑的影片剪辑元件的链接名称。这是在“链接属性”对话框中的“标识符”字段中输入的名称。

newname 附加到该影片剪辑的影片剪辑实例的唯一名称。

depth 一个整数，指定 SWF 文件所放位置的深度级别。

initObject (Flash Player 6 和更高版本支持) 包含要用来填充新附加的影片剪辑的属性的对象。此参数使动态创建的影片剪辑能够接收剪辑参数。如果 *initObject* 不是对象，则将被忽略。*initObject* 的所有属性均将复制到新实例中。构造函数可使用通过 *initObject* 指定的属性。此参数是可选的。

返回

对新创建的实例的引用。

说明

方法；从库中取一个元件并将其附加到舞台上由 *my_mc* 指定的 SWF 文件中。使用 `removeMovieClip()` 或 `unloadMovie()` 可删除用 `attachMovie()` 附加的 SWF 文件。

示例

下面的示例将链接标识符为“circle”的元件附加到位于 SWF 文件舞台上的影片剪辑实例中。

```
on (release) {  
    thing.attachMovie( "circle", "circle1", 2 );  
}
```

另请参见

```
MovieClip.removeMovieClip(), MovieClip.unloadMovie(), Object.registerClass(),  
removeMovieClip()
```

MovieClip.beginFill()

可用性

Flash Player 6。

用法

```
my_mc.beginFill([rgb[, alpha]])
```

参数

rgb 一个十六进制颜色值（例如，红色是 0xFF0000，蓝色是 0x0000FF，等等）。如果未提供或未定义该值，则不创建填充。

alpha 一个介于 0 到 100 之间的整数，指定填充的 alpha 值。如果未提供该值，则使用 100（纯色）。如果该值小于 0，则 Flash 使用 0。如果该值大于 100，则 Flash 使用 100。

返回

无。

说明

方法；指示新的绘画路径的开始。如果存在一个开放路径（即如果当前的绘制位置不等于 `moveTo()` 方法中指定的上一个位置），并且该路径具有与其关联的填充，则用线条闭合该路径，然后进行填充。这类似于调用 `endFill()` 方法时的情形。如果当前没有填充与该路径相关联，则必须调用 `endFill()` 才能应用填充。

另请参见

[MovieClip.beginGradientFill\(\)](#), [MovieClip.endFill\(\)](#)

MovieClip.beginGradientFill()

可用性

Flash Player 6。

用法

```
my_mc.beginGradientFill(fillType, colors, alphas, ratios, matrix)
```

参数

fillType 字符串 "linear" 或字符串 "radial"。

colors 一个数组，包括要在渐变中使用的 RGB 十六进制颜色值（例如，红色是 0xFF0000，蓝色是 0x0000FF，等等）。

alphas 一个数组，包括与 *colors* 数组中颜色相对应的 alpha 值；有效值范围为 0 到 100。如果该值小于 0，则 Flash 使用 0。如果该值大于 100，则 Flash 使用 100。

ratios 颜色配额的数组；有效值范围为 0 到 255。该值按 100% 定义了对颜色进行采样处的宽度的百分比。

matrix 一个变形矩阵，它是具有下列两组属性之一的对象：

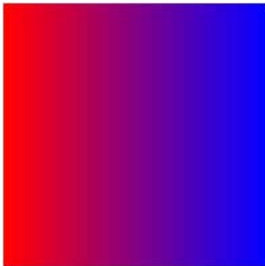
- *a*、*b*、*c*、*d*、*e*、*f*、*g*、*h*、*i*，它们用于描述如下形式的 3 x 3 矩阵：

```
a b c  
d e f  
g h i
```

下面的示例使用 `beginGradientFill()` 方法，其 *matrix* 参数是具有这些属性的对象。

```
_root.createEmptyMovieClip( "grad", 1 );  
with ( _root.grad )  
  
{  
  
    colors = [ 0xFF0000, 0x0000FF ];  
    alphas = [ 100, 100 ];  
    ratios = [ 0, 0xFF ];  
    matrix = { a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1 };  
    beginGradientFill( "linear", colors, alphas, ratios, matrix );  
    moveto(100,100);  
    lineto(100,300);  
    lineto(300,300);  
    lineto(300,100);  
    lineto(100,100);  
    endFill();  
  
}
```

如果 *matrixType* 属性不存在，则其余参数都是必需的；如果缺少其中任何一个，该函数都会失败。该矩阵缩放、平移、旋转和倾斜在 (-1,-1) 和 (1,1) 处定义的单位渐变。



- *matrixType*、*x*、*y*、*w*、*h*、*r*。

这些属性表示下列含义：*matrixType* 是字符串 "box"，*x* 是相对于该渐变左上角父级剪辑的注册点的水平位置，*y* 是相对于该渐变左上角父级剪辑的注册点的垂直位置，*w* 是渐变的宽度，*h* 是渐变的高度，*r* 是渐变的旋转角度（以弧度为单位）。

下面的示例使用 `beginGradientFill()` 方法，其 *matrix* 参数是具有这些属性的对象。

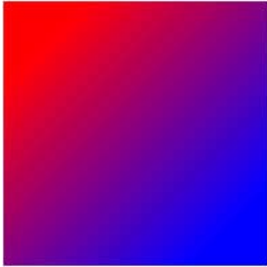
```
_root.createEmptyMovieClip( "grad", 1 );  
with ( _root.grad )  
  
{  
  
    colors = [ 0xFF0000, 0x0000FF ];  
    alphas = [ 100, 100 ];  
    ratios = [ 0, 0xFF ];  
    matrix = { matrixType:"box", x:100, y:100, w:200, h:200, r:(45/  
180)*Math.PI };  
    beginGradientFill( "linear", colors, alphas, ratios, matrix );
```

```

moveto(100,100);
lineto(100,300);
lineto(300,300);
lineto(300,100);
lineto(100,100);
endFill();
}

```

如果 *matrixType* 属性存在，则它必须等于 "box"，并且其余参数都是必需的。如果这些条件中的任何一个不满足，则该函数将失败。



返回

无。

说明

方法；指示新的绘画路径的开始。如果第一个参数为 *undefined*，或者未传递任何参数，则该路径将不填充。如果存在一个开放路径（即如果当前绘画位置不等于 *moveTo()* 方法中指定的上一个位置），并且该路径具有与其关联的填充，则用线条闭合该路径，然后进行填充。这类似于调用 *endFill()* 时的情形。

如果存在下列任意一种情况，则该方法将失败：

- *colors*、*alphas* 和 *ratios* 参数中的项目数不相等。
- *fillType* 参数不是 "linear" 或 "radial"。
- 表示 *matrix* 参数的对象中的任意一个字段缺少或无效。

示例

下列代码使用两种方法绘制两个堆积矩形，这两个矩形具有红蓝渐变填充并使用 5 磅纯绿色笔触。

```

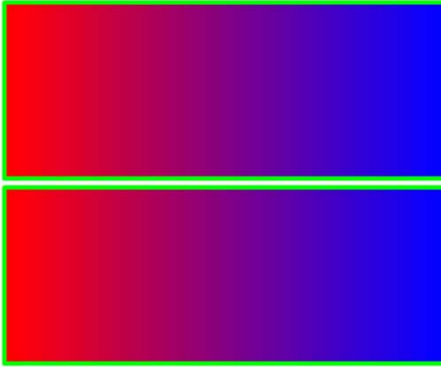
_root.createEmptyMovieClip("goober",1);
with ( _root.goober )
{
    colors = [ 0xFF0000, 0x0000FF ];
    alphas = [ 100, 100 ];
    ratios = [ 0, 0xFF ];
    lineStyle( 5, 0x00ff00 );
    matrix = { a:500,b:0,c:0,d:0,e:200,f:0,g:350,h:200,i:1};
    beginGradientFill( "linear", colors, alphas, ratios, matrix );
    moveto(100,100);
    lineto(100,300);
    lineto(600,300);
    lineto(600,100);
    lineto(100,100);
    endFill();
}

```

```

matrix = { matrixType:"box", x:100, y:310, w:500, h:200, r:(0/180)*Math.PI };
beginGradientFill( "linear", colors, alphas, ratios, matrix );
moveto(100,310);
lineto(100,510);
lineto(600,510);
lineto(600,310);
lineto(100,310);
endFill();
}

```



另请参见

[MovieClip.beginFill\(\)](#), [MovieClip.endFill\(\)](#), [MovieClip.lineStyle\(\)](#),
[MovieClip.lineTo\(\)](#), [MovieClip.moveTo\(\)](#)

MovieClip.clear()

可用性

Flash Player 6。

用法

```
my_mc.clear()
```

参数

无。

返回

无。

说明

方法；删除使用影片剪辑绘画方法（包括用 [MovieClip.lineStyle\(\)](#) 指定的线条样式）在运行时创建的所有图形。创作过程中（用 Flash 绘画工具）手动绘制的形状和线条不受影响。

另请参见

[MovieClip.lineStyle\(\)](#)

MovieClip.createEmptyMovieClip()

可用性

Flash Player 6。

用法

```
my_mc.createEmptyMovieClip(instanceName, depth)
```

参数

instanceName 标识新影片剪辑的实例名称的字符串。

depth 指定新影片剪辑的深度的整数。

返回

对新创建的影片剪辑的引用。

说明

方法；创建作为现有影片剪辑子级的空影片剪辑。该方法的行为类似于 `attachMovie()` 方法，但是不必为新的影片剪辑提供外部链接名称。新创建的空影片剪辑的注册点为左上角。如果缺少任意一个参数，则该方法将失败。

另请参见

[MovieClip.attachMovie\(\)](#)

MovieClip.createTextField()

可用性

Flash Player 6。

用法

```
my_mc.createTextField(instanceName, depth, x, y, width, height)
```

参数

instanceName 一个字符串，指示新文本字段的实例名称。

depth 一个正整数，指定新文本字段的深度。

x 一个整数，指定新文本字段的 x 坐标。

y 一个整数，指定新文本字段的 y 坐标。

width 一个正整数，指定新文本字段的宽度。

height 一个正整数，指定新文本字段的高度。

返回

无。

说明

方法；创建一个新的、空文本字段作为由 *my_mc* 指定的影片剪辑的子级。可以使用 `createTextField()` 在 SWF 文件播放时创建文本字段。文本字段位于 (*x*, *y*)，尺寸为 *width* 乘 *height*。参数 *x* 和 *y* 都相对于容器影片剪辑；这些参数与文本字段的 `_x` 和 `_y` 属性相对应。参数 *width* 和 *height* 与文本字段的 `_width` 和 `_height` 属性相对应。

文本字段的默认属性如下所示：

```
type = "dynamic"
border = false
background = false
password = false
multiline = false
html = false
embedFonts = false
variable = null
maxChars = null
```

用 `createTextField()` 创建的文本字段可以接收下列默认 `TextFormat` 对象：

```
font = "Times New Roman"
size = 12
textColor = 0x000000
bold = false
italic = false
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
bullet = false
tabStops = [] (empty array)
```

示例

下面的示例创建一个宽 300，高 100 的文本字段，其 *x* 坐标为 100，*y* 坐标为 100，该文本字段没有边框，文本为红色并带下划线。

```
_root.createTextField("mytext",1,100,100,300,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = false;

myformat = new TextFormat();
myformat.color = 0xff0000;
myformat.bullet = false;
myformat.underline = true;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

另请参见

[TextFormat 类](#)

MovieClip._currentframe

可用性

Flash Player 4。

用法

my_mc._currentframe

说明

属性（只读）；返回由 *my_mc* 指定的时间轴中播放头所处的帧的编号。

示例

下面的示例使用 `_currentframe` 属性指示影片剪辑 `actionClip_mc` 的播放头从当前位置前进 5 帧。

```
actionClip_mc.gotoAndStop(_currentframe + 5);
```

MovieClip.curveTo()

可用性

Flash Player 6。

用法

my_mc.curveTo(controlX, controlY, anchorX, anchorY)

参数

controlX 一个整数，指定控制点相对于父级影片剪辑注册点的水平位置。

controlY 一个整数，指定控制点相对于父级影片剪辑注册点的垂直位置。

anchorX 一个整数，指定下一个锚记点相对于父级影片剪辑注册点的水平位置。

anchorY 一个整数，指定下一个锚记点相对于父级影片剪辑注册点的垂直位置。

返回

无。

说明

方法；使用由 (*controlX, controlY*) 指定的控制点，以当前线条样式绘制从当前绘画位置到 (*anchorX, anchorY*) 的曲线。当前绘画位置随后设置为 (*anchorX, anchorY*)。如果正在其中进行绘制的影片剪辑中含有用 Flash 绘画工具创建的内容，则调用 `curveTo()` 将在此内容下进行绘制。如果在对 `moveTo()` 进行任何调用之前调用了 `curveTo()`，则当前绘画位置默认为 (0, 0)。如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

示例

下面的示例用极细磅值、纯蓝色线条和纯红色填充绘制一个圆。

```
_root.createEmptyMovieClip( "circle", 1 );
with ( _root.circle )
{
    lineStyle( 0, 0x0000FF, 100 );
    beginFill( 0xFF0000 );
```



```

        moveTo( 500, 500 );
        curveTo( 600, 500, 600, 400 );
        curveTo( 600, 300, 500, 300 );
        curveTo( 400, 300, 400, 400 );
        curveTo( 400, 500, 500, 500 );
        endFill();
    }
}

```

另请参见

[MovieClip.beginFill\(\)](#), [MovieClip.createEmptyMovieClip\(\)](#), [MovieClip.endFill\(\)](#), [MovieClip.lineTo\(\)](#), [MovieClip.moveTo\(\)](#)

MovieClip._droptarget

可用性

Flash Player 4。

用法

```
my_mc._droptarget
```

说明

属性（只读）；以斜杠语法记号表示法返回 *my_mc* 放置到的影片剪辑实例的绝对路径。
 _droptarget 属性始终返回以斜杠 (/) 开始的路径。若要将实例的 _droptarget 属性与引用进行比较，请使用 [eval\(\)](#) 函数将返回值从斜杠语法转换为点语法表示的引用。

注意：如果您正使用动作脚本 2.0，则必须执行此转换，因为动作脚本 2.0 不支持斜杠语法。

示例

下面的示例计算 *garbage* 影片剪辑实例的 _droptarget 属性并使用 [eval\(\)](#) 将其从斜杠语法转换为点语法表示的引用。然后将 *garbage* 引用与对 *trash* 影片剪辑实例的引用进行比较。如果两个引用相等，则将 *garbage* 的可见性设置为 false。如果它们不相等，则将 *garbage* 实例重置为其原始位置。

```

if (eval(garbage._droptarget) == _root.trash) {
    garbage._visible = false;
} else {
    garbage._x = x_pos;
    garbage._y = y_pos;
}

```

变量 *x_pos* 和 *y_pos* 是用下列脚本在 SWF 文件的第 1 帧上设置的：

```

x_pos = garbage._x;
y_pos = garbage._y;

```

另请参见

[startDrag\(\)](#)

MovieClip.duplicateMovieClip()

可用性

Flash Player 5。

用法

```
my_mc.duplicateMovieClip(newname, depth [,initObject])
```

参数

newname 一个用于重制影片剪辑的唯一标识符。

depth 一个用于指定深度的唯一编号，指定的 SWF 文件将放在该位置上。

initObject (Flash Player 6 和更高版本支持此函数。) 包含用于填充重制影片剪辑的属性的对象。此参数使动态创建的影片剪辑能够接收剪辑参数。如果 *initObject* 不是对象，则将被忽略。*initObject* 的所有属性均将复制到新实例中。构造函数可使用 *initObject* 指定的属性。此参数是可选的。

返回

对重制的影片剪辑的引用。

说明

方法；在 SWF 文件播放时创建指定影片剪辑的实例。无论调用 `duplicateMovieClip()` 方法时原始影片剪辑位于哪一帧，重制的影片剪辑总是从第 1 帧开始播放。父级影片剪辑中的变量不复制到重制的影片剪辑中。对于用 `duplicateMovieClip()` 创建的影片剪辑，如果对其父级调用 `duplicateMovieClip()`，则不重制该影片剪辑。如果删除父级影片剪辑，则重制的影片剪辑也被删除。

另请参见

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

MovieClip.enabled

可用性

Flash Player 6。

用法

```
my_mc.enabled
```

说明

属性；一个布尔值，指示按钮影片剪辑是否处于启用状态。`enabled` 的默认值为 `true`。如果将 `enabled` 设置为 `false`，则不再调用按钮影片剪辑的回调方法和 `on action` 事件处理函数，并且禁用 `Over`、`Down` 和 `Up` 帧。`enabled` 属性不影响按钮影片剪辑的时间轴；如果影片剪辑正在播放，则该影片剪辑将继续播放。影片剪辑将继续接收影片剪辑事件（例如，`mouseDown`、`mouseUp`、`keyDown` 和 `keyUp`）。

`enabled` 属性仅控制按钮影片剪辑的按钮式 (button-like) 属性。可以在任何时候更改 `enabled` 属性；可以立即启用或禁用修改后的按钮影片剪辑。可以从原型对象中读出 `enabled` 属性。如果将 `enabled` 设置为 `false`，则该对象将不包含在 `Tab` 键的自动排序中。

MovieClip.endFill()

可用性

Flash Player 6。

用法

```
my_mc.endFill()
```

参数

无。

返回

无。

说明

方法；对自从上次调用 `beginFill()` 或 `beginGradientFill()` 以来添加的线条和曲线应用填充。Flash 使用的是对 `beginFill()` 或 `beginGradientFill()` 的上一次调用中指定的填充。如果当前绘画位置不等于 `moveTo()` 方法中指定的上一个位置，而且定义了填充，则用线条闭合该路径，然后进行填充。

MovieClip.focusEnabled

可用性

Flash Player 6。

用法

```
my_mc.focusEnabled
```

说明

属性；如果值为 `undefined` 或 `false`，则除非是按钮影片剪辑，否则影片剪辑不能接收输入焦点。如果 `focusEnabled` 属性值为 `true`，则即使影片剪辑不是按钮影片剪辑，它也可以接收输入焦点。

MovieClip._focusrect

可用性

Flash Player 6。

用法

```
my_mc._focusrect
```

说明

属性；一个布尔值，指定当影片剪辑具有键盘焦点时其周围是否有黄色矩形。此属性可以覆盖全局 `_focusrect` 属性。

MovieClip._framesloaded

可用性

Flash Player 4。

用法

my_mc._framesloaded

说明

属性（只读）；从 SWF 文件流中已经加载的帧数。该属性很有用，可确定特定帧及其前面所有帧的内容是否已经加载，并且是否可在浏览器本地使用。该属性对于监视大 SWF 文件的下载很有用。例如，可能需要向用户显示一条消息以表明 SWF 文件正在下载，直到 SWF 文件中的指定帧完成下载为止。

示例

下面的示例使用 `_framesloaded` 属性在所有帧都已加载后开始播放 SWF 文件。如果未加载完所有帧，则会等比例增大影片剪辑实例 `loader` 的 `_xscale` 属性以创建进度栏。

```
if (_framesloaded >= _totalframes) {
    gotoAndPlay ("Scene 1", "start");
} else {
    _root.loader._xscale = (_framesloaded/_totalframes)*100;
}
```

另请参见

[MovieClipLoader 类](#)

MovieClip.getBounds()

可用性

Flash Player 5。

用法

my_mc.getBounds(targetCoordinateSpace)

参数

targetCoordinateSpace 其坐标系统要用作参考点的时间轴的目标路径。

返回

一个对象，带有属性 `xMin`、`xMax`、`yMin` 和 `yMax`。

说明

方法；返回作为实例的最小和最大 `x`、`y` 坐标值的属性，该实例是由 *my_mc* 为 *targetCoordinateSpace* 参数指定的。

注意：分别使用 `MovieClip.localToGlobal()` 和 `MovieClip.globalToLocal()` 将影片剪辑的本地坐标转换为舞台坐标或将舞台坐标转换为本地坐标。

示例

在下面的示例中，将 `getBounds()` 返回的对象分配给标识符 `clipBounds`。然后可以访问每个属性的值并将其用在脚本中。在该脚本中，将另一个影片剪辑实例 `clip2` 与 `clip` 并排放置。

```
clipBounds = clip.getBounds(_root);  
clip2._x = clipBounds.xMax;
```

另请参见

[MovieClip.globalToLocal\(\)](#), [MovieClip.localToGlobal\(\)](#)

MovieClip.getBytesLoaded()

可用性

Flash Player 5。

用法

```
my_mc.getBytesLoaded()
```

参数

无。

返回

一个整数，指示所加载的字节数。

说明

方法；返回已为 *my_mc* 指定的影片剪辑加载（流处理）的字节数。您可以将此值与 [MovieClip.getBytesTotal\(\)](#) 返回的值进行比较以确定已加载影片剪辑的百分比。

另请参见

[MovieClip.getBytesTotal\(\)](#)

MovieClip.getBytesTotal()

可用性

Flash Player 5。

用法

```
my_mc.getBytesTotal()
```

参数

无。

返回

一个整数，指示 *my_mc* 的总大小（以字节为单位）。

说明

方法；以字节为单位返回 *my_mc* 指定的影片剪辑的大小。对于那些外部的影片剪辑（加载到某个目标或某个级别的根 SWF 文件或影片剪辑），返回值为 SWF 文件的大小。

另请参见

[MovieClip.getBytesLoaded\(\)](#)

MovieClip.getDepth()

可用性

Flash Player 6。

用法

```
my_mc.getDepth()
```

参数

无。

返回

一个整数。

说明

方法；返回影片剪辑实例的深度。有关更多信息，请参见第 114 页的“管理影片剪辑的深度”。

另请参见

[MovieClip.getInstanceAtDepth\(\)](#), [MovieClip.getNextHighestDepth\(\)](#),
[MovieClip.swapDepths\(\)](#)

MovieClip.getInstanceAtDepth()

可用性

Flash Player 7。

用法

```
my_mc.getInstanceAtDepth(depth)
```

参数

depth 一个整数，指定要查询的深度级别。

返回

对位于指定深度的 MovieClip 实例的引用，或者如果该深度处没有影片剪辑，则为 undefined。

说明

方法；用于确定特定深度是否已被某个影片剪辑占用。您可以在使用 [MovieClip.attachMovie\(\)](#)、[MovieClip.duplicateMovieClip\(\)](#) 或 [MovieClip.createEmptyMovieClip\(\)](#) 前使用此方法确定要传递给这些方法的深度参数所指定的深度是否已包含影片剪辑。有关更多信息，请参见第 114 页的“管理影片剪辑的深度”。

另请参见

[MovieClip.getDepth\(\)](#), [MovieClip.getNextHighestDepth\(\)](#), [MovieClip.swapDepths\(\)](#)

MovieClip.getNextHighestDepth()

可用性

Flash Player 7。

用法

```
my_mc.getNextHighestDepth()
```

参数

无。

返回

一个整数，反映下一个可用的深度值，采用该深度值的对象将显示在 *my_mc* 中同一级和同一层上所有其它对象之上。

说明

方法；用于确定可传递到 `MovieClip.attachMovie()`、`MovieClip.duplicateMovieClip()` 或 `MovieClip.createEmptyMovieClip()` 的深度值以确保 Flash 将该影片剪辑呈现在当前影片剪辑中同一级和同一层上所有其它对象的前面。返回值为 0 或更大的数字（即，不返回负数）。

有关更多信息，请参见第 114 页的“管理影片剪辑的深度”。

另请参见

`MovieClip.getDepth()`、`MovieClip.getInstanceAtDepth()`、`MovieClip.swapDepths()`

MovieClip.getSWFVersion()

可用性

Flash Player 7。

用法

```
my_mc.getSWFVersion()
```

参数

无。

返回

一个整数，该整数指定在发布加载到 *my_mc* 中的 SWF 文件时以其为目标播放器的 Flash Player 版本。

说明

方法；返回一个整数，指示发布的 *my_mc* 的 Flash Player 版本。如果 *my_mc* 是 JPEG 文件，或者如果出现错误并且 Flash 不能确定 *my_mc* 的 SWF 版本，则返回 -1。

MovieClip.getTextSnapshot()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
my_mc.getTextSnapshot();
```

参数

无。

返回

一个包含来自 my_mc 的静态文本的 TextSnapshot 对象；或者，如果 my_mc 不包含任何静态文本，则为空字符串。

说明

方法：返回一个 TextSnapshot 对象，该对象包含指定影片剪辑的所有静态文本字段中的文本；不包括子影片剪辑中的文本。

Flash 连接文本并将文本放置于 TextSnapshot 对象中，放置的顺序反映影片剪辑中静态文本字段的 Tab 键索引顺序。没有 Tab 键索引值的文本字段将以随机的顺序放置于该对象中，并且放在来自具有 Tab 键索引值的字段的所有文本前。没有换行符或格式指示一个字段从哪里结束、另一个字段从哪里开始。

注意：您不能在 Flash 中为静态文本指定 Tab 键索引值。不过，其它产品可以这样做；例如 Macromedia FlashPaper 就可以。

TextSnapshot 对象的内容不是动态的；即，如果影片剪辑移到不同的帧，或者以某种方式被更改（例如，影片剪辑中的对象被添加或删除），则 TextSnapshot 对象可能不表示影片剪辑中的当前文本。若要确保该对象的内容是最新的，应根据需要重新发出此命令。

另请参见

[TextSnapshot 对象](#)

MovieClip.getURL()

可用性

Flash Player 5。

用法

```
my_mc.getURL(URL [,window, variables])
```

参数

URL 要从中获得文档的 URL 位置。

window 一个可选参数，它指定名称、帧或表达式，这些内容指定文档将加载到其中的窗口或 HTML 框架。也可以使用下列保留的目标名称之一：_self 指定当前窗口中的当前帧，_blank 指定一个新窗口，_parent 指定当前帧的父级，_top 指定当前窗口中的顶级帧。

variables 一个可选参数，指定与要加载 SWF 文件关联的变量的发送方法。如果没有变量，则省略该参数；否则请指定是使用 GET 方法还是使用 POST 方法加载变量。GET 将变量追加到 URL 的末尾，用于发送少量的变量。POST 在单独的 HTTP 标头中发送变量，用于发送大量的变量。

返回

无。

说明

方法；从指定 URL 将文档加载到指定窗口。getUrl 方法还可以通过 GET 或 POST 方法向 URL 中定义的另一个应用程序传递变量。

另请参见

[getUrl\(\)](#)

MovieClip.globalToLocal()

可用性

Flash Player 5。

用法

```
my_mc.globalToLocal(point)
```

参数

point 用通用 [Object](#) 类创建的对象名称或标识符。该对象指定 x 和 y 坐标作为属性。

返回

无。

说明

方法；将 *point* 对象从舞台（全局）坐标转换为影片剪辑的（本地）坐标。

示例

下面的示例将 *point* 对象的全局 x 和 y 坐标转换为影片剪辑的本地坐标。

```
onClipEvent(mouseMove) {  
    point = new object();  
    point.x = _root._xmouse;  
    point.y = _root._ymouse;  
    globalToLocal(point);  
    trace(_root._xmouse + " " + _root._ymouse);  
    trace(point.x + " " + point.y);  
    updateAfterEvent();  
}
```

另请参见

[MovieClip.getBounds\(\)](#), [MovieClip.localToGlobal\(\)](#)

MovieClip.gotoAndPlay()

可用性

Flash Player 5。

用法

```
my_mc.gotoAndPlay(frame)
```

参数

frame 表示将播放头转到的帧编号的数字，或者表示将播放头转到的帧的标签的字符串。

返回

无。

说明

方法；从指定帧开始播放 SWF 文件。如果您要指定场景以及帧，则使用 [gotoAndPlay\(\)](#)。

MovieClip.gotoAndStop()

可用性

Flash Player 5。

用法

```
my_mc.gotoAndStop(frame)
```

参数

frame 播放头要转到的帧的编号。

返回

无。

说明

方法；将播放头移到该影片剪辑的指定帧并停在那里。

另请参见

[gotoAndStop\(\)](#)

MovieClip._height

可用性

Flash Player 4。

用法

my_mc._height

说明

属性；影片剪辑的高度（以像素为单位）。

示例

下面的代码示例在用户单击鼠标按钮时设置影片剪辑的高度和宽度。

```
onClipEvent(mouseDown) {  
    _width=200;  
    _height=200;  
}
```

MovieClip._highquality

可用性

Flash Player 6。

用法

my_mc._highquality

说明

属性（全局）；指定应用于当前 SWF 文件的锯齿消除级别。指定 2（最高品质）可应用高品质，并使位图平滑处理始终打开。指定 1（高品质），则应用消除锯齿功能；如果 SWF 文件不包含动画，这将对位图进行平滑处理。指定 0（低品质），则不消除锯齿。此属性可以覆盖全局 [_highquality](#) 属性。

示例

```
my_mc._highquality = 2;
```

另请参见

[_quality](#)

MovieClip.hitArea

可用性

Flash Player 6。

用法

```
my_mc.hitArea
```

返回

对影片剪辑的引用。

说明

属性；将另一个影片剪辑指定为按钮影片剪辑的点击区域。如果 `hitArea` 属性不存在或者为 `null` 或 `undefined`，则按钮影片剪辑自身将用作点击区域。`hitArea` 属性的值可以是对影片剪辑对象的引用。

可以在任何时候更改 `hitArea` 属性；修改后的按钮影片剪辑会立即获得新的点击区域行为。指定为点击区域的影片剪辑不必是可见的；虽然有的点击区域不可见，但可以点击测试其图形形状。可以从原型对象读出 `hitArea` 属性。

MovieClip.hitTest()

可用性

Flash Player 5。

用法

```
my_mc.hitTest(x, y, shapeFlag)
```

```
my_mc.hitTest(target)
```

参数

x 舞台上点击区域的 *x* 坐标。

y 舞台上点击区域的 *y* 坐标。

x 和 *y* 坐标都在全局坐标空间中定义。

target 可能与由 *my_mc* 指定的实例交叉或重叠的点击区域的目标路径。*target* 参数通常表示一个按钮或文本输入字段。

shapeFlag 一个布尔值，指定是计算指定实例的整个形状 (*true*) 还是仅计算边框 (*false*)。只有当用 *x* 和 *y* 坐标参数标识点击区域时，才可以指定该参数。

返回

如果 *my_mc* 与指定的点击区域重叠，则为布尔值 `true`，否则为 `false`。

说明

方法；计算由 *my_mc* 指定的实例以确定它与由 *target* 或 *x* 和 *y* 坐标参数所标识的点击区域是否重叠或交叉。

用法 1：按照 *shapeFlag* 的设置，将 *x* 和 *y* 坐标与指定实例的形状或边框进行比较。如果 *shapeFlag* 设置为 `true`，则只计算舞台上实例实际占据的区域，如果 *x* 和 *y* 重叠任意一点，则返回 `true` 值。这对于确定影片剪辑是否在指定的点击区域或热点区域中很有用。

用法 2：计算 *target* 与指定实例的边框，如果它们在任意一点上重叠或交叉，则返回 `true`。

示例

下面的示例使用带有 `_xmouse` 和 `_ymouse` 属性的 `hitTest()` 来确定鼠标指针是否位于目标的边框上方：

```
if (hitTest( _root._xmouse, _root._ymouse, false));
```

下面的示例使用 `hitTest()` 来确定影片剪辑 `ball` 是否与影片剪辑 `square` 重叠或交叉：

```
if(_root.ball.hitTest(_root.square)){
    trace("ball intersects square");
}
```

另请参见

[MovieClip.getBounds\(\)](#), [MovieClip.globalToLocal\(\)](#), [MovieClip.localToGlobal\(\)](#)

MovieClip.lineStyle()

可用性

Flash Player 6。

用法

```
my_mc.lineStyle([thickness[, rgb[, alpha]]])
```

参数

thickness 一个整数，以磅为单位指示线条的粗细；有效值为 0 到 255。如果未指定数值，或者该参数为 `undefined`，则不绘制线条。如果传递的值小于 0，则 Flash 使用 0。数值 0 指示极细的粗细；最大粗细为 255。如果传递的值大于 255，则 Flash 解释程序使用 255。

rgb 线条的十六进制颜色值（例如，红色为 `0xFF0000`，蓝色为 `0x0000FF`，等等）。如果未指示该值，则 Flash 使用 `0x000000`（黑色）。

alpha 一个整数，指示线条颜色的 alpha 值；有效值为 0 到 100。如果未指示该值，则 Flash 使用 100（纯色）。如果该值小于 0，则 Flash 使用 0。如果该值大于 100，则 Flash 使用 100。

返回

无。

说明

方法；指定一个线条样式，供 Flash 用于对 `lineTo()` 和 `curveTo()` 的后续调用，直到使用不同的参数调用 `lineStyle()` 为止。可以在绘制路径的中间调用 `lineStyle()` 来为路径中的不同线条段指定不同的样式。

注意：调用 `clear` 会将 `lineStyle()` 重置为 `undefined`。

示例

下面的代码用 5 磅、纯洋红色线条绘制一个无填充的三角形。

```
_root.createEmptyMovieClip( "triangle", 1 );
with ( _root.triangle )
{
    lineStyle( 5, 0xff00ff, 100 );
    moveTo( 200, 200 );
```

```
lineTo( 300,300 );
lineTo( 100, 300 );
lineTo( 200, 200 );
}
```

另请参见

[MovieClip.beginFill\(\)](#), [MovieClip.beginGradientFill\(\)](#), [MovieClip.clear\(\)](#),
[MovieClip.curveTo\(\)](#), [MovieClip.lineTo\(\)](#), [MovieClip.moveTo\(\)](#)

MovieClip.lineTo()

可用性

Flash Player 6。

用法

```
my_mc.lineTo(x, y)
```

参数

x 一个整数，指示相对于父级影片剪辑的注册点的水平位置。

y 一个整数，指示相对于父级影片剪辑的注册点的垂直位置。

返回

无。

说明

方法；使用当前线条样式从当前绘画位置向 (*x*, *y*) 绘制线条；当前绘画位置随后被设置为 (*x*, *y*)。如果正在其中绘制的影片剪辑包含用 Flash 绘画工具创建的内容，则调用 `lineTo()` 将在该内容下面进行绘制。如果在对 `moveTo()` 方法进行任何调用之前调用了 `lineTo()`，则当前绘画位置默认为 (0, 0)。如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

示例

下面的示例绘制一个没有线条、具有半透明蓝色填充的三角形。

```
_root.createEmptyMovieClip( "triangle", 1 );
with (_root.triangle){
    beginFill (0x0000FF, 50);
    lineStyle (5, 0xFF00FF, 100);
    moveTo (200, 200);
   .lineTo (300, 300);
   .lineTo (100, 300);
   .lineTo (200, 200);
    endFill();
}
```

另请参见

[MovieClip.beginFill\(\)](#), [MovieClip.createEmptyMovieClip\(\)](#), [MovieClip.endFill\(\)](#),
[MovieClip.lineStyle\(\)](#), [MovieClip.moveTo\(\)](#)

MovieClip.loadMovie()

可用性

Flash Player 5。

用法

```
my_mc.loadMovie("url" [,variables])
```

参数

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

variables 一个可选参数，指定发送或加载变量所使用的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

返回

无。

说明

方法；在播放原始 SWF 文件的时候，将 SWF 文件或 JPEG 文件加载到 Flash Player 中的影片剪辑中。

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

如果不使用 `loadMovie()` 方法，则 Flash Player 显示单个 SWF 文件，然后关闭。使用 `loadMovie()` 方法可以一次显示几个 SWF 文件并且无需加载另一个 HTML 文档即可在 SWF 文件间进行切换。

加载到影片剪辑的 SWF 文件或图像会继承该影片剪辑的位置、旋转和缩放属性。可以用影片剪辑的目标路径来定位加载的 SWF 文件。

使用 `unloadMovie()` 方法删除用 `loadMovie()` 方法加载的 SWF 文件或图像。使用 `loadVariables()` 方法可以保持 SWF 文件处于活动状态，并用新值更新变量。

另请参见

```
loadMovie(), loadMovieNum(), MovieClip.loadVariables(), MovieClip.unloadMovie(),  
unloadMovie(), unloadMovieNum()
```

MovieClip.loadVariables()

可用性

Flash Player 5；行为在 Flash Player 7 中发生了变化。

用法

```
my_mc.loadVariables("url", variables)
```

参数

url 包含要加载变量的外部文件的绝对或相对 URL。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

variables 可选参数，指定发送变量所使用的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

返回

无。

说明

方法；从外部文件读取数据并设置 *my_mc* 中变量的值。外部文件可以由 CGI 脚本、Active Server Page (ASP) 或 PHP 脚本生成的文本文件，并且可以包含任意数量的变量。

此方法还可用于使用新值更新活动影片剪辑中的变量。

此方法要求 URL 的文本使用标准的 MIME 格式：application/x-www-form-urlencoded（CGI 脚本格式）。

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 www.someDomain.com 的 SWF 文件可以从位于 store.someDomain.com 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 someDomain.com 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“Flash Player 安全功能”）。例如，位于 www.someDomain.com 的 SWF 文件只能从同样位于 www.someDomain.com 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

另请参见

[loadMovie\(\)](#), [loadVariables\(\)](#), [loadVariablesNum\(\)](#), [MovieClip.unloadMovie\(\)](#)

MovieClip.localToGlobal()

可用性

Flash Player 5。

用法

```
my_mc.localToGlobal(point)
```

参数

point 用 [Object](#) 类创建的对象名称或标识符，并指定 x 和 y 坐标作为属性。

返回

无。

说明

方法；将 *point* 对象从影片剪辑（本地）坐标转换为舞台（全局）坐标。

示例

下面的示例将 *point* 对象的 x 和 y 坐标从影片剪辑（本地）坐标转换为舞台（全局）坐标。本地 x 和 y 坐标是用 `_xmouse` 和 `_ymouse` 属性指定的，以获取鼠标指针位置的 x 和 y 坐标。

```
onClipEvent(mouseMove) {
```



```

point = new object();
point.x = _xmouse;
point.y = _ymouse;
_root.out3 = point.x + " == " + point.y;
_root.out = _root._xmouse + " == " + _root._ymouse;
localToGlobal(point);
_root.out2 = point.x + " == " + point.y;
updateAfterEvent();
}

```

另请参见

[MovieClip.globalToLocal\(\)](#)

MovieClip._lockroot

可用性

Flash Player 7。

用法

my_mc._lockroot

说明

属性；指定将 SWF 文件加载到影片剪辑中时 `_root` 所指的内容。默认情况下，`_lockroot` 属性为 `undefined`。您可以在正在被加载的 SWF 文件中或正在加载影片剪辑的处理函数中设置此属性。

例如，假设有一个名为 `Games.fla` 的文档，该文档让用户选择要玩的游戏并将该游戏（例如 `Chess.swf`）加载到 `game_mc` 影片剪辑中。您要确保如果 `_root` 用于 `Chess.swf` 中，在被加载到 `Games.swf` 中之后，它仍然是指 `Chess.swf` 中的 `_root`。如果您可以访问 `Chess.fla` 并将其发布到 Flash Player 7 或更高版本，则可以向其添加以下语句：

```
this._lockroot = true;
```

如果您无法访问 `Chess.fla`（例如，如果您从其他人的站点加载 `Chess.swf`），则可以在加载时设置其 `_lockroot` 属性，如下所示。在这种情况下，只要向 Flash Player 7 或更高版本发布 `Games.swf`，便可以向任何版本的 Flash Player 发布 `Chess.swf`。

```

onClipEvent (load)
{
    this._lockroot = true;
}
game_mc.loadMovie ("Chess.swf");

```

如果在所有 SWF 文件中都没有使用 `this._lockroot = true` 语句，则在将 `Chess.swf` 加载到 `Games.swf` 中之后，`Chess.swf` 中的 `_root` 将引用 `Games.swf` 中的 `_root`。

另请参见

[_root](#), [MovieClip.attachMovie\(\)](#), [MovieClip.loadMovie\(\)](#)

MovieClip.menu

可用性

Flash Player 7。

用法

```
my_mc.menu = contextMenu
```

参数

contextMenu 一个 ContextMenu 对象。

说明

属性；将指定的 ContextMenu 对象与影片剪辑 *my_mc* 关联起来。ContextMenu 类用于修改当用户在 Flash Player 中右击 (Windows) 或按住 Control 键单击 (Macintosh) 时显示的上下文菜单。

示例

以下示例将 ContextMenu 对象 *menu_cm* 指定给影片剪辑 *content_mc*。ContextMenu 对象包含一个标有 “Print...” 的自定义菜单项，该菜单项具有名为 *doPrint()* 的关联回调处理函数。

```
var menu_cm = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Print...", doPrint));
function doPrint(menu, obj) {
    // 此处为打印代码
}
content_mc.menu = menu_cm;
```

另请参见

[Button.menu](#), [ContextMenu](#) 类, [ContextMenuItem](#) 类, [TextField.menu](#)

MovieClip.moveTo()

可用性

Flash Player 6。

用法

```
my_mc.moveTo(x, y)
```

参数

x 一个整数，指示相对于父级影片剪辑的注册点的水平位置。

y 一个整数，指示相对于父级影片剪辑的注册点的垂直位置。

返回

无。

说明

方法；将当前绘画位置移动到 (*x*, *y*)。如果缺少任何一个参数，则此方法将失败，并且当前绘画位置不改变。

示例

该示例使用 5 磅的纯洋红色线条绘制无填充的三角形。第一行创建一个要进行绘制的空影片剪辑。在 `with` 语句内部，定义一个线条类型，然后由 `moveTo()` 方法指示起始绘画位置。

```
_root.createEmptyMovieClip( "triangle", 1 );
with ( _root.triangle )
{
    lineStyle( 5, 0xff00ff, 100 );
    moveTo( 200, 200 );
    lineTo( 300,300 );
    lineTo( 100, 300 );
    lineTo( 200, 200 );
}
```

另请参见

[MovieClip.createEmptyMovieClip\(\)](#), [MovieClip.lineStyle\(\)](#), [MovieClip.lineTo\(\)](#)

MovieClip._name

可用性

Flash Player 4。

用法

my_mc._name

说明

属性；由 *my_mc* 指定的影片剪辑的实例名称。

MovieClip.nextFrame()

可用性

Flash Player 5。

用法

my_mc.nextFrame()

参数

无。

返回

无。

说明

方法；将播放头转到下一帧并停止。

另请参见

[nextFrame\(\)](#)

MovieClip.onData

可用性

Flash Player 6。

用法

```
my_mc.onData = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当影片剪辑从 `loadVariables()` 或 `loadMovie()` 调用接收数据时调用。必须定义一个在调用事件处理函数时执行的函数。

此处理函数只能与某些影片剪辑一起使用，对于这些影片剪辑，它们在库中有与类关联的元件。如果要在特定影片剪辑收到数据时调用事件处理函数，则必须使用 `onClipEvent(data)` 而不能使用此处理函数。当任何影片剪辑收到数据时，都会调用处理函数 `onClipEvent(data)`。

示例

下面的示例说明了 `MovieClip.onData()` 和 `onClipEvent(data)` 的正确用法。

```
// symbol_mc 是库中的影片剪辑元件。  
// 它被链接到 MovieClip 类。  
// 对于 symbol_mc 的每个实例，在这些实例被实例化并显示在时间轴上将触发  
// 下面的函数。  
symbol_mc.onData = function() {  
    trace("The movie clip has received data");  
}  
  
// dynamic_mc 是使用 MovieClip.loadMovie() 加载的影片剪辑。  
// 此代码尝试在加载该剪辑时调用函数，  
// 但此代码不起作用，因为加载的 SWF 不是库中与 MovieClip 类关联的  
// 元件。  
function output()  
{  
    trace("Will never be called.");  
}  
dynamic_mc.onData = output;  
dynamic_mc.loadMovie("replacement.swf");  
  
// 对于任何显示在时间轴上的影片剪辑，系统都将调用下面的函数，  
// 而不管该影片剪辑是否存在于库中。  
// 因此，当实例化 symbol_mc 时  
// 以及加载 replacement.swf 时都将调用此函数。  
onClipEvent( data ) {  
    trace("The movie clip has received data");  
}
```

另请参见

[onClipEvent\(\)](#)

MovieClip.onDragOut

可用性

Flash Player 6。

用法

```
my_mc.onDragOut = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当按下鼠标按钮并且指针滑出对象时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 onDragOut 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onDragOut = function () {  
    trace ("onDragOut called");  
};
```

另请参见

[MovieClip.onDragOver](#)

MovieClip.onDragOver

可用性

Flash Player 6。

用法

```
my_mc.onDragOver = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当鼠标指针在影片剪辑外拖动并且随后拖过该影片剪辑时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onDragOver` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onDragOver = function () {  
    trace ("onDragOver called");  
};
```

另请参见

[MovieClip.onDragOut](#)

MovieClip.onEnterFrame

可用性

Flash Player 6。

用法

```
my_mc.onEnterFrame = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；以 SWF 文件的帧频持续调用。首先处理与 `enterFrame` 剪辑事件关联的动作，然后才处理附加到受影响帧的所有帧动作。

必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onEnterFrame` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onEnterFrame = function () {  
    trace ("onEnterFrame called");  
};
```

MovieClip.onKeyDown

可用性

Flash Player 6。

用法

```
my_mc.onKeyDown = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当影片剪辑具有输入焦点并有按键被按下时调用。调用 onKeyDown 事件处理函数时没有参数。您可以使用 [Key.getAscii\(\)](#) 和 [Key.getCode\(\)](#) 方法来确定按下了哪个键。必须定义一个在调用事件处理函数时执行的函数。

仅当影片剪辑的输入焦点被启用和设置后，onKeyDown 事件处理函数才能工作。首先，必须将影片剪辑的 `focusEnabled` 属性设置为 `true`。然后，必须使该剪辑获得焦点。这可以通过使用 [Selection.setFocus\(\)](#) 或通过设置 Tab 键以导航至该剪辑来实现。

如果使用 [Selection.setFocus\(\)](#)，则必须将影片剪辑的路径传递给 [Selection.setFocus\(\)](#)。当鼠标移动后，非常容易让其它元素重新获得焦点。

示例

下面的示例为 onKeyDown() 方法定义一个函数，该函数将 [trace\(\)](#) 动作发送到“输出”面板。

```
my_mc.onKeyDown = function () {  
    trace ("onKeyDown called");  
};
```

以下示例设置输入焦点。

```
MovieClip.focusEnabled = true;  
Selection.setFocus(MovieClip);
```

另请参见

[MovieClip.onKeyUp](#)

MovieClip.onKeyUp

可用性

Flash Player 6。

用法

```
my_mc.onKeyUp = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当释放按键时调用。调用 onKeyUp 事件处理函数时没有参数。您可以使用 [Key.getAscii\(\)](#) 和 [Key.getCode\(\)](#) 方法来确定按下了哪个键。必须定义一个在调用事件处理函数时执行的函数。

仅当影片剪辑的输入焦点被启用和设置后，onKeyUp 事件处理函数才能工作。首先，必须将影片剪辑的 `focusEnabled` 属性设置为 `true`。然后，必须使该剪辑获得焦点。这可以通过使用 [Selection.setFocus\(\)](#) 或通过设置 Tab 键以导航至该剪辑来实现。

如果使用 [Selection.setFocus\(\)](#)，则必须将影片剪辑的路径传递给 [Selection.setFocus\(\)](#)。当鼠标移动后，非常容易让其它元素重新获得焦点。

示例

下面的示例为 onKeyUp 方法定义一个函数，该函数将 [trace\(\)](#) 动作发送到“输出”面板。

```
my_mc.onKeyUp = function () {  
    trace ("onKeyUp called");  
};
```

以下示例设置输入焦点：

```
MovieClip.focusEnabled = true;  
Selection.setFocus(MovieClip);
```


MovieClip.onKillFocus

可用性

Flash Player 6。

用法

```
my_mc.onKillFocus = function (newFocus) {  
    // 此处是您的语句  
}
```

参数

newFocus 要接收键盘焦点的对象。

返回

无。

说明

事件处理函数；当影片剪辑失去键盘焦点时调用。onKillFocus 方法接收一个参数 *newFocus*，该参数是一个对象，表示要接收焦点的新对象。如果没有对象接收焦点，则 *newFocus* 将包含值 null。

MovieClip.onLoad

可用性

Flash Player 6。

用法

```
my_mc.onLoad = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当影片剪辑被实例化并出现在时间轴上时调用。必须定义一个在调用事件处理函数时执行的函数。

此处理函数只能与某些影片剪辑一起使用，对于这些影片剪辑，它们在库中有与类关联的元件。如果要在特定影片剪辑加载时（例如当使用 `MovieClip.loadMovie()` 动态加载 SWF 文件时）调用事件处理函数，必须使用 `onClipEvent(load)` 而不能使用此处理函数。当任何影片剪辑加载时，都会调用处理函数 `onClipEvent(load)`。

示例

下面的示例说明了 `MovieClip.onLoad()` 和 `onClipEvent(load)` 的正确用法。

```
// symbol_mc 是库中的影片剪辑元件。  
// 它被链接到 MovieClip 类。  
// 对于 symbol_mc 的每个实例，在这些实例被实例化并显示在时间轴上将触发  
// 下面的函数。  
symbol_mc.onLoad = function() {  
    trace("The movie clip is loaded");  
}  
  
// dynamic_mc 是使用 MovieClip.loadMovie() 加载的影片剪辑。  
// 此代码尝试在加载该剪辑时调用函数，  
// 但此代码不起作用，因为加载的 SWF 不是库中与 MovieClip 类关联的  
// 元件。  
function output()  
{  
    trace("Will never be called.");  
}  
dynamic_mc.onLoad = output;  
dynamic_mc.loadMovie("replacement.swf");  
  
// 对于任何显示在时间轴上的影片剪辑，系统都将调用下面的函数，  
// 而不管该影片剪辑是否存在于库中。  
// 因此，当实例化 symbol_mc 时  
// 以及加载 replacement.swf 时都将调用此函数。  
onClipEvent( load ) {  
    trace("The movie clip is loaded");  
}
```

另请参见

[onClipEvent\(\)](#)

MovieClip.onMouseDown

可用性

Flash Player 6。

用法

```
my_mc.onMouseDown = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当按下鼠标按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onMouseDown` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onMouseDown = function () {  
    trace ("onMouseDown called");  
}
```

MovieClip.onMouseMove

可用性

Flash Player 6。

用法

```
my_mc.onMouseMove = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当鼠标移动时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onMouseMove` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onMouseMove = function () {  
    trace ("onMouseMove called");  
};
```

MovieClip.onMouseUp

可用性

Flash Player 6。

用法

```
my_mc.onMouseUp = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当释放鼠标按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onMouseUp` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onMouseUp = function () {  
    trace ("onMouseUp called");  
};
```

MovieClip.onPress

可用性

Flash Player 6。

用法

```
my_mc.onPress = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当鼠标指针处于影片剪辑之上而用户单击鼠标按钮时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onPress` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onPress = function () {  
    trace ("onPress called");  
};
```

MovieClip.onRelease

可用性

Flash Player 6。

用法

```
my_mc.onRelease = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当释放按钮影片剪辑时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onPress` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onRelease = function () {  
    trace ("onRelease called");  
};
```

MovieClip.onReleaseOutside

可用性

Flash Player 6。

用法

```
my_mc.onReleaseOutside = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；该函数在这种情况下调用：即在影片剪辑内部按下鼠标按钮后，鼠标指针在该影片剪辑外面时释放鼠标按钮。

必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onReleaseOutside` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onReleaseOutside = function () {  
    trace ("onReleaseOutside called");  
};
```

MovieClip.onRollOut

可用性

Flash Player 6。

用法

```
my_mc.onRollOut = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当鼠标指针移出影片剪辑区域的外面时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onRollOut` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onRollOut = function () {  
    trace ("onRollOut called");  
};
```

MovieClip.onRollOver

可用性

Flash Player 6。

用法

```
my_mc.onRollOver = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当鼠标指针滑过影片剪辑区域时调用。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `onRollOver` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onRollOver = function () {  
    trace ("onRollOver called");  
};
```

MovieClip.onSetFocus

可用性

Flash Player 6。

用法

```
my_mc.onSetFocus = function(oldFocus){  
    // 此处是您的语句  
}
```

参数

oldFocus 将失去焦点的对象。

返回

无。

说明

事件处理函数；当影片剪辑接收键盘焦点时调用。*oldFocus* 参数是失去焦点的对象。例如，如果用户按 Tab 键将输入焦点从影片剪辑移到文本字段，则 *oldFocus* 包含该影片剪辑实例。

如果以前没有具有焦点的对象，则 *oldFocus* 包含 `null` 值。

MovieClip.onUnload

可用性

Flash Player 6。

用法

```
my_mc.onUnload = function() {  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；从时间轴删除影片剪辑后，在第 1 帧中调用。Flash 先处理与 `onUnload` 事件处理函数关联的动作，然后将动作附加到受影响的帧。必须定义一个在调用事件处理函数时执行的函数。

示例

下面的示例为 `MovieClip.onUnload` 方法定义一个函数，该函数将 `trace()` 动作发送到“输出”面板。

```
my_mc.onUnload = function () {  
    trace ("onUnload called");  
};
```

MovieClip._parent

可用性

Flash Player 5。

用法

```
my_mc._parent.property  
_parent.property
```

说明

属性；指向包含当前影片剪辑或对象的影片剪辑或对象的引用。当前对象是包含引用 `_parent` 的动作脚本代码的对象。使用 `_parent` 属性来指定一个相对路径，该路径指向当前影片剪辑或对象上级的影片剪辑或对象。

可以使用 `_parent` 在显示列表中攀升多个级别，如下所示：

```
_parent._parent._alpha = 20;
```

另请参见

[Button._parent](#), [_root](#), [targetPath](#), [TextField._parent](#)

MovieClip.play()

可用性

Flash Player 5。

用法

```
my_mc.play()
```

参数

无。

返回

无。

说明

方法：在影片剪辑的时间轴中移动播放头。

另请参见

[play\(\)](#)

MovieClip.prevFrame()

可用性

Flash Player 5。

用法

```
my_mc.prevFrame()
```

参数

无。

返回

无。

说明

方法；将播放头转到前一帧并停止。

另请参见

[prevFrame\(\)](#)

MovieClip.removeMovieClip()

可用性

Flash Player 5。

用法

```
my_mc.removeMovieClip()
```

参数

无。

返回

无。

说明

方法；删除用 [duplicateMovieClip\(\)](#)、[MovieClip.duplicateMovieClip\(\)](#) 或 [MovieClip.attachMovie\(\)](#) 创建的影片剪辑实例。

MovieClip._rotation

可用性

Flash Player 4。

用法

```
my_mc._rotation
```

说明

属性；影片剪辑距其原始方向的旋转程度（以度为单位）。从 0 到 180 的值表示顺时针方向的旋转；从 0 到 -180 的值表示逆时针方向的旋转。对于此范围之外的值，可以通过加上或减去 360 获得该范围内的值。例如，语句 `my_mc._rotation = 450` 与 `my_mc._rotation = 90` 相同。

另请参见

[Button._rotation](#), [TextField._rotation](#)

MovieClip.setMask()

可用性

Flash Player 6。

用法

```
my_mc.setMask(mask_mc)
```

参数

my_mc 要使用遮罩的影片剪辑的实例名称。

mask_mc 将成为遮罩的影片剪辑的实例名称。

返回

无。

说明

方法；使参数 *mask_mc* 中的影片剪辑成为遮罩，该遮罩展示由 *my_mc* 参数指定的影片剪辑。

此方法允许将带有复杂的多图层内容的多帧影片剪辑用作遮罩。可以在运行时打开或关闭遮罩。但是，不能将同一遮罩用于多个遮罩受体（如果使用遮罩层，则这是可能的）。如果在使用遮罩的影片剪辑中有设备字体，则它们可以进行绘制但不使用遮罩。不能将影片剪辑设置为其自己的遮罩，例如 `my_mc.setMask(my_mc)`。

如果创建包含影片剪辑的遮罩层，然后对其应用 `setMask()` 方法，则 `setMask()` 调用会获得优先权并且这是不可逆的。例如，在名为 `UIMask` 的遮罩层中有一个影片剪辑，该影片剪辑对包含名为 `UIMaskee` 的另一个影片剪辑的其它图层使用遮罩。如果在 SWF 文件播放时调用 `UIMask.setMask(UIMaskee)`，则从这一刻开始，由 `UIMaskee` 对 `UIMask` 使用遮罩。

若要取消用动作脚本创建的遮罩，可以将值 `null` 传递给 `setMask()` 方法。下面的代码将在不影响时间轴中的遮罩层的情况下取消遮罩。

```
UIMask.setMask(null);
```

示例

下面的代码使用影片剪辑 `circleMask_mc` 对影片剪辑 `theMaskee_mc` 使用遮罩。

```
theMaskee_mc.setMask(circleMask_mc);
```

MovieClip._soundbuftime

可用性

Flash Player 6。

用法

```
my_mc._soundbuftime
```

说明

属性（全局）；一个整数，指定在声音开始进入流之前，预先缓冲的秒数。

MovieClip.startDrag()

可用性

Flash Player 5。

用法

```
my_mc.startDrag([lock, [left, top, right, bottom]])
```

参数

lock 一个布尔值，指定可拖动影片剪辑是锁定到鼠标位置中央 (`true`)，还是锁定到用户首次单击该影片剪辑的位置上 (`false`)。此参数是可选的。

left、*top*、*right*、*bottom* 相对于影片剪辑父级坐标的值，这些值指定该影片剪辑的约束矩形。这些参数是可选的。

返回

无。

说明

方法；允许用户拖动指定的影片剪辑。该影片剪辑将一直保持可拖动，直到通过对 [MovieClip.stopDrag\(\)](#) 的调用明确停止为止，或者直到另一个影片剪辑变为可拖动为止。在同一时间只有一个影片剪辑是可拖动的。

另请参见

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

MovieClip.stop()

可用性

Flash Player 5。

用法

```
my_mc.stop()
```

参数

无。

返回

无。

说明

方法；停止当前正在播放的影片剪辑。

另请参见

[stop\(\)](#)

MovieClip.stopDrag()

可用性

Flash Player 5。

用法

```
my_mc.stopDrag()
```

参数

无。

返回

无。

说明

方法；终止 [MovieClip.startDrag\(\)](#) 方法。用该方法变为可拖动的影片剪辑将一直保持可拖动状态，直到添加了 [stopDrag\(\)](#) 方法为止，或者直到另一个影片剪辑变为可拖动为止。在同一时间只有一个影片剪辑是可拖动的。

另请参见

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

MovieClip.swapDepths()

可用性

Flash Player 5。

用法

```
my_mc.swapDepths(depth)  
my_mc.swapDepths(target)
```

参数

depth 一个数字，指定 *my_mc* 将被放置的深度级别。

target 一个字符串，指定其深度将被 *my_mc* 指定的实例交换的影片剪辑实例。两个实例必须具有相同的父级影片剪辑。

返回

无。

说明

方法；用由 *target* 参数指定的影片剪辑或者当前占据着 *depth* 参数中指定的深度级别的影片剪辑交换指定实例 (*my_mc*) 的堆叠，即 z 顺序（深度级别）。两个影片剪辑必须具有相同的父级影片剪辑。交换影片剪辑的深度级别的作用是将一个影片剪辑移到另一个影片剪辑的前面或后面。如果调用该方法时影片剪辑正在补间，则补间会停止。有关更多信息，请参见[第 114 页的“管理影片剪辑的深度”](#)。

另请参见

```
_level, MovieClip.getDepth(), MovieClip.getInstanceAtDepth(),  
MovieClip.getNextHighestDepth()
```

MovieClip.tabChildren

可用性

Flash Player 6。

用法

```
my_mc.tabChildren
```

说明

属性；默认情况下为 `undefined`。如果 `tabChildren` 为 `undefined` 或 `true`，则影片剪辑的子级包含在 Tab 键的自动排序中。如果 `tabChildren` 的值为 `false`，则影片剪辑的子级不包含在 Tab 键的自动排序中。

示例

一个构建为影片剪辑的列表框 UI 窗口小部件包含几个项目。用户可以单击每项来进行选择，因此每项都是一个按钮。但是，只有列表框自身应为 Tab 键停靠位。列表框内部的项目应该排除在 Tab 键排序之外。为实现此目的，该列表框的 `tabChildren` 属性应设置为 `false`。

如果使用 `tabIndex` 属性，则 `tabChildren` 属性不起作用；`tabChildren` 属性只影响 Tab 键的自动排序。

另请参见

[Button.tabIndex](#), [MovieClip.tabEnabled](#), [MovieClip.tabIndex](#), [TextField.tabIndex](#)

MovieClip.tabEnabled

可用性

Flash Player 6。

用法

```
my_mc.tabEnabled
```

说明

属性；指定 `my_mc` 是否包含在 Tab 键的自动排序中。默认情况下它是 `undefined`。

如果 `tabEnabled` 为 `undefined`，则只有它定义至少一个按钮处理函数（例如 [MovieClip.onRelease](#)）后，该对象才包括在 Tab 键的自动排序中。如果 `tabEnabled` 为 `true`，则该对象包括在 Tab 键的自动排序中。如果 `tabIndex` 属性也设置为某个值，则该对象也包括在 Tab 键的自定义排序中。

如果 `tabEnabled` 为 `false`，则即使设置了 `tabIndex` 属性，该对象也不包括在 Tab 键的自动或自定义排序中。但是，如果 [MovieClip.tabChildren](#) 为 `true`，则即使 `tabEnabled` 为 `false`，影片剪辑的子级可能仍包含在 Tab 键的自动排序中。

另请参见

[Button.tabEnabled](#), [MovieClip.tabChildren](#), [MovieClip.tabIndex](#), [TextField.tabEnabled](#)

MovieClip.tabIndex

可用性

Flash Player 6。

用法

```
my_mc.tabIndex
```

说明

属性；使您可以自定义影片中对象的 Tab 键排序。默认情况下，tabIndex 属性为 undefined。可以对按钮、影片剪辑或文本字段实例设置 tabIndex。

如果 SWF 文件中的一个对象包含 tabIndex 属性，则禁用 Tab 键的自动排序，并且使用该 SWF 文件中对象的 tabIndex 属性来计算 Tab 键排序。Tab 键的自定义排序仅包括那些具有 tabIndex 属性的对象。

tabIndex 属性必须为正整数。将根据这些对象的 tabIndex 属性按升序对这些对象进行排序。tabIndex 值为 1 的对象位于 tabIndex 值为 2 的对象之前。Tab 键的自定义排序会忽略 SWF 文件中对象的层次关系。SWF 文件中具有 tabIndex 属性的所有对象都按 Tab 键顺序排列。多个对象不应使用相同的 tabIndex 值。

另请参见

[Button.tabIndex](#), [TextField.tabIndex](#)

MovieClip._target

可用性

Flash Player 4。

用法

```
my_mc._target
```

说明

属性（只读）；返回 my_mc 参数指定的影片剪辑实例的目标路径。

MovieClip._totalframes

可用性

Flash Player 4。

用法

```
my_mc._totalframes
```

说明

属性（只读）；返回 MovieClip 参数中指定的影片剪辑实例中的总帧数。

MovieClip.trackAsMenu

可用性

Flash Player 6。

用法

```
my_mc.trackAsMenu
```

说明

属性；指示其它按钮或影片剪辑是否可接收鼠标按钮释放事件的布尔值属性。这将允许您创建菜单。您可以对任何按钮或影片剪辑对象设置 `trackAsMenu` 属性。如果 `trackAsMenu` 属性不存在，则默认行为是 `false`。

可以在任何时候更改 `trackAsMenu` 属性；修改后的按钮影片剪辑会立即获得新的行为。

另请参见

[Button.trackAsMenu](#)

MovieClip.unloadMovie()

可用性

Flash Player 5。

用法

```
my_mc.unloadMovie()
```

参数

无。

返回

无。

说明

方法；删除影片剪辑实例的内容。保留实例属性和剪辑处理函数。

若要删除实例（包括其属性和剪辑处理函数），请使用 [MovieClip.removeMovieClip\(\)](#)。

另请参见

[MovieClip.attachMovie\(\)](#), [MovieClip.loadMovie\(\)](#), [unloadMovie\(\)](#), [unloadMovieNum\(\)](#)

MovieClip._url

可用性

Flash Player 4。

用法

```
my_mc._url
```

说明

属性（只读）；获取从中下载影片剪辑的 SWF 文件的 URL。

MovieClip.useHandCursor

可用性

Flash Player 6。

用法

```
my_mc.useHandCursor
```

说明

属性；一个布尔值，指示当鼠标滑过按钮影片剪辑时是否显示手形光标（指示手光标）。useHandCursor 的默认值为 true。如果将 useHandCursor 设置为 true，则当鼠标滑过按钮影片剪辑时会显示用于按钮的指示手光标。如果 useHandCursor 为 false，则改用箭头光标。

可以在任何时候更改 useHandCursor 属性；修改后的按钮影片剪辑会立即获得新的光标行为。可以从原型对象读出 useHandCursor 属性。

MovieClip._visible

可用性

Flash Player 4。

用法

```
my_mc._visible
```

说明

属性；一个布尔值，指示 my_mc 指定的影片剪辑是否可见。不可见的影片剪辑（_visible 属性设置为 false）处于禁用状态。例如，不能单击 _visible 设置为 false 的影片剪辑中的按钮。

另请参见

[Button._visible](#), [TextField._visible](#)

MovieClip._width

可用性

Flash Player 4，作为只读属性。

用法

```
my_mc._width
```

说明

属性；影片剪辑的宽度（以像素为单位）。

示例

下面的示例在用户单击鼠标按钮时设置影片剪辑的高度和宽度属性。

```
onClipEvent(mouseDown) {  
    _width=200;  
    _height=200;  
}
```

另请参见

[MovieClip._height](#)

MovieClip._x

可用性

Flash Player 3。

用法

`my_mc._x`

说明

属性；设置影片剪辑 *x* 坐标的整数，该坐标相对于父级影片剪辑的本地坐标。如果影片剪辑在主时间轴中，则其坐标系统将舞台的左上角作为 (0, 0)。如果影片剪辑位于另一个具有变形的影片剪辑中，则该影片剪辑位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 90 度的影片剪辑，该影片剪辑的子级将继承逆时针旋转 90 度的坐标系统。影片剪辑的坐标指的是注册点的位置。

另请参见

[MovieClip._xscale](#), [MovieClip._y](#), [MovieClip._yscale](#)

MovieClip._xmouse

可用性

Flash Player 5。

用法

`my_mc._xmouse`

说明

属性（只读）；返回鼠标位置的 *x* 坐标。

另请参见

[Mouse 类](#), [MovieClip._ymouse](#)

MovieClip._xscale

可用性

Flash Player 4。

用法

`my_mc._xscale`

说明

属性；确定从影片剪辑的注册点开始应用的影片剪辑的水平缩放比例 (*percentage*)。默认注册点为 (0,0)。

缩放本地坐标系统将影响 `_x` 和 `_y` 属性的设置，这些设置是以整像素定义的。例如，如果父级影片剪辑缩放到 50%，则设置 `_x` 属性将移动影片剪辑中的对象，距离为在影片设置为 100% 时其像素数的一半。

另请参见

[MovieClip._x](#), [MovieClip._y](#), [MovieClip._yscale](#)

MovieClip._y

可用性

Flash Player 3。

用法

`my_mc._y`

说明

属性；设置影片剪辑的 y 坐标，该坐标相对于父级影片剪辑的本地坐标。如果影片剪辑在主时间轴中，则其坐标系统将舞台的左上角作为 (0, 0)。如果影片剪辑位于另一个具有变形的影片剪辑中，则该影片剪辑位于包含它的影片剪辑的本地坐标系统中。因此，对于逆时针旋转 90 度的影片剪辑，该影片剪辑的子级将继承逆时针旋转 90 度的坐标系统。影片剪辑的坐标指的是注册点的位置。

另请参见

[MovieClip._x](#), [MovieClip._xscale](#), [MovieClip._yscale](#)

MovieClip._ymouse

可用性

Flash Player 5。

用法

`my_mc._ymouse`

说明

属性（只读）；指示鼠标位置的 y 坐标。

另请参见

[MouseEvent](#) 类, [MovieClip._xmouse](#)

MovieClip._yscale

可用性

Flash Player 4。

用法

`my_mc._yscale`

说明

属性；设置从影片剪辑注册点开始应用的该影片剪辑的垂直缩放比例 (*percentage*)。默认注册点为 (0,0)。

缩放本地坐标系统将影响 `_x` 和 `_y` 属性的设置，这些设置是以整像素定义的。例如，如果将父级影片剪辑放到 50%，则设置 `_x` 属性将移动影片剪辑中的对象，距离为在影片为 100% 时其像素数的一半。

另请参见

[MovieClip._x](#), [MovieClip._xscale](#), [MovieClip._y](#)

MovieClipLoader 类

可用性

Flash Player 7。

说明

此类用于实现在 SWF 或 JPEG 文件正被加载到（下载到）影片剪辑中时提供状态信息的侦听器回调。若要使用 `MovieClipLoader` 功能，请使用 `MovieClipLoader.loadClip()`（而不是 `loadMovie()` 或 `MovieClip.loadMovie()`）来加载 SWF 文件。

在发出 `MovieClipLoader.loadClip()` 命令后，以下事件按下列顺序发生：

- 在下载的文件的第一字节写入磁盘后，调用 `MovieClipLoader.onLoadStart()` 侦听器。
- 如果您已实现了 `MovieClipLoader.onLoadProgress()` 侦听器，则在加载过程中调用它。

注意：您可以在加载过程中随时调用 `MovieClipLoader.getProgress()`。

- 在整个下载的文件都写入磁盘后，调用 `MovieClipLoader.onLoadComplete()` 侦听器。
- 在执行下载的文件的第一帧动作后，调用 `MovieClipLoader.onLoadInit()` 侦听器。

在调用 `MovieClipLoader.onLoadInit()` 后，您可以设置属性，使用方法，以及与加载的影片交互。

如果文件未能完全加载，则调用 `MovieClipLoader.onLoadError()` 侦听器。

MovieClipLoader 类的方法概要

方法	说明
<code>MovieClipLoader.addListener()</code>	注册一个对象，以便在调用 <code>MovieClipLoader</code> 事件处理函数时接收通知。
<code>MovieClipLoader.getProgress()</code>	返回使用 <code>MovieClipLoader.loadClip()</code> 正加载的文件的已加载字节数和该文件的总字节数。

方法	说明
MovieClipLoader.loadClip()	在播放原始影片时，将 SWF 或 JPEG 文件加载到 Flash Player 中的影片剪辑中。
MovieClipLoader.removeListener()	删除已使用 MovieClipLoader.addListener() 注册的对象。
MovieClipLoader.unloadClip()	删除已通过 MovieClipLoader.loadClip() 加载的影片剪辑。

MovieClipLoader 类的侦听器概要

侦听器	说明
MovieClipLoader.onLoadComplete()	在使用 MovieClipLoader.loadClip() 加载的文件已完全下载时调用。
MovieClipLoader.onLoadError()	在使用 MovieClipLoader.loadClip() 加载的文件未能加载时调用。
MovieClipLoader.onLoadInit()	当执行加载的剪辑的第一帧上的动作时调用。
MovieClipLoader.onLoadProgress()	加载过程中每当将加载的内容写入磁盘时调用。
MovieClipLoader.onLoadStart()	当对 MovieClipLoader.loadClip() 的调用已成功开始下载文件时调用。

MovieClipLoader 类的构造函数

可用性

Flash Player 7。

用法

`new MovieClipLoader()`

参数

无。

返回

无。

说明

构造函数；创建一个 `MovieClipLoader` 对象，您可以使用该对象实现许多侦听器，以在正下载 SWF 或 JPEG 文件时对事件做出响应。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.addListener\(\)](#)

MovieClipLoader.addListener()

可用性

Flash Player 7。

用法

```
my_mcl.addListener(listenerObject)
```

参数

listenerObject 侦听来自 MovieClipLoader 事件处理函数的回调通知的对象。

返回

无。

说明

方法；注册一个对象以在调用 MovieClipLoader 事件处理函数时接收通知。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.onLoadComplete\(\)](#), [MovieClipLoader.onLoadError\(\)](#),
[MovieClipLoader.onLoadInit\(\)](#), [MovieClipLoader.onLoadProgress\(\)](#),
[MovieClipLoader.onLoadStart\(\)](#), [MovieClipLoader.removeListener\(\)](#)

MovieClipLoader.getProgress()

可用性

Flash Player 7。

用法

```
my_mcl.getProgress(target_mc)
```

参数

target_mc 一个使用 [MovieClipLoader.loadClip\(\)](#) 加载的 SWF 或 JPEG 文件。

返回

一个具有以下两个整数属性的对象：`bytesLoaded` 和 `bytesTotal`。

说明

方法；返回使用 [MovieClipLoader.loadClip\(\)](#) 正加载的文件的已加载字节数和该文件的总字节数；对于压缩的影片，它反映压缩的字节数。此方法使您可以显式请求此信息，而不是（或除此之外还）编写 [MovieClipLoader.onLoadProgress\(\)](#) 侦听器函数。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.onLoadProgress\(\)](#)

MovieClipLoader.loadClip()

可用性

Flash Player 7。

用法

```
my_mcl.loadClip("url", target )
```

参数

url 要加载的 SWF 文件或 JPEG 文件的绝对或相对 URL。相对路径必须相对于级别 0 处的 SWF 文件。绝对 URL 必须包括协议引用，例如 http:// 或 file:///。文件名不能包括磁盘驱动器说明。

target 影片剪辑的目标路径，或者指定 Flash Player 中影片将加载到的级别的整数。目标影片剪辑将替换为加载的影片或图像。

返回

无。

说明

方法；在播放原始影片时，将 SWF 或 JPEG 文件加载到 Flash Player 中的影片剪辑中。使用此方法可以一次显示几个影片并且无需加载另一个 HTML 文档即可在影片间进行切换。

使用此方法（而不是 `loadMovie()` 或 `MovieClip.loadMovie()`）具有以下若干优点：

- 在加载开始时调用 `MovieClipLoader.onLoadStart()` 处理函数。
- 在无法加载剪辑时调用 `MovieClipLoader.onLoadError()` 处理函数。
- 在加载进程正进行时调用 `MovieClipLoader.onLoadProgress()` 处理函数。
- 在执行该剪辑的第一帧中的动作后调用 `MovieClipLoader.onLoadInit()` 处理函数，以便您可以处理加载的剪辑。

加载到影片剪辑的影片或图像会继承该影片剪辑的位置、旋转和缩放属性。可以用该影片剪辑的目标路径来定位加载的影片。

您可以使用此方法将一个或多个文件加载到单个影片剪辑或级别中；将 `MovieClipLoader` 侦听器对象作为参数传递给正加载的目标影片剪辑实例。或者，您可以为加载的每一文件创建不同的 `MovieClipLoader` 对象。

使用 `MovieClipLoader.unloadClip()` 删除用此方法加载的影片或图像，或者取消正在进行中的加载操作。

示例

以下示例举例说明如何使用多个 `MovieClipLoader` 方法和侦听器。

```
// 首先设置侦听器
var my_mcl = new MovieClipLoader();
myListener = new Object();
myListener.onLoadStart = function (target_mc)
{
    myTrace ("*****First my_mcl instance*****");
    myTrace ("Your load has begun on movie clip . = " + target_mc);
    var loadProgress = my_mcl.getProgress(target_mc);
    myTrace(loadProgress.bytesLoaded + " = bytes loaded at start" );
    myTrace(loadProgress.bytesTotal + " = bytes total at start");
}
```

```

}
myListener.onLoadProgress = function (target_mc, loadedBytes, totalBytes)
{
myTrace ("*****First my_mcl instance Progress*****");
myTrace ("onLoadProgress() called back on movie clip " + target_mc);
myTrace(loadedBytes + " = bytes loaded at progress callback " );
myTrace(totalBytes + " = bytes total at progress callback \n");
}
myListener.onLoadComplete = function (target_mc)
{
myTrace ("*****First my_mcl instance*****");
myTrace ("Your load is done on movie clip = " + target_mc);
var loadProgress = my_mcl.getProgress(target_mc);
myTrace(loadProgress.bytesLoaded + " = bytes loaded at end");
myTrace(loadProgress.bytesTotal + " = bytes total at end=");
}
myListener.onLoadInit = function (target_mc)
{
myTrace ("*****First my_mcl instance*****");
myTrace ("Movie clip = " + target_mc + " is now initialized");
// 您现在可以进行所需的任何设置, 例如:
target_mc._width = 100;
target_mc._width = 100;
}
myListener.onLoadError = function (target_mc, errorCode)
{
myTrace ("*****First my_mcl instance*****");
myTrace ("ERROR CODE = " + errorCode);
myTrace ("Your load failed on movie clip = " + target_mc + "\n");
}
my_mcl.addListener(myListener);
// 现在将这些文件加载到其目标中。
// 加载到影片剪辑中 - 字符串用作目标
my_mcl.loadClip("http://www.somedomain.somewhere.com/
someFile.swf", "_root.myMC");
my_mcl.loadClip("http://www.somedomain.somewhere.com/someOtherFile.swf",
_level0.myMC2");
// 失败的加载
my_mcl.loadClip("http://www.somedomain.somewhere.com/someFile.jpg",
_root.myMC5);

// 加载到影片剪辑中 - 影片剪辑实例用作目标。
my_mcl.loadClip("http://www.somedomain.somewhere.com/someOtherFile.jpg",
_level0.myMC3);

// 加载到 _level1
my_mcl.loadClip("file:///C:/media/images/somePicture.jpg", 1);

// 第二次设置侦听器
var another_mcl = new MovieClipLoader();
myListener2 = new Object();
myListener2.onLoadStart = function (target_mc)
{
myTrace ("*****Second my_mcl instance*****");
myTrace ("Your load has begun on movie clip22 . = " + target_mc);
var loadProgress = my_mcl.getProgress(target_mc);
myTrace(loadProgress.bytesLoaded + " = bytes loaded at start" );
myTrace(loadProgress.bytesTotal + " = bytes total at start");
}
myListener2.onLoadComplete = function (target_mc)
{

```



```

myTrace ("*****Second my_mc1 instance*****");
myTrace ("Your load is done on movie clip = " + target_mc);
var loadProgress = my_mc1.getProgress(target_mc);
myTrace(loadProgress.bytesLoaded + " = bytes loaded at end");
myTrace(loadProgress.bytesTotal + " = bytes total at end" );
}
myListener2.onLoadError = function (target_mc, errorCode)
{
myTrace ("*****Second my_mc1 instance*****");
myTrace ("ERROR CODE = " + errorCode);
myTrace ("Your load failed on movie clip = " + target_mc + "\n");
}
another_mc1.addListener(myListener2);
// 现在将这些文件加载到其目标中 (使用 MovieClipLoader 的第二个实例)
another_mc1.loadClip("http://www.somedomain.somewhere.com/yetAnotherFile.jpg",
_root.myMC4);
// 在完成下载以及已调用 my_mc1.onLoadInit 后
// 发出以下语句。
// my_mc1.removeListener(myListener)
// my_mc1.removeListener(myListener2)

```

另请参见

[MovieClipLoader.unloadClip\(\)](#)

MovieClipLoader.onLoadComplete()

可用性

Flash Player 7。

用法

```

listenerObject.onLoadComplete() = function(target_mc) {
    // 此处是您的语句
}

```

参数

listenerObject 一个已使用 [MovieClipLoader.addListener\(\)](#) 加载的侦听器对象。
target_mc 通过 [MovieClipLoader.loadClip\(\)](#) 方法加载的影片剪辑。

返回

无。

说明

侦听器；在使用 [MovieClipLoader.loadClip\(\)](#) 加载的文件已完全下载时调用。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.addListener\(\)](#), [MovieClipLoader.onLoadStart\(\)](#),
[MovieClipLoader.onLoadError\(\)](#)

MovieClipLoader.onLoadError()

可用性

Flash Player 7。

用法

```
listenerObject.onLoadError() = function(target_mc, errorCode) {  
    // 此处是您的语句  
}
```

参数

listenerObject 一个已使用 [MovieClipLoader.addListener\(\)](#) 加载的侦听器对象。

target_mc 通过 [MovieClipLoader.loadClip\(\)](#) 方法加载的影片剪辑。

errorCode 解释失败原因的字符串。

返回

以下两个字符串之一：“URLNotFound”或“LoadNeverCompleted”。

说明

侦听器；在使用 [MovieClipLoader.loadClip\(\)](#) 加载的文件未能加载时调用。

如果 [MovieClipLoader.onLoadStart\(\)](#) 和 [MovieClipLoader.onLoadComplete\(\)](#) 侦听器均未调用，则返回字符串“URLNotFound”。例如，如果服务器关闭或未找到该文件，则不调用这些侦听器。

如果已调用 [MovieClipLoader.onLoadStart\(\)](#) 但未调用 [MovieClipLoader.onLoadComplete\(\)](#)，则返回字符串“LoadNeverCompleted”。例如，如果调用 [MovieClipLoader.onLoadStart\(\)](#)，但下载由于服务器超载或服务器崩溃等原因而中断，将不调用 [MovieClipLoader.onLoadComplete\(\)](#)。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

MovieClipLoader.onLoadInit()

可用性

Flash Player 7。

用法

```
listenerObject.onLoadInit() = function(target_mc) {  
    // 此处是您的语句  
}
```

参数

listenerObject 一个已使用 [MovieClipLoader.addListener\(\)](#) 加载的侦听器对象。

target_mc 通过 [MovieClipLoader.loadClip\(\)](#) 方法加载的影片剪辑。

返回

无。

说明

侦听器；当执行加载的剪辑的第一帧上的动作时调用。在调用此侦听器后，您可以设置属性，使用方法，以及与加载的影片交互。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.onLoadStart\(\)](#)

MovieClipLoader.onLoadProgress()

可用性

Flash Player 7。

用法

```
listenerObject.onLoadProgress() =  
    function(target_mc [, loadedBytes [, totalBytes ] ] ) {  
        // 此处是您的语句  
    }
```

参数

listenerObject 一个已使用 [MovieClipLoader.addListener\(\)](#) 加载的侦听器对象。

target_mc 通过 [MovieClipLoader.loadClip\(\)](#) 方法加载的影片剪辑。

loadedBytes 在调用该侦听器时已加载的字节数。

totalBytes 在正加载的文件中的总字节数。

返回

无。

说明

侦听器；在加载过程中每当将加载的内容写入磁盘时（即，在 [MovieClipLoader.onLoadStart\(\)](#) 和 [MovieClipLoader.onLoadComplete\(\)](#) 之间）调用。您可以使用此方法以及使用 *loadedBytes* 和 *totalBytes* 参数，显示与下载的进度有关的信息。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.getProgress\(\)](#)

MovieClipLoader.onLoadStart()

可用性

Flash Player 7。

用法

```
listenerObject.onLoadStart() = function(target_mc) {  
    // 此处是您的语句  
}
```

参数

listenerObject 一个已使用 [MovieClipLoader.addListener\(\)](#) 添加的侦听器对象。

target_mc 通过 [MovieClipLoader.loadClip\(\)](#) 方法加载的影片剪辑。

返回

无。

说明

侦听器；在对 [MovieClipLoader.loadClip\(\)](#) 的调用已成功开始下载文件时调用。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

另请参见

[MovieClipLoader.onLoadError\(\)](#), [MovieClipLoader.onLoadInit\(\)](#),
[MovieClipLoader.onLoadComplete\(\)](#)

MovieClipLoader.removeListener()

可用性

Flash Player 7。

用法

```
my_mcl.removeListener(listenerObject)
```

参数

listenerObject 一个已使用 [MovieClipLoader.addListener\(\)](#) 加载的侦听器对象。

返回

无。

说明

方法；删除用来在调用 [MovieClipLoader](#) 事件处理函数时接收通知的对象。

示例

请参见 [MovieClipLoader.loadClip\(\)](#)。

MovieClipLoader.unloadClip()

可用性

Flash Player 7。

用法

```
my_mcl.unloadClip(target)
```

参数

target 传递到对 *my_mcl*.loadClip() 的相应调用的字符串或整数。

返回

无。

说明

方法；删除通过 [MovieClipLoader.loadClip\(\)](#) 已加载的影片剪辑。如果您在正加载影片时发出此命令，则调用 [MovieClipLoader.onLoadError\(\)](#)。

另请参见

[MovieClipLoader.loadClip\(\)](#)

NaN

可用性

Flash Player 5。

用法

NaN

说明

变量；代表 NaN（非数字）的、具有 IEEE-754 标准值的预定义变量。若要确定某个数字是否是 NaN，请使用 [isNaN\(\)](#)。

另请参见

[isNaN\(\)](#), [Number.NaN](#)

ne（不等于 - 字符串专用）

可用性

Flash Player 4。不鼓励使用此运算符，推荐使用 [!=（不等于）](#) 运算符。

用法

expression1 ne *expression2*

参数

expression1、*expression2* 数字、字符串或变量。

返回

一个布尔值。

说明

运算符（比较）；将 *expression1* 与 *expression2* 进行比较，如果 *expression1* 不等于 *expression2*，则返回 true；否则，返回 false。

另请参见

[!=（不等于）](#)

NetConnection 类

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此类。有关更多信息，请参见 Flash Communication Server 文档。

说明

NetConnection 类提供从本地驱动器或 HTTP 地址播放 FLV 文件流的方法。有关视频回放的更多信息，请参见[第 174 页的“动态回放外部 FLV 文件”](#)。

NetConnection 类的方法摘要

方法	说明
NetConnection.connect()	打开您可以通过它从 HTTP 地址或本地文件系统回放视频 (FLV) 文件的本地连接。

NetConnection 类的构造函数

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此类。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
new NetConnection()
```

参数

无。

返回

无。

说明

构造函数；创建 NetConnection 对象，您可以将该对象与 NetStream 对象一起使用来回放本地视频流 (FLV) 文件。在创建 NetConnection 对象后，使用 [NetConnection.connect\(\)](#) 进行实际连接。

与在 Flash 文档中嵌入视频相比，播放外部 FLV 文件有多个好处，例如更好的性能和内存管理以及独立的视频和 Flash 帧频。有关更多信息，请参见[第 174 页的“动态回放外部 FLV 文件”](#)。

另请参见

[NetStream 类](#)，[Video.attachVideo\(\)](#)

NetConnection.connect()

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此方法。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_nc.connect(null);
```

参数

无（您必须传递 null）。

返回

无。

说明

构造函数；打开您可以通过它从 HTTP 地址或本地文件系统回放视频 (FLV) 文件的本地连接。

另请参见

[NetStream 类](#)

NetStream 类

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此类。有关更多信息，请参见 Flash Communication Server 文档。

说明

NetStream 类提供从本地文件系统或 HTTP 地址播放 Flash 视频 (FLV) 文件的方法和属性。您使用 NetStream 对象以通过 NetConnection 对象对视频进行流式处理。与在 Flash 文档中嵌入视频相比，播放外部 FLV 文件有多个好处，例如更好的性能和内存管理以及独立的视频和 Flash 帧频。该类提供若干方法和属性，您可以利用这些方法和属性在一个文件加载和播放时跟踪该文件的进度，以及便于用户控制回放（停止或暂停等）。

有关视频回放的更多信息，请参见第 174 页的“动态回放外部 FLV 文件”。

NetStream 类的方法摘要

NetConnection 和 NetStream 类的以下方法和属性用于控制 FLV 回放。

方法	目的
NetStream.close()	关闭流但不清除视频对象。
NetStream.pause()	暂停或恢复流的回放。
NetStream.play()	开始回放外部视频 (FLV) 文件。
NetStream.seek()	搜寻 FLV 文件中的特定位置。
NetStream.setBufferTime()	指定将数据存入缓冲区多长时间后开始显示流。

NetStream 类的属性摘要

属性	说明
NetStream.bufferLength	数据当前存在于缓冲区中的秒数。
NetStream.bufferTime	只读；由 NetStream.setBufferTime() 分配给缓冲区的秒数。
NetStream.bytesLoaded	只读；已加载到播放器中的数据的字节数。
NetStream.bytesTotal	只读；正加载到播放器中的文件的总大小（以字节为单位）。
NetStream.currentFps	每秒所显示的帧的数目。
NetStream.time	只读；播放头的位置，以秒为单位。

NetStream 类的事件处理函数摘要

事件处理函数	说明
NetStream.onStatus	每当状态更改或发布针对 NetStream 对象的错误时调用。

NetStream 类的构造函数

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此类。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
new NetStream(my_nc)
```

参数

my_nc 一个 NetConnection 对象。

返回

无。

说明

构造函数；创建可用于通过指定的 NetConnection 对象播放 FLV 文件的流。

示例

以下代码首先构造新的 NetConnection 对象 *my_nc*，并且使用它构造称为 *videoStream_ns* 的新 NetStream 对象。

```
my_nc = new NetConnection();  
my_nc.connect(null);  
videoStream_ns = new NetStream(my_nc);
```

另请参见

[NetConnection 类](#)，[NetStream 类](#)，[Video.attachVideo\(\)](#)

NetStream.bufferLength

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此属性。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.bufferLength
```

说明

只读属性；数据当前位于缓冲区的秒数。您可以将此属性与 [NetStream.bufferTime](#) 一起使用以评估如何关闭将要满的缓冲区；例如，要向正等待数据加载到缓冲区中的用户显示反馈。

另请参见

[NetStream.bytesLoaded](#)

NetStream.bufferTime

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此属性。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
myStream.bufferTime
```

说明

只读属性；由 [NetStream.setBufferTime\(\)](#) 分配给缓冲区的秒数。默认值为 .1（十分之一秒）。若要确定当前位于缓冲区中的秒数，请使用 [NetStream.bufferLength](#)。

另请参见

[NetStream.time](#)

NetStream.bytesLoaded

可用性

Flash Player 7。

用法

```
my_ns.bytesLoaded
```

说明

只读属性；已加载到播放器中的数据字节数。您可以将此方法与 [NetStream.bytesTotal](#) 一起使用以评估如何关闭将要满的缓冲区；例如，要向正等待数据加载到缓冲区中的用户显示反馈。

另请参见

[NetStream.bufferLength](#)

NetStream.bytesTotal

可用性

Flash Player 7。

用法

```
my_ns.bytesLoaded
```

说明

只读属性；正加载到播放器中的文件的总大小（以字节为单位）。

另请参见

[NetStream.bytesLoaded](#), [NetStream.bufferTime](#)

NetStream.close()

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此方法。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.close()
```

参数

无。

返回

无。

说明

方法；停止播放流上的所有数据，将 `NetStream.time` 属性设置为 0，使该流可用于其它用途。此命令还删除已使用 HTTP 下载的 FLV 文件的本地副本。

示例

以下 `onDisconnect()` 函数关闭一个连接并删除已存储在本地磁盘上的 `someFile.flv` 的临时副本。

```
my_nc = new NetConnection();
my_nc.connect(null);
my_ns = new NetStream(my_nc);
my_ns.play("http://www.someDomain.com/videos/someFile.flv");

function onDisconnect() {
    my_ns.close();
}
```

另请参见

[NetStream.pause\(\)](#), [NetStream.play\(\)](#)

NetStream.currentFps

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此属性。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.currentFps
```

说明

只读属性；每秒显示的帧数。如果您正导出 FLV 文件以在多个系统上回放，可以在测试期间检查该值以帮助确定在导出该文件时应用了多大程度的压缩。

NetStream.onStatus

可用性

Flash Player 7。

注意: 当与 Flash Communication Server 一起使用时, Flash Player 6 也支持此处理函数。有关更多信息, 请参见 Flash Communication Server 文档。

用法

```
my_ns.onStatus = function(infoObject) {  
    // 此处是您的代码  
}
```

参数

infoObject 按照状态或错误信息定义的参数。有关此参数的更多信息, 请参见下面的“说明”。

返回

无。

说明

事件处理函数; 每当状态发生更改或发布针对 NetStream 对象的错误时调用。如果要对此事件处理函数做出响应, 则必须创建一个函数来处理该信息对象。

信息对象具有一个 code 属性 (该属性包含的字符串说明 onStatus 处理函数的结果) 和一个 level 属性 (该属性包含字符串 "Status" 或 "Error")。

除了此 onStatus 处理函数外, Flash 还提供称作 System.onStatus 的“超级”函数。如果为特定对象调用了 onStatus 但未分配任何函数对其进行响应, 则 Flash 将处理分配到 System.onStatus 的函数 (如果存在)。

以下事件在发生某些 NetStream 活动时通知您。

Code 属性	Level 属性	含义
NetStream.Buffer.Empty	Status	数据的接收速度不足以填充缓冲区。数据流将在缓冲区重新填充前中断, 此时将发送 NetStream.Buffer.Full 消息, 并且该流将再次开始播放。
NetStream.Buffer.Full	Status	缓冲区已满并且流将开始播放。
NetStream.Play.Start	Status	回放已开始。
NetStream.Play.Stop	Status	回放已结束。
NetStream.Play.StreamNotFound	Error	无法找到传递到 play() 方法的 FLV。

示例

以下示例将有关流的数据写入日志文件。

```
my_ns.onStatus = function(info)  
{  
    _root.log_stream += "Stream status.\n";  
    _root.log_stream += "Event:" + info.code + "\n";  
    _root.log_stream += "Type:" + info.level + "\n";  
}
```

另请参见

[System.onStatus](#)

NetStream.pause()

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此方法。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.pause( [ pauseResume ] )
```

参数

pauseResume 可选：一个布尔值，指定是暂停播放 (true) 还是恢复播放 (false)。如果您省略此参数，NetStream.pause() 将会在暂停播放和恢复播放之间切换：第一次对指定的流调用该方法时，该流暂停播放；下一次调用该方法时，该流恢复播放。

返回

无。

说明

方法；暂停或恢复流的回放。

第一次您调用此方法时（不发送参数），它将暂停播放；下一次调用此方法时，它恢复播放。您可能要将此方法附加到一个按钮上，用户可以按下该按钮来暂停或恢复回放。

示例

以下示例阐释该方法的某些用法。

```
my_ns.pause(); // 第一次发出时暂停播放
my_ns.pause(); // 恢复播放
my_ns.pause(false); // 没有影响，播放继续
my_ns.pause(); // 暂停播放
```

另请参见

[NetStream.close\(\)](#), [NetStream.play\(\)](#)

NetStream.play()

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此方法。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.play("fileName");
```

参数

fileName 用引号括起来的要播放的 FLV 文件的名称。支持 http:// 和 file:// 格式；file:// 位置始终相对于 SWF 文件的位置。

返回

无。

说明

方法；开始外部视频 (FLV) 文件的回放。若要查看视频数据，您必须调用 [Video.attachVideo\(\)](#) 方法；与视频一起进行流式处理的音频或只包含音频的 FLV 文件自动播放。

如果要控制与 FLV 文件关联的音频，您可以使用 [MovieClip.attachAudio\(\)](#) 将音频路由到影片剪辑；然后可以创建 [Sound](#) 对象来控制音频的某些属性。有关更多信息，请参见 [MovieClip.attachAudio\(\)](#)。

如果无法找到 FLV 文件，则调用 [NetStream.onStatus](#) 事件处理函数。如果您要停止当前正播放的流，请使用 [NetStream.close\(\)](#)。

您可以播放与 SWF 文件存储于同一目录中或存储于其子目录中的本地 FLV 文件；但不能导航到更高级别的目录。例如，如果 SWF 文件位于名为 /training 的目录中，并且您要播放存储于 /training/videos 目录中的视频，则应使用以下语法：

```
my_ns.play("file://videos/videoName.flv");
```

若要播放存储于 /training 目录中的视频，应使用以下语法：

```
my_ns.play("file://videoName.flv");
```

示例

以下示例说明使用 [NetStream.play\(\)](#) 命令的若干方法。

```
// 播放用户计算机上的文件
// joe_user 目录是存储 SWF 文件的目录
//   的子目录
my_ns.play("file://joe_user/flash/videos/lectureJune26.flv");

// 播放服务器上的文件
my_ns.play("http://someServer.someDomain.com/flash/video/orientation.flv");
```

另请参见

[MovieClip.attachAudio\(\)](#), [NetStream.close\(\)](#), [NetStream.pause\(\)](#),
[Video.attachVideo\(\)](#)

NetStream.seek()

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此方法。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.seek(numberOfSeconds)
```

参数

numberOfSeconds 要移入 FLV 文件的以秒为单位的时间近似值。播放头移向最接近 *numberOfSeconds* 的关键帧。

- 若要返回到该流的开始处，请为 *numberOfSeconds* 传递 0。
- 若要从此流的开始处向前搜寻，应传递您要前进的秒数。例如，若要将播放头放置于距开始处 15 秒的位置，请使用 *myStream.seek(15)*。
- 若要相对于当前位置进行搜寻，请传递 *mystream.time + n* 或 *mystream.time - n*，以相对于当前位置分别向前或向后搜寻 *n* 秒。例如，若要从当前位置后退 20 秒，请使用 *my_ns.seek(my_ns.time - 20)*。

返回

无。

说明

方法；搜寻距流的开始处的最接近于指定秒数的关键帧。该流在达到该流中的指定位置时恢复播放。

示例

以下示例说明几种使用 `NetStream.seek()` 命令的方法。

```
// 返回到流的开始处
my_ns.seek(0);

// 移到距流的开始处 30 秒钟的位置
my_ns.seek(30);

// 向后移到距当前位置 3 分钟的位置
my_ns.seek(my_ns.time - 180);
```

另请参见

[NetStream.play\(\)](#), [NetStream.time](#)

NetStream.setBufferTime()

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此方法。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.setBufferTime(numberOfSeconds)
```

参数

numberOfSeconds Flash 开始显示数据前存入缓冲区的数据的秒数。默认值是 .1（十分之一秒）。

说明

方法；指定将消息存入缓冲区多长时间后开始显示流。例如，如果您要确保流的最初 15 秒无中断播放，则将 *numberOfSeconds* 设置为 15；Flash 只在将 15 秒的数据存入缓冲区后才开始播放该流。

另请参见

[NetStream.bufferTime](#)

NetStream.time

可用性

Flash Player 7。

注意：当与 Flash Communication Server 一起使用时，Flash Player 6 也支持此属性。有关更多信息，请参见 Flash Communication Server 文档。

用法

```
my_ns.time
```

说明

只读属性；以秒为单位的播放头的位置。

另请参见

[NetStream.bufferLength](#), [NetStream.bytesLoaded](#)

new

可用性

Flash Player 5。

用法

```
new constructor()
```

参数

constructor 一个函数，其后括号中可以是任何可选参数。此函数通常是要构造的对象类型的名称（如 `Array`、`Number` 或 `Object`）。

返回

无。

说明

运算符 `new` 创建一个初始时匿名的新对象，然后调用由 *constructor* 参数标识的函数。`new` 运算符将括号中的任何可选参数和用关键字 `this` 引用的新建对象传递给函数。然后构造函数可以用 `this` 来设置对象的变量。

示例

下面的示例创建 `Book()` 函数，然后使用 `new` 运算符来创建对象 `book1` 和 `book2`。

```
function Book(name, price){
    this.name = name;
    this.price = price;
}

book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```

示例

下面的示例使用 `new` 运算符来创建具有 18 个元素的 `Array` 对象：

```
golfCourse_array = new Array(18);
```

另请参见

[\[\]](#)（数组访问），[{}](#)（对象初始值设定项）

newline

可用性

Flash Player 4。

用法

`newline`

参数

无。

返回

无。

说明

常量；插入一个回车符 (\n)，该回车符在代码生成的文本输出中插入一个空行。`newline` 可用来为代码中的函数或动作所获取的信息留出空间。

示例

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

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

nextFrame()

可用性

Flash 2。

用法

`nextFrame()`

参数

无。

返回

无。

说明

函数；将播放头转到下一帧并停止。

示例

在这个示例中，当用户单击此按钮时，播放头转到下一帧并停下来。

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

nextScene()

可用性

Flash 2。

用法

`nextScene()`

参数

无。

返回

无。

说明

函数；将播放头移到下一场景的第 1 帧并停止。

示例

在这个示例中，当用户释放此按钮时，播放头被移到下一场景的第 1 帧。

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

另请参见

[prevScene\(\)](#)

not

可用性

Flash Player 4。不鼓励使用此运算符，推荐使用 `!`（逻辑 NOT）运算符。

用法

`not expression`

参数

expression 转换为布尔值的变量或其它表达式。

说明

运算符；在 Flash Player 4 中执行逻辑 NOT 运算。

另请参见

`!`（逻辑 NOT）

null

可用性

Flash Player 5。

用法

`null`

参数

无。

返回

无。

说明

常量；一个可以赋予变量或者可以由函数返回（在函数不提供数据时）的特殊值。可以使用 `null` 表示缺少的或者未定义数据类型的值。

示例

在用于数值的情况下，`null` 计算结果为 0。可对 `null` 执行相等测试。在这个语句中，二元树节点没有左分支，因此左分支的字段可以设置为 `null`。

```
if (tree.left == null) {  
    tree.left = new TreeNode();  
}
```

Number 类

可用性

Flash Player 5（已成为 Flash Player 6 本身的对象，Flash Player 6 大大提高了性能）。

说明

`Number` 类是 `Number` 数据类型的简单包装对象。使用与 `Number` 类关联的方法和属性可以操作基本数值。此类与 JavaScript `Number` 类完全相同。

调用 `Number` 对象的方法时必须使用构造函数，但是当调用 `Number` 对象的属性时则不需要使用构造函数。下面的示例指示调用 `Number` 对象的方法和属性的语法。

下面的示例调用 `Number` 对象的 `toString()` 方法，此调用返回字符串“1234”。

```
myNumber = new Number (1234);  
myNumber.toString();
```

此示例调用 `Number` 对象的 `MIN_VALUE` 属性（也称作常量）：

```
smallest = Number.MIN_VALUE
```

Number 类的方法摘要

方法	说明
<code>Number.toString()</code>	返回 <code>Number</code> 对象的字符串表示形式。
<code>Number.valueOf()</code>	返回 <code>Number</code> 对象的原始值。

Number 类的属性摘要

属性	说明
<code>Number.MAX_VALUE</code>	表示最大可表示数（双精度 IEEE-754 标准）的常量。此数字大约为 1.79E+308。
<code>Number.MIN_VALUE</code>	表示最小可表示数（双精度 IEEE-754 标准）的常量。此数字大约为 5e-324。
<code>Number.NaN</code>	表示“非数字”（NaN）值的常量。
<code>Number.NEGATIVE_INFINITY</code>	表示负无穷大值的常量。
<code>Number.POSITIVE_INFINITY</code>	表示正无穷大值的常量。此值等同于全局变量 <code>Infinity</code> 。

Number 类的构造函数

可用性

Flash Player 5。

用法

```
new Number(value)
```

参数

value 要创建的 Number 对象的数值，或者要转换为数字的值。

返回

无。

说明

构造函数；新建一个 Number 对象。当使用 `Number.toString()` 和 `Number.valueOf()` 时必须使用 Number 构造函数。而使用 Number 对象的属性时不需要使用构造函数。`new Number` 构造函数主要用作占位符。Number 对象与 `Number()` 函数不同，后者将参数转换为原始值。

示例

下面的代码构造新的 Number 对象。

```
n1 = new Number(3.4);
n2 = new Number(-10);
```

另请参见

[Number\(\)](#)

Number.MAX_VALUE

可用性

Flash Player 5。

用法

`Number.MAX_VALUE`

说明

属性；最大可表示的数（双精度 IEEE-754 标准）。此数字大约为 1.79E+308。

Number.MIN_VALUE

可用性

Flash Player 5。

用法

`Number.MIN_VALUE`

说明

属性；最小可表示的数（双精度 IEEE-754 标准）。此数字大约为 5e-324。

Number.NaN

可用性

Flash Player 5。

用法

`Number.NaN`

说明

属性；表示“非数字”（NaN）的 IEEE-754 标准值。

另请参见

[isNaN\(\)](#)、[NaN](#)

Number.NEGATIVE_INFINITY

可用性

Flash Player 5。

用法

`Number.NEGATIVE_INFINITY`

说明

属性；指定表示负无穷大的 IEEE-754 标准值。此属性的值与常数 `-Infinity` 的值相同。

负无穷大是当数学运算或函数返回的值超过可表示的负值时，返回的一个特殊数值。

Number.POSITIVE_INFINITY

可用性

Flash Player 5。

用法

`Number.POSITIVE_INFINITY`

说明

属性；指定表示正无穷大的 IEEE-754 标准值。此属性的值与常数 [Infinity](#) 的值相同。

正无穷大是当数学运算或函数返回的值大于可表示的值时，返回的一个特殊数值。

Number.toString()

可用性

Flash Player 5 ；行为在 Flash Player 7 中发生了变化。

用法

`myNumber.toString(radix)`

参数

radix 指定要用于数字到字符串转换的基数（从 2 到 36）。如果未指定 *radix* 参数，则默认值为 10。

返回

字符串。

说明

方法；返回指定 Number 对象 (*myNumber*) 的字符串表示形式。

如果 *myNumber* 未定义，则返回值如下：

- 在以 Flash Player 6 或更低版本为目标播放器发布的文件中，结果是 0。
- 在以 Flash Player 7 或更高版本为目标播放器发布的文件中，结果是 NaN。

示例

下面的示例使用 2 和 8 用于 *radix* 参数并返回包含数字 9 的相应表示形式的字符串。

```
myNumber = new Number (9);  
trace(myNumber.toString(2)); // 1001  
trace(myNumber.toString(8)); // 11
```

另请参见

[NaN](#)

Number.valueOf()

可用性

Flash Player 5。

用法

```
myNumber.valueOf()
```

参数

无。

返回

一个数字。

说明

方法；返回指定 Number 对象的原始值类型。

Number()

可用性

Flash Player 4 ；行为在 Flash Player 7 中发生了变化。

用法

```
Number(expression)
```

参数

expression 要转换为数字的表达式。

返回

一个数字或 NaN。

说明

函数；将参数 *expression* 转换为数字并按如下规则返回一个值：

- 如果 *expression* 为数字，则返回值为 *expression*。
- 如果 *expression* 为布尔值，当 *expression* 为 `true` 时，返回值为 1 ；当 *expression* 为 `false` 时，返回值为 0。
- 如果 *expression* 为字符串，则该函数尝试将 *expression* 解析为一个带有可选尾随指数的十进制数字，例如 1.57505e-3。
- 如果 *expression* 为 `undefined`，则返回值如下：
 - 在以 Flash Player 6 或更低版本为目标播放器发布的文件中，结果是 0。
 - 在以 Flash Player 7 或更高版本为目标播放器发布的文件中，结果是 NaN。

此函数用于转换导入 Flash 5 或更高版本创作环境的、包含不鼓励使用的运算符的 Flash 4 文件。有关更多信息，请参见 [&（按位 AND 运算符）](#)。

另请参见

NaN, [Number 类](#)

Object 类

可用性

Flash Player 5（已成为 Flash Player 6 本身的对象，Flash Player 6 大大提高了性能）。

说明

Object 类位于动作脚本类层次结构的根处。此类包含 JavaScript Object 类所提供功能的一小部分。

Object 类的方法摘要

方法	说明
<code>Object.addProperty()</code>	在对象上创建 getter/setter 属性。
<code>Object.registerClass()</code>	将影片剪辑元件与动作脚本对象类相关联。
<code>Object.toString()</code>	将指定对象转换为字符串然后返回它。
<code>Object.unwatch()</code>	删除 <code>Object.watch()</code> 创建的监视点。
<code>Object.valueOf()</code>	返回对象的原始值。
<code>Object.watch()</code>	注册当动作脚本对象的指定属性更改时要调用的事件处理函数。

Object 类的属性摘要

属性	说明
<code>Object.__proto__</code>	指向对象构造函数的 <code>prototype</code> 属性的引用。

Object 类的构造函数

可用性

Flash Player 5。

用法

```
new Object([value])
```

参数

value 要转换为对象的数字、布尔值或字符串。此参数是可选的。如果未指定 *value*，则该构造函数创建一个未定义属性的新对象。

返回

无。

说明

构造函数；新建一个 Object 对象。

Object.addProperty()

可用性

Flash Player 6。在外部类文件中，您可以使用 `get` 或 `set` 来代替此方法。

用法

`myObject.addProperty(prop, getFunc, setFunc)`

参数

- `prop` 要创建的对象属性的名称。
- `getFunc` 被调用以获取属性值的函数；此参数是一个函数对象。
- `setFunc` 被调用以设置属性值的函数；此参数是一个函数对象。如果向此参数传递 `null` 值，则该属性为只读。

返回

如果属性创建成功，则返回值为 `true`；否则，返回 `false`。

说明

方法；创建一个 `getter/setter` 属性。当 Flash 读取 `getter/setter` 属性时，它调用 `get` 函数，而该函数的返回值成为 `prop` 的值。当 Flash 写入 `getter/setter` 属性时，它调用 `set` 函数，并且将新值作为参数传递给它。如果具有给定名称的属性已经存在，新属性将覆盖它。

“`get`”函数没有参数。它的返回值可以为任何类型。它的类型可以在两次调用之间改变。返回值视作该属性的当前值。

“`set`”函数只有一个参数，即该属性的新值。例如，如果属性 `x` 由语句 `x = 1` 进行赋值，则将数字类型的参数 `1` 传递给 `set` 函数。忽略 `set` 函数的返回值。

可以向原型对象添加 `getter/setter` 属性。如果向一个原型对象添加 `getter/setter` 属性，则继承此原型对象的所有对象实例都将继承 `getter/setter` 属性。这样就能够在一个位置（即原型对象）添加 `getter/setter` 属性，然后使它传播到类的所有实例（与向原型对象添加方法非常相似）。如果为继承的原型对象中的 `getter/setter` 属性调用 `get/set` 函数，则传递给该 `get/set` 函数的引用将是最初引用的对象，而不是该原型对象。

如果调用不正确，`Object.addProperty()` 可能出错并失败。下表描述了可能发生的错误：

出错条件	后果
<code>prop</code> 不是有效的属性名；例如，一个空字符串。	返回 <code>false</code> ，而且不添加该属性。
<code>getFunc</code> 不是有效的函数对象。	返回 <code>false</code> ，而且不添加该属性。
<code>setFunc</code> 不是有效的函数对象。	返回 <code>false</code> ，而且不添加该属性。

示例

用法 1：某个对象具有两个内部方法：`setQuantity()` 和 `getQuantity()`。当设置或获取了属性 `bookcount` 后，该属性可用于调用这些方法。第三个内部方法 `getTitle()` 返回一个与属性 `bookname` 关联的只读值：

```
function Book() {
    this.setQuantity = function(numBooks) {
```

```

        this.books = numBooks;
    }

    this.getQuantity = function() {
        return this.books;
    }

    this.getTitle = function() {
        return "Catcher in the Rye";
    }

    this.addProperty("bookcount", this.getQuantity, this.setQuantity);
    this.addProperty("bookname", this.getTitle, null);
}

myBook = new Book();
myBook.bookcount = 5;
order = "You ordered "+myBook.bookcount+" copies of "+myBook.bookname;

```

当脚本获取 `myBook.bookcount` 的值时，动作脚本解释程序将自动调用 `myBook.getQuantity()`。当脚本修改 `myBook.bookcount` 的值时，该解释程序调用 `myObject.setQuantity()`。bookname 属性未指定 `set` 函数，因此忽略修改 `bookname` 的尝试。

用法 2：上述示例中的 `bookcount` 和 `bookname` 正常工作，但将属性 `bookcount` 和 `bookname` 添加到 `Book` 对象的每个实例中。这意味着要拥有这些属性，该对象的每个实例都必须具有两个属性位置。如果类中有许多像 `bookcount` 和 `bookname` 这样的属性，则它们可能占用大量的内存。为避免这种情况，可以向 `Book.prototype` 添加属性：

```

function Book () {}
Book.prototype.setQuantity = function(numBooks) {
    this.books = numBooks;
}
Book.prototype.getQuantity = function() {
    return this.books;
}
Book.prototype.getTitle = function() {
    return "Catcher in the Rye";
}
Book.prototype.addProperty("bookcount", Book.prototype.getQuantity,
    Book.prototype.setQuantity);
Book.prototype.addProperty("bookname", Book.prototype.getTitle, null);
myBook = new Book();
myBook.bookcount = 5;
order = "You ordered "+myBook.bookcount+" copies of "+myBook.bookname;

```

现在，`bookcount` 和 `bookname` 属性只存在于一个地方：`Book.prototype` 对象。但是效果与用法 1 中的代码相同，用法 1 将 `bookcount` 和 `bookname` 直接添加到每个实例。如果访问 `Book` 实例中的 `bookcount` 或 `bookname`，则上溯原型链并找到 `Book.prototype` 中的 `getter/setter` 属性。

用法 3：内置属性 `TextField.scroll` 和 `TextField.maxscroll` 为 `getter/setter` 属性。`TextField` 对象具有内部方法 `getScroll()`、`setScroll()` 和 `getMaxScroll()`。`TextField` 构造函数创建 `getter/setter` 属性，并将它们指向这些内部 `get/set` 方法，如下所示：

```

this.addProperty("scroll", this.getScroll, this.setScroll);
this.addProperty("maxscroll", this.getMaxScroll, null);

```

当脚本获取 `myTextField.scroll` 的值时，动作脚本解释程序自动调用 `myTextField.getScroll()`。当脚本修改 `myTextField.scroll` 的值时，解释程序调用 `myTextField.setScroll()`。`maxscroll` 属性未指定 `set` 函数，因此忽略修改 `maxscroll` 的尝试。

用法 4：虽然内置属性 `TextField.scroll` 和 `TextField.maxscroll` 在用法 3 示例中可以发挥作用，但会将属性 `scroll` 和 `maxscroll` 添加到 `TextField` 对象的每个实例中。这意味着拥有这些属性，则该对象的每个实例都必须具有两个属性位置。如果类中有许多像 `scroll` 和 `maxscroll` 这样的属性，则它们可能占用大量的内存。为避免这种情况，可以向 `TextField.prototype` 添加 `scroll` 和 `maxscroll` 属性：

```
TextField.prototype.addProperty("scroll", this.getScroll, this.setScroll);
TextField.prototype.addProperty("maxscroll", this.getMaxScroll, null);
```

现在，`scroll` 和 `maxscroll` 属性只存在于一个地方：`TextField.prototype` 对象。但是效果与上面的代码相同，上面的代码将 `scroll` 和 `maxscroll` 直接添加到每个实例。如果访问 `TextField` 实例中的 `scroll` 或 `maxscroll`，则上溯原型链并找到 `TextField.prototype` 中的 getter/setter 属性。

Object.__proto__

可用性

Flash Player 5。

用法

```
myObject.__proto__
```

参数

无。

说明

属性；引用创建 *myObject* 的构造函数的 `prototype` 属性。在创建所有对象时，将自动赋予这些对象 `__proto__` 属性。动作脚本解释程序使用 `__proto__` 属性访问对象构造函数的 `prototype` 属性，以查找该对象从其类中继承的属性和方法。

Object.registerClass()

可用性

Flash Player 6。如果您使用的是外部类文件，则可以使用“链接属性”或“元件属性”对话框中的“动作脚本 2.0 类”字段将一个对象与一个类关联，而不是使用此方法。

用法

```
Object.registerClass(symbolID, theClass)
```

参数

symbolID 影片剪辑元件的链接标识符，或动作脚本类的字符串标识符。

theClass 指向动作脚本类的构造函数的引用，如果为 `null`，则取消注册元件。

返回

如果类注册成功，则返回值为 `true`；否则，返回 `false`。

说明

方法；将影片剪辑元件与动作脚本对象类相关联。如果元件不存在，则 Flash 在字符串标识符和对象类之间创建关联关系。

时间轴放置指定影片剪辑元件的实例时，该实例注册到由 *theClass* 参数指定的类，而不是注册到 `MovieClip` 类。

使用 `MovieClip.attachMovie()` 或 `MovieClip.duplicateMovieClip()` 创建指定影片剪辑元件的实例时，该实例注册到由 *theClass* 指定的类，而不是注册到 `MovieClip` 类。如果 *theClass* 为 `null`，则此方法删除任何与指定影片剪辑元件或类标识符相关联的动作脚本类定义。对于影片剪辑元件，该影片剪辑的任何现有实例保持不变，但此元件的新实例将与默认类 `MovieClip` 相关联。

如果元件已注册到某个类，则此方法用新注册将其替换。

如果影片剪辑实例是由时间轴放置或使用 `attachMovie()` 或 `duplicateMovieClip()` 创建的，则动作脚本使用指向此对象的关键字 `this` 调用适当类的构造函数。此构造函数在调用时不带有参数。

如果使用此方法将影片剪辑注册到动作脚本类而不是 `MovieClip`，则该影片剪辑元件不继承内置 `MovieClip` 类的方法、属性和事件，除非将 `MovieClip` 类包括在新类的原型链中。下面的代码创建一个新的名为 *theClass* 的 `ActionScript` 类，它继承了 `MovieClip` 类的属性：

```
theClass.prototype = new MovieClip();
```

另请参见

`MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`

Object.toString()

可用性

Flash Player 5。

用法

```
myObject.toString()
```

参数

无。

返回

字符串。

说明

方法；将指定对象转换为字符串然后将其返回。

Object.unwatch()

可用性

Flash Player 6。

用法

```
myObject.unwatch (prop)
```

参数

prop 一个字符串，不应再进行监视的对象属性的名称。

返回

一个布尔值。

说明

方法；删除 `Object.watch()` 创建的监视点。如果监视点成功删除，则此方法返回 `true` 值；否则它返回 `false` 值。

Object.valueOf()

可用性

Flash Player 5。

用法

```
myObject.valueOf()
```

参数

无。

返回

指定对象的原始值，或对象本身。

说明

方法；返回指定对象的原始值。如果此对象没有原始值，则返回对象本身。

Object.watch()

可用性

Flash Player 6。

用法

```
myObject.watch( prop, callback [, userData] )
```

参数

prop 一个字符串，表示要监视的对象属性的名称。

callback 当监视的属性发生变化时要调用的函数。此参数为函数对象，而非字符串形式的函数名。*callback* 的格式是 *callback(prop, oldval, newval, userData)*。

userData 传递给 *callback* 方法的动作脚本数据的任意片段。如果省略 *userData* 参数，则将 *undefined* 传递给回调方法。此参数是可选的。

返回

如果监视点创建成功，则返回 *true* 值；否则，返回 *false* 值。

说明

方法；注册一个当动作脚本对象的某个指定属性更改时要调用的事件处理函数。当该属性更改时，调用该事件处理函数，并且 *myObject* 作为包含对象。您必须从 *Object.watch* 方法返回新值，否则将为监视的对象属性指定值 *undefined*。

通过返回已修改的 *newval*（或 *oldval*），监视点可以筛选（或取消）赋值。如果删除已为其设置了监视点的属性，则该监视点并不消失。如果以后重新创建此属性，则该监视点依然起作用。若要删除监视点，请使用 *Object.unwatch* 方法。

在一个属性上只能注册一个监视点。对于同一属性，对 *Object.watch()* 的后续调用会替换原来的监视点。

`Object.watch()` 方法的行为类似于 Netscape JavaScript 1.2 和更高版本中的 `Object.watch()` 函数。其主要区别在于 `userData` 参数，该参数是 Flash 对 `Object.watch()` 的增补，Netscape Navigator 不支持它。可将 `userData` 参数传递给事件处理函数，并在该事件处理函数中使用它。

`Object.watch()` 方法不能监视 `getter/setter` 属性。`Getter/setter` 属性以一种“懒惰计算”的方式进行操作，即直到实际查询属性时才确定属性的值。“懒惰计算”常常效率很高，原因在于属性并非不停地更新；而是在需要时才进行计算。但是，`Object.watch()` 需要计算属性才能激发此属性上的监视点。若要与 `getter/setter` 属性一起使用，`Object.watch()` 必须不断地计算属性，可是这样做的效率很低。

通常，动作脚本的预定义属性（如 `_x`、`_y`、`_width` 和 `_height`）为 `getter/setter` 属性，因此不能用 `Object.watch()` 进行监视。

另请参见

`Object.addProperty()`、`Object.unwatch()`

Object()

可用性

Flash Player 5。

用法

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

参数

value 一个数值、字符串或布尔值。

返回

一个对象。

说明

转换函数；创建新的空对象，或者将指定的数值、字符串或布尔值转换为一个对象。此命令等效于使用 `Object` 构造函数创建对象（请参见第 530 页的“[Object 类的构造函数](#)”）。

on()

可用性

Flash 2。在 Flash 2 中，不是所有的事件都受支持。

用法

```
on(mouseEvent) {  
    // 此处是您的语句  
}
```

参数

statement(s) 发生 *mouseEvent* 时执行的指令。

mouseEvent 是称作“事件”的触发器。当发生此事件时，执行事件后面大括号中的语句。可以为 *mouseEvent* 参数指定下面的任何值：

- `press` 在鼠标指针经过按钮时按下鼠标按钮。
- `release` 在鼠标指针经过按钮时释放鼠标按钮。
- `releaseOutside` 当鼠标指针在按钮之内时按下按钮后，将鼠标指针移到按钮之外，此时释放鼠标按钮。
- `rollOut` 鼠标指针滑出按钮区域。
- `rollOver` 鼠标指针滑过按钮。
- `dragOut` 在鼠标指针滑过按钮时按下鼠标按钮，然后滑出此按钮区域。
- `dragOver` 在鼠标指针滑过按钮时按下鼠标按钮，然后滑出此按钮，再滑回此按钮。
- `keyPress ("key")` 按下指定的键。对于此参数的 `key` 部分，需指定键控代码或键常量。有关与标准键盘上的键相关联的键控代码的列表，请参见第 745 页的附录 C “键盘键和键控代码值”；有关键常量的列表，请参见第 374 页的“Key 类的属性概要”。

说明

事件处理函数；指定触发动作的鼠标事件或按键事件。

示例

在下面的脚本中，当按下鼠标按钮时，将执行 `startDrag()` 动作，当释放鼠标按钮时，将执行条件脚本，然后放下该对象。

```
on (press) {  
    startDrag("rabbit");  
}  
on (release) {  
    trace(_root.rabbit._y);  
    trace(_root.rabbit._x);  
    stopDrag();  
}
```

另请参见

[onClipEvent\(\)](#)

onClipEvent()

可用性

Flash Player 5。

用法

```
onClipEvent(movieEvent){  
    // 此处是您的语句  
}
```

参数

movieEvent 是一个称作事件 的触发器。当事件发生时，执行该事件后面大括号中的语句。可以为 *movieEvent* 参数指定下面的任何值：

- **load** 影片剪辑一旦被实例化并出现在时间轴中时，即启动此动作。
- **unload** 在从时间轴中删除影片剪辑之后，此动作在第 1 帧中启动。在向受影响的帧附加任何动作之前，先处理与 **Unload** 影片剪辑事件关联的动作。
- **enterFrame** 以影片剪辑帧频不断触发的动作。首先处理与 **enterFrame** 剪辑事件关联的动作，然后才处理附加到受影响帧的所有帧动作。
- **mouseMove** 每次移动鼠标时启动此动作。**_xmouse** 和 **_ymouse** 属性用于确定当前鼠标位置。
- **mouseDown** 当按下鼠标左键时启动此动作。
- **mouseUp** 当释放鼠标左键时启动此动作。
- **keyDown** 当按下某个键时启动此动作。使用 **Key.getCode()** 获取有关最后按下的键的信息。
- **keyUp** 当释放某个键时启动此动作。使用 **Key.getCode()** 方法获取有关最后按下的键的信息。
- **data** 当在 **loadVariables()** 或 **loadMovie()** 动作中接收数据时启动此动作。当与 **loadVariables()** 动作一起指定时，**data** 事件只在加载最后一个变量时发生一次。当与 **loadMovie()** 动作一起指定时，获取数据的每一部分时，**data** 事件都重复发生。

说明

事件处理函数；触发为特定影片剪辑实例定义的动作。

示例

下面的语句在导出 SWF 文件时包括来自外部文件的脚本；当加载包含脚本中的动作所附加到的影片剪辑时，运行这些动作：

```
onClipEvent (load) {  
    #include "myScript.as"  
}
```

下面的示例将 **onClipEvent()** 与 **keyDown** 影片事件一起使用。**keyDown** 影片事件通常与 **Key** 对象的一个或多个方法和属性一起使用。下面的脚本使用 **Key.getCode()** 找出用户按下了哪个键；如果按下的键与 **Key.RIGHT** 属性相匹配，则将影片转到下一帧；如果按下的键与 **Key.LEFT** 属性相匹配，则将影片转到上一帧。

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

下面的示例将 `onClipEvent()` 与 `mouseMove` 影片事件一起使用。`__xmouse` 和 `__ymouse` 属性跟踪每次鼠标移动时的鼠标位置。

```
onClipEvent(mouseMove) {  
    stageX=__root.__xmouse;  
    stageY=__root.__ymouse;  
}
```

另请参见

[Key 类](#), [MovieClip.__xmouse](#), [MovieClip.__ymouse](#), [on\(\)](#), [updateAfterEvent\(\)](#)

onUpdate

可用性

Flash Player 6。

用法

```
function onUpdate () {  
    ...statements...;  
}
```

参数

无。

返回

无。

说明

事件处理函数；`onUpdate` 是为用于组件的实时预览影片定义的。当舞台上组件的实例具有实时预览影片时，每次组件实例的组件参数发生变化，创作工具都会调用实时预览影片的 `onUpdate` 函数。创作工具调用 `onUpdate` 函数时不使用参数，并且将忽略其返回值。应该在实时预览影片的主时间轴上声明 `onUpdate` 函数。

在实时预览影片中定义 `onUpdate` 函数是可选的。

有关实时预览影片的更多信息，请参见使用组件。

示例

`onUpdate` 函数使实时预览影片可以更新其视觉外观以匹配组件参数的新值。当用户更改组件属性检查器或“组件参数”面板中的参数值时，即会调用 `onUpdate`。`onUpdate` 函数将执行某些操作对其自身进行更新。例如，如果组件包括 `color` 参数，则 `onUpdate` 函数可能会更改实时预览中影片剪辑的颜色以反映新的参数值。另外，它可以将新颜色存储在内部变量中。

这里是一个使用 `onUpdate` 函数的示例，该示例通过实时预览影片中的空影片剪辑传递参数值。假设某个带有标签的按钮组件具有变量 `labelColor`，该变量指定文本标签的颜色。以下代码在组件影片主时间轴的第 1 帧中：

```
// 定义 textColor 参数变量指定按钮标签文本的颜色。  
buttonLabel.textColor = labelColor;
```

在实时预览影片中，将一个名为“xch”的空影片剪辑放置在实时预览影片中。然后将以下代码放置到实时预览影片的第 1 帧中。将“xch”添加到 `labelColor` 变量路径以通过 `my_mc` 影片剪辑传递变量：

```
// 编写一个 onUpdate 函数，将“my_mc.”添加到参数变量名中：
function onUpdate () {
    buttonLabel.textColor = my_mc.labelColor;
}
```

or

可用性

Flash 4。不鼓励使用此运算符，推荐使用 `||`（逻辑 OR）运算符。

用法

condition1 or condition2

参数

condition1、*2* 计算结果为 `true` 或 `false` 的表达式。

返回

无。

说明

运算符；计算 *condition1* 和 *condition2*，如果两个表达式中的任何一个为 `true`，则整个表达式为 `true`。

另请参见

`||`（逻辑 OR），`|`（按位 OR）

ord

可用性

Flash Player 4。不鼓励使用此函数，推荐使用 `String` 类的方法和属性。

用法

ord(character)

参数

character 要转换为 ASCII 代码数字的字符。

返回

无。

说明

字符串函数；将字符转换为 ASCII 代码数字。

另请参见

[String 类](#)

`_parent`

可用性

Flash Player 5。

用法

```
_parent.property  
_parent._parent.property
```

说明

标识符；指定或返回一个引用，该引用指向包含当前影片剪辑或对象的影片剪辑或对象。当前对象是包含引用 `_parent` 的动作脚本代码的对象。使用 `_parent` 可以指定一个相对路径，该路径指向当前影片剪辑或对象之上的影片剪辑或对象。

示例

在下面的示例中，影片剪辑 `desk` 是影片剪辑 `classroom` 的子级。当以下脚本在影片剪辑 `desk` 内执行时，播放头将跳到影片剪辑 `classroom` 时间轴中的第 10 帧。

```
_parent.gotoAndStop(10);
```

另请参见

[_root](#), [targetPath](#)

parseFloat()

可用性

Flash Player 5。

用法

```
parseFloat(string)
```

参数

string 要读取并转换为浮点数的字符串。

返回

一个数字或 NaN。

说明

函数；将字符串转换为浮点数。此函数读取（或“分析”）并返回字符串中的数字，直到它到达不是数字（其初始含义为数字）部分的字符。如果字符串不是以一个可以分析的数字开始的，则 `parseFloat` 返回 NaN。有效整数前面的空白将被忽略，有效整数后面的非数值字符也将被忽略。

示例

下面的示例使用 `parseFloat` 函数计算各种类型的数字。

```
parseFloat(" -2") 返回 -2
```

```
parseFloat(" 2.5") 返回 2.5
```

```
parseFloat("3.5e6") 返回 3.5e6 或 3500000
```

```
parseFloat("foobar") 返回 NaN
```

```
parseFloat(" 5.1") 返回 5.1
```

```
parseFloat("3.75math") 返回 3.75
```

```
parseFloat("0garbage") 返回 0
```

另请参见

[NaN](#)

parseInt

可用性

Flash Player 5。

用法

```
parseInt(expression [, radix])
```

参数

expression 转换为整数的字符串。

radix 可选；表示要分析数字的基数（基）的整数。合法值为 2 到 36。

返回

一个数字或 NaN。

说明

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

示例

下面的示例使用 parseInt 函数计算各种类型的数字。

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

```
// 返回 3
```

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

```
// 返回 NaN
```

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

```
// 返回 4
```

以下是十六进制转换的示例：

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

```
// 返回 1016
```

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

```
// 返回 1000
```

以下是一个二进制转换的示例：

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

```
// 返回 10 （二进制 1010 的十进值表示形式）
```

以下是八进制数字分析的示例：

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

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

```
// 返回 511 （八进制 777 的十进值表示形式）
```


play()

可用性

Flash 2。

用法

play()

参数

无。

返回

无。

说明

函数；在时间轴中向前移动播放头。

示例

下面的代码使用 if 语句检查用户输入的名称值。如果用户输入 Steve，则调用 play() 动作，而且播放头在时间轴中向前移动。如果用户输入 Steve 以外的任何其它内容，则不播放 SWF 文件，而显示带有变量名 alert 的文本字段。

```
stop();
if (name == "Steve") {
    play();
} else {
    alert="You are not Steve!";
}
```

prevFrame()

可用性

Flash 2。

用法

```
prevFrame()
```

参数

无。

返回

无。

说明

函数；将播放头转到前一帧并停止。如果当前帧为第 1 帧，则播放头不移动。

示例

当用户单击附加了以下处理函数的按钮时，播放头移到上一帧。

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

另请参见

[MovieClip.prevFrame\(\)](#)

prevScene()

可用性

Flash 2。

用法

```
prevScene()
```

参数

无。

返回

无。

说明

函数；将播放头移到前一场景的第 1 帧并停止。

另请参见

[nextScene\(\)](#)

print()

可用性

Flash Player 4.20。

注意：如果您要为 Flash Player 7 或更高版本进行创作，则可以创建一个 `PrintJob` 对象，它将为您（和用户）提供对打印进程的更多控制。有关更多信息，请参见 [PrintJob 类](#) 条目。

用法

```
print(target, "Bounding box")
```

参数

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

Bounding box 设置影片剪辑打印区域的修饰符。将此参数用引号引起来，然后指定以下值之一：

- *bmovie* 将影片中某一特定帧的边框指定为该影片中所有可打印帧的打印区域。为要将其边框用作打印区域的帧分配一个 *#b* 帧标签。
- *bmax* 将所有可打印帧的所有边框的合并区域指定为打印区域。当影片中可打印帧的大小各不相同，可指定 *bmax*。
- *bframe* 指定将每个可打印帧的边框用作该帧的打印区域。这将为每个帧更改打印区域，并缩放对象以符合打印区域。如果在每个帧中有不同大小的对象，而您希望每个对象都充满打印页面，请使用 *bframe*。

返回

无。

说明

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

如果对 *Bounding box* 参数使用 *bmovie*，但未向帧分配 *#b* 标签，则打印区域由加载的影片的舞台大小来确定。（加载的影片不继承主影片的舞台大小。）

必须先完全加载完影片中的所有可打印元素才能开始打印。

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

示例

下面的示例打印影片剪辑 *my_mc* 中所有可打印的帧，而打印区域则由带有 *#b* 帧标签的帧的边框来定义：

```
print(my_mc,"bmovie");
```

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

```
print(my_mc,"bframe");
```

另请参见

[printAsBitmap\(\)](#), [printAsBitmapNum\(\)](#), [PrintJob 类](#), [printNum\(\)](#)

printAsBitmap()

可用性

Flash Player 4.20。

注意：如果您要为 Flash Player 7 或更高版本进行创作，则可以创建一个 `PrintJob` 对象，它将为您（和用户）提供对打印进程的更多控制。有关更多信息，请参见 [PrintJob 类](#) 条目。

用法

```
printAsBitmap(target, "Bounding box")
```

参数

target 要打印的影片剪辑的实例名称。默认情况下，打印影片中的所有帧。如果要打印影片中的特定帧，可将 #P 帧标签附加到这些帧。

Bounding box 设置影片打印区域的修饰符。将此参数用引号引起来，然后指定以下值之一：

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

返回

无。

说明

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

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

如果对 *Bounding box* 参数使用 `bmovie`，但未向帧分配 #b 标签，则打印区域由加载的影片的舞台大小来确定。（加载的影片不继承主影片的舞台大小。）

必须先完全加载完影片中的所有可打印元素才能开始打印。

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

另请参见

[print\(\)](#), [printAsBitmapNum\(\)](#), [PrintJob 类](#), [printNum\(\)](#)

printAsBitmapNum()

可用性

Flash Player 5。

注意：如果您要为 Flash Player 7 或更高版本进行创作，则可以创建一个 `PrintJob` 对象，它将为您（和用户）提供对打印进程的更多控制。有关更多信息，请参见 [PrintJob 类](#) 条目。

用法

```
printAsBitmapNum(level, "Bounding box")
```

参数

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

Bounding box 设置影片打印区域的修饰符。将此参数用引号引起来，然后指定以下值之一：

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

返回

无。

说明

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

如果影片不包含 Alpha 透明度或色彩效果，建议您使用 `printNum()` 以获得更好的品质效果。

如果对 *Bounding box* 参数使用 `bmovie`，但未向帧分配 `#b` 标签，则打印区域由加载的影片的舞台大小来确定。（加载的影片不继承主影片的舞台大小。）

必须先完全加载完影片中的所有可打印元素才能开始打印。

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

另请参见

`print()`, `printAsBitmap()`, [PrintJob 类](#), `printNum()`

PrintJob 类

可用性

Flash Player 7。

说明

PrintJob 类用于创建内容并将其打印为一页或多页。除了对 print() 方法提供的打印功能进行改进之外，此类还允许呈现屏幕上未显示的动态内容、通过单个打印对话框提示用户，并且可以打印未经缩放的文档，使用的比例与内容的比例相同。此功能特别适用于呈现和打印外部动态内容，例如数据库内容和动态文本。

另外，使用由 PrintJob.start() 填充的属性，文档可以访问您的用户打印机设置，例如页高度、宽度和方向，并且您可以配置文档以动态设置适用于打印机设置的 Flash 内容的格式。

PrintJob 类的方法摘要

您必须以下表中列出的顺序使用 PrintJob 类的方法。

方法	说明
<code>PrintJob.start()</code>	显示操作系统的打印对话框并开始后台处理。
<code>PrintJob.addPage()</code>	将一页添加到打印后台处理程序。
<code>PrintJob.send()</code>	将经过后台处理的页发送到打印机。

PrintJob 类的构造函数

可用性

Flash Player 7。

用法

```
my_pj = new PrintJob()
```

参数

无。

返回

无。

说明

构造函数；创建一个可用于打印一页或多页的 PrintJob 对象。

若要实现打印作业，请按照所显示的顺序使用这些方法：

```
// 创建 PrintJob 对象
my_pj = new PrintJob();                                // 实例化对象

// 显示打印对话框
my_pj.start();                                          // 启动打印作业

// 将指定的区域添加到打印作业
// 为要打印的每一页都重复一次
my_pj.addPage([params]);                              // 将页发送到后台处理程序
```

```
my_pj.addPage([params]);  
my_pj.addPage([params]);  
my_pj.addPage([params]);
```

```
// 将页从后台处理程序发送到打印机  
my_pj.send();
```

```
// 打印页
```

```
// 进行清理  
delete my_pj;
```

```
// 删除对象
```

在您自己的 `PrintJob` 对象实现中，应先检查 `PrintJob.start()` 和 `PrintJob.addPage()` 的返回值，然后再继续打印。请参见 `PrintJob.addPage()` 的示例。

在任何已创建的 `PrintJob` 对象不再活动之前（即，已成功完成或失败），不能创建 `PrintJob` 对象。如果在第一个 `PrintJob` 对象仍处于活动状态时尝试（通过调用 `new PrintJob()`）创建另一个 `PrintJob` 对象，则将不会创建第二个 `PrintJob` 对象。

示例

请参见 `PrintJob.addPage()`。

另请参见

`PrintJob.addPage()`, `PrintJob.send()`, `PrintJob.start()`

PrintJob.addPage()

可用性

Flash Player 7。

用法

```
my_pj.addPage(target [, printArea] [, options] [, frameNumber])
```

参数

target 要打印的影片剪辑的级别或实例名称。传递一个数字来指定级别（例如，0 是 `_root` 影片）或传递一个字符串（括在引号中）来指定影片剪辑的实例名称。

printArea 一个指定要打印的区域的可选对象，格式如下：

```
{xMin:topLeft, xMax:topRight, yMin:bottomLeft, yMax:bottomRight}
```

您为 *printArea* 指定的坐标表示屏幕像素，这些屏幕像素相对于 `_root` 影片（如果 *target* = 0）的注册点或 *target* 指定的级别或影片剪辑的注册点。您必须提供所有四个坐标。宽度 (*xMax*-*xMin*) 和高度 (*yMax*-*yMin*) 必须都大于 0。

磅是用于打印的度量单位，像素是屏幕的度量单位；一磅的大小等于一个像素的大小。您可以使用以下换算公式将英寸或厘米转换为缇、像素或磅（1 缇为 1/20 像素）：

- 1 像素 = 1 磅 = 1/72 英寸 = 20 缇 (twip)
- 1 英寸 = 72 像素 = 72 磅 = 1440 缇
- 1 厘米 = 567 缇

注意：如果以前曾使用 `print()`、`printAsBitmap()`、`printAsBitmapNum()` 或 `printNum()` 从 Flash 进行打印，则使用了 `#b` 帧标签来指定打印区域。当使用 `addPage()` 方法时，您必须使用 *printArea* 参数来指定打印区域；`#b` 帧标签将被忽略。

如果省略了 *printArea* 参数或错误地传递了该参数，则将打印 *target* 的整个舞台区域。如果不想为 *printArea* 指定值，但想为 *options* 或 *frameNumber* 指定值，则为 *printArea* 传递 `null`。

options 一个可选参数；指定是作为矢量打印还是作为位图打印，格式如下：

```
{printAsBitmap:Boolean}
```

默认情况下，将以矢量格式打印页。若要作为位图打印 *target*，则为 `printAsBitmap` 传递 `true`。默认值是 `false`，该值表示要求进行矢量打印。当确定要使用的值时，请记住以下建议：

- 如果要打印的内容包括位图图像，则应使用 `{printAsBitmap:true}` 以包括任何透明和色彩效果。
- 如果内容不包括位图图像，则应忽略此参数或者使用 `{printAsBitmap:false}` 以较高品质的矢量格式打印内容。

如果省略或错误地传递了 *options*，则执行矢量打印。如果您不想为 *options* 指定值，但想为 *frameNumber* 指定值，则为 *options* 传递 `null`。

frameNumber 一个可选数字，用于指定要打印的帧；请注意，将不调用该帧上的任何动作脚本。如果省略此参数，则打印 *target* 中的当前帧。

注意：如果以前曾使用 `print()`、`printAsBitmap()`、`printAsBitmapNum()` 或 `printNum()` 从 Flash 进行打印，则可能在多个帧上使用了 `#p` 帧标签指定要打印哪些页。若要使用 `PrintJob.addPage()` 打印多个帧，您必须对每个帧发出 `PrintJob.addPage()` 命令；`#p` 帧标签将被忽略。有关以编程的方式实现此目的的方法，请参见本条目稍后部分中的示例。

返回

如果页被成功发送到打印后台处理程序，则为布尔值 `true`；否则为 `false`。

说明

方法；将指定的级别或影片剪辑作为单个页发送到打印后台处理程序。在使用此方法之前，您必须使用 `PrintJob.start()`；在对打印作业调用了一次或多次 `PrintJob.addPage()` 之后，您必须使用 `PrintJob.send()` 将经过后台处理的页发送到打印机。

如果此方法返回 `false`（例如，如果您未调用 `PrintJob.start()` 或者用户取消了打印作业），则对 `PrintJob.addPage()` 的任何后续调用都将失败。但是，如果此前对 `PrintJob.addPage()` 的调用成功，则结束的 `PrintJob.send()` 命令会将经成功后台处理的所有页发送到打印机。

如果为 `printArea` 传递了一个值，则 `xMin` 和 `yMin` 坐标对应于页上可打印区域的左上角（坐标 `0,0`）；可打印区域是由 `PrintJob.start()` 设置的 `pageHeight` 和 `pageWidth` 属性确定的。因为打印输出会和页上可打印区域的左上角对齐，所以如果 `printArea` 中定义的区域大于页上的可打印区域，则打印输出在右边和 / 或底部会被截断。如果没有为 `printArea` 传递值并且舞台大于可打印区域，则发生同样类型的截断操作。

如果要在打印前对影片剪辑进行缩放，可在调用此方法前设置其 `MovieClip._xscale` 和 `MovieClip._yscale` 属性，并在打印后将它们设回原始值。影片剪辑的缩放与 `printArea` 无关。也就是说，如果指定打印大小为 `50 x 50` 像素的区域，则将打印 `2500` 像素。如果对影片剪辑进行了缩放，同样会打印 `2500` 像素，但是将以缩放的大小进行打印。

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

示例

下面的示例说明了发布 `addPage()` 命令的多种方式。

```
my_btn.onRelease = function()
{
    var pageCount = 0;

    var my_pj = new PrintJob();

    if (my_pj.start())
    {
        // 以矢量格式打印 _root 影片的全部当前帧
        if (my_pj.addPage(0))
        {
            pageCount++;

            // 从 0,0 开始，以矢量格式打印 _root 影片当前帧的
            // 400 像素宽 500 像素高的区域
            if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500}))
            {
                pageCount++;

                // 从 0,0 开始，以位图格式打印 _root 影片第 1 帧的
                // 400 像素宽 500 像素高的区域
                if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500},
                    {printAsBitmap:true}, 1))
                {
                    pageCount++;

                    // 从 0,0 右侧 50 像素、下方 70 像素开始，
                    // 以矢量格式打印级别 5 的第 4 帧的
```

```

// 500 像素宽 600 像素高的区域
if (my_pj.addPage(5, {xMin:50,xMax:550,yMin:70,yMax:670},null, 4))
{
    pageCount++;

    // 从 0,0 开始, 以位图格式打印
    // “dance_mc” 影片剪辑的第 3 帧的
    // 400 像素宽 400 像素高的区域
    if (my_pj.addPage("dance_mc",
        {xMin:0,xMax:400,yMin:0,yMax:400},{printAsBitmap:true}, 3))
    {
        pageCount++;

        // 从 0,0 开始, 以矢量格式打印
        // “dance_mc” 影片剪辑的第 3 帧的 400 像素宽 600 像素高的区域,
        // 大小为实际大小的 50%
        var x = dance_mc._xscale;
        var y = dance_mc._yscale;
        dance_mc._xscale = 50;
        dance_mc._yscale = 50;

        if (my_pj.addPage("dance_mc",
            {xMin:0,xMax:400,yMin:0,yMax:600},null, 3))
        {
            pageCount++;
        }

        dance_mc._xscale = x;
        dance_mc._yscale = y;
    }
}

}
}

}

if (pageCount)
{
    my_pj.send();
}
delete my_pj;
}

```

另请参见

[PrintJob.send\(\)](#), [PrintJob.start\(\)](#)

PrintJob.send()

可用性

Flash Player 7。

用法

```
my_pj.send()
```

参数

无。

返回

无。

说明

方法；用在 `PrintJob.start()` 和 `PrintJob.addPage()` 之后，将经过后台处理的页发送到打印机。

示例

请参见 `PrintJob.addPage()`。

另请参见

`PrintJob.addPage()`, `PrintJob.start()`

PrintJob.start()

可用性

Flash Player 7。

用法

```
my_pj.start()
```

参数

无。

返回

如果出现打印对话框时用户单击“确定”，则为布尔值 true，如果用户单击“取消”或出现错误，则为 false。

说明

方法；显示操作系统的打印对话框并开始后台处理。打印对话框允许用户更改打印设置，然后填充以下只读属性（请注意，1 磅等于屏幕上的 1 个像素）：

属性	类型	单位	备注
PrintJob.paperHeight	数字	磅	纸张整体高度
PrintJob.paperWidth	数字	磅	纸张整体宽度
PrintJob.pageHeight	数字	磅	页上实际可打印区域的高度；忽略任何用户设置的边距
PrintJob.pageWidth	数字	磅	页上实际可打印区域的宽度；忽略任何用户设置的边距
PrintJob.orientation	字符串	无	“纵向”或“横向”

当用户在打印对话框中单击“确定”之后，播放器开始在后台将打印作业处理到操作系统。您应该发布影响打印输出的任何动作脚本命令，然后可以使用 `PrintJob.addPage()` 命令开始将页发送到后台处理程序。如果愿意，可以使用此方法返回的高度、宽度和方向属性确定如何设置打印输出的格式。

因为在单击“确定”后用户马上会看到诸如“正在打印第 1 页”这样的信息，所以应该尽快调用 `PrintJob.addPage()` 和 `PrintJob.send()` 命令。

如果此方法返回 false（例如，如果用户单击“取消”而不是“确定”），则对 `PrintJob.addPage()` 和 `PrintJob.send()` 的任何后续调用都将失败。但是，如果只测试此返回值，而不需要发出 `PrintJob.addPage()` 命令，则仍需要删除 `PrintJob` 对象，以确保清除该打印后台处理程序，如下所示。

```
var my_pj = new PrintJob();
var myResult = my_pj.start();
if(myResult){
    // 此处为 addPage() 和 send() 语句
}
delete my_pj;
```

示例

请参见 `PrintJob.addPage()`。

另请参见

[PrintJob.addPage\(\)](#), [PrintJob.send\(\)](#)

printNum()

可用性

Flash Player 5。

注意：如果您要为 Flash Player 7 或更高版本进行创作，则可以创建一个 [PrintJob](#) 对象，它将为您（和用户）提供对打印进程的更多控制。有关更多信息，请参见 [PrintJob](#) 类条目。

用法

```
printNum (level, "Bounding box")
```

参数

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

Bounding box 设置影片打印区域的修饰符。将此参数用引号引起来，然后指定以下值之一：

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

返回

无。

说明

函数；根据在 *Bounding box* 参数中指定的边界（"bmovie"、"bmax"、"bframe"）打印 Flash Player 中的级别。如果要打印目标影片中的特定帧，请将 #p 帧标签附加到这些帧。尽管使用 `printNum()` 实现的打印品质高于使用 `printAsBitmapNum()`，但是不能使用 `printNum()` 打印带有 Alpha 透明度或特殊色彩效果的影片。

如果对 *Bounding box* 参数使用 **bmovie**，但未向帧分配 #b 标签，则打印区域由加载的影片的舞台大小来确定。（加载的影片不继承主影片的舞台大小。）

必须先完全加载完影片中的所有可打印元素才能开始打印。

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

另请参见

[print\(\)](#), [printAsBitmap\(\)](#), [printAsBitmapNum\(\)](#), [PrintJob](#) 类

private

可用性

Flash Player 6。

用法

```
class someClassName{  
    private var name;  
    private function name() {  
        // 此处是您的语句  
    }  
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

name 要指定为 private 的变量或函数的名称。

说明

关键字；指定变量或函数只对声明或定义该变量或函数的类或该类的子类可用。默认情况下，变量或函数对调用它的任何类都可用。使用此关键字可以限制对变量或函数的访问。有关更多信息，请参见第 144 页的“控制成员访问”。

您只能在类定义中使用此关键字，不能在接口定义中使用。

另请参见

`public`, `static`

public

Flash Player 6。

用法

```
class someClassName{
    public var name;
    public function name() {
        // 此处是您的语句
    }
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

name 要指定为 public 的变量或函数的名称。

说明

关键字；指定变量或函数对调用它的任何类都可用。因为默认情况下变量和函数是公共的，所以使用此关键字主要是出于格式上的原因。例如，在包含 private 或 static 变量的代码块中，您可能要使用此关键字来保持格式一致。

示例

下面两个代码块在功能上是相同的。

```
private var age:Number;
public var name:String;
static var birth>Date;
```

```
private var age:Number;
var name:String;
static var birth>Date;
```

有关更多信息，请参见[第 144 页的“控制成员访问”](#)。

另请参见

[private](#), [static](#)

`_quality`

可用性

Flash Player 5。

用法

`_quality`

说明

属性（全局）；设置或获取用于影片的呈现品质。设备字体始终是带有锯齿的，因此不受 `_quality` 属性的影响。

`_quality` 属性可设置为下列值：

- "LOW" 低呈现品质。不消除图形的锯齿，位图不进行平滑处理。
- "MEDIUM" 中等呈现品质。使用 2 x 2 网格（以像素为单位）消除图形锯齿，但不对位图进行平滑处理。适用于不包含文本的影片。
- "HIGH" 高呈现品质。使用 4 x 4 网格（以像素为单位）消除图形锯齿，如果影片是静态的，则对位图进行平滑处理。这是 Flash 使用的默认呈现品质设置。
- "BEST" 极高的呈现品质。使用 4 x 4 网格（以像素为单位）消除图形锯齿，并且始终对位图进行平滑处理。

示例

下面的示例将呈现品质设置为 LOW：

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

另请参见

[_highquality, toggleHighQuality\(\)](#)

random

可用性

Flash Player 4。不鼓励在 Flash 5 中使用该函数，而推荐使用 [Math.random\(\)](#)。

用法

```
random(value)
```

参数

value 一个整数。

返回

一个整数。

说明

函数；返回一个随机整数，此整数介于 0 和 *value* 参数中指定的整数减 1 之间。

示例

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

```
random(5);
```

另请参见

[Math.random\(\)](#)

removeMovieClip()

可用性

Flash Player 4。

用法

```
removeMovieClip(target)
```

参数

target 用 [duplicateMovieClip\(\)](#) 创建的影片剪辑实例的目标路径，或者是用 [MovieClip.attachMovie\(\)](#) 或 [MovieClip.duplicateMovieClip\(\)](#) 创建的影片剪辑的实例名称。

返回

无。

说明

函数；删除指定的影片剪辑。

另请参见

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

return

可用性

Flash Player 5。

用法

```
return[expression]
```

参数

expression 要作为函数值计算并返回的字符串、数字、数组或对象。此参数是可选的。

返回

如果提供了 *expression* 参数，则返回计算的结果。

说明

语句；指定由函数返回的值。return 动作计算 *expression* 并将结果作为它在其中执行的函数的值返回。return 动作导致函数停止运行，并用返回值代替函数。如果单独使用 return 语句，则它返回 null。

您不能返回多个值。如果尝试返回多个值，则将只返回最后一个值。在下面的示例中，返回 c：

```
return a, b, c ;
```

示例

下面的示例在 sum() 函数体内使用 return 动作，以返回三个参数相加后的值。下一行代码调用 sum() 并将返回值赋予变量 newValue：

```
function sum(a, b, c){  
    return a + b + c;  
}  
newValue = sum(4, 32, 78);  
trace(newValue);  
// 将 114 发送到“输出”面板
```

另请参见

[function](#)

_root

可用性

Flash Player 5。

用法

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

参数

movieClip 影片剪辑的实例名称。

action 动作或方法。

property MovieClip 对象的属性。

说明

属性；指定或返回指向根影片时间轴的引用。如果影片有多个级别，则根影片时间轴位于包含当前正在执行脚本的级别上。例如，如果级别 1 中的脚本计算 `_root`，则返回 `_level1`。

指定 `_root` 与在当前级别内用斜杠记号 (/) 指定绝对路径的效果相同。

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

示例

下面的示例停止包含当前正在执行脚本的级别的时间轴：

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

下面的示例将当前级别中的时间轴移至第 3 帧：

```
_root.gotoAndStop(3);
```

另请参见

[MovieClip._lockroot](#), [_parent](#), [targetPath](#)

scroll

可用性

Flash Player 4。

用法

```
textFieldVariableName.scroll = x
```

说明

属性；一种不鼓励使用的属性，控制如何显示与某变量关联的文本字段中的信息。scroll 属性定义文本字段开始显示内容的位置；设置此属性后，当用户滚动该文本字段时，Flash Player 将更新此属性。scroll 属性很有用，该属性可以帮助用户定位到长篇文章的特定段落，或者创建滚动文本字段。可以获取和修改此属性。

示例

下面是附加到“向上”按钮上的代码，该按钮用于滚动文本字段 myText：

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

另请参见

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

Selection 类

可用性

Flash Player 5。

说明

利用 Selection 类可以设置和控制插入点所在的文本字段，即，具有焦点的字段。选择范围 (Selection-span) 索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

Selection 类没有构造函数，这是因为在同一时间只能有一个当前具有焦点的字段。

Selection 类的方法摘要

方法	说明
Selection.addListener()	注册一个对象，以便在调用 onSetFocus 时接收通知。
Selection.getBeginIndex()	返回选择范围的开始索引。如果没有索引或当前没有选中的字段，则返回 -1。
Selection.getCaretIndex()	返回当前具有焦点的选择范围中当前插入符号（插入点）的位置。如果没有插入符号位置，或者当前没有具有焦点的选择范围，则返回 -1。
Selection.getEndIndex()	返回选择范围的结束索引。如果没有索引或当前没有选中的字段，则返回 -1。
Selection.getFocus()	返回当前具有焦点的文本字段的变量的名称。如果当前没有具有焦点的文本字段，则返回 null。
Selection.removeListener()	删除用 addListener() 注册的对象。

方法	说明
<code>Selection.setFocus()</code>	使与指定变量关联的文本字段获得焦点。
<code>Selection.setSelection()</code>	设置选择范围的开始索引和结束索引。

Selection 类的侦听器摘要

侦听器	说明
<code>Selection.onSetFocus</code>	当输入焦点更改时获得通知。

Selection.addListener()

可用性

Flash Player 6。

用法

```
Selection.addListener(newListener)
```

参数

newListener 具有 `onSetFocus` 方法的对象。

返回

无。

说明

方法；注册一个对象，以接收键盘焦点更改通知。当焦点更改时（例如，每当调用 `Selection.setFocus()` 时），所有用 `addListener()` 注册的侦听对象都调用它们的 `onSetFocus` 方法。可以有多个对象侦听焦点更改通知。如果已经注册了侦听器 *newListener*，则不会发生任何更改。

Selection.getBeginIndex()

可用性

Flash Player 5。

用法

```
Selection.getBeginIndex()
```

参数

无。

返回

一个整数。

说明

方法；返回选择范围开始处的索引。如果不存在索引，或者当前没有具有焦点的文本字段，则该方法返回 -1。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

Selection.getCaretIndex()

可用性

Flash Player 5。

用法

```
Selection.getCaretIndex()
```

参数

无。

返回

一个整数。

说明

方法；返回闪烁插入点（插入符号）位置的索引。如果没有显示闪烁插入点，则该方法返回 -1。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

Selection.getEndIndex()

可用性

Flash Player 5。

用法

```
Selection.getEndIndex()
```

参数

无。

返回

一个整数。

说明

方法；返回当前具有焦点的选择范围的结束索引。如果不存在索引，或者当前没有具有焦点的选择范围，则该方法返回 -1。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。

Selection.setFocus()

可用性

Flash Player 5。按钮和文本字段的实例名称在 Flash Player 6 和更高版本中可正常工作。

用法

```
Selection.setFocus()
```

参数

无。

返回

字符串或 `null`。

说明

方法；返回具有焦点的文本字段的变量名。如果没有任何文本字段具有焦点，则该方法返回 `null`。如果当前具有焦点的是一个按钮，并且该按钮是 `Button` 对象，则 `getFocus()` 以字符串形式返回目标路径。如果当前具有焦点的是一个文本字段，并且该文本字段是 `TextField` 对象，则 `getFocus()` 以字符串形式返回目标路径。

如果按钮影片剪辑是当前具有焦点的按钮，则 `Selection.setFocus()` 返回该按钮影片剪辑的目标路径。如果当前具有焦点的是具有实例名称的文本字段，则 `Selection.setFocus()` 返回该 `TextField` 对象的目标路径。否则，它返回该文本字段的变量名。

Selection.onSetFocus

可用性

Flash Player 6。

用法

```
someListener.onSetFocus = function(oldFocus, newFocus){  
    statements;  
}
```

说明

侦听器；当输入焦点更改时获得通知。若要使用 `onSetFocus`，必须创建一个侦听器对象。然后可以为 `onSetFocus` 定义一个函数，并使用 `addListener()` 将该侦听器注册到 `Selection` 对象，如下所示：

```
somelister = new Object();  
somelister.onSetFocus = function () { ... };  
Selection.addListener(somelister);
```

侦听器可以使不同的代码片段协同工作，因为多个侦听器可以接收有关单个事件的通知。

另请参见

[Selection.addListener\(\)](#)

Selection.removeListener()

可用性

Flash Player 6。

用法

```
Selection.removeListener(listener)
```

参数

listener 不再接收焦点通知的对象。

返回

如果成功删除了 *listener*，则该方法返回值 `true`。如果未成功删除 *listener*（例如，如果 *listener* 不在 `Selection` 对象的侦听器列表中），则该方法返回值 `false`。

说明

方法；删除以前用 `addListener()` 注册的对象。

Selection.setFocus()

可用性

Flash Player 5。按钮和影片剪辑的实例名称只在 Flash Player 6 和更高版本中可正常工作。

用法

```
Selection.setFocus(instanceName)
```

参数

instanceName 一个字符串，指定按钮、影片剪辑或文本字段的实例名的路径。

返回

事件。

说明

方法；使 *instanceName* 指定的可选择（可编辑）文本字段、按钮或影片剪辑具有焦点。
instanceName 参数必须是表示实例路径的文本字符串。可以使用点或斜杠记号指定路径。也可以使用相对路径或绝对路径。如果使用动作脚本 2.0，则必须使用点记号表示法。

如果传递 `null`，则清除当前焦点。

示例

下面的示例使与主时间轴上的 `myVar` 关联的文本字段具有焦点。因为 *instanceName* 参数是绝对路径，所以您可以从任何时间轴调用该动作。

```
Selection.setFocus("_root.myVar");
```

在下面的示例中，与 `myVar` 关联的文本字段位于主时间轴上名为 `myClip` 的影片剪辑中。您可以使用下面两种路径之一来设置焦点；第一条语句使用的是相对路径，第二条语句使用的是绝对路径。

```
Selection.setFocus("myClip.myVar");  
Selection.setFocus("_root.myClip.myVar");
```


Selection.setSelection()

可用性

Flash Player 5。

用法

```
Selection.setSelection(start, end)
```

参数

start 选择范围的开始索引。

end 选择范围的结束索引。

返回

无。

说明

方法；设置当前具有焦点的文本字段的选择范围。新的选择范围将从 *start* 参数中指定的索引开始，到 *end* 参数中指定的索引结束。选择范围索引是从零开始的（例如，第一个位置为 0、第二个位置为 1，依此类推）。如果当前没有具有焦点的文本字段，则此方法无效。

set

可用性

Flash Player 6。

用法

```
function set property(varName) {  
    // 此处是您的语句  
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

property 您要用来指 set 将访问的属性的单词；此值必须与对应的 get 命令中使用的值相同。

varName 设置您正在指定的值的局部变量。

返回

无。

说明

关键字；允许根据您在外部类文件中定义的类型“设置”与对象关联的属性。使用隐式设置方法允许您访问对象的属性，而不用直接访问对象。隐式获取 / 设置方法是对动作脚本 1 中 Object.addProperty() 方法的句法简化。

有关更多信息，请参见第 151 页的“隐式获取 / 设置方法”。

另请参见

[get](#), [Object.addProperty\(\)](#)

set variable

可用性

Flash Player 4。

用法

`set(variable, expression)`

参数

variable 保存 *expression* 参数值的标识符。

expression 分配给变量的值。

返回

无。

说明

语句；为变量赋值。*variable* 是保存数据的容器。容器本身始终不变，但内容可以更改。通过在 SWF 文件播放时更改变量的值，可以记录和保存有关用户所执行操作的信息、记录当 SWF 文件播放时更改的值，或者计算某条件是 `true` 还是 `false`。

变量可以保存任何类型的数据（例如，字符串、数字、布尔值、对象或影片剪辑）。每个 SWF 文件和影片剪辑的时间轴都有其自己的变量集，每个变量又都有其自己的独立于其它时间轴上的变量的值。

`set` 语句中不支持严格数据类型指定。如果使用此语句将某个变量设置为一个值，而该值的数据类型与类文件中与该变量关联的数据类型不同，则不引发编译器错误。

示例

此示例设置一个名为 `orig_x_pos` 的变量，该变量存储 `ship` 影片剪辑的原始 *x* 轴位置，以便以后在 SWF 文件中将船 (`ship`) 重置到其起始位置。

```
on (release) {  
    set("orig_x_pos", getProperty ("ship", _x ));  
}
```

上面的代码与下面的代码结果一样：

```
on (release) {  
    orig_x_pos = ship._x;  
}
```

另请参见

[var, call\(\)](#)

setInterval()

可用性

Flash Player 6。

用法

```
setInterval(functionName, interval [, param1, param2, ..., paramN])
```

参数

functionName 一个函数名或者一个对匿名函数的引用。

interval 对 *functionName* 参数的两次调用之间的时间（以毫秒为单位）。

param1, param2, ..., paramN 传递到 *function* 或 *methodName* 参数的可选参数。

返回

间隔标识符，可以将该标识符传递给 `clearInterval()` 以取消该间隔。

说明

函数；在播放 SWF 文件时，每隔一定的时间，就调用函数、方法或对象。可以使用间隔函数更新来自数据库的变量或更新时间显示。

如果 *interval* 小于 SWF 文件的帧频（例如，每秒 10 帧 [fps] 相当于 100 毫秒），则按照尽可能接近 *interval* 的时间间隔调用间隔函数。而且必须使用 `updateAfterEvent()` 函数来确保以足够的频率刷新屏幕。如果 *interval* 大于 SWF 文件的帧频，则只有在每次播放头进入帧时才调用该间隔函数，这样可以尽可能减小每次刷新屏幕的影响。

示例

用法 1：下面的示例每隔 1000 毫秒（每隔 1 秒）调用一次匿名函数。

```
setInterval( function(){ trace("interval called"); }, 1000 );
```

用法 2：下面的示例定义两个事件处理函数并分别调用它们。对 `setInterval()` 的两次调用的结果都是每隔 1000 毫秒就向“输出”面板发送字符串“interval called”。对 `setInterval()` 的第一个调用将调用 `callback1()` 函数，该函数包含 `trace()` 动作。对 `setInterval()` 的第二个调用将“interval called”字符串作为参数传递给函数 `callback2()`。

```
function callback1() {  
    trace("interval called");  
}
```

```
function callback2(arg) {  
    trace(arg);  
}
```

```
setInterval( callback1, 1000 );  
setInterval( callback2, 1000, "interval called" );
```

用法 3：此示例使用对象的方法。当要调用为对象定义的方法时，必须使用此语法。

```
obj = new Object();  
obj.interval = function() {  
    trace("interval function called");  
}
```

```
setInterval( obj, "interval", 1000 );
```

```
obj2 = new Object();
obj2.interval = function(s) {
    trace(s);
}
setInterval( obj2, "interval", 1000, "interval function called" );
```

必须使用第二种格式的 `setInterval()` 语法来调用对象的方法，如下所示：

```
setInterval( obj2, "interval", 1000, "interval function called" );
```

另请参见

[clearInterval\(\)](#), [updateAfterEvent\(\)](#)

setProperty()

可用性

Flash Player 4。

用法

```
setProperty(target, property, value/expression)
```

参数

target 到要设置其属性的影片剪辑实例名称的路径。

property 要设置的属性。

value 属性的新文本值。

expression 计算结果为属性新值的公式。

返回

无。

说明

函数；当影片播放时，更改影片剪辑的属性值。

示例

当单击按钮时，此语句将名为 `star` 的影片剪辑的 `_alpha` 属性设置为 30%：

```
on (release) {
    setProperty("star", _alpha, "30");
}
```

另请参见

[getProperty](#)

SharedObject 类

可用性

Flash Player 6。

说明

共享对象相当强大：它们提供用户计算机上永久对象间的实时数据共享。您可以将本地共享对象看作“Cookie”。

您可以使用本地共享对象维持本地永久性。这是使用共享对象最简单的方式。例如，您可以调用 `SharedObject.getLocal()` 在播放器中创建共享对象，例如具有内存的计算器。因为共享对象在本地是永久性的，所以当 SWF 文件结束时 Flash 会将其数据属性保存在用户的计算机上。SWF 文件下次运行时，计算器仍包含 SWF 文件结束时它所具有的值。或者，如果在 SWF 结束前将共享对象的属性设置为 `null`，则当 SWF 文件下次运行时，计算器打开时不带有任何以前的值。

若要创建本地共享对象，请使用以下语法：

```
// 创建本地共享对象  
so = SharedObject.getLocal("foo");
```

本地磁盘空间注意事项

在可用内存和磁盘空间用完之前，客户端上的本地共享对象总是永久性的。

默认情况下，Flash 可以在本地保存多达 100K 大小的永久性远程共享对象。当您尝试保存一个较大的对象时，Flash Player 将显示“本地存储”对话框，用户可以使用该对话框允许或拒绝请求访问的域的本地存储。确保舞台大小至少为 215 x 138 像素；这是 Flash 显示该对话框所需的最小大小。



如果用户单击“允许”，则将保存该对象，并用 `SharedObject.Flush.Success` 的 `code` 属性调用 `SharedObject.onStatus`；如果用户单击“拒绝”，则不保存该对象，并用 `SharedObject.Flush.Failed` 的 `code` 属性调用 `SharedObject.onStatus`。

用户还可以指定特定域的永久本地存储设置，方法是在 SWF 文件播放时右击 (Windows) 或按住 `Control` 键单击 (Macintosh)，选择“设置”，然后打开“本地存储”面板。



下表总结用户的磁盘空间选择如何与共享对象互相影响：

- 如果用户选择“从不”，则从不在本地保存对象，为该对象发出的所有 `SharedObject.flush()` 命令都返回 `false`。
- 如果用户选择“无限制”（一直向右移动滑块），则在本地保存对象直到达到最大可用磁盘空间。
- 如果用户选择“无”（一直向左移动滑块）则为该对象发出的所有 `SharedObject.flush()` 命令都返回 `"pending"`，播放器将询问用户是否可以分配额外的磁盘空间以容纳该对象，如上所述。
- 如果用户选择“10 KB”、“100 KB”、“1 MB”或“10 MB”，则将在本地保存对象，并且如果这些对象的大小未超过指定的空间量，则 `SharedObject.flush()` 返回 `true`。如果需要更多空间，`SharedObject.flush()` 将返回 `"pending"`，播放器将询问用户是否可以分配额外的磁盘空间以容纳该对象，如上所述。

另外，如果用户选择的值小于本地永久数据当前使用的磁盘空间量，则播放器将警告用户将删除所有本地保存的共享对象。



注意：在创作环境中运行的 Flash Player 没有大小限制。

SharedObject 类的方法摘要

方法	说明
<code>SharedObject.clear()</code>	清除共享对象中的所有数据并从磁盘删除共享对象。
<code>SharedObject.flush()</code>	将本地永久共享对象立即写入本地文件。
<code>SharedObject.getLocal()</code>	返回对本地永久共享对象的引用，该对象只可用于当前客户端。
<code>SharedObject.getSize()</code>	获取共享对象的当前大小（以字节为单位）。

SharedObject 类的属性摘要

属性（只读）	说明
<code>SharedObject.data</code>	分配到该对象 <code>data</code> 属性的属性集合，可以共享和 / 或存储这些属性。

SharedObject 类的事件处理函数摘要

事件处理函数	说明
<code>SharedObject.onStatus</code>	每次为共享对象公布错误、警告或信息性通知时调用。

SharedObject 类的构造函数

有关创建本地共享对象的信息，请参见 `SharedObject.getLocal()`。

SharedObject.clear()

可用性

Flash Player 7。

用法

```
my_so.clear()
```

参数

无。

返回

无。

说明

方法；清除共享对象中的所有数据并从磁盘删除共享对象。对 *my_so* 的引用仍然处于活动状态，*my_so* 现在已空。

SharedObject.data

可用性

Flash Player 6。

用法

myLocalSharedObject.data

说明

只读属性；分配到对象 `data` 属性的属性集合，可以共享和 / 或存储这些属性。每个属性都可以是所有基本 ActionScript 或 JavaScript 类型（数组、数字、布尔值等）的对象。例如，下面几行将值分配到共享对象的不同方面：

```
itemsArray = new Array(101,346,483);
currentUserIsAdmin = true;
currentUserName = "Ramona";
so.data.itemNumbers = itemsArray;
so.data.adminPrivileges = currentUserIsAdmin;
so.data.userName = currentUserName;
```

如果对象是永久性的，则将保存共享对象 `data` 属性的所有属性。

注意：不要将值直接赋予共享对象的 `data` 属性（例如 `so.data = someValue`）；Flash 将忽略这些赋值。

若要删除本地共享对象的属性，请使用诸如 `delete so.data.attributeName` 这样的代码；将本地共享对象的属性设置为 `null` 或 `undefined` 不会删除该属性。

若要创建共享对象的“私有”值（该对象正在使用时只有客户端实例才可以使用并且当对象关闭时不与该对象存储在一起的值），请创建名称不为 `data` 的属性来存储它们，如下面的示例所示。

```
so.favoriteColor = "blue";
so.favoriteNightClub = "The Bluenote Tavern";
so.favoriteSong = "My World is Blue";
```

示例

下面的示例将当前流设置为用户的选择。

```
curStream = _root.so.data.msgList[selected].streamName;
```

另请参见

[Sound 类](#)

SharedObject.flush()

可用性

Flash Player 6。

用法

```
myLocalSharedObject.flush([minimumDiskSpace])
```

参数

minimumDiskSpace 一个可选整数，指定必须为此对象分配的字节数。默认值为 0。

返回

布尔值 true 或 false，或者是字符串值 "pending"。

- 如果用户允许此域中的对象进行本地信息存储，并且所分配的空间量足够存储该对象，则此方法返回 true。（如果已为 *minimumDiskSpace* 传递了一个值，则所分配的空间量必须至少等于使此方法能够返回 true 的值）。
- 如果用户允许此域中的对象进行本地信息存储，但所分配的空间量不足以存储该对象，则此方法返回 "pending"。
- 如果用户已永久拒绝此域中的对象进行本地信息存储，或者如果 Flash 出于任何原因无法保存对象，则此方法返回 false。

说明

方法；将本地永久共享对象立即写入本地文件。如果您不使用此方法，则 Flash 会在共享对象会话结束时（也就是说，在 SWF 文件关闭时、在共享对象不再具有引用而被作为垃圾回收时、或者在您调用 [SharedObject.data](#) 时）将共享对象写入文件。

如果此方法返回 "pending"，Flash Player 将显示对话框要求用户增加磁盘空间量以供此域中的对象使用。若要允许将来保存共享对象时其空间能够“增长”，从而避免返回值 "pending"，请为 *minimumDiskSpace* 传递一个值。当 Flash 尝试写入文件时，它查找传递到 *minimumDiskSpace* 的字节数，而不是查找以共享对象的当前大小保存该对象刚刚够用的空间。

例如，如果预期共享对象增长到最大大小 500 字节，则即使它开始时要小得多，也为 *minimumDiskSpace* 传递 500。如果 Flash 要求用户为该共享对象分配磁盘空间，它将要求 500 字节。在用户分配了请求的空间量之后，当以后尝试刷新该对象时（只要其大小不超过 500 字节），Flash 将无需要求更多的空间。

在用户响应该对话框之后，此方法将被再次调用并返回 true 或 false；另外，将用 [SharedObject.Flush.Success](#) 或 [SharedObject.Flush.Failed](#) 的 code 属性调用 [SharedObject.onStatus](#)。

有关更多信息，请参见第 573 页的“本地磁盘空间注意事项”。

示例

下面的函数获取共享对象 s0，并用用户提供的设置填充可写属性。最后，调用 `flush()` 以保存设置并分配最少 1000 字节的磁盘空间。

```
this.SyncSettingsCore=function(soname, override, settings)
{
    var S0=SharedObject.getLocal(soname, "http://www.mydomain.com/app/sys");

    // 设置列表索引
    var i;
```

```

// 对于设置中每个指定的值：
// 如果 override 为 true，则将永久设置设置为所提供的值。
// 如果 override 为 false，则获取永久设置，除非
// 没有永久设置，在这种情况下，将其设置为所提供的值。
for (i in settings) {
    if (override || (SO.data[i] == null)) {
        SO.data[i] = settings[i];
    } else {
        settings[i] = SO.data[i];
    }
}
SO.flush(1000);
}

```

SharedObject.getLocal()

可用性

Flash Player 6。

用法

`SharedObject.getLocal(objectName [, localPath])`

注意：正确的语法是 `SharedObject.getLocal`。若要将该对象分配到变量，请使用类似于 `myLocalSO = SharedObject.getLocal` 的语法。

参数

objectName 对象的名称。该名称可以包含正斜杠 (/)；例如 `work/addresses` 就是合法名称。共享对象名称中不允许使用空格，也不允许使用以下字符：

~ % & \ ; : " ' , < > ? #

localPath 一个可选字符串参数；该参数指定指向创建共享对象的 SWF 文件的完整路径或部分并确定共享对象的本地存储位置。默认值是完整路径。

返回

一个对共享对象的引用，该共享对象在本地是永久性的并且只可用于当前客户端。如果 Flash 无法创建或找到共享对象（例如，如果指定了 *localPath* 但不存在此目录），则此方法返回 `null`。

说明

方法；返回对只可用于当前客户端的本地永久共享对象的引用。

为了避免名称冲突，Flash 会考虑创建共享对象的 SWF 文件的位置。例如，如果位于 `www.myCompany.com/apps/stockwatcher.swf` 的 SWF 文件创建了一个名为 `portfolio` 的共享对象，该共享对象不会与位于 `www.yourCompany.com/photoshoot.swf` 的 SWF 文件所创建的另一个名为 `portfolio` 的对象冲突，这是因为这两个 SWF 文件源于两个不同的目录。

示例

下面的示例将用户输入的最后一个帧保存到本地共享对象 `kookie`。

```

// 获取 kookie
so = sharedobject.getLocal("kookie");

// 获取 kookie 的用户并转到为此用户保存的帧编号。
if (so.data.user != undefined) {
    this.user = so.data.user;
}

```

```
        this.gotoAndStop(so.data.frame);  
    }  
}
```

下面的代码块放置在每个 SWF 帧上。

```
// 在每个帧上，调用 rememberme 函数保存帧编号。  
function rememberme() {  
    so.data.frame=this._currentFrame;  
    so.data.user="John";  
}
```

SharedObject.getSize()

可用性

Flash Player 6。

用法

```
myLocalSharedObject.getSize()
```

参数

无。

返回

一个指定共享对象大小（以字节为单位）的数值。

说明

方法；获取共享对象的当前大小（以字节为单位）。

Flash 通过逐句调试共享对象的每个数据属性计算该对象的大小；对象具有的数据属性越多，评估其大小所花的时间就越长。出于此原因，评估对象大小可能要花费巨大的处理开销。因此，除非有特定需求，否则应避免使用此方法。

示例

下面的示例获取共享对象 so 的大小。

```
var soSize= this.so.getSize();
```

SharedObject.onStatus

可用性

Flash Player 6。

用法

```
myLocalSharedObject.onStatus = function(infoObject) {  
    // 此处是您的语句  
}
```

参数

infoObject 按照状态消息定义的参数。

返回

无。

说明

事件处理函数；每次为共享对象公布错误、警告或信息性通知时调用。如果要响应此事件处理函数，您必须创建一个函数来处理共享对象生成的信息对象。

信息对象具有一个 `code` 属性（该属性包含的字符串说明 `onStatus` 处理函数的结果）和一个 `level` 属性（该属性包含字符串 `"Status"` 或 `"Error"`）。

除了此 `onStatus` 处理函数外，Flash 还提供称作 `System.onStatus` 的“超级”函数。如果为特定对象调用了 `onStatus` 但未分配任何函数对其进行响应，则 Flash 将处理分配到 `System.onStatus` 的函数（如果存在）。

当发生特定的 `SharedObject` 活动时，以下事件将发出通知。

Code 属性	Level 属性	含义
<code>SharedObject.Flush.Failed</code>	<code>Error</code>	返回 "pending" 的 <code>SharedObject.flush()</code> 命令失败（当 Flash Player 显示“本地存储设置”对话框时，用户未为共享对象分配额外的磁盘空间）。
<code>SharedObject.Flush.Success</code>	<code>Status</code>	返回 "pending" 的 <code>SharedObject.flush()</code> 命令成功完成（用户为共享对象分配了额外的磁盘空间）。

另请参见

[SharedObject.getLocal\(\)](#), [System.onStatus](#)

Sound 类

可用性

Flash Player 5。

说明

`Sound` 类使您可以控制影片中的声音。可以在影片正在播放时从库中向该影片剪辑添加声音，并控制这些声音。如果在创建新 `Sound` 对象时没有指定 `target`，则可以使用方法控制整个影片的声音。

在调用 `Sound` 类的方法之前，您必须使用构造函数 `new Sound` 创建 `Sound` 对象。

Sound 类的方法摘要

方法	说明
Sound.attachSound()	附加在参数中指定的声音。
Sound.getBytesLoaded()	返回为指定声音加载的字节数。
Sound.getBytesTotal()	以字节为单位返回声音的大小。
Sound.getPan()	返回上一个 setPan() 调用的值。
Sound.getTransform()	返回上一个 setTransform() 调用的值。
Sound.getVolume()	返回上一个 setVolume() 调用的值。
Sound.loadSound()	将 MP3 文件加载到 Flash Player 中。
Sound.setPan()	设置声音的左 / 右均衡。
Sound.setTransform()	设置要在每个扬声器中播放的每个声道（左声道和右声道）的音量。
Sound.setVolume()	设置声音的音量级别。
Sound.start()	从头开始播放声音，或者可选择从参数中设置的某偏移点开始播放声音。
Sound.stop()	停止指定声音或当前播放的所有声音。

Sound 类的属性摘要

属性	说明
Sound.duration	声音的长度，以毫秒为单位。
Sound.ID3	提供对作为 MP3 文件一部分的元数据的访问。
Sound.position	声音已播放的毫秒数。

Sound 类的事件处理函数摘要

事件处理函数	说明
Sound.onID3	每次有新的 ID3 数据可用时调用。
Sound.onLoad	加载声音时调用。
Sound.onSoundComplete	声音停止播放时调用。

Sound 类的构造函数

可用性

Flash Player 5。

用法

```
new Sound([target])
```

参数

target Sound 对象操作的影片剪辑实例。此参数是可选的。

返回

无。

说明

构造函数；为指定的影片剪辑创建新的 Sound 对象。如果没有指定目标实例，则 Sound 对象控制影片中的所有声音。

示例

下面的示例创建名为 `global_sound` 的新 Sound 对象。示例中的第二行调用 `setVolume()` 并将影片中的所有声音的音量调整为 50%。

```
global_sound = new Sound();  
global_sound.setVolume(50);
```

下面的示例创建一个新 Sound 对象，将目标影片剪辑 `my_mc` 传递给它，然后调用 `start` 方法，该方法开始播放 `my_mc` 中的所有声音。

```
movie_sound = new Sound(my_mc);  
movie_sound.start();
```

Sound.attachSound()

可用性

Flash Player 5。

用法

```
my_sound.attachSound("idName")
```

参数

idName 库中导出声音的标识符。该标识符位于“链接属性”对话框。

返回

无。

说明

方法；将 *idName* 参数中指定的声音附加到指定的 Sound 对象。该声音必须位于当前 SWF 文件的库中，并且必须已经在“链接属性”对话框中指定为导出。必须调用 `Sound.start()` 才能开始播放该声音。

为了确保从 SWF 文件中的任何场景都可以控制声音，请将声音放置在 SWF 文件的主时间轴上。

Sound.duration

可用性

Flash Player 6。

用法

```
my_sound.duration
```

说明

属性（只读）；声音的持续时间，以毫秒为单位。

Sound.getBytesLoaded()

可用性

Flash Player 6。

用法

```
my_sound.getBytesLoaded()
```

参数

无。

返回

一个整数，指示所加载的字节数。

说明

方法；返回为指定 Sound 对象加载（进入流）的字节数。可以比较 `getBytesLoaded()` 的值与 `getBytesTotal()` 的值，以确定已加载声音的百分比。

另请参见

[Sound.getBytesTotal\(\)](#)

Sound.getBytesTotal()

可用性

Flash Player 6。

用法

```
my_sound.getBytesTotal()
```

参数

无。

返回

一个整数，以字节为单位指示指定 Sound 对象的总大小。

说明

方法；以字节为单位返回指定 Sound 对象的大小。

另请参见

[Sound.getBytesLoaded\(\)](#)

Sound.getPan()

可用性

Flash Player 5。

用法

```
my_sound.getPan();
```

参数

无。

返回

一个整数。

说明

方法；返回在上一次 `setPan()` 调用中设置的面板级别，这是一个从 -100（左）到 100（右）之间的整数。（0 平衡地设置左右声道。）该面板设置控制 SWF 文件中当前和将来声音的左右均衡。

此方法是用 `setVolume()` 或 `setTransform()` 累积的。

另请参见

[Sound.setPan\(\)](#)

Sound.getTransform()

可用性

Flash Player 5。

用法

```
my_sound.getTransform();
```

参数

无。

返回

一个具有属性的对象，这些属性包含指定声音对象的声道百分比值。

说明

方法；返回用上一次 [Sound.setTransform\(\)](#) 调用设置的指定 Sound 对象的声音转换信息。

Sound.getVolume()

可用性

Flash Player 5。

用法

```
my_sound.getVolume()
```

参数

无。

返回

一个整数。

说明

方法；返回音量级别，这是一个从 0 到 100 之间的整数，其中 0 表示关闭，100 表示最大音量。默认设置为 100。

另请参见

[Sound.setVolume\(\)](#)

Sound.ID3

可用性

Flash Player 6 ；行为在 Flash Player 7 中进行了更新。

用法

```
my_sound.ID3
```

说明

属性（只读）；提供对作为 MP3 文件一部分的元数据的访问。

MP3 声音文件可以包含 ID3 标签，ID3 标签提供有关该文件的元数据。如果使用 [Sound.attachSound\(\)](#) 或 [Sound.loadSound\(\)](#) 加载的 MP3 声音包含 ID3 标签，则您可以查询这些属性。只支持使用 UTF-8 字符集的 ID3 标签。

Flash Player 6 40 版和更高版本使用 `Sound.id3` 属性支持 ID3 1.0 和 ID3 1.1 标签。Flash Player 7 添加了对 ID3 2.0 标签的支持，特别是 2.3 和 2.4。为了向后兼容，同时支持 `Sound.id3` 和 `Sound.ID3`。只有小写的 `id3` 才支持代码提示（请参见第 56 页的“使用代码提示”）。

下表列出了标准 ID3 2.0 标签以及这些标签所表示内容的类型，您可以以 `my_sound.ID3.COMM`、`my_sound.ID3.TIME` 等格式查询这些标签。MP3 文件可以包含此表中
所列标签之外的其它标签，`Sound.ID3` 也提供对这些标签的访问。

属性	说明
COMM	注释
TALB	唱片 / 影片 / 演出名称
TBPM	每分钟节拍数
TCOM	作曲家
TCON	内容类型
TCOP	版权信息
TDAT	日期
TDLY	播放列表延迟
TENC	编码器
TEXT	歌词作者 / 乐谱作者
TFLT	文件类型
TIME	时间
TIT1	内容组描述
TIT2	标题 / 歌曲名称 / 内容描述
TIT3	副标题 / 主要描述
TKEY	最初关键字
TLAN	语言

属性	说明
TLEN	长度
TMED	媒体类型
TOAL	原唱片 / 影片 / 演出名称
TOFN	原文件名
TOLY	原歌词作者 / 乐谱作者
TOPE	原歌手 / 演奏者
TORY	最初发行年份
TOWN	文件所有者 / 许可证持有人
TPE1	领衔演奏者 / 独唱（奏）者
TPE2	乐队 / 交响乐队 / 伴奏
TPE3	指挥 / 主要演奏者
TPE4	翻译、混录员、或以其它方式进行修改的人员
TPOS	作品集的部分
TPUB	发行人
TRCK	音轨编号 / 作品集中的位置
TRDA	录制日期
TRSN	Internet 电台名称
TRSO	Internet 电台所有者
TSIZ	大小
TSRC	ISRC（国际标准音像制品编码）
TSSE	编码使用的软件 / 硬件和设置
TYER	年份
WXXX	URL 链接帧

Flash Player 6 支持若干种 ID3 1.0 标签。如果这些标签不在 MP3 文件中，而相应的 ID3 2.0 标签却在 MP3 文件中，则 ID3 2.0 标签会被复制到下表所示的 ID3 1.0 属性中。此过程提供与您已编写的、读取 ID3 1.0 属性的脚本的向后兼容性。

ID3 2.0 标签	相应的 ID3 1.0 属性
COMM	Sound.id3.comment
TALB	Sound.id3.album
TCON	Sound.id3.genre
TIT2	Sound.id3.songname
TPE1	Sound.id3.artist

ID3 2.0 标签	相应的 ID3 1.0 属性
TRCK	Sound.id3.track
TYER	Sound.id3.year

示例

有关此属性用法的示例，请参见 [Sound.onID3](#)。

另请参见

[Sound.attachSound\(\)](#), [Sound.loadSound\(\)](#)

Sound.loadSound()

可用性

Flash Player 6。

用法

```
my_sound.loadSound("url", isStreaming)
```

参数

- url* MP3 声音文件在服务器上的位置。
- isStreaming* 一个布尔值，指示声音是声音流 (true) 还是事件声音 (false)。

返回

无。

说明

方法；将 MP3 文件加载到 Sound 对象中。可以使用 *isStreaming* 参数指示该声音是事件声音还是声音流。

事件声音在完全加载后才能播放。它们由动作脚本 Sound 类进行管理，并且响应此类的所有方法和属性。

声音流在下载的同时播放。当接收的数据足以启动解压缩程序时，播放开始。

使用此方法加载的所有 MP3（事件或流）都保存在用户系统上浏览器的文件缓存中。

示例

下面的示例加载事件声音：

```
my_sound.loadSound( "http://serverpath:port/mp3filename", false);
```

下面的示例加载声音流：

```
my_sound.loadSound( "http://serverpath:port/mp3filename", true);
```

另请参见

[Sound.onLoad](#)

Sound.onID3

可用性

Flash Player 7。

用法

```
my_sound.onID3 = function(){  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；每次有新的 ID3 数据可用于使用 [Sound.attachSound\(\)](#) 或 [Sound.loadSound\(\)](#) 加载的 MP3 文件时调用。此处理程序提供对 ID3 数据的访问而无需轮询。如果文件中同时出现 ID3 1.0 和 ID3 2.0 标签，则此处理函数将被调用两次。

示例

下面的示例用 trace 命令将 song.mp3 的 ID3 属性发送到“输出”面板。

```
my_sound = new Sound();  
my_sound.onID3 = function(){  
    for( var prop in my_sound.ID3 ){  
        trace( prop + " :" + my_sound.ID3[prop] );  
    }  
}  
my_sound.loadSound("song.mp3", false);
```

另请参见

[Sound.attachSound\(\)](#), [Sound.ID3](#), [Sound.loadSound\(\)](#)

Sound.onLoad

可用性

Flash Player 6。

用法

```
my_sound.onLoad = function(success){  
    // 此处是您的语句  
}
```

参数

success 如果 *my_sound* 已成功加载，则为布尔值 `true`，否则为 `false`。

返回

无。

说明

事件处理函数；当加载声音时自动调用。必须创建在调用此处理函数时执行的函数。您可以使用匿名函数或命名函数（有关每种情况的示例，请参见 [Sound.onSoundComplete](#)）。您应在调用 *my_sound.loadSound()* 之前定义此处理函数。

另请参见

[Sound.loadSound\(\)](#)

Sound.onSoundComplete

可用性

Flash Player 6。

用法

```
my_sound.onSoundComplete = function(){  
    // 此处是您的语句  
}
```

参数

无。

返回

无。

说明

事件处理函数；当声音播放完时自动调用。您可以使用此处理函数在声音播放完毕后触发 SWF 文件中的事件。

必须创建一个在调用该处理函数时执行的函数。既可以使用匿名函数也可以使用命名函数。

示例

用法 1：下面的示例使用匿名函数：

```
my_sound = new Sound();  
my_sound.attachSound("mySoundID");
```

```
my_sound.onSoundComplete = function() {  
    trace("mySoundID completed");  
}  
my_sound.start();
```

用法 2：下面的示例使用命名函数：

```
function callback1() {  
    trace("mySoundID completed");  
}
```

```
my_sound = new Sound();  
my_sound.attachSound("mySoundID");  
my_sound.onSoundComplete = callback1;  
my_sound.start();
```

另请参见

[Sound.onLoad](#)

Sound.position

可用性

Flash Player 6。

用法

```
my_sound.position
```

说明

属性（只读）；声音已播放的毫秒数。如果声音是循环的，则在每次循环开始时，位置将被重置为 0。

Sound.setPan()

可用性

Flash Player 5。

用法

```
my_sound.setPan(pan);
```

参数

pan 一个整数，指定声音的左右均衡。有效值的范围为 -100 到 100，其中 -100 表示仅使用左声道，100 表示仅使用右声道，而 0 表示在两个声道间平均地均衡声音。

返回

一个整数。

说明

方法；确定声音在左右声道（扬声器）中是如何播放的。对于单声道声音，*pan* 确定声音通过哪个扬声器（左或右）进行播放。

示例

下面的示例创建一个 **Sound** 对象 *my_sound*，并从库中为其附加具有标识符 **L7** 的声音。它还调用 *setVolume()* 和 *setPan()* 来控制 **L7** 声音。

```
onClipEvent(mouseDown) {  
    // 创建一个声音对象  
    my_sound = new Sound(this);  
    // 从库中附加声音  
    my_sound.attachSound("L7");  
    // 将音量设置为 50%  
    my_sound.setVolume(50);  
    // 关闭右声道中的声音  
    my_sound.setPan(-100);  
    // 从声音的第 30 秒开始播放并播放 5 遍  
    my_sound.start(30, 5);  
}
```

另请参见

[Sound.attachSound\(\)](#), [Sound.setPan\(\)](#), [Sound.setTransform\(\)](#), [Sound.setVolume\(\)](#),
[Sound.start\(\)](#)

Sound.setTransform()

可用性

Flash Player 5。

用法

```
my_sound.setTransform(soundTransformObject)
```

参数

soundTransformObject 一个用通用 Object 类的构造函数创建的对象。

返回

无。

说明

方法；设置 Sound 对象的声音转换（即，均衡）信息。

soundTransformObject 参数是一个用通用 Object 类的构造函数方法创建的对象，该对象具有指定声音在左右声道（扬声器）间如何分布的参数。

声音占用相当大的磁盘空间和内存。因为立体声声音使用的数据是单声道声音的两倍，通常情况下最好使用 22 KHz 6 位单声道声音。您可以使用 `setTransform()` 将单声道声音播放为立体声、将立体声声音播放为单声道以及向声音添加有趣的效果。

soundTransformObject 的属性如下：

- ll 一个百分数值，指定左输入在左扬声器里播放的量 (0-100)。
- lr 一个百分数值，指定右输入在左扬声器里播放的量 (0-100)。
- rr 一个百分数值，指定右输入在右扬声器里播放的量 (0-100)。
- rl 一个百分数值，指定左输入在右扬声器里播放的量 (0-100)。

这些参数的净结果由以下公式表示：

```
leftOutput = left_input * ll + right_input * lr  
rightOutput = right_input * rr + left_input * rl
```

left_input 或 *right_input* 的值由 SWF 文件中的声音类型（立体声或单声道）确定。

立体声声音将声音输入在左右扬声器之间平均分配，且在默认情况下具有下面的转换设置：

```
ll = 100  
lr = 0  
rr = 100  
rl = 0
```

单声道声音在左扬声器里播放所有的声音输入，且在默认情况下具有下面的转换设置：

```
ll = 100  
lr = 100  
rr = 0  
rl = 0
```

示例

下面的示例说明了这样一个设置，它可以通过使用 `setTransform()` 实现，但不能通过使用 `setVolume()` 或 `setPan()` 实现，即使将后两种方法结合起来使用也无法实现该设置。

下面的代码新建一个 `soundTransformObject` 对象并设置其属性，以便只在左声道中播放来自两个声道的声音。

```
mySoundTransformObject = new Object;  
mySoundTransformObject.ll = 100;  
mySoundTransformObject.lr = 100;  
mySoundTransformObject.rr = 0;  
mySoundTransformObject.rl = 0;
```

若要将 `soundTransformObject` 对象应用到 `Sound` 对象，则需要使用 `setTransform()` 将此对象传递给 `Sound` 对象，如下所示：

```
my_sound.setTransform(mySoundTransformObject);
```

下面的示例将立体声声音播放为单声道声音；`soundTransformObjectMono` 对象具有下列参数：

```
mySoundTransformObjectMono = new Object;  
mySoundTransformObjectMono.ll = 50;  
mySoundTransformObjectMono.lr = 50;  
mySoundTransformObjectMono.rr = 50;  
mySoundTransformObjectMono.rl = 50;  
my_sound.setTransform(soundTransformObjectMono);
```

此示例只播放左声道音量的一半，而将左声道的剩余部分添加到右声道；`soundTransformObjectHalf` 对象具有下列参数：

```
mySoundTransformObjectHalf = new Object;  
mySoundTransformObjectHalf.ll = 50;  
mySoundTransformObjectHalf.lr = 0;  
mySoundTransformObjectHalf.rr = 100;  
mySoundTransformObjectHalf.rl = 50;  
my_sound.setTransform(soundTransformObjectHalf);
```

另请参见

[Object 类](#)

Sound.setVolume()

可用性

Flash Player 5。

用法

```
my_sound.setVolume(volume)
```

参数

volume 一个从 0 到 100 之间的数字，表示声音级别。100 为最大音量，而 0 为没有音量。默认设置为 100。

返回

无。

说明

方法；设置 Sound 对象的音量。

示例

下面的示例将音量设置为 50%，并随时间的推移，不断地将声音从左扬声器逐渐传输到右扬声器：

```
onClipEvent (load) {  
    i = -100;  
    my_sound = new Sound();  
    my_sound.setVolume(50);  
}  
onClipEvent (enterFrame) {  
    if (i <= 100) {  
        my_sound.setPan(i++);  
    }  
}
```

另请参见

[Sound.setPan\(\)](#), [Sound.setTransform\(\)](#)

Sound.start()

可用性

Flash Player 5。

用法

```
my_sound.start([secondOffset, loop])
```

参数

secondOffset 一个可选参数，用于从某个特定点开始播放声音。例如，如果您有一个 30 秒的声音，而您希望该声音从中间开始播放，可将 *secondOffset* 参数指定为 15。并非声音延迟 15 秒，而是从 15 秒标记处开始播放。

loop 一个可选参数，用于指定声音连续播放的次数。

返回

无。

说明

方法；如果未指定参数，则从开头开始播放最近附加的声音；或者从参数 *secondOffset* 指定的声音点处开始播放。

另请参见

[Sound.stop\(\)](#)

Sound.stop()

可用性

Flash Player 5。

用法

```
my_sound.stop(["idName"])
```

参数

idName 一个可选参数，用于指定要停止播放的某个特定声音。*idName* 参数必须置于引号 (" ") 之中。

返回

无。

说明

方法；如果未指定参数，则停止当前播放的所有声音，否则只停止在 *idName* 参数中指定的声音。

另请参见

[Sound.start\(\)](#)

_soundbuftime

可用性

Flash Player 4。

用法

```
_soundbuftime = integer
```

参数

integer 在 SWF 文件开始进入流之前缓冲的秒数。

说明

属性（全局）；规定声音流缓冲的秒数。默认值为 5 秒。

Stage 类

可用性

Flash Player 6。

说明

Stage 类是一个顶级类，不必使用构造函数即可访问其方法、属性和处理函数。

此类的方法和属性用于访问和操作有关 SWF 文件边界的信息。

Stage 类的方法摘要

方法	说明
Stage.addListener()	添加一个检测 SWF 文件的大小何时进行了调整的侦听器对象。
Stage.removeListener()	从 Stage 对象删除侦听器对象。

Stage 类的属性摘要

属性	说明
Stage.align	SWF 文件在播放器或浏览器中的对齐方式。
Stage.height	舞台的高度，以像素为单位。
Stage.scaleMode	SWF 文件的当前缩放设置。
Stage.showMenu	显示或隐藏 Flash Player 上下文菜单中的默认项。
Stage.width	舞台的宽度，以像素为单位。

Stage 类的事件处理函数摘要

事件处理函数	说明
Stage.onResize	当 Stage.scaleMode 设置为 "noScale" 而且 SWF 文件的大小进行了调整时调用。

Stage.addListener()

可用性

Flash Player 6。

用法

```
Stage.addListener(myListener)
```

参数

myListener 侦听来自 [Stage.onResize](#) 事件的回调通知的对象。

返回

无。

说明

方法；检测 SWF 文件的大小何时进行了调整（仅当 `Stage.scaleMode = "noScale"` 的情况下）。当缩放设置为默认影片缩放设置（`"showAll"`）或其它缩放设置（`"exactFit"` 和 `"noBorder"`）时，`addListener()` 方法不起作用。

若要使用 `addListener()`，必须先创建一个侦听器对象。舞台侦听器对象接收来自 `Stage.onResize` 的通知。

示例

此示例新建一个名为 `myListener` 的侦听器对象。然后使用 `myListener` 调用 `onResize`，并定义一个当触发 `onResize` 时将调用的函数。最后，此代码将 `myListener` 对象添加到 `Stage` 对象的回调列表中。侦听器对象允许多个对象侦听调整大小的通知。

```
myListener = new Object();  
myListener.onResize = function () { ... }  
Stage.scaleMode = "noScale"  
Stage.addListener(myListener);
```

另请参见

[Stage.onResize](#), [Stage.removeListener\(\)](#)

Stage.align

可用性

Flash Player 6。

用法

Stage.align

说明

属性；指示 SWF 文件在播放器或浏览器中的当前对齐方式。

下表列出了 align 属性的值。此处未列出的值会将 SWF 文件居中放置在播放器或浏览器区域内。

值	垂直	水平
"T"	顶部	中间
"B"	底部	中间
"L"	中间	左侧
"R"	中间	右侧
"TL"	顶部	左侧
"TR"	顶部	右侧
"BL"	底部	左侧
"BR"	底部	右侧

Stage.height

可用性

Flash Player 6。

用法

Stage.height

说明

属性（只读）；以像素为单位指示舞台的当前高度。当 Stage.scaleMode 的值为 "noScale" 时，height 属性表示播放器的高度。当 Stage.scaleMode 的值不为 "noScale" 时，height 表示 SWF 文件的高度。

另请参见

[Stage.align](#), [Stage.scaleMode](#), [Stage.width](#)

Stage.onResize

可用性

Flash Player 6。

用法

```
myListener.onResize = function(){  
    // 此处是您的语句  
}
```

参数

无。

参数

无。

返回

无。

说明

事件处理函数；当 `Stage.scaleMode` 设置为 "noScale" 并且 SWF 文件的大小调整时调用此函数。可以利用此事件处理函数编写一个函数，当 SWF 文件的大小调整时布置舞台上对象的布局。

示例

当舞台大小调整时，下面的示例在“输出”面板中显示一条消息。

```
Stage.scaleMode = "noScale"  
myListener = new Object();  
myListener.onResize = function () {  
    trace("Stage size is now " + Stage.width + " by " + Stage.height);  
}  
Stage.addListener(myListener);  
// 稍后, 调用 Stage.removeListener(myListener)
```

另请参见

[Stage.addListener\(\)](#), [Stage.removeListener\(\)](#)

Stage.removeListener()

可用性

Flash Player 6。

用法

```
Stage.removeListener(myListener)
```

参数

myListener 用 `addListener()` 添加到对象回调列表中的对象。

返回

一个布尔值。

说明

方法；删除用 `addListener()` 创建的侦听器对象。

另请参见

[Stage.addListener\(\)](#)

Stage.scaleMode

可用性

Flash Player 6。

用法

```
Stage.scaleMode = "value"
```

说明

属性；指示舞台内 SWF 文件的当前缩放设置。`scaleMode` 属性将 SWF 文件强制设置为特定的缩放模式。默认情况下，SWF 文件使用在“发布设置”对话框中设置的 HTML 参数。

`scaleMode` 属性可以使用以下值：“exactFit”、“showAll”、“noBorder”和“noScale”。任何其它值都会将 `scaleMode` 属性设置为默认值“showAll”。

Stage.showMenu

可用性

Flash Player 6。

用法

```
Stage.showMenu
```

说明

属性（只读）；指定显示或隐藏 Flash Player 上下文菜单中的默认项。如果 `showMenu` 设置为 `true`（默认设置），所有上下文菜单项都将显示。如果 `showMenu` 设置为 `false`，则只出现“设置”项。

另请参见

[ContextMenu](#) 类, [ContextMenuItem](#) 类

Stage.width

可用性

Flash Player 6。

用法

Stage.width

说明

属性（只读）；以像素为单位指示舞台的当前宽度。当 [Stage.scaleMode](#) 的值为 "noScale" 时，width 属性表示播放器的宽度。当 [Stage.scaleMode](#) 的值不为 "noScale" 时，width 表示 SWF 文件的宽度。

另请参见

[Stage.align](#), [Stage.height](#), [Stage.scaleMode](#)

startDrag()

可用性

Flash Player 4。

用法

[startDrag\(target,\[lock ,left , top , right, bottom\]\)](#)

参数

target 要拖动的影片剪辑的目标路径。

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

left、*top*、*right*、*bottom* 相对于影片剪辑父级坐标的值，这些值指定该影片剪辑的约束矩形。这些参数是可选的。

返回

无。

说明

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

示例

若要创建用户可以放在任何位置的影片剪辑，可将 [startDrag\(\)](#) 和 [stopDrag\(\)](#) 动作附加到该影片剪辑内的某个按钮上。

```
on (press) {  
    startDrag(this,true);  
}  
on (release) {  
    stopDrag();  
}
```

另请参见

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

static

可用性

Flash Player 6。

用法

```
class someClassName{
    static var name;
    static function name() {
        // 此处是您的语句
    }
}
```

注意：若要使用此关键字，必须在 FLA 文件的“发布设置”对话框的“Flash”选项卡上指定“动作脚本 2.0”和“Flash Player 6 或更高版本”。仅支持在外部脚本文件中使用此关键字，而不支持在用“动作”面板编写的脚本中使用此关键字。

参数

name 要指定为 static 的变量或函数的名称。

说明

关键字；指定某个变量或函数只为每个类创建一次，而不是在基于该类的每个对象中都创建。有关更多信息，请参见第 145 页的“实例成员和类成员”。

您只能在类定义中使用此关键字，不能在接口定义中使用。

另请参见

[private](#), [public](#)

stop()

可用性

Flash 2。

用法

stop

参数

无。

返回

无。

说明

函数；停止当前正在播放的 SWF 文件。此动作最通常的用法是用按钮控制影片剪辑。

stopAllSounds()

可用性

Flash Player 3。

用法

```
stopAllSounds()
```

参数

无。

返回

无。

说明

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

示例

下面的代码可以应用到一个按钮，这样当单击此按钮时，将停止 SWF 文件中的所有声音。

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

另请参见

[Sound 类](#)

stopDrag()

可用性

Flash Player 4。

用法

```
stopDrag()
```

参数

无。

返回

无。

说明

函数；停止当前的拖动操作。

示例

此代码在用户释放鼠标按钮时，在实例 my_mc 上停止拖动动作：

```
on (press) {  
    startDrag("my_mc");  
}
```

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

另请参见

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

" "（字符串分隔符）

可用性

Flash Player 4。

用法

`"text"`

参数

text 一个字符。

返回

无。

说明

字符串分隔符；在字符的前后加上引号后，引号指示这些字符具有其字面值；这些字符将被视作一个字符串，而不是一个变量、数值或其它动作脚本元素。

示例

此示例使用引号指示变量 *yourGuess* 的值是文本字符串 “Prince Edward Island”，而不是变量名。province 的值是一个变量，而不是文本；要确定 province 的值，必须找到 *yourGuess* 的值。

```
yourGuess = "Prince Edward Island";  
  
on (release) {  
    province = yourGuess;  
    trace(province);  
}  
  
// 在“输出”面板中显示 “Prince Edward Island”
```

另请参见

[String 类](#), [String\(\)](#)

String 类

可用性

Flash Player 5（已成为 Flash Player 6 本身的对象，Flash Player 6 大大提高了性能）。

说明

String 类是字符串原始数据类型的包装，提供用于操作原始字符串值类型的方法和属性。您可以通过 String() 函数将任何对象的值转换为字符串。

除了 concat()、fromCharCode()、slice() 和 substr() 之外，String 对象的所有其它方法都是通用方法。这意味着这些方法本身先调用 this.toString()，然后再执行它们的操作，而且您可以将这些方法用于其它非 String 对象。

因为所有字符串索引都是从零开始的，所以任何字符串 x 的最后一个字符的索引都是 x.length - 1。

可以使用构造函数方法 new String 或者使用字符串文本值调用 String 类的任何方法。如果您指定了一个字符串，则动作脚本解释程序自动将其转换为一个临时 String 对象，再调用方法，然后放弃该临时 String 对象。您还可以将 String.length 属性用于字符串。

请不要将字符串和 String 对象相混淆。在下面的示例中，第一行代码创建字符串 s1，而第二行代码创建 String 对象 s2。

```
s1 = "foo"
s2 = new String("foo")
```

除非您确实需要使用 String 对象，否则请使用字符串。

String 类的方法摘要

方法	说明
String.charAt()	返回字符串中特定位置处的字符。
String.charCodeAt()	返回指定索引处字符的值，此值为介于 0 到 65535 之间的一个 16 位整数。
String.concat()	合并两个字符串的文本，并返回一个新字符串。
String.fromCharCode()	返回由参数中指定的字符组成的字符串。
String.indexOf()	返回指定子字符串的第一个匹配项的位置。
String.lastIndexOf()	返回指定子字符串的最后一个匹配项的位置。
String.slice()	提取字符串的一部分，并返回一个新字符串。
String.split()	通过将字符串分隔为子字符串，从而将 String 对象拆分为字符串数组。
String.substr()	返回字符串内指定数量的字符，字符计数从指定的位置开始。
String.substring()	返回字符串内两个索引间的字符。
String.toLowerCase()	将字符串转换为小写然后返回结果；不更改原始对象的内容。
String.toUpperCase()	将字符串转换为大写然后返回结果；不更改原始对象的内容。

String 类的属性摘要

属性	说明
<code>String.length</code>	一个非从零开始的整数，表示指定 String 对象中的字符数。

String 类的构造函数

可用性

Flash Player 5。

用法

```
new String(value)
```

参数

value 新 String 对象的初始值。

返回

无。

说明

构造函数；创建一个新 String 对象。

另请参见

`String()`, " " (字符串分隔符)

String.charAt()

可用性

Flash Player 5。

用法

```
my_str.charAt(index)
```

参数

index 整数，指定字符在字符串中的位置。第一个字符由 0 表示，最后一个字符由 *my_str*.length-1 表示。

返回

一个字符。

说明

方法；返回参数 *index* 所指定位置处的字符。如果 *index* 不是 0 到 *string.length* - 1 之间的数字，则返回一个空字符串。

此方法与 `String.charCodeAt()` 类似，所不同的是它返回的值是字符，而不是 16 位整数字符代码。

示例

在下面的示例中，对字符串 "Chris" 的第一个字母调用了此方法。

```
my_str = new String("Chris");  
i = my_str.charCodeAt(0); // i = "C"
```

String.charCodeAt()

可用性

Flash Player 5。

用法

```
my_str.charCodeAt(index)
```

参数

index 整数，指定字符在字符串中的位置。第一个字符由 0 表示，最后一个字符由 *my_str.length-1* 表示。

返回

一个整数。

说明

方法；返回一个 0 到 65535 之间的 16 位整数，表示由 *index* 指定的字符。如果 *index* 不是 0 到 *string.length - 1* 之间的数字，则返回 NaN。

此方法与 [String.charAt\(\)](#) 类似，所不同的是它返回的值是 16 位整数字符代码，而不是字符。

示例

在下面的示例中，对字符串 "Chris" 的第一个字母调用了该方法。

```
my_str = new String("Chris");  
i = my_str.charCodeAt(0); // i = 67
```

String.concat()

可用性

Flash Player 5。

用法

```
my_str.concat(value1,...valueN)
```

参数

value1...valueN 零或多个要连接的值。

返回

字符串。

说明

方法；将该 String 对象的值与参数合并，并返回新组成的字符串；而原始值 *my_str* 不变。

String.fromCharCode()

可用性

Flash Player 5。

用法

```
String.fromCharCode(c1,c2,...cN)
```

参数

c1,*c2*,...*cN* 表示 ASCII 值的十进制整数。

返回

字符串。

说明

方法；返回一个由参数中 ASCII 值表示的字符组成的字符串。

示例

此示例使用 `fromCharCode()` 在电子邮件地址中插入一个 @ 字符。

```
address_str = "dog" + String.fromCharCode(64) + "house.net";  
trace(address_str); // dog@house.net
```

String.indexOf()

可用性

Flash Player 5。

用法

```
my_str.indexOf(substring, [startIndex])
```

参数

substring 一个整数或字符串，指定要在 *my_str* 中搜索的子字符串。

startIndex 一个可选整数，指定要在 *my_str* 中搜索该子字符串的起始点。

返回

指定子字符串的第一个匹配项的位置，或 -1。

说明

方法；搜索字符串，并返回在调用字符串内 *startIndex* 处或之后找到的 *substring* 的第一个匹配项的位置。如果没有找到 *substring*，则此方法返回 -1。

另请参见

[String.lastIndexOf\(\)](#)

String.lastIndexOf()

可用性

Flash Player 5。

用法

```
my_str.lastIndexOf(substring, [startIndex])
```

参数

substring 一个整数或字符串，指定要搜索的字符串。

startIndex 一个可选整数，指定搜索 *substring* 的起始点。

返回

指定子字符串的最后一个匹配项的位置，或 -1。

说明

方法；在字符串中从右向左搜索，并返回调用此方法的字符串中在 *startIndex* 之前找到的 *substring* 的最后一个匹配项的索引。如果没有找到 *substring*，则此方法返回 -1。

另请参见

[String.indexOf\(\)](#)

String.length

可用性

Flash Player 5。

用法

```
my_str.length
```

说明

属性；一个非从零开始的整数，表示指定 String 对象中的字符数。

因为所有字符串索引都是从零开始的，所以任何字符串 *x* 的最后一个字符的索引都是 *x.length - 1*。

String.slice()

可用性

Flash Player 5。

用法

```
my_str.slice(start, [end])
```

参数

start 指定片段起始点索引的数字。如果 *start* 是一个负数，则起始点从字符串的结尾开始确定，-1 表示最后一个字符。

end 一个比片段终点索引大 1 的整数。提取出来的字符串中不包括由 *end* 参数索引指定的字符。如果省略了此参数，将使用 `String.length`。如果 *end* 是一个负数，则终点根据从字符串的结尾向后数确定，-1 表示最后一个字符。

返回

指定字符串的子字符串。

说明

方法；返回一个字符串，该字符串包括从 *start* 字符直到 *end* 字符（但不包括该字符）之间的所有字符。不修改原始 `String` 对象。如果未指定 *end* 参数，则子字符串的结尾就是原字符串的结尾。如果 *start* 的值大于或等于 *end* 的值，则此方法返回一个空字符串。

示例

下面的示例先设置一个变量 `text`，再创建一个 `String` 对象 `my_str`，然后将 `text` 变量传递给该对象。`slice()` 方法提取该变量中所包含字符串的一部分，然后 `trace()` 将其发送到“输出”面板。下面的示例说明对 *end* 参数使用正值和负值两种情况。

```
text = "Lexington";
my_str = new String( text );
trace(my_str.slice( 1, 3 )); // "ex"
trace(my_str.slice( 1, -6 )); // "ex"
```

另请参见

[String.substr\(\)](#), [String.substring\(\)](#)

String.split()

可用性

Flash Player 5。

用法

```
my_str.split("delimiter", [limit])
```

参数

delimiter 拆分 *my_str* 所依据的字符或字符串。

limit 要放入数组中的项目数。此参数是可选的。

返回

包含 *my_str* 的子字符串的数组。

说明

方法；在指定的 *delimiter* 参数出现的所有位置断开 *String* 对象，将其拆分为多个子字符串，然后以数组形式返回这些子字符串。如果使用空字符串 ("") 作为分隔符，则字符串中的每个字符都将被作为元素放入数组中，如下面的代码所示。

```
my_str = "Joe";  
i = my_str.split("");  
trace (i);
```

“输出”面板将显示以下结果：

```
J,o,e
```

如果未定义 *delimiter* 参数，则会将整个字符串放入返回数组的第一个元素中。

示例

下面的示例返回含有五个元素的数组。

```
my_str = "P, A, T, S, Y";  
my_str.split(",");
```

此示例返回一个具有以下两个元素的数组：“P” 和 “A”。

```
my_str.split(", ", 2);
```

String.substr()

可用性

Flash Player 5。

用法

```
my_str.substr(start, [length])
```

参数

start 一个整数，指示 *my_str* 中用于创建子字符串的第一个字符的位置。如果 *start* 为一个负数，则起始位置从字符串的结尾开始确定，-1 表示最后一个字符。

length 要创建的子字符串中的字符数。如果没有指定 *length*，则子字符串包括从 *start* 开始直到字符串结尾的所有字符。

返回

指定字符串的子字符串。

说明

方法；返回字符串中的字符，这些字符从 *start* 参数所指定的索引开始，直至达到 *length* 参数所指定的字符数为止。substr 方法不更改由 *my_str* 指定的字符串，而是返回一个新字符串。

String.substring()

可用性

Flash Player 5。

用法

```
my_str.substring(start, [end])
```

参数

start 一个整数，指示 *my_str* 中用于创建子字符串的第一个字符的位置。*start* 的有效值为 0 到 *String.length* - 1。如果 *start* 为负值，则使用 0。

end 一个整数，为 1 + *my_str* 中要提取的最后一个字符的索引。*end* 的有效值为 1 到 *String.length*。提取出来的字符串中不包括由 *end* 参数索引指定的字符。如果省略了此参数，将使用 *String.length*。如果此参数是一个负值，则使用 0。

返回

字符串。

说明

方法；返回一个字符串，此字符串由 *start* 和 *end* 参数指定的两点之间的字符组成。如果未指定 *end* 参数，则子字符串的结尾就是原字符串的结尾。如果 *start* 的值等于 *end* 的值，则此方法返回一个空字符串。如果 *start* 的值大于 *end* 的值，则在函数执行前两个参数将自动互换，且原始值不变。

String.toLowerCase()

可用性

Flash Player 5。

用法

```
my_str.toLowerCase()
```

参数

无。

返回

字符串。

说明

方法；返回 String 对象的一个副本，其中所有的大写字符转换为小写。原始值不变。

String.toUpperCase()

可用性

Flash Player 5。

用法

```
my_str.toUpperCase()
```

参数

无。

返回

字符串。

说明

方法；返回 String 对象的一个副本，其中所有的小写字符转换为大写。原始值不变。

String()

可用性

Flash Player 4；行为在 Flash Player 7 中发生了变化。

用法

```
String(expression)
```

参数

expression 要转换为字符串的表达式。

返回

字符串。

说明

函数；返回指定参数的字符串表示形式，如下所示：

如果 *expression* 是数字，则返回的字符串为此数字的文本表示形式。

如果 *expression* 为字符串，则返回的字符串就是 *expression*。

如果 *expression* 是一个对象，则返回值为该对象的字符串表示形式，这是通过调用该对象的字符串属性生成的；如果不存在此类属性，则是通过调用 `Object.toString()` 生成的。

如果 *expression* 为 `undefined`，则返回值如下：

- 在为 Flash Player 6 或更早版本发布的文件中，结果是空字符串 ("")。
- 在为 Flash Player 7 或更高版本发布的文件中，结果是 `undefined`。

如果 *expression* 为布尔值，则返回的字符串为 "true" 或 "false"。

如果 *expression* 是一个影片剪辑，则返回值是以斜杠 (/) 记号表示的该影片剪辑的目标路径。

注意：动作脚本 2.0 不支持斜杠记号。

另请参见

`Number.toString()`, `Object.toString()`, [String 类](#), " " (字符串分隔符)

substring

可用性

Flash Player 4。不鼓励使用此函数，而推荐使用 `String.substr()`。

用法

```
substring("string", index, count)
```

参数

string 从中提取新字符串的字符串。

index 要提取的第一个字符的编号。

count 要包含在所提取字符串中的字符数，其中不包括 *index* 字符。

返回

无。

说明

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

另请参见

`String.substr()`

super

可用性

Flash Player 6。

用法

```
super.method([arg1, ..., argN])  
super([arg1, ..., argN])
```

参数

method 要在超类中调用的方法。

arg1 可选参数，这些参数或者传递给方法的超类版本（语法 1），或者传递给超类的构造函数（语法 2）。

返回

两种格式都调用一个函数。该函数可以返回任何值。

说明

运算符：第一种语法格式可以用于对象方法体内，用以调用方法的超类版本，而且可以选择向超类方法传递参数 (*arg1 ... argN*)。这对于创建某些子类方法很有用，这些子类方法在向超类方法添加附加行为的同时，又调用这些超类方法执行其原始行为。

第二种语法格式可以用于构造函数体内，用以调用此构造函数的超类版本，而且可以选择向它传递参数。这对于创建子类很有用，该子类在执行附加的初始化的同时，又调用超类构造函数执行超类初始化。

switch

可用性

Flash Player 4。

用法

```
switch (expression){  
  caseClause:  
  [defaultClause:]  
}
```

参数

expression 任何表达式。

caseClause 一个 case 关键字，其后跟有一个表达式、冒号和一组语句，如果在使用全等 (==) 的情况下，此处的表达式与 switch 的 *expression* 参数相匹配，则执行这组语句。

defaultClause 一个 default 关键字，其后跟有一组语句，如果 case 表达式都不与 switch 的 *expression* 参数全等 (==) 匹配时，将执行这些语句。

返回

无。

说明

语句；创建动作脚本语句的分支结构。像 if 动作一样，switch 动作测试一个条件，并在条件返回 true 值时执行语句。

示例

在下面的示例中，如果 number 参数的计算结果为 1，则执行 case 1 后面的 trace() 动作，如果 number 参数的计算结果为 2，则执行 case 2 后面的 trace() 动作，依此类推。如果 case 表达式与 number 参数都不匹配，则执行 default 关键字后面的 trace() 动作。

```
switch (number) {
  case 1:
    trace ("case 1 tested true");
    break;
  case 2:
    trace ("case 2 tested true");
    break;
  case 3:
    trace ("case 3 tested true");
    break;
  default:
    trace ("no case tested true")
}
```

在下面的示例中，第一个 case 组中没有 break，因此如果 number 为 1，则 A 和 B 都将被发送到 “输出” 面板：

```
switch (number) {
  case 1:
    trace ("A");
  case 2:
    trace ("B");
    break;
  default:
    trace ("D")
}
```

另请参见

=== (全等) , break, case, default, if

System 类

可用性

Flash Player 6。

说明

这是一个顶级类，包括功能对象（请参见 [System.capabilities 对象](#)）、安全对象（请参见 [System.security 对象](#)），以及下面列出的方法、属性和事件处理函数。

System 类的方法摘要

方法	说明
System.setClipboard()	用文本字符串替换系统剪贴板的内容。
System.showSettings()	显示 Flash Player “设置” 面板。

System 类的属性摘要

方法	说明
System.exactSettings	指定当访问本地设置时是使用超级域匹配规则还是完全域匹配规则。
System.useCodepage	通知 Flash Player 是使用 Unicode 来解释外部文本文件，还是使用运行播放器的操作系统的传统代码页来解释外部文本文件。

System 类的事件处理函数摘要

方法	说明
System.onStatus	为某些对象提供超级事件处理函数

System.exactSettings

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

`System.exactSettings`

说明

属性；指定当访问本地设置（例如摄像头或麦克风访问权限）或本地永久数据（共享对象）时是使用超级域匹配规则还是完全域匹配规则。对于为 Flash Player 7 或更高版本发布的文件，默认值为 `true`，对于为 Flash Player 6 发布的文件，默认值为 `false`。

如果此值为 `true`，位于 `here.xyz.com` 的 SWF 文件的设置和数据将存储在 `here.xyz.com` 中，位于 `there.xyz.com` 的 SWF 文件的设置和数据将存储在 `there.xyz.com` 中，依此类推。如果此值为 `false`，则位于 `here.xyz.com`、`there.xyz.com` 和 `xyz.com` 的 SWF 文件的设置和数据是共享的，都将存储在 `xyz.com` 中。

如果某些文件将此属性设置为 `false` 而其它文件将其设置为 `true`，则将出现不同子域中的 Swf 文件共享设置和数据的情况。例如，如果位于 `here.xyz.com` 的 SWF 文件将此属性设置为 `false` 而位于 `xyz.com` 的 SWF 文件将此属性设置为 `true`，则两个文件将使用相同的设置和数据（即 `xyz.com` 中的设置和数据）。如果这不是您所希望的情况，则确保在每个文件中设置此属性以正确地表示所需的存储设置和数据的位置。

如果要更改此属性的默认值，则在文档的第一帧发布 `System.exactSettings = false` 命令。在发生任何需要访问本地设置的活动（例如 `System.ShowSettings()` 或 `SharedObject.getLocal()`）之后，将不能更改此属性。

如果使用 `loadMovie()`、`MovieClip.loadMovie()` 或 `MovieClipLoader.loadClip()` 将一个 SWF 文件加载到另一个 SWF 文件中，则为 Flash Player 7 发布的所有文件共享

`System.exactSettings` 的单个值，为 Flash Player 6 发布的所有文件共享

`System.exactSettings` 的单个值。因此，如果在一个为某个特定 Player 版本发布的文件中指定此属性的值，则应该在要加载的所有文件中执行此操作。如果加载多个文件，在最后一个加载的文件中指定的设置将覆盖所有以前指定的设置。

有关 Flash 中如何实现域匹配的更多信息，请参见第 166 页的“Flash Player 安全功能”。

另请参见

[SharedObject.getLocal\(\)](#)、[System.showSettings\(\)](#)

System.onStatus

可用性

Flash Player 6。

说明

事件处理函数：为某些对象提供“超级”事件处理函数。

LocalConnection、NetStream 和 SharedObject 对象提供 onStatus 事件处理函数，该事件处理函数使用信息对象提供信息、状态或错误消息。若要响应此事件处理函数，您必须创建一个函数来处理信息对象，同时还必须了解所返回的信息对象的格式和内容。

除了为上面列出的对象提供特定的 onStatus 方法之外，Flash 还提供名为 System.onStatus 的“超级”函数。如果为 level 属性为 "error" 的特定对象调用了 onStatus 但未分配任何函数对其进行响应，则 Flash 将处理分配到 System.onStatus 的函数（如果存在）。

注意：Camera 和 Microphone 类也具有 onStatus 处理函数，但不传递 level 属性为 "error" 的信息对象。因此，如果不为这些处理函数指定函数，则将不调用 System.onStatus。

下面的示例说明如何创建通用和特定函数来处理 onStatus 方法发送的信息对象。

```
// 创建通用函数
System.onStatus = function(genericError)
{
    // 您的脚本将在此处执行更有意义的操作
    trace("An error has occurred.Please try again.");
}

// 为 NetStream 对象创建函数
// 如果 NetStream 对象返回的信息对象不同于
// 下面列出的对象，其 level 属性为 "error",
// 则将调用 System.onStatus

videoStream_ns.onStatus = function(infoObject) {
    if (infoObject.code == "NetStream.Play.StreamNotFound") {
        trace("Could not find video file.");
    }
}
```

另请参见

[Camera.onStatus](#), [LocalConnection.onStatus](#), [Microphone.onStatus](#),
[NetStream.onStatus](#), [SharedObject.onStatus](#)

System.setClipboard()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
System.setClipboard(string)
```

参数

string 要放置在系统剪贴板上的纯文本字符串，替换系统剪贴板上的当前内容（如果有的话）。如果传递文本字符串而不是 String 类型的变量，则将文本引在引号中。

返回

如果文本被成功放置在剪贴板上，则为布尔值 true，否则为 false。

说明

方法；用指定的文本字符串替换系统剪贴板上的内容。

System.showSettings()

可用性

Flash Player 6。

用法

```
System.showSettings([panel])
```

参数

panel 一个可选数字，指定要显示哪个 Flash Player “设置” 面板，如下表所示。

为 <i>panel</i> 传递的值	显示的“设置”面板
无（参数被忽略）或不支持的值	用户上次关闭 Player “设置” 面板时，处于打开状态的那个面板
0	保密性
1	本地存储
2	麦克风
3	摄像头

返回

无。

说明

方法；显示指定的 Flash Player “设置” 面板，使用户可以执行以下操作：

- 允许或拒绝访问摄像头和麦克风
- 指定可供共享对象使用的本地磁盘空间

- 选择默认摄像头和麦克风
- 指定麦克风增益和回声抑制设置

例如，如果您的应用程序需要使用摄像头，则可以通知用户在“Privacy Settings”（隐私设置）面板中选择“Allow”（允许），然后发出 `System.showSettings(0)` 命令。（应确保舞台大小至少为 215 x 138 像素；这是 Flash 显示该面板所需的最小大小。）

另请参见

`Camera.get()`, `Microphone.get()`, `SharedObject.getLocal()`

System.useCodepage

可用性

Flash Player 6。

用法

`System.useCodepage`

说明

属性；一个布尔值，它通知 Flash Player 是使用 Unicode 来解释外部文本文件，还是使用运行播放器的操作系统的传统代码页来解释外部文本文件。`system.useCodepage` 的默认值为 `false`。

- 当该属性设置为 `false` 时，Flash Player 按 Unicode 解释外部文本文件。（保存这些文件时，必须使用 Unicode 对其进行编码。）
- 当该属性设置为 `true` 时，Flash Player 使用运行播放器的操作系统的传统代码页来解释外部文本文件。

对于作为外部文件包含或加载的文本（使用 `#include` 命令，`loadVariables()` 或 `getURL` 动作，或 `LoadVars` 或 `XML` 对象），保存这些文本文件时必须使用 Unicode 对其进行编码，这样 Flash Player 才能将其识别为 Unicode。若要使用 Unicode 对外部文件进行编码，请在支持 Unicode 的应用程序（例如，Windows 2000 上的“记事本”）中保存这些文件。

如果包含或加载了非 Unicode 编码的外部文本文件，则应将 `system.useCodepage` 设置为 `true`。在加载数据的 SWF 文件的第一帧中，在最前面添加以下代码（第一行代码）：

```
system.useCodepage = true;
```

如果有这一行代码，Flash Player 将使用运行 Flash Player 的操作系统的传统代码页来解释外部文本。对于英文 Windows 操作系统，该代码页通常为 CP1252；对于日文操作系统，该代码页通常为 Shift-JIS。如果将 `system.useCodepage` 设置为 `true`，Flash Player 6 和更高版本处理文本的方式与 Flash Player 5 相同。（Flash Player 5 在处理文本时，将所有文本都视为按运行播放器的操作系统的传统代码页编码的文本。）

如果将 `system.useCodepage` 设置为 `true`，应注意您的外部文本文件中使用的字符必须包含在运行播放器的操作系统的传统代码页中，这样才能显示该文本。例如，如果您加载了一个包含中文字符的外部文本文件，这些字符在使用 CP1252 代码页的系统上将无法显示，因为该代码页不包含中文字符。

若要确保所有平台上的用户都能查看您的 SWF 文件中使用的外部文本文件，应使用 Unicode 对所有外部文本文件进行编码，并将 `System.useCodepage` 设置保留为其默认设置 `false`。这样，Flash Player 6 和更高版本将按 Unicode 解释文本。

System.capabilities 对象

可用性

Flash Player 6。

说明

使用 System.capabilities 对象可以确定承载 SWF 文件的系统和播放器的功能。这样，您就可以针对不同的格式对内容进行调整。例如，移动电话的屏幕（黑白，100 x 100 像素）就与 1000 x 1000 像素的彩色 PC 屏幕不同。为了向尽可能多的用户提供适当的内容，可以使用 System.capabilities 对象来确定用户所拥有的设备的类型。然后，您可以指定服务器根据不同的设备功能发送不同的 SWF 文件，或者通知 SWF 文件根据设备的功能改变它的播放方式。

可以使用 GET 或 POST HTTP 方法来发送功能信息。下面是一个设备的服务器字符串示例，该设备不支持 MP3、屏幕为 400 x 200 像素、8 x 4 厘米：

```
"A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&SB=f&DE
B=t&V=WIN%207%2C0%2C0%2C226&M=Macromedia%20Windows&R=1152x864&DP=72&COL=colo
r&AR=1.0&OS=Windows%20XP&L=en&PT=External&AVD=f&LFD=f"
```

System.capabilities 对象的属性摘要

属性	说明	服务器字符串
System.capabilities.avHardwareDisable	只读；指定用户的摄像头和麦克风是否启用。	AVD
System.capabilities.hasAccessibility	指示播放器是否正在支持在 Flash Player 和辅助功能之间进行通讯的系统上运行。	ACC
System.capabilities.hasAudio	指示播放器是否正在具有音频功能的系统上运行。	A
System.capabilities.hasAudioEncoder	指示播放器是否正在能够对音频流（例如，来自麦克风的音频流）进行编码的系统上运行。	AE
System.capabilities.hasEmbeddedVideo	指示播放器是否正在支持嵌入视频的系统上运行。	EV
System.capabilities.hasMP3	指示播放器是否正在具有 MP3 解码器的系统上运行。	MP3
System.capabilities.hasPrinting	指示播放器是否正在支持打印的系统上运行。	PR
System.capabilities.hasScreenBroadcast	指示播放器是否支持开发将通过 Flash Communication Server 运行的屏幕广播应用程序。	SB
System.capabilities.hasScreenPlayback	指示播放器是否支持回放正在通过 Flash Communication Server 运行的屏幕广播应用程序。	SP
System.capabilities.hasStreamingAudio	指示播放器是否可以播放流音频。	SA
System.capabilities.hasStreamingVideo	指示播放器是否可以播放流视频。	SV
System.capabilities.hasVideoEncoder	指示播放器是否能够对视频流（例如，来自 Web 摄像头的视频流）进行编码。	VE

属性	说明	服务器字符串
<code>System.capabilities.isDebugger</code>	指示播放器是官方发布的版本，还是特殊的调试版本。	DEB
<code>System.capabilities.language</code>	指示运行播放器的系统的语言。	L
<code>System.capabilities.localFileReadDisable</code>	只读；指定播放器是否将尝试从用户的硬盘读取内容（包括播放器启动时播放的第一个 SWF 文件）。	LFD
<code>System.capabilities.manufacturer</code>	指示 Flash Player 的制造商。	M
<code>System.capabilities.os</code>	指示承载 Flash Player 的操作系统。	OS
<code>System.capabilities.pixelAspectRatio</code>	指示屏幕的像素高宽比。	AR
<code>System.capabilities.playerType</code>	指示播放器的类型：独立、外部、插件或 ActiveX。	PT
<code>System.capabilities.screenColor</code>	指示屏幕是彩色、灰度、还是黑白的。	COL
<code>System.capabilities.screenDPI</code>	指示屏幕分辨率（每英寸的点数），以像素为单位。	DP
<code>System.capabilities.screenResolutionX</code>	指示屏幕的水平大小。	R
<code>System.capabilities.screenResolutionY</code>	指示屏幕的垂直大小。	R
<code>System.capabilities.serverString</code>	一个 URL 编码的字符串，用于指定各个 <code>System.capabilities</code> 属性的值。	无
<code>System.capabilities.version</code>	包含 Flash Player 版本和平台信息的字符串。	V

System.capabilities.avHardwareDisable

可用性

Flash Player 7。

用法

`System.capabilities.avHardwareDisable`

说明

只读属性；一个布尔值，指定是否启用了用户的摄像头和麦克风。

另请参见

`Camera.get()`, `Microphone.get()`, `System.showSettings()`

System.capabilities.hasAccessibility

可用性

Flash Player 6 版本 65。

用法

```
System.capabilities.hasAccessibility
```

说明

属性；一个布尔值，指示播放器是否正在支持在 Flash Player 和辅助功能之间进行通讯的环境中运行。其服务器字符串为 ACC。

另请参见

[Accessibility.isActive\(\)](#), [Accessibility.updateProperties\(\)](#), [_accProps](#)

System.capabilities.hasAudio

可用性

Flash Player 6。

用法

```
System.capabilities.hasAudio
```

说明

属性；一个布尔值，指示播放器是否正在具有音频功能的系统上运行。其服务器字符串为 A。

System.capabilities.hasAudioEncoder

可用性

Flash Player 6。

用法

```
System.capabilities.hasAudioEncoder
```

说明

属性；一个布尔值，指示播放器是否能够对音频流（例如，来自麦克风的音频流）进行编码。其服务器字符串为 AE。

System.capabilities.hasEmbeddedVideo

可用性

Flash Player 6。

用法

```
System.capabilities.hasEmbeddedVideo
```

说明

属性；一个布尔值，指示播放器是否正在支持嵌入视频的系统上运行。其服务器字符串为 EV。

System.capabilities.hasMP3

可用性

Flash Player 6。

用法

```
System.capabilities.hasMP3
```

说明

属性；一个布尔值，指示播放器是否正在具有 MP3 解码器的系统上运行。其服务器字符串为 MP3。

System.capabilities.hasPrinting

可用性

Flash Player 6。

用法

```
System.capabilities.hasPrinting
```

说明

属性；一个布尔值，指示播放器是否正在支持打印的系统上运行。其服务器字符串为 PR。

System.capabilities.hasScreenBroadcast

可用性

Flash Player 6。

用法

```
System.capabilities.hasScreenBroadcast
```

说明

属性；一个布尔值，指示播放器是否支持开发将通过 Flash Communication Server 运行的屏幕广播应用程序。其服务器字符串为 SB。

System.capabilities.hasScreenPlayback

可用性

Flash Player 6。

用法

```
System.capabilities.hasScreenPlayback
```

说明

属性；一个布尔值，指示播放器是否支持回放正在通过 Flash Communication Server 运行的屏幕广播应用程序。其服务器字符串为 SP。

System.capabilities.hasStreamingAudio

可用性

Flash Player 6。

用法

`System.capabilities.hasStreamingAudio`

说明

属性；一个布尔值，指示播放器是否可以播放流音频。其服务器字符串为 SA。

System.capabilities.hasStreamingVideo

可用性

Flash Player 6。

用法

`System.capabilities.hasStreamingVideo`

说明

属性；一个布尔值，指示播放器是否可以播放流视频。其服务器字符串为 SV。

System.capabilities.hasVideoEncoder

可用性

Flash Player 6。

用法

`System.capabilities.hasVideoEncoder`

说明

属性；一个布尔值，指示播放器是否能够对视频流（例如，来自 Web 摄像头的视频流）进行编码。其服务器字符串为 VE。

System.capabilities.isDebugger

可用性

Flash Player 6。

用法

`System.capabilities.isDebugger`

说明

属性；一个布尔值，指示播放器是正式发布的版本 (`false`)，还是特殊的调试版本 (`true`)。其服务器字符串为 DEB。

System.capabilities.language

可用性

Flash Player 6。

用法

System.capabilities.language

说明

属性；指示运行播放器的系统的语言。此属性可以指定为 ISO 639-1 标准的两个小写字母表示的语言代码加上 ISO 3166 标准的大写字母表示的国家 / 地区代码子标签（后者为可选）。这些代码表示运行播放器的系统的语言。这些语言本身是用英语标签指定的。例如，“fr”指法语。

语言	标签	所支持的国家 / 地区及标签
捷克语	cs	
丹麦语	da	
荷兰语	nl	
英语	en	
芬兰语	fi	
法语	fr	
德语	de	
匈牙利语	hu	
意大利语	it	
日语	ja	
朝鲜语	ko	
挪威语	no	
其它 / 未知	xu	
波兰语	pl	
葡萄牙语	pt	
俄语	ru	
简体中文	zh	中国（简体中文）：zh-CN
西班牙语	es	
瑞典语	sv	
繁体中文	zh	中国台湾地区（繁体中文）：zh-TW
土耳其语	tr	

System.capabilities.localFileReadDisable

可用性

Flash Player 7。

用法

`System.capabilities.localFileReadDisable`

说明

只读属性；一个布尔值，指定 Flash Player 是否尝试从用户的硬盘读取任何内容（包括 Flash Player 启动时播放的第一个 SWF 文件）。

System.capabilities.manufacturer

可用性

Flash Player 6。

用法

`System.capabilities.manufacturer`

说明

属性；一个字符串，指示 Flash Player 的制造商。其格式为 "Macromedia *OSName*"（*OSName* 可能为 "Windows"、"Macintosh"、"Linux" 或 "Other OS Name"）。其服务器字符串为 M。

System.capabilities.os

可用性

Flash Player 6。

用法

`System.capabilities.os`

说明

属性；一个字符串，指示当前操作系统。os 属性可能会返回以下字符串："Windows XP"、"Windows 2000"、"Windows NT"、"Windows 98/ME"、"Windows 95"、"Windows CE"（仅在 Flash Player SDK 中可用，在桌面版本中不可用）、"Linux" 和 "MacOS"。其服务器字符串为 OS。

System.capabilities.pixelAspectRatio

可用性

Flash Player 6。

用法

`System.capabilities.pixelAspectRatio`

说明

属性；指示屏幕像素高宽比的整数。其服务器字符串为 AR。

System.capabilities.playerType

可用性

Flash Player 7。

用法

`System.capabilities.playerType`

说明

属性；一个字符串，指示播放器的类型。此属性可以具有以下值："StandAlone"、"External"、"PlugIn" 或 "ActiveX"。其服务器字符串为 PT。

System.capabilities.screenColor

可用性

Flash Player 6。

用法

`System.capabilities.screenColor`

说明

属性；指示屏幕是彩色的 (color)、灰度的 (gray)、还是黑白的 (bw)。其服务器字符串为 COL。

System.capabilities.screenDPI

可用性

Flash Player 6。

用法

`System.capabilities.screenDPI`

说明

属性；指示屏幕分辨率（每英寸的点数 (dpi)），以像素为单位。其服务器字符串为 DP。

System.capabilities.screenResolutionX

可用性

Flash Player 6。

用法

`System.capabilities.screenResolutionX`

说明

属性；指示屏幕最大水平分辨率的整数。其服务器字符串为 R（它返回屏幕的宽度和高度）。

System.capabilities.screenResolutionY

可用性

Flash Player 6。

用法

`System.capabilities.screenResolutionY`

说明

属性；指示屏幕最大垂直分辨率的整数。其服务器字符串为 `R`（它返回屏幕的宽度和高度）。

System.capabilities.serverString

可用性

Flash Player 6。

用法

`System.capabilities.serverString`

说明

属性；一个 URL 编码的字符串，用于指定各个 `System.capabilities` 属性的值，如下例所示：

```
A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&SB=f&DE
B=t&V=WIN%207%2C0%2C0%2C226&M=Macromedia%20Windows&R=1152x864&DP=72&COL=color&A
R=1.0&OS=Windows%20XP&L=en&PT=External&AVD=f&LFD=f
```

System.capabilities.version

可用性

Flash Player 6。

用法

`System.capabilities.version`

说明

属性；一个包含 Flash Player 平台和版本信息的字符串，例如，"`WIN 7,0,0,231`"。其服务器字符串为 `V`。

System.security 对象

可用性

Flash Player 6。

说明

此对象包含的方法指定不同域中的 SWF 文件如何相互通讯。

System.security 对象的方法摘要

方法	说明
<code>System.security.allowDomain()</code>	允许指定的域中的 SWF 文件访问调用此方法的 SWF 文件中的对象和变量，或与调用此方法的 SWF 文件位于同一个域中的任何其它 SWF 文件中的对象和变量。
<code>System.security.allowInsecureDomain()</code>	允许指定的域中的 SWF 文件访问调用此方法的 SWF 文件中的对象和变量，该文件是使用 HTTPS 协议进行承载的。

System.security.allowDomain()

可用性

Flash Player 6 ；行为在 Flash Player 7 中发生了变化。

用法

```
System.security.allowDomain("domain1", "domain2, ... domainN")
```

参数

domain1, domain2, ... domainN 指定域的字符串，这些域可以访问包含 `System.Security.allowDomain()` 调用的文件中的对象和变量。可以按照以下格式指定域：

- "domain.com"
- "http://domain.com"
- "http://IPAddress"

说明

方法 ；允许指定的域中的 SWF 文件访问调用此方法的 SWF 文件中的对象和变量，或与调用此方法的 SWF 文件位于同一个域中的任何其它 SWF 文件中的对象和变量。

在 Flash Player 7 或更高版本中播放的文件中，传递的参数必须遵守完全域命名规则。例如，若要允许位于 `www.domain.com` 或 `store.domain.com` 的 SWF 文件进行访问，则必须传递这两个域名：

```
// 对于 Flash Player 6
System.security.allowDomain("domain.com");
// 允许运行于 Flash Player 7 或更高版本中的 SWF 文件
// 进行访问的对应命令
System.security.allowDomain("www.domain.com"."store.domain.com");
```

另外，对于运行于 Flash Player 7 或更高版本中的文件，不能使用此方法让使用安全的协议 (HTTPS) 承载的 SWF 文件允许来自使用不安全的协议承载的 SWF 文件的访问；您必须改用 `System.security.allowInsecureDomain()`。

示例

位于 `www.macromedia.com/MovieA.swf` 的 SWF 文件包含以下代码行。

```
System.security.allowDomain("www.shockwave.com");
loadMovie("http://www.shockwave.com/MovieB.swf", _root.my_mc);
```

因为 MovieA 包含 `allowDomain()` 命令，所以 MovieB 可以访问 MovieA 中的对象和变量。如果 MovieA 中没有该命令，Flash 安全实施机制将不允许 MovieA 访问 MovieB 的对象和变量。

System.security.allowInsecureDomain()

可用性

Flash Player 7。

用法

```
System.Security.allowInsecureDomain("domain")
```

参数

domain 一个完全域名，例如 “www.myDomainName.com” 或 “store.myDomainName.com”。

返回

无。

说明

方法；允许指定的域中的 SWF 文件访问调用此方法的 SWF 文件中的对象和变量，该文件是使用 HTTPS 协议进行承载的。

默认情况下，使用 HTTPS 协议承载的 SWF 文件只能被其它使用 HTTPS 协议承载的 SWF 文件访问。这种实现保持了 HTTPS 协议所提供的完整性。

因为此方法损害了 HTTPS 安全性，所以 Macromedia 不推荐使用此方法覆盖默认的行为。但是有时您可能需要这样做。例如，如果您必须允许为 Flash Player 6 发布的 HTTP 文件访问为 Flash Player 7 或更高版本发布的 HTTPS 文件。

为 Flash Player 6 发布的 SWF 文件可以使用 [System.security.allowDomain\(\)](#) 允许 HTTP 对 HTTPS 的访问。但是，由于 Flash Player 7 中安全性的实现方式不同，您必须使用 [System.Security.allowInsecureDomain\(\)](#) 允许在为 Flash Player 7 或更高版本发布的 SWF 文件中进行这种访问。

示例

在本例中，在安全的域中承载数学测试，以便只有注册的学生才能进行访问。您还开发了一些 SWF 文件用于说明某些概念，您将这些文件放置在不安全的域中。您希望学生能够从包含有关概念信息的 SWF 文件访问测试。

```
// 此 SWF 文件位于 https://myEducationSite.somewhere.com/mathTest.swf
// 概念文件位于 http://myEducationSite.somewhere.com
System.Security.allowInsecureDomain("myEducationSite.somewhere.com")
```

另请参见

[System.security.allowDomain\(\)](#), [System.exactSettings](#)

targetPath

可用性

Flash Player 5。

用法

```
targetPath(movieClipObject)
```

参数

movieClipObject 对要获取其目标路径的影片剪辑的引用（例如，`_root` 或 `_parent`）。

返回

包含指定影片剪辑的目标路径的字符串。

说明

函数；返回包含 *movieClipObject* 的目标路径的字符串。此目标路径以点记号表示形式返回。若要获取以斜杠记号表示的目标路径，请使用 `_target` 属性。

示例

此示例在加载影片剪辑时，立即显示该影片剪辑的目标路径。

```
onClipEvent (load) {  
    trace(targetPath(this));  
}
```

另请参见

[eval\(\)](#)

tellTarget

可用性

Flash Player 3。（在 Flash 5 中不鼓励使用；建议使用点记号表示法和 `with` 动作。）

用法

```
tellTarget("target") {  
    statement(s);  
}
```

参数

target 一个字符串，指定要控制的时间轴的目标路径。

statement(s) 条件计算结果为 `true` 时要执行的指令。

返回

无。

说明

不鼓励使用的动作；将在 *statements* 参数中指定的指令应用到在 *target* 参数中指定的时间轴。`tellTarget` 动作对导航控制很有帮助。可以将 `tellTarget` 分配给停止或开始舞台上其它地方的影片剪辑的按钮。还可以使影片剪辑转到此剪辑的特定帧。例如，可以将 `tellTarget` 分配给停止或开始舞台上影片剪辑的按钮，或者提示影片剪辑跳至特定帧的按钮。

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

示例

此 `tellTarget` 语句控制主时间轴上的影片剪辑实例 `ball`。`ball` 实例的第 1 帧为空白而且有一个 `stop()` 动作，所以它在舞台上不可见。单击具有下列动作的按钮时，`tellTarget` 通知 `ball` 中的播放头转至第 2 帧（动画开始处）。

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

下面的示例使用点记号表示法达到同样的结果。

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

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

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

另请参见

[with](#)

TextField 类

可用性

Flash Player 6。

说明

SWF 文件中的所有动态文本字段和输入文本字段都是 `TextField` 类的实例。可以在属性检查器中为文本字段指定实例名称，并且可以在动作脚本中使用 `TextField` 类的方法和属性对文本字段进行操作。`TextField` 的实例名称显示在“影片管理器”中及“动作”面板的“插入目标路径”对话框中。

`TextField` 类继承自 [Object](#) 类。

若要动态创建文本字段，可以使用 `MovieClip.createTextField()`。

TextField 类的方法摘要

方法	说明
<code>TextField.addListener()</code>	注册一个对象，以便在调用 <code>onChanged</code> 和 <code>onScroller</code> 事件处理函数时接收通知。
<code>TextField.getFontList()</code>	返回播放器的主机系统上字体名称的数组。
<code>TextField.getDepth()</code>	返回文本字段的深度。
<code>TextField.getNewTextFormat()</code>	获取分配给新插入文本的默认文本格式。
<code>TextField.getTextFormat()</code>	返回包含文本字段中一些或全部文本的格式信息的 <code>TextFormat</code> 对象。
<code>TextField.removeListener()</code>	删除侦听器对象。
<code>TextField.removeTextField()</code>	删除使用 <code>MovieClip.createTextField()</code> 创建的文本字段。
<code>TextField.replaceSel()</code>	替换当前的所选内容。
<code>TextField.setNewTextFormat()</code>	为由用户或方法插入的文本设置 <code>TextFormat</code> 对象。
<code>TextField.setTextFormat()</code>	为文本字段中指定范围内的文本设置一个 <code>TextFormat</code> 对象。

TextField 类的属性摘要

属性	说明
<code>TextField._alpha</code>	文本字段实例的透明度值。
<code>TextField.autoSize</code>	控制文本字段的自动对齐和大小调整。
<code>TextField.background</code>	指示文本字段是否具有背景填充。
<code>TextField.backgroundColor</code>	指示背景填充的颜色。
<code>TextField.border</code>	指示文本字段是否具有边框。
<code>TextField.borderColor</code>	指示边框的颜色。
<code>TextField.bottomScroll</code>	文本字段中最下面的可见行。只读。
<code>TextField.embedFonts</code>	指示文本字段是使用嵌入字体轮廓还是使用设备字体。
<code>TextField._height</code>	文本字段实例的高度，以像素为单位。它只影响文本字段的边框，不影响边框的粗细和文本字体大小。
<code>TextField._highquality</code>	指示 SWF 文件的呈现品质。
<code>TextField.hscroll</code>	指示文本字段的水平滚动值。
<code>TextField.html</code>	指示文本字段的当前最大滚动位置。
<code>TextField.htmlText</code>	包含文本字段内容的 HTML 表示形式。
<code>TextField.length</code>	文本字段中的字符数。只读。
<code>TextField.maxChars</code>	文本字段最多可容纳的字符数。
<code>TextField.maxhscroll</code>	<code>TextField.hscroll</code> 的最大值。只读。
<code>TextField.maxscroll</code>	<code>TextField.scroll</code> 的最大值。只读。

属性	说明
<code>TextField.menu</code>	将 <code>ContextMenu</code> 对象与文本字段关联。
<code>TextField.mouseWheelEnabled</code>	指示当鼠标指针停在文本字段上且用户滚动鼠标滚轮时，Flash Player 是否应自动滚动多行文本字段。
<code>TextField.multiline</code>	指示文本字段是否包含多行。
<code>TextField._name</code>	文本字段实例的实例名称。
<code>TextField._parent</code>	对此实例的父级实例的引用；其类型为 <code>Button</code> 或 <code>MovieClip</code> 。
<code>TextField.password</code>	指示文本字段是否隐藏输入字符。
<code>TextField._quality</code>	指示 SWF 文件的呈现品质。
<code>TextField.restrict</code>	用户可输入文本字段的字符集。
<code>TextField._rotation</code>	文本字段实例的旋转度数。
<code>TextField.scroll</code>	指示文本字段的当前滚动位置。
<code>TextField.selectable</code>	指示文本字段是否可选。
<code>TextField._soundbuftime</code>	在声音进入流之前，必须预先缓冲的声音的时间量。
<code>TextField.tabEnabled</code>	指示影片剪辑是否包括在 Tab 键的自动排序中。
<code>TextField.tabIndex</code>	指示对象的 Tab 键顺序。
<code>TextField._target</code>	指定文本字段实例的目标路径。只读。
<code>TextField.text</code>	文本字段中的当前文本。
<code>TextField.textColor</code>	文本字段中当前文本的颜色。
<code>TextField.textHeight</code>	文本字段边框的高度。
<code>TextField.textWidth</code>	文本字段边框的宽度。
<code>TextField.type</code>	指示文本字段是输入文本字段还是动态文本字段。
<code>TextField._url</code>	创建文本字段实例的 SWF 文件的 URL。只读。
<code>TextField.variable</code>	与文本字段关联的变量名。
<code>TextField._visible</code>	确定文本字段实例是隐藏还是可见的布尔值。
<code>TextField._width</code>	文本字段实例的宽度，以像素为单位。它只影响文本字段的边框，不影响边框的粗细和文本字体大小。
<code>TextField.wordWrap</code>	指示文本字段是否自动换行。
<code>TextField._x</code>	文本字段实例的 x 坐标
<code>TextField._xmouse</code>	指针相对于文本字段实例的 x 坐标。只读。
<code>TextField._xscale</code>	指定水平缩放文本字段实例的百分比的值。
<code>TextField._y</code>	文本字段实例的 y 坐标。
<code>TextField._ymouse</code>	指针相对于文本字段实例的 y 坐标。只读。
<code>TextField._yscale</code>	指定垂直缩放文本字段实例的百分比的值。

TextField 类的事件处理函数摘要

事件处理函数	说明
<code>TextField.onChangeed</code>	在文本字段更改时调用。
<code>TextField.onKillFocus</code>	在文本字段失去焦点时调用。
<code>TextField.onScroller</code>	在文本字段滚动属性之一发生改变时调用。
<code>TextField.onSetFocus</code>	在文本字段接收焦点时调用。

TextField 类的侦听器摘要

方法	说明
<code>TextField.onChangeed</code>	在文本字段更改时获得通知。
<code>TextField.onScroller</code>	在文本字段的 <code>scroll</code> 或 <code>maxscroll</code> 属性更改时获得通知。

TextField.addListener()

可用性

Flash Player 6。

用法

```
my_txt.addListener(listener)
```

参数

listener 一个具有 `onChangeed` 或 `onScroller` 事件处理函数的对象。

返回

无。

说明

方法；注册一个对象，以便在调用 `onChangeed` 和 `onScroller` 事件处理函数时接收通知。当文本字段更改或滚动时，首先将调用 `TextField.onChangeed` 和 `TextField.onScroller` 事件处理函数，然后调用任何注册为侦听器的对象的 `onChangeed` 和 `onScroller` 事件处理函数。可将多个对象注册为侦听器。

若要从文本字段删除侦听器对象，可以调用 `TextField.removeListener()`。

事件源将把对该文本字段实例的一个引用作为参数传递给 `onScroller` 和 `onChangeed` 处理函数。您可以通过将参数放入事件处理函数方法来捕获此数据。例如，以下代码使用 `txt` 作为传递给 `onScroller` 事件处理函数的参数。然后，在一条 `trace` 语句中使用该参数将文本字段的实例名称发送到“输出”面板。

```
myTextField.onScroller = function (txt) {  
    trace (txt._name + " changed");  
};
```

示例

下面的示例为输入文本字段 `myText` 定义一个 `onChange` 处理函数。然后定义一个新的侦听器对象，`myListener`，并为该对象定义一个 `onChanged` 处理函数。当文本字段 `myText` 发生更改时，将调用此处理函数。最后一行代码调用 `TextField.addListener` 向文本字段 `myText` 注册侦听器对象 `myListener`，这样，在 `myText` 更改时将通知该侦听器对象。

```
myText.onChange = function (txt) {  
    trace (txt._name + " changed");  
};  
myListener = new Object();  
myListener.onChanged = function (txt) {  
    trace(txt._name + " changed and notified myListener");  
};  
  
myText.addListener(myListener);
```

另请参见

[TextField.onChangeed](#), [TextField.onScroller](#), [TextField.removeListener\(\)](#)

TextField._alpha

可用性

Flash Player 6。

用法

my_txt._alpha

说明

属性；设置或获取由 *my_txt* 指定的文本字段的 Alpha 透明度值。有效值为 0（完全透明）到 100（完全不透明）。默认值为 100。

示例

下列代码在单击按钮时，将名为 `text1_txt` 的文本字段的 `_alpha` 属性设置为 30%：

```
on (release) {  
    text1_txt._alpha = 30;  
}
```

另请参见

[Button._alpha](#), [MovieClip._alpha](#)

TextField.autoSize

可用性

Flash Player 6。

用法

my_txt.autoSize

说明

属性；控制文本字段的自动大小调整和对齐。autoSize 可接受的值是 "none"（默认值）、"left"、"right" 和 "center"。设置 autoSize 属性时，true 与 "left" 同义，而 false 与 "none" 同义。

autoSize、multiline 和 wordWrap 的值确定文本字段是否扩展或收缩到左边、右边或底边。您可以使用以下代码并为 autoSize、multiline 和 wordWrap 输入不同的值，观察当这些值更改时字段如何调整大小。

```
createTextField("my_txt", 1, 0, 0, 200, 20);
with (my_txt) {
    border = true;
    borderColor = 0x000000;
    multiline = false;
    wordWrap = false;
    autoSize = "none";
    text = "Here is a whole bunch of text that won't fit in the field ";
}
```

示例

下面的语句将文本字段 my_txt 的 autosize 属性设置为 "center"。

```
my_txt.autosize = "center";
```

TextField.background

可用性

Flash Player 6。

用法

my_txt.background

说明

属性；如果为 true，则文本字段具有背景填充。如果为 false，则文本字段没有背景填充。

TextField.backgroundColor

可用性

Flash Player 6。

用法

```
my_txt.backgroundColor
```

说明

属性；文本字段背景的颜色。默认值为 0xFFFFFF（白色）。即使当前没有背景（只有当文本字段有边框时，背景颜色才可见），也可获取或设置此属性。

另请参见

[TextField.background](#)

TextField.border

可用性

Flash Player 6。

用法

```
my_txt.border
```

说明

属性；如果为 true，则文本字段具有边框。如果为 false，则文本字段没有边框。

TextField.borderColor

可用性

Flash Player 6。

用法

```
my_txt.borderColor
```

说明

属性；文本字段边框的颜色，默认值为 0x000000（黑色）。即使当前没有边框，也可获取或设置此属性。

另请参见

[TextField.border](#)

TextField.bottomScroll

可用性

Flash Player 6。

用法

```
my_txt.bottomScroll
```

说明

属性（只读）；指示 *my_txt* 中当前最底端可见行的整数（从 1 开始的索引）。可将文本字段看作文本块上的一个“窗口”。[TextField.scroll](#) 属性是此窗口中最顶端可见行的索引（从 1 开始）。

在该文本字段中，第 [TextField.scroll](#) 行和第 [TextField.bottomScroll](#) 行之间的所有文本当前都可见。

TextField.condenseWhite

可用性

Flash Player 6。

用法

```
my_txt.condenseWhite
```

说明

属性；一个布尔值，指定当 HTML 文本字段在浏览器中呈现时是否删除字段中的额外空白（空格、换行符等）。默认值为 `false`。

如果将此值设置为 `true`，则必须使用标准 HTML 命令（如 `
` 和 `<P>`）将换行符放在文本字段中。

如果 *my_txt.html* 为 `false`，则忽略此属性。

另请参见

[TextField.html](#)

TextField.embedFonts

可用性

Flash Player 6。

用法

```
my_txt.embedFonts
```

说明

属性；一个布尔值，当此值为 `true` 时，使用嵌入字体轮廓呈现文本字段。如果为 `false`，则使用设备字体呈现文本字段。

TextField.getDepth()

可用性

Flash Player 6。

用法

```
my_txt.getDepth()
```

参数

无。

返回

一个整数。

说明

方法；返回文本字段的深度。

TextField.getFontList()

可用性

Flash Player 6。

用法

```
TextField.getFontList()
```

参数

无。

返回

一个数组。

说明

方法；全局 [TextField](#) 类的一个静态方法。调用此方法时，不需要指定特定的文本字段（例如 my_txt）。此方法返回播放器的主机系统上字体名称的数组。（而不返回当前加载的 SWF 文件中所有字体的名称。）这些名称为 string 类型。

示例

下面的代码显示 getFontList() 返回的字体列表。

```
font_array = TextField.getFontList();
for( i in font_array){
    trace(font_array[i]);
}
```

TextField.getNewTextFormat()

可用性

Flash Player 6。

用法

```
my_txt.getNewTextFormat()
```

参数

无。

返回

TextFormat 对象。

说明

方法；返回一个 TextFormat 对象，该对象包含此文本字段的文本格式对象的副本。文本格式对象是新插入文本（例如，使用 `replaceSel()` 方法插入的文本或由用户输入的文本）接收到的格式。调用 `getNewTextFormat()` 时，返回的 TextFormat 对象的所有属性均已定义。所有属性都不为 `null`。

TextField.getTextFormat()

可用性

Flash Player 6。

用法

```
my_txt.getTextFormat()  
my_txt.getTextFormat(index)  
my_txt.getTextFormat(beginIndex, endIndex)
```

参数

index 一个整数，指定字符串中的某个字符。
beginIndex、*endIndex* 整数，指定 *my_txt* 内文本块的开始和结束位置。

返回

一个对象。

说明

方法；用法 1：返回一个 TextFormat 对象，该对象包含文本字段中所有文本的格式设置信息。在结果 TextFormat 对象中只设置文本字段中所有文本共有的属性。所有混合型属性（意味着它在文本中的不同位置有不同的值）的值都设置为 `null`。

用法 2：返回一个 TextFormat 对象，该对象包含 *index* 处文本字段的文本格式的副本。

用法 3：返回一个 TextFormat 对象，该对象包含从 *beginIndex* 到 *endIndex* 的文本范围内的格式设置信息。

另请参见

[TextField.getNewTextFormat\(\)](#), [TextField.setNewTextFormat\(\)](#), [TextField.setTextFormat\(\)](#)

TextField._height

可用性

Flash Player 6。

用法

my_txt._height

说明

属性；文本字段的高度，以像素为单位。

示例

下面的代码示例设置文本字段的高度和宽度。

```
my_txt._width = 200;  
my_txt._height = 200;
```

TextField._highquality

可用性

Flash Player 6。

用法

my_txt._highquality

说明

属性（全局）；指定应用于当前 SWF 文件的锯齿消除级别。指定 2（最高品质），则应用高品质，位图平滑处理始终打开。指定 1（高品质），则应用锯齿消除功能；如果 SWF 文件不包含动画，将对位图进行平滑处理。指定 0（低品质），则不消除锯齿。

另请参见

[_quality](#)

TextField.hscroll

可用性

Flash Player 6。

用法

```
my_txt.hscroll
```

返回

一个整数。

说明

属性；指示当前水平滚动位置。如果 `hscroll` 属性为 0，则不能水平滚动文本。

有关滚动文本的更多信息，请参见第 135 页的“创建滚动文本”。

示例

下面的示例水平滚动文本。

```
on (release) {  
    my_txt.hscroll += 1;  
}
```

另请参见

[TextField.maxhscroll](#), [TextField.scroll](#)

TextField.html

可用性

Flash Player 6。

用法

```
my_txt.html
```

说明

属性；指示文本字段是否包含 HTML 表示形式的标记。如果 `html` 属性为 `true`，则该文本字段为 HTML 文本字段。如果 `html` 为 `false`，则该文本字段为非 HTML 文本字段。

另请参见

[TextField.htmlText](#)

TextField.htmlText

可用性

Flash Player 6。

用法

```
my_txt.htmlText
```

说明

属性；如果该文本字段为 HTML 文本字段，则此属性包含该文本字段内容的 HTML 表示形式。如果该文本字段不是 HTML 文本字段，则其行为与 `text` 属性相同。可以通过以下方法指示某个文本字段是 HTML 文本字段：在属性检查器中指示；或通过将该文本字段的 `html` 属性设置为 `true` 进行指示。

示例

在下面的示例中，文本字段 `text2` 中的文本以粗体呈现。

```
text2.html = true;  
text2.htmlText = "<b> this is bold text </b>";
```

另请参见

[TextField.html](#)

TextField.length

可用性

Flash Player 6。

用法

```
my_txt.length
```

返回

一个数字。

说明

属性（只读）；指示文本字段中的字符数。此属性的返回值与 `text.length` 相同，但是它更快。如 `Tab`（“\t”）之类的字符视为一个字符。

TextField.maxChars

可用性

Flash Player 6。

用法

```
my_txt.maxChars
```

说明

属性；指示此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` 允许的字符数多；`maxChars` 属性只是指示用户可以输入多少文本。如果此属性的值为 `null`，则对用户可以输入的文本量没有限制。

TextField.maxhscroll

可用性

Flash Player 6。

用法

```
my_txt.maxhscroll
```

说明

属性（只读）；指示 `TextField.hscroll` 的最大值。

TextField.maxscroll

可用性

Flash Player 6。

用法

```
TextField.maxscroll
```

说明

属性（只读）；指示 `TextField.scroll` 的最大值。

有关滚动文本的更多信息，请参见第 135 页的“创建滚动文本”。

TextField.menu

可用性

Flash Player 7。

用法

```
my_txt.menu = contextMenu
```

参数

contextMenu 一个 ContextMenu 对象。

说明

属性；将 ContextMenu 对象 *contextMenu* 与文本字段 *my_txt* 相关联。ContextMenu 类用于修改当用户在 Flash Player 中右击 (Windows) 或按住 Control 键单击 (Macintosh) 时显示的上下文菜单。

此属性仅在可选（可编辑）文本字段上起作用；在非可选文本字段上，它不起作用。

示例

下面的示例将 ContextMenu 对象 *menu_cm* 分配给文本字段 *news_txt*。ContextMenu 对象包含一个标记为“Print”的自定义菜单项，该项带有一个名为 *doPrint()*、执行打印操作的（不显示出来）的关联回调处理函数：

```
var menu_cm = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Print...", doPrint));
function doPrint(menu, obj) {
    // 此处为打印代码
}
news_txt.menu = menu_cm;
```

另请参见

[Button.menu](#), [ContextMenu](#) 类, [ContextMenuItem](#) 类, [MovieClip.menu](#)

TextField.mouseWheelEnabled

可用性

Flash Player 7。

用法

```
my_txt.mouseWheelEnabled
```

说明

属性；一个布尔值，指示当鼠标指针位于文本字段上且用户滚动鼠标滚轮时，Flash Player 是否应自动滚动多行文本字段。默认情况下，此值为 *true*。如果您想让文本字段在用户滚动鼠标滚轮时不随之滚动，或要实施您自己的文本字段滚动方式，可以使用此属性。

另请参见

[Mouse.onMouseWheel](#)

TextField.multiline

可用性

Flash Player 6。

用法

```
my_txt.multiline
```

说明

属性；指示文本字段是否为多行文本字段。如果此值为 `true`，则该文本字段为多行文本字段；如果此值为 `false`，则该文本字段为单行文本字段。

TextField._name

可用性

Flash Player 6。

用法

```
my_txt._name
```

说明

属性；由 *my_txt* 指定的文本字段的实例名称。

TextField.onChanged

可用性

Flash Player 6。

用法

```
my_txt.onChanged = function(){  
    // 此处是您的语句  
}
```

参数

无。

返回

文本字段的实例名称。

说明

事件处理函数；在文本字段的内容更改时调用。默认情况下它是未定义的；可以在脚本中定义它。

对该文本字段实例的一个引用将被作为参数传递给 `onChanged` 处理函数。您可以通过将参数放入事件处理函数方法来捕获此数据。例如，以下代码使用 `txt` 作为传递给 `onChanged` 事件处理函数的参数。然后，在 `trace()` 语句中使用该参数将文本字段的实例名称发送到“输出”面板。

```
myTextField.onChanged = function (txt) {  
    trace (txt._name + " changed");  
};
```

TextField.onKillFocus

可用性

Flash Player 6。

用法

```
my_txt.onKillFocus = function(newFocus){  
    // 此处是您的语句  
}
```

参数

newFocus 要接收焦点的对象。

返回

无。

说明

事件处理函数；在文本字段失去键盘焦点时调用。onKillFocus 方法接收一个参数 *newFocus*，该参数是一个对象，表示要接收焦点的新对象。如果没有对象接收焦点，则 *newFocus* 将包含值 null。

TextField.onScroller

可用性

Flash Player 6。

用法

```
my_txt.onScroller = function(textFieldInstance){  
    // 此处是您的语句  
}
```

参数

textFieldInstance 指向滚动位置被更改的 TextField 对象的引用。

返回

无。

说明

事件处理函数；在文本字段滚动属性中的某一个发生更改时调用。

对该文本字段实例的一个引用将被作为参数传递给 onScroller 处理函数。您可以通过将参数放入事件处理函数方法来捕获此数据。例如，以下代码使用 txt 作为传递给 onScroller 事件处理函数的参数。然后，在 trace() 语句中使用该参数将文本字段的实例名称发送到“输出”面板。

```
myTextField.onScroller = function (txt) {  
    trace (txt._name + " scrolled");  
};
```

另请参见

[TextField.hscroll](#), [TextField.maxhscroll](#), [TextField.maxscroll](#), [TextField.scroll](#)

TextField.onSetFocus

可用性

Flash Player 6。

用法

```
my_txt.onSetFocus = function(oldFocus){  
    // 此处是您的语句  
}
```

参数

oldFocus 将失去焦点的对象。

返回

无。

说明

事件处理函数；在文本字段接收键盘焦点时调用。*oldFocus* 参数是失去焦点的对象。例如，如果用户按下 Tab 键将输入焦点从某个按钮移动到某个文本字段，则 *oldFocus* 包含该文本字段实例。

如果以前没有具有焦点的对象，则 *oldFocus* 包含 null 值。

TextField._parent

可用性

Flash Player 6。

用法

```
my_txt._parent.property  
_parent.property
```

说明

属性；指向包含当前文本字段或对象的影片剪辑或对象的引用。当前对象是包含引用 *_parent* 的动作脚本代码的对象。

使用 *_parent* 可以指定一个相对路径，该路径指向当前文本字段之上的影片剪辑或对象。可以使用 *_parent* 在显示列表中攀升多个级别，如下所示：

```
_parent._parent._alpha = 20;
```

另请参见

[Button._parent](#), [MovieClip._parent](#), [_root](#), [targetPath](#)

TextField.password

可用性

Flash Player 6。

用法

```
my_txt.password
```

说明

属性；如果 `password` 的值为 `true`，则该文本字段为密码文本字段，它将隐藏输入字符。如果为 `false`，则该文本字段不是密码文本字段。

TextField._quality

可用性

Flash Player 6。

用法

```
my_txt._quality
```

说明

属性（全局）；设置或获取用于 SWF 文件的呈现品质。设备字体始终是带有锯齿的，因此不受 `_quality` 属性的影响。

注意：虽然可以针对 `TextField` 对象指定此属性，但它实际上是一个全局属性，您可以简单地使用 `_quality` 指定其值。有关更多信息，请参见 [_quality](#)。

TextField.removeListener()

可用性

Flash Player 6。

用法

```
my_txt.removeListener(listener)
```

参数

`listener` 不再从 `TextField.onChanged` 或 `TextField.onScroller` 接收通知的对象。

返回

如果成功删除了 `listener`，则该方法的返回值为 `true`。如果未能成功删除 `listener`（例如，如果 `listener` 不在该 `TextField` 对象的侦听器列表上），则该方法的返回值为 `false`。

说明

方法；删除以前使用 `TextField.addListener()` 向文本字段实例注册的侦听器对象。

TextField.removeTextField()

可用性

Flash Player 6。

用法

```
my_txt.removeTextField()
```

说明

方法；删除由 *my_txt* 指定的文本字段。只能对使用 `MovieClip.createTextField()` 创建的文本字段执行此操作。当调用此方法时，将删除文本字段。此方法与 `MovieClip.removeMovieClip()` 相似。

TextField.replaceSel()

可用性

Flash Player 6。

用法

```
my_txt.replaceSel(text)
```

参数

text 一个字符串。

返回

无。

说明

方法；使用 *text* 参数的内容替换当前所选内容。使用当前默认字符格式和默认段落格式，在当前所选内容的所在位置插入文本。即使该文本字段是 HTML 文本字段，也不会将这些文本视为 HTML。

可以使用 `replaceSel()` 方法插入并删除文本，而不破坏其余文本的字符和段落格式。

在发布此命令之前，您必须使用 `Selection.setFocus()` 将焦点放置在字段上。

另请参见

[Selection.setFocus\(\)](#)

TextField.replaceText()

可用性

Flash Player 7。

用法

```
my_txt.replaceText(beginIndex, endIndex, text)
```

说明

方法；在指定的文本字段中，使用 *text* 参数的内容替换由 *beginIndex* 和 *endIndex* 参数所指定的范围内的字符。

TextField.restrict

可用性

Flash Player 6。

用法

```
my_txt.restrict
```

说明

属性；指示用户可输入到文本字段中的字符集。如果 `restrict` 属性的值为 `null`，则可以输入任何字符。如果 `restrict` 属性的值为空字符串，则任何字符都不能输入。如果 `restrict` 属性的值为一个字符串，则只能在文本字段中输入该字符串中的字符。从左向右扫描该字符串。范围可以使用短划线 (-) 进行指定。它只限制用户交互；脚本可将任何文本放入文本字段中。此属性不与属性检查器中的“嵌入字体轮廓”复选框同步。

如果此字符串以 ^ 开头，则先接受所有字符，然后从接受字符集中排除字符串中 ^ 之后的字符。如果此字符串不以 ^ 开头，则最初不接受任何字符，然后将字符串中的字符包括在接受字符集中。

示例

下面的示例仅允许在文本字段中输入大写字母、空格和数字：

```
my_txt.restrict = "A-Z 0-9";
```

下面的示例包含除小写字母之外的所有字符：

```
my_txt.restrict = "^a-z";
```

可以使用反斜杠输入 ^ 或 - 的本义。认可的反斜杠序列为 \-、\^ 或 \\。反斜杠在字符串中必须是一个本义字符，因此在动作脚本中指定时必须使用两个反斜杠。例如，下面的代码只包含短划线 (-) 和 插入符号 (^)：

```
my_txt.restrict = "\\-\\^";
```

可在字符串中的任何地方使用 ^，以在包含字符与排除字符之间进行切换。下面的代码只包含除大写字母 Q 之外的大写字母：

```
my_txt.restrict = "A-Z^Q";
```

可以使用 \u 转义序列构造 `restrict` 字符串。下面的代码只包含从 ASCII 32（空格）到 ASCII 126（代字号）之间的字符。

```
my_txt.restrict = "\u0020-\u007E";
```

TextField._rotation

可用性

Flash Player 6。

用法

```
my_txt._rotation
```

说明

属性；文本字段从其原始方向的旋转（以度为单位）。从 0 到 180 的值表示顺时针方向的旋转；从 0 到 -180 的值表示逆时针方向的旋转。对于此范围之外的值，可以通过加上或减去 360 获得该范围内的值。例如，语句 `my_txt._rotation = 450` 与 `my_txt._rotation = 90` 相同。

另请参见

[Button._rotation](#), [MovieClip._rotation](#)

TextField.scroll

可用性

Flash Player 6。

用法

`my_txt.scroll`

说明

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，或者创建滚动文本字段。可以获取和修改此属性。

有关滚动文本的更多信息，请参见[第 135 页的“创建滚动文本”](#)。

示例

下面是附加到用于滚动 `my_txt` 文本字段的“向上”按钮上的代码。

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

另请参见

[TextField.hscroll](#), [TextField.maxscroll](#)

TextField.selectable

可用性

Flash Player 6。

用法

`my_txt.selectable`

说明

属性；一个布尔值，指示文本字段是否可选（可编辑）。值为 `true` 表示文本可选。

TextField.setNewTextFormat()

可用性

Flash Player 6。

用法

```
my_txt.setNewTextFormat(textFormat)
```

参数

textFormat 一个 TextFormat 对象。

返回

无。

说明

方法；为新插入的文本（例如文本字段中使用 `replaceSel()` 方法插入的文本或由用户输入的文本）设置一个 TextFormat 对象。每个文本字段都有一个新文本格式。插入文本时，将新文本格式分配给新文本。

文本格式在新 TextFormat 对象中设置。它包含字符和段落格式设置信息。字符格式设置信息描述单个字符的外观；例如，字体名称、磅值、颜色和关联 URL。段落格式设置信息描述段落的外观；例如，左边距、右边距、首行缩进和左对齐、右对齐及居中对齐。

另请参见

```
TextField.getNewTextFormat(), TextField.getTextFormat(),  
TextField.setTextFormat()
```

TextField.setTextFormat()

可用性

Flash Player 6。

用法

```
my_txt.setTextFormat (textFormat)  
my_txt.setTextFormat (index, textFormat)  
my_txt.setTextFormat (beginIndex, endIndex, textFormat)
```

参数

textFormat 一个包含字符和段落格式设置信息的 TextFormat 对象。

index 整数，指定 my_txt 内的一个字符。

beginIndex 一个整数。

endIndex 一个整数，指定所需文本范围后的第一个字符。

返回

无。

说明

方法；为文本字段中指定范围内的文本设置一个 `TextFormat` 对象。可以为文本字段中的每个字符分配一种文本格式。测试段落的第一个字符的文本格式，以执行整个段落的段落格式设置。`setTextFormat()` 方法更改应用于文本字段中单个字符、字符组或整体文本的文本格式。

文本格式在新 `TextFormat` 对象中设置。它包含字符和段落格式设置信息。字符格式设置信息描述单个字符的外观，例如，字体名称、磅值、颜色及关联 URL。段落格式设置信息描述段落的外观，例如，左边距、右边距、首行缩进和左对齐、右对齐及居中对齐。

用法 1：将 `textFormat` 的属性应用于文本字段中的所有文本。

用法 2：将 `textFormat` 的属性应用于 `index` 处的字符。

用法 3：将 `textFormat` 参数的属性应用于参数 `beginIndex` 到 `endIndex` 之间的文本范围。

请注意，用户手动插入的任何文本或使用 `TextField.replaceSel()` 替换的任何文本均不采用在 `setTextFormat()` 调用中指定的格式。若要设置 `TextField` 对象的默认格式，可使用 `TextField.setNewTextFormat()`。

示例

此示例新建一个名为 `myTextFormat` 的 `TextFormat` 对象，并将其 `bold` 属性设置为 `true`。然后调用 `setTextFormat()`，将新文本格式应用到 `my_txt` 文本字段。

```
myTextFormat = new TextFormat();
myTextFormat.bold = true;
my_txt.setTextFormat(myTextFormat);
```

另请参见

[TextField.setNewTextFormat\(\)](#), [TextFormat](#) 类

TextField._soundbuftime

可用性

Flash Player 6。

用法

```
my_txt._soundbuftime
```

说明

属性（全局）；一个整数，指定在声音开始进入流之前，预先缓冲的秒数。

TextField.StyleSheet 类

可用性

Flash Player 7。

说明

使用 TextField.StyleSheet 类可以创建包含文本格式设置规则（例如，字体大小、颜色和其它格式样式）的样式表对象。然后，可以将样式表定义的样式应用到包含 HTML 或 XML 格式文本的 TextField 对象。该 TextField 对象所包含的文本将自动根据该样式表对象定义的标签样式进行格式设置。可以使用文本样式来定义新的格式标签，重新定义内置的 HTML 标签，或创建可应用到某些 HTML 标签的样式类。

若要将样式应用到某个 TextField 对象，请将样式表对象赋给该 TextField 对象的 `styleSheet` 属性。

有关更多信息，请参见第 122 页的“使用层叠样式表对文本进行格式设置”。

TextField.StyleSheet 类的方法摘要

方法	说明
<code>TextField.StyleSheet.getStyle()</code>	返回与指定样式名相关联的样式表对象的副本。
<code>TextField.StyleSheet.getStyleNames()</code>	返回一个数组，其中包含该样式表对象中注册的所有样式的名称。
<code>TextField.StyleSheet.load()</code>	开始将一个 CSS 文件载入该样式表对象。
<code>TextField.StyleSheet.parseCSS()</code>	对 CSS 文本字符串进行分析，并创建指定的样式。
<code>TextField.StyleSheet.setStyle()</code>	在该样式表对象中添加新样式。

TextField.StyleSheet 类的事件处理函数摘要

方法	说明
<code>TextField.StyleSheet.onLoad</code>	回调处理函数，在 <code>TextField.StyleSheet.load()</code> 操作完成时调用。

TextField.StyleSheet 类的构造函数

可用性

Flash Player 7。

用法

```
new TextField.StyleSheet()
```

返回

无。

说明

构造函数；创建一个 TextField.StyleSheet 对象。

TextField.StyleSheet.getStyle()

可用性

Flash Player 7。

用法

```
styleSheet.getStyle(styleName)
```

参数

styleName 一个字符串，指定要获取的样式的名称。

返回

一个对象。

说明

方法；返回与名为 *styleName* 的样式相关联的样式对象的副本。如果没有样式对象与 *styleName* 相关联，则返回 *null*。

示例

假设一个名为 `textStyles` 的样式表对象加载了一个名为 `styles.css` 的外部样式表文件。该文件中只有一个名为 `heading` 的样式，它定义了 `font-family`、`font-size` 和 `font-weight` 属性，如下所示。

```
// 在 styles.css 中
heading {
    font-family:Arial;
    font-size:24px;
    font-weight:bold;
}
```

下面的代码将加载该 CSS 文件中的样式，然后在“输出”面板中显示各个属性名和它们的值。

```
var styleSheet = new TextField.styleSheet();
styleSheet.load("styles.css");
var sectionStyle = styleSheet.getStyle("heading");
for(property in sectionStyle) {
    var propName = property;
    var propValue = sectionStyle[property];
    trace(propName + " : " + propValue);
}
```

“输出”面板中将显示以下内容：

```
fontfamily :Arial
fontsize :24px
fontweight :bold
```

另请参见

[TextField.StyleSheet.setStyle\(\)](#)

TextField.StyleSheet.getStyleNames()

可用性

Flash Player 7。

用法

```
styleSheet.getStyleNames()
```

参数

无。

返回

一个数组。

说明

方法；返回一个数组，其中包含该样式表中注册的所有样式的名称（字符串形式）。

示例

此示例创建一个名为 `styleSheet` 的样式表对象，其中包含两个样式：`heading` 和 `bodyText`。然后将调用该样式表对象的 `getStyleNames()` 方法，将结果赋给数组 `names_array`，并在“输出”面板中显示该数组的内容。

```
var styleSheet= new TextField.StyleSheet();
styleSheet.setStyle("heading", {
    fontsize:'24px'
});
styleSheet.setStyle("bodyText", {
    fontsize:'12px'
});
var names_array = styleSheet.getStyleNames();
trace(names_array.join("\n"));
```

下面是在“输出”面板中显示的内容：

```
bodyText
heading
```

另请参见

[TextField.StyleSheet.getStyle\(\)](#)

TextField.StyleSheet.load()

可用性

Flash Player 7。

用法

```
styleSheet.load(url)
```

参数

url 要加载的 CSS 文件的 URL。该 URL 必须与 SWF 文件当前位于的 URL 在同一个域中。

返回

无。

说明

方法；开始将该 CSS 文件载入 *styleSheet*。加载操作是异步进行的；使用 [TextField.StyleSheet.onLoad](#) 回调处理函数可以确定该文件何时加载完毕。

该 CSS 文件必须与加载它的 SWF 文件位于完全相同的域中。有关跨域加载数据的限制的信息，请参见第 166 页的“Flash Player 安全功能”。

示例

以下示例将名为 `styles.css`（未显示）的 CSS 文件加载到样式表对象 `styleObj` 中。当该文件成功完成加载后，该样式表对象将应用到名为 `news_txt` 的 TextField 对象。

```
var styleObj = new TextField.StyleSheet();
styleObj.load("styles.css");
styleObj.onLoad = function (success) {
    if (success){
        news_txt.styleSheet = styleObj;
    }
}
```

另请参见

[TextField.StyleSheet.onLoad](#)

TextField.StyleSheet.onLoad

可用性

Flash Player 7。

用法

```
styleSheet.onLoad = function (success) {}
```

参数

success 一个布尔值，指示是否成功加载了该 CSS 文件。

返回

无。

说明

回调处理函数；在 [TextField.StyleSheet.load\(\)](#) 操作完成后调用。如果成功加载了该样式表，*success* 参数为 true。如果未收到该文档，或从服务器接收响应时出现错误，则 *success* 参数为 false。

示例

以下示例将名为 `styles.css`（未显示）的 CSS 文件加载到样式表对象 `styleObj` 中。当该文件成功完成加载后，该样式表对象将应用到名为 `news_txt` 的 `TextField` 对象。

```
var styleObj = new TextField.StyleSheet();
styleObj.load("styles.css");
styleObj.onLoad = function (success) {
    if (success){
        news_txt.styleSheet = styleObj;
    }
}
```

另请参见

[TextField.StyleSheet.load\(\)](#)

TextField.StyleSheet.parseCSS()

可用性

Flash Player 7。

用法

```
styleSheet.parseCSS(cssText)
```

参数

cssText 要分析的 CSS 文本（一个字符串）。

返回

一个布尔值，指示是否成功分析了该文本（如果成功，则为 `true`；如果失败，则为 `false`）。

说明

方法；分析 *cssText* 中的 CSS 并加载相应的样式表。如果 *cssText* 中的某个样式在 *styleSheet* 中已经存在，将保留 *styleSheet* 中的属性，而 *styleSheet* 中只会添加或更改 *cssText* 中有的属性。

若要扩展本机 CSS 分析功能，可通过创建 TextField.StyleSheet 类的子类来覆盖此方法。有关更多信息，请参见[第 143 页的“创建子类”](#)。

TextField.StyleSheet.setStyle()

可用性

Flash Player 7。

用法

```
styleSheet.setStyle(name, style)
```

参数

name 一个字符串，指定要添加到样式表中的样式的名称。

style 一个描述样式的对象，或 `null`。

返回

无。

说明

方法；将具有指定名称的新样式添加到该样式表对象中。如果该样式表中没有具有指定名称的样式，将添加该样式。如果该样式表中已经有具有指定名称的样式，将替换该样式。如果 *style* 参数为 `null`，将删除具有指定名称的样式。

Flash Player 将创建传递给此方法的样式对象的一个副本。

示例

下面的代码将一个名为 `emphasized` 的样式添加到样式表 `myStyleSheet` 中。该样式包含两个样式属性：`color` 和 `fontWeight`。该样式对象用 `{}` 运算符定义。

```
myStyleSheet.setStyle("emphasized", {color:'#000000',fontWeight:'bold'});
```

您也可以使用 `Object` 类的一个实例来创建样式对象，然后将该对象作为 *style* 参数传递给此方法，如下例所示。

```
var styleObj = new Object();
styleObj.color = '#000000';
styleObj.fontWeight = 'bold';
myStyleSheet.setStyle("emphasized", styleObj);
delete styleObj;
```

注意：最后一行代码 (`delete styleObj`) 删除传递给 `setStyle()` 的原始样式对象。虽然此步骤不是必需的，但可以降低内存使用量，因为 Flash Player 会创建传递给 `setStyle()` 的样式对象的副本。

另请参见

[{} \(对象初始值设定项\)](#)

TextField.styleSheet

可用性

Flash Player 7。

用法

```
my_txt.styleSheet = TextField.StyleSheet
```

说明

属性；将一个样式表附加到由 *my_txt* 指定的文本字段。有关创建样式表的信息，请参见 [TextField.StyleSheet](#) 类条目和 [第 122 页](#) 的“使用层叠样式表对文本进行格式设置”。

TextField.tabEnabled

可用性

Flash Player 6。

用法

```
my_txt.tabEnabled
```

说明

属性；指定 *my_txt* 是否包括在 Tab 键的自动排序中。默认情况下它是 `undefined`。

如果 `tabEnabled` 属性为 `undefined` 或 `true`，则该对象包括在 Tab 键的自动排序中。如果 `tabIndex` 属性也设置为某个值，则该对象也包括在 Tab 键的自定义排序中。如果 `tabEnabled` 为 `false`，则即使设置了 `tabIndex` 属性，该对象也不包括在 Tab 键的自动或自定义排序中。

另请参见

[Button.tabEnabled](#), [MovieClip.tabEnabled](#)

TextField.tabIndex

可用性

Flash Player 6。

用法

```
my_txt.tabIndex
```

参数

无。

返回

无。

说明

属性；可用于自定义 SWF 文件中对象的 Tab 键排序。可以设置按钮、影片剪辑或文本字段实例的 `tabIndex` 属性；默认情况下，它为 `undefined`。

如果 SWF 文件中当前显示的任意对象都包含 `tabIndex` 属性，则禁用 Tab 键的自动排序，而使用该 SWF 文件中对象的 `tabIndex` 属性来计算 Tab 键排序。Tab 键的自定义排序仅包括那些具有 `tabIndex` 属性的对象。

`tabIndex` 属性必须为正整数。将根据这些对象的 `tabIndex` 属性按升序对这些对象进行排序。`tabIndex` 值为 1 的对象排在 `tabIndex` 值为 2 的对象的前面。如果两个对象具有相同的 `tabIndex` 值，则在 Tab 键排序中哪个排在前面是 `undefined`。

由 `tabIndex` 属性定义的 Tab 键的自定义排序是平构的。这意味着不考虑 SWF 文件中对象的层次结构关系。SWF 文件中具有 `tabIndex` 属性的所有对象都排入 Tab 键顺序中，而 Tab 键顺序由 `tabIndex` 值的顺序确定。如果两个对象具有相同的 `tabIndex` 值，则哪个在前面是 `undefined`。多个对象不应使用相同的 `tabIndex` 值。

另请参见

[Button.tabIndex](#), [MovieClip.tabIndex](#)

TextField._target

可用性

Flash Player 6。

用法

```
my_txt._target
```

说明

属性（只读）；由 `my_txt` 指定的文本字段的目标路径。

TextField.text

可用性

Flash Player 6。

用法

```
my_txt.text
```

说明

属性；指示文本字段中的当前文本。行用回车符（“\r”即 ASCII 13）分隔。该属性包含文本字段中普通的无格式文本，不包含 HTML 标签，即使该文本字段为 HTML。

另请参见

[TextField.htmlText](#)

TextField.textColor

可用性

Flash Player 6。

用法

```
my_txt.textColor
```

说明

属性；指示文本字段中文本的颜色。

TextField.textHeight

可用性

Flash Player 6。

用法

```
my_txt.textHeight
```

说明

属性；指示文本的高度。

TextField.textWidth

可用性

Flash Player 6。

用法

```
my_txt.textWidth
```

说明

属性；指示文本的宽度。

TextField.type

可用性

Flash Player 6。

用法

```
my_txt.type
```

说明

属性；指定文本字段的类型。有两种值："dynamic" 指定动态文本字段（用户不能对其进行编辑）；"input" 指定输入文本字段。

示例

```
my_txt.type = "dynamic";
```

TextField._url

可用性

Flash Player 6。

用法

```
my_txt._url
```

说明

属性（只读）；获取创建文本字段的 SWF 文件的 URL。

TextField.variable

可用性

Flash Player 6。

用法

```
my_txt.variable
```

说明

属性；与文本字段关联的变量的名称。此属性的类型为 String。

TextField._visible

可用性

Flash Player 6。

用法

```
my_txt._visible
```

说明

属性；一个布尔值，指示文本字段 *my_txt* 是否可见。不可见的文本字段（`_visible` 属性设置为 `false`）被禁用。

另请参见

[Button._visible](#), [MovieClip._visible](#)

TextField._width

可用性

Flash Player 6。

用法

```
my_txt._width
```

说明

属性；文本字段的宽度，以像素为单位。

示例

下面的示例设置文本字段的高度和宽度属性：

```
my_txt._width=200;  
my_txt._height=200;
```

另请参见

[MovieClip._height](#)

TextField.wordWrap

可用性

Flash Player 6。

用法

```
my_txt.wordWrap
```

说明

属性；指示文本字段是否自动换行的布尔值。如果 `wordWrap` 的值为 `true`，则该文本字段自动换行；如果值为 `false`，则该文本字段不自动换行。

TextField._x

可用性

Flash Player 6。

用法

```
my_txt._x
```

说明

属性；一个整数，设置文本字段相对于父级影片剪辑的本地坐标的 x 坐标。如果文本字段在主时间轴上，则其坐标系将舞台的左上角作为 (0, 0)。如果文本字段在具有变形的影片剪辑中，则该文本字段位于其所属影片剪辑的本地坐标系中。因此，对于逆时针旋转 90 度的影片剪辑，其中的文本字段将继承逆时针旋转 90 度的坐标系。文本字段的坐标指的是注册点的位置。

另请参见

[TextField._xscale](#), [TextField._y](#), [TextField._yscale](#)

TextField._xmouse

可用性

Flash Player 6。

用法

```
my_txt._xmouse
```

说明

属性（只读）；返回鼠标位置相对于文本字段的 x 坐标。

另请参见

[TextField._ymouse](#)

TextField._xscale

可用性

Flash Player 6。

用法

```
my_txt._xscale
```

说明

属性；确定从文本字段注册点开始应用的文本字段的水平缩放比例，以百分比形式表示。默认注册点为 (0,0)。

另请参见

[TextField._x](#), [TextField._y](#), [TextField._yscale](#)

TextField._y

可用性

Flash Player 6。

用法

```
my_txt._y
```

说明

属性；文本字段相对于父级影片剪辑本地坐标的 *y* 坐标。如果文本字段在主时间轴中，则其坐标系将舞台的左上角作为 (0, 0)。如果文本字段位于另一个具有变形的影片剪辑中，则该文本字段位于其所属影片剪辑的本地坐标系中。因此，对于逆时针旋转 90 度的影片剪辑，其中的文本字段将继承逆时针旋转 90 度的坐标系。文本字段的坐标指的是注册点的位置。

另请参见

[TextField._x](#), [TextField._xscale](#), [TextField._yscale](#)

TextField._ymouse

可用性

Flash Player 6。

用法

```
my_txt._ymouse
```

说明

属性（只读）；指示鼠标位置相对于文本字段的 *y* 坐标。

另请参见

[TextField._xmouse](#)

TextField._yscale

可用性

Flash Player 6。

用法

```
my_txt._yscale
```

说明

属性；从文本字段的注册点开始应用的文本字段的垂直缩放比例，以百分比形式表示。默认注册点为 (0,0)。

另请参见

[TextField._x](#), [TextField._xscale](#), [TextField._y](#)

TextFormat 类

可用性

Flash Player 6。

说明

TextFormat 类描述字符格式设置信息。

必须在使用构造函数 `new TextFormat()` 创建了 TextFormat 对象之后，才能调用其方法。

可将 TextFormat 参数设置为 `null`，以指示它们是未定义的。使用 `TextField.setTextFormat()` 将 TextFormat 对象应用到文本字段时，只会应用该对象已定义的属性，如下面的示例所示：

```
my_fmt = new TextFormat();
my_fmt.bold = true;
my_txt.setTextFormat(my_fmt);
```

此代码首先创建一个空 TextFormat 对象，该对象的所有属性都未定义，然后将 `bold` 属性设置为一个定义的值。

代码 `my_txt.setTextFormat(my_fmt)` 只更改该文本字段默认文本格式的 `bold` 属性，因为 `bold` 属性是 `my_fmt` 中定义的唯一属性。该文本字段默认文本格式的所有其它内容保持不变。

调用 `TextField.getTextFormat()` 时，将返回一个 TextFormat 对象，该对象的所有属性均已定义；所有属性都不为 `null`。

TextFormat 类的方法摘要

方法	说明
<code>TextFormat.getTextExtent()</code>	返回文本字符串的文本度量信息。

TextFormat 类的属性摘要

属性	说明
<code>TextFormat.align</code>	指示段落的对齐方式。
<code>TextFormat.blockIndent</code>	指示块的缩进，以磅为单位。
<code>TextFormat.bold</code>	指示文本是否为粗体。
<code>TextFormat.bullet</code>	指示文本是否在带有项目符号的列表中。
<code>TextFormat.color</code>	指示文本的颜色。
<code>TextFormat.font</code>	指示带有文本格式的文本的字体名称。
<code>TextFormat.indent</code>	指示从左边距到段落中第一个字符的缩进。
<code>TextFormat.italic</code>	指示文本是否为斜体。
<code>TextFormat.leading</code>	指示行与行之间的垂直间距量（称为 前导 ）。
<code>TextFormat.leftMargin</code>	以磅为单位指示段落的左边距。
<code>TextFormat.rightMargin</code>	以磅为单位指示段落的右边距。
<code>TextFormat.size</code>	指示文本的磅值。

属性	说明
<code>TextFormat.tabStops</code>	指定 Tab 键的自定义停靠位。
<code>TextFormat.target</code>	指示浏览器中显示超链接的窗口。
<code>TextFormat.underline</code>	指示文本是否带有下划线。
<code>TextFormat.url</code>	指示文本所链接到的 URL。

TextFormat 类的构造函数

可用性

Flash Player 6。

用法

```
new TextFormat([font, [size, [color, [bold, [italic, [underline, [url, [target, [align, [leftMargin, [rightMargin, [indent, [leading]]]]]]]]]]])
```

参数

font 以字符串形式表示的文本字体名称。

size 指示磅值的整数。

color 使用此文本格式的文本的颜色。包含三个 8 位 RGB 分量的数字；例如，0xFF0000 为红色、0x00FF00 为绿色。

bold 指示文本是否为粗体的布尔值。

italic 指示文本是否为斜体的布尔值。

underline 指示文本是否带有下划线的布尔值。

url 使用此文本格式的文本超链接到的 URL。如果 *url* 为空字符串，则表示文本没有超链接。

target 显示超链接的目标窗口。如果目标窗口为空字符串，则文本显示在默认目标窗口 `_self` 中。如果 *url* 参数设置为空字符串或值 `null`，虽然您可以获取或设置此属性，但该属性不起作用。

align 以字符串形式表示的段落对齐方式。如果为 "left"，则段落为左对齐。如果为 "center"，则段落居中。如果为 "right"，则段落为右对齐。

leftMargin 指示段落的左边距，以磅为单位。

rightMargin 指示段落的右边距，以磅为单位。

indent 一个整数，指示从左边距到段落中第一个字符的缩进量。

leading 一个数字，指示行与行之间的前导垂直间距离。

返回

无。

说明

构造函数；使用指定的属性创建一个 `TextFormat` 对象。然后可更改 `TextFormat` 对象的属性以更改文本字段的格式设置。

任何参数都可设置为 `null` 以指示该参数未定义。所有的参数都是可选的；任何被忽略的参数都被视为 `null`。

TextFormat.align

可用性

Flash Player 6。

用法

```
my_fmt.align
```

说明

属性；以字符串形式指示段落的对齐方式。以字符串形式表示的段落的对齐方式。如果为 "left"，则段落为左对齐。如果为 "center"，则段落居中。如果为 "right"，则段落为右对齐。默认值为 `null`，它指示未定义该属性。

TextFormat.blockIndent

可用性

Flash Player 6。

用法

```
my_fmt.blockIndent
```

说明

属性；以磅为单位指示块缩进的数字。块缩进应用于整个文本块，即文本的所有行。相反，普通缩进 ([TextFormat.indent](#)) 只影响各段的第一行。如果此属性为 `null`，则 `TextFormat` 对象没有指定块缩进。

TextFormat.bold

可用性

Flash Player 6。

用法

```
my_fmt.bold
```

说明

属性；指示文本是否为粗体的布尔值。默认值为 `null`，它指示未定义该属性。

TextFormat.bullet

可用性

Flash Player 6。

用法

```
my_fmt.bullet
```

说明

属性；一个布尔值，指示文本为带项目符号的列表的一部分。在带项目符号的列表中，文本的各段都是缩进的。项目符号显示在各段第一行的左侧。默认值为 `null`。

TextFormat.color

可用性

Flash Player 6。

用法

```
my_fmt.color
```

说明

属性；指示文本的颜色。包含三个 8 位 RGB 分量的数字；例如，0xFF0000 为红色、0x00FF00 为绿色。

TextFormat.font

可用性

Flash Player 6。

用法

```
my_fmt.font
```

说明

属性；使用该文本格式的文本的字体名称，以字符串形式表示。默认值为 `null`，它指示未定义该属性。

TextFormat.getTextExtent()

可用性

Flash Player 6。Flash Player 7 中支持可选的 *width* 参数。

用法

```
my_fmt.getTextExtent(text, [width])
```

参数

text 一个字符串。

width 可选参数；一个表示宽度的数字（以像素为单位），指定的文本应在该处换行。

返回

一个对象，它具有以下属性：`width`、`height`、`ascent`、`descent`、`textFieldHeight`、`textFieldWidth`。

说明

方法；返回使用由 *my_fmt* 指定的格式表示的文本字符串 *text* 的文本度量信息。文本字符串被视为纯文本（而不是 HTML）。

此方法返回一个具有以下六个属性的对象：`ascent`、`descent`、`width`、`height`、`textFieldHeight` 和 `textFieldWidth`。所有度量值均以像素为单位。

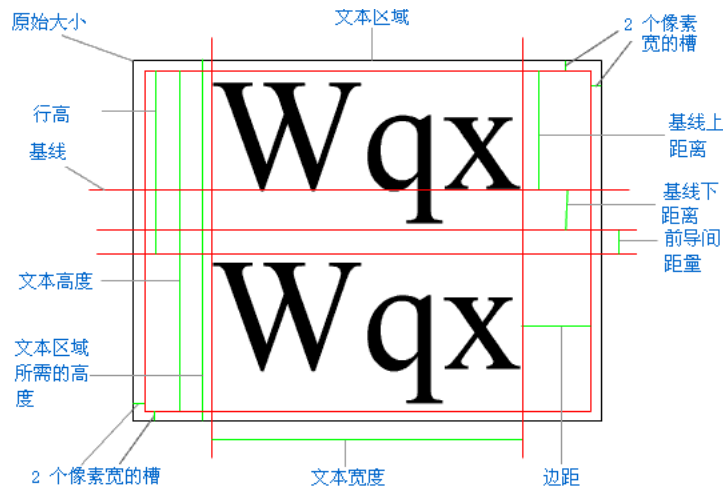
如果指定了 *width* 参数，指定的文本将自动换行。这样，您就可以确定文本框在显示所有指定的文本时，它的高度是多少。

ascent 和 descent 度量值分别提供一行文本的基线上面或下面的距离。第一行文本的基线位于该文本字段的原点加上其 ascent 度量值的位置。

width 和 height 度量值提供该文本字符串的宽度和高度。textFieldHeight 和 textFieldWidth 度量值提供文本字段对象显示整个文本字符串所需要具有的高度和宽度。文本字段的四周有 2 个像素宽的“槽”，因此 textFieldHeight 的值等于 height 的值 + 4；同样，textFieldWidth 的值也始终等于 width 的值 + 4。

如果要根据文本度量值创建文本字段，应使用 textFieldHeight（而不是 height）和 textFieldWidth（而不是 width）。

下图中介绍这些度量值。



设置 TextFormat 对象时，所有属性设置必须与创建文本字段时所设置的属性完全一样，包括字体名称、字体大小和前导间距离。前导间距离的默认值为 2。

示例

此示例创建一个单行文本字段，其大小刚好足够使用指定格式显示一个文本字符串。

```
var text = "Small string";

// 创建一个 TextFormat 对象
// 并应用其属性。
var txt_fmt = new TextFormat();
with(txt_fmt) {
    font = "Arial";
    bold = true;
}

// 获取该文本字符串使用指定格式时
// 的度量信息。
var metrics = txt_fmt.getTextExtent(text);

// 创建一个文本字段，其大小刚好足够显示该文本。
this.createTextField("textField", 0, 100, 100, metrics.textFieldWidth,
    metrics.textFieldHeight);
textField.border = true;
textField.wordWrap = true;
// 将相同的文本字符串和
```

```
// TextFormat 对象分配给该 TextField 对象。  
textField.text = text;  
textField.setTextFormat(txt_fmt);
```

下面的示例创建一个 100 像素宽的多行文本字段，其高度足够使用指定的格式显示一个字符串。

```
// 创建一个 TextFormat 对象。  
var txt_fmt:TextFormat= new TextFormat();  
  
// 为该 TextFormat 对象指定格式属性：  
txt_fmt.font = "Arial";  
txt_fmt.bold = true;  
txt_fmt.leading = 4;  
  
// 要显示的文本字符串  
var textToDisplay:String = "Macromedia Flash 7, now with improved text  
metrics.";   
  
// 获取在 100 像素换行处  
// 该字符串的文本度量信息  
var metrics:Object = txt_fmt.getTextExtent(textToDisplay, 100);  
  
// 使用刚刚获取的度量信息创建一个  
// 新的 TextField 对象。  
this.createTextField ("textField", 0, 50, 50-metrics.ascent, 100,  
    metrics.textFieldHeight)  
textField.wordWrap = true;  
textField.border = true;  
// 将该文本和 TextFormat 对象分配给该 TextObject：  
textField.text = textToDisplay;  
textField.setTextFormat(aformat);
```

TextFormat.indent

可用性

Flash Player 6。

用法

```
my_fmt.indent
```

说明

属性；指示从左边距到段落中第一个字符的缩进的整数。默认值为 null，它指示未定义该属性。

另请参见

[TextFormat.blockIndent](#)

TextFormat.italic

可用性

Flash Player 6。

用法

```
my_fmt.italic
```

说明

属性；指示该文本格式中文本是否为斜体的布尔值。默认值为 null，它指示未定义该属性。

TextFormat.leading

可用性

Flash Player 6。

用法

```
my_fmt.leading
```

说明

属性；行与行之间的垂直间距量（称为前导）。默认值为 `null`，它指示未定义该属性。

TextFormat.leftMargin

可用性

Flash Player 6。

用法

```
my_fmt.leftMargin
```

说明

属性；段落的左边距，以磅为单位。默认值为 `null`，它指示未定义该属性。

TextFormat.rightMargin

可用性

Flash Player 6。

用法

```
my_fmt.rightMargin
```

说明

属性；段落的右边距，以磅为单位。默认值为 `null`，它指示未定义该属性。

TextFormat.size

可用性

Flash Player 6。

用法

```
my_fmt.size
```

说明

属性；该文本格式中文本的磅值。默认值为 `null`，它指示未定义该属性。

TextFormat.tabStops

可用性

Flash Player 6。

用法

```
my_fmt.tabStops
```

说明

属性；使用一个非负整数的数组指定自定义 Tab 键停靠位。以磅为单位指定每个 Tab 键的停靠位。如果没有指定自定义 Tab 键停靠位 (`null`)，则默认 Tab 键停靠位为 4（平均字符宽度）。

TextFormat.target

可用性

Flash Player 6。

用法

```
my_fmt.target
```

说明

属性；指示显示超链接的目标窗口。如果目标窗口为空字符串，则文本显示在默认目标窗口 `_self` 中。如果 `TextFormat.url` 属性是空字符串或 `null`，虽然您可以获取或设置此属性，但该属性不起作用。

TextFormat.underline

可用性

Flash Player 6。

用法

```
my_fmt.underline
```

说明

属性，一个布尔值，指定使用此文本格式的文本下面是否有下划线（如果有，则为 `true`；如果没有，则为 `false`）。此下划线类似于用 `<U>` 标签产生的下划线，但后者不是“真正”的下划线，因为它不能正确地跳过字母的下伸部分。默认值为 `null`，它指示未定义该属性。

TextFormat.url

可用性

Flash Player 6。

用法

```
my_fmt.url
```

说明

属性；指示该文本格式中的文本链接所至的 URL。如果 `url` 属性为空字符串，则文本没有超链接。默认值为 `null`，它指示未定义该属性。

TextSnapshot 对象

可用性

创作 : Flash MX 2004。
回放 : 为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

说明

TextSnapshot 对象可用于处理影片剪辑中的静态文本。例如，您可以使用它们用比动态文本所允许的更高的精确对文本进行布局，但仍以只读方式访问该文本。
不使用构造函数创建 TextSnapshot 对象；它是由 `MovieClip.getTextSnapshot()` 返回的。

TextSnapshot 对象的方法摘要

方法	说明
<code>TextSnapshot.findText()</code>	返回指定文本的第一个匹配项的位置。
<code>TextSnapshot.getCount()</code>	返回字符数。
<code>TextSnapshot.getSelected()</code>	指定指定范围内的任何文本是否已被 <code>TextSnapshot.setSelected()</code> 选中。
<code>TextSnapshot.getSelectedText()</code>	返回一个包含 <code>TextSnapshot.setSelected()</code> 指定的所有字符的字符串。
<code>TextSnapshot.getText()</code>	返回一个包含指定范围内字符的字符串。
<code>TextSnapshot.hitTestTextNearPos()</code>	用于确定对象中的哪个字符在指定的坐标上或靠近指定的坐标。
<code>TextSnapshot.setSelectColor()</code>	指定当突出显示使用 <code>TextSnapshot.setSelected()</code> 命令选中的字符时要使用的颜色。
<code>TextSnapshot.setSelected()</code>	指定是选择还是取消选择一系列字符。

TextSnapshot.findText()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
my_snap.findText( startIndex, textToFind, caseSensitive )
```

参数

startIndex 一个整数，指定在 *my_snap* 中搜索指定文本的起始点。

textToFind 一个字符串，指定要搜索的文本。如果指定文本字符串而不是 String 类型的变量，则将该字符串引在引号中。

caseSensitive 一个布尔值，指定 *my_snap* 中文本的大小写是否必须和 *textToFind* 中字符串的大小写匹配。

返回

指定文本的第一个匹配项的从零开始的索引位置，或 -1。

说明

方法；搜索指定的 TextSnapshot 对象，并返回在 *startIndex* 或其后找到的 *textToFind* 的第一个匹配项的位置。如果未找到 *textToFind*，则此方法返回 -1。

另请参见

[TextSnapshot.getText\(\)](#)

TextSnapshot.getCount()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
my_snap.getCount()
```

参数

无。

返回

一个表示指定的 TextSnapshot 对象中字符数的整数。

说明

方法；返回 TextSnapshot 对象中的字符数。

另请参见

[TextSnapshot.getText\(\)](#)

TextSnapshot.getSelected()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
my_snap.getSelected(from, to)
```

参数

from 一个整数，指示要检查的 *my_snap* 中第一个字符的位置。*from* 的有效值为 0 到 [TextSnapshot.getCount\(\)](#) - 1。如果 *from* 为负值，则使用 0。

to 一个整数，为 1+ 要检查的 *my_snap* 中最后一个字符的索引。*to* 的有效值为 0 到 [TextSnapshot.getCount\(\)](#)。提取出来的字符串中不包括由 *to* 参数索引指定的字符。如果省略了此参数，则使用 [TextSnapshot.getCount\(\)](#)。如果此值小于或等于 *from* 的值，则使用 *from*+1。

返回

如果给定范围内至少一个字符已被对应的 [TextSnapshot.setSelected\(\)](#) 命令选中，则为布尔值 true，否则为 false。

说明

方法；返回一个布尔值，该值指定 TextSnapshot 对象在指定范围内是否包含所选的文本。

若要搜索所有字符，则为 *from* 传递值 0，为 *to* 传递 [TextSnapshot.getCount\(\)](#)（或任何一个非常大的数字）。若要搜索单个字符，则为 *to* 传递值 *from*+1。

另请参见

[TextSnapshot.getSelectedText\(\)](#), [TextSnapshot.getText\(\)](#)

TextSnapshot.getSelectedText()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
mySnapshot.getSelectedText( [ includeLineEndings ] )
```

参数

includeLineEndings 一个可选布尔值，该值指定是否在返回的字符串中的适当位置插入换行符。默认值为 *false*。

返回

一个字符串，该字符串包含对应的 `TextSnapshot.setSelected()` 命令指定的所有字符。

说明

方法；返回一个字符串，该字符串包含对应的 `TextSnapshot.setSelected()` 命令指定的所有字符。如果未选择任何字符，则返回空字符串。

如果为 *includeLineEndings* 传递值 *true*，则在返回的字符串中认为适当的位置插入换行符。在这种情况下，返回字符串可能比输入范围要长。如果 *includeLineEndings* 为 *false* 或被省略，则返回所选的文本，不添加任何字符。

另请参见

[TextSnapshot.getSelected\(\)](#)

TextSnapshot.getText()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
mySnapshot.getText(from, to [, includeLineEndings ] )
```

参数

from 一个整数，指示要包括在返回的字符串中的 *my_snap* 第一个字符的位置。*from* 的有效值为 0 到 [TextSnapshot.getCount\(\)](#) - 1。如果 *from* 为负值，则使用 0。

to 一个整数，为 1+ 要检查的 *my_snap* 中最后一个字符的索引。*to* 的有效值为 0 到 [TextSnapshot.getCount\(\)](#)。提取出来的字符串中不包括由 *to* 参数索引指定的字符。如果省略了此参数，则使用 [TextSnapshot.getCount\(\)](#)。如果此值小于或等于 *from* 的值，则使用 *from*+1。

includeLineEndings 一个可选布尔值，该值指定是否在返回的字符串中的适当位置插入换行符。默认值为 *false*。

返回

一个包含指定范围中字符的字符串，如果指定范围中未找到任何字符，则为空字符串。

说明

方法；返回一个字符串，该字符串包含 *from* 和 *to* 参数指定的所有字符。如果未选择任何字符，则返回空字符串。

若要返回所有字符，则为 *from* 传递值 0，为 *to* 传递 [TextSnapshot.getCount\(\)](#)（或任何一个非常大的数字）。若要返回单个字符，则为 *to* 传递值 *from*+1。

如果为 *includeLineEndings* 传递值 *true*，则在返回的字符串中认为适当的位置插入换行符。在这种情况下，返回字符串可能比输入范围要长。如果 *includeLineEndings* 为 *false* 或被省略，则返回所选的文本，不添加任何字符。

另请参见

[TextSnapshot.getSelectedText\(\)](#)

TextSnapshot.hitTestTextNearPos()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
my_snap.hitTestTextNearPos(x, y [, maxDistance] )
```

参数

x 一个数字，表示包含 *my_snap* 中文本的影片剪辑的 *x* 坐标。

y 一个数字，表示包含 *my_snap* 中文本的影片剪辑的 *y* 坐标。

maxDistance 一个可选数字，表示从 *x*、*y* 坐标出发搜索文本的最大距离。此距离是从每个字符的中心点开始测量的。默认值为 0。

返回

一个整数，表示 *my_snap* 中与指定的 *x*、*y* 坐标最近的字符的索引值，如果未找到任何字符，则为 -1。

说明

方法；用于确定 TextSnapshot 对象中哪个字符在影片剪辑（该影片剪辑包含 *my_snap* 中的文本）中指定的 *x*、*y* 坐标上或离该坐标最近。

如果为 *maxDistance* 传递值 0 或将其省略，则 *x*、*y* 坐标指定的位置必须在 *my_snap* 的边框内。

另请参见

[MovieClip.getTextSnapshot\(\)](#), [MovieClip._x](#), [MovieClip._y](#)

TextSnapshot.setSelectColor()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
mySnapshot.setSelectColor(hexColor);
```

参数

hexColor 用于字符（这些字符已被对应的 [TextSnapshot.setSelected\(\)](#) 命令选中）旁边边框的颜色，以 0xRRGGBB 格式表示。

返回

无。

说明

方法；指定当突出显示使用 [TextSnapshot.setSelected\(\)](#) 命令选中的字符时要使用的颜色。该颜色一定是不透明的；不能指定透明值。

TextSnapshot.setSelected()

可用性

创作：Flash MX 2004。

回放：为 Flash Player 6 或更高版本发布的 SWF 文件，在 Flash Player 7 或更高版本中播放。

用法

```
mySnapshot.setSelected(from, to, select)
```

参数

from 一个整数，指示要选择的 *my_snap* 中第一个字符的位置。*from* 的有效值是 0 到 `TextSnapshot.getCount()` - 1。如果 *from* 为负值，则使用 0。

to 一个整数，为 1+ 要检查的 *my_snap* 中最后一个字符的索引。*to* 的有效值为 0 到 `TextSnapshot.getCount()`。提取出来的字符串中不包括由 *to* 参数索引指定的字符。如果省略了此参数，则使用 `TextSnapshot.getCount()`。如果此值小于或等于 *from* 的值，则使用 *from*+1。

select 一个布尔值，指定是应该选择文本 (`true`) 还是应该取消选择文本 (`false`)。

返回

无。

说明

方法；指定是选择还是取消选择 `TextSnapshot` 对象中的一系列字符。被选择的字符绘制时后面有一个带有颜色的矩形，与字符的边框匹配。边框的颜色由 `TextSnapshot.setSelectColor()` 确定。

若要选择或取消选择所有字符，为 *from* 传递值 0，为 *to* 传递 `TextSnapshot.getCount()`（或任何一个非常大的数字）。若要指定单个字符，则为 *to* 传递值 *from*+1。

因为字符是被单个地标为选中的，所以您可以多次发布此命令以选择多个字符；也就是说，使用此命令不会取消选择已由此命令设置的其它字符。

this

可用性

Flash Player 5。

用法

this

说明

标识符；引用对象或影片剪辑实例。在脚本执行时，this 引用包含该脚本的影片剪辑实例。在调用方法时，this 包含对包括所调用方法的对象的引用。

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

因为 this 是在包含它的脚本的上下文中计算的，所以脚本中不能使用 this 引用类文件中定义的变量：

```
// 在文件 applyThis.as 中
class applyThis{
    var str:String = "Defined in applyThis.as";
    function conctStr(x:String):String{
        return x+x;
    }

    function addStr():String{
        return str;
    }
}

// 在 FLA 中使用以下代码测试影片
import applyThis;

var obj:applyThis = new applyThis();
var abj:applyThis = new applyThis();
abj.str = "defined in FLA";

trace(obj.addStr.call(abj,null));    // 在 FLA 中定义
trace(obj.addStr.call(this,null));   // 未定义
trace(obj.addStr.call(obj,null));    // 在 applyThis.as 中定义
```

同样，若要调用动态类中定义的函数，您必须使用 this 设置该函数的范围：

```
// simple.as 不正确的版本
dynamic class simple{
    function callfunc(){
        trace(func());
    }
}

// simple.as 正确的版本
dynamic class simple{
    function callfunc(){
        trace(this.func());
    }
}

// FLA 文件中的语句
import simple;
```

```
var obj:simple = new simple();
obj.num = 0;
obj.func = function():Boolean{
    return true;
}
obj.callfunc(); // simple.as 不正确版本中的语法错误
```

示例

在下面的示例中，关键字 `this` 引用 `Circle` 对象。

```
function Circle(radius) {
    this.radius = radius;
    this.area = Math.PI * radius * radius;
}
```

在下面分配给帧的语句中，关键字 `this` 引用当前的影片剪辑。

```
// 将当前影片剪辑的 alpha 属性设置为 20
this._alpha = 20;
```

在下面的 `onClipEvent()` 处理函数内的语句中，关键字 `this` 引用当前的影片剪辑。

```
// 加载该影片剪辑时，对当前影片剪辑
// 启动一个 startDrag() 操作。

onClipEvent (load) {
    startDrag (this, true);
}
```

另请参见

[on\(\)](#), [onClipEvent\(\)](#)

throw

可用性

Flash Player 7。

用法

`throw expression`

说明

语句；生成（“抛出”）一个可由 `catch()` 或 `finally()` 代码块处理（“捕获”）的错误。如果异常未被 `catch` 或 `finally` 代码块捕获，抛出值的字符串表示形式将被发送到“输出”面板。

通常，抛出的错误是 `Error` 类或其子类的实例（请参见下面的示例）。

参数

expression 一个动作脚本表达式或对象。

示例

在此示例中，一个名为 `checkEmail()` 的函数检查传递给它的字符串是否为格式正确的电子邮件地址。如果该字符串不包含 `@` 符号，则该函数将抛出一个错误。

```
function checkEmail(email:String) {  
    if (email.indexOf("@") == -1) {  
        throw new Error("Invalid email address");  
    }  
}
```

然后，下面的代码在一个 `try` 代码块内调用 `checkEmail()` 函数，将一个文本字段 (`email_txt`) 中的文本作为参数传递给该函数。如果该字符串参数中不包含有效的电子邮件地址，将在一个文本字段 (`error_txt`) 中显示该错误消息。

```
try {  
    checkEmail("Joe Smith");  
} catch (e) {  
    error_txt.text = e.toString();  
}
```

在此示例中，将抛出 `Error` 类的一个子类。我们对 `checkEmail()` 函数进行了修改，它将抛出该子类的一个示例。（有关更多信息，请参见第 143 页的“创建子类”。）

```
// 定义 Error 的子类 InvalidEmailError  
// 在 InvalidEmailError.as 中：  
class InvalidEmailAddress extends Error {  
    var message = "Invalid email address.";  
}
```

```
function checkEmail(email:String) {  
    if (email.indexOf("@") == -1) {  
        throw new InvalidEmailAddress();  
    }  
}
```

另请参见

[Error 类](#), [try...catch...finally](#)

toggleHighQuality()

可用性

Flash 2 ；不鼓励使用，推荐使用 [_quality](#)。

用法

```
toggleHighQuality()
```

参数

无。

返回

无。

说明

不鼓励使用的函数；在 Flash Player 中开启和关闭消除锯齿功能。消除锯齿可使对象的边缘平滑并会减缓 SWF 播放的速度。此动作会影响 Flash Player 中的所有 SWF 文件。

示例

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

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

另请参见

[_highquality](#), [_quality](#)

trace()

可用性

Flash Player 4。

用法

`trace(expression)`

参数

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

返回

无。

说明

语句；在测试模式下，计算表达式并在“输出”面板中显示结果。

在测试影片时，使用此动作可在“输出”面板中记录编程注释或显示消息。使用 *expression* 参数可以检查是否存在某种条件，或在“输出”面板中显示值。`trace()` 动作类似于 JavaScript 中的 `alert` 函数。

可以使用“发布设置”中的“Omit Trace Actions”（省略 Trace 动作）命令将 `trace()` 动作从导出的 SWF 文件中删除。

示例

此示例来自一个游戏，在该游戏中一个名为 `my_mc` 的可拖动影片剪辑实例必须在特定目标上释放。我们使用一个条件语句来计算 `_droptarget` 属性并根据 `my_mc` 的释放位置执行不同的动作。在脚本的末尾使用 `trace()` 动作来计算 `my_mc` 影片剪辑的位置，并在“输出”面板中显示结果。如果 `my_mc` 的行为与预期的不一样（例如，如果它对齐的目标不正确），则由 `trace()` 动作发送到“输出”面板的值将有助于您确定脚本中的问题。

```
on (press) {
    my_mc.startDrag();
}

on (release) {
    if(eval(_droptarget) != target) {
        my_mc._x = my_mc.xValue;
        my_mc._y = my_mc.yValue;
    } else {
        var my_mc_xValue = my_mc._x;
        var my_mc_yValue = my_mc._y;
        target = "_root.pasture";
    }
    trace("my_mc_xValue = " + my_mc_xValue);
    trace("my_mc_yValue = " + my_mc_yValue);
    stopDrag();
}
```

true

可用性

Flash Player 5。

用法

```
true
```

说明

常量；表示与 `false` 相反的唯一布尔值。

另请参见

[false](#)

try..catch..finally

可用性

Flash Player 7。

用法

```
try {  
    // ... try 代码块 ...  
} finally {  
    // ... finally 代码块 ...  
}  
  
try {  
    // ... try 代码块 ...  
} catch(error[:ErrorType1]) {  
    // ... catch 代码块 ...  
} [catch(error[:ErrorTypeN]) {  
    // ... catch 代码块 ...  
}] [finally {  
    // ... finally 代码块 ...  
}]
```

参数

error 从 `throw` 语句抛出的表达式，通常是 `Error` 类或其子类的一个实例。

ErrorType 可选参数；指定 *error* 标识符的类型。`catch` 子句只捕获指定类型的错误。

说明

关键字；在它们所包括的代码块内，如果发生错误，将对错误进行响应。如果 `try` 代码块内的任何代码抛出了一个错误（使用 `throw` 动作），控制将传递到 `catch` 代码块（如果有），然后传递到 `finally` 代码块（如果有）。无论是否有错误被抛出，`finally` 代码块都会执行。如果 `try` 代码块内的代码未抛出错误（也就是说，如果 `try` 代码块正常完成），则仍会执行 `finally` 代码块内的代码。即使 `try` 代码块使用 `return` 语句退出，`finally` 代码块仍会执行。

`try` 代码块后面必须跟有 `catch` 代码块、`finally` 代码块，或两者都有。一个 `try` 代码块可以有多个 `catch` 代码块，但只能有一个 `finally` 代码块。您可以根据需要嵌套多层 `try` 代码块（层数没有限制）。

在 `catch` 处理函数内指定的 `error` 参数必须为一个简单的标识符，例如，`e` 或 `theException` 或 `x`。您还可以为 `catch` 处理函数中的变量指定类型。如果指定了不同的错误类型，再结合使用多个 `catch` 代码块，可以捕获从一个 `try` 代码块抛出的多种类型的错误。

抛出的异常是对象时，如果抛出的对象是指定类型的一个子类，则类型将匹配。如果抛出的是特定类型的错误，将执行处理相应错误的 `catch` 代码块。如果抛出的不是指定类型的异常，则不会执行 `catch` 代码块，而会自动将该异常从 `try` 代码块抛出到与其相匹配的 `catch` 处理函数。

如果在某个函数内抛出了错误，而该函数中没有 `catch` 处理函数，则动作脚本解释器退出该函数以及任何调用函数，直到找到一个 `catch` 代码块。在此过程中，在各层上都会调用 `finally` 处理函数。

示例

下面的示例说明如何创建 `try..finally` 语句。由于 `finally` 代码块中的代码肯定会执行，因此，它通常用于在 `try` 代码块执行完毕后，执行必要的“清理”代码。在此示例中，`finally` 代码块用于删除一个动作脚本对象，而不管是否有错误发生。

```
var account = new Account()
try {
    var returnVal = account.getAccountInfo();
    if(returnVal != 0) {
        throw new Error("Error getting account information.");
    }
}
finally {
    // 无论是否有错误发生，都会删除“account”对象。
    if(account != null) {
        delete account;
    }
}
```

下面的示例演示一个 `try..catch` 语句。`try` 代码块内的代码将会执行。如果 `try` 代码块内的任何代码抛出异常，控制将传递到 `catch` 代码块。该代码块使用 `Error.toString()` 方法在一个文本字段中显示该错误消息。

```
var account = new Account()
try {
    var returnVal = account.getAccountInfo();
    if(returnVal != 0) {
        throw new Error("Error getting account information.");
    }
} catch (e) {
    status_txt.text = e.toString();
}
```

下面的示例演示一个带有多个 `catch` 代码块的 `try` 代码块，而且在 `catch` 代码块中指定了错误类型。`try` 代码块将根据所发生错误的类型抛出不同类型的对象。在此示例中，`myRecordSet` 是一个名为 `RecordSet` 的类（假设有这样一个类）的实例。该类的 `sortRows()` 方法可能会抛出以下两种不同类型的错误：`RecordSetException` 和 `MalformedRecord`。

在此示例中，`RecordSetException` 和 `MalformedRecord` 对象是 `Error` 类的子类。它们都是在其自己的 `AS` 类文件中定义的。（有关更多信息，请参见第 137 页的第 9 章“使用动作脚本 2.0 创建类”。）

```
// 在 RecordSetException.as 中：
class RecordSetException extends Error {
    var message = "Record set exception occurred."
}
// 在 MalformedRecord.as 中：
```

```
class MalformedRecord extends Error {
    var message = "Malformed record exception occurred.";
}
```

在 `RecordSet` 类的 `sortRows()` 方法内，将根据所发生异常的类型抛出上面定义的两个错误对象之一。下面的代码片断显示了此代码的大致结构。

```
// 在 RecordSet.as 类文件内 ...
function sortRows() {
    ...
    if(recordSetErrorCondition) {
        throw new RecordSetException();
    }
    if(malformedRecordCondition) {
        throw new MalformedRecord();
    }
    ...
}
```

最后，在另一个 AS 文件或 FLA 脚本中，以下代码对 `RecordSet` 类的一个实例调用 `sortRows()` 方法。它为 `sortRows()` 所抛出的各种类型的错误都分别定义了 `catch` 代码块。

```
try {
    myRecordSet.sortRows();
} catch (e:RecordSetException) {
    trace("Caught a recordset exception");
} catch (e:MalformedRecord) {
    trace("Caught a malformed record exception");
}
```

另请参见

[Error 类](#), [throw](#), [class](#), [extends](#)

typeof

可用性

Flash Player 5。

用法

`typeof(expression)`

参数

expression 一个字符串、影片剪辑、按钮、对象或函数。

说明

运算符；放在单个参数前的一元运算符。typeof 运算符让 Flash 解释器计算 *expression*；其结果为一个字符串，指示该表达式是字符串、影片剪辑、对象、函数、数字，还是布尔值。下表显示对各类表达式使用 typeof 运算符的结果：

参数	输出
字符串	string
影片剪辑	movieclip
按钮	object
文本字段	object
数字	number
布尔值	boolean
对象	object
函数	function

undefined

可用性

Flash Player 5。

用法

undefined

参数

无。

返回

无。

说明

一个特殊值，通常用于指示变量尚未赋值。对未定义值的引用返回特殊值 `undefined`。动作脚本代码 `typeof(undefined)` 返回字符串 `"undefined"`。`undefined` 类型的唯一值就是 `undefined`。

在为 Flash Player 6 或更早版本发布的文件中，`undefined.toString()` 的值为 `""`（空字符串）。在为 Flash Player 7 或更高版本发布的文件中，`undefined.toString()` 的值为 `undefined`。

`undefined` 值与特殊值 `null` 相似。使用相等运算符对 `null` 和 `undefined` 进行比较时，它们的比较结果为相等。

示例

在此示例中，变量 `x` 尚未声明，所以其值为 `undefined`。在代码的第一部分，我们使用相等运算符 (`==`) 对 `x` 的值与值 `undefined` 进行比较，并将相应的结果发送到“输出”面板。在代码的第二部分，我们使用相等运算符对值 `null` 和 `undefined` 进行比较。

```
// x 尚未声明
trace ("The value of x is " + x);
if (x == undefined) {
    trace ("x is undefined");
} else {
    trace ("x is not undefined");
}

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

下面的结果显示在“输出”面板中。

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


unescape

可用性

Flash Player 5。

用法

```
unescape(x)
```

参数

x 要转义的十六进制序列字符串。

返回

对 URL 编码的参数进行解码所得到的字符串。

说明

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

示例

下面的示例说明了转义到反向转义的转换过程。

```
escape("Hello{[World]}");
```

转义结果如下：

```
("Hello%7B%5BWorld%5D%7D");
```

使用 `unescape` 可以返回到原始格式：

```
unescape("Hello%7B%5BWorld%5D%7D");
```

结果如下：

```
Hello{[World]}
```

unloadMovie()

可用性

Flash Player 3。

用法

```
unloadMovie(target)
```

参数

target 影片剪辑的目标路径。

返回

无。

说明

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

示例

下面的示例卸载主时间轴上的影片剪辑 `draggable_mc`，并将 `movie.swf` 加载到级别 4 中。

```
on (press) {  
    unloadMovie ("_root.draggable_mc");  
    loadMovieNum ("movie.swf", 4);  
}
```

下面的示例卸载已经加载到级别 4 中的影片。

```
on (press) {  
    unloadMovieNum (4);  
}
```

另请参见

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

unloadMovieNum()

可用性

Flash Player 3。

用法

```
unloadMovieNum(level)
```

参数

level 所加载影片的级别 (*_levelN*)。

返回

无。

说明

函数；从 Flash Player 中删除通过 `loadMovieNum()` 加载的影片。若要卸载通过 `loadMovie()` 加载的影片，应使用 `unloadMovie()`，而不应使用 `unloadMovieNum()`。

另请参见

[loadMovie\(\)](#), [loadMovieNum\(\)](#), [unloadMovie\(\)](#)

updateAfterEvent()

可用性

Flash Player 5。

用法

```
updateAfterEvent()
```

参数

无。

返回

无。

说明

函数；在 `onClipEvent()` 处理函数内调用此函数时，或将其作为传递给 `setInterval()` 的函数或方法的一部分进行调用时，它将更新显示（与为影片设置的帧频无关）。如果对 `updateAfterEvent` 的调用不在 `onClipEvent()` 处理函数内，也不是传递给 `setInterval()` 的函数或方法的一部分，则 Flash 将忽略该调用。

另请参见

[onClipEvent\(\)](#), [setInterval\(\)](#)

var

可用性

Flash Player 5。

用法

```
var variableName [= value1] [...,variableNameN [=valueN]]
```

参数

variableName 一个标识符。

value 赋给该变量的值。

返回

无。

说明

语句；用于声明局部或时间轴变量。

- 如果变量是在函数内声明的，则这些变量是局部变量。它们是为该函数声明的，在函数调用结束时到期。
- 如果变量不是在块 ({}) 内声明的，但执行该动作列表时使用的是 `call()` 动作，则这些变量也是局部变量，它们在当前列表结束时到期。
- 如果变量不是在块中声明的，且执行当前动作列表时使用的不是 `call()` 动作，则这些变量被解释为时间轴变量。但是，您无需使用 `var` 来声明时间轴变量。

您不能将范围限于另一个对象的变量声明为局部变量：

```
my_array.length = 25;      // 正确
var my_array.length = 25;  // 语法错误
```

当使用 `var` 时，您可以精确键入变量，请参见第 33 页的“严格数据类型指定”

注意：在外部脚本中定义类还支持 `public`、`private` 和 `static` 变量范围。请参见第 137 页的第 9 章“使用动作脚本 2.0 创建类”以及 `private`、`public` 和 `static`。

Video 类

可用性

Flash Player 6；播放 Flash 视频 (FLV) 文件的功能已添加到 Flash Player 7 中。

说明

使用 Video 类可以直接在舞台上显示实时视频流，而无需将其嵌入您的 SWF 文件中。可以使用 `Camera.get()` 来捕获视频。在为 Flash Player 7 和更高版本发布的文件中，您还可以使用 Video 类通过 HTTP 或在本地文件系统中回放 Flash 视频 (FLV) 文件。有关更多信息，请参见第 174 页的“动态回放外部 FLV 文件”、`NetConnection` 类和 `NetStream` 类。

Video 对象的使用方式类似于影片剪辑。就像放置在舞台上的其它对象一样，您可以控制 Video 对象的多种属性。例如，可以通过使用 Video 对象的 `_x` 和 `_y` 属性在舞台上移动该对象；可以通过使用其 `_height` 和 `_width` 属性更改其大小等。

若要显示视频流，首先将 Video 对象放置在舞台上。然后使用 `Video.attachVideo()` 将视频流附加到 Video 对象。

将 Video 对象放置在舞台上：

- 1 如果 “库” 面板不可见，选择 “窗口” > “库” 以显示该面板。
- 2 单击 “库” 面板标题栏右侧的 “选项” 菜单，然后选择 “新建视频”，在库中添加一个嵌入的 Video 对象。
- 3 将该 Video 对象拖放到舞台上，然后使用属性检查器给它起一个唯一的实例名称，例如，my_video。（不要将其命名为 “Video”。）

Video 类的方法摘要

方法	说明
<code>Video.attachVideo()</code>	指定将在舞台上的 Video 对象的边界内显示的视频流。
<code>Video.clear()</code>	清除该 Video 对象中当前显示的图像。

Video 类的属性摘要

属性	说明
<code>Video.deblocking</code>	指定在视频进入流时，视频压缩器根据需要应用的解块过滤器的行为。
<code>Video.height</code>	只读；视频流的高度，以像素为单位。
<code>Video.smoothing</code>	指定在缩放视频时是否应对其进行平滑处理（插补数据）。
<code>Video.width</code>	只读；视频流的宽度，以像素为单位。

Video.attachVideo()

可用性

Flash Player 6 ；处理 Flash 视频 (FLV) 文件的功能已添加到 Flash Player 7 中。

用法

```
my_video.attachVideo(source)
```

参数

source 一个正在捕获视频数据或 NetStream 对象的 Camera 对象。若要切断与该 Video 对象的连接，可在 *source* 参数中传递 null。

返回

无。

说明

方法；指定将在舞台上的 Video 对象的边界内显示的视频流 (*source*)。视频流要么是通过 [NetStream.play\(\)](#) 命令显示的 FLV 文件（即 Camera 对象），要么是 null。如果 *source* 为 null，则该 Video 对象中将不再播放视频。

如果 FLV 文件只包含音频，则无需使用此方法；当发布 NetStream.play() 命令时，将自动播放 FLV 文件的音频部分。

如果要控制与 FLV 文件关联的音频，您可以使用 [MovieClip.attachAudio\(\)](#) 将音频路由到影片剪辑；然后可以创建 Sound 对象来控制音频的某些属性。有关更多信息，请参见 [MovieClip.attachAudio\(\)](#)。

示例

下面的示例在本地播放实时视频。

```
my_cam = Camera.get();  
my_video.attachVideo(my_cam); // my_video 是舞台上的一个 Video 对象
```

下面的示例播放一个以前录制的名为 myVideo.flv 的文件，该文件与 SWF 文件存储在同一个目录中。

```
var nc:NetConnection = new NetConnection();  
nc.connect(null);  
var ns:NetStream = new NetStream(my_nc);  
my_video.attachVideo(ns); // my_video 是舞台上的一个 Video 对象  
ns.play("myVideo.flv");
```

另请参见

[Camera 类](#), [NetStream 类](#)

Video.clear()

可用性

Flash Player 6。

用法

```
my_video.clear()
```

参数

无。

返回

无。

说明

方法；清除该 Video 对象中当前显示的图像。例如，如果要显示待机信息，则可以使用此方法。这样就不必隐藏该 Video 对象了。

另请参见

[Video.attachVideo\(\)](#)

Video.deblocking

可用性

Flash Player 6。

用法

```
my_video.deblocking  
my_video.deblocking = setting
```

说明

属性；指定在视频进入流时，视频压缩器根据需要应用的解决过滤器的行为。下面是 *setting* 可以接受的值：

- 0 （默认值）：让视频压缩器根据需要应用解决过滤器。
- 1: 从不使用解决过滤器。
- 2: 始终使用解决过滤器。

解决过滤器对整体回放性能有影响，而对于高带宽视频，通常不需要使用它。如果您的系统不够强大，在启用此过滤器的情况下回放视频可能会有困难。

Video.height

可用性

Flash Player 6。

用法

my_video.height

说明

只读属性；一个整数，指示该视频流的高度（以像素为单位）。对于实时流，此值与正在捕获该视频流的 `Camera` 对象的 `Camera.height` 属性相同。对于 FLV 文件，此值是以 FLV 形式导出的文件的高度。

例如，您可能需要使用此属性来确保用户以捕获时的相同大小观看视频，而无论 `Video` 对象在舞台上的实际大小是什么。

示例

用法 1：下面的示例设置 `Video` 对象的高度和宽度值以和 FLV 文件的值匹配。应该在调用 `NetStream.onStatus`（`code` 属性为 `NetStream.Buffer.Full`）之后调用此代码。如果在 `code` 属性为 `NetStream.Play.Start` 时调用此代码，则高度和宽度值将为 0，这是因为 `Video` 对象尚未具有加载的 FLV 文件的高度和宽度。

```
// Clip 是包含  
// 视频对象 “my_video” 的影片剪辑的实例名称。  
_root.Clip._width = _root.Clip.my_video.width;  
_root.Clip._height = _root.Clip.my_video.height;
```

用法 2：以下示例让用户通过按一个按钮来设置正在 Flash Player 中显示的视频流的高度和宽度，使其与捕获该视频流时的高度和宽度相同。

```
on (release) {  
    _root.my_video._width = _root.my_video.width  
    _root.my_video._height = _root.my_video.height  
}
```

另请参见

[MovieClip._height](#), [Video.width](#)

Video.smoothing

可用性

Flash Player 6。

用法

my_video.smoothing

说明

属性；一个布尔值，指定在缩放该视频时是否应进行平滑处理（插补数据）。播放器必须处于高品质模式，平滑处理才起作用。默认值为 `false`（不进行平滑处理）。

Video.width

可用性

Flash Player 6。

用法

`my_video.width`

说明

只读属性；一个整数，指示视频流的宽度（以像素为单位）。对于实时流，此值与正在捕获该视频流的 `Camera` 对象的 `Camera.width` 属性相同。对于 FLV 文件，此值是以 FLV 文件形式导出的文件的宽度。

例如，您可能需要使用此属性来确保用户以捕获时的相同大小观看视频，而无论 `Video` 对象在舞台上的实际大小是什么。

示例

请参见 `Video.height` 的示例。

void

可用性

Flash Player 5。

用法

`void (expression)`

说明

运算符；一个一元运算符，它将放弃 `expression` 值并返回未定义值。`void` 操作符经常用于使用 `==` 运算符测试未定义值的比较运算中。

while

可用性

Flash Player 4。

用法

```
while(condition) {  
    statement(s);  
}
```

参数

condition 每次执行 while 动作时都要重新计算的表达式。

statement(s) 条件计算结果为 true 时要执行的指令。

返回

无。

说明

语句；测试一个表达式，只要该表达式为 true，就重复运行循环中的语句或语句序列。

在运行该语句块之前，首先测试 *condition*；如果测试返回 true，则运行该语句块。如果该条件为 false，则跳过该语句块，并执行 while 动作语句块之后的第一条语句。

通常当计数器变量小于某指定值时，使用循环执行动作。在每个循环的结尾递增计数器的值，直到达到指定值为止。此时，*condition* 不再为 true，因此循环结束。

while 语句执行下面一系列步骤。第 1 步至第 4 步的每次重复，称作循环的一次迭代。每次迭代的开始将重新测试 *condition*，如下面的步骤所示：

- 1 计算表达式 *condition*。
- 2 如果 *condition* 的计算结果为 true，或是一个可转换为布尔值 true 的值（比如一个非零数字），则转到第 3 步。
否则，while 语句结束，并从 while 循环之后的下一语句继续执行。
- 3 运行语句块 *statement(s)*。
- 4 转到步骤 1。

另请参见

`do while`, `continue`, `for`, `for..in`

with

可用性

Flash Player 5。

用法

```
with (object) {  
    statement(s);  
}
```

参数

object 动作脚本对象或影片剪辑的一个实例。

statement(s) 包括在大括号中的一个动作或一组动作。

返回

无。

说明

语句；允许您使用 *object* 参数指定一个对象（比如影片剪辑），并使用 *statement(s)* 参数计算该对象中的表达式和动作。这可以使您不必重复书写对象的名称或路径。

object 参数将成为读取 *statement(s)* 参数中的属性、变量与函数的上下文。例如，如果 *object* 为 *my_array*，且指定的属性中有 *length* 和 *concat* 这样两个属性，则这些属性将自动作为 *my_array.length* 和 *my_array.concat* 进行读取。在另一个示例中，如果 *object* 为 *state.california*，则 *with* 动作中的任何动作或语句将从 *california* 实例中调用。

查找 *statement(s)* 参数中某个标识符的值时，动作脚本将从 *object* 指定的范围链的开头开始查找，并按照特定的顺序在范围链的每个级别中搜索该标识符。

with 动作使用范围链解析标识符，该范围链从下面列表中的第一项开始，一直到最后一项结束：

- 在最里面的 *with* 动作的 *object* 参数中指定的对象。
- 在最外面的 *with* 动作的 *object* 参数中指定的对象。
- 激活的对象。（当调用函数时自动创建的临时对象，该函数包含函数所调用的局部变量。）
- 包含当前执行脚本的影片剪辑。
- 全局对象（诸如 *Math* 与 *String* 的内置对象）。

若要在 *with* 动作中设置变量，该变量必须已在 *with* 动作外部进行了声明，或者您必须输入该变量所生存时间轴的完整路径。如果在 *with* 动作中设置了未声明的变量，则 *with* 动作将根据范围链查找该值。如果该变量尚不存在，则将在调用 *with* 动作的时间轴上设置此新值。

在 Flash 5 或更高版本中，*with* 动作替换了已不鼓励使用的 *tellTarget* 动作。建议您使用 *with* 而不使用 *tellTarget*，因为前者是对 ECMA-262 标准的一个标准动作脚本扩展功能。*with* 动作与 *tellTarget* 动作的主要区别在于：*with* 采用对影片剪辑或其它对象的引用作为其参数，而 *tellTarget* 采用标识影片剪辑的目标路径字符串作为其参数，且不能用于指向对象。

示例

下面的示例设置 *someOther_mc* 实例的 *_x* 和 *_y* 属性，然后指示 *someOther_mc* 转到第 3 帧并停止。

```
with (someOther_mc) {  
    _x = 50;
```

```

    _y = 100;
    gotoAndStop(3);
}

```

下面的代码片断显示不使用 `with` 动作如何编写上面的代码。

```

someOther_mc._x = 50;
someOther_mc._y = 100;
someOther_mc.gotoAndStop(3);

```

还可使用 `tellTarget` 动作编写这段代码。但是，如果 `someOther_mc` 不是影片剪辑，而是一个对象，则不能使用 `with` 动作。

```

tellTarget ("someOther_mc") {
    _x = 50;
    _y = 100;
    gotoAndStop(3);
}

```

`with` 动作对于同时访问一个范围链列表中的多项十分有用。在下面的示例中，内置 `Math` 对象被放置在范围链的前面。将 `Math` 设置为默认对象可以将标识符 `cos`、`sin` 和 `PI` 分别解析为 `Math.cos`、`Math.sin` 和 `Math.PI`。标识符 `a`、`x`、`y` 和 `r` 不是 `Math` 对象的方法或属性，但由于它们处于函数 `polar()` 的对象激活范围内，所以会将它们解析为相应的局部变量。

```

function polar(r) {
    var a, x, y;
    with (Math) {
        a = PI * r * r;
        x = r * cos(PI);
        y = r * sin(PI/2);
    }
    trace("area = " + a);
    trace("x = " + x);
    trace("y = " + y);
}

```

可以使用嵌套的 `with` 动作访问多重范围中的信息。在下面的示例中，实例 `fresno` 和实例 `salinas` 是实例 `california` 的子级。该语句设置 `fresno` 和 `salinas` 的 `_alpha` 值，而不改变 `california` 的 `_alpha` 值。

```

with (california){
    with (fresno){
        _alpha = 20;
    }
    with (salinas){
        _alpha = 40;
    }
}

```

另请参见

[tellTarget](#)

XML 类

可用性

Flash Player 5（已成为 Flash Player 6 本身的对象，Flash Player 6 大大提高了性能）。

说明

使用 XML 类的方法和属性可以加载、分析、发送、生成和操作 XML 文档树。

必须先使用构造函数 `new XML()` 创建 XML 对象，然后才能调用 XML 类的方法。

XML 类的方法摘要

方法	说明
<code>XML.setRequestHeader()</code>	添加或更改 POST 操作的 HTTP 标头。
<code>XML.appendChild()</code>	在指定对象的子级列表结尾追加一个节点。
<code>XML.cloneNode()</code>	克隆指定的节点，并可选择递归克隆所有子级。
<code>XML.createElement()</code>	创建一个新的 XML 元素。
<code>XML.createTextNode()</code>	创建一个新的 XML 文本节点。
<code>XML.getBytesLoaded()</code>	返回为指定 XML 文档加载的字节数。
<code>XML.getBytesTotal()</code>	返回 XML 文档的大小，以字节为单位。
<code>XML.hasChildNodes()</code>	如果指定的节点有子级节点，则返回 <code>true</code> ；否则返回 <code>false</code> 。
<code>XML.insertBefore()</code>	在指定节点的子级列表中的一个现有节点之前插入节点。
<code>XML.load()</code>	从 URL 中加载文档（由 XML 对象指定）。
<code>XML.parseXML()</code>	将 XML 文档分析为指定的 XML 对象树。
<code>XML.removeNode()</code>	从指定节点的父级中将该节点删除。
<code>XML.send()</code>	将指定的 XML 对象发送到 URL。
<code>XML.sendAndLoad()</code>	将指定的 XML 对象发送到 URL，并将服务器响应加载到另一 XML 对象中。
<code>XML.toString()</code>	将指定的节点与其任何子级转换为 XML 文本。

XML 类的属性摘要

属性	说明
<code>XML.contentType</code>	表示传输到服务器的 MIME 类型。
<code>XML.docTypeDecl</code>	设置和返回关于 XML 文档的 DOCTYPE 声明的信息。
<code>XML.firstChild</code>	只读；引用指定节点列表中的第一个子级。
<code>XML.ignoreWhite</code>	当设置为 <code>true</code> 时，在分析过程中将放弃仅包含空白的文本节点。
<code>XML.lastChild</code>	引用指定节点列表中的最后一个子级。
<code>XML.loaded</code>	只读；检查指定的 XML 对象是否已加载。

属性	说明
<code>XML.nextSibling</code>	只读；引用父级节点的子级列表中的下一个同级。
<code>XML.nodeName</code>	XML 对象的节点名称。
<code>XML.nodeType</code>	指定节点的类型（XML 元素或文本节点）。
<code>XML.nodeValue</code>	如果指定节点为文本节点，则为该节点的文本。
<code>XML.parentNode</code>	只读；引用指定节点的父级节点。
<code>XML.previousSibling</code>	只读；引用父级节点的子级列表中的前一个同级。
<code>XML.status</code>	一个数字状态代码，指示 XML 文档分析操作成功或失败。
<code>XML.xmlDecl</code>	指定有关文档的 XML 声明的信息。

XML 类的集合摘要

方法	说明
<code>XML.attributes</code>	返回一个包含指定节点所有属性的关联数组。
<code>XML.childNodes</code>	只读；返回一个包含对指定节点的子级节点的引用的数组。

XML 类的事件处理函数摘要

事件处理函数	说明
<code>XML.onData</code>	一个事件处理函数，当 XML 文本从服务器上被完全下载时，或者当从服务器上下载 XML 文本的过程中出现错误时进行调用。
<code>XML.onLoad()</code>	一个事件处理函数，它返回一个布尔值，指示是否用 <code>XML.load()</code> 或 <code>XML.sendAndLoad()</code> 成功加载了该 XML 对象。

XML 类的构造函数

可用性

Flash Player 5。

用法

```
new XML([source])
```

参数

source 对其进行分析以创建新的 XML 对象的 XML 文本。

返回

无。

说明

构造函数；创建一个新的 XML 对象。必须先使用构造函数创建一个 XML 对象，然后才能调用 XML 类的方法。

注意：`createElement()` 和 `createTextNode()` 方法是用于在 XML 文档树中创建元素和文本节点的“构造函数”方法。

示例

用法 1：下面的示例创建一个新的空 XML 对象。

```
my_xml = new XML();
```

用法 2：下面的示例通过分析 *source* 参数中指定的 XML 文本创建一个 XML 对象，并使用得到的 XML 文档树填充新创建的 XML 对象。

```
anyOtherXML = new XML("<state>California<city>san francisco</city></state>");
```

另请参见

[XML.createElement\(\)](#), [XML.createTextNode\(\)](#)

XML.setRequestHeader()

可用性

Flash Player 6。

用法

```
xml.setRequestHeader(headerName, headerValue)
```

```
xml.setRequestHeader(["headerName_1", "headerValue_1" ... "headerName_n",  
"headerValue_n"])
```

参数

headerName HTTP 请求标头名称。

headerValue 与 *headerName* 关联的值。

返回

无。

说明

方法，添加或更改用 POST 动作发送的 HTTP 请求标头（如 Content-Type 或 SOAPAction）。在第一种用法中，向该方法传递了两个字符串：*headerName* 和 *headerValue*。在第二种用法中，传递了字符串、替代标头名称和标头值的数组。

如果通过多次调用来设置相同的标头名称，则每个后继值将替换在上一次调用中设置的值。

您不能使用此方法添加或更改下列标准 HTTP 标头：Accept-Ranges、Age、Allow、Allowed、Connection、Content-Length、Content-Location、Content-Range、ETag、Host、Last-Modified、Locations、Max-Forwards、Proxy-Authenticate、Proxy-Authorization、Public、Range、Retry-After、Server、TE、Trailer、Transfer-Encoding、Upgrade、URI、Vary、Via、Warning 和 WWW-Authenticate。

示例

以下示例将值为 Foo 的自定义 HTTP 标头 SOAPAction 添加到名为 my_xml 的 XML 对象中。

```
my_xml.setRequestHeader("SOAPAction", "'Foo'");
```

下一个示例创建名为 headers 的数组，它包含两个替代 HTTP 标头及其关联值。然后将该数组作为参数传递给 addRequestHeader() 方法。

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];  
my_xml.setRequestHeader(headers);
```

另请参见

[LoadVars.setRequestHeader\(\)](#)

XML.appendChild()

可用性

Flash Player 5。

用法

```
my_xml.appendChild(childNode)
```

参数

childNode 要添加到指定 XML 对象的子级列表中的子级节点。

返回

无。

说明

方法；将指定的子级节点追加到 XML 对象的子级列表中。将追加的子级节点放入树结构中的同时，将该子级节点从其现有的父级节点（如果有的话）中删除。

示例

下面的示例从 doc1 中克隆最后一个节点，并将其追加到 doc2 中。

```
doc1 = new XML(src1);  
doc2 = new XML();  
node = doc1.lastChild.cloneNode(true);  
doc2.appendChild(node);
```

XML.attributes

可用性

Flash Player 5。

用法

```
my_xml.attributes
```

参数

无。

返回

一个数组。

说明

属性；一个包含指定 XML 对象的所有属性的关联数组。

示例

下面的示例将 XML 属性的名称写到“输出”窗口。

```
str = "<mytag name=\"Val\"> item </mytag>";
```



```
doc = new XML(str);
y = doc.firstChild.attributes.name;
    trace (y);
doc.firstChild.attributes.order = "first";
z = doc.firstChild.attributes.order
    trace(z);
```

下面的内容将被写到 “输出” 面板：

```
Val
first
```

XML.childNodes

可用性

Flash Player 5。

用法

```
my_xml.childNodes
```

参数

无。

返回

一个数组。

说明

属性（只读）；指定 XML 对象的子级组成的数组。数组中的每个元素都是对表示子级节点的 XML 对象的引用。这是一个只读属性，无法用于操作子级节点。若要操作子级节点，应使用 [XML.appendChild\(\)](#)、[XML.insertBefore\(\)](#) 和 [XML.removeNode\(\)](#)。

对于文本节点 (`nodeType == 3`)，此属性未定义。

另请参见

[XML.nodeType](#)

XML.cloneNode()

可用性

Flash Player 5。

用法

```
my_xml.cloneNode(deep)
```

参数

deep 布尔值，指定是否递归克隆指定 XML 对象的子级。

返回

一个 XML 节点。

说明

方法；构造并返回一个类型、名称、值和属性与指定 XML 对象均相同的新 XML 节点。如果 *deep* 设置为 `true`，则递归克隆所有子级节点，这将得到一个与原始对象文档树完全相同的副本。

返回的克隆节点与被克隆项目的树不再相关联。因此，`nextSibling`、`parentNode` 和 `previousSibling` 的值均为 `null`。如果不执行剪贴板复制，`firstChild` 和 `lastChild` 也为 `null`。

XML.contentType

可用性

Flash Player 6。

用法

```
my_xml.contentType
```

说明

属性；调用 `XML.send()` 或 `XML.sendAndLoad()` 方法时发送到服务器的 MIME 类型。默认值为 `application/x-www-form-urlencoded`。

另请参见

[XML.send\(\)](#), [XML.sendAndLoad\(\)](#)

XML.createElement()

可用性

Flash Player 5。

用法

```
my_xml.createElement(name)
```

参数

name 要创建的 XML 元素的标签名。

返回

一个 XML 元素。

说明

方法；使用参数中指定的名称创建一个新的 XML 元素。新元素开始时没有父级、子级和同级。该方法返回一个引用，该引用指向新创建的表示元素的 XML 对象。此方法与 `createTextNode()` 是为 XML 对象创建节点的构造函数方法。

XML.createTextNode()

可用性

Flash Player 5。

用法

```
my_xml.createTextNode(text)
```

参数

text 用于创建新文本节点的文本。

返回

无。

说明

方法；使用指定文本创建一个新的 XML 文本节点。该新节点开始时没有父级，且文本节点不能有子级或同级。这种方法返回对表示新文本节点的 XML 对象的引用。此方法与 `createElement()` 是为 XML 对象创建节点的构造函数方法。

XML.docTypeDecl

可用性

Flash Player 5。

用法

```
my_xml.XMLdocTypeDecl
```

说明

属性；指定有关 XML 文档的 DOCTYPE 声明的信息。在将 XML 文本分析为 XML 对象后，该 XML 对象的 XML.docTypeDecl 属性将被设置为该 XML 文档的 DOCTYPE 声明的文本。例如，`<!DOCTYPE greeting SYSTEM "hello.dtd">`。此属性是使用 DOCTYPE 声明的字符串表示形式设置的，而不是使用 XML 节点对象设置的。

动作脚本的 XML 分析器不是具有验证功能的分析器。分析器读取 DOCTYPE 声明，并将其存储在 docTypeDecl 属性中，但不执行 DTD 验证。

如果在执行分析操作的过程中未遇到 DOCTYPE 声明，XML.docTypeDecl 将被设置为未定义。

[XML.toString\(\)](#) 在输出完存储在 XML.xmlDecl 中的 XML 声明后，将立即输出 XML.docTypeDecl 的内容，然后才输出 XML 对象中的任何其它文本。如果 XML.docTypeDecl 未定义，则不输出 DOCTYPE 声明。

示例

下面的示例使用 XML.docTypeDecl 设置 XML 对象的 DOCTYPE 声明：

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

另请参见

[XML.toString\(\)](#), [XML.xmlDecl](#)

XML.firstChild

可用性

Flash Player 5。

用法

```
my_xml.firstChild
```

说明

属性（只读）；计算指定的 XML 对象，并引用父级节点的子级列表中的第一个子级。如果节点没有子级，则此属性为 null。如果节点为文本节点，则该属性未定义。这是只读属性，无法用于操作子级节点；请使用 [appendChild\(\)](#)、[insertBefore\(\)](#) 和 [removeNode\(\)](#) 来操作子级节点。

另请参见

[XML.appendChild\(\)](#), [XML.insertBefore\(\)](#), [XML.removeNode\(\)](#)

XML.getBytesLoaded()

可用性

Flash Player 6。

用法

```
XML.getBytesLoaded()
```

参数

无。

返回

一个整数，指示所加载的字节数。

说明

方法；返回为 XML 文档加载（进入流）的字节数。可以对 `getBytesLoaded()` 的值与 `getBytesTotal()` 的值进行比较，从而确定 XML 文档已加载的百分比。

另请参见

[XML.getBytesTotal\(\)](#)

XML.getBytesTotal()

可用性

Flash Player 6。

用法

```
XML.getBytesTotal()
```

参数

无。

返回

一个整数。

说明

方法；以字节为单位返回 XML 文档的大小。

另请参见

[XML.getBytesLoaded\(\)](#)

XML.hasChildNodes()

可用性

Flash Player 5。

用法

```
my_xml.hasChildNodes()
```

参数

无。

返回

一个布尔值。

说明

方法；如果指定的 XML 对象有子级节点，则返回 `true`；否则返回 `false`。

示例

下面的示例在用户定义的函数中使用来自 XML 对象的信息。

```
if (rootNode.hasChildNodes()) {  
    myfunc (rootNode.firstChild);  
}
```

XML.ignoreWhite

可用性

Flash Player 5。

用法

```
my_xml.ignoreWhite = boolean  
XML.prototype.ignoreWhite = boolean
```

参数

boolean 一个布尔值 (`true` 或 `false`)。

说明

属性；默认设置为 `false`。当设置为 `true` 时，在分析过程中将放弃仅包含空白的文本节点。带有前导或尾随空白的文本节点不受影响。

用法 1：可以为单个 XML 对象设置 `ignoreWhite` 属性，如下面的代码所示：

```
my_xml.ignoreWhite = true
```

用法 2：可以为 XML 对象设置默认 `ignoreWhite` 属性，如下面的代码所示：

```
XML.prototype.ignoreWhite = true
```

XML.insertBefore()

可用性

Flash Player 5。

用法

```
my_xml.insertBefore(childNodes, beforeNode)
```

参数

childNodes 要插入的节点。

beforeNode *childNodes* 插入点前面的节点。

返回

无。

说明

方法；在 XML 对象的子级列表中插入一个新的子级节点，插入位置在 *beforeNode* 节点之前。如果 *beforeNode* 参数未定义或为 null，将使用 `appendChild()` 添加该节点。如果 *beforeNode* 不是 *my_xml* 的子级，则插入将失败。

XML.lastChild

可用性

Flash Player 5。

用法

```
my_xml.lastChild
```

说明

属性（只读）；计算 XML 对象，并引用父级节点的子级列表中的最后一个子级。如果该节点没有子级，则此方法返回 null。这是只读属性，无法用于操作子级节点；请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 来操作子级节点。

另请参见

[XML.appendChild\(\)](#), [XML.insertBefore\(\)](#), [XML.removeNode\(\)](#)

XML.load()

可用性

Flash Player 5 ；行为在 Flash Player 7 中发生了变化。

用法

```
my_xml.load(url)
```

参数

url 要加载的 XML 文档所位于的 URL。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

返回

无。

说明

方法；从指定的 URL 中加载 XML 文档，并使用下载的 XML 数据替换指定 XML 对象的内容。该 URL 是相对的，并且通过 HTTP 调用。加载过程是异步的；它不会在 `load()` 方法执行后立即结束。

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 `someDomain.com` 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“Flash Player 安全功能”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

执行 `load()` 时，XML 对象的属性 `loaded` 将被设置为 `false`。在 XML 数据下载完毕后，`loaded` 属性将被设置为 `true`，并调用 `onLoad()` 方法。直到 XML 数据完全下载后，才开始分析。如果该 XML 对象以前包含任何 XML 树，它们将被放弃。

您可以指定自己的事件处理函数来代替 `onLoad()` 方法。

示例

下面是一个使用 `XML.load()` 的简单示例：

```
doc = new XML();
doc.load ("theFile.xml");
```

另请参见

[XML.loaded](#), [XML.onLoad\(\)](#)

XML.loaded

可用性

Flash Player 5。

用法

`my_xml.loaded`

说明

属性（只读）；确定由 `XML.load()` 调用启动的文档加载过程是否已完成。如果该过程成功完成，则此方法返回 `true`；否则返回 `false`。

示例

下面的示例在一个简单的脚本中使用 `XML.loaded`。

```
if (doc.loaded) {  
    gotoAndPlay(4);  
}
```

XML.nextSibling

可用性

Flash Player 5。

用法

`my_xml.nextSibling`

说明

属性（只读）；计算 XML 对象，并引用父级节点的子级列表中的下一个同级。如果该节点没有下一个同级节点，则此方法返回 `null`。这是一个只读属性，无法用于操作子级节点。请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 来操作子级节点。

另请参见

[XML.appendChild\(\)](#)、[XML.insertBefore\(\)](#)、[XML.removeNode\(\)](#)

XML.nodeName

可用性

Flash Player 5。

用法

`my_xml.nodeName`

说明

属性；XML 对象的节点名称。如果该 XML 对象是一个 XML 元素 (`nodeType == 1`)，则 `nodeName` 是 XML 文件中表示该节点的标签的名称。例如，`TITLE` 是 HTML 中 `TITLE` 标签的 `nodeName`。如果该 XML 对象是一个文本节点 (`nodeType == 3`)，则 `nodeName` 为 `null`。

另请参见

[XML.nodeType](#)

XML.nodeType

可用性

Flash Player 5。

用法

```
my_xml.nodeType
```

说明

属性（只读）；采用或返回一个 `nodeType` 值。值为 1 表示 XML 元素；值为 3 表示文本节点。

另请参见

[XML.nodeValue](#)

XML.nodeValue

可用性

Flash Player 5。

用法

```
my_xml.nodeValue
```

说明

属性；该 XML 对象的节点值。如果该 XML 对象是一个文本节点，则 `nodeType` 为 3，而 `nodeValue` 为该节点的文本。如果该 XML 对象是一个 XML 元素（节点类型为 1），则其 `nodeValue` 为 `null`，而且该属性为只读。

另请参见

[XML.nodeType](#)

XML.onData

可用性

Flash Player 5

用法

```
my_xml.onData = function(src) {  
    // 此处是您的语句  
}
```

参数

src 服务器发送的原始数据，通常为 XML 格式。

返回

无。

说明

事件处理函数；当 XML 文本从服务器上被完全下载后，或当从服务器上下载 XML 文本的过程中出现错误时调用该处理函数。在分析 XML 之前调用这个处理函数，因此它可用于调用一个自定义分析例程，而不必使用 Flash XML 分析器。XML.onData 方法或者返回 undefined 值，或者返回一个包含从服务器上下载的 XML 文本的字符串。如果返回值为 undefined，说明从服务器上下载 XML 时出现了错误。

默认情况下，XML.onData 方法将调用 XML.onLoad()。可以用您自己定义的行为覆盖 XML.onData 方法，但这时就不再调用 XML.onLoad() 了，除非在您的 XML.onData 实现中又调用了该方法。

示例

onData 方法的默认实现如下例所示：

```
XML.prototype.onData = function (src) {  
    if (src == undefined) {  
        this.onLoad(false);  
    } else {  
        this.parseXML(src);  
        this.loaded = true;  
        this.onLoad(true);  
    }  
}
```

可以覆盖 XML.onData 方法，从而截获 XML 文本而不对其进行分析。

XML.onLoad()

可用性

Flash Player 5。

用法

```
my_xml.onLoad = function (success) {  
    // 此处是您的语句  
}
```

参数

success 一个布尔值，指示 `XML.load()` 或 `XML.sendAndLoad()` 操作是否成功加载了该 XML 对象。

返回

无。

说明

事件处理函数；接收到来自服务器的 XML 文档时由 Flash Player 调用。如果成功地收到了 XML 文档，则 *success* 参数为 `true`。如果未收到该文档，或在接收来自服务器响应时出现错误，则 *success* 参数为 `false`。该方法的默认实现是不活动的。若要覆盖默认实现，必须指定一个包含您自己的动作的函数。

示例

下面的示例为一个简单的电子商务店面应用程序创建一个简单的 SWF 文件。 `sendAndLoad()` 方法传输包含用户名和密码的 XML 元素，并设置一个 `onLoad` 处理函数来处理服务器的应答。

```
function myOnLoad(success) {  
    if (success){  
        if (e.firstChild.nodeName == "LOGINREPLY_xml" &&  
            e.firstChild.attributes.status == "OK") {  
            gotoAndPlay("loggedIn")  
        } else {  
            gotoAndStop("loginFailed")  
        }  
    } else {  
        gotoAndStop("connectionFailed")  
    }  
}  
var myLoginReply_xml = new XML();  
myLoginReply_xml.onLoad = myOnLoad;  
my_xml.sendAndLoad("http://www.samplestore.com/login.cgi",  
    myLoginReply_xml);
```

另请参见

[function](#), [XML.load\(\)](#), [XML.sendAndLoad\(\)](#)

XML.parentNode

可用性

Flash Player 5。

用法

```
my_xml.parentNode
```

说明

属性（只读）；引用指定 XML 对象的父级节点，如果该节点没有父级，则返回 `null`。这是一个只读属性，无法用于操作子级节点；请使用 `appendChild()`、`insertBefore()` 和 `removeNode()` 来操作子级。

XML.parseXML()

可用性

Flash Player 5。

用法

```
my_xml.parseXML(source)
```

参数

source 要分析并传递给指定 XML 对象的 XML 文本。

返回

无。

说明

方法；分析 *source* 参数中指定的 XML 文本，并使用得到的 XML 树填充指定的 XML 对象。XML 对象中任何现有的树将被放弃。

XML.previousSibling

可用性

Flash Player 5。

用法

```
my_xml.previousSibling
```

说明

属性（只读）；返回一个对父级节点的子级列表中的前一个同级的引用。如果该节点没有前一个同级节点，则此属性的值为 `null`。这是一个只读属性，不能用于操作子级节点；请使用 `XML.appendChild()`、`XML.insertBefore()` 和 `XML.removeNode()` 来操作子级节点。

XML.removeNode()

可用性

Flash Player 5。

用法

```
my_xml.removeNode()
```

参数

无。

返回

无。

说明

方法；从指定 XML 对象的父级中将该对象删除。该节点的所有后代也将被删除。

XML.send()

可用性

Flash Player 5。

用法

```
my_xml.send(url, [window])
```

参数

url 指定 XML 对象的目标 URL。

window 显示服务器返回数据的浏览器窗口：_self 指定当前窗口中的当前帧，_blank 指定一个新窗口，_parent 指定当前帧的父级，_top 指定当前窗口中的顶级帧。此参数是可选的；如果未指定 *window* 参数，则与指定 _self 相同。

返回

无。

说明

方法；将指定的 XML 对象编码为 XML 文档，并使用 POST 方法将其发送到指定的 URL。

XML.sendAndLoad()

可用性

Flash Player 5 ；行为在 Flash Player 7 中发生了变化。

用法

```
my_xml.sendAndLoad(url, targetXMLObject)
```

参数

url 指定 XML 对象的目标 URL。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

targetXMLObject 一个由 XML 构造函数方法创建的 XML 对象，该对象将接收来自服务器的返回信息。

返回

无。

说明

方法；将指定的 XML 对象编码为 XML 文档，并使用 POST 方法将其发送到指定的 URL，下载服务器的响应，然后将其加载到参数中指定的 *targetXMLObject* 中。服务器响应的加载方式与 `load()` 方法使用的方式相同。

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 `someDomain.com` 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“Flash Player 安全功能”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件。有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

执行 `load()` 时，XML 对象的属性 `loaded` 将被设置为 `false`。在 XML 数据下载完毕后，`loaded` 属性将被设置为 `true`，并调用 `onLoad()` 方法。直到 XML 数据完全下载后，才开始分析。如果该 XML 对象以前包含任何 XML 树，它们将被放弃。

另请参见

[XML.load\(\)](#)

XML.status

可用性

Flash Player 5。

用法

```
my_xml.status
```

说明

属性；自动设置并返回一个数值，该数值指示 XML 文档是否成功地分析为 XML 对象。这些数字状态代码以及每个代码的说明列表如下：

- 0 没有错误；成功地完成了分析。
- -2 一个 CDATA 部分没有正确结束。
- -3 XML 声明没有正确结束。
- -4 DOCTYPE 声明没有正确结束。
- -5 一个注释没有正确结束。
- -6 一个 XML 元素有格式错误。
- -7 内存不足。
- -8 一个属性值没有正确结束。
- -9 一个开始标签没有匹配的结束标签。
- -10 遇到一个没有匹配的开始标签的结束标签。

XML.toString()

可用性

Flash Player 5。

用法

```
my_xml.toString()
```

参数

无。

返回

字符串。

说明

方法；计算指定的 XML 对象；构造一个包括节点、子级和属性的 XML 结构的文本表示形式，并以字符串形式返回结果。

对于顶级 XML 对象（那些用构造函数创建的对象），XML.toString() 首先输出文档的 XML 声明（存储在 XML.xmlDecl 中），随后输出文档的 DOCTYPE 声明（存储在 XML.docTypeDecl 中），然后输出该对象中所有 XML 节点的文本表示形式。如果 XML.xmlDecl 未定义，则不输出 XML 声明。如果 XML.docTypeDecl 未定义，则不输出 DOCTYPE 声明。

示例

下面的代码是 XML.toString() 方法的一个示例，该示例将 <h1>test</h1> 发送到 “输出” 面板。

```
node = new XML("<h1>test</h1>");
trace(node.toString());
```

另请参见

[XML.docTypeDecl](#), [XML.xmlDecl](#)

XML.xmlDecl

可用性

Flash Player 5。

用法

```
my_xml.xmlDecl
```

说明

属性；指定有关文档的 XML 声明的信息。将 XML 文档分析为 XML 对象之后，该属性被设置为文档的 XML 声明的文本。使用 XML 声明的字符串表示形式而不是 XML 节点对象设置该属性。如果在分析操作过程中未遇到 XML 声明，则该属性将被设置为 undefined。XML.toString() 方法首先输出 XML.xmlDecl 的内容，然后才输出 XML 对象中的其它文本。如果 XML.xmlDecl 包含 undefined 类型，则不输出 XML 声明。

示例

下面的示例使用 XML.xmlDecl 为一个 XML 对象设置 XML 文档声明。

```
my_xml.xmlDecl = "<?xml version=\"1.0\" ?>";
```

下面是一个 XML 声明的示例：

```
<?xml version="1.0" ?>
```

另请参见

[XML.docTypeDecl](#), [XML.toString\(\)](#)

XMLNode 类

可用性

Flash Player 5。

说明

XMLNode 类支持以下属性、方法和集合；有关它们的用法的信息，请参见相应的 XML 类条目。

属性、方法或集合	相应的 XML 类条目
appendChild()	XML.appendChild()
attributes	XML.attributes
childNodes	XML.childNodes

属性、方法或集合	相应的 XML 类条目
<code>cloneNode()</code>	XML.cloneNode()
<code>firstChild</code>	XML.firstChild
<code>hasChildNodes()</code>	XML.hasChildNodes()
<code>insertBefore()</code>	XML.insertBefore()
<code>lastChild</code>	XML.lastChild
<code>nextSibling</code>	XML.nextSibling
<code>nodeName</code>	XML.nodeName
<code>nodeType</code>	XML.nodeType
<code>nodeValue</code>	XML.nodeValue
<code>parentNode</code>	XML.parentNode
<code>previousSibling</code>	XML.previousSibling
<code>removeNode()</code>	XML.removeNode()
<code>toString()</code>	XML.toString()

另请参见

[XML 类](#)

XMLSocket 类

可用性

Flash Player 5。

说明

XMLSocket 类用于实现客户端套接字。利用客户端套接字，运行 Flash Player 的计算机可以与由 IP 地址或域名标识的服务器计算机进行通讯。对于要求滞后时间较短的客户端 / 服务器应用程序，如实时聊天系统，XMLSocket 类非常有用。传统的基于 HTTP 的聊天解决方案频繁轮询服务器，并使用 HTTP 请求来下载新的消息。与此相反，XMLSocket 聊天解决方案保持与服务器的开放连接，这一连接允许服务器即时发送传入的消息，而无需客户端发出请求。

若要使用 XMLSocket 类，服务器计算机必须运行可识别 XMLSocket 类使用的协议的守护程序。协议如下所示：

- XML 消息通过全双工 TCP/IP 流套接字连接发送。
- 每个 XML 消息都是一个完整的 XML 文档，以零字节结束。
- 通过一个 XMLSocket 连接发送和接收的 XML 消息数没有限制。

XMLSocket 对象连接到服务器的方式和位置受下列限制：

- `XMLSocket.connect()` 方法只能连接到端口号大于或等于 1024 的 TCP 端口。这种限制的一个后果是，与 XMLSocket 对象通讯的服务器守护程序也必须分配到端口号大于或等于 1024 的端口。端口号小于 1024 的端口通常由系统服务（如 FTP、Telnet 和 HTTP）使用，因此，出于安全方面的考虑，禁止 XMLSocket 对象使用这些端口。这种端口号方面的限制可以减少不恰当地访问和滥用这些资源的可能性。

- XMLSocket.connect() 方法只能连接到 SWF 文件所在域中的计算机。这一限制不适用于在本地磁盘外运行的 SWF 文件。（这一限制与 loadVariables()、XML.sendAndLoad() 和 XML.load() 的安全规则相同。）若要连接到在 SWF 所在域之外的其它域中运行的服务器守护程序，可以在该服务器上创建一个允许从特定域进行访问的安全策略文件。有关为 XMLSocket 连接创建策略文件的更多信息，请参见第 168 页的“关于允许跨域数据加载”。

将服务器设置为与 XMLSocket 对象进行通讯可能会遇到一些困难。如果您的应用程序不需要进行实时交互，请使用 loadVariables() 动作或 Flash 的基于 HTTP 的 XML 服务器连接（XML.load()、XML.sendAndLoad()、XML.send()），而不要使用 XMLSocket 类。

若要使用 XMLSocket 类的方法，您必须首先使用构造函数 new XMLSocket 创建一个新的 XMLSocket 对象。

XMLSocket 类的方法摘要

方法	说明
<code>XMLSocket.close()</code>	关闭一个打开的套接字连接。
<code>XMLSocket.connect()</code>	建立一个到指定服务器的连接。
<code>XMLSocket.send()</code>	向服务器发送一个 XML 对象。

XMLSocket 类的事件处理函数摘要

事件处理函数	说明
<code>XMLSocket.onClose()</code>	当 XMLSocket 连接关闭时调用的事件处理函数。
<code>XMLSocket.onConnect()</code>	一个事件处理函数，在通过 XMLSocket.connect() 启动的连接请求成功或失败后，Flash Player 将调用此函数。
<code>XMLSocket.onData()</code>	当 XML 消息已从服务器上下载后调用的事件处理函数。
<code>XMLSocket.onXML()</code>	当 XML 对象从服务器到达时调用的事件处理函数。

XMLSocket 类的构造函数

可用性

Flash Player 5。

用法

`new XMLSocket()`

参数

无。

返回

无。

说明

构造函数；创建一个新的 XMLSocket 对象。XMLSocket 对象开始时未与任何服务器连接。必须调用 `XMLSocket.connect()` 将该对象连接到服务器。

XMLSocket.close()

可用性

Flash Player 5。

用法

```
myXMLSocket.close()
```

参数

无。

返回

无。

说明

方法；关闭 XMLSocket 对象指定的连接。

另请参见

[XMLSocket.connect\(\)](#)

XMLSocket.connect()

可用性

Flash Player 5；行为在 Flash Player 7 中发生了变化。

用法

```
myXMLSocket.connect(host, port)
```

参数

host 一个完全限定 DNS 域名，或一个 aaa.bbb.ccc.ddd 形式的 IP 地址。也可指定 `null`，这将连接到 SWF 文件所在的主机服务器。如果发布此调用的 SWF 文件运行在 Web 浏览器上，则 *url* 必须与 SWF 文件位于同一个域中；有关详细信息，请参见下面的“说明”。

port 主机上用于建立连接的 TCP 端口号。端口号必须为 1024 或更高。

返回

一个布尔值。

说明

方法；使用指定的 TCP 端口（必须为 1024 或更大）建立一个到指定 Internet 主机的连接，并根据是否成功建立了连接，返回 `true` 或 `false`。如果您不知道 Internet 宿主计算机的端口号，请与您的网络管理员联系。

如果将 *host* 参数指定为 `null`，则所联系的主机为调用 `XMLSocket.connect()` 的 SWF 文件所在的主机。例如，如果 SWF 文件是从 `http://www.yoursite.com` 下载的，则将 *host* 参数指定为 `null` 与输入 `www.yoursite.com` 的 IP 地址效果相同。

在运行于 Flash Player 7 以前版本的播放器的 SWF 文件中，*url* 必须与发布此调用的 SWF 文件位于同一个超级域中。例如，位于 `www.someDomain.com` 的 SWF 文件可以从位于 `store.someDomain.com` 的 SWF 文件加载变量，这是因为这两个文件都在同一个超级域 `someDomain.com` 中。

如果任何版本的 SWF 文件运行在 Flash Player 7 或更高版本中，*url* 必须处于完全相同的域中（请参见第 166 页的“Flash Player 安全功能”）。例如，位于 `www.someDomain.com` 的 SWF 文件只能从同样位于 `www.someDomain.com` 的 SWF 文件加载变量。如果要从其它域中加载变量，则可以在承载被访问的 SWF 文件的服务器上放置一个跨域策略文件（必须将其放置在运行于与套接字服务器所在的同一个域中端口 80 上的 HTTP 服务器上）。有关更多信息，请参见第 168 页的“关于允许跨域数据加载”。

执行 `load()` 时，XML 对象的属性 `loaded` 将被设置为 `false`。在 XML 数据下载完毕后，`loaded` 属性将被设置为 `true`，并调用 `onLoad()` 方法。直到 XML 数据完全下载后，才开始分析。如果该 XML 对象以前包含任何 XML 树，它们将被放弃。

如果 `XMLSocket.connect()` 返回 `true` 值，则表示连接过程的初始阶段是成功的；随后将调用 `XMLSocket.onConnect` 方法以确定最终连接是否成功。如果 `XMLSocket.connect()` 返回 `false`，则表示未能建立连接。

示例

下面的示例使用 `XMLSocket.connect()` 连接到 SWF 文件所在的主机，然后使用 `trace` 显示返回值，指示连接是否成功。

```
function myOnConnect(success) {
    if (success){
        trace ("Connection succeeded!")
    } else {
        trace ("Connection failed!")
    }
}
socket = new XMLSocket()
socket.onConnect = myOnConnect
if (!socket.connect(null, 2000)) {
    trace ("Connection failed!")
}
```

另请参见

[function, XMLSocket.onConnect\(\)](#)

XMLSocket.onClose()

可用性

Flash Player 5。

用法

```
myXMLSocket.onClose() = function() {
    // 此处是您的语句
}
```

参数

无。

返回

无。

说明

事件处理函数；只有当某个打开的连接被服务器关闭时才调用此函数。此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含您自己的动作的函数。

另请参见

[function, XMLSocket.onConnect\(\)](#)

XMLSocket.onConnect()

可用性

Flash Player 5。

用法

```
myXMLSocket.onConnect(success)
// 此处是您的语句
}
```

参数

success 一个布尔值，指示是否成功建立了套接字连接（true 或 false）。

返回

无。

说明

事件处理函数；在通过 [XMLSocket.connect\(\)](#) 启动的连接请求成功或失败后，Flash Player 将调用该函数。如果连接成功，则 *success* 参数为 true；否则 *success* 参数为 false。

此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含您自己的动作的函数。

示例

下面的示例说明在一个简单的聊天应用程序中为 onConnect 方法指定替换函数的过程。

该函数根据是否成功建立了连接，控制为用户显示哪一屏幕。如果连接成功建立，则为用户显示标记为 startChat 的帧上的主聊天屏幕。如果连接不成功，则为用户显示标记为 connectionFailed 的帧上的带到疑难解答信息的屏幕。

```
function myOnConnect(success) {
    if (success) {
        gotoAndPlay("startChat")
    } else {
        gotoAndStop("connectionFailed")
    }
}
```

在使用构造函数方法创建了 XMLSocket 对象后，脚本使用赋值运算符设置 onConnect 方法：

```
socket = new XMLSocket();
socket.onConnect = myOnConnect;
```

最后，启动连接。如果 `connect()` 返回 `false`，则 SWF 文件将直接跳转到标记为 `connectionFailed` 的帧，且永远不会调用 `onConnect`。如果 `connect()` 返回 `true`，则 SWF 文件将跳转到标记为 `waitForConnection` 的帧，即“请稍候”屏幕。SWF 文件将停留在 `waitForConnection` 帧，直到调用 `onConnect` 处理函数。该函数将在以后某个时刻执行，具体取决于网络的滞后时间。

```
if (!socket.connect(null, 2000)) {
    gotoAndStop("connectionFailed")
} else {
    gotoAndStop("waitForConnection")
}
```

另请参见

[function, XMLSocket.connect\(\)](#)

XMLSocket.onData()

可用性

Flash Player 5。

用法

```
XMLSocket.onData = function(src) {
    // 此处是您的语句
}
```

参数

src 一个字符串，包含服务器发送的数据。

返回

无。

说明

事件处理函数；当从服务器上下载完消息时调用，以零字节结束。您可以覆盖 `XMLSocket.onData` 以截获服务器发送的数据，而不将其分析为 XML。如果您传输的是任意格式的数据包，而且希望在数据到达时直接操纵这些数据，而不让 Flash Player 将数据分析为 XML，则上述方法会非常有用。

默认情况下，`XMLSocket.onData` 方法调用 `XMLSocket.onXML` 方法。如果用您自己的行为覆盖了 `XMLSocket.onData`，则将不再调用 `XMLSocket.onXML`，除非在您的 `XMLSocket.onData` 实现中又调用了该方法。

```
XMLSocket.prototype.onData = function (src) {
    this.onXML(new XML(src));
}
```

在上面的示例中，*src* 参数是一个包含从服务器上下载的 XML 文本的字符串。零字节终止符不包含在该字符串中。

XMLSocket.onXML()

可用性

Flash Player 5。

用法

```
myXMLSocket.onXML(object) = function() {  
    // 此处是您的语句  
}
```

参数

object 一个 XML 对象，它包含从服务器上接收到的已经过分析的 XML 文档。

返回

无。

说明

事件处理函数；在包含 XML 文档的指定 XML 对象通过打开的 XMLSocket 连接到达时，Flash Player 将调用此函数。XMLSocket 连接可用于在客户端与服务器之间传输数量无限的 XML 文档。每个文档都以 0（零）字节终止。当 Flash Player 收到 0 字节时，它分析自从前一个 0 字节以来收到的所有 XML，如果这是收到的第一条消息，则分析建立连接以来收到的所有 XML。每一批经过分析的 XML 都被视作单个 XML 文档，并被传递给 onXML 方法。

此方法的默认实现不执行任何动作。若要覆盖默认实现，必须指定一个包含您定义的动作的函数。

示例

下面的函数在一个简单的聊天应用程序中覆盖 onXML 方法的默认实现。函数 myOnXML 指示该聊天应用程序识别一个 XML 元素 (MESSAGE)，该元素的格式如下所示。

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.
```

必须首先在该 XMLSocket 对象中设置 onXML 处理函数，如下所示：

```
socket.onXML = myOnXML;
```

假定函数 displayMessage() 是一个用户定义的函数，该函数显示用户收到的消息。

```
function myOnXML(doc) {  
    var e = doc.firstChild;  
    if (e != null && e.nodeName == "MESSAGE") {  
        displayMessage(e.attributes.user, e.attributes.text);  
    }  
}
```

另请参见

[function](#)

XMLSocket.send()

可用性

Flash Player 5。

用法

myXMLSocket.send(object)

参数

object 要传输到服务器的 XML 对象或其它数据。

返回

无。

说明

方法；将在 *object* 参数中指定的 XML 对象或数据转换成一个字符串，并将其传输到服务器，后面跟有一个零字节。如果 *object* 是一个 XML 对象，则该字符串是这个 XML 对象的 XML 文本表示形式。发送操作是异步的；它将立即返回，但数据可能会以后传输。

XMLSocket.send() 方法不返回指示数据是否成功传输的值。

如果 *myXMLSocket* 对象未连接到服务器（使用 [XMLSocket.connect\(\)](#)），则 XMLSocket.send() 操作将失败。

示例

下面的示例说明如何指定用户名和密码，并将 XML 对象 *my_xml* 发送到服务器：

```
loginReplyXML = new XML();
var myLogin = my_xml.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
my_xml.appendChild(myLogin);
myXMLSocket.send(my_xml);
```

另请参见

[XMLSocket.connect\(\)](#)

附录 A

错误消息

如果您在发布文件时指定动作脚本 2.0（默认设置），Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 将提供增强的编译时错误报告。下表列出了 Flash 编译器能够生成的错误消息。

错误编号	消息文本
1093	需要类名。
1094	“extends” 关键字后应该为基类名称。
1095	成员属性的使用有误。
1096	同一个成员名称不能重复多次。
1097	所有成员函数都需要有名称。
1099	类定义中不允许使用此语句。
1100	已经定义了一个具有此名称的类或接口。
1101	类型不匹配。
1102	没有名称为 “<ClassName>” 的类。
1103	没有名称为 “<propertyName>” 的属性。
1104	尝试在非函数上执行函数调用。
1105	赋值语句中类型不匹配。找到 [lhs-type]，但需要 [rhs-type]。
1106	该成员是私有成员，不能访问。
1107	不允许在接口中声明变量。
1108	不允许在接口中声明事件。
1109	不允许在接口中声明 getter/setter 函数。
1110	不允许在接口中定义私有成员。
1111	不允许在接口中定义函数体。
1112	类不能扩展其本身。
1113	接口不能扩展其本身。
1114	未定义具有此名称的接口。

错误编号	消息文本
1115	类不能扩展接口。
1116	接口不能扩展类。
1117	“implements” 关键字后应该为接口名。
1118	类不能实现类，而只能实现接口。
1119	该类必须从接口 “interfaceName” 实现方法 “methodName”。
1120	接口方法的实现必须为方法，而不能为属性。
1121	类不能多次扩展同一接口。
1122	接口方法的实现不符合其定义。
1123	此构造仅可用于动作脚本 1.0。
1124	此构造仅在动作脚本 2.0 中可用。
1125	接口中不允许定义静态成员。
1126	返回的表达式必须与该函数的返回类型匹配。
1127	此函数中需要一个 return 语句。
1128	在类的外面使用了属性。
1129	返回类型为 Void 的函数不能返回值。
1130	“extends” 子句必须出现在 “implements” 子句的前面。
1131	“:” 后应该有类型标识符。
1132	接口必须使用 “extends” 关键字，而不能使用 “implements” 关键字。
1133	一个类不能扩展多个类。
1134	一个接口不能扩展多个接口。
1135	没有名称为 “<methodName>” 的方法。
1136	接口定义中不允许使用此语句。
1137	设置函数要求只带有一个参数。
1138	获取函数要求不带有任何参数。
1139	类只能在外部动作脚本 2.0 类脚本中定义。
1140	动作脚本 2.0 类脚本只能定义类或接口构造。
1141	此类的名称 (<A.B.C>) 与已加载的另一个类的名称 (<A.B>) 冲突。
1142	无法加载类 “<ClassName>”。
1143	接口只能在外部动作脚本 2.0 类脚本中定义。
1144	不能在静态函数中访问实例变量。
1145	类和接口定义不能嵌套。
1146	所引用的属性没有 static 特性。
1147	对父项的调用与父构造函数不匹配。

错误编号	消息文本
1148	接口方法只允许有 public 特性。
1149	Import 关键字不能用作指令。
1150	必须将您的影片导出为 Flash 7 格式才能使用此动作。
1151	必须将您的影片导出为 Flash 7 格式才能使用此表达式。
1152	此异常子句的位置不正确。
1153	类必须只有一个构造函数。
1154	构造函数不能返回值。
1155	构造函数不能指定返回类型。
1156	变量的类型不能为 Void。
1157	函数参数的类型不能为 Void。
1158	静态成员只能直接通过类访问。
1159	实现的多个接口包含相同的方法，但类型不同。
1160	已经有一个用此名称定义类或接口。
1161	不能删除类、接口或内置类型。
1162	没有具有此名称的类。
1163	关键字 “<keyword>” 是动作脚本 2.0 的保留字，不能在此处使用。
1164	自定义属性定义没有结束。
1165	每个动作脚本 2.0 .as 文件中只能定义一个类或接口。
1166	正在编译的类 (<A.b>) 与导入的类 (<A.B>) 不匹配。
1167	必须输入类名。
1168	输入类名中有语法错误。
1169	输入接口名中有语法错误。
1170	输入的基类名称中有语法错误。
1171	输入的基接口名称中有语法错误。
1172	必须输入接口名。
1173	必须输入类名或接口名。
1174	输入类名或接口名中有语法错误。
1175	不能从此范围访问 “variable”。
1176	“get/set/private/public/static” 属性出现多次。
1177	类属性的使用有误。
1178	实例变量和函数不能用于初始化静态变量。
1179	在以下类之间发现运行时循环：%1
1180	当前的目标 Flash Player 不支持调试。

错误编号	消息文本
1181	当前的目标 Flash Player 不支持 releaseOutside 事件。
1182	当前的目标 Flash Player 不支持 dragOver 事件。
1183	当前的目标 Flash Player 不支持 dragOut 事件。
1184	当前的目标 Flash Player 不支持拖放动作。
1185	当前的目标 Flash Player 不支持 loadMovie 动作。
1186	当前的目标 Flash Player 不支持 getURL 动作。
1187	当前的目标 Flash Player 不支持 FSCommand 动作。
1188	类定义或接口定义中不允许使用 Import 语句。
1189	不能导入类 “<A.B>”，因为其叶名称已经解析为正在定义的类 “<C.B>”。
1190	不能导入类 “<A.B>”，因为其叶名称已经解析为已导入的类 “<C.B>”。
1191	类的实例变量只能初始化为编译时常量表达式。
1192	类成员函数的名称不能与超类的构造函数的名称相同。
1193	此类的名称 (<ClassName>) 与已加载的另一个类的名称冲突。
1194	必须在构造函数体内首先调用超构造函数。
1195	标识符 “<className>” 在运行时不会解析为内置对象 “<ClassName>”。
1196	类 “<A.B.ClassName>” 必须在相对路径为 “<A.B>” 的文件中定义。
1197	通配符 “*” 在类名 “<ClassName>” 中的使用有误。
1198	成员函数 “<classname>” 的大小写与正在定义的类的名称 (<ClassName>) 不同，因此在运行时不会被视作该类的构造函数。
1199	for-in 循环迭代变量的类型只能为 String。
1200	构造函数不能返回值。
1201	构造函数只能具有 public 和 private 属性。

附录 B

运算符的优先级和结合律

下表列出了所有动作脚本运算符及其结合律，按优先级从高到低排列。

运算符	说明	结合律
最高优先级		
+	一元加号	从右到左
-	一元减号	从右到左
~	按位 “非”	从右到左
!	逻辑 “非”	从右到左
not	逻辑 “非” (Flash 4 样式)	从右到左
++	后递增	从左到右
--	后递减	从左到右
()	函数调用	从左到右
[]	数组元素	从左到右
.	结构成员	从左到右
++	前递增	从右到左
--	前递减	从右到左
new	分配对象	从右到左
delete	取消分配对象	从右到左
typeof	对象类型	从右到左
void	返回未定义值	从右到左
*	乘号	从左到右
/	除号	从左到右
%	求模	从左到右
+	加号	从左到右
add	字符串连接 (原为 &t)	从左到右

运算符	说明	结合律
-	减号	从左到右
<<	按位左移位	从左到右
>>	按位右移位	从左到右
>>>	按位右移位（无符号）	从左到右
<	小于	从左到右
<=	小于或等于	从左到右
>	大于	从左到右
>=	大于或等于	从左到右
instanceof	是否为其实例	从左到右
lt	小于（字符串版本）	从左到右
le	小于或等于（字符串版本）	从左到右
gt	大于（字符串版本）	从左到右
ge	大于或等于（字符串版本）	从左到右
==	等于	从左到右
!=	不等于	从左到右
eq	等于（字符串版本）	从左到右
ne	不等于（字符串版本）	从左到右
&	按位“与”	从左到右
^	按位“异或”	从左到右
	按位“或”	从左到右
&&	逻辑“与”	从左到右
and	逻辑“与” (Flash 4)	从左到右
	逻辑“或”	从左到右
or	逻辑“或” (Flash 4)	从左到右
?:	条件	从右到左
=	赋值	从右到左
*=, /=, %=, +=, -=, &=, =, ^=, <<=, >>=, >>>=	复合赋值	从右到左
,	逗号	从左到右

最低优先级

附录 C

键盘键和键控代码值

下面的表中列出了标准键盘上的所有键，及其相应的 ASCII 键控代码值，这些值用于在动作脚本中标识键。有关更多信息，请参见第 181 页的第 12 章 “动作脚本字典” 中的 [Key](#) 类条目。

字母 A 到 Z 和标准数字 0 到 9

下表列出了标准键盘上字母 A 到 Z 和数字 0 到 9 的键，及其相应的 ASCII 键控代码值，这些值用于在动作脚本中标识这些键。

字母或数字键	键控代码
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82

字母或数字键	键控代码
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

数字键盘上的键

下表列出了数字键盘上的键，及其相应的 ASCII 键控代码值，这些值用于在动作脚本中标识这些键。

数字键盘键	键控代码
数字键盘 0	96
数字键盘 1	97
数字键盘 2	98
数字键盘 3	99
数字键盘 4	100
数字键盘 5	101
数字键盘 6	102
数字键盘 7	103
数字键盘 8	104
数字键盘 9	105

数字键盘键	键控代码
乘号	106
加号	107
Enter	108
减号	109
小数点	110
除号	111

功能键

下表列出了标准键盘上的功能键，及其相应的 ASCII 键控代码值，这些值用于在动作脚本中标识这些键。

功能键	键控代码
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
F13	124
F14	125
F15	126

其它键

下表列出了标准键盘上除了字母、数字、数字键盘键和功能键之外的其它键，及其相应的 ASCII 键控代码值，这些值用于在动作脚本中标识这些键。

Key	键控代码
Backspace	8
Tab	9
Clear	12

Key	键控代码
Enter	13
Shift	16
Control	17
Alt	18
Caps Lock	20
Esc	27
空格键	32
Page Up	33
Page Down	34
End	35
Home	36
左箭头	37
向上箭头	38
右箭头	39
向下箭头	40
Insert	45
Delete	46
Help	47
Num Lock	144
; :	186
= +	187
- _	189
/ ?	191
` ~	192
[{	219
\	220
] }	221
" '	222

附录 D

为早期的 Flash Player 版本编写脚本

Macromedia Flash MX 2004 和 Macromedia Flash MX Professional 2004 发布后，动作脚本有了相当大的变化。为 Flash Player 7 创建内容时，您可以利用动作脚本的全部功能。虽然仍可以使用 Flash MX 2004 为早期的 Flash Player 版本创建内容，但您将无法使用动作脚本的所有元素。

本章为您提供指导，帮助您编写在句法上符合目标播放器版本的脚本。

关于以早期的 Flash Player 版本作为目标播放器

编写脚本时，可以使用动作脚本字典（请参见第 181 页的第 12 章“动作脚本字典”）中各个元素的“可用性”信息来确定要作为目标播放器的 Flash Player 版本是否支持您要使用的元素。您也可以通过显示“动作”工具箱来确定哪些元素可以使用；目标版本所不支持的元素突出显示为黄色。

如果要为 Flash Player 6 或 Flash Player 7 创建内容，应使用动作脚本 2.0。它具有许多动作脚本 1 所没有的重要功能，例如，改进的编译器错误报告功能和更坚实的面向对象编程功能。

要了解某些功能在为 Flash Player 7 发布的文件中的实现方式与在为早期的播放器版本发布的文件中的实现方式有哪些不同之处，请参见第 14 页的“将现有脚本移植到 Flash Player 7”。

若要在发布文档时指定要使用的播放器和动作脚本版本，请选择“文件”>“发布设置”，然后在“Flash”选项卡上进行选择。如果需要以 Flash Player 4 作为目标播放器，请参见下一节。

使用 Flash MX 2004 为 Flash Player 4 创建内容

若要使用 Flash MX 2004 为 Flash Player 4 创建内容，请在“发布设置”对话框（“文件”>“发布设置”）的“Flash”选项卡上指定“Flash Player 4”。

Flash Player 4 动作脚本只有一种基本的原始数据类型，用于处理数字和字符串。在为 Flash Player 4 创作应用程序时，必须使用已不鼓励使用的字符串运算符。这些运算符位于“动作”工具箱的“Deprecated”>“Operators”类别中。

发布适用于 Flash Player 4 的文档时，可以使用以下 Flash MX 2004 功能：

- 数组和对象访问运算符 ([])
- 点运算符 (.)
- 逻辑运算符、赋值运算符，以及前递增和后递增 / 递减运算符
- 求模运算符 (%)，以及 Math 类的所有方法和属性

Flash Player 4 本身不支持以下语言元素。Flash MX 2004 将它们导出为级数渐近值，使用这种方法创建的结果数字精确度较低。另外，由于 SWF 文件中包含级数渐近值，这些语言元素在 FlashPlayer 4 SWF 文件中占用的空间要比在 Flash Player 5 或更高版本的 SWF 文件中多。

- for、while、do..while、break 和 continue 动作
- print() 和 printAsBitmap() 动作
- switch 动作

有关详细信息，请参见第 749 页的“关于以早期的 Flash Player 版本作为目标播放器”。

使用 Flash MX 2004 打开 Flash 4 文件

Flash 4 动作脚本只有一种真正的数据类型：字符串。它在表达式中使用不同类型的运算符，用于指示将值作为字符串还是数字来处理。在其后的 Flash 版本中，对所有数据类型都可以使用一组运算符。

在使用 Flash 5 或更高版本打开在 Flash 4 中创建的文件时，Flash 会自动转换动作脚本表达式，以使它们与新语法兼容。在动作脚本代码中可以看到如下的数据类型和运算符转换：

- Flash 4 中的 = 运算符用于数字等式。在 Flash 5 和更高的版本中，== 是等于运算符，而 = 是赋值运算符。Flash 4 文件中的所有 = 运算符都将被自动转换为 ==。
- Flash 会自动执行类型转换，以确保运算符按照预期方式运行。由于引入了多种数据类型，下列运算符有了新的意义：
+, ==, !=, <>, <, >, >=, <=

在 Flash 4 动作脚本中，这些运算符始终是数字运算符。在 Flash 5 和更高版本中，它们的行为根据操作数的数据类型而不同。为了防止在导入的文件中出现语义差异，在这些运算符的所有操作数两边都会插入一个 Number() 函数。（常数已经明显是数字，所以不会在常数两边插入 Number() 函数）。

- 在 Flash 4 中，转义序列 \n 生成一个回车符 (ASCII 13)。在 Flash 5 和更高版本中，为了遵循 ECMA-262 标准，\n 会生成一个换行符 (ASCII 10)。Flash 4 FLA 文件中的 \n 序列将被自动转换为 \r。
- Flash 4 中的 & 运算符用于连接字符串。在 Flash 5 和更高版本中，& 是按位“与”运算符。字符串连接运算符现在为 add。Flash 4 中的所有 & 运算符都将被自动转换为 add 运算符。
- Flash 4 中许多函数的后面都不需要括号，例如 Get Timer、Set Variable、Stop 和 Play。为了保持语法一致，getTimer 函数和所有动作的后面现在都需要有括号。这些括号在转换时会自动添加。
- 在 Flash 5 和更高版本中，对不存在的影片剪辑执行 getProperty 函数时，它所返回的值为 undefined，而不是 0。在 Flash 4 之后的动作脚本中，undefined == 0 为 false（在 Flash 4 中，undefined == 1）。在 Flash 5 和更高版本中，通过在等式比较中引入 Number() 函数，在转换 Flash 4 文件时解决了此问题。在下面的示例中，Number() 会强制将 undefined 转换为 0，因此比较将会成功：

```
getProperty("clip", _width) == 0  
Number(getProperty("clip", _width)) == Number(0)
```

注意：如果在 Flash 4 动作脚本中使用了任何 Flash 5 或更高版本的保留字作为变量名，在 Flash MX 2004 中进行编译时该语法将返回一个错误。若要解决问题，请重命名这些变量（在所有出现的位置）。请参见第 30 页的“关键字”和第 36 页的“命名变量”。

使用斜杠语法

斜杠语法在 Flash 3 和 4 中表示影片剪辑或变量的目标路径。在斜杠语法中，使用的是斜杠，而不是点；此外，要表示一个变量，变量前面应为一个冒号：

```
myMovieClip/childMovieClip:myVariable
```

若要用 Flash Player 5 和更高版本支持的点语法（请参见第 27 页的“点语法”）编写同样的目标路径，应该使用下面的代码：

```
myMovieClip.childMovieClip.myVariable
```

斜杠语法通常与 `tellTarget` 动作配合使用，我们也不再建议使用该动作。现在，`with` 动作要比 `tellTarget` 更好，因为它与点语法的兼容性更好。有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `tellTarget` 和 `with`。

附录 E

使用动作脚本 1 进行面向对象的编程

本附录中的信息节选自 Macromedia Flash MX 文档，提供有关使用动作脚本 1 对象模型编写脚本的说明。之所以在此处提供这些信息是出于以下原因：

- 如果要编写支持 Flash Player 5 的面向对象脚本，则必须使用动作脚本 1。
- 如果您已使用动作脚本 1 编写过面向对象的脚本，而尚未准备好转换到动作脚本 2.0，则可以使用本附录来查找或浏览编写动作脚本 1 脚本时需要的信息。

如果您从未使用动作脚本编写过面向对象的脚本，而且不需要以 Flash Player 5 作为目标播放器，则不应使用本附录中的信息，因为我们不鼓励使用动作脚本 1 来编写面向对象的脚本；而应参见第 137 页的第 9 章“使用动作脚本 2.0 创建类”以获得有关使用动作脚本 2.0 的信息。

注意：本附录中的某些示例使用了 `Object.RegisterClass()` 方法。只有 Flash Player 6 和更高版本中才支持该方法；如果您要以 Flash Player 5 作为目标播放器，请不要使用该方法。

关于动作脚本 1

动作脚本是一种面向对象的编程语言。面向对象的编程使用对象（或称作数据结构）将控制对象行为和外观的属性和方法组合在一起。使用对象可以组织并重用代码。定义了一个对象后，可以通过名称引用该对象，而无需在每次使用它时重新进行定义。

类是对象的通用类别。类定义一系列具有公共属性，并且能够以相同的方式进行控制的对象。属性是定义对象的特性，例如，其大小、位置、颜色、透明度，等等。属性针对某个类进行定义，而属性的值则针对该类中各个单独的对象设置。方法是可以设置或检索对象的属性的函数。例如，您可以定义一个方法来计算某个对象的大小。与属性类似，方法针对某个对象类进行定义，然后针对该类中各个单独的对象调用。

动作脚本提供了几个内置类，包括 `MovieClip` 类和其它一些类。您也可以创建类，为您的应用程序定义各种对象类别。

动作脚本中的对象可以是纯数据容器，或者可以在舞台上以图形化方式表示为影片剪辑、按钮或文本字段。所有影片剪辑都是内置类 `MovieClip` 的实例，所有按钮都是内置类 `Button` 的实例。每个影片剪辑实例都包含 `MovieClip` 类的所有属性（例如，`_height`、`_rotation`、`_totalframes`）和所有方法（例如，`gotoAndPlay()`、`loadMovie()`、`startDrag()`）。

若要定义类，您需要创建一个称作构造函数 的特殊函数。（内置类具有内置的构造函数。）例如，如果需要应用程序中某个自行车骑手的相关信息，您可以创建一个构造函数 `Biker()`，它具有属性 `time` 和 `distance`，以及能够告诉您自行车行进速度的方法 `getSpeed()`：

```
function Biker(t, d) {  
    this.time = t;  
    this.distance = d;  
}
```

```

    this.getSpeed = function() {return this.time / this.distance;};
}

```

在此示例中，要创建一个需要以下两条信息（或称作参数）才能完成工作的函数：t 和 d。当调用该函数创建对象的新实例时，会将这些参数传递给它。下面的代码创建 Biker 对象的两个实例，名称分别为 emma 和 hamish。

```

emma = new Biker(30, 5);
hamish = new Biker(40, 5);

```

在面向对象的脚本中，类可以按照特定的顺序相互接收属性和方法，这称为继承。您可以使用继承来扩展或重新定义类的属性和方法。从其它类继承属性和方法的类称为子类。向其它类传递属性和方法的类称为超类。类可以既是子类又是超类。

对象是一种复杂的数据类型，它包括零个或多个属性和方法。每个属性都有名称和值，就像变量一样。属性附加在对象上，并且包含可以更改和检索的值。这些值可以是任何数据类型：字符串、数字、布尔值、对象、影片剪辑或未定义。下列属性属于各种不同的数据类型：

```

customer.name = "Jane Doe";
customer.age = 30;
customer.member = true;
customer.account.currentRecord = 000609;
customer.mcInstanceName._visible = true;

```

对象的属性也可以是对象。在上面示例的第 4 行，account 是对象 customer 的属性，而 currentRecord 是对象 account 的属性。currentRecord 属性的数据类型是数字。

使用动作脚本 1 创建自定义对象

若要创建自定义对象，您需要定义构造函数。构造函数的名称总是与它创建的对象类型的名称相同。可以在构造函数体中使用关键字 this 来引用该构造函数创建的对象；在调用构造函数时，Flash 会将 this 作为隐藏参数传递给它。例如，下面就是一个构造函数，用于创建具有属性 radius 的圆：

```

function Circle(radius) {
    this.radius = radius;
}

```

定义了构造函数之后，必须创建该对象的一个实例。在构造函数名称的前面使用 new 运算符，并给新实例分配一个变量名。例如，下面的代码将使用 new 运算符创建一个 Circle 对象，其半径为 5，然后将其分配给变量 myCircle：

```

myCircle = new Circle(5);

```

注意：对象的范围与其被分配到的变量的范围相同。

使用动作脚本 1 将方法分配给自定义对象

您可以在对象的构造函数中定义对象的方法。但是，我们不建议采用这种方式，因为每次使用构造函数时都要定义方法，如下例所示。在此示例中，将创建方法 area() 和 diameter()：

```

function Circle(radius) {
    this.radius = radius;
    this.area = Math.PI * radius * radius;
    this.diameter = function() {return 2 * this.radius;}
}

```

每个构造函数都有一个 `prototype` 属性，此属性是在定义该函数时自动创建的。`prototype` 属性表示用该函数创建的对象默认属性值。对象的每个新实例都有一个 `__proto__` 属性，用于引用创建它的构造函数的 `prototype` 属性。因此，如果将方法分配给对象的 `prototype` 属性，则该对象的任何新创建的实例都可以使用这些方法。最好将方法分配给构造函数的 `prototype` 属性，因为它只存在于一个位置，并且由该对象（或类）的新实例引用。可以使用 `prototype` 和 `__proto__` 属性扩展对象，这样能够以面向对象的方式重用代码。（有关更多信息，请参见[第 756 页的“使用动作脚本 1 创建继承”](#)。）

下面的过程将显示如何将 `area()` 方法分配给自定义的 `Circle` 对象。

将方法分配给自定义对象：

- 1 定义构造函数 `Circle()`，如下所示。

```
function Circle(radius) {  
    this.radius = radius;  
}
```

- 2 定义 `Circle` 对象的 `area()` 方法。`area()` 方法用于计算圆的面积。可以使用函数文本定义 `area()` 方法，并将 `area` 属性分配给该圆的原型对象，如下所示：

```
Circle.prototype.area = function () {  
    return Math.PI * this.radius * this.radius;  
};
```

- 3 创建 `Circle` 对象的一个实例，如下所示：

```
var myCircle = new Circle(4);
```

- 4 调用新创建的 `myCircle` 对象的 `area()` 方法，如下所示：

```
var myCircleArea = myCircle.area();
```

动作脚本将在 `myCircle` 对象中搜索 `area()` 方法。由于该对象没有 `area()` 方法，因此会在它的原型对象 `Circle.prototype` 中搜索 `area()` 方法。动作脚本会在找到该方法后调用它。

使用动作脚本 1 定义事件处理函数方法

可以为影片剪辑创建一个动作脚本类，然后在新创建的类的原型对象中定义事件处理函数方法。在原型对象中定义方法可使该元件的所有实例都以同一种方式响应事件。

也可以将 `onClipEvent()` 或 `on()` 事件处理函数动作添加到一个单独的实例中，以提供只有当该实例的事件发生时才运行的独特指令。`onClipEvent()` 和 `on()` 动作不会重写事件处理函数方法；两种事件都会使它们的脚本运行。不过，如果在原型对象中定义了事件处理函数方法，同时也为某个特定实例定义了事件处理函数方法，那么该实例定义会重写原型定义。

在对象的原型对象中定义事件处理函数方法：

- 1 将一个链接 ID 为 `theID` 的影片剪辑元件放到库中。

- 2 在“动作”面板上（“窗口” > “开发面板” > “动作”），使用 `function` 动作定义一个新类，如下所示：

```
// 定义类  
function myClipClass(){}  

```

这个新类将被指定给由时间轴添加到应用程序中的影片剪辑的所有实例，或者用 `attachMovie()` 或 `duplicateMovieClip()` 方法添加到应用程序中的影片剪辑的所有实例。如果想让这些影片剪辑能够访问内置 `MovieClip` 对象的方法和属性，则需要让新类继承 `MovieClip` 类的方法和属性。

3 输入如下代码：

```
// 从 MovieClip 类继承方法和属性
myClipClass.prototype = new MovieClip();
```

现在，myClipClass 类继承了 MovieClip 类的所有属性和方法。

4 输入如下代码，定义这个新类的事件处理函数方法：

```
// 为 myClipClass 类定义事件处理函数方法
myClipClass.prototype.onLoad = function() {trace ("movie clip loaded");}
myClipClass.prototype.onEnterFrame = function() {trace ("movie clip entered
frame");}
```

5 如果“库”面板没有打开，请选择“窗口”>“库”打开它。

6 选择想与新类关联的元件，然后从“库”面板右上角的弹出菜单中选择“链接”。

7 在“链接属性”对话框中，选择“为动作脚本导出”。

8 在“标识符”框中输入一个标识符。

对于想与新类关联的所有元件，它们的标识符必须都相同。在 myClipClass 示例中，标识符是 theID。

9 在“脚本”窗格中输入如下代码：

```
// 注册类
Object.registerClass("theID", myClipClass);
_root.attachMovie("theID", "myName", 1);
```

这样就向 myClipClass 类注册了链接标识符为 theID 的元件。myClipClass 的所有实例的事件处理函数方法的行为与第 4 步中定义的一样。同时它们的行为也类似于 MovieClip 类的所有实例，因为您在第 3 步中指示新类继承 MovieClip 类的属性和方法。

```
function myClipClass(){}

myClipClass.prototype = new MovieClip();
myClipClass.prototype.onLoad = function(){
    trace("movie clip loaded");
}
myClipClass.prototype.onPress = function(){
    trace("pressed");
}

myClipClass.prototype.onEnterFrame = function(){
    trace("movie clip entered frame");
}

myClipClass.prototype.myfunction = function(){
    trace("myfunction called");
}

Object.registerClass("myclipID", myClipClass);
_root.attachMovie("myclipID", "ablue2", 3);
```

使用动作脚本 1 创建继承

继承是一种组织、扩展和重用功能的方式。子类会从超类继承属性和方法，并添加自己的专用属性和方法。例如，就现实世界而言，自行车是一个超类，而山地车和三轮车则是该超类的子类。这两个子类都包含，或者说是继承了超类的方法和属性（例如 wheels）。每个子类还具有它们自己的属性和方法，这些属性和方法扩展了超类（例如，MountainBike 子类具有 gears 属性）。在动作脚本中可以使用 prototype 和 __proto__ 元素来创建继承。

所有的构造函数都有 `prototype` 属性，该属性是在定义该函数时自动创建的。`prototype` 属性表示用该函数创建的对象默认属性值。可以使用 `prototype` 属性将属性和方法分配给类。（有关更多信息，请参见第 754 页的“使用动作脚本 1 将方法分配给自定义对象”。）

类的所有实例都具有 `__proto__` 属性，该属性会指出这些实例继承自哪个对象。在使用构造函数创建对象时，会将 `__proto__` 属性设置为引用其构造函数的 `prototype` 属性。

继承会按照明确的层级向下延伸。在调用对象的属性或方法时，动作脚本会查看该对象，确定是否存在这样一个元素。如果不存在，动作脚本会查看该对象的 `__proto__` 属性以获得信息（`myObject.__proto__`）。如果该属性不是该对象的 `__proto__` 对象的属性，动作脚本会查看 `myObject.__proto__.__proto__`，依此类推。

下面的示例将定义构造函数 `Bike()`：

```
function Bike (length, color) {  
    this.length = length;  
    this.color = color;  
}
```

下面的代码会将 `roll()` 方法添加到 `Bike` 类：

```
Bike.prototype.roll = function() {this._x = _x + 20;};
```

可以创建一个将 `Bike` 作为超类的 `MountainBike` 类，这样就不用再向 `MountainBike` 类和 `Tricycle` 类中添加 `roll()` 方法了：

```
MountainBike.prototype = new Bike();
```

现在，可以调用 `MountainBike` 的 `roll()` 方法，如下所示：

```
MountainBike.roll();
```

影片剪辑不会相互继承。若要在影片剪辑间创建继承，可以使用 `Object.registerClass()` 方法给影片剪辑分配一个除 `MovieClip` 之外的类。请参见第 181 页的第 12 章“动作脚本字典”中的 `Object.registerClass()`。

有关继承的更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `Object.__proto__`、`#initclip`、`#endinitclip` 和 `super` 条目。

使用动作脚本 1 向对象中添加 `getter/setter` 属性

您可以使用 `Object.addProperty()` 方法为对象创建 `getter/setter` 属性。

`Getter` 函数没有参数。它的返回值可以为任何类型。它的类型可以在两次调用之间改变。返回值视作该属性的当前值。`Setter` 函数只有一个参数，即该属性的新值。例如，如果属性 `x` 由语句 `x = 1` 进行赋值，则会将数字类型的参数 1 传递给 `setter` 函数。`Setter` 函数的返回值将被忽略。

当 Flash 读取 `getter/setter` 属性时，它调用 `getter` 函数，而该函数的返回值将成为 `prop` 的值。当 Flash 写入 `getter/setter` 属性时，它调用 `setter` 函数，并将新值作为参数传递给它。如果具有给定名称的属性已经存在，新属性将覆盖它。

可以向原型对象添加 `getter/setter` 属性。如果向一个原型对象添加 `getter/setter` 属性，则继承此原型对象的所有对象实例都将继承 `getter/setter` 属性。这样就能够在一个位置（即原型对象）添加 `getter/setter` 属性，然后使它传播到类的所有实例（与向原型对象添加方法非常相似）。如果为继承的原型对象中的 `getter/setter` 属性调用 `getter/setter` 函数，则传递给该 `getter/setter` 函数的引用将是最初引用的对象，而不是该原型对象。

有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `Object.addProperty()`。

测试模式中的“调试” > “列出变量”命令支持使用 `Object.addProperty()` 方法添加到对象中的 `getter/setter` 属性。以这种方式添加到对象中的属性与该对象中的其它属性一起显示在“输出”面板中。在“输出”面板中，`getter/setter` 属性用前缀 `[getter/setter]` 标识。有关“列出变量”命令的更多信息，请参见第 68 页的“使用“输出”面板”。

在动作脚本 1 中使用 Function 对象属性

使用 `Function` 对象的 `call()` 和 `apply()` 方法可以指定要将函数应用于哪个对象，以及要传递给该函数的参数值。动作脚本中的每个函数都由一个 `Function` 对象表示，因此所有函数都支持 `call()` 和 `apply()` 方法。使用构造函数创建自定义类时，或使用函数为自定义类定义方法时，可以调用该函数的 `call()` 和 `apply()` 方法。

在动作脚本 1 中使用 Function.call() 方法调用函数

`Function.call()` 方法调用由 `Function` 对象表示的函数。

几乎在所有情况下，都可以使用函数调用运算符 `()` 代替 `call()` 方法。函数调用运算符使代码简明易读。`call()` 方法主要用于需要显式控制函数调用的 `this` 参数的情况。通常，如果将函数作为对象的方法来调用，则在函数体内，`this` 设置为 `myObject`，如下所示：

```
myObject.myMethod(1, 2, 3);
```

在某些情况下，您可能希望 `this` 指向其它地方，例如以下这种情况：函数必须作为对象的方法进行调用，但该函数实际上并不是作为该对象的方法存储的。

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

将值 `null` 传递给 `thisObject` 参数即可将函数作为常规函数（而不是作为对象的方法）进行调用。例如，下面的函数调用是等效的：

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `Function.call()`。

使用 `Function.call` 方法调用函数：

- 应使用以下语法。

```
myFunction.call(thisObject, parameter1, ..., parameterN)
```

该方法具有以下参数：

- 参数 `thisObject` 指定 `this` 在函数体内的值。
- 参数 `parameter1, ..., parameterN` 指定要传递给 `myFunction` 的参数。可以指定零个或多个参数。

在动作脚本 1 中使用 Function.apply() 指定要将函数应用于哪个对象

`Function.apply()` 方法指定将在动作脚本调用的任何函数内使用的 `this` 的值。此方法还指定要传递给任何被调用函数的参数。

参数被指定为 `Array` 对象。如果在脚本实际执行前，无法知道要传递的参数的数量，那么这种方法通常很有用。

有关更多信息，请参见第 181 页的第 12 章“动作脚本字典”中的 `Function.apply()`。

使用 `Function.apply()` 指定要将函数应用于哪个对象：

- 应使用以下语法。

```
myFunction.apply(thisObject, argumentsObject)
```

该方法具有以下参数：

- 参数 *thisObject* 指定要将 *myFunction* 应用于哪个对象。
- 参数 *argumentsObject* 定义一个数组，该数组的元素将作为参数传递给 *myFunction*。

A

- ActiveX 控件 165
- ASCII 值 85
 - 功能键 747
 - 键盘键 745
 - 其它键 747
 - 数字键盘键 746
- 按键, 捕获 85
- 安全性 166–169
 - 和策略文件 168
 - 将脚本移植到 Flash Player 7 16, 18, 19
 - 跨域数据访问 167, 168
- 按位运算符 42

B

- 包 150
 - 命名 150
- 保留字。请参见关键字
- 本地变量 37
 - 范例 37
 - 和严格数据类型指定 37
 - 在函数中 46
- 比较运算符 40
- 变量
 - 避免名称冲突 120
 - 测试 36
 - 动态设置 43
 - 多个赋值 42
 - 发送到 URL 83
 - 关于 36
 - 和调试器变量选项卡 64
 - 和调试器监视点列表 65
 - 后缀 55
 - 检查和设置值 36
 - 命名 55
 - 命名规则 36
 - 确定范围 37

- 确定数据类型 33
- 已定义 26
- 引用值 39
- 在调试器中修改 64
- 在脚本中使用 38
- 在影片和服务器间传输 159
- 传递内容 38
- 转换成 XML 161
- 变量列表命令 69
- 变量选项卡, 调试器 64
- 编译时, 已定义 10
- 表达式
 - 比较值 40
 - 处理值 39
 - 给多个变量赋值 42
 - 已定义 25
- 标点平衡, 检查 60
- 标识符, 已定义 25
- 播放器, 执行应用程序 164
- 播放影片剪辑 82
- 布尔值 32
 - 比较 41
 - 已定义 24
- 捕获按键 85

C

- CSS。请参见层叠样式表
- 参量。请参见参数
- 参数
 - 已定义 26
 - 在括号中 28
 - 传递给函数 46
- 策略文件 168
 - 必须具有名称 crossdomain.xml 168
 - 另请参见安全性
- 测试影片
 - 和 Unicode 61

- 和键盘控制 61
- 测试。请参见调试
- 层叠样式表
 - 对文本进行格式设置 122
 - 和 TextField.StyleSheet 类 123
- 合并样式 125
- 加载 124
- 将样式分配到内置 HTML 标签 126
- 应用到文本字段 125
- 应用样式类 126
- 用于 HTML 标签的示例 127
- 用于 XML 标签的示例 129
- 用于定义新标签 128
- 在动作脚本中定义样式 125
- 支持的属性 123
- 常数 24, 30
- 重复动作 50
- 冲突, 检测 93
 - 影片剪辑之间 95
 - 在影片剪辑和舞台点之间 94
- 重制, 影片剪辑 112
- 处理函数。请参见事件处理函数
- 处理数字 31
- 初始化影片剪辑属性 117
- 创建对象 101
- 从远程文件获取信息 157
- 错误
 - 错误消息列表 739
 - 名称冲突 37
 - 语法 54

D

- 大括号 27
 - 检查匹配对 60
- 大括号。请参见大括号
- DOM (文本对象模型), XML 160
- 代码
 - 格式设置 60, 61
 - 跟踪行 67
 - 显示行号 61
 - 选择一行 67
 - 自动换行 61
- 代码提示 54
 - 触发 54, 55, 56
 - 使用 56
 - 手动显示 58
 - 未显示 57
 - 指定设置 56
- 代码中的缩进, 启用 61
- 代码中的行号, 显示 61

- 代码中的自动换行, 启用 61
- 导出脚本和语言编码 24
- 导航
 - 控制 81
 - 跳到某一帧或场景 82
- 导入
 - 脚本, 和语言编码 24
 - 类 151
- 导入类 151
- 等于运算符 42
 - 不同于赋值运算符 42
- 严格 42
- 点语法 27
- 点运算符 43
- 调试
 - 编译器错误消息 739
- 动画, 元件和 32
- 动态类 152
- 动作
 - 重复 50
 - 已定义 24
- 动作工具箱 52
 - 黄色项目 54
- 动作脚本
 - 动作脚本 2.0 编译器错误消息 739
 - 动作脚本 2.0 概述 20, 137
 - 将动作脚本 2.0 类分配给影片剪辑 116
 - 在动作脚本 1 中不支持严格数据类型指定 34
- 动作脚本编辑器 52, 54
- 动作面板 51
- 断点
 - 关于 66
 - 和外部文件 66
 - 在调试器中设置 66
- 对象
 - 遍历子级 50
 - 创建 101
 - 访问属性 101
 - 和面向对象的编程 137
 - 数据类型 32
 - 调用方法 102
 - 已定义 25
- 对象列表命令 69
- 对象属性
 - 访问 43
 - 将值赋予 101
- 对于循环和实例成员 145
- 多种语言, 在脚本中使用 23
- 多维数组 44
- 多重继承, 不允许 143

E

- ECMA-262
 - 符合性 15
 - 规范 23
- Escape 快捷键 59

F

- Flash Player
 - 标准菜单视图 164
 - 方法 165
 - 获取最新版本 71
 - 将 SWF 文件缩放到 164
 - 屏蔽上下文菜单 164
 - 全屏显示 164
 - 调试版本 61
 - 通讯 164
 - 显示上下文菜单 164
- Flash Player 7
 - 和 ECMA-262 符合性 15
 - 迁移现有的脚本 14, 169
 - 新的安全模型 14, 16, 18, 19
 - 新增的和经改进的语言元素 13
- FLV（外部视频）文件 174
 - 预加载 177
- fscommand() 函数
 - 命令和参量 164
 - 使用 164
 - 与 Director 通讯 165
- 发送信息
 - 给远程文件 157
 - 通过 TCP/IP 158
 - URL 编码格式 157
 - 以 XML 格式 157
- 范例脚本 96
- 方法
 - 对象, 调用 102
 - 声明 142
 - 异步的 158
 - 已定义 25
 - 用于控制影片剪辑 108
- 访问对象属性 43
- 分号 28
- 父级影片剪辑, 已定义 107
- 附加, 声音 91
- 服务器, 打开持续连接 163
- 服务器端脚本
 - XML 格式 161
 - 语言 157
- 赋值运算符
 - 不同于等于运算符 42

- 复合 43
- 关于 42

G

- getAscii() 方法 85
- getURL() 方法 83
- 跟踪代码行 67
- 工具提示。请参见代码提示
- 功能键, ASCII 键控代码值 747
- 构造函数
 - 概述 144
 - 已定义 24
- 构造函数, 范例 753
- 关键字 25
 - 列出的 30
- 光标, 创建自定义 83
- 广播器对象 76
- 滚动文本 135

H

- hitTest() 方法 93
- HTML
 - 括在引号中的标签 130
 - 设置内置标签的样式 126
 - 使用 标签排列文本 130, 131, 134
 - 使用层叠样式表定义标签 128
 - 使用样式的示例 127
 - 在文本字段中使用 130
 - 支持的标签 130
- HTTP 协议 157
 - 与服务器端脚本通讯 158
- HTTPS 协议 157
- 函数 25
 - 本地变量 46
 - 定义 45
 - 返回值 46
 - 范例 26
 - 构造函数 753
 - 和方法 25
 - 将参数传递给 46
 - 内置 45
 - 调用 47
 - 异步的 158
 - 用于控制影片剪辑 108
 - 转换 31
 - 自定义 45
- 后缀 55
- 绘制
 - 线条和填充 95
 - 形状 115

获取鼠标指针位置 84

I

ID3 标签 174

IP 地址

和安全性 166

和策略文件 168

J

JavaScript

alert 语句 70

国际标准 23

和 Netscape 166

和动作脚本 23

将消息发送到 164

Netscape Navigator 文档 23

级别 44

加载 108

继承 138

和子类 143

只允许从一个类继承 143

JPEG 文件

加载到影片剪辑 110, 172

预加载 175

在文本字段中嵌入 134

计数器, 重复动作 50

基于组件的体系结构, 已定义 107

加载的 SWF 文件

标识 44

删除 108

检测冲突 93

检查

已加载的数据 158

语法和标点 60

键控代码, ASCII

功能键 747

获取 85

其它键 747

数字键盘 746

字母和数字键 745

键盘控制

和测试影片 61

激活影片剪辑 87

键盘, ASCII 键控代码值 745

键入变量 33

监视点选项卡, 调试器 65

将备注添加到脚本 29

脚本

测试 61

导入和导出 24

范例 96

关于编写和调试 49

就地固定 53

纠正文本显示问题 24

控制流 49

控制执行 49

迁移到 Flash Player 7 14, 169

声明变量 38

调试 61

用于固定脚本的快捷键 53

注释 29

脚本窗格

关于 52

上方的按钮 52

使用脚本 52

脚本窗口 (仅限 Flash Professional) 51

脚本导航器 52

交互性, 在 SWF 文件中

创建 81

技术 83

节点 160

结合律, 运算符 40, 743

接口 138

创建和使用 147-148

结束语句 28

静态成员。请参见类成员

就地固定脚本 53

K

可扩展标记语言。请参见 XML

空值数据类型 32

快捷键

用于固定脚本 53

L

loadMovie() 函数 158

loadVariables() 函数 158

LoadVars 对象 159

类

不支持重载 144

创建的示例 138

创建和使用 141

创建属性和方法 142

创建外部类文件 139

创建子类 143

dynamic 152

导入 151

分配给影片剪辑 116

关于编译和导出 153

和面向对象的编程 138

- 获取 / 设置方法 151
- 接口 147–148
- 解析类引用 149
- 扩展 143
- 类路径 148
- 命名 142
- public 和 private 成员属性 144
- 实例成员和类成员 145
- 已定义 24, 101
- 以内联方式初始化属性 142
- 用包整理 150
- 另请参见类, 内置
- 在运行时初始化属性 117
- 在运行时扩展 152
- 指定导出帧 153
- 只能在外部文件中定义 139, 141

- 类成员 102
 - 创建 145
 - 对于每个类仅创建一次 145
 - 和子类 146
 - 使用类成员的示例 145
- 类成员的 private 属性 144
- 类成员的 public 属性 144
- 类的获取 / 设置方法 151
- 类路径

- 全局和文档级 148
- 搜索顺序 149
- 修改 149
- 已定义 148
- 类文件, 创建 139
- 类, 内置 101–106
 - 扩展 143
- 联合运算 43
- 链接, 影片剪辑 112
- 链接标识符 112, 116
- 链接属性对话框 112, 116
- 连接字符串 31
- 逻辑运算符 41

M

- Macromedia Director, 通讯 165
- MIME 格式, 标准 159
- movienname_DoFSCommand 函数 164
- MP3 文件
 - 和 ID3 标签 174
 - 加载到影片剪辑 173
 - 预加载 177
- 密码和远程调试 62
- 面向对象的编程 137
 - 另请参见类

- 名称冲突 37
- 命名变量 36, 55
- 命名约定
 - 包 150
 - 类 142
- 默认编码 23
- 目标路径
 - 输入 45
 - 已定义 26
 - 指定 44

N

- Netscape DevEdge Online 23
- Netscape, 支持的 JavaScript 方法 166
- 内置函数 45

O

- on() 和 onClipEvent() 处理函数 77
 - 范围 79
 - 附加到影片剪辑 78
- onClipEvent() 处理函数 97

P

- 排版印刷约定 10
- 平衡 (声音), 控制 92

Q

- 嵌套影片剪辑, 已定义 107
- 全局变量 38
 - 和严格数据类型指定 34
- 全局类路径 148

R

- _root 属性和加载的影片剪辑 109

S

- setRGB 方法 89
- statements
 - 结束 28
 - trace 语句 70
 - 组合 27
- SWD 文件, 已定义 62
- SWF 文件
 - 保持原始大小 164
 - 创建声音控件 90
 - Flash Player 中的控制 165
 - 放在网页上 83
 - 加载到影片剪辑 172
 - 加载和卸载 108

- 缩放到 Flash Player 164
- 跳到某一帧或场景 82
- 预加载 175
 - 另请参见影片剪辑
 - 在文本字段中嵌入 134
 - 传递信息 157
- 删除
 - 加载的 SWF 文件 108
 - 影片剪辑 112
- 设备字体, 遮蔽 116
- 设置代码格式 60, 61
- 深度
 - 管理 114
 - 确定实例 114
 - 确定下一个可用的 114
 - 确定影片剪辑的 114
 - 已定义 114
- 声音
 - 附加到时间轴 91
 - 控制 90
 - 另请参见外部媒体
 - 平衡控件 92
- 事件, 已定义 25, 75
- 事件处理函数
 - 范围 79
 - 分配函数 76
 - 附加到按钮或影片剪辑 77
 - 检查 XML 数据 158
 - 已定义 25, 75
 - 由动作脚本类定义 76
 - 与 on() 和 onClipEvent() 77
- 事件处理函数方法 75
- 事件模型
 - on() 和 onClipEvent() 处理函数 77
 - 事件处理函数方法 75
 - 事件侦听器 76
- 事件侦听器 76
 - 范围 79
 - 由动作脚本类定义 77
- 时间轴变量 37
- 实例
 - 创建的示例 141
 - 已定义 25, 101
- 实例成员 145
- 实例化对象 101
- 实例名称
 - 动态设置 43
 - 已定义 25, 107
 - 与变量名称相比较 120
 - 指定 44
- 视频, 导入的替代方法 174
- “视图选项”弹出菜单 59, 61
- 鼠标位置, 获取 84
- 鼠标指针。请参见光标
- “输出”面板 68
 - 变量列表命令 69
 - 对象列表命令 69
 - 和 trace 语句 70
 - 选项 68
- 数据, 外部 157
 - 安全功能 166
 - 发送和加载 157
 - 和 LoadVars 对象 159
 - 和 XML 160
 - 和 XMLSocket 对象 163
 - 和服务器端脚本 158
 - 和消息 164
 - 检查是否已加载 158
 - 跨域 SWF 间的访问 167
- 数据类型 30
 - 布尔值 32
 - 对象 32
 - MovieClip 32
 - null 32
 - 确定 33
 - 声明 33
 - 数字 31
 - undefined 33
 - 为元素指定 33
 - 严格类型指定 33
 - 已定义 25
 - 转换 31, 34
 - 自动指定 33
 - 字符串 31
- 属性
 - 常数 30
 - 对象, 访问 101
 - 访问 43
 - 声明 142
 - 已定义 26
 - 影片剪辑 110
 - 在运行时初始化 117
- 属性选项卡, 调试器 65
- 数值运算符 40
- 数字
 - 处理 31
 - 转换为 32 位整型 42
- 数字键盘, ASCII 键控代码值 746
- 数组, 多维 44
- 数组访问运算符 43

检查匹配对 60

T

Tab 键, 和测试影片 61

TCP/IP 连接

发送信息 158

XMLSocket 对象 163

text

编码 24

滚动 135

获取测量信息 122

确定 TextField 对象的所需大小 122

使用 标签在图像旁排列文本 131

在运行时分配到文本字段 120

文本

另请参见文本字段

TextField 类 119

创建滚动文本 135

TextField.StyleSheet 类 122

创建文本样式 125

和 TextField.styleSheet 属性 122, 125

和层叠样式表 123

TextFormat 类 121

this 关键字 97

套接字连接

范例脚本 163

关于 163

特殊字符 31

跳到 URL 83

条件, 检查 49

调试 61

编译器错误消息 739

从远程位置 62

调试播放器 61

列出变量 69

列出对象 69

使用 trace 语句 70

使用“输出”面板 68

文本字段属性 70

异常处理 13

调试播放器 61

调试器

按钮 67

变量 64

从上下文菜单选择 63

Flash 调试播放器 61

监视点列表 65

设置断点 66

使用 61

属性选项卡 65

调用方法 32

停止影片剪辑 82

图标

在调试器中 67

在脚本窗格上方 52

突出显示语法 54

图像

加载到影片剪辑 110

另请参见外部媒体

在文本字段中嵌入 134

拖动影片剪辑 111

U

Unicode

和测试影片命令 24, 61

支持 23

URL 编码格式, 发送信息 157

UTF-8 (Unicode) 23

W

Web 应用程序, 持续连接 163

外部类文件

创建 139

使用类路径进行查找 148

外部媒体 171–177

播放 FLV 文件 174

加载 MP3 文件 173

加载 SWF 文件和 JPEG 文件 172

加载概述 171

使用原因 171

预加载 175, 177

外部源, 连接 Flash 157

未定义的数据类型 33

文本编码 24

文本字段 119

避免变量名称冲突 120

格式设置 121

和 HTML 文本 126

默认属性 122

确定所需大小 122

实例和变量名称比较 120

使用层叠样式表进行格式设置 122

显示属性以进行调试 70

应用层叠样式表 125

另请参见 TextField 类、TextFormat 类、
TextField.StyleSheet 类

在嵌入的图像旁排列文本 130, 131

在运行时创建和删除 120

文档级类路径 148

问号, 包括在字符串中 31

舞台, 将元件附加到影片剪辑 112

X

XML 160

DOM 160

范例变量转换 160

分层 160

使用样式的示例 129

通过 TCP/IP 套接字发送信息 158

用 XML 方法发送信息 157

在服务器端脚本中 161

XML 类, 方法 160

XMLSocket 对象

方法 163

检查数据 158

使用 163

系统事件, 定义 75

系统要求 9

小括号 28

检查匹配对 60

消息框, 显示 165

斜杠语法 27

动作脚本 2.0 中不支持 27

信息, 在 SWF 文件间传递 157

选项弹出菜单

在调试器中 63

在动作面板中 53, 54

在“输出”面板中 68

循环 50

动作 50

Y

严格等于运算符 42

严格数据类型指定 33

动作脚本 1 中不支持 34

和本地变量 37

和全局变量 34

颜色

在动作工具箱中 54

在脚本窗格中 54

值, 设置 89

样式表。请参见层叠样式表

异步动作 158

异常处理 13

已加载的数据, 检查 158

疑难解答。请参见调试

音量, 创建滑动控件 92

引用变量 37

引用数据类型 30

影片剪辑

遍历子级 50

重制 112

创建空实例 111

创建子类 116

方法 108

方法, 用于绘制形状 115

方法和函数比较 107

分配按钮状态 78

父级, 已定义 107

附加 on() 和 onClipEvent() 处理函数 78

共享 112

管理深度 114

函数 108

和 _root 属性 109

和 with 语句 108

检测冲突 93

将 MP3 文件加载到 173

将 SWF 文件和 JPEG 文件加载到 172

开始和停止 82

控制 107

列出变量 69

列出对象 69

嵌套, 已定义 107

确定深度 114

确定下一个可用的深度 114

删除 112

实例名称, 已定义 107

使用键盘激活 87

数据类型 32

属性 110

属性, 在运行时初始化 117

添加参数 113

调节颜色 89

调用多个方法 108

调用方法 107

拖动 111

用作遮罩 115

另请参见 SWF 文件

在播放时更改属性 110

在调试器中改变属性 65

在文本字段中嵌入 134

在舞台上附加到元件 112

在运行时初始化属性 117

在运行时创建 111

指定实例名称 44

子级, 已定义 107

用户事件, 定义 75

优先级, 运算符 743

与 Flash Player 通讯 164

语法

- 大括号 27
- 点 27
- 分号 28
- 规则 26
- 检查 60
- 区分大小写 26–27
- 突出显示 54
- 小括号 28
- 斜杠 27
- 域名和安全性 166
- 语言, 在脚本中使用多种 23
- 远程调试 62
- 远程文件, 通讯 157
- 远程站点, 持续连接 163
- 原始数据类型 30
- 运算符 25
 - 按位 42
 - 比较 40
 - 等于 42
 - 点 43
 - 赋值 42
 - 和值进行组合 39
 - 结合律 40, 743
 - 逻辑 41
 - string 41
 - 数值 40
 - 数组访问 43
 - 优先级 743
- 运行时, 已定义 10

Z

- 在影片和服务端间传输变量 159
- 暂停 (跟踪) 代码 67
- 遮罩 115
 - 和设备字体 116
 - 忽略笔触 115, 116
- 侦听器对象 76
 - 注销 77
- 值, 在表达式中处理 39
- 执行顺序
 - 脚本 49
 - 运算符结合律 40
 - 运算符优先级 40
- 指针。请参见光标
- 中括号。请参见数组访问运算符
- 注册点, 和加载的图像 110
- 注释 29
- 术语 24
- 传递值
 - 按内容 38

- 按引用 39
- 转换函数 31
- 转换数据类型 31, 34
- 转义序列 31
- 自定义函数 45
- 字符串 31
- 字符串运算符 41
- 字符序列。请参见字符串
- 子级影片剪辑, 已定义 107
- 子节点 160
- 子类
 - 创建 143
 - 和类成员 146
 - 为影片剪辑创建 116
- 资源, 附加 10
- 组合语句 27

