macromedia®

FLASH® 8

学习 Flash 中的 ActionScript 2.0

商标

1 Step RoboPDF、ActiveEdit、ActiveTest、Authorware、Blue Sky Software、Blue Sky、Breeze、Breezo、Captivate、Central、ColdFusion、Contribute、Database Explorer、Director、Dreamweaver、Fireworks、Flash、FlashCast、FlashHelp、Flash Lite、FlashPaper、Flash Video Encoder、Flex、Flex Builder、Fontographer、FreeHand、Generator、HomeSite、Jrun、MacRecorder、Macromedia、MXML、RoboEngine、RoboHelp、RoboInfo、RoboPDF、Roundtrip、Roundtrip HTML、Shockwave、SoundEdit、Studio MX、UltraDev 和 WebHelp 是 Macromedia, Inc. 的注册商标或商标，可能已经在美国或其它司法辖区（包括全球范围）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由 Sorenson Media, Inc. 授权。

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

Macromedia Flash 8 视频采用 On2 TrueMotion 视频技术。© 1992-2005 On2 Technologies, Inc. 保留所有权利。http://www.on2.com。

Visual SourceSafe 是 Microsoft 公司在美国和 / 或其它国家（地区）的注册商标或商标。

鸣谢

项目管理：Sheila McGinn

撰稿：Jen deHaan、Peter deHaan、Joey Lott

责任编辑：Rosana Francescato

主编：Lisa Stanziano

编辑：Linda Adler、Geta Carlson、Evelyn Eldridge、John Hammett、Mary Kraemer、Noreen Maher、Jessie Wood、Anne Szabla

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

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

特别感谢 Jody Bleyle、Mary Burger、Lisa Friendly、Stephanie Gowin、Bonnie Loo、Mary Ann Walsh、Erick Vera、Yi Tan、测试版测试人员以及整个 Flash 和 Flash Player 工程小组与质量保证小组。

第一版：2005 年 9 月

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

目 录

简介

Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 是专业的标准创作工具，可以制作出极富感染力的 Web 内容。ActionScript 是用来向 Flash 应用程序添加交互性的语言，此类应用程序可以是简单的 SWF 动画文件，也可以是更复杂的功能丰富的 Internet 应用程序。您不必使用 ActionScript 就可以使用 Flash，但是，如果您要提供基本或复杂的与用户的交互性、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）或者想以其它方式让您的 SWF 文件具有更可靠的用户体验，则可能需要使用 ActionScript。

有关更多信息，请参见以下主题：

目标读者

本手册假定您已安装了 Flash Basic 8 或 Flash Professional 8 并了解用户界面的用法。您应该知道如何在 Flash 创作环境中将对象放置于舞台上并对其进行操作。如果以前使用过脚本语言，则会发现 ActionScript 与其它脚本语言非常类似。不过，即使您刚刚开始学习编程，ActionScript 基础知识也不难学。您可以从简单的命令入手，逐步掌握更复杂的功能。您可以向您的文件中添加大量交互性，而无须学习（或编写）大量的代码。

系统要求

除 Flash 8 之外，ActionScript 2.0 没有任何其它的系统要求。

本文档假定您的 Flash 文件使用默认的发布设置：Flash Player 8 和 ActionScript 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范例可能无法正常使用。如果要为早期版本的 Flash Player 开发应用程序，请参见第 703 页的附录 D “为早期的 Flash Player 版本编写脚本”。

更新 Flash XML 文件

始终安装最新的 Flash XML 文件很重要。Macromedia 有时在 Flash Player 的点版本（次级版本）中引入功能。当这样的版本推出时，应更新您的 Flash 版本以获取最新的 XML 文件。否则，如果您使用了 Flash Player 版本（随同 Flash 一起安装）中不可用的新属性或新方法， Flash 8 编译器可能会生成错误。

例如， Flash Player 7 (7.0.19.0) 中包含 System 对象的新方法 `System.security.loadPolicyFile`。要访问此方法，您必须使用 Player Updater 安装程序更新随 Flash 安装的所有 Flash Player。否则， Flash 编译器会显示错误。

请记住：可以安装比您的 Flash 版本早一个或一个以上主版本的 Player Updater。这样，您将获得所需的 XML 文件，但在发布到旧版本的 Flash Player 时不会出现任何编译器错误。有时旧版本也可以使用新方法或新属性，获取最新的 XML 文件可将尝试访问旧方法或旧属性时出现的编译器错误减到最少。

关于本文档

本手册提供 ActionScript 的语法概述以及有关在处理不同对象类型时如何使用 ActionScript 的信息。有关语法和每个语言元素的用法的详细信息，请参见《ActionScript 2.0 语言参考》。

有关更多信息，请参见以下主题：

- 第 10 页的 “学习 ActionScript 2.0 全书概述”
- 第 14 页的 “关于范例文件”
- 第 12 页的 “在本文档中使用的术语”
- 第 13 页的 “复制和粘贴代码”

学习 ActionScript 2.0 全书概述

下面的列表总结了本手册的内容：

- 第 1 章 “Flash 8 ActionScript 中的新增功能”介绍了 ActionScript 的新功能、对编译器和调试器的改进以及 ActionScript 2.0 语言的新编程模型。
- 第 2 章 “编写和编辑 ActionScript 2.0”介绍了 Flash 中 ActionScript 编辑器的功能，以更方便地进行代码编写。
- 第 3 章 “关于 ActionScript”对 ActionScript 语言进行了简单介绍并详细介绍了如何选择使用 ActionScript 的各个版本。
- 第 10 章 “数据和数据类型”介绍了关于数据、数据类型和变量的术语和基本概念。在整本手册中您都将用到这些概念。

- 第 4 章“语法和语言基础知识”介绍了 ActionScript 语言的术语和基本概念。在整本手册中您都将用到这些概念。
- 第 5 章“函数和方法”介绍了如何编写不同类型的函数和方法以及如何在应用程序中使用这些函数和方法。
- 第 6 章“类”介绍了如何在 ActionScript 中创建自定义类和对象。本章还列出了 ActionScript 中的内置类，并简要概述了如何使用这些内置类来访问 ActionScript 中的强大功能。
- 第 7 章“继承”介绍了 ActionScript 语言中的继承，并说明了如何扩展内置类或自定义类。
- 第 8 章“接口”介绍了如何在 ActionScript 中创建和使用接口。
- 第 9 章“处理事件”介绍了几种不同的事件处理方法：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。
- 第 11 章“使用影片剪辑”介绍了影片剪辑以及可用来控制影片剪辑的 ActionScript。
- 第 12 章“使用文本和字符串”介绍了在 Flash 中控制文本和字符串的不同方法，包括有关文本格式设置和 FlashType（高级文本呈现，如消除锯齿文本）的信息。
- 第 13 章“动画、滤镜和绘画”介绍了如何通过使用 ActionScript 创建基于代码的动画和图像、向对象添加滤镜和进行绘图。
- 第 14 章“用 ActionScript 创建交互操作”介绍了一些简单方法，这些方法可用来创建具有更强交互性的应用程序，包括控制 SWF 文件的播放时间、创建自定义指针以及创建声音控件。
- 第 15 章“使用图像、声音和视频”介绍了如何在 Flash 应用程序中导入外部媒体文件，如位图图像、MP3 文件、Flash 视频 (FLV) 文件和其它 SWF 文件。本章还概述了如何在应用程序中使用视频，以及如何创建进度条和加载动画。
- 第 16 章“使用外部数据”介绍了如何在应用程序中通过使用服务器端或客户端脚本来处理来自外部源的数据。本章介绍了如何将数据与应用程序集成。
- 第 17 章“了解安全性”说明了 Flash Player 中的安全性，这些安全机制与在本地硬盘上使用 SWF 文件有关。本章还介绍了跨域安全性问题，并说明了如何从服务器或跨域加载数据。
- 第 18 章“调试应用程序”介绍了 Flash 中能使编写应用程序的工作变得更轻松的 ActionScript 调试器。
- 第 19 章“ActionScript 2.0 的最佳做法和编码约定”说明了使用 Flash 和编写 ActionScript 的最佳做法。本章还列出了标准编码惯例（如变量命名）和其它惯例。
- 附录 A“错误消息”列出了可由 Flash 编译器生成的错误消息。
- 附录 B“不赞成使用的 Flash 4 运算符”列出了所有已不赞成使用的 Flash 4 运算符及其结合律。

- 附录 C“键盘键和键控代码值”列出了标准键盘上的所有键，以及用于在 ActionScript 中标识键的相应 ASCII 键控代码值。
- 附录 D“为早期的 Flash Player 版本编写脚本”为您提供指导，帮助您编写在语法上符合播放器目标版本要求的脚本。
- 附录 E“使用 ActionScript 1.0 进行面向对象的编程”提供了有关使用 ActionScript 1.0 对象模型编写脚本的信息。
- 附录 F“术语”列出了使用 ActionScript 语言时的常用术语，并对这些术语进行了说明。

本手册说明如何使用 ActionScript 语言。有关语言元素本身的信息，请参见《ActionScript 2.0 语言参考》。

印刷惯例

本手册使用以下印刷惯例：

- 代码字体 (`Code font`) 指示 ActionScript 代码。
- 粗体代码字体 (`Bold code font`)，通常出现在程序中，指示需要修改的代码或需要为已经添加到 FLA 文件中的代码添加的代码。在某些情况下，它可能会用于突出显示要查看的代码。
- **粗体文本**指示需要键入到用户界面中的数据，例如文件名或实例名称。

在本文档中使用的术语

在本手册中使用了以下术语：

- “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。

ActionScript 术语 （例如方法 和对象）在第 717 页的附录 F “术语”中定义。

复制和粘贴代码

将 “帮助”面板中的 ActionScript 粘贴到 FLA 或 ActionScript 文件中时，必须注意特殊字符。特殊字符包括特殊引号（也称弯引号或智能引号）。ActionScript 编辑器并不对这些字符进行解释，如果尝试在 Flash 中编译代码，将引发错误。

如果引号字符没有显示正确的代码颜色，则可以将其确定为特殊字符。也就是说，如果所有字符串均没有更改为代码编辑器中的颜色，则需要用常规直引号字符替换特殊字符。如果直接在 ActionScript 编辑器中键入单引号或双引号字符，键入的始终是直引号字符。如果代码中有错误的字符类型（特殊引号或弯引号），则编译器（在测试或发布 SWF 文件时）将报告错误并通知您这一情况。

注意	从其它位置（如 Web 页或 Microsoft Word 文档）粘贴 ActionScript 时，也可能会遇到特殊引号。

复制粘贴代码时请注意正确换行。从某些位置粘贴代码时，代码换行位置可能有误。如果怀疑换行符有问题，请确定语法颜色在 ActionScript 编辑器中是否正确。您可能会希望将“操作”面板中的代码与“帮助”面板中的代码进行比较以查看代码是否匹配。尝试在 ActionScript 编辑器中打开 “自动换行”可帮助解决代码中的多余换行符（在 “脚本”窗口中选择 “视图” > “自动换行”，或从 “操作”面板弹出菜单中选择 “自动换行”）。

其它资源

除了介绍 ActionScript 的本手册之外，还有一些有关 Flash 其它主题（如组件和 Macromedia Flash Lite）的手册。在 “帮助”面板中（“帮助” > “Flash 帮助”），可以通过查看默认目录访问每个手册。单击 “清除”按钮可查看每个可用的手册，有关更多信息，请参见第 16 页的 “从何处查找关于其它主题的文档”。

有关其它可用资源的更多信息，请参见以下主题：

- 第 14 页的 “关于范例文件”
- 第 14 页的 “从何处查找 PDF 文件或印刷的文档”
- 第 15 页的 “关于 LiveDocs”
- 第 16 页的 “其它在线资源”
- 第 16 页的 “从何处查找关于其它主题的文档”

关于范例文件

随 Flash 安装了许多可供使用的基于 ActionScript 的范例文件。这些范例文件演示代码如何在 FLA 文件中工作，而这通常是很有用的学习工具。本手册中的章节经常引用这些文件，我们也建议您了解一下硬盘上的范例文件文件夹。

这些范例文件中包括一些应用程序 FLA 文件，这些文件使用了随 Flash 安装的 Flash 常见功能。这些应用程序旨在向新的 Flash 开发人员介绍 Flash 应用程序的功能，并向高级开发人员演示 Flash 功能如何在各种环境中工作。

这些基于 ActionScript 的范例源文件位于硬盘上的 Samples 文件夹中。

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/。

以下面向组件的范例文件包含很多 ActionScript 代码，可能非常有用。这些范例文件也位于硬盘上的 Samples 文件夹中：

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

您也可以从 Internet 上下载其它范例文件。下面的 Web 页包含其它范例文件的链接和说明：www.macromedia.com/go/flash_samples_cn/。

从何处查找 PDF 文件或印刷的文档

如果您希望阅读打印格式的文档，可下载每个 “帮助”手册的 PDF 版本。转到 www.macromedia.com/support/documentation/，并选择您感兴趣的产品。您可以查看或下载手册的 PDF 版本，或链接到手册的 LiveDocs 版本。

通常，还可以购买印刷版的文档。有关最新消息，请转到文档支持站点并选择 Flash Basic 8 或 Flash Professional 8。

关于 LiveDocs

除了从 “帮助”面板访问文档之外，还可以从 LiveDocs 网站进行访问。 LiveDocs 网站包含所有的 “Flash 帮助”页面，还可能包含阐明、更新或更正文档部分内容的评注。在 “帮助”面板中的页面底部单击 “在 LiveDocs 中查看评论”可显示 LiveDocs Web 站点上的相应页面。转到 http://livedocs.macromedia.com 可查看 LiveDocs 格式的所有可用文档列表。

技术作者将对 LiveDocs 网站进行监控。 LiveDocs 的优点之一是可以查看阐明文档或更正软件发布后发现的任何错误或问题的评注。但是不适合在 LiveDocs 中提出帮助请求，例如询问有关代码不能正常工作的问题，或要对有关软件或安装问题发表看法，或询问如何使用 Flash 进行创作等。此处主要用于提供有关文档的反馈 （例如，您注意到可能需要进一步说明的某个句子或段落）。

单击按钮添加 LiveDocs 评论时，要考虑系统可接受的评论类型。请仔细阅读这些指南，否则可能会将您的评论从网站中删除。

有关 Flash 的问题，请在 Macromedia 网站论坛上提出：www.macromedia.com/support/forums/。网站论坛上有许多 Macromedia 员工、Team Macromedia 志愿者、Macromedia 用户组管理员和成员，甚至还有监控这些论坛的技术作者，是提问的最佳场所。

软件工程师不监控 LiveDocs 系统，但他们监控着 Flash 意见簿。如果您在软件中发现错误，或是希望 Flash 有所改进，请在 www.macromedia.com/go/wish 填写意见表。在 LiveDocs 上报告的错误或改进请求将不会添加到正式的错误数据库中。如果希望工程师看到您的错误或请求，必须使用意见表。

粘贴 Web 内容 （包括 LiveDocs）时，请务必注意特殊字符和换行符。 Macromedia 已尽量删除代码范例中的所有特殊字符，但是如果您在粘贴代码时遇到问题，请参见第 13 页的 “复制和粘贴代码”。

其它在线资源

有几种在线资源，它们提供了大量说明、帮助和指导，以帮助您学习 Macromedia Flash 8。请经常查看以下 Web 站点以了解最新信息：

Macromedia **开发人员中心网站** (www.macromedia.com/cn/devnet) 定期进行更新，以提供有关 Flash 的最新信息，同时还提供专家用户的建议、高级主题、示例、提示、教程（包括多方教程）和其它更新。请经常查看此 Web 站点，了解有关 Flash 的最新消息以及如何最有效地使用该程序。

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 社区中其它资源的链接。

Macromedia Weblogs **网站** (http://weblogs.macromedia.com) 提供了 Macromedia 员工和社区 weblog（也称 blog）的列表。

Macromedia **网站论坛** (http://webforums.macromedia.com) 提供了许多论坛，您可以在这里提出关于 Flash、您的应用程序或 ActionScript 语言等方面的具体问题。这些论坛由 Team Macromedia 志愿者监控，Macromedia 员工也经常会访问这些论坛。如果您不确定应到何处求教，或不知道如何解决某一问题，最好先到 Flash 论坛看一看。

Macromedia **社区网站** (www.macromedia.com/community) 定期举办 MacroChats，这是涵盖各种主题的系列实况演示，由 Macromedia 员工或社区成员组织。请经常查看此 Web 站点以获得更新，并请注册参加 Macrochats 聊天。

从何处查找关于其它主题的文档

以下手册提供了有关 ActionScript 2.0 常见主题的其它信息：

- 有关 ActionScript 语言的组成元素的信息，请参见 《ActionScript 2.0 语言参考》。
- 有关在 Flash 创作环境中工作的信息，请参见如何使用帮助。
- 有关使用组件的信息，请参见 《使用组件》。

第 1 章

1 Flash 8 ActionScript 中的新增功能

Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 提供了几项增强功能，使您能够轻松使用 ActionScript (AS) 语言编写可靠的脚本。本章讨论的新功能包括新增的语言元素（请参见第 19 页的 “ActionScript 语言中的添加项”）、经改进的编辑工具（请参见第 24 页的 “ActionScript 编辑更改”）、对安全模型的更改以及其它与 ActionScript 相关的创作工具改进。

有关更多信息，请参见以下主题：

ActionScript 2.0 和 Flash 8 中的新增功能

自从在几年前引入以来，ActionScript 语言已经得到了改进和发展。每一次发布 Flash 新版本时，都会在 ActionScript 语言中添加一些关键字、对象、方法和其它语言元素，还有一些针对 Flash 8 创作环境的 ActionScript 相关改进。 Flash Basic 8 和 Flash Professional 8 在表现功能（如滤镜和混合模式）和应用程序开发方面引入了一些新的语言元素，例如 JavaScript 集成 (ExternalInterface) 及文件输入和输出（FileReference 和 FileReferenceList）。

本节概述了 Flash 8 中新增的或经改进的 ActionScript 语言元素和类，以及与 ActionScript 相关的创作工具改进。有关 ActionScript 2.0 中的特定新增功能列表，请参见第 19 页的 “ActionScript 语言中的添加项”。若要在脚本中使用任何新增的语言元素，必须在发布文档时将 Flash Player 8（默认设置）设置为目标播放器。

Flash Basic 8 和 Flash Professional 8 中同时新增了以下这些功能（另有说明除外）：

- ActionScript 编辑器的增强功能使您可以显示脚本中的隐藏字符。有关更多信息，请参见第 47 页的 “显示隐藏字符”。
- 现在，对于 ActionScript 文件，在 “脚本”窗口和 “动作”面板中均可以使用 “调试”选项。
- 包含 XML 文件和类文件的 Configuration 目录经过了重新组织。有关详细信息，请参见第 57 页的 “随 Flash 8 一起安装的配置文件”。

- 您可以设置首选参数，以在处理应用程序时重新加载修改后的脚本文件，这样可以帮助您避免使用旧版脚本文件，或是用旧版文件覆盖较新的脚本文件。有关更多信息，请参见第 37 页的 “关于 ActionScript 首选参数”。
- Flash Basic 8 和 Flash Professional 8 中都可以使用 “脚本”窗口。这表示现在您在这两个程序中都能创建 ActionScript 文件。
- “脚本助手”（与早期 Flash 版本中的 “标准模式”类似）可以帮助您在不了解语法的情况下编写代码。有关“脚本助手”的更多信息，请参见第 51 页的“关于“脚本助手””。
- 在运行时可以加载新型图像文件，其中包括渐进式 JPEG 图像以及非动画的 GIF 和 PNG 文件。如果加载一个动画文件，则会显示动画的第一帧。
- 可以为库中存储的位图和声音文件分配链接标识符，这表示可以向 “舞台”附加图像或使用共享库中的资产。
- 通过缓存实例的位图图像，位图缓存可帮助提高应用程序的运行时性能。可以使用 ActionScript 代码来访问此属性。有关更多信息，请参见第 431 页的 “关于位图缓存、滚动和性能”。
- 9 切片缩放使您无需加宽勾勒影片剪辑的笔触即可缩放影片剪辑实例。在 Flash Basic 8 和 Flash Professional 8，或 Flash 8 创作工具中，可以使用 ActionScript 代码来访问此功能。有关更多信息，请参见第 499 页的 “在 ActionScript 中使用 9 切片缩放”。有关如何访问创作工具中 9 切片缩放功能的信息，请参见 《使用 Flash》中的第 69 页的 “关于 9 切片缩放和影片剪辑元件”。
- 现在可以在 “发布设置”对话框中向 FLA 文件添加元数据信息。可以使用此对话框为 FLA 文件添加一个名称和描述，从而提高联机搜索的可见性。
- “字符串”面板经过了改进，添加了对 String 字段和语言 XML 文件的多行支持。有关更多信息，请参见第 405 页的 “关于 “字符串”面板”。
- Flash Player 中内置了一个新的垃圾回收器，该垃圾回收器使用了增量回收器来提高性能。
- 创建可访问应用程序的工作流程得到了改进。在 Flash Player 8 中，开发人员无需将全部对象都添加到内容的 Tab 键索引，屏幕读取器也能正确进行读取。有关 Tab 键索引的更多信息，请参见 《ActionScript 2.0 语言参考》中的 `tabIndex (Button.tabIndex 属性)`、`tabIndex (MovieClip.tabIndex 属性)` 和 `tabIndex (TextField.tabIndex 属性)`。
- Flash Player 增强了本地文件的安全性，以在运行硬盘上的 SWF 文件时为本地文件提供额外的安全保护。有关本地文件安全的信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。
- 借助 ActionScript 代码，您可以使用 Drawing API 来控制所绘制笔触的线条样式。有关新线条样式的信息，请参见第 489 页的 “使用线条样式”。
- 借助 ActionScript 代码，您可以使用 Drawing API 创建更为复杂的渐变，用来填充各种形状。有关渐变填充的信息，请参见第 488 页的 “使用复杂的渐变填充”。

- 您可以使用 ActionScript 代码对舞台上的对象（例如影片剪辑实例）应用多种滤镜。有关滤镜和 ActionScript 的信息，请参见第 452 页的 “通过 ActionScript 使用滤镜”。
- 您可以使用 FileReference 和 FileReferenceList API 将文件上载到服务器。有关更多信息，请参见第 574 页的 “关于文件上载和下载”。
- 您可以使用 ActionScript 代码访问一些应用颜色和操作颜色的新增高级方法。有关更多信息，请参见《ActionScript 2.0 语言参考》中的第 510 页的 “设置颜色值”和 `ColorTransform (flash.geom.ColorTransform)`。
- 对文本处理也进行了多处改进，其中包括 TextField 和 TextFormat 类的新选项、新属性和新参数。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `TextField` 和 `TextFormat`。
- 您可以使用 ActionScript 代码来访问消除锯齿高级功能 (FlashType)。有关更多信息，请参见第 364 页的 “关于字体呈现和消除锯齿文本”。
- 在测试应用程序时，可以删除 ASO 文件。在创作工具中选择 “控制” > “删除 ASO 文件” 或 “控制” > “删除 ASO 文件和测试影片”。有关信息，请参见第 212 页的 “使用 ASO 文件”。

有关 Flash 8 的 ActionScript 2.0 中所添加的具体类、语言元素、方法和属性的列表，请参见第 19 页的 “ActionScript 语言中的添加项”。

ActionScript 语言中的添加项

本节列出了 Flash 8 中新增的或改进的 ActionScript 语言元素和类添加项。以下类和语言元素是在 Flash Player 8 中新近添加或支持的项。

Flash 8 的 ActionScript 2.0 中添加了下列类：

- BevelFilter 类 （在 flash.filters 包中）使您可以向对象添加斜角效果。
- BitmapData 类 （在 flash.display 包中）使您可以创建和操作任意大小的透明或不透明位图图像。
- BitmapFilter 类 （在 flash.display 包中）是滤镜效果的基类。
- BlurFilter 类使您可以对 Flash 中的对象应用模糊效果。
- ColorMatrixFilter 类 （在 flash.filters 包中）使您可以对 ARGB 颜色和 alpha 值应用转换。
- ColorTransform 类（在 flash.geom 包中）使您可以调整影片剪辑中的颜色值。为了支持此类，在该版本中已不赞成使用 Color 类。
- ConvolutionFilter 类 （在 flash.filters 包中）使您可以应用矩阵卷积滤镜效果。
- DisplacementMapFilter 类 （在 flash.filters 包中）使您可以使用来自 BitmapData 对象的像素值在对象上执行置换。
- DropShadowFilter 类 （在 flash.filters 包中）使您可以向对象添加投影。

- ExternalInterface 类（在 flash.external 包中）使您可以使用 ActionScript 与 Flash Player 容器（承载 Flash 应用程序的系统，例如使用 JavaScript 的浏览器或桌面应用程序）进行通信。
- FileReference 类（在 flash.net 包中）使您可以在用户的计算机和服务器之间上载和下载文件。
- FileReferenceList 类（在 flash.net 包中）使您可以选择一个或多个要上载的文件。
- GlowFilter 类（在 flash.filters 包中）使您可以向对象添加发光效果。
- GradientBevelFilter 类（在 flash.filters 包中）使您可以向对象添加渐变斜角。
- GradientGlowFilter 类（在 flash.filters 包中）使您可以向对象添加渐变发光效果。
- IME 类（在 System 类中）使您可以在 Flash Player 中运用操作系统的输入法编辑器 (IME)。
- Locale 类（在 mx.lang 包中）使您可以控制如何在 SWF 文件中显示多语言文本。
- Matrix 类（在 flash.geom 包中）表示一个转换矩阵，用于确定如何将点从一个坐标空间映射至另一个坐标空间。
- Point 类（在 flash.geom 包中）表示二维坐标系统中的某个位置，其中 x 表示水平轴，y 表示垂直轴。
- Rectangle 类（在 flash.geom 包中）使您可以创建和修改 Rectangle 对象。
- TextRenderer 类（在 flash.text 包中）为嵌入字体提供消除锯齿功能。
- Transform 类（在 flash.geom 包中）收集关于应用于 MovieClip 实例的颜色转换和坐标操作的数据。

Flash 8 中正式添加了对 AsBroadcaster 类的支持。

在 ActionScript 的现有类基础上新增的语言元素、方法和函数有：

- `showRedrawRegions` 全局函数使调试器播放器可以勾勒正在重绘的屏幕区域（正更新的脏区域）。该函数会使播放器显示重绘的内容，但不允许您控制重绘区域。
- Button 类中的 `blendMode` 属性，用于设置按钮实例的混合模式。
- Button 类的 `cacheAsBitmap` 属性，该属性允许您将对象作为实例的内部位图图像缓存。
- Button 类的 `filters` 属性，它是一个索引数组，其中包含着与按钮相关联的每个滤镜对象。
- Button 类的 `scale9Grid` 属性，它是一个矩形区域，用于定义实例的九个缩放区域。
- System.capabilities 类的 `hasIME` 属性，用于指示系统是否安装了 IME。
- Date 类的 `getUTCYear` 属性，该属性根据世界时返回此日期的年份。
- Key 类的 `isAccessible()` 方法会返回一个布尔值，指示根据安全限制，所按下的最后一个键是否可供其它 SWF 文件访问。

- LoadVars 类的 onHTTPStatus 事件处理函数，该处理程序返回从服务器返回的状态代码（例如值 404 表示找不到页面）。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 onHTTPStatus (LoadVars.onHTTPStatus 处理函数)。
- MovieClip 类的 attachBitmap() 方法，用于向影片剪辑附加位图图像。有关信息，请参见《ActionScript 2.0 语言参考》中的 BitmapData (flash.display.BitmapData)。
- MovieClip 类的 beginBitmapFill() 方法，该方法用位图图像填充影片剪辑。
- MovieClip 类中 beginGradientFill() 方法的 spreadMethod、interpolationMethod 和 focalPointRatio 参数。该方法用位图图像填充绘画区域，且可以重复或平铺该位图以填满区域。
- MovieClip 类的 blendMode 属性，用于设置实例的混合模式。
- MovieClip 类的 cacheAsBitmap 属性，该属性允许将对象作为实例的内部位图图像进行缓存。
- MovieClip 类的 filters 属性，一个索引数组，其中包含当前与实例相关的每个滤镜对象。
- MovieClip 类的 getRect() 方法，该方法返回指定实例的最小和最大坐标值属性。
- MovieClip 类的 lineGradientStyle() 方法，用于指定绘制路径时 Flash 使用的渐变线条样式。
- MovieClip 类中 lineStyle() 方法的 pixelHinting、noScale、capsStyle、jointStyle 和 miterLimit 参数。这些参数用于指定在绘制线条时可以使用的线条样式种类。
- MovieClip 类的 opaqueBackground 属性，用于将影片剪辑的不透明背景色设置为 RGB 十六进制值所指定的颜色。
- MovieClip 类的 scale9Grid 属性，一个矩形区域，用于定义实例的九个缩放区域。
- MovieClip 类的 scrollRect 属性，它可使您快速滚动影片剪辑内容，在一个窗口中查看更多的内容。
- MovieClip 类的 transform 属性，用于设置与影片剪辑的矩阵、颜色转换和像素绑定有关的设置。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 Transform (flash.geom.Transform)。
- MovieClipLoader.onLoadComplete 事件处理函数的 status 参数返回从服务器返回的状态代码（例如，值 404 表示找不到页面）。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 onLoadComplete (MovieClipLoader.onLoadComplete 事件侦听器)。
- MovieClipLoader 类的 onLoadError 事件处理函数，当用 MovieClipLoader.loadClip() 加载文件失败时会调用此事件处理函数。

- SharedObject.getLocal() 方法的 secure 参数用于确定对此共享对象的访问是否被限制为通过 HTTPS 连接发送的 SWF 文件。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 getLocal (SharedObject.getLocal 方法)。
- System.security 类的 sandboxType 属性用于指示调用方 SWF 文件在其中操作的安全“沙箱”的类型。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 sandboxType (security.sandboxType 属性)。
- TextField 类的 antiAliasType 属性，用于设置为 TextField 实例所使用的消除锯齿类型。
- TextField class 类的 filters 属性，一个索引数组，其中包含与当前 TextField 实例相关联的每个滤镜对象。
- TextField 类的 gridFitType 属性，用于设置为实例所使用的网格固定类型。有关网格固定和 TextField.gridFitType 的信息，请参见《ActionScript 2.0 语言参考》中的 gridFitType (TextField.gridFitType 属性)。
- TextField 类的 sharpness 属性，用于设置 TextField 实例的字型边缘清晰度。必须将 antiAliasType() 方法设置为 advanced 才能使用此属性。
- TextField 类的 thickness 属性，用于设置 TextField 实例的字型边缘粗细。必须将 antiAliasType() 方法设置为 advanced 才能使用此属性。
- TextFormat 类中 align 属性的 justify 值，用于对齐指定段落。
- TextFormat 类的 indent 属性，该属性允许您使用负值。
- TextFormat 类的 kerning 属性，该属性允许您打开或关闭 TextFormat 对象的字距调整。
- TextFormat 类的 leading 属性，该属性允许您使用负的前导值，以使行间距小于文本高度。这使您可以将应用程序中的文本行紧密地排列在一起。
- TextFormat 类的 letterSpacing 属性，用于指定在字符间统一分配的空格量。
- Video 类的 _alpha 属性，该属性指示为视频对象指定的透明度值。
- Video 类的 _height 属性，用于指示视频实例的高度。
- Video 类的 _name 属性，用于指示视频实例的名称。
- Video 类的 _parent 属性，用于指示包含视频实例的影片剪辑实例或对象。
- Video 类的 _rotation 属性，该属性允许设置视频实例的旋转量 （以度为单位）。
- Video 类的 _visible 属性，该属性允许设置视频实例的可见性。
- Video 类的 _width 属性，该属性允许设置视频实例的宽度。
- Video 类的 _x 属性，该属性允许设置视频实例的 x 坐标。
- Video 类的 _xmouse 属性，该属性允许设置鼠标指针位置的 x 坐标。
- Video 类的 _xscale 属性，该属性允许设置视频实例的水平缩放百分比。
- Video 类的 _y 属性，该属性允许设置视频实例的 y 坐标。

- Video 类的 _ymouse 属性，该属性允许设置鼠标指针位置的 y 坐标。
- Video 类的 _yscale 属性，该属性允许设置视频实例的垂直缩放百分比。
- XML 类的 onHTTPStatus 事件处理函数，该处理程序返回从服务器返回的状态代码（例如，值 404 表示找不到页面）。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 onHTTPStatus (XML.onHTTPStatus 处理函数)。
- XMLNode 类的 localName 属性，该属性返回 XML 节点对象的完整名称（包括前缀和本地名称）。
- XMLNode 类的 namespaceURI 属性，该属性读取 XML 节点前缀所解析到的命名空间的 URI。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 namespaceURI (XMLNode.namespaceURI 属性)。
- XMLNode 类的 prefix 属性，该属性读取节点名称的前缀。
- XMLNode 类的 getNamespaceForPrefix() 方法，该方法返回与指定节点前缀相关联的命名空间 URI。
- XMLNode 类的 getPrefixForNamespace 方法，该方法返回与节点指定命名空间 URI 相关联的前缀。

关于不赞成使用的语言元素

在 Flash Player 8 中已不赞成使用某些语言元素。有关 Flash Player 8 中不赞成使用的语言元素和替代语言元素的列表，请参见《ActionScript 2.0 语言参考》中的下列各节：

- 不推荐使用的类摘要
- 不推荐使用的函数摘要
- 不推荐使用的属性摘要
- 不推荐使用的运算符摘要

ActionScript 编辑更改

“动作”面板和“脚本”窗口中的 ActionScript 编辑器在几方面进行了更新，与以前的工具版本相比，现在的编辑器更加强大，也更易于使用。本节将总结这些更改。

查看隐藏字符 现在，当您在“动作”面板或“脚本”窗口中编写脚本文件时，可以使用“脚本”窗格、“调试器”面板和“输出”面板中的“选项”弹出菜单查看或隐藏那些隐藏字符。有关此功能的信息，请参见第 47 页的“显示隐藏字符”。

“动作”面板中添加了“脚本助手” 在 Flash 的早期版本中，可以在“动作”面板中以标准模式（即通过填写选项和参数来创建代码）或以专家模式（即直接在“脚本”窗格中添加命令）工作。这些选项在 Flash MX 2004 或 Flash MX Professional 2004 中不可用。不过，在 Flash Basic 8 和 Flash Professional 8 中，能以脚本助手模式工作，此模式类似于标准模式，但是比它更可靠。有关“脚本助手”的信息，请参见《使用 Flash》中的第 13 章“使用“脚本助手”编写 ActionScript”。有关“脚本助手”的教程，请参见《使用 Flash》中的第 13 章“使用“脚本助手”创建 startDrag/stopDrag 事件”。

重新加载修改过的文件 在处理应用程序时您可以重新加载修改后的脚本文件。届时会出现一条警告消息，提示您将重新加载与您正在使用的应用程序相关联的修改后脚本文件。对于同时进行应用程序开发的工作小组来说，此功能格外有用，因为它可帮助您避免使用过时的旧脚本，或是用旧版脚本覆盖较新的脚本。如果脚本被移至它处或被删除，便会出现一个警告消息，提示您根据需要保存文件。有关更多信息，请参见第 37 页的“关于 ActionScript 首选参数”。

对本地安装 SWF 文件安全模型的更改

Flash Player 8 采有了经过改进的新安全模型，借助此模型，本地计算机上的 Flash 应用程序和 SWF 文件可以与 Internet 和本地文件系统通信，而不是从远程 Web 服务器上运行。在您开发 Flash 应用程序时，必须指出是否允许该 SWF 文件与网络或本地文件系统进行通信。

注释 在本段描述中，本地 SWF 文件是指在用户计算机本地安装的 SWF 文件，它不接受网站提供的服务，也不包括放映 (EXE) 文件。

在以前的 Flash Player 版本中，本地 SWF 文件可与任何远程计算机或本地计算机上的其它 SWF 文件进行交互，无需配置任何安全设置。在 Flash Player 8 中，如果不配置安全设置，同一应用程序中的 SWF 文件中将不能与本地文件系统和网络（例如 Internet）建立连接。这是为了您的安全考虑，因此 SWF 文件不能读取您硬盘上的文件，也不能将这些文件的内容在 Internet 上发送。

这种安全限制会影响本地部署的所有内容，而无论其是早期版本内容 （用早期版本的 Flash 创建的 FLA 文件）还是在 Flash 8 中创建的文件。使用 Flash MX 2004 或更早的创作工具，可以测试在本地运行且访问 Internet 的 Flash 应用程序。在 Flash Player 8 中，这样的应用程序现在将提示用户指定是否允许与 Internet 通信。

在测试硬盘上的文件时，请遵循几个步骤，以确定该文件是一个受信任 （安全）的本地文档，还是一个潜在的不受信任 （不安全）的文档。如果您是在 Flash 创作环境 （例如，选择 “控制” > “测试影片”）中创建的文件，则该文件将是一个受信任的文件，因为它本身处于测试环境中。

在 Flash Player 7 以及更早的版本中，本地 SWF 文件具有访问本地文件系统和网络的权限。在 Flash Player 8 中，本地 SWF 文件可以有三种不同的权限级别：

- 仅能访问本地文件系统 （默认级别） 。本地 SWF 文件可以读取本地文件系统和统一命名约定 (UNC) 网络路径上的文件，但不能与 Internet 进行通信。
- 仅能访问网络。本地 SWF 文件只能访问网络 （例如 Internet），但不能访问在其上安装该 SWF 文件的本地文件系统。
- 可以同时访问本地文件系统和网络。本地 SWF 文件可以从安装 SWF 的本地文件系统进行读取，可以从授予其权限的任何服务器中读取或向其中写入，还可以与授予其权限的网络或本地文件系统上的其它 SWF 文件串联脚本。

有关每种权限级别的更多详细信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。

该版本中对 System.security.allowDomain 也有一些小的改动，同时还改进了 System.security.allowInsecureDomain。有关本地文件安全的更多信息，请参见第 17 章 “了解安全性”

第 2 章

编写和编辑 ActionScript 2.0

2

在 Macromedia Flash Basic 8 或 Macromedia Flash Professional 8 中编写 ActionScript 代码时，应使用 “动作”面板或 “脚本”窗口。“动作”面板和 “脚本”窗口包含全功能代码编辑器（称为 ActionScript 编辑器），其中包括代码提示和着色、代码格式设置、语法加亮显示、语法检查、调试、行数、自动换行等功能，并在两个不同视图中支持 Unicode。有关 ActionScript 编辑器的更多信息，请参见第 31 页的“使用“动作”面板和“脚本”窗口”。

可以使用以下两个方法中的任意一个在 Flash 中编写 ActionScript 代码。可以编写属于 Flash 文档部分的脚本 （即嵌入在 FLA 文件中的脚本），或编写外部脚本 （存储在外部文件中的脚本和类）。不能使用 “动作”面板编写外部脚本。

在 FLA 文件中编写脚本时，要使用 “动作”面板中的 ActionScript 编辑器。“动作”面板中的 “脚本”窗格内包含 ActionScript 编辑器，且面板中还支持各种工具，更便于脚本编写。这些工具中包括：“动作”工具箱 使您能够快速访问核心 ActionScript 语言元素；“脚本导航器” 可以帮助您在文档中的所有脚本之间导航；以及 “脚本助手”模式 向您提示创建脚本所需的元素。有关 “动作”面板的更多信息，请参见第 32 页的 “关于 “动作”面板”。有关 “脚本助手”的更多信息，请参见第 51 页的 “关于 “脚本助手””。

需要创建外部脚本时，应使用 “脚本”窗口中的 ActionScript 编辑器创建新的 ActionScript 文件 （也可以使用您喜爱的文本编辑器创建外部 AS 文件）。在 “脚本”窗口中，ActionScript 编辑器具有代码帮助功能，例如代码提示和着色、语法检查等，与上述 “动作”面板类似。有关 “脚本”窗口的更多信息，请参见第 33 页的 “关于 “脚本”窗口”。

Flash 还通过各种行为提供了进一步的编写脚本帮助。行为是一些预定义的 ActionScript 函数，您可以将它们附加到您的 Flash 文档中的对象上，而无须自己创建 ActionScript 代码。有关行为的更多信息，请参见第 54 页的 “关于行为”。

有关处理事件的更多信息，请参见以下各节：

关于 ActionScript 和事件

在 Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 中，事件发生时会执行 ActionScript 代码：例如，在加载影片剪辑时、在进入时间轴上的关键帧时或者在用户单击某个按钮时。事件可以由用户或系统触发。用户单击鼠标按钮或按键；在满足特定条件或进程完成（SWF 文件加载、时间轴到达特定的帧、图形完成下载等）时，系统会触发相关事件。

事件发生时，您应编写一个事件处理函数，从而在该事件发生时让一个动作响应该事件。了解事件发生的时间和位置将有助于您确定在什么位置、以什么样的方式用一个动作响应该事件，以及在各种情况下分别应该使用哪些 ActionScript 工具。有关更多信息，请参见第 30 页的 “关于通过编写脚本来处理事件”。

事件可以划分为以下几类：鼠标和键盘事件，发生在用户通过鼠标和键盘与 Flash 应用程序交互时；剪辑事件，发生在影片剪辑内；帧事件，发生在时间轴上的帧中。

有关可以编写处理事件的脚本的种类的信息，请参见第 30 页的 “关于通过编写脚本来处理事件”。

鼠标和键盘事件

用户与 SWF 文件或应用程序交互时触发鼠标和键盘事件。例如，当用户滑过一个按钮时，将发生 `Button.onRollOver` 或 `on(rollOver)` 事件；当用户单击某个按钮时，将发生 `Button.onRelease` 事件；如果按下键盘上的某个键，则发生 `on(keyPress)` 事件。可在帧上编写代码或向实例附加脚本，以处理这些事件以及添加所需的所有交互操作。

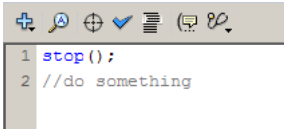
剪辑事件

在影片剪辑中，您可以响应用户进入或退出场景或使用鼠标或键盘与场景进行交互时触发的多个剪辑事件。例如，可以在用户进入场景时将外部 SWF 文件或 JPG 图像加载到影片剪辑中，或允许用户使用移动鼠标的方法在场景中调整元素的位置。

帧事件

在主时间轴或影片剪辑时间轴上，当播放头进入关键帧时会发生系统事件 -- 这叫做帧事件。帧事件可用于根据时间的推移（沿时间轴移动）触发动作或与舞台上当前显示的元素交互。如果向一个关键帧中添加了一个脚本，则在回放期间到达该关键帧时将执行该脚本。附加到帧上的脚本称为帧脚本。

帧脚本最常见的一种用法是在到达特定的关键帧后停止回放。这是使用 `stop()` 函数实现的。您可以选择一个关键帧，然后将 `stop()` 函数作为脚本元素添加到 “动作”面板中。

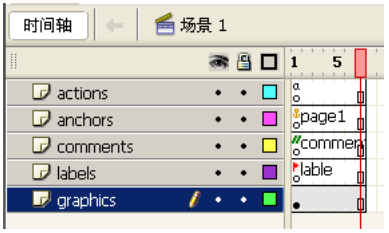


将 SWF 文件停止在特定的关键帧后，您需要执行某种动作。例如，可以使用帧脚本来动态更新某个标签的值、管理舞台上各元素的交互等等。

组织 ActionScript 代码

您可以将脚本附加到关键帧和对象实例（影片剪辑、按钮和其它元件）中。但是，如果您的 ActionScript 代码散布于多个关键帧和对象实例中，调试您的应用程序将非常困难。另外，这还会导致难于在不同的 Flash 应用程序之间共享代码。因此，在 Flash 中创建 ActionScript 时遵循编码最佳做法是非常重要的。

不要将脚本附加到关键帧、影片剪辑和按钮这些元素中，而应通过调用一些驻留在一个中心位置的函数来响应事件。一种办法是，尽可能将嵌入的 ActionScript 附加到时间轴的第一帧或第二帧，这样您就不必搜索 FLA 文件来查找所有代码了。常见的做法是创建一个名为 actions 的图层，并将 ActionScript 代码放置在该图层上。



在将所有脚本附加到各个元素时，同时也会将所有代码嵌入到 FLA 文件中。如果在其它 Flash 应用程序之间共享代码对您来说很重要，请使用 “脚本”窗口或您最喜爱的文本编辑器来创建一个外部 ActionScript (AS) 文件。

通过创建外部文件，您的代码将更加模块化，组织结构也更好。随着项目的扩展，这种简便性带来的好处将超出您的想像。如果您在与其他开发人员共同开发一个项目，外部文件还有助于进行调试和源文件控制管理。

若要使用外部 AS 文件中包含的 ActionScript 代码，您可以在 FLA 中创建一个脚本，然后使用 `#include` 语句来访问存储在外部的代码，如下例所示：

```
#include "../core/Functions.as"
```

您还可以使用 ActionScript 2.0 来创建自定义类。您必须将自定义类存储在外部 AS 文件中，并在脚本中使用 `import` 语句来使类导出到 SWF 文件中，而不要使用 `#include` 语句。有关编写类文件的更多信息，请参见第 170 页的 “编写自定义类文件”；有关导入类文件的更多信息，请参见第 174 页的 “关于导入类文件”。还可以使用组件 （预构建的影片剪辑）共享代码和功能，例如 UI 元素和脚本。

注意

发布、导出、测试或调试 FLA 文件时，外部文件中的 ActionScript 代码将被编译成 SWF 文件。因此，如果对一个外部文件进行了任何更改，则必须保存该文件，并重新编译使用该文件的任何 FLA 文件。

在 Flash 8 中编写 ActionScript 时，应使用 “动作”面板或 “脚本”窗口，或两者同时使用。是使用 “动作”面板还是 “脚本”窗口取决于您如何响应事件、如何组织代码，而最重要的是，取决于编码最佳做法的要求。

有关编码最佳做法和惯例的更多信息，请参见第 663 页的 “ActionScript 编码约定”。

采用行为（预定义的 ActionScript 函数）时（请参见第 54 页的 “关于行为”），必须考虑其它工作流程和代码组织事项。

关于通过编写脚本来处理事件

编写事件代码可以分为两大类：在时间轴上（在关键帧中）发生的事件和在对象实例（影片剪辑、按钮和组件）上发生的事件。 SWF 文件或应用程序的交互可以散布于项目中的多个元素中，您可能很想试一试直接将脚本添加给这些元素。但是，Macromedia 建议不要直接将脚本添加给这些元素 （关键帧和对象）。您应该转而通过调用驻留在一个中心位置的函数来响应事件，如 “组织 ActionScript 代码”中所述。

使用 “动作” 面板和 “脚本” 窗口

若要在 FLA 文件中创建脚本，可以直接将 ActionScript 输入 “动作” 面板。若要创建包含或导入到应用程序中的外部脚本，可以使用 “脚本” 窗口（选择 “文件” > “新建”，再选择 “ActionScript 文件”）或您喜爱的文本编辑器。

使用 “动作” 面板或 “脚本” 窗口时，实际上是在使用 ActionScript 编辑器的功能编写、格式化和编辑代码。“动作” 面板和 “脚本” 窗口都具有 “脚本” 窗格（在其中键入代码）和 “动作” 工具箱。除了 “脚本” 窗口的代码帮助功能之外，“动作” 面板还提供了一些其它功能。Flash 在 “动作” 面板中提供这些功能是因为它们在 FLA 文件中编辑 ActionScript 的上下文时特别有用。

若要显示 “动作” 面板，请执行以下操作之一：

- 选择 “窗口” > “动作”。
- 按下 F9 键。

若要显示 “脚本” 窗口，请执行以下操作之一：

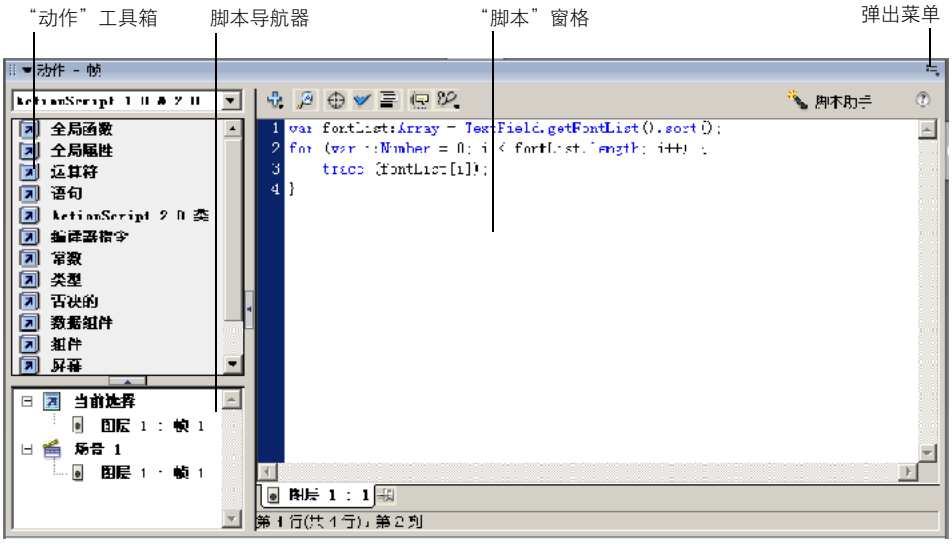
- 若要开始编写新脚本，请选择 “文件” > “新建”，再选择 “ActionScript 文件”。
- 要打开现有脚本，请选择 “文件” > “打开”，然后打开现有 AS 文件。
- 要编辑已打开的脚本，请单击显示该脚本的名称的文档选项卡。

有关更多信息，请参见以下主题：

- 第 32 页的 “关于 “动作” 面板”
- 第 33 页的 “关于 “脚本” 窗口”

关于 “动作”面板

使用 “动作”面板在 Flash 文档 （FLA 文件）中创建 ActionScript。“动作”面板由三个窗格构成，每个窗格都为创建和管理脚本提供支持。



“动作”工具箱 使用 “动作”工具箱浏览 ActionScript 语言元素 （函数、类、类型等）的分类列表，然后将其插入到 “脚本”窗格中。要将脚本元素插入到 “脚本”窗格中，您可以双击该元素，或直接将它拖动到 “脚本”窗格中。还可以使用 “动作”面板工具栏中的 “添加” (+) 按钮来将语言元素添加到脚本中。有关更多信息，请参见第 35 页的 “关于 “动作”面板和 “脚本”窗口工具栏”。

脚本导航器 脚本导航器可显示包含脚本的 Flash 元素 （影片剪辑、帧和按钮) 的分层列表。使用脚本导航器可在 Flash 文档中的各个脚本之间快速移动。

如果单击脚本导航器中的某一项目，则与该项目关联的脚本将显示在 “脚本”窗格中，并且播放头将移到时间轴上的相应位置。如果您双击脚本导航器中的某一项，则该脚本将被固定 （就地锁定）。有关更多信息，请参见第 52 页的 “将脚本固定在 “动作”面板中”。

“脚本”窗格 在 “脚本”窗格中键入代码。“脚本”窗格为您在一个全功能编辑器 （称作 ActionScript 编辑器）中创建脚本提供了必要的工具，该编辑器中包括代码的语法格式设置和检查、代码提示、代码着色、调试以及其它一些简化脚本创建的功能。有关更多信息，请参见第 31 页的 “使用 “动作”面板和 “脚本”窗口”。

有关 “动作”面板工具栏中各个按钮的信息，请参见第 34 页的 “关于在 “动作”面板和 “脚本”窗口中编码”。有关 “动作”面板功能的更多信息，请参见以下主题：

关于 “脚本”窗口

创建新的 ActionScript、Flash Communication 或 Flash JavaScript 文件时，可以在 “脚本”窗口中编写和编辑 ActionScript。应使用 “脚本”窗口编写和编辑外部脚本文件。“脚本”窗口中支持语法着色、代码提示和其它编辑器选项。

可以在 “脚本”窗口中创建外部 ActionScript、ActionScript 通信和 Flash JavaScript 文件。根据您所创建的外部脚本文件的类型，“动作”工具箱将为您提供可用于各种类型的语言元素的完整列表。

在使用 “脚本”窗口时，您会发现有些代码帮助功能（如脚本导航器、“脚本助手”模式和 “行为”）不可用。这是因为这些功能仅在创建 Flash 文档时才有用，在创建外部脚本文件时用不到。

您还会注意到，在 “动作”面板中提供的许多选项在 “脚本”窗口中也不可用。“脚本”窗口支持以下编辑器选项：“动作”工具箱、查找和替换、语法检查、自动套用格式、代码提示和调试选项（仅限于 ActionScript 文件）。另外，“脚本”窗口支持显示行号、隐藏字符和自动换行。

显示 “脚本”窗口：

1. 选择 “文件” > “新建”。
2. 选择要创建的外部文件类型（ActionScript 文件、Flash Communication 文件或 Flash JavaScript 文件）。

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。有关 “脚本”窗口中各项功能的更多信息，请参见以下主题：

- 第 35 页的 “关于 “动作”面板和 “脚本”窗口工具栏”
- 第 36 页的 “关于 ActionScript 编辑选项”
- 第 39 页的 “关于 Flash 中的代码提示”
- 第 44 页的 “设置代码格式”
- 第 45 页的 “使用语法加亮显示”
- 第 46 页的 “使用行号和自动换行”
- 第 46 页的 “使用 Escape 快捷键”
- 第 47 页的 “显示隐藏字符”
- 第 48 页的 “使用 “查找”工具”
- 第 49 页的 “检查语法和标点”
- 第 49 页的 “导入和导出脚本”

关于在 “动作”面板和 “脚本”窗口中编码

可在其中编辑代码的 “脚本”窗格是 “动作”面板和 “脚本”窗口中的主要元素。“动作”面板和 “脚本”窗口提供了基本脚本编辑和代码帮助功能，例如代码提示、着色、自动套用格式等。

通过菜单系统可以从 “动作”面板或 “脚本”窗口的工具栏中访问帮助您编辑代码的功能，也可以从 “脚本”窗格自身中访问。

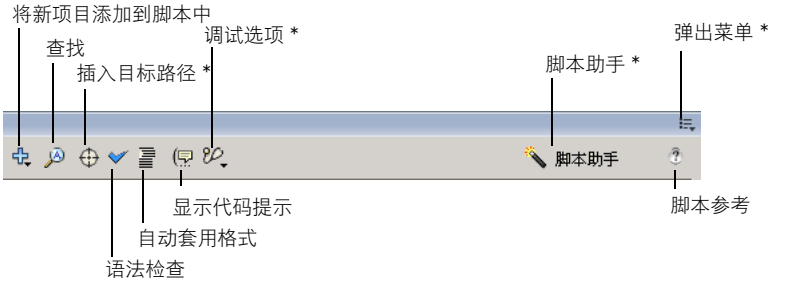
在以下主题中将介绍 ActionScript 编辑器（“动作”面板和 “脚本”窗口）中的很多功能:

- 第 35 页的 “关于 “动作”面板和 “脚本”窗口工具栏”
- 第 36 页的 “关于 ActionScript 编辑选项”
- 第 37 页的 “关于 ActionScript 首选参数”
- 第 39 页的 “关于 Flash 中的代码提示”
- 第 44 页的 “设置代码格式”
- 第 45 页的 “使用语法加亮显示”
- 第 46 页的 “使用行号和自动换行”
- 第 46 页的 “使用 Escape 快捷键”
- 第 47 页的 “显示隐藏字符”
- 第 48 页的 “使用 “查找”工具”
- 第 49 页的 “检查语法和标点”
- 第 49 页的 “导入和导出脚本”

有关 “动作”面板的特定功能 （例如脚本固定和脚本导航器），请参见第 51 页的 “关于 “动作”面板功能”。

关于 “动作”面板和 “脚本”窗口工具栏

“动作”面板和 “脚本”窗口工具栏中包含指向代码帮助功能的链接，有助于简化在 ActionScript 中进行的编码工作。根据您使用的是 “动作”面板还是 “脚本”面板中的 ActionScript 编辑器，工具栏有所不同。下图显示了 “动作”面板工具栏中包含的功能。标出的选项仅在 “动作”面板中可用。



工具栏中的功能将在第 31 页的 “使用 “动作”面板和 “脚本”窗口”中进行详细讨论。随后提供了对 “动作”面板和 “脚本”窗口工具栏中按钮的概要。

附注	以下一些选项仅在 “动作”面板中提供。这些功能将标记为仅限 “动作”面板。

将新项目添加到脚本中 显示 ActionScript 工具箱中也有的所有语言元素。从语言元素的分类列表中选择一项添加到脚本中。

查找 在 ActionScript 代码中查找和替换文本。有关更多信息，请参见第 48 页的 “使用 “查找”工具”。

插入目标路径 仅限 “动作”面板。帮助您为脚本中的某个动作设置绝对或相对目标路径。有关更多信息，请参见第 53 页的 “插入目标路径”。

语法检查 检查当前脚本中的语法错误。语法错误列在 “输出”面板中。有关更多信息，请参见第 49 页的 “检查语法和标点”。

自动套用格式 设置您的脚本的格式以实现正确的编码语法和更好的可读性。可以在 “首选参数”对话框中设置自动套用格式首选参数，从 “编辑”菜单或通过 “动作”面板弹出菜单可访问此对话框。有关更多信息，请参见第 44 页的 “设置代码格式”。

显示代码提示 如果您已经关闭了自动代码提示，可以使用 “显示代码提示”手动显示您正在编写的代码行的代码提示。有关更多信息，请参见第 51 页的“关于“脚本助手””。

调试选项 在脚本中设置和删除断点，以便在调试 Flash 文档时可以停止然后逐行跟踪脚本中的每一行。与 “动作”面板中一样，现在 “脚本”窗口中也提供了调试选项，但仅限于 ActionScript 文件。此选项对于 ActionScript Communication 和 Flash JavaScript 文件禁用。有关调试 Flash 文档的更多信息，请参见第 633 页的 “调试脚本”。有关设置和删除断点的信息，请参见第 641 页的 “设置和删除断点”。

脚本助手 仅限 “动作”面板。在 “脚本助手”模式中，将提示您输入创建脚本所需的元素。有关更多信息，请参见第 51 页的 “关于 “脚本助手””。

参考 显示针对 “脚本”窗格中选中的 ActionScript 语言元素的参考帮助主题。例如，如果单击 `import` 语句，再单击“参考”，将在“帮助”面板中显示 `import` 的帮助主题。

弹出菜单 仅限 “动作”面板。包含许多适用于 “动作”面板或 “脚本”窗口的命令和首选参数。例如，您可以在 ActionScript 编辑器中设置行号和自动换行，访问 ActionScript 首选参数，以及导入或导出脚本。有关更多信息，请参见第 36 页的 “关于 ActionScript 编辑选项”。

关于 ActionScript 编辑选项

“脚本”窗口和 “动作”面板中还提供了许多代码帮助功能，这些工具大大方便了脚本的编写和维护。这些工具选项可从 “动作”面板或 “脚本”窗口工具栏和 “动作”面板弹出菜单中获得。在 “脚本”窗口中编辑 ActionScript 时，工具栏和 Flash 菜单系统中会提供这些选项。

“动作”面板提供了比 “脚本”窗口中更多的选项。这是因为这些附加选项在创建嵌入到 Flash 文档中的 ActionScript 的上下文时才有用，但在编写外部 ActionScript 文件时用不到。有关这些选项中哪些在 “脚本”窗口中可用的信息，请参见第 33 页的 “关于 “脚本”窗口”。

“脚本”窗口和 “动作”面板中可用的选项将在第 35 页的 “关于 “动作”面板和 “脚本”窗口工具栏”中讨论。

以下选项可以从 “动作”面板弹出菜单以及 “脚本”窗口中的各种菜单中获得。

注释 以下一些选项仅出现在 “动作”面板中。这些功能标记为仅限 “动作”面板。

重新加载代码提示 仅限 “动作”面板。如果通过编写自定义方法来自定义 “脚本助手”模式，则可以重新加载代码提示，而无需重启 Flash 8。

固定脚本 仅限 “动作”面板。固定（就地锁定）当前显示在 “脚本”窗格中的脚本。有关更多信息，请参见第 52 页的 “将脚本固定在 “动作”面板中”。

关闭脚本 仅限 “动作”面板。关闭当前打开的脚本。

关闭所有脚本 仅限 “动作”面板。关闭当前打开的所有脚本。

转到行 在 “脚本”窗格中找到并加亮显示指定的行。

查找和替换 在 “脚本” 窗格中查找和替换脚本中的文本。有关更多信息，请参见第 48 页的 “使用 “查找” 工具”。

再次查找 重复查找您在 “查找” 工具中输入的最后一个搜索字符串。有关更多信息，请参见第 48 页的 “使用 “查找” 工具”。

导入脚本 允许您将脚本文件 (ActionScript) 导入到 “脚本” 窗格中。有关更多信息，请参见第 50 页的 “导入和导出首选参数”。

导出脚本 将当前的脚本导出到外部 ActionScript (AS) 文件中。有关更多信息，请参见第 50 页的 “导入和导出首选参数”。

Esc 快捷键 快速将常见的语言元素和语法结构输入到脚本中。例如，在 “脚本” 窗格中按 Esc+g+p 时，`gotoAndPlay()` 函数将插入到脚本中。从 “动作” 面板弹出菜单中选择 “Esc 快捷键” 选项后，所有可用的 Escape 快捷键都会出现在 “动作” 工具箱中。有关更多信息，请参见第 46 页的 “使用 Escape 快捷键”。

隐藏字符 查看脚本中的隐藏字符。隐藏的字符有空格、制表符和换行符。有关更多信息，请参见第 47 页的 “显示隐藏字符”。

行号 在 “脚本” 窗格中显示行号。有关更多信息，请参见第 46 页的 “使用行号和自动换行”。

首选参数 仅限 “动作” 面板。显示 ActionScript “首选参数” 对话框。有关更多信息，请参见第 37 页的 “关于 ActionScript 首选参数”。

自动换行 要使超出 “脚本” 窗口当前大小的脚本自动换行，请从 “动作” 面板弹出菜单中选择 “自动换行”。如果使用的是 “脚本” 窗口，请从 “查看” 菜单中选择 “自动换行”。有关更多信息，请参见第 46 页的 “使用行号和自动换行”。

将 “动作” 组合至 仅限 “动作” 面板。允许您将 “动作” 面板 （包括 “动作” 工具箱和脚本导航器）与 Flash 创作环境中的其它面板组合在一起。

另外，“动作” 面板弹出菜单中还包括 “打印”、“帮助” 和面板调整命令。

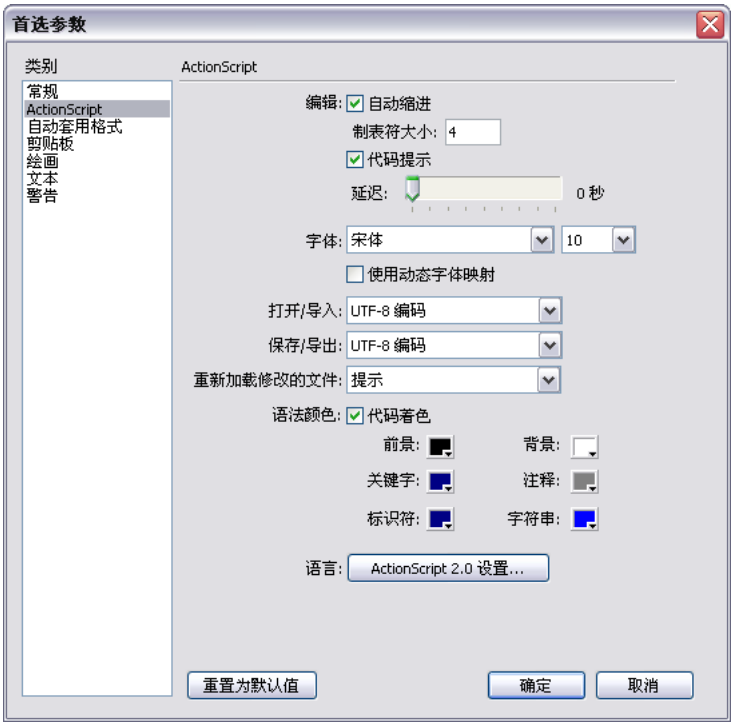
关于 ActionScript 首选参数

无论是在 “动作” 面板还是在 “脚本” 窗口中编辑代码，都可以设置和修改一组首选参数。例如，您可以控制自动缩进、代码提示和着色，以及许多其它基本代码编辑功能。

访问 ActionScript 首选参数：

1. 若要通过 “动作” 面板访问 FLA 文件中的 ActionScript 首选参数，请从弹出菜单中选择 “首选参数”、或选择 “编辑” > “首选参数” (Windows)，或 “Flash” > “首选参数” (Macintosh)，并单击 “类别” 列表中的 “ActionScript”。
2. 若要在 “脚本” 窗口中访问 ActionScript 首选参数，请选择 “编辑” > “首选参数”，然后单击 “ActionScript” (Windows) 或 “Flash” > “首选参数”，然后单击 “ActionScript” (Macintosh)。

下图显示了可以在 Flash 8 中更改的 ActionScript 设置。



可以设置以下首选参数：

自动缩进　如果打开了自动缩进，在左括号 (或左大括号 { 之后键入的文本将按照 ActionScript 首选参数中的 “制表符大小”设置自动缩进。有关更多信息，请参见第 44 页的 “设置代码格式”。

制表符大小　指定自动缩进打开时新行中将偏移的字符数。

代码提示　在“脚本”窗格中启用代码提示。有关使用代码提示的更多信息，请参见第 39 页的 “关于 Flash 中的代码提示”。

延迟　指定代码提示出现之前的延迟（以秒为单位）。

字体　指定 “脚本”窗格中使用的字体。

使用动态字体映射　检查以确保所选的字体系列具有呈现每个字符所必需的字型。如果没有，Flash 会替换上一个包含必需字符的字体系列。有关更多信息，请参见第 44 页的 “设置代码格式”。

编码　指定打开、保存、导入和导出 ActionScript 文件时使用的字符编码。有关更多信息，请参见第 49 页的 “导入和导出脚本”。

重新加载修改过的文件　可以选择何时查看有关脚本文件是否修改、移动或删除的警告。可选的选项包括 “总是”、“从不” 或 “提示”。

- **总是**　发现更改时不显示警告，自动重新加载文件。
- **从不**　发现更改时不显示警告，文件保留当前状态。
- **提示**　（默认）发现更改时显示警告，可以选择是否重新加载文件。

构建涉及到外部脚本文件的应用程序时，因为已打开应用程序，所以此功能可以帮助您避免覆盖团队成员已修改过的脚本，还可以防止使用旧的脚本版本发布应用程序。警告允许您自动关闭脚本，重新打开较新的修改后的版本。

语法颜色　指定脚本中代码着色的颜色。启用代码着色后，可以选择在 “脚本” 窗格中显示的颜色。

语言　打开 “ActionScript 设置” 对话框。有关更多信息，请参见第 55 页的 “修改类路径”。

关于 Flash 中的代码提示

使用 “动作” 面板或 “脚本” 窗口时，可以使用若干功能来帮助您编写语法正确的代码。代码提示帮助您快速而准确地编写代码。代码提示包括工具提示（包含正确的语法）和菜单（允许您选择方法和属性名）。下面各节显示了如何使用这些功能编写代码。

关于触发代码提示

在 “动作” 面板中或 “脚本” 窗口中工作时，Flash 可以检测到正在输入的动作并显示代码提示。有两种不同样式的代码提示：包含该动作的完整语法的工具提示，和列出可能的方法或属性名的弹出菜单（有时被称为代码完成 的一种形式）。当您严格指定对象类型或严格命名对象时，会出现参数、属性和事件的弹出菜单，在本节剩下的部分中将对此进行讨论。

如果您双击 “动作” 工具箱中的某一项或单击 “动作” 面板 / “脚本” 窗口工具栏中的 “添加” (+) 按钮，以向 “脚本” 窗格添加动作，则有时会出现代码提示。有关在代码提示出现时如何使用它们的信息，请参见第 40 页的 “使用代码提示”。

首选参数	将为那些不要求您为其创建和命名类实例的本地类（例如 Math、Key、Mouse 等）自动启用代码提示。

为确保启用代码提示，必须在 ActionScript “首选参数” 对话框中选中 “代码提示” 选项。有关更多信息，请参见第 32 页的 “关于 “动作” 面板”。

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。

若要为自动代码提示指定设置，请执行以下操作之一：

- 在“动作”面板或“脚本”窗口中，选择“编辑”>“首选参数”(Windows) 或“Flash”>“首选参数”(Macintosh)，在“类别”列表中单击“ActionScript”，然后启用或禁用“代码提示”。
- 在“动作”面板中，从弹出菜单（位于“动作”面板的右上角）中选择“首选参数”，然后在 ActionScript 首选参数中启用或禁用“代码提示”。

如果您启用代码提示，则还能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 ActionScript，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。

指定代码提示延迟：

1. 在“动作”面板或“脚本”窗口中，从主菜单中选择“编辑”>“首选参数”(Windows) 或“Flash”>“首选参数”(Macintosh)。
2. 在“类别”列表中单击“ActionScript”。
3. 使用滑块选择延迟的时间。

   延迟时间以秒为单位。

使用工具提示样式的代码提示：

1. 在需要小括号的元素（例如方法名称之后，`if` 或 `do...while` 之类的命令等）后键入一个左小括号 [(] 以显示代码提示。

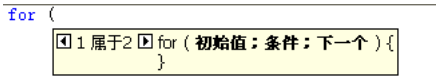
   代码提示就出现了。

提示 如果代码提示未出现，请确保未禁用 ActionScript 首选参数中的“代码提示”（“编辑”>“首选参数”(Windows) 或“Flash”>“首选参数”(Macintosh)，然后在“类别”列表中单击“ActionScript”）。要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 42 页的“关于使用后缀触发代码提示”），或者严格指定了变量或对象的类型（请参见第 42 页的“关于通过指定对象类型触发代码提示”）。

2. 输入参数的值。

   如果有多个参数，则用逗号分隔这些值。对于函数或语句，例如 `for` 循环，则用分号分隔参数。

   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。



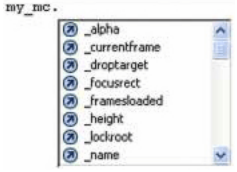
3. 要使代码提示消失，请执行以下操作之一：

   - 键入右小括号 [)]。
   - 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。



提示 如果代码提示未出现，请确保未禁用 ActionScript 首选参数中的 “代码提示”（“编辑” > “首选参数” (Windows) 或 “Flash” > “首选参数” (Macintosh)，然后在 “类别” 列表中单击 “ActionScript”）。若要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 42 页的 “关于使用后缀触发代码提示”），或者严格指定了变量或对象的类型（请参见第 42 页的 “关于通过指定对象类型触发代码提示”）。

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 Enter 键或 Tab 键，或者双击该项。
4. 若要使代码提示消失，请执行以下操作之一：

   - 选择一个菜单项。
   - 单击菜单窗口的上方或下方。
   - 如果已经键入了左小括号 [(]，则再键入一个右小括号 [)]。
   - 按下 Escape 键。

手动显示代码提示：

1. 在可以出现代码提示的代码位置单击，如下面的所示位置：
   - 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）
   - 在方法名称中的小括号 [()] 之间
2. 执行以下操作之一：
   - 在 “动作”面板或 “脚本”窗口工具栏中单击 “显示代码提示”。
   - 按下 Control+ 空格键 (Windows) 或者 Command+ 空格键 (Macintosh)。
   - 如果正在 “动作”面板中工作，请从弹出菜单中选择 “显示代码提示”。

关于通过指定对象类型触发代码提示

在您使用 ActionScript 2.0 时，可以严格指定基于内置类（例如 Button、Array 等）的变量的类型。如果您这样做，“脚本”面板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 就会在一个弹出菜单中显示可用于 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 284 页的 “关于指定数据类型和严格数据类型指定”。有关在代码提示出现时使用代码的信息，请参见第 40 页的 “使用代码提示”。

关于使用后缀触发代码提示

如果您使用的是 ActionScript 1，或者在创建对象时未严格指定类型 （请参见第 42 页的 “关于通过指定对象类型触发代码提示”）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。例如，如果您键入以下代码

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

您可以键入以下两项中的任意一项 （变量名称后面跟有句点）：

```
my_array.
my_cam.
```

将出现 Array 和 Camera 对象的代码提示。

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然后再键入一个句点，就会显示代码提示。

尽管在严格指定对象的类型时不需要使用后缀来触发代码提示，但一直使用后缀有助于理解代码。

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	_array
Button	_btn
Camera	_cam
Color	_color
ContextMenu	_cm
ContextMenuItem	_cmi
Date	_date
Error	_err
LoadVars	_lv
LocalConnection	_lc
Microphone	_mic
MovieClip	_mc
MovieClipLoader	_mcl
PrintJob	_pj
NetConnection	_nc
NetStream	_ns
SharedObject	_so
Sound	_sound
String	_str
TextField	_txt
TextFormat	_fmt
Video	_video
XML	_xml
XMLNode	_xmlnode
XMLSocket	_xmlsocket

有关在代码提示出现时使用它们的信息，请参见第 40 页的 “使用代码提示”。

关于使用注释触发代码提示

您也可以使用 ActionScript 注释来指定对象的类，以用于代码提示。下面的示例将告诉 ActionScript，实例 `theObject` 的类是 Object，依此类推。如果要在这些注释后输入 `mc` 并后跟句点，将出现代码提示，显示 MovieClip 方法和属性的列表。如果要输入 `theArray` 并后跟句点，将出现一个菜单，显示 Array 方法和属性的列表，依此类推。

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

不过，Macromedia 建议使用严格数据类型指定（请参见第 42 页的“关于通过指定对象类型触发代码提示”）或使用后缀（请参见第 42 页的“关于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，从而使代码更易理解。有关使用代码提示的更多信息，请参见第 40 页的 “使用代码提示”。

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。此外，您可以选择是否使用动态字体映射，它可以确保在处理多语言文本时使用正确的字体。

设置格式选项：

1. 在“动作”面板中，从弹出菜单（位于“动作”面板的右上角）中选择“首选参数”。在“首选参数”对话框中，选择“自动套用格式”。

   还可以在“脚本”窗口中选择“编辑”>“首选参数”(Windows) 或“Flash”>“首选参数”(Macintosh)。在“首选参数”对话框中，选择“自动套用格式”。

2. 选择任意“自动套用格式”选项。

   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；您必须手动将这些设置应用于现有代码。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，您必须手动设置代码格式。

若要根据“自动套用格式”设置来设置代码格式，请执行以下操作之一：

- 在“动作”面板或“脚本”窗口工具栏中单击“自动套用格式”按钮。
- 在“动作”面板中，从弹出菜单中选择“自动套用格式”。
- 按下 Control+Shift+F 组合键 (Windows) 或 Command+Shift+F 组合键 (Macintosh)。
- 在“脚本”窗口中，选择“工具”>“自动套用格式”。

使用动态字体映射：

- 若要打开或关闭动态字体映射，请在 “首选参数”对话框中相应选择或取消选择使用动态字体映射。

  由于动态字体映射会在进行脚本撰写时延长运行时间，默认情况下它是关闭的。如果您正在处理多语言文本，则应该打开动态字体映射，这有助于确保使用正确的字体。

使用自动缩进：

- 若要打开或关闭自动缩进，请在 “首选参数”对话框中相应选择或取消选择自动缩进。

  如果打开了自动缩进，在左小括号 [(] 或左大括号 ({) 之后键入的文本将按照 ActionScript 首选参数中的 “制表符大小”设置自动缩进。

  在脚本中，选择相应行并按 Tab 则可以缩进该行。要取消缩进，请选择该行，然后按下 Shift+Tab 组合键。

使用语法加亮显示

在 ActionScript 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 ActionScript 语法，脚本将不会运行。

当您在 Flash Basic 8 和 Flash Professional 8 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 文件版本设置为 Flash 7，仅受 Flash Player 8 支持的 ActionScript 就在 “动作”工具箱中显示为黄色。（有关设置 Flash Player SWF 文件版本的信息，请参见《使用 Flash》中的第 17 章 “为 Flash SWF 文件格式设置发布选项”。）

您还可以设置首选参数，在编写时对脚本进行 Flash 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，用深蓝色显示关键字。在您键入代码时，如果键入 `var`，则单词 `var` 以蓝色显示。但是，如果您错误地键入了 `vae`，则单词 `vae` 将保持为黑色，使您能注意到键入的单词有误。有关关键字的信息，请参见第 86 页的 “关于关键字”。

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” (Windows) 或 “Flash ” > “首选参数” (Macintosh)，在 “类别”列表中单击 “ActionScript”并指定语法着色设置。
- 在 “动作”面板中，从弹出菜单 （位于 “动作”面板的右上角）中选择 “首选参数”，并在 ActionScript 首选参数中指定语法着色设置。
- 将鼠标指针置于 “脚本”窗格中，按下 Control-U (Windows) 或 Command-U (Macintosh)。

您可以更改关键字、注释、标识符和字符串的颜色设置。有关标识符和字符串的信息，请参见第 717 页的 “术语”和第 282 页的 “String 数据类型”。有关注释的信息，请参见第 80 页的 “关于注释”。

使用行号和自动换行

您可以选择是否查看行号以及是否让长代码行换行。通常应该启用行号和自动换行，以便于更加轻松地编辑代码。编辑或修改代码时，行号便于代码滚动和解析。自动换行使您不用水平滚动很长的代码行（特别是在撰写环境中或屏幕分辨率较低时）。

若要启用或禁用行号，请执行以下操作之一：

- 在“动作”面板中，从弹出菜单中选择“行号”。
- 在“脚本”窗口中，选择“工具”>“行号”。
- 按下 Control+Shift+L (Windows) 或 Command+Shift+L (Macintosh)。

若要启用或禁用自动换行，请执行以下操作之一：

- 在“动作”面板中，从弹出菜单中选择“自动换行”。
- 在“脚本”窗口中，选择“工具”>“自动换行”。
- 按下 Control+Shift+W (Windows) 或 Command+Shift+W (Macintosh)。

使用 Escape 快捷键

您可以通过使用 Escape 快捷键（按下 Escape 键，再按下两个别的键），向脚本添加许多元素。

提示	这些快捷键不同于启动某些菜单命令的快捷键。

例如，如果正在“脚本”窗格中进行工作并按下 Escape+d+o，则会将以下代码放置在脚本中：

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

插入点就放置在单词 `while` 后面，您可以在此处开始键入条件。同样，如果您按下 Escape+c+h，则会将以下代码放置在脚本中，插入点就放置在小括号 [`()`] 之间，您可以在此处开始键入条件：

```
catch () {
}
```

如果您要了解（或想得到相关提示）哪些命令具有 Escape 快捷键，则可以在 ActionScript 工具箱中的元素旁显示它们。

显示或隐藏 Escape 快捷键：

- 从“动作”面板弹出菜单中选择或取消选择“Esc 快捷键”。

  Escape 快捷键显示在 ActionScript 工具箱中的元素旁边。

显示隐藏字符

在编写或设置 ActionScript 代码格式时，您会在脚本中输入空格、制表符和换行符。这些对于代码的直观组织结构当然是有好处且是必要的。但是，在遇到不属于字符串值一部分的双字节空格时，Flash 编译器则会生成错误。在“脚本”窗格中显示隐藏的字符使您能够进行查看，然后删除双字节空格。

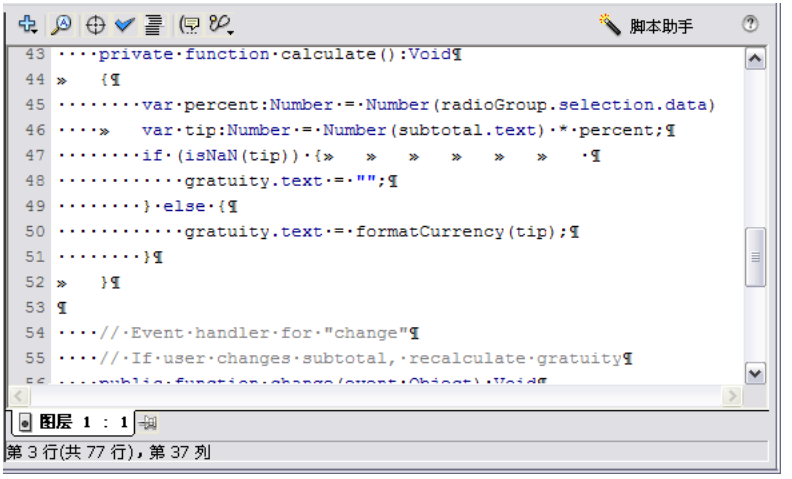
以下符号用于显示每个隐藏的字符：

单字节空格	·
双字节空格		
制表符	»
换行符	¶

若要显示隐藏字符，请执行以下操作之一：

- 从弹出菜单中选择“隐藏字符”。
- 按下 Control+Shift+8 (Windows) 或 Command+Shift+8 (Macintosh)。

显示了隐藏的字符后，“脚本”窗格的外观如下所示：



使用 “查找”工具

“查找”工具允许您查找并根据需要替换脚本中的文本字符串。可以替换所查找的文本在脚本中的第一个实例或所有实例。还可以指定是否要求文本的大小写匹配。

在脚本中查找文本：

1. 从 “动作”面板或 “脚本”窗口工具栏中，选择 “查找”工具或按下 Control+F (Windows) 或 Command+F (Macintosh)。
2. 输入要在脚本中查找的搜索字符串。
3. 单击 “查找下一个”。

   如果脚本中存在要搜索的文本或字符，那么该单词或字符将在“脚本”窗格中加亮显示。

在脚本中查找并替换文本：

1. 从 “动作”面板或 “脚本”窗口工具栏中，单击 “查找”工具或按下 Control+F (Windows) 或 Command+F (Macintosh)。
2. 输入要在脚本中查找和替换的搜索字符串。
3. 在 “替换为”文本框中，输入新字符串。
4. 单击 “查找下一个”。

   如果字符串显示在脚本中，则将加亮显示。
5. 单击 “替换”替换字符串，或单击 “全部替换”替换该字符串出现的全部位置。

在“查找”工具中输入了搜索字符串后，您可以通过从弹出菜单中选择 “再次查找”进行重复搜索。

检查语法和标点

要确定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，您不必退出 FLA 文件就可以迅速检查 ActionScript 代码。语法错误列在 “输出” 面板中。您还可以检查代码块两边的小括号、大括号或中括号是否齐全。

检查语法时，也就检查了当前的脚本。如果当前脚本调用 ActionScript 2.0 的类，则会编译这些类，同时还会检查它们的语法。但不会检查可能位于 FLA 文件中的其它脚本。

要进行语法检查，请执行以下操作之一：

- 在 “动作” 面板或 “脚本” 窗口工具栏中，单击 “语法检查”。
- 在 “动作” 面板中，从弹出菜单中选择 “语法检查”。
- 选择 “脚本” 窗格 （以使其具有焦点），然后按下 Control+T (Windows) 或 Command+T (Macintosh)。

注意

如果在 “脚本” 窗口内单击外部 ActionScript 2.0 类文件中的 “语法检查”，全局类路径将影响此进程。有时将生成错误 （即使全局类路径设置正确），这是因为编译器没有意识到正在编译此类。有关编译类的更多信息，请参见第 210 页的 “编译和导出类”。

要检查标点是否完整，请执行以下操作之一：

- 在脚本中的大括号 ({})、中括号 ([]) 或小括号 [()] 之间单击。
- 按下 Control+' （单引号） (Windows) 或 Command+' （单引号） (Macintosh) 来加亮显示大括号、中括号或小括号之间的文本。

  加亮显示有助于检查开始标点符号是否有与之对应的结束标点符号。

导入和导出脚本

既可能会将脚本导入到 “动作” 面板或 “脚本” 窗口中，也可能会将脚本导出到外部 ActionScript 文件中。这两种做法对于在不同的 Flash 应用程序和开发小组之间共享代码都是有好处的。

导入外部 AS 文件：

- 要将外部脚本导入到您在 “脚本” 窗格中处理的脚本中，请将插入点放在外部脚本的第一行要放置到的位置，然后执行以下两种操作之一：
  - 在 “动作” 面板中，从弹出菜单中选择 “导入脚本” 或按下 Control+Shift+I (Windows) 或 Command+Shift+I (Macintosh)。
  - 在 “脚本” 窗口中，从 “文件” 菜单中选择 “导入脚本” 或按下 Control+Shift+I (Windows) 或 Command+Shift+I (Macintosh)。

可以从 “动作” 面板中导出脚本。使用 “脚本” 窗口时，并非进行导出，因为可以保存 AS 文件。

从 “动作”面板中导出脚本：

1. 选择要导出的脚本，然后从弹出菜单中选择 “导出脚本”或按下 Control+Shift+X (Windows) 或 Command+Shift+X (Macintosh)。

   将显示 “另存为”对话框。

2. 保存该 ActionScript (AS) 文件。

Flash 支持多种不同的字符编码格式 （包括 Unicode），在导入或导出脚本时可以指定要使用的格式。有关更多信息，请参见第 49 页的 “导入和导出脚本”和第 50 页的 “导入和导出首选参数”。

ActionScript 的 Unicode 支持

Flash 8 支持 ActionScript 的 Unicode 文本编码。这意味着您可以在 ActionScript 文件中包含不同语言的文本。例如，您可以在同一文件中包含英语、日语和法语的文本。

注意

在英文系统上使用非英文应用程序时，如果 SWF 文件路径的任何部分具有不能使用多字节字符集 (MBCS) 编码方案表示的字符，则 “测试影片”命令 （请参见第 633 页的 “调试脚本”）将失败。例如，日文路径 （在日文系统上可以使用）就不能在英文系统上使用。该应用程序中所有使用外部播放器的地方都具有此局限性。

导入和导出首选参数

您可以设置 ActionScript 首选参数，以指定在导入或导出 ActionScript 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 ActionScript 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 ActionScript 文件时出现警告消息，则可以更改导出编码首选参数或在 ActionScript 首选参数中关闭此警告。

为导入或导出 ActionScript 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” (Windows) 或 “Flash” > “首选参数” (Macintosh)）中，在 “类别”列表中单击 “ActionScript”。
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 编码”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 编码”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 Flash 系统菜单中，选择 “编辑” > “首选参数” (Windows) 或 “Flash” > “首选参数” (Macintosh)，然后从 “类别” 列表中单击 “警告”。
2. 选择或取消选择 “导出 ActionScript 文件过程中编码发生冲突时发出警告”。

关于 “动作” 面板功能

以下功能仅在 “动作” 面板中可用。而在 “脚本” 窗口中不可用。虽然 “动作” 面板具有 “脚本” 窗口的所有功能，但是 “脚本” 窗口用于不同功能。“动作” 面板必须支持一些 FLA 文件相关功能，这些功能将在以下各节进行介绍。对于 “脚本” 窗口和 “动作” 面板中同时可用的功能，请参见第 34 页的 “关于在 “动作” 面板和 “脚本” 窗口中编码”中的各节。

对于仅在 “动作” 面板中可用的功能，请参见以下各节：

- 第 51 页的 “关于 “脚本助手” ”
- 第 52 页的 “将脚本固定在 “动作” 面板中”
- 第 53 页的 “插入目标路径”

关于 “脚本助手”

“脚本助手” 将提示您输入脚本的元素，有助于您更轻松地向 Flash SWF 文件或应用程序中添加简单的交互性。对于那些不喜欢编写自己的脚本，或者那些喜欢工具所提供的简便性的用户来说，“脚本助手” 模式是理想的选择。

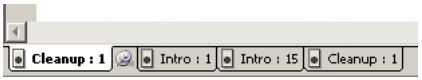
“脚本助手” 与 “动作” 面板配合使用，提示您选择选项和输入参数。例如，不用从头编写脚本，您可以从 “动作” 工具箱中（或使用工具栏上的 “添加” (+) 命令）选择一个语言元素，将它拖动到 “脚本” 窗格中，然后使用 “脚本助手” 帮助完成脚本。

在下面的示例中，`gotoAndPlay` 函数被添加到 “脚本” 窗格中。“脚本助手” 显示使用此 ActionScript 函数所需的所有提示；在本例中显示场景名称、类型和帧编号。

将脚本固定在 “动作”面板中

如果您没有将 FLA 文件中的代码集中放置于一个位置 （在第 29 页的 “组织 ActionScript 代码”中对此进行了讨论），或者如果您正在使用行为 （请参见第 54 页的 “关于行为”），则可以在 “动作”面板中固定 多个脚本，以更易于在脚本之间移动。固定脚本表示将代码开放在 “动作”面板中的某个位置，然后可以方便地在每个开放脚本中进行单击。

在下图中，与时间轴上当前位置关联的脚本位于名为 “Cleanup”的图层的第 1 帧上。（最左侧的选项卡始终在时间轴上您的位置的后面。）该脚本也被固定了 （显示为最右侧的选项卡）。另外两个脚本也被固定了：其中一个位于名为 “Intro”的图层的第 1 帧上，另一个位于第 15 帧上。您可以通过单击这些选项卡或通过使用快捷键如“Control+Shift+.（句点）”在多个固定的脚本之间移动。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。如下图所示，在 “动作”面板中打开了多个脚本，您可以单击每个选项卡，在脚本间移动。



提示

如果 “脚本”窗格中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”窗格左下角上左侧的选项卡可以显示与时间轴上的位置相关联的 ActionScript。

固定脚本：

1. 将您的鼠标指针放在时间轴上，以使脚本出现在 “动作”面板中 “脚本”窗格左下角的选项卡内。
2. 执行以下操作之一：
   - 单击该选项卡右侧的图钉图标。
   - 右键单击 (Windows) 或按住 Control 键并单击 (Macintosh) 该选项卡，然后选择 “固定脚本”。
   - 从弹出菜单 （位于 “动作”面板的右上角）中选择 “固定脚本”。
   - 将鼠标指针置于 “脚本”窗格中，在 Windows 中按下 Control+= （等号）或在 Macintosh 上按下 Command+=。

要解除一个或多个脚本的固定，请执行以下操作之一：

- 如果被固定的脚本出现在 “动作”面板中 “脚本”窗格左下角的选项卡内，则单击该选项卡右侧的图钉图标。
- 右键单击 (Windows) 或者按住 Control 键并单击 (Macintosh) 一个选项卡，然后选择 “关闭脚本”或 “关闭所有脚本”。
- 从弹出菜单 （位于 “动作”面板的右上角）中选择 “关闭脚本”或 “关闭所有脚本”。
- 将鼠标指针置于 “脚本”窗格中，在 Windows 中按下 Control+- （减号）或在 Macintosh 上按下 Command+-。

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+-
将焦点移到右侧的选项卡	Control+Shift+.（句点）	Command+Shift+.
将焦点移到左侧的选项卡	Control+Shift+,（逗点）	Command+Shift+,
取消对所有脚本的固定	Control+Shift+-（减号）	Command+Shift+-

插入目标路径

在脚本中创建的许多动作都会影响影片剪辑、按钮和其它元件实例。要将这些动作应用到时间轴上的实例上，需要设置目标路径 要作为目标的实例地址。可以设置绝对或相对目标路径。

“动作”面板中提供的 “插入目标路径”工具提示您在脚本中输入选中的动作的目标路径。

插入目标路径：

1. 在脚本中选择某一动作并将指针放在该动作上。
2. 单击 “动作”面板工具栏上的 “插入目标路径”。

   出现 “插入目标路径”对话框。
3. 执行以下操作之一：
   - 手动输入目标实例的路径。
   - 从可用目标的列表中选择目标。
4. 选择 “绝对”或 “相对”路径选项。
5. 单击 “确定”。

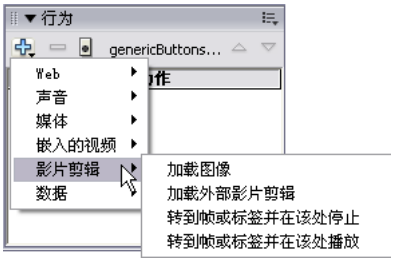
   该路径将附加到动作上。

关于行为

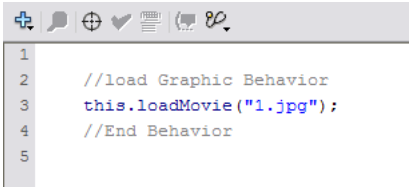
行为是一些预定义的 ActionScript 函数，您可以将它们附加到您的 Flash 文档中的对象上，而无须自己创建 ActionScript 代码。行为提供了预先编写的 ActionScript 功能，例如帧导航、加载外部 SWF 文件和 JPEG、控制影片剪辑的堆叠顺序，以及影片剪辑拖动功能。

行为在构建 Flash 应用程序时可为您提供方便，使用它们可以避免编写 ActionScript；也可以反过来利用其了解 ActionScript 在特定情形下的工作方式。

仅当在 Flash 文档中进行操作时才可以使用行为，在外部脚本文件中不可用。通常，您可以在文档中选择一个触发对象（影片剪辑或按钮），选择“行为”面板上的“添加”按钮以显示可用的行为，然后选择您想要的行为，如下面的示例所示：



这样，该行为就添加到了对象中并显示在“动作”面板中。



关于 ActionScript 发布设置

可用两种方式编辑 ActionScript。使用“动作”面板可以编辑嵌入到 Flash 文档中的 ActionScript。或者可以使用“脚本”窗口编辑在 Flash 文档之外的独立脚本文件中的 ActionScript。因为“动作”面板和“脚本”窗口实质上是使用同一个 ActionScript 编辑器的两种不同视图，所以 Flash 中的 ActionScript 设置和首选参数会应用到这两种视图。

您可以编辑 Flash 文档的发布设置，以更改在发布文档时将使用的 ActionScript 的版本。您还可以绕过全局 ActionScript 类路径，为当前文档设置类路径。

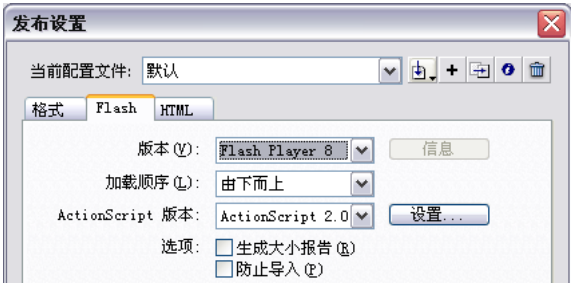
有关修改 ActionScript 发布设置的更多信息，请参见第 55 页的“修改 ActionScript 发布设置”。有关设置文档级或全局级类路径的更多信息，请参见第 55 页的“修改类路径”。

修改 ActionScript 发布设置

默认情况下，发布 Flash 文档时，ActionScript 版本设置为 2.0，而类路径则从全局类路径设置继承。如果需要更改 ActionScript 的版本或指定文档级类路径，则可以通过编辑发布设置来完成。

更改 ActionScript 版本：

1. 选择 “文件” > “发布设置”，然后选择 “Flash” 选项卡。



2. 从弹出菜单中选择 ActionScript 版本。

   默认情况下，ActionScript 2.0 处于选中状态。如果在 ActionScript 1.0 中而不是在 2.0 中编写脚本，应在发布 Flash 文档之前更改此设置。

ActionScript 2.0 编译器可编译所有 ActionScript 1.0 代码，只是有以下例外：如果选择 ActionScript 2.0 作为 ActionScript 版本，则用来表示影片剪辑路径的 “斜杠 (/) 语法”（例如，`parentClip/testMC:varName= "hello world"`）将生成编译错误。要解决此问题，可以用点 (.) 符号替换斜杠重写代码，或者选择 ActionScript 1.0 编译器。

可以使用 “设置” 按钮（位于 “ActionScript 版本” 弹出菜单旁边）修改文档级类路径。有关更多信息，请参见第 55 页的 “修改类路径”。

修改类路径

使用 ActionScript 2.0 时，还可以设置文档级类路径。在创建您自己的类并且想要覆盖在 ActionScript 首选参数中设置的全局 ActionScript 类路径时，这样做很有用。

在发布设置中更改类路径仅适用于当前的 Flash 文件。

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 文件的 “发布设置” 对话框。在上述两种情况下，您都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 “.”）。

修改全局类路径：

1. 选择“编辑”>“首选参数”(Windows) 或“Flash”>“首选参数”(Macintosh)，打开“首选参数”对话框。
2. 在“类别”列表中单击“ActionScript”，然后单击“ActionScript 2.0 设置”。
3. 执行以下操作之一：
   - 要将某个目录添加到类路径中，请单击“浏览到路径”，浏览到要添加的目录，然后单击“确定”。

     也可以单击“添加新路径 (+)”，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”。
   - 要编辑现有的类路径目录，请在“类路径”列表中选择该路径，单击“浏览到路径”，浏览到要添加的目录，然后单击“确定”。

     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”。
   - 要从类路径中删除某个目录，请在“类路径”列表中选择该路径，然后单击“从路径删除”。

注意	不要删除绝对全局类路径（参见“全局和文档级类路径”）。Flash 要使用此类路径访问内置类。如果您意外删除了这个类路径，可将 $(LocalData)/Classes 作为新的类路径进行添加，从而恢复该类路径。

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“ActionScript 版本”弹出菜单旁边的“设置”。
4. 执行以下操作之一：
   - 要将某个目录添加到类路径中，请单击“浏览到路径”，浏览到要添加的目录，然后单击“确定”。

     也可以单击“添加新路径 (+)”，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”。
   - 要编辑现有的类路径目录，请在“类路径”列表中选择该路径，单击“浏览到路径”，浏览到要添加的目录，然后单击“确定”。

     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”。
   - 要从类路径中删除某个目录，请在“类路径”列表中选择该路径，然后单击“从路径删除”。

有关设置和修改类路径的更多信息，请参见第 175 页的“关于设置和修改类路径”。

随 Flash 8 一起安装的配置文件

在安装 Flash Basic 8 或 Flash Professional 8 时，几个与 ActionScript 相关的配置文件夹和文件将放置到您的系统中。可以使用这些文件对创作环境进行特定的配置。和以往一样，请小心修改并保存修改的文件的备份。

ActionScript 类文件夹 此文件夹包含 Flash Professional 8 或 Flash Basic 8 中包括的所有 ActionScript 类（AS 文件）。通常，此文件夹的路径如下：

- Windows：Hard Disk\Documents and Settings\ 用户 \Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\Classes。
- Macintosh: Hard Disk/Users/ 用户 /Library/Application Support/Macromedia/Flash 8/ 语言 /Configuration/Classes。

  Classes 文件夹被组织到包含相应目录的文件夹中，这些目录中包含 Flash Player 7 (FP7) 和 Flash Player 8 (FP8) 的类。还包含 mx 包 (mx) 的一个目录，mx 可用于两种播放器和 ASO 文件 (aso)。有关 ASO 文件的更多信息，请参见第 212 页的“使用 ASO 文件”。有关组织此目录的更多信息，请参见 Classes 文件夹中的自述文件。

Include 类文件夹 包含所有全局 ActionScript include 文件，位于：

- Windows：Hard Disk\Documents and Settings\ 用户 \Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\Include。
- Macintosh：Hard Disk/Users/ 用户 /Library/Application Support/Macromedia/Flash 8/ 语言 /Configuration/Include。

ActionsPanel.xml 配置文件 包括用于 ActionScript 代码提示的配置文件，位于：

- Windows：Hard Disk\Documents and Settings\ 用户 \Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\ActionsPanel\ActionScript_1_2。
- Macintosh：Hard Disk/Users/ 用户 /Library/Application Support/Macromedia/Flash 8/ 语言 /Configuration/ActionsPanel/ActionScript_1_2。

AsColorSyntax.xml 配置文件 用于 ActionScript 代码颜色语法加亮显示的配置文件，位于：

- Windows：Hard Disk\Documents and Settings\ 用户 \Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\ActionsPanel\。
- Macintosh：Hard Disk/Users/ 用户 /Library/Application Support/Macromedia/Flash 8/ 语言 /Configuration/ActionsPanel。

第 3 章

3 关于 ActionScript

ActionScript 2.0 中的面向对象编程 (OOP) 功能是基于当前正由 ECMA TC39-TG1 开发的 ECMAScript 4 草案建议（请参见 www.mozilla.org/js/language/es4/index.html）开发的。由于 ECMA-4 建议尚未成为一个标准，并且仍然在不断更改，ActionScript 2.0 并不严格遵循此规范。

ActionScript 2.0 支持 ActionScript 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。ActionScript 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。ActionScript 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 284 页的 “关于指定数据类型和严格数据类型指定”），并且还提供了已大大改进了的编译器错误（请参见第 687 页的附录 A “错误消息”）。

ActionScript 2.0 的主要特点包括：

- 使用 ActionScript 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 ActionScript 2.0 开发的应用程序。

注意	在 Flash 8 中创建的新文件的默认发布设置是 ActionScript 2.0。如果您计划修改采用 ActionScript 1.0 的现有 FLA 文件以使用 ActionScript 2.0 语法，请确保该 FLA 文件在其发布设置中指定 ActionScript 2.0。如果没有指定，虽然 Flash 将不一定生成编译器错误，但您的文件将不会正确编译。

有关在 Flash 中使用 ActionScript 2.0 编写面向对象程序的更多信息，请参见第 163 页的第 6 章 “类”。

尽管 Macromedia 建议使用 ActionScript 2.0，但您仍然可以继续使用 ActionScript 1.0 语法，特别是当您所进行的是更为传统的 Flash 工作（如不要求用户交互的简单动画）时。

什么是 ActionScript

ActionScript 2.0 包括以下主要功能：

熟悉的面向对象编程 (OOP) 模型 ActionScript 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。ActionScript 2.0 中实现了几个面向对象的概念和关键字，例如，类、接口和包。如果您曾经使用过 Java 编程，那么您对这些概念一定很熟悉。

ActionScript 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”。使用 ActionScript 2.0，可以创建自定义类和扩展 Flash 的内置类。

严格数据类型指定 ActionScript 2.0 还允许您为变量、函数参数和函数返回类型显式指定数据类型。例如，下面的代码声明一个名为 `userName`、类型为 String 的变量，String 是一种内置的 ActionScript 数据类型（即类）。

```
var userName:String = "";
```

编译器警告和错误 以上两种功能（OOP 模型和严格数据类型指定）使创作工具和编译器能够提供编译器警告和错误消息，帮助您用比以往 Flash 中更快的速度找出应用程序中的错误。

在使用 ActionScript 2.0 时，请确保 FLA 文件的发布设置指定为“ActionScript 2.0”。对于在 Flash MX 2004 和 Flash 8 中创建的文件，这是默认设置。但是，如果您打开使用 ActionScript 1.0 的旧版 FLA 文件并用 ActionScript 2.0 对其进行改写，则请将 FLA 文件的发布设置更改为 ActionScript 2.0。如果不这样做，FLA 文件将不会正确编译，也不会生成错误。

关于如何在 ActionScript 1.0 和 ActionScript 2.0 之间进行选择

在 Flash 中启动新的文档或应用程序时，必须决定如何组织其关联文件。在某些项目中您可能会使用类，例如构建应用程序或复杂的 FLA 文件时，但并非所有文档都使用类。例如，文档中的许多简短示例都不使用类。对于小型应用程序或简单的 FLA 文件来说，使用类存储功能并非最容易或最佳的解决方案。将 ActionScript 放入文档中通常会更为有效。在此情况下，可尝试将所有代码放在时间轴上尽可能少的帧上，并避免将代码放在 FLA 文件中的实例（如按钮或影片剪辑）上或实例中。

建立小型项目时，使用类或外部代码文件组织 ActionScript（而不是在 FLA 文件中添加 ActionScript）通常会增大工作量。有时候，将所有 ActionScript 代码保留在 FLA 文件中（而不是放在导入的某个类中）会较为容易。这并不意味着您必须使用 ActionScript 1.0。您可能会决定通过使用具有严格的数据类型指定以及新的方法和属性的 ActionScript 2.0，将代码放在 FLA 文档中。 ActionScript 2.0 还提供一种遵循其它编程语言标准的语法。这使得该语言学习起来更加容易和更具价值。例如，如果遇到基于相同结构和语法标准的另一种语言，您会对 ActionScript 感到非常熟悉。或者，您可以将这些知识应用到将来学习的其它语言中。ActionScript 2.0 允许使用一组附加的语言元素通过面向对象的方法开发应用程序，这对应用程序开发非常有利。

在某些情况下，无法选择要使用的 ActionScript 版本。如果针对旧版 Flash Player 创建 SWF 文件（如移动设备应用程序），则必须使用与 Flash Player 具有多种设备兼容性的 ActionScript 1.0。

请记住，不管是什么版本的 ActionScript，均应遵循一些良好做法。这些做法中有许多做法（例如，区分大小写保持一致、使用代码完成、增强可读性、避免在实例名称中使用关键字以及保持一致的命名约定）对两种版本都适用。

如果计划在 Flash 的未来版本中更新应用程序，或者扩充该应用程序并使其更加复杂，应该使用 ActionScript 2.0 和类，从而可以更容易地更新和修改应用程序。

了解 ActionScript 和 Flash Player

如果您编译一个包含 ActionScript 2.0 的 SWF 文件，且该文件的 “发布设置”设置为 Flash Player 6 和 ActionScript 1.0，只要代码不使用 ActionScript 2.0 类，代码就会正常工作。除了 Flash Player，代码不区分大小写。因此，如果您编译 SWF 文件，且该文件的“发布设置”设置为 Flash Player 7 或 8 和 ActionScript 1.0，则 Flash 会强制区分大小写。

如果“发布设置”设置为 ActionScript 2.0，编译时将强制为 Flash Player 7 和 8 使用数据类型批注（严格数据类型）。

发布应用程序时，ActionScript 2.0 将编译为 ActionScript 1.0 字节码，因此使用 ActionScript 2.0 时可以将 Flash Player 6、7 或 8 设置为目标播放器。

第 4 章

4 语法和语言基础知识

学习 ActionScript 语法和语句就像学习如何将单词放在一起组成句子然后将句子放在一起组成段落一样。 ActionScript 就是这么简单。例如，英语中是用句点结束一个句子，而 ActionScript 中则是用分号结束一条语句。在 ActionScript 语言中，可以键入 `stop()` 动作来停止影片剪辑实例的播放头或使 SWF 文件停止循环。或者，您也可以编写数千行代码来创建一个交互式银行业应用程序。如您所看到的， ActionScript 既可以做非常简单的事情，也可以做非常复杂的事情。

在第 10 章“数据和数据类型”中，您已经学习了 ActionScript 语言是如何使用数据的，并学习了如何在代码中设置数据的格式。本章演示如何使用语法 在 ActionScript 中构成语句。本章包含很多简短的代码片段和演示基本语言概念的一些示例。本章后面的各章包含更长、更复杂的代码示例，这些示例联合运用了您在本章中所学的基础知识，可以帮助您掌握这些基础知识。

本部分描述的一般规则适用于所有 ActionScript。大多数 ActionScript 术语还有自己的一些要求；有关特定术语的规则，请参见 《ActionScript 2.0 语言参考》中的相应条目。

对于 ActionScript 的新用户而言，应用 ActionScript 的基本规则创建设计完美的程序可能是一个挑战。有关如何应用本部分所述规则的更多信息，请参见第 651 页的第 19 章“ActionScript 2.0 的最佳做法和编码约定”。

注释 在本章中，可以将 ActionScript 直接添加到时间轴上的帧上。在后面的各章中，可以使用类将 ActionScript 与 FLA 文件分隔开。

有关使用 ActionScript 语法和语言基础知识的更多信息，请参见以下主题：

关于语法、语句和表达式

ActionScript 语言是由内置类构成的。必须使用正确的 ActionScript 语法 来构成语句，才能使代码在 Flash 中正确地编译和运行。这里，语法是指编程所用的语言的语法和拼写。编译器无法识别不正确的语法，因此，如果语法有问题，当您尝试在测试环境中测试文档时，会在 “输出”面板看到错误或警告。因此，语法是帮助您构成正确 ActionScript 代码的规则和准则的集合。

语句 是告诉 FLA 文件执行操作的指令，例如执行特定的动作。例如，可以使用条件语句确定某一条件是否为 true 或是否成立。然后，可以根据条件的真假执行指定的动作，例如函数或表达式。`if` 语句是一个条件语句，它可以通过对一个条件求值来确定代码中发生的下一个动作。

```
// if 语句
if (condition) {
  // 语句;
}
```

有关语句的更多信息，请参见第 88 页的 “关于语句”。

表达式 与语句不同，它是代表值的 ActionScript 元件的任意合法组合。表达式具有值，而值和属性都有类型。表达式可以包含运算符和操作数、值、函数和过程。表达式遵循 ActionScript 的优先级和关联的规则。通常， Flash Player 解释表达式，然后返回一个可以在应用程序中使用的值。

例如，以下代码是一个表达式：

```
x + 2
```

在上面的表达式中，`x` 和 `2` 是操作数，`+` 是运算符。有关运算符和操作数的更多信息，请参见第 118 页的 “关于运算符”。有关对象和属性的更多信息，请参见第 282 页的 “Object 数据类型”。

设置 ActionScript 的格式的方式还决定着代码的可维护性程度。例如，对于没有缩进或注释、或包含不一致的格式和命名约定的 FLA 文件，其逻辑含义将极难理解。对 ActionScript 的代码块（如循环和 `if` 语句）进行缩进后，代码在遇到问题时更易于阅读和调试。有关设置 ActionScript 格式的更多信息，请参见第 679 页的 “设置 ActionScript 语法的格式”。还可以在这些部分看到正确的 ActionScript 格式设置。

有关语法和语言基础知识的更多信息，请参见以下主题：

- “ActionScript 和 JavaScript 之间的差异”
- “关于区分大小写”

ActionScript 和 JavaScript 之间的差异

ActionScript 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 ActionScript，但是，如果您了解 JavaScript，则会感到 ActionScript 很熟悉。

本手册并不是一本讲解一般编程技术的教材。不过，介绍一般编程概念和 JavaScript 语言的资源很多。

- ECMAScript (ECMA-262) edition 3 语言规范派生自 JavaScript，并作为 JavaScript 语言的国际标准使用。 ActionScript 就是基于该规范。有关更多信息，请参见 www.ecma-international.org/publications/standards/Ecma-262.htm。
- Java 技术站点上提供了关于面向对象编程的教程 (http://java.sun.com/docs/books/tutorial/java/index.html)，这些教程虽然是针对 Java 语言的，但对于理解那些您可以应用到 ActionScript 的概念也很有用。

下表描述了 ActionScript 和 JavaScript 的一些差异：

- ActionScript 不支持特定于浏览器的对象，例如 Document、 Window 和 Anchor。
- ActionScript 没有为所有 JavaScript 内置对象都提供支持。
- ActionScript 不支持某些 JavaScript 语法构造，例如语句标签。
- 在 ActionScript 中，`eval()` 函数只能执行变量引用。
- ActionScript 2.0 支持 ECMA-262 规范中没有的一些功能，例如类和强类型。这些功能中的很多功能都取自 ECMAScript (ECMA-262) 第 3 版语言规范 （请参见 www.ecma-international.org/publications/standards/Ecma-262.htm）。
- ActionScript 不支持使用 RegExp 对象的正则表达式。但是， Macromedia Central 却支持 RegExp 对象。有关 Macromedia Central 的更多信息，请访问 www.macromedia.com/software/central。

关于区分大小写

在为 Flash Player 7 及更高的版本编写 ActionScript 时，代码是区分大小写的。这意味着变量的大小写稍有不同就会被视为是彼此不同的。下面的 ActionScript 代码表明了这一规则：

```
// 使用混合的大小写字母
var firstName:String = "Jimmy";
// 全部使用小写字母
trace(firstname); // undefined
```

或者，您也可以编写以下代码：

```
// 在面向 Flash Player 8
// 以及 ActionScript 1.0 或 ActionScript 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:Number = 10;
var myvar:Number = 10;
var mYvAr:Number = 10;
```

注意 使用大小写的不同来区分变量或任何标识符不是一种好的做法。有关变量命名的更多信息，请参见第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。

针对 Flash Player 的版本（Flash Player 6 和更低版本）进行发布时，Flash 在“输出”面板中跟踪字符串 `Jimmy`。因为 Flash Player 7 和更高版本区分大小写，所以 `firstName` 和 `firstname` 是两个不同的变量（不论是使用 ActionScript 1.0 还是使用 ActionScript 2.0）。这是一个需要理解的重要概念。如果为 Flash Player 6 或更低版本创建的 FLA 文件具有大小写不匹配的变量，则在将该文件或应用程序转换为面向 Flash Player 的更新版本时，功能性和文件可能会受到破坏。

因此，最好遵循一致的大小写约定，如本手册中采用的约定。这样做还更便于区分变量、类和函数名称。不要使用大小写来区别两个标识符。更改实例、变量或类名称 - 不要仅更改大小写。有关编码约定的更多信息，请参见第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。

如果使用的 Web 服务使用它自己的变量命名规则以及将变量从服务器返回到 SWF 文件时的大小写使用规则，则区分大小写具有很大的影响。例如，如果使用 ColdFusion Web 服务，来自某个结构或对象的属性名称可能需要全部大写，如 `FIRSTNAME`。除非在 Flash 中使用同样的大小写，否则可能会遇到意想不到的结果。

注意 区分大小写还会影响加载到 SWF 文件中的外部变量，例如使用 `LoadVars.load()` 加载的外部变量。

外部脚本会区分大小写，例如 ActionScript 2.0 类文件、使用 `#include` 命令导入的脚本，以及 FLA 文件中的脚本。如果遇到运行时错误并且您要导出到多个版本的 Flash Player，则应该同时检查外部脚本文件和 FLA 文件中的脚本，以确认您使用的是一致的大小写形式。

区分大小写是在每个 SWF 文件的基础上实施的。如果一个严格（区分大小写）的 Flash Player 8 应用程序调用非严格的 Flash Player 6 SWF 文件，则在 Player 6 SWF 文件中执行的 ActionScript 是非严格的。例如，如果使用 `loadMovie()` 将 Flash Player 6 SWF 文件加载到 Flash Player 8 SWF 文件中，则第 6 版 SWF 文件保持不区分大小写，而第 8 版 SWF 文件则作为区分大小写处理。

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。有关详细信息，请参见第 86 页的 “关于保留字”。

关于点语法和目标路径

在 ActionScript 中，应使用点 (.) 运算符（点语法）访问属于舞台上的对象或实例的属性或方法。您还可以使用点运算符来确定实例（例如影片剪辑）、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。以下几部分演示了如何书写点语法表达式。

要控制影片剪辑、加载的 SWF 文件或按钮，必须指定目标路径。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。为了指定影片剪辑或按钮的目标路径，您必须为影片剪辑或按钮分配一个实例名称。可以通过在属性检查器中选择该实例并键入实例名称来命名影片剪辑实例。如果您使用 ActionScript 创建实例，则可以使用代码指定实例名称。您可以使用目标路径为影片剪辑分配一个动作，或者获取或设置变量或属性的值。

有关指定实例名称和使用点语法将实例设定为目标的更多信息，请参见以下主题：

- 第 68 页的 “关于使用点语法将实例设定为目标”
- 第 72 页的 “关于作用域和目标设定”
- 第 73 页的 “使用 “插入目标路径”按钮”
- 第 73 页的 “关于斜杠语法”

有关对象和属性的更多信息，请参见第 282 页的 “Object 数据类型”。

关于使用点语法将实例设定为目标

要编写用于控制实例（如影片剪辑）或用于操作加载的 SWF 文件中的资产的 ActionScript，必须在代码中指定其名称和地址。这叫做目标路径。要将 SWF 文件中的对象设定为目标（或寻址到这些对象），请使用点语法（又称点记号）。例如，需要先将一个影片剪辑或按钮实例设定为目标，然后才能对它应用一个动作。点语法可帮助您创建要将其设定为目标的实例的路径。要设定为目标的实例的路径有时称为目标路径。

FLA 文件具有一种特别的层次结构。可以在舞台上创建实例或使用 ActionScript。甚至可以创建包含在其它实例中的实例。或者，您也可以让实例嵌套在多个其它实例中。对于任何实例，只要对它进行了命名，就可以操作它。

您可以用实例名称 命名实例，并且可以用两种不同的方法指定实例名称 （下面将演示这两种方法）：

- 手动指定，具体方法是选择一个实例并在属性检查器中键入实例的名称 （实例在舞台上时）。
- 使用 ActionScript 动态指定。可以使用 ActionScript 创建一个实例，并在创建该实例时为它分配一个实例名称。

要在属性检查器中给实例分配实例名称，请在 “实例名称”文本框中键入名称。

还可以使用 ActionScript 为创建的对象分配一个实例名称。这非常简单，编写以下代码即可:

```
this.createEmptyMovieClip("pic_mc", this.getNextHighestDepth());
pic_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

此代码创建了一个新的影片剪辑，并为它指定了实例名称 pic_mc。然后，可以使用代码操作 `pic_mc` 实例，例如将图像加载到其中，如以上代码中所示。

有关使用作用域的更多信息，请参见第 72 页的 “关于作用域和目标设定”和第 297 页的 “关于变量和作用域”。

将实例设定为目标

如想让某一实例在 SWF 文件中工作，需要将此实例设定为目标并告诉此实例执行什么操作，例如为它指定一个动作或更改其属性。通常需要通过创建一个目标路径来定义该实例在 SWF 文件中的位置（例如，在哪个时间轴上或嵌套在哪个实例中）。请记住，您已经为 FLA 文件中的许多实例分配了实例名称，然后向 FLA 文件中添加了使用这些实例名称的代码。在执行这些操作时，您应将此特定实例设定为目标并告诉此实例应执行什么操作（例如，移动播放头或打开 Web 页）。有关对象和属性的更多信息，请参见第 282 页的 “Object 数据类型”。

将一个实例设定为目标：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 选择 “文件” > “另存为”，将文件命名为 **target.fla**。
3. 使用椭圆工具在舞台上绘制一个形状。绘制一个任意大小和颜色的椭圆。
4. 使用选择工具在舞台上选择该椭圆。

提示	请记着选择笔触和填充 （如果需要）。

5. 选择“修改”>“转换为元件”，选择“影片剪辑”选项，然后单击“确定”创建一个元件。
6. 在舞台上选择影片剪辑，并在属性检查器中为该实例分配一个实例名称 **myClip**。
7. 插入一个新的图层，将其重命名为 **actions**。
8. 在 actions 图层的第 1 帧中添加以下 ActionScript：

```
myClip._xscale = 50;
```

此行代码在舞台上将 `myClip` 实例设定为目标。 ActionScript 将该实例缩小到其原始宽度的一半。因为 ActionScript 与影片剪辑元件在同一个时间轴上，所以只需使用实例名称就可以将实例设定为目标。如果该实例在不同的时间轴上或嵌套在另一个实例内，则需要相应地修改目标路径。

将嵌套实例设定为目标

还可以将嵌套于其它实例中的实例设定为目标。或许，您需要将另一个影片剪辑实例放置到第 68 页的 “将实例设定为目标”练习中的 myClip 实例之内。您还可以使用 ActionScript 将嵌套的实例设定为目标。在继续进行下面的练习之前，需要完成第 68 页的 “将实例设定为目标”中的练习，然后按照以下步骤将嵌套的实例设定为目标。

将嵌套实例设定为目标：

1. 打开在将实例设定为目标的过程中创建的 target.fla，将其重命名为 **target2.fla**。
2. 在舞台上双击 myClip 实例。
3. 选择椭圆工具，在 myClip 实例内部绘制另一个椭圆。
4. 选择所绘新形状，然后选择 “修改” > “转换为元件”。
5. 选择 “影片剪辑”选项，并单击 “确定”。
6. 选择新实例，然后在属性检查器的 “实例名称”文本框中键入 **myOtherClip**。
7. 单击编辑栏中场景 1 以返回主时间轴。

8. 在 actions 图层的第 1 帧中添加以下 ActionScript：

```
myClip.myOtherClip._xscale = 50;
```

此 ActionScript 将 myOtherClip 实例的大小调整为其当前宽度的 50%。因为 target.fla 文件修改了 myClip 实例 _xscale 属性，且 myOtherClip 是一个嵌套元件，所以您将发现 myOtherClip 将变成其原始宽度的 25%。

如果使用具有自己的时间轴的嵌套影片剪辑，可以使用类似于以下代码片断的代码操作嵌套实例的时间轴中的播放头：

```
myClip.nestedClip.gotoAndPlay(15);
myClip.someOtherClip.gotoAndStop("tweenIn");
```

请注意，您操作的剪辑 （例如 nestedClip）恰好在执行动作之前出现。在下面几部分中您会发现这一趋势。

您不限于访问舞台上实例的预定义的方法和属性，如以上示例中所示；您还可以在影片剪辑中设置变量 - 如下面的代码中所示，此代码在 starClip 影片剪辑中设置了一个变量：

```
starClip.speed = 1.1;
starClip.gravity = 0.8;
```

如果在 starClip 影片剪辑实例中以前就存在 speed 或 gravity 变量，则以前的值在设置新值时立即将被覆盖。可以向 starClip 影片剪辑添加新属性，因为 MovieClip 类是使用 dynamic 关键字定义的。dynamic 关键字指定基于指定类（本例中为 MovieClip）的对象可以在运行时添加和访问动态属性。有关 dynamic 语句的更多信息，请参见 《ActionScript 2.0 语言参考》中的 dynamic 语句。

将动态实例和加载的内容设定为目标

还可以使用 ActionScript 创建一个对象，并可在以后使用目标路径将它设定为目标。例如，可以使用下面的 ActionScript 创建一个影片剪辑。然后，可以使用 ActionScript 更改该影片剪辑的旋转角度，如下一个示例中所示。

将动态创建的影片剪辑实例设定为目标：

1. 创建一个新的 Flash 文档，然后将该文件另存为 **targetClip.fla**。
2. 插入一个新的图层，将其重命名为 **actions**。
3. 在 actions 图层的第 1 帧中添加以下 ActionScript：

```
this.createEmptyMovieClip("rotateClip", this.getNextHighestDepth());
trace(rotateClip);
rotateClip._rotation = 50;
```

4. 选择 “控制” > “测试影片” 对文档进行测试。

因为有 trace 语句，所以您可以说您创建了一个影片剪辑，但是在舞台上什么也看不到。虽然您添加了一些可以创建影片剪辑实例的代码，但是除非您向该影片剪辑中添加一些内容，否则在舞台上您什么也看不到。例如，可以向该影片剪辑中加载一个图像。

5. 返回到创作环境，打开 “动作”面板。

6. 在步骤 3 中添加的代码之后键入以下 ActionScript：

```
rotateClip.loadMovie("http://www.helpexamples.com/flash/images/
  image1.jpg");
```

此代码会将一个图像加载到使用代码创建的 rotateClip 影片剪辑中。您在使用 ActionScript 将 rotateClip 实例设定为目标。

7. 选择 “控制” > “测试影片”对文档进行测试。

现在，您会看到这个图像在舞台上按顺时针方向旋转了 50 度。

您还可以将 SWF 文件中已加载到基本 SWF 文件中的那些文件设定为目标或标识那些文件。

标识加载的 SWF 文件：

- 使用 `_levelX`，其中 *X* 是在加载 SWF 文件的 `loadMovie()` 函数中指定的级别数字。

  例如，加载到第 99 级的 SWF 文件的目标路径为 `_level99`。下面的示例将一个 SWF 文件加载到第 99 级中，并将其可见性设置为 `false`：

```
// 将 SWF 加载到第 99 级。
loadMovieNum("contents.swf", 99);
// 将第 99 级的可见性设置为 false。
loaderClip.onEnterFrame = function(){
  _level99._visible = false;
};
```

提示

通常，如果可在不同的深度将内容加载到影片剪辑中，则最好避免使用级别。通过使用 `MovieClip.getNextHighestDepth()` 方法可以在舞台上动态地创建新影片剪辑实例，而不用检查是否在特定的深度上已经存在实例。

使用路径设置变量

可以设置嵌套到其它实例中的实例的变量。例如，如果要设置位于另一个表单中的表单的变量，可以使用下面的代码。实例 submitBtn 在主时间轴上的 `formClip` 的内部：

```
this.formClip.submitBtn.mouseOver = true;
```

可以用此方式表示特定的对象（例如影片剪辑或文本字段）的方法或属性。例如，一个对象的属性可能是

```
myClip._alpha = 50;
```

关于作用域和目标设定

嵌套实例时，嵌套了另一个影片剪辑的影片剪辑被称为被嵌套的实例的父级。被嵌套的实例被称为子实例。主舞台和主时间轴实质上就是影片剪辑本身，因此可以将它们作为影片剪缉设定为目标。有关作用域的更多信息，请参见第 297 页的 “关于变量和作用域”。

可以使用 ActionScript 将父实例和父时间轴设定为目标。在您需要将当前的时间轴设定为目标时，可以使用 `this` 关键字。例如，在将当前主时间轴上名为 myClip 的影片剪辑设定为目标时，可以使用

```
this.myClip.
```

或者，也可以去掉 `this` 关键字，仅使用

```
myClip
```

可以选择添加 `this` 关键字来保证可读性和一致性。有关建议的编码做法的更多信息，请参见第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。

如果跟踪此影片剪辑，则对于以上两个代码片断中的任意一个，您都会看到 `_level0.myClip` 出现在 “输出”面板中。但是，如果 myClip 影片剪辑内部有 ActionScript，而您希望将主时间轴设定为目标，则您需要将该影片剪辑的父级 （即主舞台）设定为目标。双击一个影片剪辑，将下面的 ActionScript 放置到该影片剪辑的时间轴上：

```
trace("me: " + this);
trace("my parent: " + this._parent);
```

测试 SWF 文件，您将看到 “输出”面板中显示以下消息：

```
me: _level0.myClip
my parent: _level0
```

这表示您已将主时间轴设定为目标。可以使用 `parent` 来创建对象的相对路径。例如，如果影片剪辑 `dogClip` 嵌套入动画影片剪辑 animalClip 的内部，则实例 dogClip 上的以下语句会指示 animalClip 停止动画：

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

如果您熟悉 Flash 和 ActionScript，您可能已注意到人们会使用 `_root` 作用域。`_root` 作用域通常是指当前 Flash 文档的主时间轴。除非绝对必要，否则应避免使用 `_root` 作用域。您可以使用相对目标路径来取代 `_root`。

如果在代码中使用 `_root`，在将 SWF 文件加载到另一个 Flash 文档中时可能会遇到错误。在将 SWF 文件加载到一个不同的 SWF 文件中时，加载的文件中的 `_root` 可能指向该文件加载到的 SWF 文件的根作用域，而不是像您所期望的那样指向它自己的根。这可能会导致不可预料的结果，也可能会导致完全无法运行。

使用 “插入目标路径”按钮

有时要花些时间弄清楚给定的目标路径是什么，或者一段代码所需的目标路径是什么。如果要将一个已在舞台上的实例设定为目标，可以使用 “目标路径”按钮来确定该实例的路径是什么。

使用 “插入目标路径”按钮：

1. 打开 “动作”面板（“窗口” > “动作”），单击 “插入目标路径”按钮。对话框中显示当前的文档中的影片剪辑。
2. 从对话框中的列表中选择一个实例。
3. 单击 “确定”。
4. 选定的实例的目标路径会出现在 “脚本”窗格中。

关于斜杠语法

斜杠语法在 Flash 3 和 4 中表示影片剪辑或变量的目标路径。此语法受 Flash Player 7 和更低版本中的 ActionScript 1.0 支持，但是不受 ActionScript 2.0 和 Flash Player 7/Flash Player 8 的支持。

不建议使用斜杠语法，除非没有其它选择，例如，如果要专门为 Flash Player 4 或 Flash Lite 1.1 （和更低的版本）创建内容，则必须使用斜杠语法。有关 Flash Lite 的更多信息，请参见 Flash Lite 产品页。

关于语言标点符号

在 Flash 中有几种语言标点符号。最常用的标点符号种类有分号 (;)、冒号 (:)、小括号 [()] 和大括号 ({})。这些标点符号中的每一种在 Flash 语言中都有特殊的含义，可帮助定义数据类型、终止语句或构造 ActionScript。以下几部分讨论如何在代码中使用标点符号。

有关语言标点符号的更多信息，请参见以下主题：

- 第 74 页的 “分号和冒号”
- 第 75 页的 “大括号”
- 第 78 页的 “小括号”
- 第 79 页的 “关于文本”
- 第 80 页的 “关于注释”

有关点 (.) 运算符和数组访问 ([]) 运算符的更多信息，请参见第 125 页的 “使用点运算符和数组访问运算符” 。有关空白和代码格式设置的更多信息，请参见第 679 页的 “设置 ActionScript 语法的格式”。

分号和冒号

ActionScript 语句以分号 (;) 字符结束，如下面两行代码中所示：

```
var myNum:Number = 50;
myClip._alpha = myNum;
```

可以省略分号字符，ActionScript 编译器会认为每行代码表示单个语句。不过，最好还是使用分号，因为这样可使您的代码可读性更好。在 “动作”面板或 “脚本”窗口中单击 “自动套用格式”按钮后，默认情况下，尾随的分号将附加到语句的结尾。

注释	使用分号终止语句使您能够在单个行中放置不止一条语句，但是这样做往往会使代码难以阅读。

另一个使用分号的地方是 for 循环中。您可以使用分号分隔参数，如下面的示例中所示。该示例从 0 循环到 9，然后在 “输出”面板中显示每个数字：

```
var i:Number;
for (i = 0; i < 10; i++) {
  trace(i); // 0,1,...,9
}
```

在代码中使用冒号 (:) 为变量指定数据类型。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，如下面的示例所示：

```
// 严格指定变量或对象的类型
var myNum:Number = 7;
var myDate:Date = new Date();
// 严格指定参数的类型
function welcome(firstName:String, myAge:Number) {
}
// 严格指定参数和返回值的类型
function square(num:Number):Number {
  var squared:Number = num * num;
  return squared;
}
```

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。下面的代码片段中将创建自定义类型 Student 的一个新对象：

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

还可以将对象指定为 Function 或 Void 数据类型。有关指定数据类型的更多信息，请参见第 275 页的第 10 章 “数据和数据类型”。

大括号

使用大括号 ({}) 标点符号将 ActionScript 事件、类定义和函数组合成块。可以将左大括号与声明放在同一行中。

注意 也可以将左大括号放在声明下边的一行。编码约定建议将左大括号放在同一行中以保持一致性。有关大括号和编码约定的信息，请参见第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。

在构成控制结构的每个语句前后添加大括号 （例如 `if..else` 或 `for`），即使该控制结构只包含一个语句。此良好做法可帮助您避免因忘记给代码添加大括号而导致 ActionScript 出现错误。下面的示例显示使用拙劣的形式编写的代码：

```
var numUsers:Number;
if (numUsers == 0)
  trace("no users found.");
```

尽管此代码有效，但仍然会被视为形式拙劣，因为语句前后不存在大括号。

提示 单击 “检查语法”按钮，大括号将添加到该语句中。

在此情况下，如果在 trace 语句后添加另一个语句，则无论 `numUsers` 变量是否等于 0，另一个语句都会执行，这会导致意想不到的结果。因此，请添加大括号，使代码看起来如下面的示例所示：

```
var numUsers:Number;
if (numUsers == 0) {
  trace("no users found");
}
```

在下面的示例中，将创建一个事件侦听器对象和一个 MovieClipLoader 实例。

```
var imgUrl:String = "http://www.helpexamples.com/flash/images/image1.jpg";
this.createEmptyMovieClip("img_mc", 100);
var mclListener:Object = new Object();
mclListener.onLoadStart = function() {
  trace("starting");
};
mclListener.onLoadInit = function(target_mc:MovieClip):Void {
  trace("success");
};
mclListener.onLoadError = function(target_mc:MovieClip):Void {
  trace("failure");
};
var myClipl:MovieClipLoader = new MovieClipLoader();
myClipl.addListener(mclListener);
myClipl.loadClip(imgUrl, img_mc);
```

下一个示例显示了可用于创建 Student 对象的简单的类文件。可以在第 163 页的第 6 章 “类”中学习关于类文件的更多知识。

在 ActionScript 文件中使用大括号：

1. 选择 “文件” > “新建”并选择 “ActionScript 文件”。
2. 选择 “文件” > “另存为”，将此新文档另存为 **Student.as**。
3. 将以下 ActionScript 添加到该 AS 文件中。

```
// Student.as
class Student {
  private var _id:String;
  private var _firstName:String;
  private var _middleName:String;
  private var _lastName:String;

  public function Student(id:String, firstName:String, middleName:String,
  lastName:String) {
    this._id = id;
    this._firstName = firstName;
    this._middleName = middleName;
    this._lastName = lastName;
  }
  public function get firstName():String {
    return this._firstName;
  }
  public function set firstName(value:String):Void {
    this._firstName = value;
  }
  // ...
}
```

4. 保存该类文件。
5. 选择 “文件” > “新建”，然后单击 “Flash 文档”创建一个新的 FLA 文件。
6. 将该新的 FLA 文件另存为 **student_test.fla**。
7. 在主时间轴的第 1 帧上键入下面的 ActionScript：

```
// student_test.fla
import Student;
var firstStudent:Student = new Student("cst94121", "John", "H.", "Doe");
trace(firstStudent.firstName); // John
firstStudent.firstName = "Craig";
trace(firstStudent.firstName); // Craig
```

8. 选择 “文件” > “保存”以保存对 student_test.fla 所做的更改。
9. 选择 “控制” > “测试影片”对 FLA 和 AS 文件进行测试。

下一个示例演示了使用函数时如何使用大括号。

将大括号用于函数：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”创建一个新的 FLA 文件。
2. 选择 “文件” > “另存为”，将此新文件命名为 **checkform.fla**。
3. 将 Label 组件的实例从 “组件”面板拖到舞台上。
4. 打开属性检查器（“窗口” > “属性” > “属性”），选择该 Label 组件实例，然后在“实例名称”文本框中键入实例名称 **status_lbl**。
5. 在 W（宽度）文本框中键入 **200** 将该组件的大小调整为 200 像素宽。
6. 将 TextInput 组件的实例拖到舞台上，然后为它指定实例名称 **firstName_ti**。
7. 将 Button 组件的实例拖到舞台上，然后为它指定实例名称 **submit_button**。
8. 在时间轴中选择第 1 帧，在 “动作”面板中，添加下面的 ActionScript：

```
function checkForm():Boolean {
  status_lbl.text = "";
  if (firstName_ti.text.length == 0) {
    status_lbl.text = "Please enter a first name.";
    return false;
  }
  return true;
}
function clickListener(evt_obj:Object):Void {
  var success:Boolean = checkForm();
};
submit_button.addEventListener("click", clickListener);
```

9. 选择 “文件” > “保存”以保存该 Flash 文档。
10. 选择 “控制” > “测试影片”在创作环境中对代码进行测试。

    在 SWF 文件中，如果在 `firstName_ti` TextInput 组件中没有加入文本时就单击舞台上的 Button 实例，就会显示一条错误消息。该错误消息出现在 Label 组件中，告知用户他们需要输入名字。

下一个示例使用大括号显示了如何在对象内创建和定义属性。在此例中，属性在对象中是通过在大括号 ({}) 标点符号中指定变量名称来定义的：

```
var myObject:Object = {id:"cst94121", firstName:"John", middleName:"H.",
  lastName:"Doe"};
var i:String;
for (i in myObject) {
  trace(i + ": " + myObject[i]);
}
/*
  id: cst94121
  名: John
  中间名: H.
  姓: Doe
*/
```

还可以使用空的大括号对作为 new Object() 函数的语法快捷方式。例如，以下代码将创建一个空的对象实例：

```
var myObject:Object = {};
```

提示 记住一定让每个左大括号都有和它配对的右大括号。

小括号

在 ActionScript 中定义函数时，将参数放在小括号 [()] 标点符号里面，如下面的几行代码中所示：

```
function myFunction(myName:String, myAge:Number, happy:Boolean):Void {
  // 此处是您的代码。
}
```

调用函数时，还要将传递给该函数的所有参数都包含在小括号中，如下面的示例所示：

```
myFunction("Carl", 78, true);
```

可使用小括号覆盖 ActionScript 的优先顺序或增强 ActionScript 语句的可读性。这意味着可以通过在某些值两边添加中括号来改变计算值的顺序，如下面的示例所示：

```
var computedValue:Number = (circleClip._x + 20) * 0.8;
```

由于优先顺序的存在，如果未使用小括号或使用两个单独的语句，则将首先计算乘法，这意味着首先计算 20 * 0.8。然后结果 16 被添加到 circleClip._x 的当前值中，并最终赋予 computedValue 变量。

如果您不使用小括号，则必须添加一条语句来计算表达式，如下面的示例所示：

```
var tempValue:Number = circleClip._x + 20;
var computedValue:Number = tempValue * 0.8;
```

同使用中括号和大括号一样，需要确保每个左小括号都对应一个右小括号。

关于文本

文本 是直接出现在代码中的值。文本是 Flash 文档中的常数 （保持不变）值。文本的例子包括 true、 false、 0、 1、 52，甚至字符串 “foo”。

下面的例子都是文本：

```
17
"hello"
-3
9.4
null
undefined
true
false
```

文本还可以组合起来构成复合文本。数组文本括在中括号 ([]) 中，使用逗号 (,) 分隔各数组元素。数组文本可用于初始化一个数组。下面的示例显示了使用数组文本进行初始化的两个数组。可以使用 new 语句将复合文本作为参数传递给 Array 类构造函数，还可以在实例化任何内置 ActionScript 类的实例时直接赋予文本值。

```
// 使用 new 语句
var myStrings:Array = new Array("alpha", "beta", "gamma");
var myNums:Array = new Array(1, 2, 3, 5, 8);

// 直接分配文本
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1, 2, 3, 5, 8];
```

文本还可以用于初始化通用对象。通用对象是对象类的一个实例。对象文本括在大括号 ({}) 中，应当使用逗号 (,) 分隔各对象属性。每个属性都用冒号 (:) 声明，冒号将属性的名称与属性的值分隔开了。

可以使用 new 语句创建一个通用对象并将对象文本作为参数传递给对象类构造函数，也可以在声明实例时直接将对象文本赋给实例。下面的示例创建了一个新的通用对象，并将该对象初始化为具有三个属性：propA、 propB 和 propC，每个属性的值分别设置为 1、 2 和 3。

```
// 使用 new 语句
var myObject:Object = new Object({propA:1, propB:2, propC:3});

// 直接分配文本
var myObject:Object = {propA:1, propB:2, propC:3};
```

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 firstStr，而第二行代码创建 String 对象 secondStr：

```
var firstStr:String = "foo"
var secondStr:String = new String("foo")
```

除非您确实需要使用 String 对象实现更好的性能，否则请使用字符串。有关字符串的更多信息，请参见第 404 页的 “关于字符串和 String 类”。

关于注释

注释是一种使用简单易懂的句子对代码进行注解的方法，编译器不会对注释进行求值计算。可以在代码中使用注释来描述代码的作用或描述返回到文档中的数据。注释可帮助您记住重要的编码决定，并且对其他任何阅读您的代码的人也有帮助。注释必须清楚地解释代码的意图，而不是仅仅翻译代码。如果代码中有些内容阅读起来含义不明显，则应对其添加注释。

极力建议使用注释将备注添加到脚本。注释会记录您在代码中所做的决策，解答 “如何做” 和 “为何做” 这两类问题。它们可以使 ActionScript 更容易理解。例如，可以在注释中描述某个问题的临时解决办法。这样，您或其他开发人员就很容易找到代码中需要更新或修复的部分。如果在 Flash 或 Flash Player 的未来版本中纠正了问题或者对有问题的部分做了改进，则可通过删除该临时解决办法来改进 ActionScript。

请避免使用混乱的注释。用一行等号 (=) 或星号 (*) 创建注释块或在注释周围形成隔离标记，就属于混乱的注释。请改用空白来分隔注释与 ActionScript。如果使用 “动作” 面板或 “脚本” 窗口中的 “自动套用格式” 按钮来设置 ActionScript 格式，则会删除空白。请记住将空白添加回代码中，或使用单个注释行 (//) 来维持间距；设置代码格式后，删除这些行要比尝试确定空白原先在什么位置更容易。

部署项目前，请将任何多余注释从代码中删除，例如，“定义变量 x 和 y” 或对其他开发人员来说一眼就能看懂的其它注释。如果发现 ActionScript 中有许多多余的注释，请考虑是否需要改写代码中的某些部分。如果需要包括许多关于代码如何工作的注释，则通常表示 ActionScript 不够完美且不够直观。

如果启用了语法着色，注释在默认情况下为灰色。注释可以具有任意长度，且不会影响导出文件的大小，并且它们不必遵循 ActionScript 语法或关键字的规则。

注意

使用注释在那些用于培训用户的 ActionScript 中最为重要。如果您在创建用于讲授 Flash 技术的应用程序范例，或者在编写关于 ActionScript 的文章或教程，则应当为代码添加注释。

单行注释

单行注释用于为代码中的单个行添加注释。可以注释掉单行代码，也可以为一段代码实现的功能添加一个简短的说明。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 (//)，如下面的代码所示：

```
// 以下代码设置用于表示年龄的本地变量。
var myAge:Number = 26;
```

单行注释通常用于解释很小的代码片断。对于任何单行能写下的短注释，都可以使用单行注释。下面的示例包含单行注释：

```
while (condition) {
  // 处理条件语句
}
```

多行注释

对于长度为几行的注释，可以使用多行注释 （又称 “块注释”）。开发人员通常使用多行注释描述文件、数据结构、方法和文件说明。它们通常放在文件的开头和方法的前面或内部。

要创建注释块，请在注释行开头添加 /*，在注释块末尾添加 */。此方法允许您创建很长的注释，而无需在每一行的开头都添加 //。若对多个连续的行使用 //，在修改注释时可能会产生一些问题。

多行注释的格式如下。

```
/*
   下面的 ActionScript 初始化在主菜单或子菜单系统中使用的变量。变量被用来跟踪单击了哪些选项。
*/
```

提示 如果将注释字符 （/* 和 */）置于注释开头和结尾处的独立的行上，则通过在注释字符之前放置双斜杠字符 (//) 可以很容易地将这些注释字符注释掉 （例如， ///* 和 //*/）。这些使您能够快速且容易地对代码进行注释和取消注释。

通过将大段脚本放在注释块中 （称为注释掉 部分脚本） ，您可以测试脚本的特定部分。例如，当以下脚本运行时，将不执行注释块中的任何代码：

```
// 以下代码运行。
var x:Number = 15;
var y:Number = 20;

// 以下的代码被注释掉了，将不会运行。
/*
// 创建新的 Date 对象
var myDate:Date = new Date();
var currentMonth:Number = myDate.getMonth();
// 将月份数转换为月份名称
var monthName:String = calcMonth(currentMonth);
var year:Number = myDate.getFullYear();
var currentDate:Number = myDate.getDate();
*/

// 以下代码运行。
var namePrefix:String = "My name is";
var age:Number = 20;
```

提示 注释块前面留出一个空行是一种很好的做法。

尾随注释

尾随注释用于在代码所在的行内添加注释。这些注释与 ActionScript 代码出现在同一行。开发人员通常使用尾随注释指示变量包含的值，或描述或注释一行 ActionScript 返回的值。尾随注释的格式如下：

```
var myAge:Number = 26; // 表示年龄的变量
trace(myAge); // 26
```

在代码右边留出空格，然后再写注释，以便读者能区分注释和代码。如果可能，应尝试使注释与注释相互对齐，如下面的代码所示。

```
var myAge:Number = 28;              // 我的年龄
var myCountry:String = "Canada";  // 我的国家（地区）
var myCoffee:String = "Hortons";  // 我最喜爱的咖啡
```

如果使用“自动套用格式”功能（在“动作”面板中单击“自动套用格式”按钮），则尾随注释会移至下一行。请在设置代码格式之后添加这些注释，否则在使用“自动套用格式”按钮之后必须修改这些注释的位置。

类中的注释

在类和接口中使用注释记录，可以帮助开发人员了解类的内容。可以在所有类文件的开头使用一段注释，其中提供类名称、其版本号、日期和您的版权。例如，您可以为类创建类似于以下内容的文档注释：

```
/**
  Pelican 类
  1.2 版
  10/10/2005
  版权所有 Macromedia, Inc.
*/
```

块注释用于描述文件、数据结构、方法和文件说明。它们通常放在文件的开头和方法的前面或内部。

在典型的类或接口文件中有两种注释：文档注释和实现注释。文档注释用于描述代码规范，而不描述实现。可以使用文档注释描述接口、类、方法和构造函数。实现注释用于注释掉代码或对代码特定节的实现提供注释。

应为每个类、接口或成员使用一个文档注释，并应将其紧挨着声明放在声明的前面。如果您有其它不适合放在文档注释中的文档信息，请使用实现注释（采用块注释或单行注释的形式）。实现注释紧随声明之后。

这两种注释使用的分隔符略有区别。文档注释是以 /** 和 */ 分隔的，而实现注释则是以 /* 和 */ 分隔的。

提示	不要添加与正在阅读的类没有直接关联的注释。例如，不要添加用于描述相应包的注释。

在类文件中还可以使用单行注释、块注释和尾随注释。有关这些注释类型的更多信息，请参见以下各部分：

- 第 80 页的 “单行注释”
- 第 81 页的 “多行注释”
- 第 82 页的 “尾随注释”

关于常数和关键字

常数和关键字是 ActionScript 语法的重要组成部分。常数是具有固定值 （无法改变的值）的属性，因此它们是在整个应用程序中都不发生改变的值。

Flash 包含几个预定义的常数，这些常数可以简化应用程序的开发。可以在 Key 类中找到常数的示例，该类包含多个属性，例如 Key.ENTER 或 Key.PGDN。如果您依赖于常数，您不用记着 Enter 和 Page Down 键的键控代码值分别为 13 和 34。使用常数值不仅会使开发和调试变得更容易，而且可以使您的代码更易让您的伙伴开发人员阅读。

在 ActionScript 中关键字用于执行特定种类的动作。基于此原因，它们还是保留字，所以不能将它们用作标识符 （例如变量、函数或标签名称） 。保留关键字的示例包括 if、 else、 this、 function 和 return。

有关常数和关键字的更多信息，请参见以下主题：

- 第 84 页的 “使用常数”
- 第 86 页的 “关于关键字”
- 第 86 页的 “关于保留字”

有关对象和属性的更多信息，请参见第 282 页的 “Object 数据类型”。有关该语言中的常数（例如 false 和 NaN）的列表，请参见《ActionScript 2.0 语言参考》中的“ActionScript 语言元素” > “常数类别”。

使用常数

常数是具有固定值（无法改变的值）的属性；换句话说，它们是在整个应用程序中都不发生改变的值。ActionScript 语言包含多个预定义的常数。例如，常数 BACKSPACE、ENTER、SPACE 和 TAB 是 Key 类的属性，指代键盘的按键。常数 Key.TAB 的含义始终不变：它代表键盘上的 Tab 键。在应用程序中比较和使用不发生变化的值时，常数非常用于。

要测试用户是否按下了 Enter 键，可以使用下面的语句：

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.getCode() == Key.ENTER) {
    trace("Are you ready to play?");
  }
};
Key.addListener(keyListener);
```

要使上面的 ActionScript 正常工作，可能必须在创作环境中禁用快捷键。从主菜单中选择“控制”>“测试影片”，然后，在播放器中预览 SWF 文件的过程中，从 SWF 文件的预览窗口中选择“控制”>“禁用快捷键”。

在 Flash 中无法创建您自己的常数值，除非在创建您自己的具有私有成员变量的自定义类时才允许创建您自己的常数值。在 Flash 中无法创建“只读”变量。

变量应为小写字母或混合大小写的字母，但是常数（不会更改的变量）应为大写字母。应使用下划线分隔单词，如下面的 ActionScript 所示：

```
var BASE_URL:String = "http://www.macromedia.com"; // 常数
var MAX_WIDTH:Number = 10;                          // 常数
```

请以大写形式来书写静态常数，并用下划线来分隔单词。不要直接对数字常数进行编码，除非常数为 1、0 或 -1，这些常数可以在 for 循环中用作计数器值。

要引用某些定值的属性，则可使用常数。这可以帮助您查找代码中的打字错误；但如果使用了文本，则可能找不到这些错误。使用常数还可以在单独一个位置上更改值。有关文本的更多信息，请参见第 79 页的“关于文本”。

例如，下一个示例中的类定义创建了三个常数，遵循 ActionScript 2.0 所用的命名约定。

在应用程序中使用常数：

1. 选择“文件”>“新建”，然后选择“ActionScript 文件”创建一个 AS 文件。
2. 将此新文件命名为 **ConstExample.as**。
3. 在“脚本”窗口中键入下面的代码：

```
class ConstExample {
  public static var EXAMPLE_STATIC:String = "Global access";
  public var EXAMPLE_PUBLIC:String = "Public access";
  private var EXAMPLE_PRIVATE:String = "Class access";
}
```

EXAMPLE_STATIC 属性是一个静态属性，这意味着该属性应用于整个类，而不是应用于类的一个特定实例。必须使用类的名称（而不是实例的名称）访问类的静态属性。无法通过类实例访问静态属性。

4. 创建一个新的 Flash 文档，并将它另存为 **const.fla**。
5. 打开“动作”面板，在时间轴的第 1 帧上键入以下代码：

```
trace(ConstExample.EXAMPLE_STATIC); // 输出: Global access
```

将 EXAMPLE_STATIC 属性声明为静态时，可以使用此代码访问该属性的值。

6. 选择“控制”>“测试影片”对文档进行测试。

将在“输出”面板中看到 Global access。

7. 在“动作”面板中，将以下代码键入到在步骤 5 中添加的代码之后。

```
trace(ConstExample.EXAMPLE_PUBLIC); // 错误
trace(ConstExample.EXAMPLE_PRIVATE); // 错误
```

8. 选择“控制”>“测试影片”对文档进行测试。

EXAMPLE_PUBLIC 和 EXAMPLE_PRIVATE 属性不是静态属性。尝试通过类访问它们的值时，会看到以下错误消息：

```
The property being referenced does not have the static attribute.
```

要访问非静态属性，必须通过该类的实例访问它的值。因为 EXAMPLE_PUBLIC 属性是公有属性，所以可用于该类定义以外的代码。

9. 在“动作”面板中，删除在步骤 5 和步骤 7 中添加的 trace 语句。
10. 在“动作”面板中键入以下代码：

```
var myExample:ConstExample = new ConstExample();
trace(myExample.EXAMPLE_PUBLIC); // 输出: Public access
```

此代码对 myExample 实例进行实例化并访问 EXAMPLE_PUBLIC 属性。

11. 选择“控制”>“测试影片”对文档进行测试。

将在“输出”面板中看到 Public access。

12. 在“动作”面板中，删除在步骤 10 中添加的 trace 语句。
13. 在“动作”面板中键入以下代码。

```
trace(myExample.EXAMPLE_PRIVATE); // 错误
```

EXAMPLE_PRIVATE 属性是一个私有属性，因此仅在该类定义中可用。

14. 选择“控制”>“测试影片”对文档进行测试。

将在“输出”面板中看到 The member is private and cannot be accessed。

有关内置类和创建自定义类的更多信息，请参见第 163 页的第 6 章“类”。

关于关键字

关键字是 ActionScript 中用于执行一项特定操作的单词。例如，var 关键字用于声明变量。下面一行代码中显示了 var 关键字：

```
var myAge:Number = 26;
```

关键字是具有特定含义的保留字：例如，使用 class 关键字定义一个新的 ActionScript 类；使用 var 关键字声明本地变量。保留关键字的其它示例有：if、else、this、function 和 return。

关键字不能用作标识符（例如变量、函数或标签名称），也不应在 FLA 文件中的其它位置将它们用作其它目的（例如实例名称）。您已经大量使用了 var 关键字，特别是在第 275 页的第 10 章“数据和数据类型”中。ActionScript 在语言中针对一些特别用途保留了一些单词。因此，不能将关键字用作标识符（例如变量、函数或标签名称）。可以在第 86 页的“关于保留字”中找到这些关键字的列表。

关于保留字

保留字 是一些单词，因为这些单词是保留给 ActionScript 使用的，所以不能在代码中将它们用作标识符。保留字包括关键字，关键字是 ActionScript 语句和保留给将来使用的一些单词。这意味着不应将它们用于命名变量、实例、自定义类等；这样做会导致您的工作中出现技术问题。

下表列出了 Flash 中可引发脚本错误的保留关键字：

add	and	break	case
catch	class	continue	default
delete	do	dynamic	else
eq	extends	finally	for
function	ge	get	gt
if	ifFrameLoaded	implements	import
in	instanceof	interface	intrinsic
le	lt	ne	new
not	on	onClipEvent	or
private	public	return	set
static	switch	tellTarget	this
throw	try	typeof	var
void	while	with	

下表列出了为 ActionScript 或 ECMAScript (ECMA-262) edition 4 语言规范草案所保留的供将来使用的关键字。在代码中也应该避免使用这些关键字：

abstract	enum	export	short
byte	long	synchronized	char
debugger	protected	double	volatile
float	throws	transient	goto

所有内置类名称、组件类名称和接口名称都是保留字，它们不应在代码中用作标识符：

Accessibility	Accordion	Alert	Array
Binding	Boolean	Button	Camera
CellRenderer	CheckBox	Collection	Color
ComboBox	ComponentMixins	ContextMenu	ContextMenuItem
CustomActions	CustomFormatter	CustomValidator	DataGrid
DataHolder	DataProvider	DataSet	DataType
Date	DateChooser	DateField	Delta
DeltaItem	DeltaPacket	DepthManager	EndPoint
Error	FocusManager	Form	Function
Iterator	Key	Label	List
Loader	LoadVars	LocalConnection	Log
Math	Media	Menu	MenuBar
Microphone	Mouse	MovieClip	MovieClipLoader
NetConnection	NetStream	Number	NumericStepper
Object	PendingCall	PopUpManager	PrintJob
ProgressBar	RadioButton	RDBMSResolver	Screen
ScrollPane	Selection	SharedObject	Slide
SOAPCall	Sound	Stage	String
StyleManager	System	TextArea	TextField
TextFormat	TextInput	TextSnapshot	TransferObject
Tree	TreeDataProvider	TypedValue	UIComponent
UIEventDispatcher	UIObject	Video	WebService

WebServiceConnector	Window	XML	XMLConnector
XUpdateResolver			

另外还有一些单词，虽然它们不是保留字，但是也不应在 ActionScript 代码中用作标识符（例如变量或实例名称）。这些单词是供组成 ActionScript 语言的内置类使用的。因此，在代码中（例如命名变量、类或实例时）不要将属性名称、方法名称、类名称、接口名称、组件类名称以及值用作名称。

要了解这些名称都有哪些，请参考《ActionScript 2.0 语言参考》，并在“帮助”面板中搜索本书（《学习 Flash 中的 ActionScript 2.0》）中的说明和用法部分。

关于语句

语句 是告诉 FLA 文件执行操作的指令，例如执行特定的动作。例如，可以使用条件语句确定某一条件是否为 true 或成立。然后，代码可以根据条件是否为 true 执行指定的动作，例如函数或表达式。

例如，`if` 语句是一个条件语句，它可对一个条件求值以确定代码中应发生的下一个动作。

```
// if 语句
if (condition) {
  // 语句;
}
```

另一个示例是 `return` 语句，该语句返回一个结果，并将其作为执行该语句的函数的值。

设置 ActionScript 格式或编写 ActionScript 时有多种不同的方法可供选择。您所用的构成语法的方式可能与其它编写 ActionScript 的程序员的方式不相同，例如，在代码中分隔语句的方式或放置大括号 ({}) 的位置。虽然可以使用几种不同的方法构成语句，且都不会破坏代码，但是要编写结构良好的 ActionScript，需要遵循一些一般的准则。

一行仅放置一个语句以增强 ActionScript 的可读性。下面的示例中提供了我们建议使用和建议不要使用的语句：

```
theNum++;        // 建议
theOtherNum++;   // 建议
aNum++; anOtherNum++;  // 不建议
```

将变量指定为单独的语句。请看下面的 ActionScript 示例：

```
var myNum:Number = (a = b + c) + d;
```

此 ActionScript 在代码中嵌入了一个赋值语句，使代码难于阅读。如果将变量指定为单独的语句，就可以增强可读性，如下面的示例所示：

```
var a:Number = b + c;
var myNum:Number = a + d;
```

以下各部分介绍如何在 ActionScript 中构成特定语句。有关编写和设置事件格式的信息，请参见第 255 页的第 9 章 “处理事件”。

有关每个语句的更多信息，请参见以下主题：

- 第 89 页的 “关于复合语句”
- 第 89 页的 “关于条件语句”
- 第 98 页的 “使用循环重复动作”

关于复合语句

复合语句包含括在大括号 ({}) 内的多条语句。复合语句中的语句可以是任何类型的 ActionScript 语句。下面显示了典型的复合语句。

这些大括号内的语句从复合语句缩进，如下面的 ActionScript 所示：

```
var a:Number = 10;
var b:Number = 10;
if (a == b) {
  // 此代码是缩进的。
  trace("a == b");
  trace(a);
  trace(b);
}
```

此复合语句包含几个语句，但是在 ActionScript 代码中作为单个语句使用。左大括号放置在了复合语句的结尾。右大括号在行首，与复合语句的开头对齐。

有关使用大括号的更多信息，请参见第 75 页的 “大括号”。

关于条件语句

可以使用条件语句确定某一条件是否为 true 或是否成立，然后根据需要重复某个动作（使用循环），或根据该条件是否为 true 执行指定的动作（例如函数或表达式）。例如，可以确定是否定义了特定的变量或具有一个特定的值，并根据该结果执行一个代码块。另外，还可以根据用户的系统时钟所设置的时间或用户当前位置的天气更改 Flash 文档中的图形。

要根据条件是否成立执行动作或重复一个动作（创建循环语句），可以使用 `if`、`else`、`else if`、`for`、`while`、`do while`、`for..in` 或 `switch` 语句。

有关可用的条件语句以及如何编写它们的更多信息，请参见以下主题：

- 第 90 页的 “关于编写条件语句”
- 第 90 页的 “使用 if 语句”
- 第 91 页的 “使用 if..else 语句”
- 第 92 页的 “使用 if..else if 语句”

- 第 93 页的 “使用 switch 语句”
- 第 95 页的 “使用 try..catch 和 try..catch..finally 语句”
- 第 97 页的 “关于条件运算符和替代语法”

关于编写条件语句

用于检查一个条件是 true 还是 false 的语句以术语 if 开头。如果条件计算为 true，则 ActionScript 执行下一条语句。如果条件计算为 false，ActionScript 将跳到此代码块外的下一条语句。

提示	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 ((passwordTxt.text.length == 0) || (emailTxt.text.length == 0)) {
  gotoAndStop("invalidLogin");
}  else if (passwordTxt.text == userID){
  gotoAndPlay("startProgram");
}
```

在此代码片断中，如果 passwordTxt 或 emailTxt 文本字段的长度为 0（例如，用户没有输入值），Flash 文档将重定向到 invalidLogin 帧标签。如果 passwordTxt 和 emailTxt 文本字段中都包含值，且 passwordTxt 文本字段的内容与 userID 变量相匹配，SWF 文件将重定向到 startProgram 帧标签。

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。有关 switch 语句的更多信息，请参见第 93 页的 “使用 switch 语句”。

参考以下几部分以了解如何在 ActionScript 应用程序中编写不同种类的条件语句。

使用 if 语句

如果需要根据某一特定条件是否为 true 来执行一系列语句，请使用 if 语句。

```
// if 语句
if (condition) {
  // 语句;
}
```

处理 Flash 项目时将多次用到 if 语句。例如，如果您正在构建一个 Flash 站点，该站点要求用户必须在登录之后才能访问 Web 站点的特定部分，可以用一个 if 语句来验证用户是否在用户名和密码字段中输入了一些文本。

如果要使用外部数据库验证用户名和密码，可能要验证用户提交的用户名 / 密码组合是否与数据库中的某条记录相匹配。还要检查用户是否有权访问该站点中的指定部分。

如果您在 Flash 中编写动画的脚本，可能要使用 if 语句来测试舞台上的一个实例是否仍然在舞台的边界内。例如，如果一个球沿 y 轴向下移动，您可能需要检测该球何时碰撞舞台的底部边缘，以便您可以更改球的运动方向，使球呈向上弹起状态。

使用 if 语句：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

```
// 创建一个字符串来保存 AM 和 PM
var amPm:String = "AM";
// 没有向 Date 中传递任何参数，因此会返回当前的日期 / 时间
var current_date:Date = new Date();
// 如果当前的小时值大于 / 等于 12，则将 amPm 字符串设置为 “PM”。
if (current_date.getHours() >= 12) {
  amPm = "PM";
}
trace(amPm);
```

3. 选择 “控制” > “测试影片”对 ActionScript 进行测试。

   在此代码中，创建一个根据一天中的当前时间保存 AM 或 PM 的字符串。如果当前的小时值大于或等于 12，则将 amPM 字符串变量设置为 PM。最后，可以跟踪 amPm 字符串，当小时值大于或等于 12 时，将显示 PM。否则，您将看到 AM。

使用 if..else 语句

if..else 条件语句让您测试一个条件，然后如果该条件成立则执行一个代码块，否则执行一个替代代码块。

例如，以下代码测试 x 的值是否超过 20，超过时生成一条 trace() 语句，不超过时生成另一条 trace() 语句：

```
if (x > 20) {
  trace("x is > 20");
} else {
  trace("x is <= 20");
}
```

如果您不想执行替代代码块，可以仅使用 if 语句，而不用 else 语句。

Flash 中的 if..else 语句类似于 if 语句。例如，如果使用 if 语句来验证用户提供的用户名和密码与存储在数据库中的值匹配，那么可能要根据用户名和密码是否正确重定向用户。如果登录有效，可以使用 if 块将用户重定向到欢迎页。但是，如果登录无效，可以使用 else 块将用户重定向到登录表单，并显示一条错误消息。

在文档中使用 if..else 语句：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”创建一个新的 FLA 文件。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

```
// 创建一个根据一天中的时间保存 AM/PM 的字符串。
var amPm:String;
// 没有向 Date 中传递任何参数，因此返回当前的日期 / 时间。
var current_date:Date = new Date();
// 如果当前的小时值大于 / 等于 12，则将 amPm 字符串设置为 “PM”。
if (current_date.getHours() >= 12) {
  amPm = "PM";
} else {
  amPm = "AM";
}
trace(amPm);
```

3. 选择 “控制” > “测试影片”对 ActionScript 进行测试。

   在此代码中，创建一个根据一天中的当前时间保存 AM 或 PM 的字符串。如果当前的小时值大于或等于 12，则将 amPM 字符串设置为 PM。最后，可以跟踪 amPm 字符串，当小时值大于或等于 12 时，将显示 PM。否则，将在 “输出”面板中看到 AM。

使用 if..else if 语句

可以使用 if..else if 条件语句测试多个条件。if..else if 语句中可以使用以下语法：

```
// else-if 语句
if (condition) {
  // 语句;
} else if (condition) {
  // 语句;
} else {
  // 语句;
}
```

如果要检查一系列条件，就要在 Flash 项目中使用 if..else if 块。例如，如果要根据用户在一天中的访问时间在屏幕上显示不同的图像，可以创建一系列 if 语句来确定时间是清晨、下午、晚上还是夜间。然后显示适当的图像。

以下代码不仅测试 x 的值是否超过 20，还能测试 x 的值是否为负数：

```
if (x > 20) {
  trace("x is > 20");
} else if (x < 0) {
  trace("x is negative");
}
```

在文档中使用 if..else if 语句：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

```
var now_date:Date = new Date();
var currentHour:Number = now_date.getHours();
// if 当前时间早于 11AM...
if (currentHour < 11) {
  trace("Good morning");
  // else..if 当前时间早于 3PM...
} else if (currentHour < 15) {
  trace("Good afternoon");
  // else..if 当前时间早于 8PM...
} else if (currentHour < 20) {
  trace("Good evening");
  // else 当前时间在 8PM 和 11:59PM 之间
} else {
  trace("Good night");
}
```

3. 选择 “控制” > “测试影片”对 ActionScript 进行测试。

   在此代码中，创建一个名为 `currentHour` 的字符串，用于保存当前的小时值 （例如，如果当前时间是 6:19 pm，则 `currentHour` 保存数字 `18`）。可以使用 Date 类的 `getHours()` 方法获取当前的小时值。然后，可以使用 `if..else if` 语句根据返回的数字将信息显示到 “输出”面板中。有关更多信息，请参见以上代码片段中的注释。

使用 switch 语句

`switch` 语句创建 ActionScript 语句的分支结构。与 `if` 语句类似，`switch` 语句测试一个条件，并在条件返回 `true` 值时执行一些语句。

在使用 `switch` 语句时，`break` 语句指示 Flash 跳过此 case 块中其余的语句，并跳到位于包含它的 `switch` 语句后面的第一个语句。如果 case 块不包含 break 语句，就会出现一种被称为 “落空”的情况。在这种情况下，接下来的 case 语句也会执行，直到遇到 break 语句或 `switch` 语句结束才停止。下面的示例中演示了这种行为，其中第一个 case 语句不包含 break 语句，因此前两个 case （A 和 B）的代码块都会执行。

所有 `switch` 语句都应包含一个 `default` case。`default` case 应该始终为 `switch` 语句中的最后一个 case，而且应包含一个 `break` 语句来避免添加其它 case 时出现落空错误。例如，如果下面的示例中的条件的计算结果均为 A，则 case A 和 B 的语句都会执行，因为 case A 缺少 `break` 语句。当一个 case 落空时，它没有 break 语句，但在 `break` 语句的位置会有一个注释，您在下面的示例中 case A 的后面会看到这样一个注释。在编写 `switch` 语句时，请使用以下格式：

```
switch (condition) {
case A :
  // 语句
  // 落空
case B :
  // 语句
  break;
case Z :
  // 语句
  break;
default :
  // 语句
  break;
}
```

在文档中使用 switch 语句：

1. 选择“文件”>“新建”，然后选择“Flash 文档”。
2. 在时间轴中选择第 1 帧，然后在“动作”面板中，键入下面的 ActionScript：

```
var listenerObj:Object = new Object();
listenerObj.onKeyDown = function() {
  // 使用 String.fromCharCode() 方法返回一个字符串。
  switch (String.fromCharCode(Key.getAscii())) {
  case "A" :
    trace("you pressed A");
    break;
  case "a" :
    trace("you pressed a");
    break;
  case "E" :
  case "e" :
    /* E 没有 break 语句，因此如果按下 e 或 E 时会执行此块。 */
    trace("you pressed E or e");
    break;
  case "I" :
  case "i" :
    trace("you pressed I or i");
    break;
  default :
    /* 如果所按的键未被以上任何 case 所捕获，则执行此处的 default case。 */
    trace("you pressed some other key");
  }
};
Key.addListener(listenerObj);
```

3. 选择“控制”>“测试影片”对 ActionScript 进行测试。

使用键盘键入字母，包括 a、e 或 i 键。当按下这三个键时，您将看到以上 ActionScript 中的 trace 语句。此行代码创建了一个新的对象，用作 Key 类的监听器。当用户按下一个键时，可以使用此对象通知 onKeyDown() 事件。Key.getAscii() 返回用户按下或释放的最后一个键的 ASCII 码，因此需要使用 String.fromCharCode() 方法在参数中返回包含该 ASCII 值代表的字符的字符串。因为“E”没有 break 语句，所以如果用户按下 e 或 E 键，该代码块就会执行。如果用户所按的键未被前三个 case 中任一个所捕获，则会执行 default case。

使用 try..catch 和 try..catch..finally 语句

使用 try..catch..finally 代码块使您能够在 Flash 应用程序中加入错误处理。try..catch..finally 关键字允许您括起一个可能会发生错误的代码块，并对该错误作出响应。如果 try 代码块内的任何代码抛出了一个错误（使用 throw 语句），控制将传递给 catch 代码块（如果有）。然后，控制将传递给 finally 代码块（如果有）。无论是否有错误被抛出，可选的 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，try 代码块正常完成），则仍会执行 finally 代码块内的代码。

注意	即使 try 代码块使用 return 语句退出，finally 代码块仍会执行。

应使用以下格式编写 try..catch 和 try..catch..finally 语句：

```
// try-catch
try {
  // 语句
} catch (myError) {
  // 语句
}

// try-catch-finally
try {
  // 语句
} catch (myError) {
  // 语句
} finally {
  // 语句
}
```

无论何时代码抛出一个错误，您都可以编写自定义处理函数来适当地处理错误并执行相应的动作。您可能需要尝试从 Web 服务或文本文件加载外部数据，或向最终用户显示一条错误消息。还可以使用 catch 代码块尝试连接到可以提醒管理员出现了特定错误的 Web 服务，以便他 / 她可以确保应用程序正常工作。

在对一些数字执行除法运算之前使用 try..catch..finally 代码块进行数据验证：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

```
var n1:Number = 7;
var n2:Number = 0;
try {
  if (n2 == 0) {
    throw new Error("Unable to divide by zero");
  }
  trace(n1/n2);
} catch (err:Error) {
  trace("ERROR! " + err.toString());
} finally {
  delete n1;
  delete n2;
}
```

3. 选择 “控制” > “测试影片”对该文档进行测试。
4. “输出”面板将显示 `Unable to divide by zero`。
5. 返回到创作环境并更改下面一行代码：

   `var n2:Number = 0;`

   更改为：

   `var n2:Number = 2;`
6. 选择 “控制” > “Enter”来再次对该文档进行测试。

   如果 `n2` 的值等于 0，则将抛出一个错误，`catch` 代码块将捕获该错误，该错误会在“输出”面板中显示一条消息。如果 `y` 的值不等于 0，则 “输出”面板中将显示 `n1` 除以 `n2` 的结果。无论是否出现错误，`finally` 代码块都会执行，并会从 Flash 文档中删除变量 `n1` 和 `n2` 的值。

在错误发生时，除了可以抛出 Error 类的新实例外，您还可以扩展 Error 类来创建您自己的自定义错误，如下面的示例所示。

创建自定义错误：

1. 选择 “文件” > “新建”，然后创建新的 ActionScript 文件。
2. 选择 “文件” > “另存为”，将该文件命名为 **DivideByZeroException.as**。
3. 在 “脚本”窗格中键入下面的 ActionScript：

```
// 在 DivideByZeroException.as 中:
class DivideByZeroException extends Error {
   var message:String = "Divide By Zero error";
}
```

4. 保存该 ActionScript 文件。

5. 在该 ActionScript 文件所在的相同目录下新建一个名为 **exception_test.fla** 的 Flash 文档，然后保存该文件。

6. 在主时间轴的第 1 帧上，在 “动作”面板中键入下面的 ActionScript：

```
var n1:Number = 7;
var n2:Number = 0;
try {
  if (n2 == 0) {
    throw new DivideByZeroException();
  } else if (n2 < 0) {
    throw new Error("n2 cannot be less than zero");
  } else {
    trace(n1/n2);
  }
} catch (err:DivideByZeroException) {
  trace(err.toString());
} catch (err:Error) {
  trace("An unknown error occurred; " + err.toString());
}
```

7. 保存该 Flash 文档，然后选择 “控制” > “测试影片”来在测试环境中测试该文件。

   因为 `n2` 的值等于 0，Flash 抛出您的自定义 DivideByZeroException 错误类，并会在“输出”面板中显示 `Divide By Zero error`。如果在第二行中将 `n2` 的值从 `0` 更改为 `-1`，然后再次测试该 Flash 文档，您将看到 “输出”面板中显示 `An unknown error occurred; n2 cannot be less than zero`。将 `n2` 的值设置为任何大于 `0` 的数字便可使 “输出”面板中显示除法的结果。有关创建自定义类的更多信息，请参见第 163 页的第 6 章 “类”。

关于条件运算符和替代语法

如果您喜欢短代码形式，可以使用条件 (`?:`) 运算符，又称条件表达式。条件运算符允许您将简单的 `if..else` 语句转换为单行代码。该运算符有助于在实现同一效果的同时减少编写的代码量，但是它往往会使您的 ActionScript 更难以阅读。

下面的条件是用完整形式编写的，检查变量 `numTwo` 是否大于零，并返回 `numOne/numTwo` 的结果或字符串 `carrot`：

```
var numOne:Number = 8;
var numTwo:Number = 5;
if (numTwo > 0) {
  trace(numOne / numTwo); // 1.6
} else {
  trace("carrot");
}
```

使用条件表达式时，可以用以下格式编写同样的代码：

```
var numOne:Number = 8;
var numTwo:Number = 0;
trace((numTwo > 0) ? numOne/numTwo : "carrot");
```

如您所看到的，缩短语句会降低可读性，因此这样做并不好。如果必须使用条件运算符，请在小括号内放入前导条件（在问号 [?] 前）。这有助于增强 ActionScript 的可读性。下面的代码是更具可读性的 ActionScript 的示例：

```
var numOne:Number;
(numOne >= 5) ? numOne : -numOne;
```

您可以编写返回布尔值的条件语句，如下面的示例所示：

```
if (cartArr.length > 0) {
  return true;
} else {
  return false;
}
```

但是，与前面的代码相比，下面的示例中的 ActionScript 更好：

```
return (cartArr.length > 0);
```

第二段代码更短，需要计算的表达式更少。它更容易阅读和理解。

编写复杂条件时，使用小括号 [()] 对条件进行组合是恰当的形式。如果不使用小括号，您（或使用 ActionScript 的其他人）就可能会遇到运算符优先错误。有关运算符优先级的更多信息，请参见第 121 页的 “关于运算符的优先级和结合律”。

例如，下面的代码未在条件前后使用小括号：

```
if (fruit == "apple" && veggie == "leek") {}
```

下面的代码通过在条件前后添加小括号而具有了恰当的形式：

```
if ((fruit == "apple") && (veggie == "leek")) {}
```

使用循环重复动作

ActionScript 可以按指定的次数重复一个动作，或者在特定的条件成立时重复动作。循环使您能够在特定条件为 `true` 时重复执行一系列语句。在 ActionScript 中有四种类型的循环：`for` 循环、`for..in` 循环、`while` 循环和 `do..while` 循环。不同类型的循环的行为方式互不相同，而且分别适合于不同的用途。

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相应的语句；每执行一次循环，都让该语句对该变量递增或递减。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。

循环	说明
`for` 循环	使用内置计数器重复动作。
`for..in` 循环	迭代影片剪辑或对象的子级。

循环	说明
while 循环	在某个条件成立时重复动作。
do..while 循环	类似于 while 循环，差别仅在于它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。

最常用的循环类型是 for 循环，它使一个代码块按预定义的次数循环。例如，如果您有一个项目数组，并且要对该数组中的每个项目执行一系列语句，则可以使用 for 循环，从 0 循环到该数组中的项目数。另一种有用的循环类型是 for..in 循环；如果要对一个对象中的每个名称 / 值对执行循环，然后执行某种类型的动作，就非常需要使用这种循环。这种循环在调试 Flash 项目和要显示从外部源（例如 Web 服务或外部 text/XML 文件）中加载的值时很有用。如果要循环一系列语句，但是不一定要知道需要循环的次数，则可以使用最后两种循环类型（while 和 do..while）。在这种情况下可以使用 while 循环，该循环在特定的条件为 true 时循环。

ActionScript 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。本部分包含有关这些循环的一般信息。有关每种循环的更多信息，请参见以下的各个过程。

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，会再次计算该表达式。在下面的示例中，循环将执行四次：

  ```
  var i:Number = 4;
  while (i>0) {
    myClip.duplicateMovieClip("newMC" + i, i, {_x:i*20, _y:i*20});
    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。

  如下面的示例所示：

  ```
  var i:Number = 4;
  do {
    myClip.duplicateMovieClip("newMC" + i, i, {_x:i*20, _y:i*20});
    i--;
  } while (i>0);
  ```

  有关 while 语句的更多信息，请参见第 104 页的 “使用 while 循环”。

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相应语句：每执行一次循环，都让该语句对该变量递增或递减。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:Number = 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:Number = 4; i > 0; i--) {
  myClip.duplicateMovieClip("newMC" + i, i, {_x:i*20, _y:i*20});
}
```

  有关 for 语句的更多信息，请参见第 102 页的 “使用 for 循环”。

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：

```
var myObject:Object = {name:'Joe', age:25, city:'San Francisco'};
var propertyName:String;
for (propertyName in myObject) {
  trace("myObject has the property: " + propertyName + ", with the value:
  " + myObject[propertyName]);
}
```

  此示例将在 “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
```

  您可能想让脚本迭代特定类型的子级 -- 例如只迭代影片剪辑子级。您可以将 for..in 与 typeof 运算符结合使用来实现此目的。在下面的示例中，舞台上的一个影片剪辑实例中包含一个子影片剪辑实例 （称为 instance2）。将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
for (var myName in this) {
  if (typeof (this[myName]) == "movieclip") {
    trace("I have a movie clip child named " + myName);
  }
}
```

有关 for..in 语句的更多信息，请参见第 103 页的 “使用 for..in 循环”。

警告	Flash 中的迭代在 Flash Player 执行的速度非常快，但循环对处理器非常依赖。循环中的迭代次数越多，每个块内执行的语句越多，使用的处理器资源也就越多。编写不合理的循环可以导致性能问题和稳定性问题。

有关各个语句的更多信息，请参见本章内下文中的各部分，例如第 104 页的“使用 while 循环”，以及它们在《ActionScript 2.0 语言参考》中的相应条目。

关于创建和结束循环

下面的示例显示了月份名称的一个简单数组。for 循环从 0 循环到数组中的项目数，并在“输出”面板中显示每个项目。

```
var monthArr:Array = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
var i:Number;
for (i = 0; i < monthArr.length; i++) {
  trace(monthArr[i]);
}
```

使用数组时，无论数组是简单的还是复杂的，都需要知道一种被称为无限循环 的情形。无限循环，顾名思义，是一种没有结束条件的循环。这会导致严重的问题 - 使您的 Flash 应用程序崩溃，导致 Flash 文档在 Web 浏览器中停止响应或导致 Flash 文档出现非常不一致的行为。下面的代码是无限循环的一个示例：

```
// 错误的代码 - 造成一个无限循环
// 这样的错误代码会导致程序运行错误
var i:Number;
for (i = 0; i < 10; i--) {
  trace(i);
}
```

i 的值初始化为 0，当 i 大于或等于 10 时满足结束条件，而且每次迭代之后 i 的值将递减。也许您马上就会看出其中明显的错误：如果 i 的值在每次循环迭代之后递减，那么结束条件永远都无法满足。结果将因运行该循环的计算机而异，代码失败的速度将取决于 CPU 的速度和其它因素。例如，在一台给定的计算机上，该循环执行大约 142,620 次，然后显示一条错误消息。

对话框中显示以下错误消息：

```
A script in this movie is causing Flash Player to run slowly. If it continues
  to run, your computer may become unresponsive. Do you want to abort the
  script?
```

在使用循环 （特别是 while 和 do..while 循环）时，始终要确保循环可以正常退出并且不会导致无限循环。

有关控制循环的更多信息，请参见第 93 页的 “使用 switch 语句”。

使用 for 循环

for 循环允许您对于特定作用域的值迭代变量。for 循环在确切知道一系列 ActionScript 语句要重复执行的次数时非常有用。如果要在舞台上将一个影片剪辑复制特定的份数，或者对一个数组执行循环并对数组中的每个项目执行一项任务时，这种循环可能非常有用的。for 循环使用内置计数器重复动作。在 for 语句中，计数器和递增计数器的语句都是该 for 语句的一部分。使用下面的基本格式编写 for 语句：

```
for (init; condition; update) {
  // 语句;
}
```

必须为 for 语句提供三个表达式：一个设置了初始值的变量，一个用于确定循环何时结束的条件语句，和一个在每次循环中更改变量的值的表达式。例如，下面的代码循环 5 次。变量 i 的值从 0 开始以 4 结束，输出结果将是从 0 到 4 的 5 个数字，每个数字各占一行。

```
var i:Number;
for (i = 0; i < 5; i++) {
  trace(i);
}
```

在下一个示例中，第一个表达式 (i = 0) 是在第一次迭代之前计算的初始表达式。第二个表达式 (i < 5) 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。

创建 for 循环：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 在舞台上创建一个影片剪辑。
3. 在 “库”面板中右键单击该影片剪辑元件，并从上下文菜单中选择 “链接”。
4. 选择 “为 ActionScript 导出”，并在 “类”文本输入字段中键入 **libraryLinkageClassName**。单击 “确定”。
5. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

   ```
   var i:Number;
   for (i = 0; i < 5; i++) {
     this.attachMovie("libraryLinkageClassName", "clip" + i + "_mc", i,
     {_x:(i * 100)});
   }
   ```

6. 选择 “控制” > “测试影片”在 Flash Player 中测试此代码。

   请注意看这 5 个影片剪辑是如何跨舞台顶部直接复制的。此 ActionScript 在库中直接复制影片剪辑元件，并在舞台上将这些影片剪辑重新定位到 x 坐标为 0、100、200、300 和 400 像素的位置。该循环执行 5 次，变量 i 依次被赋予了从 0 到 4 的值。在该循环的最后一次迭代中，i 的值增加到 4，第二个表达式 (i < 5) 不再为 true，这将导致循环退出。

请记住在 for 语句中每个表达式后都加一个空格。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 for 语句。

使用 for..in 循环

使用 for..in 语句遍历 （或循环 访问）影片剪辑的子级、对象的属性或数组的元素。子级（见前述）包括其它影片剪辑、函数、对象和变量。 for..in 循环的常见用法包括循环执行时间轴上的实例或循环执行对象中的键 / 值对。循环执行对象可能是调试应用程序的一种有效的方法，因为它允许您查看 Web 服务或外部文档（例如文本或 XML 文件）返回的数据。

例如，可以使用 for...in 循环来循环访问一个通用对象 （对象的属性不按任何特定的顺序保存，因此属性将以不可预知的顺序出现）的属性：

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj) {
  trace(i + ": " + myObj[i]);
}
```

此代码将在 “输出”面板中输出以下信息：

```
x: 20
y: 30
```

还可以循环访问数组中的元素：

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray) {
  trace(myArray[i]);
}
```

此代码将在 “输出”面板中输出以下信息：

```
three
two
one
```

有关对象和属性的更多信息，请参见第 282 页的 “Object 数据类型”。

注意 如果对象是自定义类的一个实例，则无法循环访问该对象的属性，除非此类是动态类。即便对于动态类的实例，也只能循环访问动态添加的属性。

注意 如果只执行一条语句，则不必使用大括号 ({}) 括起 for..in 语句要执行的语句块。

下面的示例使用 for..in 迭代某对象的属性：

创建 for 循环：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

```
var myObj:Object = {name:"Tara", age:27, city:"San Francisco"};
var i:String;
for (i in myObj) {
  trace("myObj." + i + " = " + myObj[i]);
}
```

3. 选择 “控制” > “测试影片” 在 Flash Player 中测试此代码。

测试 SWF 文件时，您会看到 “输出” 面板中显示以下文本：

```
myObj.name = Tara
myObj.age = 27
myObj.city = San Francisco
```

如果在一个类文件 （外部 ActionScript 文件）中编写一个 `for..in` 循环，则实例成员在循环中不可用，但静态成员可用。然而，如果在一个 FLA 文件中为该类的实例写一个 `for..in` 循环，则实例成员在循环中可用，而静态成员不可用。有关编写类文件的更多信息，请参见第 163 页的第 6 章 “类”。有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `for..in` 语句。

使用 while 循环

使用 `while` 语句在条件成立时重复某动作，类似于 `if` 语句，只要条件为 `true` 就重复动作。

`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。如果条件计算结果为 `true`，在循环返回以再次计算条件前执行一条语句或一系列语句。条件计算结果为 `false` 后，则跳过语句或一系列语句并结束循环。在不确定要将一个代码块循环多少次时，使用 `while` 循环可能会非常有用。

例如，下面的代码将数字显示到 “输出” 面板中：

```
var i:Number = 0;
while (i < 5) {
  trace(i);
  i++;
}
```

您会看到以下数字显示到 “输出” 面板中：

```
0
1
2
3
4
```

使用 `while` 循环而非 `for` 循环的一个缺点是，在 `while` 循环中更有可能编写出无限循环。如果遗漏递增计数器变量的表达式，则 `for` 循环示例代码将无法编译；而 `while` 循环示例代码将能够编译。若没有递增 `i` 的表达式，循环将成为无限循环。

要在 FLA 文件中创建和使用 `while` 循环，请按照下例操作。

创建 while 循环：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 打开 “组件”面板，将一个 DataSet 组件拖动到舞台上。
3. 打开属性检查器 （“窗口” > “属性” > “属性”），然后键入实例名称 **users_ds**。
4. 在时间轴中选择第 1 帧，然后在 “动作”面板中，键入下面的 ActionScript：

```
var users_ds:mx.data.components.DataSet;
//
users_ds.addItem({name:"Irving", age:34});
users_ds.addItem({name:"Christopher", age:48});
users_ds.addItem({name:"Walter", age:23});
//
users_ds.first();
while (users_ds.hasNext()) {
  trace("name:" + users_ds.currentItem["name"] + ", age:" +
   users_ds.currentItem["age"]);
  users_ds.next();
}
```

5. 选择 “控制” > “测试影片”对该文档进行测试。

   “输出”面板中会显示以下信息：

```
name:Irving, age:34
name:Christopher, age:48
name:Walter, age:23
```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `while 语句`。

关于 do..while 循环

可以使用 `do..while` 语句创建与 `while` 循环同类的循环。但是，`do..while` 循环中是在代码块结束时计算表达式的值 （在代码块执行之后检查），因此该循环总是至少执行一次。只有条件计算结果为 `true` 时语句才会执行。

下面的代码显示了 `do...while` 循环的一个简单示例，即使条件不满足也会生成输出结果。

```
var i:Number = 5;
do {
  trace(i);
  i++;
} while (i < 5);
// 输出: 5
```

使用循环时，要避免编写出无限循环。如果 `do..while` 循环中的条件连续计算为 `true`，就创建了一个无限循环，将会显示警告或导致 Flash Player 崩溃。如果您知道要循环的次数，可以改用 `for` 循环。有关 `do..while 语句`的更多信息及示例，请参见《ActionScript 2.0 语言参考》。

在 ActionScript 中使用嵌套循环

下面的示例演示了如何创建一个对象数组并显示了嵌套结构中的每个值。该示例说明了如何使用 for 循环遍历数组中的每个项目，以及如何使用 for..in 循环来循环访问嵌套对象中的各键 / 值对。

将一个循环嵌套到另一个循环中：

1. 创建一个新的 Flash 文档。
2. 选择 “文件” > “另存为”，将该文档命名为 **loops.fla**。
3. 将下面的代码添加到时间轴中的第 1 帧：

```
var myArr:Array = new Array();
myArr[0] = {name:"One", value:1};
myArr[1] = {name:"Two", value:2};
//
var i:Number;
var item:String;
for (i = 0; i < myArr.length; i++) {
  trace(i);
  for (item in myArr[i]) {
    trace(item + ": " + myArr[i][item]);
  }
  trace("");
}
```

4. 选择 “控制” > “测试影片” 对代码进行测试。

   下面的内容将在 “输出” 面板中显示：

```
0
name: One
value: 1

1
name: Two
value: 2
```

   您知道数组中有多少个项目，因此可以使用一个简单的 for 循环来循环每一个项目。因为数组中的每个对象可能都有不同的名称 / 值对，所以可以使用 for..in 循环来迭代每个值，并在 “输出” 面板中显示结果。

关于数组

数组 是一个对象，其属性由表示该属性在结构中位置的数字来标识。实质上，数组是一系列项目。需要记住的重要一点是，数组中的各元素不必为相同的数据类型。可以在每个数组索引上混合使用数字、日期、字符串、对象，甚至添加一个嵌套数组。

下面的示例是一个简单的月份名称数组。

```
var myArr:Array = new Array();
myArr[0] = "January";
myArr[1] = "February";
myArr[2] = "March";
myArr[3] = "April";
```

上面的月份名称数组还可以按以下形式重写：

```
var myArr:Array = new Array("January", "February", "March", "April");
```

或者，可以使用简化语法，如下所示：

```
var myArr:Array = ["January", "February", "March", "April"];
```

数组类似于一种数据结构。一个数组好比一座办公大楼，每一层都包含不同的数据片段（例如会计部 在 3 层，工程部 在 5 层）。同样，可以在单个数组中存储不同类型的数据 （包括其它数组）。大楼的每一层可以包含多种内容 （经理办公室 和会计部 可能都在 3 层）。

数组包含元素，这些元素对应于大楼的各层。每个元素都有一个数字位置 （即索引，用于指代每个元素在数组中的位置。这类似于大楼中的每一层都有一个层号。每个元素都可以保存一个数据片段 （可以是数字、字符串、布尔值，甚至数组或对象）或为空。

您还可以控制和修改数组本身。例如，可能要将工程部搬到大楼的地下室。数组允许您来回移动值，并且允许更改数组的大小（这类似于翻新大楼，加盖更多层或去掉一些层）。同样，可以添加或删除元素以及将值移动到不同的元素中。

因此，大楼 （数组）包含多个层 （元素），都是已编号的层 （索引），并且每层都包含一个或多个部门 （值）。

有关修改数组的更多信息，请参见第 109 页的 “关于修改数组”。有关使用数组的信息以及索引的信息，请参见第 108 页的 “使用数组”。有关添加和删除元素的信息，请参见第 111 页的 “关于添加和删除元素”。有关数组访问运算符的信息，请参见第 125 页的 “使用点运算符和数组访问运算符”。

可以在您的硬盘上的 Samples 文件夹中找到范例源文件 array.fla。该范例演示了如何使用 ActionScript 进行数组操作。该范例中的代码将创建一个数组，并对两个 List 组件的项进行排序、添加和删除操作。在以下目录中可找到该范例文件：

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Arrays。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Arrays。

使用数组

在工作中可以用几种不同的方法使用数组。可以用它们来存储一系列对象，例如一组返回的项目。如果从远程 Web 服务器上加载数据，则甚至可以用嵌套对象数组的形式接收数据。通常，数组包含具有相似格式的数据。例如，如果您在 Flash 中构建一个音频应用程序，可能要将用户的播放列表以歌曲信息的数组的形式存储在对象中。每个对象都包含歌曲名称、歌手名字、歌曲持续时间以及声音文件 （例如 MP3）的位置，或可能需要与特定的文件关联的任何其它信息。

项目在数组中的位置称为索引。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]，依此类推。

有多种不同类型的数组，在下面的各部分中您将会看到这些数组。最常见的数组使用数字索引在索引数组 中查找特定的项目。第二种数组称为关联数组，它使用文本索引 （而不是数字索引）查找信息。有关常见数组的更多信息，请参见第 107 页的 “关于数组”。有关关联数组的更多信息，请参见第 115 页的 “创建关联数组”。有关多维数组的更多信息，请参见第 112 页的 “创建多维数组”。有关数组访问运算符的信息，请参见第 125 页的 “使用点运算符和数组访问运算符”。

内置 Array 类使您可以访问并操作数组。要创建 Array 对象，请使用构造函数 `new Array()` 或数组访问运算符 (`[]`)。要访问数组中的元素，还可以使用数组访问运算符 (`[]`)。下一个示例使用了一个索引数组。

在代码中使用数组：

1. 创建一个新的 Flash 文档，并将其另存为 **basicArrays.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
// 定义一个新数组
var myArr:Array = new Array();
// 在两个索引上定义值
myArr[1] = "value1";
myArr[0] = "value0";
// 迭代数组中的项目
var i:String;
for (i in myArr) {
  // 跟踪键 / 值对
  trace("key: " + i + ", value: " + myArr[i]);
}
```

在 ActionScript 的第一行中，定义一个新数组来保存值。然后，在数组的两个索引上定义数据 （`value0` 和 `value1`）。您可以使用 `for..in` 循环迭代数组中的每个项目并使用 trace 语句在 “输出” 面板中显示键 / 值对。

3. 选择 “控制” > “测试影片” 对代码进行测试。

下面的内容在 “输出” 面板中显示：

```
key: 0, value: value0
key: 1, value: value1
```

有关 for..in 循环的更多信息，请参见第 103 页的 “使用 for..in 循环”。

有关如何创建不同类型的数组的更多信息，请参见以下各部分：

- 第 112 页的 “创建索引数组”
- 第 112 页的 “创建多维数组”
- 第 115 页的 “创建关联数组”

可以在您的硬盘上的 Samples 文件夹中找到范例源文件 array.fla。该范例演示了如何使用 ActionScript 进行数组操作。该范例中的代码将创建一个数组，并对两个 List 组件的项进行排序、添加和删除操作。在以下目录中可找到该范例文件：

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Arrays。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Arrays。

关于修改数组

还可以使用 ActionScript 控制和修改数组。可以在数组作用域内移动值，或更改数组的大小。例如，如果要交换数组中两个索引上的数据，可以使用以下代码：

```
var buildingArr:Array = new Array();
buildingArr[2] = "Accounting";
buildingArr[4] = "Engineering";
trace(buildingArr); // undefined,undefined,Accounting,undefined,Engineering

var temp_item:String = buildingArr[2];
buildingArr[2] = buildingArr[4];
buildingArr[4] = temp_item;
trace(buildingArr); // undefined,undefined,Engineering,undefined,Accounting
```

您可能会想：在上面的示例中为什么要创建一个临时变量呢？如果将数组索引 4 的内容复制到数组索引 2 中（反之亦然），数组索引 2 的原始内容将会丢失。当您将值从一个数组索引复制到临时变量中时，可以将该值保存起来，以后在代码中可以安全地将它复制回去。例如，如果改用以下代码，您会发现数组索引 2 的值 (Accounting) 已经丢失。现在您有两个工程部门但是没有会计部。

```
// 错误的方法 （没有临时变量）
buildingArr[2] = buildingArr[4];
buildingArr[4] = buildingArr[2];
trace(buildingArr); // undefined,undefined,Engineering,undefined,Engineering
```

可以在您的硬盘上的 Samples 文件夹中找到范例源文件 array.fla。该范例演示了如何使用 ActionScript 进行数组操作。该范例中的代码将创建一个数组，并对两个 List 组件的项进行排序、添加和删除操作。在以下目录中可找到该范例文件：

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Arrays。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Arrays。

关于引用和查找长度

使用数组时，通常需要知道数组中存在多少个项目。在编写用于循环访问数组中的每个元素并执行一系列语句的 for 循环时，这一点非常有用。在下面的代码片段中您会看到一个示例：

```
var monthArr:Array = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
trace(monthArr); // Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
trace(monthArr.length); // 12
var i:Number;
for (i = 0; i < monthArr.length; i++) {
  monthArr[i] = monthArr[i].toUpperCase();
}
trace(monthArr); // JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
```

在上面的示例中，创建了一个数组并用月份名称进行了填充。显示数组的内容以及长度。for 循环迭代数组中的每个项目并将其值转换为大写字母，并且再次显示数组内容。

在下面的 ActionScript 中，如果您在数组中的数组索引 5 上创建了一个元素，数组的长度将返回 6 （因为数组是从 0 开始的），而并非您所期望的、数组中的实际项目数：

```
var myArr:Array = new Array();
myArr[5] = "five";
trace(myArr.length); // 6
trace(myArr); // undefined,undefined,undefined,undefined,undefined,five
```

有关 for 循环的更多信息，请参见第 102 页的 “使用 for 循环”。有关数组访问运算符的信息，请参见第 125 页的 “使用点运算符和数组访问运算符”。

可以在您的硬盘上的 Samples 文件夹中找到范例源文件 array.fla。该范例演示了如何使用 ActionScript 进行数组操作。该范例中的代码将创建一个数组，并对两个 List 组件的项进行排序、添加和删除操作。在以下目录中可找到该范例文件：

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Arrays。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Arrays。

关于添加和删除元素

数组包含多个元素，每个元素都有一个数字位置（索引），用于指代每个元素在数组中的位置。每个元素既可以保存一个数据也可以为空。元素可以保存以下数据：数字、字符串、布尔值，甚至数组或对象。

在数组中创建元素时，应尽可能按顺序创建索引。这样做在调试应用程序时会对您有所帮助。在第 110 页的 “关于引用和查找长度”中您已经看到，如果在数组中的索引 5 上指定单个值，数组长度将返回 6。这将导致数组中插入 5 个未定义的值。

下面的示例演示了如何创建一个新数组，删除特定索引上的项目，以及如何在数组中的索引上添加和替换数据：

```
var monthArr:Array = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
delete monthArr[5];
trace(monthArr); // Jan,Feb,Mar,Apr,May,undefined,Jul,Aug,Sep,Oct,Nov,Dec
trace(monthArr.length); // 12
monthArr[5] = "JUN";
trace(monthArr); // Jan,Feb,Mar,Apr,May,JUN,Jul,Aug,Sep,Oct,Nov,Dec
```

虽然删除了数组索引 5 上的项，但数组的长度仍然是 12，数组索引 5 上的项变为空白字符串，而不是完全消失。

可以在您的硬盘上的 Samples 文件夹中找到范例源文件 array.fla。该范例演示了如何使用 ActionScript 进行数组操作。该范例中的代码将创建一个数组，并对两个 List 组件的项进行排序、添加和删除操作。在以下目录中可找到该范例文件：

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Arrays。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Arrays。

创建索引数组

索引数组存储一系列值，可以为一个或多个。可以通过项目在数组中的位置查找它们，在前面的部分中您可能已经这样做了。第一个索引始终是数字 0，且添加到数组中的每个后续的元素的索引以 1 为增量递增。可以通过调用 Array 类构造函数，或使用数组文本初始化数组创建索引数组。在下一个示例中使用 Array 构造函数和数组文本创建数组。

创建索引数组：

1. 创建一个新的 Flash 文档，并将其另存为 **indexArray.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var myArray:Array = new Array();
myArray.push("one");
myArray.push("two");
myArray.push("three");
trace(myArray); // one,two,three
```

   在 ActionScript 的第一行中，定义一个新数组来保存值。

3. 选择 “控制” > “测试影片” 对代码进行测试。

   下面的内容在 “输出” 面板中显示：

```
one,two,three
```

4. 返回到创作工具，删除 “动作” 面板中的代码。
5. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var myArray:Array = ["one", "two", "three"];
trace(myArray); // one,two,three
```

   在此代码中使用数组文本为代码定义一个新数组。此代码与您在步骤 2 中编写的 ActionScript 等效。测试此代码时，您将看到 “输出” 面板中显示相同的输出结果。

创建多维数组

在 ActionScript 中，可以将数组实现为嵌套数组，其实质上是数组的数组。嵌套数组，又称多维数组，可以被看作是矩阵或网格。因此，在编程时，可能要用多维数组来为以下类型的结构建模。例如，棋盘是一个由八行八列组成的网格，可以将该棋盘建模为包含八个元素的数组，而其中每个元素也是包含八个元素的数组。

例如，考虑一个任务列表，它存储为有索引的字符串数组：

```
var tasks:Array = ["wash dishes", "take out trash"];
```

如果要将一周中每天的任务存储为一个单独的列表，可以创建一个多维数组，一周中的每天使用一个元素。每个元素包含一个索引数组，而此索引数组进而存储一个任务列表。

小心	使用数组访问运算符时，ActionScript 编译器无法检查访问的元素是否为对象的有效属性。

创建基本多维数组并从该数组检索元素：

1. 创建一个新的 Flash 文档，并将其另存为 **multiArray1.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var twoDArray:Array = new Array(new Array("one","two"), new
  Array("three", "four"));
trace(twoDArray);
```

   此数组 `twoDArray` 包含 2 个数组元素。这些元素本身也是包含两个元素的数组。在此例中，`twoDArray` 是包含 2 个嵌套数组的主数组。

3. 选择 “控制” > “测试影片”对代码进行测试。您将在 “输出”面板中看到以下内容：

```
one,two,three,four
```

4. 返回到创作工具，打开 “动作”面板。注释掉 `trace` 语句，如下所示：

```
// trace(twoDArray);
```

5. 将以下 ActionScript 添加到时间轴中第 1 帧上的代码的末尾：

```
trace(twoDArray[0][0]); // one
trace(twoDArray[1][1]); // four
```

   要检索多维数组的元素，可以在顶层数组的名称之后使用多个数组访问运算符 (`[]`)。第一个 `[]` 指向顶层数组的索引。随后的数组访问运算符指向嵌套数组的元素。

6. 选择 “控制” > “测试影片”对代码进行测试。您将在 “输出”面板中看到以下内容：

```
one
four
```

可以使用嵌套的 `for` 循环来创建多维数组。下一个示例介绍了这一方法。

使用 for 循环创建多维数组：

1. 创建一个新的 Flash 文档，并将其另存为 **multiArray2.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var gridSize:Number = 3;
var mainArr:Array = new Array(gridSize);
var i:Number;
var j:Number;
for (i = 0; i < gridSize; i++) {
  mainArr[i] = new Array(gridSize);
  for (j = 0; j < gridSize; j++) {
    mainArr[i][j] = "[" + i + "][" + j + "]";
  }
}
trace(mainArr);
```

   此 ActionScript 创建了一个 3 x 3 数组，并将每个数组节点的值设置为该节点的索引。然后跟踪该数组 (`mainArr`)。

3. 选择 “控制” > “测试影片”对代码进行测试。

   您将在 “输出”面板中看到以下内容：

```
[0][0],[0][1],[0][2],[1][0],[1][1],[1][2],[2][0],[2][1],[2][2]
```

还可以使用嵌套的 for 循环来循环访问多维数组的元素，如下一个示例所示。

使用 for 循环迭代多维数组：

1. 创建一个新的 Flash 文档，并将其另存为 **multiArray3.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
// 从上面的示例中
var gridSize:Number = 3;
var mainArr:Array = new Array(gridSize);
var i:Number;
var j:Number;
for (i = 0; i < gridSize; i++) {
  mainArr[i] = new Array(gridSize);
  for (j = 0; j < gridSize; j++) {
    mainArr[i][j] = "[" + i + "][" + j + "]";
  }
}
```

   如上面的示例所示，在此代码中，外层循环将循环访问 mainArray 的每个元素。内层循环将循环访问每个嵌套的数组并输出每个数组节点。

3. 将下面的 ActionScript 添加到时间轴中的第 1 帧上，位于在步骤 2 中输入的代码之后：

```
// 循环访问元素
var outerArrayLength:Number = mainArr.length;
for (i = 0; i < outerArrayLength; i++) {
  var innerArrayLength:Number = mainArr[i].length;
  for (j = 0; j < innerArrayLength; j++) {
    trace(mainArr[i][j]);
  }
}
```

   此 ActionScript 循环访问数组中的各元素。将每个数组的 length 属性用作循环条件。

4. 选择 “控制” > “测试影片”来查看 “输出”面板中显示的元素。您将在 “输出”面板中看到以下内容：

```
[0][0]
[0][1]
[0][2]
[1][0]
[1][1]
[1][2]
[2][0]
[2][1]
[2][2]
```

有关使用数组的信息，请参见第 108 页的 “使用数组”。有关数组元素的信息，请参见第 111 页的 “关于添加和删除元素”。有关数组访问运算符的信息，请参见第 125 页的 “使用点运算符和数组访问运算符”。

创建关联数组

关联数组类似于对象，是由无序的键 和值 组成的。关联数组使用键而不是数字索引来组织存储的值。每个键都是一个唯一的字符串，与一个值相关联并用于访问该值。值可以是数字、数组、对象等数据类型。在创建代码以查找与一个键相关联的值时，您就是在创建索引或执行查找。这可能是关联数组最常见的用途。

键和值之间的关联通常称为绑定，键和值之间相互映射。例如，地址簿可以看作是一个关联数组，其中姓名是键，电子邮件地址是值。

注释	关联数组是键和值对的无序集合。在代码中，不应期望关联数组的键按特定的顺序排列。

使用关联数组时，可以使用字符串而不是数字调用所需的数组元素，这样往往会更容易记住。缺点是，这些关联数组在循环中没有什么用，因为它们不是用数字作为索引值的。在需要频繁地按键值查找时，可以使用 它们。例如，如果您需要用一个姓名和年龄的数组来引用更多内容，可以使用关联数组。

下面的示例演示了如何创建对象以及在关联数组中定义一系列属性。

创建简单的关联数组：

1. 创建一个新的 Flash 文档。
2. 在时间轴的第 1 帧上键入下面的 ActionScript：

```
// 定义用作关联数组的对象。
var someObj:Object = new Object();
// 定义一系列属性。
someObj.myShape = "Rectangle";
someObj.myW = 480;
someObj.myH = 360;
someObj.myX = 100;
someObj.myY = 200;
someObj.myAlpha = 72;
someObj.myColor = 0xDFDFDF;
// 使用点运算符和数组访问语法显示属性。
trace(someObj.myAlpha); // 72
trace(someObj["myAlpha"]); // 72
```

ActionScript 的第一行定义了一个新对象 (someObj)，用作关联数组。接下来，在 someObj 中定义一系列属性。最后，使用点运算符和数组访问语法显示您选择的属性。

注意	**可以使用两种不同的方法访问关联数组中的变量：点语法 (someObj.myColor) 和数组语法 (someObj['myColor'])。**

3. 选择 “控制” > “测试影片” 对 ActionScript 进行测试。

“输出” 面板中将显示数字 72 两次，分别代表您所跟踪的两种 alpha 级别。

在 ActionScript 2.0 中有两种创建关联数组的方法：

- 使用对象构造函数
- 使用数组构造函数

这两种方法都将在后面的示例中进行演示。

注意	上面的示例使用对象构造函数创建了一个关联数组。

如果使用对象构造函数创建关联数组，则可以利用一个对象文本初始化您的数组。对象类的实例 （也称 “通用对象”） 在功能上等同于关联数组。事实上，对象实例实质上就是关联数组。当使用字符串键比使用数字索引更方便时，可以用关联数组实现类似于字典的功能。通用对象的每个属性名称都用作键，提供对存储的值的访问。有关文本的更多信息，请参见第 79 页的 “关于文本”。有关类的更多信息，请参见第 163 页的第 6 章 “类”。

使用对象构造函数创建关联数组：

1. 创建一个新的 Flash 文档，并将其另存为 **assocArray.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace(monitorInfo["type"] + ", " + monitorInfo["resolution"]);
```

此代码创建一个名为 monitorInfo 的关联数组，并使用一个对象文本初始化此具有两个键 / 值对的数组。

注意	如果在声明数组时不需要初始化，可以使用对象构造函数来创建数组：

```
var monitorInfo:Object = new Object();
```

3. 选择 “控制” > “测试影片”。

“输出” 面板将显示以下文本：

```
Flat Panel, 1600 x 1200
```

4. 将下面的 ActionScript 添加到时间轴中的第 1 帧上，跟在前面输入的代码之后：

```
monitorInfo["aspectRatio"] = "16:10";
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspectRatio"] + ", " + monitorInfo.colors);
```

在使用对象文本或对象类构造函数创建了数组之后，可以像此代码所演示的那样使用中括号运算符 ([]) 或点运算符 (.) 向该数组中添加新值。您刚刚键入的代码将两个新值添加到 monitorInfo 数组中。

5. 选择 “控制” > “测试影片”。

“输出”面板将显示以下文本：

```
16:10, 16.7 million
```

请注意，键可以包含空格字符。也就是说，可以与中括号一起使用，但是在尝试与点运算符一起使用时会生成一个错误。不建议在键名称中使用空格。有关中括号运算符和点运算符的更多信息，请参见第 118 页的 “关于运算符”。有关格式良好的代码的更多信息，请参见第 679 页的 “设置 ActionScript 语法的格式”。

创建关联数组的第二种方法是使用数组构造函数，然后使用中括号运算符 ([]) 或点运算符 (.) 将键和值对添加到数组中。如果将关联数组声明为数组类型，则将无法使用对象文本初始化该数组。

注意 使用数组构造函数创建关联数组没有什么优势。数组构造函数最适合于创建索引数组。

下一个示例演示了如何使用数组构造函数创建关联数组。

使用数组构造函数创建关联数组：

1. 创建一个新的 Flash 文档，并将其另存为 **assocArray2.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"] + ", " + monitorInfo["resolution"]);
```

此代码使用数组构造函数创建一个名为 monitorInfo 的关联数组，并添加了一个名为 type 的键和一个名为 resolution 的键，以及它们的值。

3. 选择 “控制” > “测试影片”。

“输出”面板将显示以下结果：

```
Flat Panel, 1600 x 1200
```

注意 使用数组构造函数创建关联数组没有什么优势。数组构造函数最适合于创建索引数组。

关联数组实质上是对象类的实例，使用数组构造函数创建关联数组并没有什么优势。尽管使用 new Array() 构造函数创建了关联数组，但在使用关联数组时不能使用任何 Array 类的方法和属性（如 sort() 或 length）。如果希望使用键 / 值对而不是数字索引，则应该使用对象类而非关联数组。

关于运算符

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。

运算符是指定如何组合、比较或更改表达式中的值的字符。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。有关表达式的更多信息，请参见第 64 页的 “关于语法、语句和表达式”。

例如，数学表达式使用数值运算符操作您使用的值。运算符字符的示例包括 +、<、* 和 =。表达式由运算符和操作数 组成，是代表值的 ActionScript 元件的任意合法组合。操作数是代码中运算符对其执行动作的那部分内容。例如，在表达式 x + 2 中，x 和 2 是操作数，而 + 是运算符。

整个代码中会频繁地使用表达式和运算符。可以联合使用运算符和值来创建表达式，或者可以调用函数。

注意

本部分介绍了如何使用每种类型的运算符；但是，由于篇幅所限，未能详细讲述每一个运算符。有关每个运算符的信息，包括未归入以下类别的特殊运算符的信息，请参见 《ActionScript 2.0 语言参考》。

代码中运算符对其执行动作的部分称为操作数。例如，可以使用加法 (+) 运算符将数字文本的值加在一起。可以执行此操作加上名为 myNum 的变量的值。

```
myNum + 3;
```

在此例中，myNum 和 3 是操作数。

本部分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：

- 第 130 页的 “关于等于运算符”
- 第 131 页的 “使用关系运算符和等于运算符”
- 第 134 页的 “关于赋值运算符”
- 第 134 页的 “使用赋值运算符”
- 第 135 页的 “关于逻辑运算符”
- 第 135 页的 “使用逻辑运算符”
- 第 137 页的 “关于按位移位运算符”
- 第 137 页的 “关于按位逻辑运算符”
- 第 137 页的 “使用按位运算符”
- 第 139 页的 “关于条件运算符”
- 第 139 页的 “在文档中使用运算符”

有关未归入这些类别的运算符的信息，请参见 《ActionScript 2.0 语言参考》，其中包含关于可以使用的所有运算符的信息。

以下各部分介绍了运算符的一些常见用法。有关在单个代码范例中使用多个运算符的更多信息，请参见第 139 页的 “在文档中使用运算符”。

使用运算符操作值

在 Flash 中运算符通常用于操作值。例如，您可能要在 Flash 中创建一个游戏，其中玩家得分的改变取决于用户与舞台上的实例的交互。可以使用一个变量来保存该值，使用运算符操作该变量的值。

例如，您可能需要增大名为 `myScore` 的变量的值。下面的示例演示了如何在代码中使用 + （加法）和 += （加法赋值）运算符来执行加法和增大值。

使用运算符操作值：

1. 创建一个新的 Flash 文档。
2. 打开 “动作” 面板 （“窗口” > “动作”），将以下代码键入到 “脚本” 窗格中：

```
// 示例 1
var myScore:Number = 0;
myScore = myScore + 1;
trace("Example one: " + myScore); // 1

// 示例 2
var secondScore:Number = 1;
secondScore += 3;
trace("Example two: " + secondScore); // 4
```

3. 选择 “控制” > “测试影片”。

“输出”面板将显示以下文本：

```
Example one: 1
Example two: 4
```

加法运算符非常简单，因为它将两个值相加在一起。在第一个代码示例中，它将 myScore 的当前值和数字 1 相加在一起，然后将结果存储到变量 myScore 中。

第二个代码示例使用加法赋值运算符在一个步骤中完成相加并分配新值。可以将 myScore = myScore + 1 行（见前一个练习）重写为 myScore++ 甚至 myScore += 1。增量运算符 (++) 是表示 myScore = myScore + 1 的简化的方法，因为它同时处理递增和赋值。在下面的 ActionScript 中可以看到增量运算符的一个示例：

```
var myNum:Number = 0;
myNum++;
trace(myNum); // 1
myNum++;
trace(myNum); // 2
```

请注意，上面的代码片段中没有赋值运算符。而是依赖于增量运算符。

可以使用运算符在条件为 true 时操作变量的值。例如，可以使用增量运算符 (++) 在条件为 true 时使变量 i 的值递增。在下面的代码中，i 小于值 10 同时条件为 true。在条件为 true 时，使用 i++ 使 i 的值加 1。

```
var i:Number;
for (i = 1; i < 10; i++) {
  trace(i);
}
```

“输出”面板中将显示数字 1 到 9，因为 i 的值不断递增直到满足结束条件（i 等于 10）时停止。所显示的最后一个值是 9。因此， SWF 文件开始播放时 i 的值是 1，轨迹完成后值是 9。

有关条件和循环的更多信息，请参见第 88 页的 “关于语句”。

关于运算符的优先级和结合律

在一条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。运算符的优先级和结合律决定了处理运算符的顺序。ActionScript 具有一种确定哪些运算符在其它运算符之前执行的层次结构。本部分的结尾处用一个表格列出了这个层次结构。

虽然对于那些熟悉算术或基本编程的人来说，编译器先处理乘法运算符 (*) 然后处理加法运算符 (+) 似乎是自然而然的事情，但实际上编译器需要显式指定先处理哪些运算符。此类指令统称为运算符优先级。

在使用乘法和加法运算符时，可以看到运算符优先级的一个示例：

```
var mySum:Number;
mySum = 2 + 4 * 3;
trace(mySum); // 14
```

您会看到该语句的输出结果为 14，这是因为乘法具有更高的运算符优先级。因此，首先计算 4 * 3，而后再将结果与 2 相加。

可以通过将表达式括在小括号中控制执行顺序。 ActionScript 定义了一个默认的运算符优先级，可以使用小括号运算符 (()) 来改变它。当在加法表达式两端加上小括号时，ActionScript 会首先执行加法：

```
var mySum:Number;
mySum = (2 + 4) * 3;
trace(mySum); // 18
```

现在该语句的输出结果为 18。

运算符也可能具有相同的优先级。在这种情况下，结合律决定运算符执行的顺序。既可以有从左到右的结合律，也可以有从右到左的结合律。

再来看一看乘法运算符。它具有从左到右的结合律，因此以下两个语句是相同的。

```
var mySum:Number;
var myOtherSum:Number;
mySum = 2 * 4 * 3;
myOtherSum = (2 * 4) * 3;
trace(mySum); // 24
trace(myOtherSum); // 24
```

可能会遇到这样的情况：同一个表达式中出现两个或多个具有相同的优先级的运算符。在这些情况下，编译器使用结合律 的规则确定首先处理哪个运算符。除了赋值运算符之外，所有二进制运算符中都是左结合 的，也就是说，先处理左边的运算符然后再处理右边的运算符。赋值运算符和条件运算符 (?:) 都是右结合 的，也就是说先处理右边的运算符然后再处理左边的运算符。有关赋值运算符的更多信息，请参见第 134 页的 “使用赋值运算符”。有关条件运算符 (?:) 的更多信息，请参见第 139 页的 “关于条件运算符”。

例如，考虑小于运算符 (<) 和大于运算符 (>)，它们具有同样的优先级。如果将这两个运算符用于同一个表达式中，因为这两个运算符都是左结合的，所以首先处理左边的运算符。也就是说，以下两个语句将生成相同的输出结果：

```
trace(3 > 2 < 1);   // false
trace((3 > 2) < 1); // false
```

首先处理大于运算符 (>)，因为操作数 3 大于操作数 2，从而得到值 true。然后值 true 被传递给小于 (<) 运算符，与操作数 1 进行运算。小于运算符 (<) 将值 true 转换为数值 1 然后将该数值与第二个操作数 1 进行比较返回值 false （值 1 不小于 1）。

在 ActionScript 中请考虑一下操作数的顺序，尤其是在设置了复杂条件并且知道其中的一个条件为 true 的概率时。例如，如果您知道条件中的 i 将大于 50，则需要先写 i<50。因此，将首先检查它，所写的第二个条件不需要经常进行检查。

下表列出了所有 ActionScript 运算符及其结合律，按优先级从高到低排列。有关使用运算符和小括号的更多信息和准则，请参见第 679 页的第 19 章“设置 ActionScript 语法的格式”。

运算符	说明	结合律
	最高优先级	
x++	后递增	从左到右
x--	后递减	从左到右
.	对象属性访问	从左到右
[ ]	数组元素	从左到右
( )	小括号	从左到右
function ( )	函数调用	从左到右
++*x*	前递增	从右到左
--*x*	前递减	从右到左
-	一元非，例如 x = -1	从左到右
~	按位 “非”	从右到左
!	逻辑 “非”	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

运算符	说明	结合律
+	一元加号	从右到左
-	一元减号	从右到左
<<	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
instanceof	获取实例所属的类 （查找对象是哪个类的实例） 需要 Flash Player 6 或更高版本	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
==	等于	从左到右
!=	不等于	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
||	逻辑 “或”	从左到右
?:	条件	从右到左
=	赋值	从右到左
c, /=, %=, +=, -=, &=, |=, ^=, <<=, >>=, >>>=	复合赋值	从右到左
,	逗号	从左到右

最低优先级

关于对字符串使用运算符

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。此规则的一个例外是精确相等 (===) 运算符。如果只有一个操作数是字符串，ActionScript 会将两个操作数都转换为数字，然后对它们执行数值比较。有关数值运算符的更多信息，请参见第 129 页的 “使用数值运算符”。

除等于运算符 (==) 以外，其它比较运算符 （>、 >=、 < 和 <=）对字符串产生的影响与运算其它值时不同。

比较运算符对字符串进行比较，以确定哪一个字符串按字母表顺序排在前面。使用大写字母的字符串优先于使用小写字母的字符串。这意味着 “Egg”排在 “chicken”之前。

```
var c:String = "chicken";
var e:String = "Egg";
trace(c < e); // false
var riddleArr:Array = new Array(c, e);
trace(riddleArr); // chicken,Egg
trace(riddleArr.sort()); // Egg,chicken
```

在 ActionScript 中，数组类的 `sort()` 方法将数组的内容按字母表顺序重新排序。您可以看到值 “Egg”排在值 “chicken”前面，因为大写字母 E 排在小写字母 c 之前。如果要在比较字符串时不区分大小写，需要在比较之前将字符串都转换为大写或小写。有关比较运算符的更多信息，请参见第 130 页的 “关于等于运算符”和第 131 页的 “使用关系运算符和等于运算符”。

在比较字符串之前，可以使用 `toLowerCase()` 或 `toUpperCase()` 方法将它们转换为同样的大小写。在下面的示例中，两个字符串都转换为小写字符串然后进行比较，现在 chicken 排在 egg 之前：

```
var c:String = "chicken";
var e:String = "Egg";
trace(c.toLowerCase() < e.toLowerCase()); // true
```

注释 比较运算符仅比较两个字符串。例如，如果一个操作数是数值，运算符将不会比较这两个值。如果一个操作数是字符串， ActionScript 会将两个操作数都转换为数字，然后按数值比较它们。

可以使用运算符操作字符串。可以使用加法 (+) 运算符连接字符串操作数。在编写 `trace` 语句时，您可能已经使用加法运算符对字符串进行了连接。例如，您可以编写以下代码：

```
var myNum:Number = 10;
trace("The variable is " +  myNum + ".");
```

测试此代码时，“输出”面板中将显示以下内容：

```
The variable is 10.
```

在上面的示例中， `trace` 语句使用 + 运算符进行连接而不是相加。在处理字符串和数字时，Flash 有时会将它们连接起来，而不是将它们按数值加起来。

例如，可以在单个文本字段中连接来自两个不同变量的两个字符串。在下面的 ActionScript 代码中，变量 myNum 与一个字符串进行连接，并且字符串显示在舞台上的 myTxt 文本字段中。

```
this.createTextField("myTxt", 11, 0, 0, 100, 20);
myTxt.autoSize = "left";
var myNum:Number = 10;
myTxt.text = "One carrot. " + myNum + " large eggplants.";
myTxt.text += " Lots of vegetable broth.";
```

此代码在实例名称为 myTxt 的文本字段中输出了以下内容：

```
One carrot. 10 large eggplants. Lots of vegetable broth.
```

上面的示例说明了如何使用加法 (+) 和加法赋值 (+=) 运算符来连接字符串。请注意第三行代码是如何使用加法运算符将变量 myNum 的值连接到文本字段中的，以及第四行代码是如何使用加法赋值运算符将一个字符串连接到该文本字段中现有的值上的。

如果文本字符串操作数中只有一个实际上是字符串，则 Flash 会将另一个操作数转换为字符串。因此，在上面的示例中 myNum 的值转换成了一个字符串。

提示 ActionScript 将字符串开头或结尾的空格作为该字符串的文本部分。

使用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 ActionScript 属性。点运算符用于将一个对象中的特定的目标索引设定为目标。例如，如果您有一个包含某些用户信息的对象，则可以在数组访问运算符中指定一个特定的键名来检索用户的姓名，如下面的 ActionScript 所示：

```
var someUser:Object = {name:"Hal", id:2001};
trace("User's name is: " + someUser["name"]); // 用户的姓名是: Hal
trace("User's id is: " + someUser["id"]); // 用户的 ID 是: 2001
```

例如，下面的 ActionScript 使用点运算符设置对象内特定的属性：

```
myTextField.border = true;
year.month.day = 9;
myTextField.text = "My text";
```

点运算符和数组访问运算符非常类似。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计算为名称，然后访问该已命名属性的值。数组访问运算符使您能够动态地设置和检索实例名称和变量。

如果您不能准确了解一个对象中有哪些键，数组访问运算符会非常有用。出现这种情况时，可以使用 for..in 循环来循环访问对象或影片剪辑并显示其内容。

使用点运算符和数组访问运算符：

1. 在新的 Flash 文档中，在主时间轴上创建一个影片剪辑。
2. 选择该影片剪辑，并打开属性检查器。
3. 键入 myClip 的一个实例名称。
4. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
myClip.spam = 5;
trace(myClip.spam); // 5
```

   如果要在当前时间轴上的 myClip 实例中设置值，可以使用点运算符或数组访问运算符，如该 ActionScript 中所示。如果您在数组访问运算符中编写了一个表达式，则将首先计算该表达式并将结果作为变量名称。

5. 选择 “控制” > “测试影片” 对该文档进行测试。

   “输出” 面板中将显示 5。

6. 返回到创作环境，用下面的代码替换 ActionScript 的第一行：

```
myClip["spam"] = 10;
```

7. 选择 “控制” > “测试影片” 对该文档进行测试。

   “输出” 面板将显示 10。

8. 返回到创作环境，双击该 myClip 实例。
9. 在该 myClip 实例中添加四个新实例。
10. 使用属性检查器将以下实例名称分别添加到这四个新实例中：**nestedClip1**、**nestedClip2**、**nestedClip3**、**nestedClip4**。
11. 将下面的代码添加到主时间轴中的第 1 帧：

```
var i:Number;
for (i = 1; i <= 4; i++) {
  myClip["nestedClip" + i]._visible = false;
}
```

   此 ActionScript 切换每个嵌套的影片剪辑的可见性。

12. 选择 “控制” > “测试影片” 对刚刚添加的 ActionScript 进行测试。

    现在，这四个嵌套实例都不可见。您使用了数组访问运算符来循环访问 myClip 实例中的各个嵌套的影片剪辑，并动态设置其可见属性。因为不需要具体地瞄准各个实例，所以您可以节省时间。

您还可以在赋值语句的左侧使用数组访问运算符，这允许您动态地设置实例、变量和对象的名称：

```
myNum[i] = 10;
```

在 ActionScript 2.0 中，在没有为对象的类定义指定 dynamic 属性时，可以使用中括号运算符来访问该对象上动态创建的属性。还可以使用此运算符创建多维数组。有关使用数组访问运算符创建多维数组的更多信息，请参见第 112 页的 “创建多维数组”。

关于后缀运算符

后缀运算符只有一个操作数，并且这类运算符递增或递减该操作数的值。虽然这些运算符是一元运算符，但是它们有别于其它一元运算符，被单独划归到了一个类别，因为它们具有更高的优先级和特殊的行为。有关一元运算符的信息，请参见第 127 页的“关于一元运算符”。

在将后缀运算符用作较长表达式的一部分时，在处理后缀运算符之前返回表达式的值。例如，下面的代码显示了如何在值递增之前返回表达式 xNum++ 的值。

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum); // 1
```

跟踪此代码时，“输出”面板中显示的文本是：

```
0
1
```

此表中的运算符具有相同的优先级：

运算符	执行的运算
++	递增 （后缀）
--	递减 （后缀）

关于一元运算符

一元运算符只有一个操作数。此组中的递增 (++) 和递减 (--) 运算符是前缀 运算符，这意味着它们在表达式中出现在操作数的前面。它们还可以出现在操作数的后面，这种情况下它们是后缀 运算符。有关后缀运算符的信息，请参见第 127 页的 “关于后缀运算符”。

前缀运算符与它们对应的后缀运算符不同，因为递增或递减操作是在返回整个表达式的值之前完成的。例如，下面的代码显示了如何在值递增之后返回表达式 xNum++ 的值。

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum); // 1
```

此表中的所有运算符都具有相同的优先级：

运算符	执行的运算
++	递增 （前缀）
--	递减 （前缀）
+	一元 +
!	一元 - （非）
typeof	返回类型信息
void	返回未定义值

关于乘法运算符

乘法运算符有两个操作数，执行乘、除或求模计算。其它数值运算符包括加法运算符。有关加法运算符的信息，请参见第 128 页的 “关于加法运算符”。

此表中的所有运算符都具有相同的优先级：

运算符	执行的运算
*	乘法
/	除法
%	求模

有关使用乘法运算符的信息，请参见第 129 页的 “使用数值运算符”。

关于加法运算符

加法运算符有两个操作数，分别执行加法或减法计算。其它数值运算符包括乘法运算符。有关乘法运算符的信息，请参见第 128 页的 “关于乘法运算符”。

此表中的运算符具有相同的优先级：

运算符	执行的运算
+	加法
-	减法

有关使用加法运算符的信息，请参见第 129 页的 “使用数值运算符”。

使用数值运算符

在 ActionScript 中，可以使用数值运算符来对值进行加、减、乘、除运算。可以执行不同种类的算术运算。最常见的一种运算符是递增运算符，其常见形式为 i++。可以使用此运算符来做许多事情。有关递增运算符的更多信息，请参见第 119 页的 “使用运算符操作值”。

可以在操作数前面 （前递增）或后面 （后递增）添加递增运算符。

理解 ActionScript 中的数值运算符：

1. 创建一个新的 Flash 文档。
2. 在时间轴的第 1 帧上键入下面的 ActionScript：

```
// 示例 1
var firstScore:Number = 29;
if (++firstScore >= 30) {
  // 应跟踪
  trace("Success! ++firstScore is >= 30");
}
// 示例 2
var secondScore:Number = 29;
if (secondScore++ >= 30) {
  // 不应跟踪
  trace("Success! secondScore++ is >= 30");
}
```

3. 选择 “控制” > “测试影片” 对 ActionScript 进行测试

   “示例 1” 代码块进行跟踪，而 “示例 2” 代码块不进行跟踪。第一个示例在与 30 进行比较之前会使用前递增 (++firstScore) 来递增并计算 firstScore 的值。因此，firstScore 会先递增到 30，然后才与 30 进行比较。

   但是，示例二使用后递增 (secondScore++) 来在执行比较之后进行计算。因此，29 会先与 30 进行比较，然后在进行计算之后再增加到 30。

有关运算符优先级的更多信息，请参见第 121 页的 “关于运算符的优先级和结合律”。

从外部源（例如 XML 文件、FlashVars、Web 服务等）中加载数据时，在使用数值运算符时要非常小心。因为 SWF 文件不能识别数字的数据类型，所以有时 Flash 会将数字视为字符串。在这种情况下，3 加 7 的结果可能会得到 37，因为 Flash 将这两个数字作为字符串来连接，而不是按数值相加。此时，您需要使用 Number() 函数手动将数据从字符串转换为数字。

关于关系运算符

关系运算符有两个操作数，它比较两个操作数的值，然后返回一个布尔值。此表中的所有运算符都具有相同的优先级：

运算符	执行的运算
<	小于
>	大于
<=	小于或等于
>=	大于或等于
instanceof	检查原型链
in	检查对象属性

有关使用关系运算符的信息，请参见第 131 页的 “使用关系运算符和等于运算符”。

关于等于运算符

等于运算符有两个操作数，它比较两个操作数的值，然后返回一个布尔值。此表中的所有运算符都具有相同的优先级：

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

有关使用等于运算符的信息，请参见第 131 页的 “使用关系运算符和等于运算符”。

使用关系运算符和等于运算符

关系运算符和等于运算符（又称比较运算符）比较表达式的值，然后返回 `true` 或 `false`（一个布尔值）。在条件语句和循环中经常会用比较运算符来指定确定循环何时停止的条件。

可以使用等于 (==) 运算符判断两个操作数的值或引用是否相等，这种比较会返回一个布尔值。字符串、数字或布尔操作数值是使用值进行比较的。对象和数组操作数是通过引用进行比较的。

在此例中，您可以看到如何使用等于运算符来测试数组的长度，并在数组中没有项时在“输出”面板中显示一条消息。

```
var myArr:Array = new Array();
if (myArr.length == 0) {
  trace("the array is empty.");
}
```

当选择“控制”>“测试影片”时，字符串 `the array is empty` 将显示在“输出”面板中。

可以使用等于运算符来比较值，但是不能用它来设置值。您可能会尝试用赋值运算符 (=) 来检查等式。

在代码中使用关系运算符和等于运算符：

1. 创建一个新的 Flash 文档。
2. 在时间轴的第 1 帧上键入下面的 ActionScript：

   ```
   var myNum:Number = 2;
   if (myNum == 2) {
     // 完成一些操作
     trace("It equals 2");
   }
   ```

   在此 ActionScript 中，您使用等于运算符 (==) 来检查是否相等。检查变量 `myNum` 是否等于 2。
3. 选择“控制”>“测试影片”。

   “输出”面板中显示字符串 `It equals 2`。
4. 返回到创作环境，并将：

   ```
   var myNum:Number = 2;
   ```

   更改为：

   ```
   var myNum:Number = 4;
   ```
5. 再次选择“控制”>“测试影片”。

   “输出”面板中不显示字符串 `It equals 2`。

6. 返回到创作环境，并将：

```
if (myNum == 2) {
```

更改为：

```
if (myNum = 2) {
```

7. 再次选择 “控制” > “测试影片”。

“输出”面板中显示字符串 `It equals 2`。

在步骤 6 中，将值 `2` 赋予 `myNum`，而不是将 `myNum` 与 `2` 进行比较。在这种情况下，无论 `myNum` 以前的值是多少，`if` 语句都会执行，这样在测试 Flash 文档时可能会导致意想不到的结果。

有关正确使用赋值运算符的更多信息，请参见第 134 页的 “使用赋值运算符”。

严格等于运算符 (===) 类似于等于运算符，不同之处只是它不执行类型转换。如果两个操作数是不同的类型，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。

下面的 ActionScript 演示了等于运算符 (==) 和严格等于运算符 (===) 之间的主要区别：

```
var num1:Number = 32;
var num2:String = new String("32");
trace(num1 == num2); // true
trace(num1 === num2); // false
```

首先，定义数值变量：`num1` 和 `num2`。如果使用等于运算符比较变量，Flash 会尝试将两个值转换为相同的数据类型，然后比较这两个值看它们是否相等。使用严格等于运算符 (===) 时，Flash 在比较值之前不会尝试执行任何数据类型转换。结果，Flash 将这两个变量看作两个不同的值。

在下面的示例中，您将使用大于或等于运算符 (>=) 来比较值，并根据用户在文本字段中输入的值执行代码。

在代码中使用大于或等于运算符：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”创建一个新的 FLA 文件。
2. 将下面的代码添加到主时间轴中的第 1 帧：

```
this.createTextField("myTxt", 20, 0, 0, 100, 20);
myTxt.type = "input";
myTxt.border = true;
myTxt.restrict = "0-9";

this.createEmptyMovieClip("submit_mc", 30);
submit_mc.beginFill(0xFF0000);
submit_mc.moveTo(0, 0);
submit_mc.lineTo(100, 0);
submit_mc.lineTo(100, 20);
submit_mc.lineTo(0, 20);
submit_mc.lineTo(0, 0);
submit_mc.endFill();
submit_mc._x = 110;

submit_mc.onRelease = function(evt_obj:Object):Void {
  var myNum:Number = Number(myTxt.text);
  if (isNaN(myNum)) {
    trace("Please enter a number");
    return;
  }
  if (myNum >= 10) {
    trace("Your number is greater than or equal to 10");
  } else {
    trace("Your number is less than 10");
  }
};
```

3. 选择 “控制” > “测试影片”对 ActionScript 进行测试。

   还可以检查特定的条件是否为 true，在条件不为 true 时执行替代代码块。
4. 在 ActionScript 中将条件改为以下形式。

```
if (myNum == 10) {
  trace("Your number is 10");
} else {
    trace("Your number is not 10");
}
```

5. 选择 “控制” > “测试影片”再次对 ActionScript 进行测试。

除严格等于 (===) 运算符以外，其它比较运算符都只在两个操作数都是字符串时才比较字符串。如果只有一个操作数是字符串，那么两个操作数都将转换为数字，然后执行数值比较。有关字符串和运算符的更多信息，请参见第 124 页的 “关于对字符串使用运算符”。有关顺序和运算符优先级将如何影响 ActionScript 的信息，请参见第 121 页的 “关于运算符的优先级和结合律”。

关于赋值运算符

赋值运算符有两个操作数，它根据一个操作数的值对另一个操作数进行赋值。此表中的所有运算符都具有相同的优先级：

运算符	执行的运算
=	赋值
*=	乘法赋值
/=	除法赋值
%=	求模赋值
+=	加法赋值
-=	减法赋值
<<=	按位左移位赋值
>>=	按位右移位赋值
>>>=	按位无符号右移位赋值
&=	按位 “与” 赋值
^=	按位 “异或” 赋值
|=	按位 “或” 赋值

有关使用赋值运算符的信息，请参见第 134 页的 “使用赋值运算符”。

使用赋值运算符

可以使用赋值运算符 (=) 将一个给定的值赋予一个变量。可以将一个字符串赋予一个变量，如下所示：

```
var myText:String = "ScratchyCat";
```

还可以使用赋值运算符给同一表达式中的几个变量赋值。在下面的语句中，值 10 会被赋予变量 numOne、numTwo 和 numThree。

```
var numOne:Number;
var numTwo:Number;
var numThree:Number;
numOne = numTwo = numThree = 10;
```

也可以使用复合赋值运算符组合多个运算。此类运算符可以对两个操作数都执行运算，然后将新值赋予第一个操作数。例如，下面这两个语句具有相同的功能：

```
var myNum:Number = 0;
myNum += 15;
myNum = myNum + 15;
```

使用赋值运算符时，如果试图在表达式中添加值，则可能会遇到问题，如下例中所示：

```
trace("the sum of 5 + 2 is: " + 5 + 2); // 5 + 2 的和是: 52
```

Flash 将值 5 和 2 连接在一起，而不是将它们相加在一起。要避免这种情况，可以用一对小括号将表达式 5+2 括起来，如下面的代码中所示：

```
trace("the sum of 5 + 2 is: " + (5 + 2)); // 5 + 2 的和是: 7
```

关于逻辑运算符

可以使用逻辑运算符对布尔值 （true 和 false）进行比较，然后根据比较结果返回一个布尔值。例如，如果两个操作数都计算为 true，则逻辑 “与”运算符 (&&) 将返回 true。如果其中一个或两个操作数都计算为 true，则逻辑 “或”运算符 (||) 将返回 true。

逻辑运算符有两个操作数，这类运算符返回布尔结果。逻辑运算符具有不同的优先级；下表中按优先级递减的顺序列出了逻辑运算符：

运算符	执行的运算
&&	逻辑 “与”
||	逻辑 “或”

有关使用逻辑运算符的信息，请参见第 135 页的 “使用逻辑运算符”。

使用逻辑运算符

通常将逻辑运算符与比较运算符结合使用，以确定 if 语句的条件。下一个示例演示了这一做法。

在代码中使用逻辑运算符：

1. 选择 “文件” > “新建”，然后创建新的 Flash 文档。
2. 打开 “动作”面板，在时间轴的第 1 帧上键入以下 ActionScript：

```
this.createTextField("myTxt", 20, 0, 0, 100, 20);
myTxt.type = "input";
myTxt.border = true;
myTxt.restrict = "0-9";

this.createEmptyMovieClip("submit_mc", 30);
submit_mc.beginFill(0xFF0000);
submit_mc.moveTo(0, 0);
submit_mc.lineTo(100, 0);
submit_mc.lineTo(100, 20);
submit_mc.lineTo(0, 20);
submit_mc.lineTo(0, 0);
submit_mc.endFill();
```

```
submit_mc._x = 110;

submit_mc.onRelease = function():Void {
  var myNum:Number = Number(myTxt.text);
  if (isNaN(myNum)) {
    trace("Please enter a number");
    return;
  }
  if ((myNum > 10) && (myNum < 20)) {
    trace("Your number is between 10 and 20");
  } else {
    trace("Your number is NOT between 10 and 20");
  }
};
```

在此 ActionScript 中，您在运行时创建了一个文本字段。如果在该文本字段中键入一个数字并单击舞台上的按钮，Flash 将使用逻辑运算符在“输出”面板中显示一条消息。消息的内容取决于键入到该文本字段中的值。

使用操作数时，需要注意顺序。在使用复杂条件时，这一点尤其值得注意。在下面的代码片段中，您可以看到如何使用逻辑 “与”运算符来检查一个数字是否在 10 和 20 之间。根据此结果，显示适当的消息。如果该数字小于 10 或大于 20，“输出”面板中将显示一条替代消息。

```
submit_mc.onRelease = function():Void {
  var myNum:Number = Number(myTxt.text);
  if (isNaN(myNum)) {
    trace("Please enter a number");
    return;
  }
  if ((myNum > 10) && (myNum < 20)) {
    trace("Your number is between 10 and 20");
  } else {
    trace("Your number is NOT between 10 and 20");
  }
};
```

关于按位移位运算符

按位移位运算符有两个操作数，将第一个操作数的各位按第二个操作数指定的长度移位。此表中的所有运算符都具有相同的优先级：

运算符	执行的运算
<<	按位左移位
>>	按位右移位
>>>	按位无符号右移位

有关使用按位运算符的信息，请参见第 137 页的 “使用按位运算符”。有关每种按位运算符的具体信息，请参见 《ActionScript 2.0 语言参考》中的相应条目。

关于按位逻辑运算符

按位逻辑运算符有两个操作数，执行位级别的逻辑运算。按位逻辑运算符具有不同的优先级；下表中按优先级递减的顺序列出了按位逻辑运算符：

运算符	执行的运算
&	按位 “与”
^	按位 “异或”
|	按位 “或”

有关使用按位运算符的信息，请参见第 137 页的 “使用按位运算符”。有关每种按位运算符的具体信息，请参见 《ActionScript 2.0 语言参考》中的相应条目。

使用按位运算符

按位运算符在内部操作浮点数，将它们转换为 32 位整数。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 位整数的每个二进制位，从而计算新的值。有关按位移位运算符的列表，请参见第 137 页的 “关于按位移位运算符”。有关按位逻辑运算符的列表，请参见第 137 页的 “关于按位逻辑运算符”。

按位运算符在 Flash 中不常用到，但是在某些情况下可能会非常有用。例如，您可能要为一个 Flash 项目构建一个权限列表，但是又不想为每种权限类型创建单独的变量。在这种情况下，可以使用按位运算符。

下面的示例说明了如何将按位 “或” 运算符与 `Array.sort()` 方法结合使用来指定排序选项。

使用按位 “或”运算符：

1. 选择 “文件” > “新建”，然后创建新的 Flash 文档。
2. 在 “动作”面板中键入以下 ActionScript：

```
var myArr:Array = new Array("Bob", "Dan", "doug", "bill", "Hank", "tom");
trace(myArr); // Bob,Dan,doug,bill,Hank,tom
myArr.sort(Array.CASEINSENSITIVE | Array.DESCENDING);
trace(myArr); // tom,Hank,doug,Dan,Bob,bill
```

   第一行定义一个随机姓名的数组，并将每个项显示到 “输出”面板中。然后调用 `Array.sort()` 方法，并使用常数值 `Array.CASEINSENSITIVE` 和 `Array.DESCENDING` 指定两个排序选项。该 sort 方法的结果是使数组中的项目按反向顺序 （z 到 a）排序。此搜索是不区分大小写的； `a` 和 `A` 被看作是相同的；但在区分大小写的搜索中， `Z` 排在 `a` 之前。

3. 选择 “控制” > “测试影片”对 ActionScript 进行测试。下面的文本在 “输出”面板中显示：

```
Bob,Dan,doug,bill,Hank,tom
tom,Hank,doug,Dan,Bob,bill
```

在 sort 方法中有五种可用的选项：

- 1 或 Array.CASEINSENSITIVE （二进制 = 1）
- 2 或 Array.DESCENDING （二进制 = 10）
- 4 或 Array.UNIQUESORT （二进制 = 100）
- 8 或 Array.RETURNINDEXEDARRAY （二进制 = 1000）
- 16 或 Array.NUMERIC （二进制 = 10000）

有三种不同的方法可用于为数组定义排序选项：

```
my_array.sort(Array.CASEINSENSITIVE | Array.DESCENDING); // 常数
my_array.sort(1 | 2); // 数字
my_array.sort(3); // 将数字加起来
```

虽然它们不是很容易就可以看出来，但是排序选项的数值实际上是按位数字 （二进制或以 2 为基数）。常量值 `Array.CASEINSENSITIVE` 等于数值 1，而且正好还是二进制值 1。常量值 `Array.DECENDING` 具有数值 2 或二进制值 10。

使用二进制数字可能会造成混乱。二进制只有两个可能的值 （1 或 0），这就是为何会将值 2 表示为 10 的原因。如果要用二进制显示数字 3，将是 11 (1+10)。数字 4 用二进制表示为 100，5 用二进制表示为 101，依此类推。

下面的 ActionScript 演示了如何通过使用按位 “与”运算符将常数 Array.DESCENDING 和 Array.NUMERIC 相加在一起来按降序对一个数值数组进行排序。

```
var scores:Array = new Array(100,40,20,202,1,198);
trace(scores); // 100,40,20,202,1,198
trace(scores.sort()); // 1,100,198,20,202,40
var flags:Number = Array.NUMERIC|Array.DESCENDING;
trace(flags); // 18 (以 10 为底)
trace(flags.toString(2)); // 10010 (二进制 -- 以 2 为底)
trace(scores.sort(flags)); // 202,198,100,40,20,1
```

关于条件运算符

条件运算符是一个三元运算符，也就是说它有三个操作数。条件运算符是应用 if..else 条件语句的一种简便方法：

运算符	执行的运算
?:	条件

有关使用条件运算符的信息和示例，请参见第 97 页的 “关于条件运算符和替代语法”。

在文档中使用运算符

下面的示例中使用 Math.round() 方法对任意位数的小数进行了四舍五入计算。此方法将参数 x 的值向上或向下舍入为最接近的整数，然后返回值。对该 ActionScript 稍做修改之后，将使 Flash 能够将数字舍入到特定小数位。

在下一个示例中，还将使用除法和乘法运算符根据答对的题数与总题数相除的结果计算用户的分数。用户的分数可乘以一个数，以显示为一个 0% 到 100% 之间的数。然后使用加法运算符将用户的分数连接到 “输出”面板中显示的一个字符串中。

在 ActionScript 中使用运算符：

1. 创建一个新的 Flash 文档。
2. 在主时间轴的第 1 帧上键入下面的 ActionScript：

```
var correctAnswers:Number = 11;
var totalQuestions:Number = 13;
// 四舍五入为最接近的整数
//var score:Number = Math.round(correctAnswers / totalQuestions * 100);
// 四舍五入为两位小数
var score:Number = Math.round(correctAnswers / totalQuestions * 100 * 100)
  / 100;
trace("You got " + correctAnswers + " out of " + totalQuestions + "
  answers correct, for a score of " + score + "%.");
```

3. 选择 “控制” > “测试影片”。

   “输出”面板将显示以下文本：

   ```
   You got 11 out of 13 answers correct, for a score of 84.62%.
   ```

   在该示例中调用 `Math.round()` 时，分数四舍五入为最接近的整数 (85) 并显示在 “输出”面板中。如果在调用 `Math.round()` 之前将该数字乘以 100，然后再除以 100，可以使 Flash 将该数字四舍五入为 2 位小数。这将得到一个更精确的分数。

4. 试着将变量 `correctAnswers` 改为 3，然后选择“控制”>“测试影片”来再次对该 SWF 文件进行测试。

如果您正在构建一个测试应用程序，可能需要使用 RadioButton 和 Label 组件创建一系列 true/false 问题或多项选择问题。在用户答完所有问题并单击 “提交”按钮后，可以将用户的答案与标准答案进行比较，然后计算他们的分数。

第 5 章

函数和方法

5

在编写 ActionScript、创建类和使用方法时，了解各种函数的使用是非常重要的。在 Flash 8 中有几种不同类型的函数可供使用。在本章中，您将了解有关函数和方法的知识：在使用内置类时如何在应用程序中使用和编写这些函数。在第 6 章 “类”中，您将创建自定义类，并经常对这些类编写函数。还将了解到如何在 ActionScript 类文件中编写函数。

您可以在自己的代码中利用这些函数向应用程序中添加交互性、动画和其它效果。本章涵盖了多种可在 Flash 应用程序中编写的函数。有关各个函数和方法的信息，以及在 Flash 中编写和使用函数及方法的练习，请参见以下主题：

关于函数和方法

方法和函数是一些可以在 SWF 文件中的任意位置重复使用的 ActionScript 代码块。您可以在 FLA 文件或外部 ActionScript 文件中编写自己的函数，然后可以从文档内的任意位置调用该函数。方法只是一些位于 ActionScript 类定义中的函数。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。在定义了函数后，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。

如果您将值作为参数传递给函数，则函数就可以使用提供的值进行计算。每个函数都有其各自的特性，而某些函数需要您传递特定类型或数量的值。如果传递的参数多于函数的需要，该函数将忽略多余的值。如果您没有传递必需的参数，则函数将为空的参数指定 `undefined` 数据类型。这可能导致在运行时出错。函数也可以返回值（请参见第 158 页的 “从函数中返回值”）。

注意	若要调用函数，该函数定义必须位于播放头所到达的帧中。

您可以将编写完善的函数看作一个 “黑匣子”。如果该函数包含详细的输入、输出和用途注释，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。

简单命名函数的基本语法为：

```
function traceMe() {
  trace("your message");
}
traceMe();
```

有关编写命名函数的信息，请参见第 146 页的 “编写命名函数”。

根据上一示例通过传递一个参数 yourMessage 构建的简单命名函数的基本语法为：

```
function traceMe(yourMessage:String) {
  trace(yourMessage);
}
traceMe("How you doing?");
```

或者，如果想要传递多个参数，可以使用以下代码：

```
var yourName:String = "Ester";
var yourAge:String = "65";
var favSoftware:String = "Flash";
function traceMe(favSoftware:String, yourName:String, yourAge:String) {
  trace("I'm " + yourName + ", I like " + favSoftware + ", and I'm " +
  yourAge + ".");
}
traceMe(favSoftware,yourName,yourAge);
```

有关传递参数的更多信息，请参见第 156 页的 “将参数传递给函数”。

您可以编写多种类型的函数。有关编写函数的更多信息，以及指向编写特定类型函数的各个部分的链接，请参见第 143 页的 “关于方法和函数的类型”。有关将方法和函数进行对比的示例，请参见第 160 页的 “了解方法”。

注意

有关使用 “脚本助手”编写代码的信息，请参见第 270 页的 “使用 “脚本助手”编写 ActionScript”、第 272 页的 “使用 “脚本助手”创建 startDrag/stopDrag 事件”以及 “ActionScript：使用脚本助手模式”教程 （该教程从第 194 页的 “打开起始文档”处开始）。

有关函数和方法的更多信息，请参见以下主题：

- 第 143 页的 “关于方法和函数的类型”

关于方法和函数的类型

属于一个类的函数称作该类的方法。在应用程序中可以使用多种类型的函数，其中包括内置函数、命名函数和用户定义的函数、匿名函数、回调函数、构造函数和函数文本。以下各部分包含如何定义这些函数的信息。

您还可以在 ActionScript 类文件中编写函数。可以在脚本中把这些函数作为方法使用。在以下的示例中， Person 类显示了构造函数方法、类方法、实例方法和存取器方法 （getter 和 setter）。此代码范例中的注释指明了这些方法在代码中出现的位置。

注意 有关编写类文件 （例如下面这个文件）的信息，请参见第 163 页的第 6 章 “类”。

```
class Person {
  public static var numPeople:Number = 0;

  // 实例成员
  private var _speed:Number;

  // 构造函数
  public function Person(speed:Number) {
    Person.numPeople++;
    this._speed = speed;
  }

  // 静态方法
  public static function getPeople():Number {
    return Person.numPeople;
  }

  // 实例方法
  public function walk(speed:Number):Void {
    this._speed = speed;
  }
  public function run():Void {
    this._speed *= 2;
  }
  public function rest():Void {
    this._speed = 0;
  }

  // getter/setter (存取器方法)
  public function get speed():Number {
    return this._speed;
  }
}
```

有关如何编写诸如以上代码范例中方法的完整演示，请参见第 163 页的第 6 章“类”。代码中使用的方法可能属于内置于 ActionScript 语言中的类。MovieClip 和 Math 就是应用程序中可能会使用的顶级类的例子。当您在代码中使用来自这些类的函数时，它们是在内置类中编写的函数（与上述代码范例类似）。另外，您还可以使用自己编写的自定义类中的方法。

不属性于任何类的函数称为顶级函数（有时称为预定义函数 或内置函数），这意味着您可以直接调用它们，而无需使用构造函数。内置于 ActionScript 语言顶级的函数示例有 `trace()` 和 `setInterval()`。

若要向代码中添加顶级函数调用，只需在 “动作”面板的 “脚本”窗格中添加单行代码即可。例如，键入以下内容：

```
trace("my message");
```

当您使用这一单行代码测试 SWF 文件时，将调用顶层函数 `trace()`，而且文本将显示在“输出”面板中。

切记：若要将方法分配给属性，则应省略方法名称后的圆括号，因为您正在向函数传递引用：

```
my_mc.myMethod = aFunction;
```

不过，当您在自己的代码内调用方法时，则需要在方法名称后面带上圆括号：

```
my_mc.myMethod();
```

注意 有关顶级函数的更多信息，请参见第 145 页的 “关于内置函数和顶级函数”。

您还可以用多种其它方法定义函数。有关每种类型函数的更多信息，请参见以下各部分：

- 第 145 页的 “关于内置函数和顶级函数”
- 第 146 页的 “编写命名函数”
- 第 147 页的 “编写匿名函数和回调函数”
- 第 149 页的 “关于函数文本”
- 第 151 页的 “将用户定义的函数设定为目标并进行调用”
- 第 150 页的 “关于构造函数”

有关编写和使用函数及方法的信息，请参见以下相关部分。有关使用函数的信息，请参见第 152 页的 “在 Flash 中使用函数”。有关使用方法的信息，请参见第 160 页的 “了解方法”。

注意 有关使用 “脚本助手”编写代码的信息，请参见第 270 页的 “使用 “脚本助手”编写 ActionScript”、第 272 页的 “使用 “脚本助手”创建 startDrag/stopDrag 事件”和 “ActionScript：使用脚本助手模式”教程 （该教程从第 194 页的 “打开起始文档”处开始）。

关于内置函数和顶级函数

正如第 141 页的“关于函数和方法”中所讨论的那样，函数是可以在 SWF 文件中的任意位置重复使用的 ActionScript 代码块。如果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。

您可以使用 ActionScript 语言中的内置函数。它们可以是第 143 页的 “关于方法和函数的类型”中所述的顶级函数；也可以是 Math 或 MovieClip 这样的内置类中的函数，可以在应用程序中将内置类中的函数当作方法使用。

您可以在 ActionScript 中使用内置函数来执行特定任务和访问信息。例如，您可以通过使用 `getTimer()` 来获取 SWF 文件已经播放的毫秒数。或者使用 `getVersion()` 获取承载该文件的 Flash Player 的版本号。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 “全局函数”类别的子类别中找到这些函数。

有些内置函数要求传递特定的值。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，这可能会导致在运行时出错。

注意	若要调用函数，该函数定义必须位于播放头所到达的帧中。

顶级函数使用起来非常方便。若要调用函数，只需使用函数名称并传递该函数所有必需的参数即可。（有关必需参数的信息，请参见 《ActionScript 2.0 语言参考》中的函数条目）。例如，将下面的 ActionScript 添加到时间轴的第 1 帧：

```
trace("my message");
```

在测试 SWF 文件时，`my message` 会出现在 “输出”面板中。顶级函数的两个其它示例是 `setInterval()` 和 `getTimer()`。下一示例将演示如何使用这两个函数。将下面的代码添加到时间轴中的第 1 帧：

```
function myTimer():Void {
  trace(getTimer());
}
var intervalID:Number = setInterval(myTimer, 100);
```

此代码使用 `getTimer()` 创建了一个简单的计时器，并使用 `setInterval()` 和 `trace()` 两个顶级函数显示 SWF 文件在 Flash Player 中播放的毫秒数。

调用顶级函数与调用用户定义的函数一样。有关更多信息，请参见第 151 页的 “将用户定义的函数设定为目标并进行调用”。有关每个函数的信息，请参见 《ActionScript 2.0 语言参考》中的相应条目。

编写命名函数

命名函数是一种通常在 ActionScript 代码中创建用来执行所有类型操作的函数。在创建 SWF 文件时，将首先编译命名函数，这表示只要已在当前帧或前面的帧中定义了函数，就可以在代码中的任意位置引用该函数。例如，如果在时间轴的第 2 帧中定义了一个函数，就不能在时间轴的第 1 帧中访问该函数。

命名函数的标准格式如下所示：

```
function functionName(parameters) {
  // 函数块
}
```

这一代码段包含以下部分：

- `functionName` 是函数的唯一名称。文档中所有函数的名称都必须唯一。
- `parameters` 包含您将传递给该函数的一个或多个参数。参数有时被称作参量。有关参数的更多信息，请参见第 156 页的 “将参数传递给函数”。
- `// function block` 包含由函数执行的所有 ActionScript 代码。这一部分包含 “执行任务”的语句。您可以将需要执行的代码放在这里。`// function block` 注释是一个占位符，函数块的代码将放入此处。

使用命名函数：

1. 创建一个名为 **namedFunc.fla** 的新文档。
2. 选择 “文件” > “导入” > “导入到库”，然后选择一个声音文件，将一个短小的声音文件导入到库中。
3. 右键单击该声音文件，并选择 “链接”。
4. 在 “标识符”文本框中键入 **mySoundID**。
5. 在时间轴中选择第 1 帧，并在 “动作”面板中，添加下面的代码：

```
function myMessage() {
  trace("mySoundID completed");
}
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = myMessage;
my_sound.start();
```

在此代码中创建了一个名为 `myMessage` 的命名函数，稍后在脚本中使用此函数来调用 `trace()` 函数。

6. 选择 “控制” > “测试影片” 对 SWF 文件进行测试。

您可以使用 `function` 语句在 ActionScript 中创建自己的函数。切记：参数是可选的；但是，如果您没有参数，仍需要带上括号。大括号 ({}) 中的内容叫做函数块。

您可以在主时间轴上或外部 ActionScript 文件 （包括类文件）内编写函数。

您还可以使用此格式在类文件中编写构造函数 （不过，函数的名称必须与类匹配）。有关构造函数的更多信息，请参见第 199 页的 “编写构造函数”。另请参见第 163 页的第 6 章 “类”以了解关于编写类中的函数的信息和示例。

编写匿名函数和回调函数

命名函数是可在定义之前或定义之后被脚本引用的函数，而匿名函数 则是引用自身的未命名函数；匿名函数在创建时便被引用。在编写 ActionScript 代码时，您将创建许多匿名函数。

在使用事件处理程序时通常要使用匿名函数。若要编写匿名函数，您需要将函数文本存储在变量中。这样，您就可以在后面的代码中引用此函数。下一示例演示如何编写匿名函数。

编写匿名函数：

1. 在 “舞台” 上创建一个影片剪辑，然后选择该剪辑。
2. 打开属性检查器，在 “实例名称” 文本框中键入 **my_mc**。
3. 在时间轴中选择第 1 帧，在 “动作” 面板中输入下面的代码：

```
var myWidth = function () {
  trace(my_mc._width);
};
// 在代码后面的部分中可以添加
myWidth();
```

4. 选择 “控制” > “测试影片”。

   影片剪辑的宽度便会显示在 “输出” 面板中。

您还可以在对象（如 XML 或 LoadVars 实例）内创建函数。可以将匿名函数与特定的事件关联，以创建回调函数。函数将在特定事件发生后调用回调函数，如在某些内容完成加载后 (`onLoad()`) 或在完成动画后 (`onMotionFinished()`)。

例如，有时需要编写 ActionScript 来处理从服务器加载到 SWF 文件的数据。将数据加载到 SWF 文件之后，即可从该位置访问数据。使用 ActionScript 检查是否完整地加载了数据非常重要。可以使用回调函数发送一个信号，指明数据已加载到文档中。

在下面这个加载远程 XML 文档的回调函数中，您将一个匿名函数与 onLoad() 事件关联起来。您可以使用 XML.load() 和回调函数，如下面的例子中所示。将下面的代码输入到时间轴中的第 1 帧上：

```
var my_xml:XML = new XML();
my_xml.onLoad = function(success:Boolean):Void {
  trace(success);
};
my_xml.load("http://www.helpexamples.com/crossdomain.xml");
```

从前面的代码片断中可以看到，onLoad() 事件处理程序使用了匿名函数来处理 onLoad() 事件。

有关回调函数的更多信息，请参见第 255 页的第 9 章 “处理事件”。

您还可以结合 setInterval() 函数使用匿名函数，如下面的代码所示，该代码每 1000 毫秒（1 秒）便使用 setInterval() 调用一次匿名函数：

```
setInterval(function() {trace("interval");}, 1000);
```

您可以使用命名函数来代替匿名函数。命名函数通常更易于阅读和理解（某些情况下例外，如回调函数）。您还可以预引命名函数，即在该函数尚未在时间轴上出现时就引用它。

您不能像使用命名函数时一样在代码的任意位置引用匿名函数（除非将其分配给变量）。例如，假定在 FLA 文件的第 5 帧上有匿名函数，如下所示：

```
// 在时间轴范围内有一个名为 my_mc 的影片剪辑
stop();
var myWidth = function () {
  trace(my_mc._width);
};
```

如果在第 1 帧上放置以下代码，它便无法引用该函数：

```
myWidth();
```

同样，下面的代码无论放置在哪一帧上，都无法正常工作：

```
myWidth();
var myWidth:Function = function () {
  trace(my_mc._width);
};
```

不过，以下代码则可以正常工作：

```
var myWidth:Function = function () {
  trace(my_mc._width);
};
myWidth();
```

注意 您还可以将 myWidth() 放置在任何包含 myWidth 函数的帧之后。

在定义命名函数时，在帧脚本中调用便可正常工作，不过，在同样的代码条件下使用匿名函数便无法正常工作：

```
// 下面的代码可正常工作，因为调用的是命名函数：
myWidth();
function myWidth() {
  trace("foo");
}

// 下面的代码不能正常工作，因为调用的是匿名函数：
myWidth();
var myWidth:Function = function () {
  trace("foo");
};
```

有关更多信息，请参见第 146 页的 “编写命名函数”。

注意

有关使用 “脚本助手”编写代码的信息，请参见第 270 页的 “使用 “脚本助手”编写 ActionScript”、第 272 页的 “使用 “脚本助手”创建 startDrag/stopDrag 事件”和 “ActionScript：使用脚本助手模式”教程 （该教程从第 194 页的 “打开起始文档”处开始）。

关于函数文本

函数文本 是一种您可以用表达式 （而不是语句）声明的未命名函数。在您需要临时使用一个函数，或者在您可以使用表达式代替函数时，函数文本非常有用。函数文本的语法是：

```
function (param1, param2, etc) {
  // 语句
};
```

例如，下面的代码就将函数文本作为表达式使用：

```
var yourName:String = "Ester";
setInterval(function() {trace(yourName);}, 200);
```

注意

在重新定义函数文本时，新的函数定义会替换旧的函数定义。

您可以将函数文本存储在变量中，以便在后面的代码中访问它。为此，您可以使用匿名函数。有关更多信息，请参见第 147 页的 “编写匿名函数和回调函数”。

关于构造函数

类的构造函数是一种特殊的函数，在使用 new 关键字创建类的实例时 （例如 var my_xml:XML = new XML();）会自动调用这种函数。构造函数的名称与包含它的类的名称相同。例如，您所创建的自定义 Person 类中可以包含以下构造函数：

```
public function Person(speed:Number) {
  Person.numPeople++;
  this._speed = speed;
}
```

然后使用下面的代码创建新实例：

```
var myPerson:Person = new Person();
```

注意 如果未在类文件中显式声明构造函数 （也就是说，如果未创建名称与类名匹配的函数），编译器将自动创建一个空的构造函数。

一个类只可以包含一个构造函数； ActionScript 2.0 中不允许重载构造函数。还有，构造函数不能具有返回类型。有关在类文件中编写构造函数的更多信息，请参见第 199 页的 “编写构造函数”。

定义全局函数和时间轴函数

在第 141 页的 “关于函数和方法”中，我们已讨论了 Flash 中提供的各种不同类型的函数。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和作用域中进行调用。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(myNum:Number):Number {
   return (myNum * 2) + 3;
};
trace(myFunction(5)) // 13
```

有关 _global 和作用域的信息，请参见第 297 页的 “关于变量和作用域”。

若要定义时间轴函数，请使用 function 语句，后跟函数名称、要传递给该函数的所有参数以及指明该函数动作的 ActionScript 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:Number):Number {
  return (Math.PI * radius * radius);
}
trace (areaOfCircle(8));
```

您还可以用多种其它方法定义函数。有关每种类型函数的更多信息，请参见以下各部分：

- 第 145 页的 “关于内置函数和顶级函数”
- 第 146 页的 “编写命名函数”
- 第 147 页的 “编写匿名函数和回调函数”
- 第 149 页的 “关于函数文本”
- 第 150 页的 “关于构造函数”
- 第 151 页的 “将用户定义的函数设定为目标并进行调用”

有关命名函数的信息，请参见第 152 页的 “函数命名”。有关在外部类文件中使用函数的详细示例，请参见第 152 页的 “在 Flash 中使用函数”和第 163 页的第 6 章 “类”。

注意

有关使用 “脚本助手”编写代码的信息，请参见第 270 页的 “使用 “脚本助手”编写 ActionScript”、第 272 页的 “使用 “脚本助手”创建 startDrag/stopDrag 事件”和 “ActionScript：使用脚本助手模式”教程 （该教程从第 194 页的 “打开起始文档”处开始）。

将用户定义的函数设定为目标并进行调用

用户定义的函数 是指用户自己创建的在应用程序中使用的函数，这与执行预定义函数的内置类中的函数相对。您可以自己命名函数，并在函数块内添加语句。前面部分介绍了如何编写各种函数，如命名函数、匿名函数和回调函数。有关命名函数的信息，请参见第 152 页的 “函数命名”，有关使用函数的信息，请参见第 152 页的 “在 Flash 中使用函数”。

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。若要调用函数，请输入函数名称的目标路径，并在必要时，在括号内传递所有必需的参数。用户定义函数有几种语法形式。下面的代码使用路径调用 `initialize()` 函数，该函数是在当前时间轴上定义的，而且不需要参数：

```
this.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：

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

有关编写命名函数的信息，请参见第 146 页的 “编写命名函数”。有关参数的更多信息，请参见第 156 页的 “将参数传递给函数”。

您还可以定义自己的命名函数。例如，下面的命名函数 `helloWorld()` 就是用户定义的：

```
function helloWorld() {
  trace("Hello world!");
};
```

下面的示例演示如何在 FLA 文件中使用用户定义的函数。

创建和调用简单的用户定义函数：

1. 创建一个新的 Flash 文档，并将它保存为 **udf.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
function traceHello(name:String):Void {
  trace("hello, " + name + "!");
}
traceHello("world"); // hello, world!
```

   上述代码创建了一个名为 `traceHello()` 的用户定义函数，该函数具有一个参数 `name`，并会输出一条问候消息。若要调用此用户定义的函数，可以从定义函数的同一时间轴调用 `traceHello`，并传入一个字符串值。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

有关命名函数的更多信息，请参见第 146 页的“编写命名函数”。类包含许多用户定义的函数。有关在类文件中编写函数的信息，请参见第 152 页的“在 Flash 中使用函数”。另请参见第 6 章“类”中的以下部分：第 179 页的“使用类文件中的方法和属性”、第 181 页的“关于公共、私有与静态方法和属性（成员）”和第 183 页的“关于类成员”。

函数命名

函数名称必须以小写字母开头。函数名称应描述该函数返回的值（如果有）。例如，如果函数要返回歌曲的标题，则可将其命名为 `getCurrentSong()`。

应为相似函数（在功能上相关的函数）的分组确立标准，因为 ActionScript 不允许重载。在面向对象编程 (OOP) 的上下文中，重载 指函数因传递给其的数据类型不同而使函数具有不同行为的能力。

和处理变量一样，您不能使用特殊的字符，而且方法名不能以数字开头。有关更多信息，请参见第 652 页的“命名约定”。有关方法命名的信息，请参见第 161 页的“方法命名”。

在 Flash 中使用函数

此部分为您演示如何在应用程序中使用函数。出于比较目的，下面的代码示例中有些代码示例使用驻留在 FLA 文件中的 ActionScript，而其它代码示例则将函数放到了类文件中。有关使用类文件中的函数的更多信息和示例，请参见第 163 页的第 6 章“类”。有关如何为类文件编写函数的详细信息和说明，请参见第 194 页的“示例：编写自定义类”。

为了减少必须的工作量并缩小 SWF 文件的大小，应尽可能重复使用代码块。一种重复使用代码的方法是多次调用一个函数，而不是每次都创建不同的代码。函数可以是一般代码片断；可以在一个 SWF 文件中使用相同的代码块达到稍有差别的多个目的。重复使用代码使您可以创建高效的应用程序，并可以将必须编写的 ActionScript 代码减小到最低限度，从而可以缩短开发时间。

您可以在 FLA 文件或类文件中创建函数，或者编写驻留在基于代码的组件中的 ActionScript 代码。下面的示例为您演示如何在时间轴上和类文件中创建函数。

提示	通过将代码包装进类文件或基于代码的组件，可以轻松地共享、分发或重复使用代码块。用户可以安装组件、将其拖动到舞台，还可以使用存储在文件中的代码，例如 Flash 中提供的基于代码的组件的工作流程（“窗口”>“公用库”>“类”）。

下面的示例为您演示如何在 FLA 文件中创建和调用函数。

若要在 FLA 文件中创建和调用函数，请执行以下操作：

1. 创建一个新的 Flash 文档，并将该文档保存为 **basicFunction.fla**。
2. 选择“窗口”>“动作”以打开“动作”面板。
3. 将下面的 ActionScript 代码输入到“脚本”窗格中：

```
function helloWorld(){
  // 语句位置
  trace("Hello world!");
};
```

   此 ActionScript 定义了名为 `helloWorld()` 的（用户定义的命名）函数。如果此时测试 SWF 文件，不会发生任何事情。例如，您不会在“输出”面板中看到 `trace` 语句。如想看到 `trace` 语句，必须调用 `helloWorld()` 函数。

4. 在该函数的后面键入下面以下 ActionScript 代码行：

```
helloWorld();
```

   此代码将调用 `helloWorld()` 函数。

5. 选择“控制”>“测试影片”对 FLA 文件进行测试。

   下面是在“输出”面板中显示的文本：`Hello world!`

有关将值（参数）传递给函数的信息，请参见第 156 页的“将参数传递给函数”。

有几种不同的方法可用来在主时间轴上编写函数。特别值得一提的是，可以使用命名函数和匿名函数。例如，在创建函数时可以使用下面的语法：

```
function myCircle(radius:Number):Number {
  return (Math.PI * radius * radius);
}
trace(myCircle(5));
```

匿名函数往往更加难读。请将下面的代码与上述代码加以比较。

```
var myCircle:Function = function(radius:Number):Number {
  // 函数块位置
  return (Math.PI * radius * radius);
};
trace(myCircle(5));
```

在使用 ActionScript 2.0 时，还可以如下例所示将函数放在类文件中：

```
class Circle {
  public function area(radius:Number):Number {
    return (Math.PI * Math.pow(radius, 2));
  }
  public function perimeter(radius:Number):Number {
    return (2 * Math.PI * radius);
  }
  public function diameter(radius:Number):Number {
    return (radius * 2);
  }
}
```

有关在类文件中编写函数的更多信息，请参见第 163 页的第 6 章 “类”。

正如您在上一代码范例中所看到的，不需要在时间轴上放置任何函数。下面的示例也将函数放在了类文件中。在使用 ActionScript 2.0 创建大型应用程序时最好采用这一做法，因为它允许您在若干应用程序中轻松重复使用代码。要想在其它应用程序中重复使用函数，可以导入现有的类（而不是从头重新编写代码），或者将函数直接复制到新的应用程序中。

在类文件中创建函数：

1. 创建一个新的 ActionScript 文档，并将该文档保存为 **Utils.as**。
2. 在 “脚本” 窗格中输入下面的 ActionScript：

```
class Utils {
  public static function randomRange(min:Number, max:Number):Number {
    if (min > max) {
      var temp:Number = min;
      min = max;
      max = temp;
    }
    return (Math.floor(Math.random() * (max - min + 1)) + min);
  }
  public static function arrayMin(num_array:Array):Number {
    if (num_array.length == 0){
      return Number.NaN;
    }
    num_array.sort(Array.NUMERIC | Array.DESCENDING);
    var min:Number = Number(num_array.pop());
    return min;
  }
  public static function arrayMax(num_array:Array):Number {
    if (num_array.length == 0){
      return undefined;
    }
    num_array.sort(Array.NUMERIC);
    var max:Number = Number(num_array.pop());
    return max;
  }
}
```

3. 选择 “文件” > “保存”以保存 ActionScript 文件。
4. 创建一个新的 Flash 文档，然后以文件名 **classFunctions.fla** 将其另存到 Utils.as 所在的文件夹中。
5. 选择 “窗口” > “动作”以打开 “动作”面板。
6. 在 “脚本”窗格中输入下面的 ActionScript：

```
var randomMonth:Number = Utils.randomRange(0, 11);
var min:Number = Utils.arrayMin([3, 3, 5, 34, 2, 1, 1, -3]);
var max:Number = Utils.arrayMax([3, 3, 5, 34, 2, 1, 1, -3]);
trace("month:" + randomMonth);
trace("min:" + min); // -3
trace("max:" + max); // 34
```

7. 选择“控制”>“测试影片”对这些文档进行测试。下面是在“输出”面板中显示的文本：

```
month: 7
min: -3
max: 34
```

提示 有关使用 “脚本助手”编写代码的信息，请参见第 270 页的 “使用 “脚本助手”编写 ActionScript”、第 272 页的 “使用 “脚本助手”创建 startDrag/stopDrag 事件”和 “ActionScript：使用脚本助手模式”教程 （该教程从第 194 页的 “打开起始文档”处开始）。

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，它可以隐藏其变量以不被 SWF 文件中的所有其它脚本看到；本地变量的调用范围限定为函数体，在该函数退出后，本地变量也将不再存在。 Flash 也将任何传递给函数的参数视为本地变量。

提示 您还可以在函数中使用常规变量。但是，如果要修改常规变量，最好使用脚本注释记录这些修改。

在函数中使用变量：

1. 创建一个新的 Flash 文档，并保存为 **flashvariables.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var myName:String = "Ester";
var myAge:String = "65";
var myFavSoftware:String = "Flash";
function traceMe(yourFavSoftware:String, yourName:String, yourAge:String)
  {
  trace("I'm " + yourName + ", I like " + yourFavSoftware + ", and I'm "
  + yourAge + ".");
}
traceMe(myFavSoftware, myName, myAge);
```

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

有关传递参数的更多信息，请参见第 156 页的 “将参数传递给函数”。有关变量和数据的更多信息，请参见第 275 页的第 10 章 “数据和数据类型”。

将参数传递给函数

参数，也叫参量，是函数对其执行代码的元素。（在本书中，术语参数 和参量 是可以互换的。）可以将参数 （值）传递给函数。然后可以使用这些参数来处理该函数。可以在函数块（函数内的语句）中使用这些值。

有时参数是必需的，而有时又是可选的。在单个函数中甚至可以同时拥有必需参数和可选参数。如果没有给函数传递足够的参数， Flash 便会将缺少的参数值设置为 `undefined`，这可能会在 SWF 文件中引起意想不到的结果。

下面名为 `myFunc()` 的函数带有参数 `someText`:

```
function myFunc(someText:String):Void {
  trace(someText);
}
```

传递该参数之后，在调用函数时可以将一个值传递给函数。此值显示在 “输出” 面板中，如下所示:

```
myFunc("This is what traces");
```

在调用函数时，应始终传递指定数量的参数，除非函数能够检查未定义的值，并设置相应的默认值。函数用传递的值替换掉函数定义中的参数；如果缺少某些参数， Flash 便将其值设置为 `undefined`。在编写 ActionScript 代码时，会经常把参数传递给函数，

还可以将多个参数传递给一个函数，如下面所示，这一点很容易做到:

```
var birthday:Date = new Date(1901, 2, 3);
trace(birthday);
```

每个参数之间都用逗号分隔符分开。ActionScript 语言中的许多内置函数都有多个参数。例如，MovieClip 类的 `startDrag()` 方法有五个参数，`lockCenter`、`left`、`top`、`right` 和 `bottom`:

```
startDrag(lockCenter:Boolean, left:Number, top:Number, right:Number,
  bottom:Number):Void
```

将参数传递给函数：

1. 创建一个新的 Flash 文档，并将该文档保存为 **parameters.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
function traceMe(yourMessage:String):Void {
  trace(yourMessage);
}
traceMe("How are you doing?");
```

   开始几行代码创建了一个名为 `traceMe()` 的用户定义函数，该函数带有一个参数 `yourMessage`。最后一行代码调用 `traceMe()` 函数，并将字符串值 `ow are you doing?"` 传递给函数。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

下一示例演示如何将多个参数传递给一个函数。

若要将多个参数传递给一个函数，请执行以下操作：

1. 创建一个新的 Flash 文档，并将该文档保存为 **functionTest.fla**。
2. 将下面的代码添加到主时间轴中的第 1 帧：

```
function getArea(width:Number, height:Number):Number {
     return width * height;
}
```

   `getArea` 函数使用两个参数，`width` 和 `height`。

3. 在函数的后面输入下面的代码：

```
var area:Number = getArea(10, 12);
trace(area); // 120
```

   `getArea()` 函数调用将把值 10 和 12 分别赋给 width 和 height，您可以将返回值保存在 `area` 实例中。然后您可以输出那些保存在 `area` 实例中的值。

4. 选择“控制”>“测试影片”对 SWF 文件进行测试。

   您可以在“输出”面板中看到 `120`。

   函数 `getArea()` 中的参数与本地变量中的值类似；它们在调用该函数时存在，在该函数退出后它们将不再存在。

在下一示例中，如果没有向 `addNumbers()` 函数传递足够多的参数，ActionScript 将返回 `NaN` 值（非数字）。

向函数传递可变数量的参数：

1. 创建一个新的 Flash 文档，并将该文档保存为 **functionTest2.fla**。
2. 将下面的代码添加到主时间轴中的第 1 帧：

```
function addNumbers(a:Number, b:Number, c:Number):Number {
  return (a + b + c);
}
trace(addNumbers(1, 4, 6)); // 11
trace(addNumbers(1, 4)); // NaN（不是数字），c 等于 undefined
trace(addNumbers(1, 4, 6, 8)); // 11
```

   如果没有向 addNumbers 函数传递足够多的参数，缺少的参量便会被赋予默认值 undefined。如果传递的参数过多，将会忽略多余的参数。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

   Flash 将显示下面的值：11、NaN、11。

从函数中返回值

使用 return 语句可以从函数中返回值。return 语句指定函数返回的值。对于在其中执行表达式的函数，return 语句返回计算结果作为函数值。return 语句会将其结果立即返回给发出调用的代码。

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 return 语句。

在函数中使用 return 语句时应遵守以下规则：

- 如果为函数指定 Void 之外的其它返回类型，则必须在函数中加入一条 return 语句，并且其后必须跟返回的值。
- 如果将返回类型指定为 Void，则不必加入 return 语句，但是如果需要加入，则其后不可以跟任何值。
- 无论返回类型如何，都可以使用 return 语句从函数中间退出。
- 如果不指定返回类型，则可以选择是否加入 return 语句。

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

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

某些函数会执行一系列的任务，但不返回值。下一示例返回了处理后的值。您将在变量中捕获该值，然后可以在您的应用程序内使用该变量。

若要返回值并在变量中捕获该值，请执行以下操作：

1. 创建一个新的 Flash 文档，并将它保存为 **return.fla**。
2. 将下面的代码添加到主时间轴中的第 1 帧：

```
function getArea(width:Number, height:Number):Number {
     return width * height;
}
```

   getArea() 函数使用两个参数：width 和 height。

3. 在函数的后面输入下面的代码：

```
var area:Number = getArea(10, 12);
trace(area); // 120
```

   getArea() 函数调用将把值 10 和 12 分别赋给 width 和 height，您可以将返回值保存在 area 实例中。然后您可以输出那些保存在 area 实例中的值。

4. 选择 “控制” > “测试影片” 对 SWF 文件进行测试。

   可以在 “输出” 面板上看到值 120。

   函数 getArea() 中的参数与本地变量中的值类似；它们在调用该函数时存在，在该函数退出后它们将不再存在。

关于嵌套函数

可以从另外一个函数内调用函数。这让您可以嵌套函数，以便利用其在 Flash 中执行特定的任务。

例如，您可以在时间轴上嵌套函数，以对一个字符串执行一些特定的任务。将下面的代码输入到时间轴中的第 1 帧上：

```
var myStr:String = "My marshmallow chicken is yellow.";
trace("Original string:" + myStr);
function formatText():Void {
  changeString("Put chicken in microwave.");
  trace("Changed string:" + myStr);
}
function changeString(newtext:String):Void {
  myStr = newtext;
}
// 调用函数。
formatText();
```

选择 “控制” > “测试影片” 对嵌套函数进行测试。在调用 formatText() 函数时，formatText() 和 changeString() 函数都对该字符串执行操作。

了解方法

方法是与类关联的函数。类可以是自定义类，也可以是作为 ActionScript 语言一部分的内置类。有关方法与函数的比较信息，请参见第 141 页的“关于函数和方法”和第 143 页的“关于方法和函数的类型”。

例如，`sortOn()` 是一个与 Array 类关联的内置方法（`sortOn` 是 Flash 中内置的预定义 Array 类的一个函数）。

在 FLA 文件中使用 sortOn() 方法：

1. 创建一个新的 Flash 文档，并保存为 **methods.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
var userArr:Array = new Array();
userArr.push({firstname:"George", age:39});
userArr.push({firstname:"Dan", age:43});
userArr.push({firstname:"Socks", age:2});
userArr.sortOn("firstname");
var userArrayLenth:Number = userArr.length;
var i:Number;
for (i = 0; i < userArrayLenth; i++) {
  trace(userArr[i].firstname);
}
```

   您使用 Array 类的 `sortOn()` 方法创建了一个名为 `userArr` 的新 Array 对象。该数组将由包含名字和年龄的三个对象填充，然后该数组再根据每个对象的 `firstname` 属性值排序。最后，对该数组中的每一项执行循环，在“输出”面板中显示名字，并按照名字的第一个字母按字母表顺序对名字进行排序。

3. 选择“控制”>“测试影片”对 SWF 文件进行测试。

   此代码将以下信息显示在“输出”面板中：

```
Dan
George
Socks
```

如第 146 页的“编写命名函数”中所演示的，当在时间轴的第 1 帧上编写下面的代码时，您的 ActionScript 代码将定义一个名为 `eatCabbage()` 的函数。

```
function eatCabbage() {
    trace("tastes bad");
}
eatCabbage();
```

但是，如果在类文件中编写 `eatCabbage()` 函数，而在 FLA 文件中调用 `eatCabbage()`，便可将 `eatCabbage()` 视为一个方法。

下一示例为您演示如何在类中创建方法。

若要比较方法和函数，请执行以下操作：

1. 创建一个新的 ActionScript 文件，然后选择 “文件” > “另存为”，并将该文件保存为 **EatingHabits.as**。
2. 在 “脚本” 窗口中输入以下 ActionScript 代码：

```
class EatingHabits {
  public function eatCabbage():Void {
    trace("tastes bad");
  }
}
```

3. Save your changes to EatingHabits.as.
4. 创建一个新的 Flash 文档，选择 “文件” > “另存为”，将其命名为 **methodTest.fla**，并将此文件保存在 EatingHabits.as 所在的同一目录中。
5. 将下面的 ActionScript 代码输入到时间轴的第 1 帧上：

```
var myHabits:EatingHabits = new EatingHabits();
myHabits.eatCabbage();
```

在使用此 ActionScript 时，将调用 EatingHabits 类的 `eatCabbage()` 方法。

注意	当您使用任何内置类的方法 （除了在本过程前面步骤中编写的自定义类之外）时，其实是在时间轴上使用方法。

6. 在前面的一行 ActionScript 之后添加以下代码：

```
function eatCarrots():Void {
    trace("tastes good");
}
eatCarrots();
```

在此代码中，编写并调用 `eatCarrots()` 函数。

7. 选择 “控制” > “测试影片” 对 SWF 文件进行测试。

方法命名

应使用动词给方法命名，用混合大小写标明连接在一起的单词，并确保第一个字母为小写。例如，可以用以下方式来命名方法：

```
sing();
boogie();
singLoud();
danceFast();
```

大多数方法名称都用动词来表示，因为方法对对象执行操作。和处理变量一样，您不能使用特殊的字符，方法名也不能以数字开头。有关更多信息，请参见第 652 页的 “命名约定”。

第 6 章

类

6

本章介绍如何在 ActionScript 2.0 中使用和编写类。类是 ActionScript 2.0 的主要部分，类在 ActionScript 2.0 中的作用比它在较早版本的 Macromedia Flash 中更为重要。在本章中，您将了解到类在 Flash 中的重要性。

本章的开头部分解释了一些基本术语，以及这些术语与类和面向对象的编程 (OOP) 的关系。接下来将指导您完成一个范例类文件，您将理解该类文件的各部分的工作原理以及类的组织方式。本章后面的部分介绍了如何创建您自己的自定义类，以及如何在 Flash 文档中使用它们。您还将了解 Flash 的类路径，以及应如何为类添加注释以便让阅读或使用您的代码的其他人可以很容易地理解代码和类的总体用途。

本节包含一些代码范例，您可以使用这些代码范例来熟悉如何在 ActionScript 2.0 中创建类。学完本章后您应能够编写典型的类文件，理解和认识 Flash 类，并且能够轻松地阅读其他人的类文件。

如果您不熟悉 ActionScript 2.0 的脚本撰写，请参见第 63 页的第 4 章“语法和语言基础知识”和第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。

有关使用自定义类和内置类的更多信息，请参见以下主题：

关于面向对象的编程和 Flash

ActionScript 2.0 是一种面向对象的语言。与 ActionScript 一样，OOP 语言也是基于 类和实例 的概念。一个类定义了可区分一系列对象的所有属性。例如，User 类代表正在使用您的应用程序的一组用户。然后，您将获得该类的实例化，对于 User 类而言，该实例化是其中一个用户，其中一个成员。该实例化将产生 User 类的一个实例，该实例具有 User 类的所有属性。

还可以将类看作数据类型 或模板，可以创建类来定义新对象类型。例如，如果应用程序中需要 Lettuce 数据类型，就可以编写 Lettuce 类。这将定义 Lettuce 对象，然后可以指定 Lettuce 方法 (`wash()`) 和属性 (`leafy` 或 `bugs`)。为定义类，需要在一个外部脚本文件中使用 `class` 关键字。您可以在 Flash 创作工具中通过选择“文件”>“新建”再选择“ActionScript 文件”来创建外部脚本文件。

Flash Basic 8 和 Flash Professional 8 都提供了 Flash Player 8，它为 ActionScript 语言添加了一些新功能，如滤镜效果、文件上载和下载以及外部 API。与以往一样，ActionScript 2.0 提供了几个在其它编程语言（如 Java）中也使用的功能强大而且大家都熟悉的 OOP 概念和关键字（如类、接口和包）。此该编程语言使您能够构建可重复使用、可扩展、性能稳定并且便于维护的程序结构。它还通过为用户提供全程的编码辅助和调试信息缩短了开发时间。您可以使用 ActionScript 2.0 创建对象并建立继承，创建自定义类，以及扩展 Flash 顶级类和内置类。在本章中，您将学习如何创建类以及如何使用自定义类。

Flash Basic 8 和 Flash Professional 8 包含大约 65 个顶级类和内置类，它们提供了基本（或者称“原始”）数据类型（Array、Boolean、Date 等）、自定义错误和事件，以及几种加载外部内容（XML、图像、原始二进制数据等）的方法。您还可以编写自己的自定义类并将它们集成到您的 Flash 文档中，甚至还可以扩展顶级类并增加自己的功能或修改现有功能。例如，在本章的第 183 页的“关于类成员”中将演示如何创建自定义 Person 类，该类中包括人员的姓名和年龄等自定义属性。然后可以在文档中将此自定义类视为一种新的数据类型，并使用 `new` 运算符创建该类的新实例。

有关使用 OOP（面向对象的编程）的更多信息，请参见以下主题：

- 第 165 页的“使用类的好处”
- 第 165 页的“关于包”
- 第 168 页的“关于值和数据类型”
- 第 168 页的“面向对象的编程基础知识”

使用类的好处

在 OOP（面向对象的编程）中，类定义一类对象。类描述对象的属性（数据）和方法（行为），这与描述建筑物特性的建筑蓝图非常相似。您可以在外部 ActionScript (AS) 文件中编写自定义类，并且可以在编译 FLA 文件时将其导入到应用程序中。

在构建较大的 Flash 应用程序时，类会非常有用，因为您可以将应用程序的大量复杂内容组织到外部类文件中。通过将大量逻辑添加到自定义类中，不仅能使代码更易于重复使用，而且还可以对 ActionScript 代码的其它部分 “隐藏”一些方法和属性。这可以帮助防止别人访问敏感信息或更改不应更改的数据。

在使用类时，还可以扩展现有的类，添加新的功能或修改现有的功能。例如，如果要创建三个非常相似的类，可以编写一个基类，然后扩展此基类来编写另外两个类。这两个类可以加入更多方法和属性，所以您不必创建三个全部复制相同代码和逻辑的类文件。

使用类的另一个优点是代码可以重复使用。例如，如果您创建一个自定义类，而该自定义类使用 Drawing 应用程序编程接口 (API) 创建了一个自定义的进度条，您就可以将该进度条类保存在您的类路径中，并通过导入该自定义类在您所有的 Flash 文档中重复使用相同代码。有关设置类路径的更多信息，请参见第 174 页的“关于导入类文件”和第 175 页的“关于设置和修改类路径”。

关于包

创建类时，可将 ActionScript 类文件组织到包 中。包是一个位于指定的类路径目录下的目录，其中包含一个或多个类文件。（请参见第 174 页的 “关于导入类文件”和第 175 页的“关于设置和修改类路径”）。包也可以包含其它的包（称为子包），每个子包都可以具有自己的类文件。

与变量一样，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 ($)，并且后面的每个字符可以是字母、数字、下划线或美元符号。包命名有一些首选方法，例如，建议避免使用下划线或美元符号字符。有关包命名的更多信息，请参见第 660 页的 “命名包”。

包通常用于将相关的类整理在一起。例如，您可能有三个分别在 Square.as、Circle.as 和 Triangle.as 中定义的相关类 Square、Circle 和 Triangle。假定您已经将这些 ActionScript 文件保存到了类路径中指定的某个目录下，如下面的示例所示：

```
// 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 ActionScript 文件放在了 Shapes 目录下，则需要更改各个类文件的名称，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，可以指定其全限定类名，或使用 `import` 语句导入该包。有关更多信息，请参见第 166 页的 “使用包”。

类和包的比较

在 OOP（面向对象的编程）中，类定义一类对象。类实际上是一些数据类型，可以用来在应用程序中定义新的对象类型。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。类的属性（在类内定义的变量）和方法统称为类的成员。若要使用由类定义的属性和方法，通常需要先创建该类的一个实例，具有所有静态成员（请参见第 225 页的 “关于类（静态）成员”，如顶级 Math 类，和第 182 页的 “静态方法和属性”）的类除外。实例与其类之间的关系类似于房子与其蓝图之间的关系。

Flash 中的包是一些目录，它们包含一个或多个类文件并位于指定的文件路径中。可以将相关的自定义类文件放进单个目录内。例如，您可能有名为 SteelWidget、PlasticWidget 和 WoodWidget 的三个相关的类，它们分别是在 SteelWidget.as、PlasticWidget.as 和 WoodWidget.as 中定义的，则可以将这些类组织到 Widget 包中。有关包的更多信息，请参见第 166 页的 “使用包”和第 197 页的 “创建和打包类文件”。

使用包

包是位于指定的类路径目录下并且包含一个或多个类文件的目录。例如，flash.filters 包是硬盘上的一个目录，其中包含了若干类文件，供 Flash 8 中的每个滤镜类型（如 BevelFilter、BlurFilter、DropShadowFilter 等）类使用。

注释	若要使用 `import` 语句，必须在 FLA 文件 “发布设置”对话框的 “Flash”选项卡中指定 ActionScript 2.0 和 Flash Player 6 或更高版本。

通过 import 语句访问类时无需指定其完全限定名称。例如，如果要在脚本中使用 BlurFilter 类，您必须通过其完全限定名称 (flash.filters.BlurFilter) 引用该类或将其导入；如果导入该类，就可以通过其类名 (BlurFilter) 引用它。下面的 ActionScript 代码演示使用 import 语句与使用完全限定类名之间的差别。

如果不导入 BlurFilter 类，则代码需要使用完全限定类名（后跟类名的包名）以便使用滤镜:

```
// 不导入
var myBlur:flash.filters.BlurFilter = new flash.filters.BlurFilter(10, 10,
  3);
```

通过用 import 语句编写的相同代码，只使用类名就可以访问 BlurFilter，而通常不必使用完全限定名称。这样可以减少键入量并降低出现键入错误的机率:

```
// 使用导入
import flash.filters.BlurFilter;
var myBlur:BlurFilter = new BlurFilter(10, 10, 3);
```

如果要导入一个包中的若干类（如 BlurFilter、DropShadowFilter 和 GlowFilter），可使用两种导入方法之一导入每个类。导入多个类的第一种方法是使用单独的 import 语句导入每个类，如下面的代码段中所示:

```
import flash.filters.BlurFilter;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
```

对包中的每个类使用单独的 import 语句既耗时又容易发生键入错误。导入包中的类的第二种方法是使用通配符导入，以导入包中某级别的所有类。下面的 ActionScript 演示使用通配符导入的示例:

```
import flash.filters.*; // 导入 flash.filters 包中的每个类
```

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。在该帧上，可以通过类名引用该包中的类，而不用使用其完全限定名称。但是，如果要在另一个帧脚本上使用该类名，则需要通过完全限定名称引用该包中的类，或向这个另外的帧添加 import 语句以导入该包中的类。

使用 import 语句时，还一定要注意，只能在指定级别导入类。例如，导入 mx.transitions 包中的所有类时，只会导入 /transitions/ 目录中的那些类，而不会导入子目录中的所有类（如 mx.transitions.easing 包中的类）。

提示

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可以不必担心 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。

关于值和数据类型

在开始编写类并使用这些类时，数据、值和类型非常重要。有关数据和类型的介绍，请参见第 275 页的第 10 章 “数据和数据类型”。在使用类时要注意，数据类型描述变量或 ActionScript 元素可以包含的信息类型，如 Boolean、Number 和 String。有关更多信息，请参见第 276 页的 “关于数据类型”。

表达式具有值，而值和属性具有类型。对于类中的一个属性，您为其设置的值和期望从中得到的值必须与该属性兼容。类型兼容性是指值的类型与正在使用的类型兼容，如下例所示：

```
var myNum:Number = 10;
```

有关严格数据类型指定的更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。

面向对象的编程基础知识

在以下各部分中，您将首先查看本章中使用的一些术语，然后再开始编写 ActionScript 代码。首先将对开发面向对象的程序需要遵循的原则进行简要介绍，这将有助于您学习本章中的示例和各节以及本书后面的内容。本章后面的内容将对这些原则进行更加深入的介绍，并将提供有关如何在 Flash 8 中实现这些原则的详细信息。

下面几节将以猫为例进行类比，将猫与 OOP （面向对象的编程）概念进行比较。

对象

考虑现实世界中的一个对象，例如一只猫。我们可以说猫具有许多属性 （或状态），例如猫名、猫龄和颜色；猫还具有各种行为，例如睡觉、吃食和发出叫声。在 OOP 的世界里，对象也具有许多属性和行为。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。

术语	这里行为一词代表其广泛的意义，而不是指 Flash 创作环境中的 “行为”面板。

有关对象的更多信息，请参见第 282 页的 “Object 数据类型”。

实例和类成员

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，它们吃食和发出叫声的方式也可能不同。但是不管它们具有怎样的个体差异，所有的猫都是同一个类别的成员；或者，就 OOP 术语而言，它们属于同一个类：猫类。在 OOP （面向对象的编程）术语中，每只猫都是 Cat 类中的一个实例。

同样，在 OOP（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总称为该类的成员。这些特性（在猫的示例中，特性包括猫名、猫龄和颜色）称为类的属性，用变量表示；行为（玩、睡觉）称为类的方法，用函数表示。

有关实例和类成员的更多信息，请参见第 183 页的 “关于类成员”和第 186 页的 “使用类成员”。

继承

面向对象编程（OOP）的主要优点之一就是可以创建（或扩展）类的子类；子类可以继承类的所有属性和方法。子类通常会定义其它方法和属性或重写超类中定义的方法或属性。子类还可以重写（为其提供自己的定义）在超类中定义的方法。

使用超类 / 子类结构的最大优点之一是，更便于在各种不同的类之间重复使用相似的代码。例如，可以构建一个名为 Animal 的超类，其中包含所有动物的共有特性和行为。接下来可以构建几个继承自 Animal 超类的子类，并添加特定于某类动物的特性和行为。

您可以创建一个继承自另一类的 Cat 类。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 子类。另一个子类（比如 Siamese 类）可以再次扩展（子类）Cat 类，依此类推。

通过编写子类可以重用代码。您不必重新创建两个类共有的所有代码，而只需对现有类加以扩展即可。

提示	在复杂的应用程序中，确定类的层次结构是设计过程中的重要部分。在开始编程之前一定要确保已经确定了这一层次结构。

有关继承和子类的更多信息，请参见第 229 页的第 7 章 “继承”。

接口

在 OOP（面向对象的编程）中，接口 可以描述为类定义的模板，需要使用实现接口的类以实现该方法模板。在猫的示例中，接口类似于猫的蓝图：通过蓝图可了解需要的部分，但并不一定提供关于这些部分的组装方法或工作原理的信息。

可以使用接口向应用程序添加结构和易维护性。由于 ActionScript 2.0 仅支持从单个超类进行扩展，因此您可以将接口以受限多次继承的形式使用。

也可将接口看作是用于将两个若没有接口便没有任何关系的类关联起来的 “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）。设计应用程序时，约定不同的类使用一组方法进行通信。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。任何实现此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。

有关继承的更多信息，请参见第 229 页的第 7 章 “继承”。有关接口的更多信息，请参见第 241 页的第 8 章 “接口”。

封装

在完美的面向对象的设计中，对象被看作包含（或封装）功能的“黑匣子”。程序员应当能够在仅知道对象的属性、方法和事件（对象的编程接口）的情况下与对象进行交互，而不需知道其实现的详细信息。此方法使程序员可以在更高的抽象层次上思考，并能提供可用于构建复杂系统的组织框架。

封装是 ActionScript 2.0 之所以包含诸如成员访问控制等功能的原因，这样实现的详细信息对于对象外的代码是私有的和不可见的。对象外代码将被强制与对象的编程接口交互，而不是与实现详细信息（可隐藏在私有方法和属性中）交互。这种方法提供了一些重要优点；例如，只要编程接口不变，对象的创建者就可以在不对对象外代码做任何更改的情况下更改对象的实现。

有关封装的更多信息，请参见第 193 页的“关于使用封装”。

多态

OOP 允许使用一种名为多态 的技术来表达单个类之间的差异，使用这种技术，类可以重写其超类的方法并定义这些方法的专用实现。在 Flash 中，子类可以定义方法（从其超类继承）的专用实现，但不能访问其超类的实现，这一点与其它编程语言相同。

例如，您可以从具有 `play()` 和 `sleep()` 方法、名为 Mammal 的类开始。然后您可以创建 Cat、Monkey 和 Dog 子类来扩展 Mammal 类。这些子类重写 Mammal 类的 `play()` 方法，来反映那些特定种类的动物的习性。Monkey 实现悬挂在树上的 `play()` 方法； Cat 实现对线球猛扑的 `play()` 方法； Dog 实现捡回球的 `play()` 方法。因为动物的 `sleep()` 功能相似，所以可以使用超类实现。

有关多态的更多信息，请参见第 229 页的第 7 章“继承”和第 236 页的“在应用程序中使用多态”。

编写自定义类文件

下面的示例检查类文件的各个部分。您将学习如何编写类，以及如何修改类以扩展可以在 Flash 中使用类的方法。您还将了解类的各个部分和如何导入它们，并了解有关在 Flash 中使用自定义类文件的相关信息。

您将从一个非常简单的类入手。下面的示例显示了一个名为 UserClass 的简单类的组织情况：

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。类结构还与接口文件相关。下图显示了此结构，可以按照下图中所示结构创建类。

- 类文件是以文档注释开头的，其中包含对代码的基本描述以及作者信息和版本信息。
- 添加 `import` 语句（如果适用）。

- 编写包语句、类声明或接口声明，如下所示：
  ```
  class UserClass {...}
  ```
- 包括任何必要的类或接口实现注释。在这些注释中，添加与整个类或接口有关的信息。
- 添加所有静态变量。先编写公共类变量，接着编写私有类变量。
- 添加实例变量。先编写公共成员变量，接着编写私有成员变量。
- 添加构造函数语句，如下面示例中的语句：
  ```
  public function UserClass(username:String, password:String) {...}
  ```
- 编写您的方法。要按功能对方法进行分组，而不要按可访问性或作用范围分组。使用这种方式的组织方法有助于提高代码的可读性和清晰度。
- 在类文件中编写 getter/setter 方法。

下面的示例介绍了一个名为 User 的简单的 ActionScript 类。

若要创建类文件，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 选择 “文件” > “另存为”，将新文件命名为 **User.as**。
3. 在 “脚本”窗口中键入以下 ActionScript 代码：

```
/**
  User 类
  作者: John Doe
  版本: 0.8
  修改时间: 08/21/2005
  版权所有: Macromedia, Inc.

  此代码定义一个自定义 User 类，以允许您创建新用户并指定用户登录信息。
*/

class User {
  // 私有实例变量
  private var __username:String;
  private var __password:String;

  // 构造函数语句
  public function User(p_username:String, p_password:String) {
    this.__username = p_username;
    this.__password = p_password;
  }

  public function get username():String {
    return this.__username;
  }
  public function set username(value:String):Void {
    this.__username = value;
  }
```

```
    public function get password():String {
      return this.__password;
    }
    public function set password(value:String):Void {
      this.__password = value;
    }
}
```

4. 保存对类文件所做的更改。

   上面的代码片断以标准的文档注释 开头，指定了类名称、作者、版本、最后一次修改类的日期、版权信息以及描述类功能的简单说明。

   User 类的构造函数语句采用以下两个参数：p_username 和 p_password，它们将被复制到类的私有实例变量 __username 和 __password 中。类中的剩余部分代码定义私有实例变量的 getter 和 setter 属性。如果要创建只读属性，那么就需要定义 getter 函数，而不是 setter 函数。例如，如果要确保用户名在定义之后不会被更改，则需要删除 User 类文件中的 username setter 函数。

5. 选择 “文件” > “新建”，然后选择 “Flash 文档”。

6. 选择 “文件” > “另存为”，将文件命名为 **user_test.fla**。将文件保存到 User.as 所在的同一目录中。

7. 在时间轴的第 1 帧上键入下面的 ActionScript：

```
import User;
var user1:User = new User("un1", "pw1");
trace("Before:");
trace("\t username = " + user1.username); // un1
trace("\t password = " + user1.password); // pw1
user1.username = "1nu";
user1.password = "1wp";
trace("After:");
trace("\t username = " + user1.username); // 1nu
trace("\t password = " + user1.password); // 1wp
```

   由于前面创建的 User 类是最基本的类，所以 Flash 文档中的 ActionScript 也非常简单易懂。第一行代码将自定义的 User 类导入到您的 Flash 文档中。通过导入 User 类，可将该类用作自定义的数据类型。

   定义 User 类的单个实例并将它分配给名为 user1 的变量。为 user1 User 对象指定一个值，并定义 un1 这样一个 username 和 pw1 这样一个 password。下面两个 trace 语句使用 User 类的 getter 函数显示 user1.username 和 user1.password 的当前值，两者均返回字符串。接下来的两行代码使用 User 类的 setter 函数为 username 和 password 变量设置新值。最后，将 username 和 password 的值显示到 “输出” 面板中。trace 语句将显示使用 setter 函数设置的、修改过的值。

8. 保存 FLA 文件，然后选择 “控制” > “测试影片” 来测试文件。

   可在 “输出” 面板中看到 `trace` 语句的结果。在接下来的示例中，您将在应用程序中使用这些文件。

硬盘上的一个范例文件演示了如何使用 XML 数据和自定义类文件创建动态菜单。该范例调用 ActionScript `XmlMenu()` 构造函数并向其传递两个参数: XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分在自定义类文件 XmlMenu.as 中。

从硬盘上的 Samples 文件夹中可以找到该范例源文件 xmlmenu.fla。

- 在 Windows 上，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

关于在应用程序中使用自定义类

在第 170 页的 “编写自定义类文件”中，您已创建了一个自定义类文件。在以下各节中，您将在应用程序中使用该类文件。创建类的工作流程至少包括以下步骤：

1. 在外部 ActionScript 类文件中定义一个类。有关定义和编写类文件的信息，请参见第 170 页的 “编写自定义类文件”。
2. 将类文件保存到指定的类路径目录 （Flash 查找类的位置）中或保存在应用程序的 FLA 文件所在的同一目录中。有关设置类路径的更多信息，请参见第 175 页的 “关于设置和修改类路径”。若要进行比较和获取有关导入类文件的更多信息，请参见第 174 页的 “关于导入类文件”。
3. 在另一个脚本中创建类的一个实例 （既可以是在一个 FLA 文档中，也可以是在一个外部脚本文件中），或通过创建基于原始类的一个子类。有关创建类的实例的更多信息，请参见第 208 页的 “在示例中创建类的实例”。

本章的以下部分包含一些示例代码，您可以使用这些示例代码来熟悉如何在 ActionScript 2.0 中创建类。如果您对 ActionScript 2.0 不熟悉，请阅读第 275 页的第 10 章 “数据和数据类型”和第 63 页的第 4 章 “语法和语言基础知识”。

有关使用自定义类的更多信息，请参见以下主题：

- 第 174 页的 “关于导入类文件”
- 第 178 页的 “在 Flash 中使用类文件”
- 第 179 页的 “使用类文件中的方法和属性”
- 第 183 页的 “关于类成员”
- 第 187 页的 “关于 getter 和 setter 方法”
- 第 178 页的 “编译器如何解析类引用”

- 第 190 页的 “关于动态类”
- 第 193 页的 “关于使用封装”
- 第 194 页的 “关于在类中使用 this 关键字”

硬盘上的一个范例文件演示了如何使用 XML 数据和自定义类文件创建动态菜单。该范例调用 ActionScript `XmlMenu()` 构造函数并向其传递两个参数: XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分在自定义类文件 XmlMenu.as 中。

在硬盘上的 Samples 文件夹中可以找到该范例源文件 xmlmenu.fla。

- 在 Windows 上，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

关于导入类文件

为了使用已定义的类或接口， Flash 必须找到包含该类或接口定义的外部 ActionScript 文件，以便导入该文件。 Flash 在其中搜索类、接口、函数和变量定义的那一组目录称为类路径。 Flash 有两个类路径设置，即全局类路径和文档级类路径：

- **全局类路径**是由所有 Flash 文档共享的类路径。全局类路径可在“首选参数”对话框（“编辑”>“首选参数”(Windows) 或 “Flash”>“首选参数”(Macintosh)，在 “类别”列表中单击 “ActionScript”，然后单击 “ActionScript 2.0 设置”）中设置。
- **文档级类路径**是专门为单个 Flash 文档定义的类路径。文档级类路径可在 “发布设置”对话框（“文件”>“发布设置”，选择“Flash”选项卡，然后单击“设置”按钮）中设置。

在导入类文件时，必须遵守以下规则：

- `import` 语句可以存在于以下位置：
  - 类文件中类定义之前的任何位置
  - 帧或对象脚本中的任何位置
  - 包含在应用程序中 （使用 `#include` 语句）的 ActionScript 文件中的任何位置。
- 可以使用此语法导入单个打包的定义：
  ```
  import flash.display.BitmapData;
  ```
- 可以使用通配符语法导入整个包：
  ```
  import flash.display.*;
  ```

还可以通过使用 include 语句在 Flash 文档 (FLA) 文件中包含 ActionScript 代码。以下规则适用于 include 语句：

- include 语句实际上就是复制和粘贴包含的 ActionScript 文件中的内容。
- ActionScript 类文件中的 include 语句与包含该文件的子目录有关。
- FLA 文件中的 include 语句只可以引入在 FLA 文件内有效的代码，在存在 include 语句的其它地方也是如此。例如，如果在类定义内有一个 include 语句，则只有属性和方法定义可以存在于包含的 ActionScript 文件中：

```
// Foo.as
class Foo {
  #include "FooDef.as"
}

// FooDef.as:
var fooProp;
function fooMethod() {}
trace("Foo"); // 类定义中不允许使用此语句。
```

有关 include 语句的更多信息，请参见《ActionScript 2.0 语言参考》中的 #include 指令。有关类路径的更多信息，请参见第 175 页的 “关于设置和修改类路径”。

关于设置和修改类路径

为了使用已定义的类或接口， Flash 必须找到包含该类或接口定义的外部 ActionScript 文件。 Flash 在其中搜索类和接口定义的目录列表被称为类路径。

创建 ActionScript 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，以加入所需的目录路径）。否则， Flash 将无法解析（亦即找到）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。（有关包的更多信息，请参见第 197 页的 “创建和打包类文件”。）

Flash 有两个类路径设置：全局类路径 和文档级类路径。全局类路径是可以被所有 Flash 文档共享的类路径。文档级类路径是专门为单个 Flash 文档定义的类路径。

全局类路径适用于外部 ActionScript 文件和 FLA 文件，可在“首选参数”对话框（Windows：“编辑”>“首选参数”(Windows) 或 “Flash”>“首选参数”(Macintosh)，从 “类别”列表中选择 “ActionScript”，然后单击 “ActionScript 2.0 设置”）中进行设置。文档级类路径可在 Flash 文档的 “发布设置”对话框（“文件”>“发布设置”，选择 “Flash”选项卡，然后单击 “设置”按钮）中进行设置。

注释	如果在编辑 ActionScript 文件时单击 “脚本”窗格上面的 “语法检查”按钮，则编译器只在全局类路径中进行查找。 ActionScript 文件在 “编辑”模式下不与 FLA 文件关联，不具有自己的类路径。

使用全局类路径

全局类路径是可以被所有 Flash 文档共享的类路径。

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 文件的 “发布设置” 对话框。在上述两种情况下，您都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 “.”）。目录在对话框中的顺序反映的是它们的搜索顺序。

默认情况下，全局类路径包含一个绝对路径和一个相对路径。绝对路径在 “首选参数” 对话框中用 $(LocalData)/Classes 表示。绝对路径的位置如下所示：

- Windows：Hard Disk\Documents and Settings\ 用户 \Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\Classes。
- Macintosh：Hard Disk/Users/ 用户 /Library/Application Support/Macromedia/Flash 8/ 语言 /Configuration/Classes。

> **注意** 请勿删除绝对全局类路径。 Flash 要使用此类路径访问内置类。如果您意外删除了这个类路径，可将 $(LocalData)/Classes 作为新的类路径进行添加，从而恢复该类路径。

全局类路径的相对路径部分用一个圆点 (.) 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。

可以使用以下步骤添加全局类路径或编辑现有的类路径。

若要修改全局类路径，请执行以下操作：

1. 选择 “编辑” > “首选参数” (Windows) 或 “Flash” > “首选参数” (Macintosh)，以打开 “首选参数” 对话框。
2. 在左列中单击 “ActionScript”，然后单击 “ActionScript 2.0 设置” 按钮。
3. 单击 “浏览到路径” 按钮，浏览到要添加的目录。
4. 浏览到要添加的路径，然后单击 “确定”。

从类路径中删除目录：

1. 在 “类路径” 列表中选择其路径。
2. 单击 “从路径删除” 按钮。

> **注意** 请勿删除绝对全局类路径。 Flash 要使用此类路径访问内置类。如果您意外删除了这个类路径，可将 $(LocalData)/Classes 作为新的类路径进行添加，从而恢复该类路径。

有关导入包的信息，请参见第 166 页的 “使用包”。

使用文档级类路径

文档级类路径仅适用于 FLA 文件。可以在 “发布设置”对话框（“文件” > “发布设置”，单击 “Flash”选项卡，然后单击 “ActionScript 2.0 设置”）中为特定的 FLA 文件设置文档级类路径。文档级类路径默认为空。在目录中创建和保存 FLA 文件时，该目录将成为指定的类路径目录。

创建类时，有时候您可能要将其存储在某个目录下，在遇到以下情况时会将该目录添加到全局类路径目录列表中：

- 如果您有一组实用程序类，并且您所有的项目都使用这组类
- 如果您要对外部 ActionScript 文件中的代码进行语法检查 （单击 “语法检查”按钮）

如果您曾经卸载并重装过 Flash，特别是在删除和覆盖了默认全局类路径目录时，由于会丢失存储在该目录中的任何类，因此可以创建一个目录以防止丢失自定义类。

例如，您可以为自定义类创建如下目录：

- Windows：Hard Disk\Documents and Settings\ 用户 \custom classes。
- Macintosh：Hard Disk/Users/ 用户 /custom classes。

然后，您可将此路径添加到全局类路径列表中 （请参见第 176 页的 “使用全局类路径”）。

Flash 尝试解析 FLA 脚本中的类引用时，首先会对为该 FLA 文件指定的文档级类路径进行搜索。如果 Flash 未能在该类路径中找到该类，或如果该类路径为空，它将对全局类路径进行搜索。如果 Flash 未能在全局类路径中找到该类，则会出现一个编译器错误。

若要修改文档级类路径：

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 “ActionScript 版本”弹出菜单旁边的 “设置”按钮。
4. 您可以手动键入文件路径，也可以单击 “浏览到路径”按钮浏览到要添加到类路径中的目录。

注意	若要编辑现有的类路径目录，请在 “类路径”列表中选择路径，单击 “浏览到路径”按钮浏览到要添加的目录，然后单击 “确定”。

注意	若要从类路径中删除某个目录，请在 “类路径”列表中选择该路径，然后单击 “删除所选路径” (-) 按钮。

有关包的更多信息，请参见第 165 页的 “关于包”。

编译器如何解析类引用

Flash 尝试解析 FLA 脚本中的类引用时，首先会对为该 FLA 文件指定的文档级类路径进行搜索。如果未能在该类路径中找到该类，或如果该路径为空， Flash 将对全局类路径进行搜索。如果在全局类路径中也未找到该类，则会出现一个编译器错误。

在 Flash Professional 中，如果在编辑 ActionScrit 文件时单击 “检查语法”按钮，则编译器将只在全局类路径中进行查找； ActionScrit 文件在“编辑”模式中不与 FLA 文件关联，不具有自己的类路径。

在 Flash 中使用类文件

若要创建 ActionScript 类的实例，请使用 `new` 运算符调用该类的构造函数。构造函数与类通常具有相同的名称，并且返回该类的实例，而您通常会将该实例分配给一个变量。例如，如果您在使用第 170 页的 “编写自定义类文件”中的 User 类，则您可以编写以下代码创建一个新 User 对象：

```
var firstUser:User = new User();
```

注意 在某些情况下，您无需创建类的实例即可使用其属性和方法。有关类 （静态）成员的更多信息，请参见第 225 页的 “关于类 （静态）成员”和第 182 页的 “静态方法和属性”。

使用点 (.) 运算符可以访问实例中的属性值。在点的左边键入实例名称，在点的右边键入属性名称。例如，在下面的语句中， `firstUser` 是实例，而 `username` 是属性：

```
firstUser.username
```

在 Flash 文档中还可以使用构成 ActionScript 语言的顶级类或内置类。例如，下面的代码创建一个新的 Array 对象，然后显示其 `length` 属性：

```
var myArray:Array = new Array("apples", "oranges", "bananas");
trace(myArray.length); // 3
```

有关在 Flash 中使用自定义类的更多信息，请参见第 206 页的 “示例：在 Flash 中使用自定义类文件”。有关构造函数的信息，请参见第 199 页的 “编写构造函数”。

使用类文件中的方法和属性

在 OOP 中，类的成员（属性或方法）可以是实例成员，也可以是类成员。实例成员是为类的每个实例创建的；在类定义中初始化实例成员时，会将其定义为该类的原型。与此相反，类成员对每个类只创建一次。（类成员也称作静态成员。）

属性是定义对象的特性。例如，length 是所有数组的属性，它指定数组中的元素个数。方法是与类关联的函数。有关函数和方法的更多信息，请参见第 141 页的第 5 章“函数和方法”。

下面的示例介绍了如何在类文件中创建方法：

```
class Sample {
  public function myMethod():Void {
    trace("myMethod");
  }
}
```

接下来可以在文档中调用该方法。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。在下面的示例中，自定义 Picture 类（在随后的练习中将用到它）的一个实例 picture01 调用 showInfo() 方法：

```
var img1:Picture = new Picture("http://www.helpexamples.com/flash/images/
  image1.jpg");
// 调用 showInfo() 方法。
img1.showInfo();
```

下一个示例演示了如何编写自定义 Picture 类以存放有关一张照片的各种信息。

要在 FLA 文件中使用 Picture 和 PictureClass 类，请执行以下操作：

1. 选择“文件”>“新建”，然后选择“ActionScript 文件”。将该文档保存为 **Picture.as**，然后单击 “确定”。

   在此文档中编写自定义 Picture 类。

2. 在 “脚本”窗口中键入以下 ActionScript 代码：

```
/**
  Picture 类
  作者: John Doe
  版本: 0.53
  修改时间: 6/24/2005
  版权所有: Macromedia, Inc.

  Picture 类可用作图像及其 URL 的容器。
*/

class Picture {
  private var __infoObj:Object;
```

```
    public function Picture(src:String) {
      this.__infoObj = new Object();
      this.__infoObj.src = src;
    }

    public function showInfo():Void {
      trace(this.toString());
    }
    private function toString():String {
      return "[Picture src=" + this.__infoObj.src + "]";
    }

    public function get src():String {
      return this.__infoObj.src;
    }
    public function set src(value:String):Void {
      this.__infoObj.src = value;
    }
  }
```

3. 保存该 ActionScript 文件。
4. 选择 “文件” > “新建”，然后选择 “Flash 文档”创建一个新的 FLA 文件。在保存 Picture 类文件的同一目录中将该文件保存为 **picture_test.fla**。
5. 将下面的 ActionScript 代码键入到时间轴的第 1 帧中：

```
var picture1:Picture = new Picture("http://www.helpexamples.com/flash/
  images/image1.jpg");
picture1.showInfo();
this.createEmptyMovieClip("img_mc", 9);
img_mc.loadMovie(picture1.src);
```

6. 保存 Flash 文档。
7. 选择 “控制” > “测试影片”来测试该文档。

   下面是在 “输出”面板中显示的文本：

```
[Picture src=http://www.helpexamples.com/flash/images/image1.jpg]
```

硬盘上的一个范例文件演示了如何使用 XML 数据和自定义类文件创建动态菜单。该范例调用 ActionScript `XmlMenu()` 构造函数并向其传递两个参数: XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分在自定义类文件 XmlMenu.as 中。

可从硬盘上的 Samples 文件夹中找到该范例源文件 xmlmenu.fla。

- 在 Windows 上，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

关于公共、私有与静态方法和属性（成员）

在外部脚本文件中编写 ActionScript 类文件时，可以创建四种类型的方法和属性。公共方法和属性、私有方法和属性、公共静态方法和属性以及私有静态方法和属性。这些方法和属性定义了 Flash 可以如何访问变量，并且它们还允许您指定代码的哪些部分可以访问特定的方法或属性。

在构建基于类的应用程序时，无论应用程序的大小如何，都要考虑方法或属性应该是私有的还是公共的，这一点特别重要。考虑到这一点可以确保代码尽可能安全。例如，如果您构建了一个 User 类，您可能不想让使用该类的人能够更改用户 ID。通过将类属性（有时称作实例成员）设置为 `private`，就可以将对该属性的访问权限仅授予该类或该类子类内的代码，也就是说任何用户都不能直接更改该属性。

公共方法和属性

`public` 关键字指定变量或函数对任何调用者都可用。由于默认情况下变量和函数是公共的，因此主要将 `this` 关键字用于实现规范化和可读性的优点，用以指示当前作用域中存在的变量。例如，在包含私有或静态变量的代码块中，您可能要使用 `this` 关键字保持格式一致。`this` 关键字可与 public 或 private 关键字共同使用。

下面的 Sample 类已经有一个名为 `myMethod()` 的公共方法：

```
class Sample {
  private var ID:Number;
  public function myMethod():Void {
    this.ID = 15;
    trace(this.ID); // 15
    trace("myMethod");
  }
}
```

如果要添加 `public` 属性，则需要使用“public”一词，而不是“private”，如下面的范例代码中所示：

```
class Sample {
  private var ID:Number;
  public var email:String;
  public function myMethod():Void {
    trace("myMethod");
  }
}
```

由于 `email` 属性是公共的，因此可以在 Sample 类中对它进行更改，也可以直接在 FLA 中更改。

私有方法和属性

private 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，变量或函数都是公共的，对任何调用者都可用。如果您想限制对变量或函数的访问，请使用 this 关键字，如下面的示例中所示：

```
class Sample {
  private var ID:Number;
  public function myMethod():Void {
    this.ID = 15;
    trace(this.ID); // 15
    trace("myMethod");
  }
}
```

如果要向上面的类中添加私有属性，只需在 var 关键字前使用关键字 private 即可。

如果尝试从 Sample 类外部访问私有 ID 属性，则将产生编译器错误并在 “输出”面板中显示一条消息。该消息指示成员为私有，不能访问。

静态方法和属性

static 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。可以在不创建类的实例的情况下访问静态类成员。静态方法和属性可以在公共作用域内设置，也可以在私有作用域内设置。

静态成员（也叫类成员）分配给类本身，而不是分配给该类的某个实例。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，如下面的代码所示：

```
trace(Math.PI / 8); // 0.392699081698724
```

如果在 “动作”面板的脚本窗格中键入这一行代码，将在 “输出”面板中显示结果。

例如，在上面的 Sample 类示例中，您可以创建一个静态变量来跟踪创建了该类的多少个实例，如下面的代码中所示：

```
class Sample {
  public static var count:Number = 0;
  private var ID:Number;
  public var email:String;
  public function Sample() {
    Sample.count++;
    trace("count updated:" + Sample.count);
  }
  public function myMethod():Void {
    trace("myMethod");
  }
}
```

每创建 Sample 类的一个新实例，构造函数方法都会跟踪到目前为止已经定义的 Sample 类实例总数。

有些顶级 ActionScript 类具有类成员（或静态成员），在本部分前面调用 Math.PI 属性时已看到这一点。类成员（属性和方法）是通过类名称来访问或调用的，而不是通过类的实例。因此，不要创建该类的实例来使用其属性和方法。

例如，顶级 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 squareRoot:Number = Math.sqrt(4);
trace(squareRoot); // 2
```

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:Number = Math.max(10, 20);
trace(largerNumber); // 20
```

有关创建类成员的更多信息，请参见第183页的“关于类成员”和第186页的“使用类成员”。

硬盘上的一个范例文件演示了如何使用 XML 数据和自定义类文件创建动态菜单。该范例调用 ActionScript XmlMenu() 构造函数并向其传递两个参数：XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分在自定义类文件 XmlMenu.as 中。

可从硬盘上的 Samples 文件夹中找到该范例源文件 xmlmenu.fla。

- 在 Windows 上，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

关于类成员

到目前为止，在本章中讨论的大多数成员（方法和属性）的类型都是实例成员。对于每个实例成员，在类的每一个实例中都有该成员的唯一副本。例如，Sample 类的 email 成员变量有一个实例成员，因为每个人都有不同的电子邮件地址。

另一种类型的成员是类成员。类成员只有一个副本，它用于整个类。在类内（但在函数外）声明的任何函数都是该类的属性。在下面的示例中，Person 类有两个属性 age 和 username，其类型分别为 Number 和 String：

```
class Person {
  public var age:Number;
  public var username: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，可以创建一个名为 getInfo() 的方法：

```
class Person {
  public var age:Number;
  public var username:String;
  public function getInfo():String {
    // getInfo() 方法定义
  }
}
```

在上面的代码片断中， Person 类的 getInfo() 方法以及 age 和 username 属性都是公共实例成员。age 属性不会是一个好的类成员，因为每个人都有不同的年龄。只有由类的所有个体所共享的属性和方法才能是类成员。

假定您想要每一个类都具有一个 species 变量，用来表示该类代表的种类的正确的拉丁文名称。对于每一个 Person 对象，其种类都是智人 (Homo sapiens)。为类的每一个实例存储 "Homo sapiens" 字符串的唯一副本是很浪费的，因此这个成员应是类成员。

类成员用 static 关键字声明。例如，您可以使用下面的代码声明 species 类成员：

```
class Person {
  public static var species:String = "Homo sapiens";
  // ...
}
```

也可以将类的方法声明为静态方法，如下面的代码所示：

```
public static function getSpecies():String {
  return Person.species;
}
```

静态方法只能访问静态属性，而不能访问实例属性。例如，下面的代码将导致出现编辑器错误，因为类方法 getAge() 引用了实例变量 age：

```
class Person {
  public var age:Number = 15;
  // ...
  public static function getAge():Number {
    return age; /* **错误**: Instance variables cannot be accessed in static
  functions. */
  }
}
```

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。

有关类成员 （也叫做静态属性）的更多信息，请参见第 182 页的 “静态方法和属性”。

硬盘上的一个范例文件演示了如何使用 XML 数据和自定义类文件创建动态菜单。该范例调用 ActionScript XmlMenu() 构造函数并向其传递两个参数：XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分在自定义类文件 XmlMenu.as 中。

可从硬盘的 Samples 文件夹中找到该范例源文件 xmlmenu.fla。

- 在 Windows 上，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

使用 Singleton 设计模式

使用类成员的一种常见方法是使用 Singleton 设计模式。设计模式 定义了构造代码的正式方法。通常，可以构建一个设计模式，将它作为一个解决常见编程问题的解决方案。有多种已建立的设计模式，如 Singleton。 Singleton 设计模式确保每个类只有一个实例，并且提供了一种以全局方式访问实例的方法。有关 Singleton 设计模式的详细信息，请参见 www.macromedia.com/devnet/mx/coldfusion/articles/design_patterns.html。

您经常会遇到这样的情况：在系统中正好只需要一个特定类型的对象。例如，在一个国际象棋游戏中只能有一个棋盘，在一个国家中只有一个首都城市。虽然只有一个对象，您仍应将此对象的功能封装在一个类中。但是，您可能需要管理和访问该对象的一个实例。使用全局变量可以达到这一目的，但对于绝大多数项目来说，不建议使用这种方法。更好的办法是允许类使用类成员来管理对象实例本身。下面的示例演示了 Singleton 设计模式的典型用法，其中仅创建一次 Singleton 实例。

若要使用 Singleton 设计模式，请执行以下操作：

1. 选择“文件”>“新建”，然后选择“ActionScript 文件”。将文档保存为 **Singleton.as**。
2. 在 “脚本”窗口中键入以下 ActionScript 代码：

```
/**
  Singleton 类
  作者: John Doe
  版本: 0.53
  修改时间: 6/24/2008
  版权所有: Macromedia, Inc.
*/

class Singleton {
  private static var instance:Singleton = null;
  public function trackChanges():Void {
    trace("tracking changes.");
  }
  public static function getInstance():Singleton {
    if (Singleton.instance == null) {
      trace("creating new Singleton.");
      Singleton.instance = new Singleton();
    }
    return Singleton.instance;
  }
}
```

3. 保存 Singleton.as 文档。
4. 选择 “文件” > “新建”，然后选择 “Flash 文档”以创建新的 FLA 文件，并在保存 Singleton 类文件的同一目录中将其保存为 **singleton_test.fla**。
5. 将下面的 ActionScript 代码键入到时间轴的第 1 帧中：

```
Singleton.getInstance().trackChanges(); // 跟踪更改。

var s:Singleton = Singleton.getInstance(); // 跟踪更改。
s.trackChanges();
```

6. 保存 Flash 文档。
7. 选择 “控制” > “测试影片”来测试该文档。

只在需要时才创建 Singleton 对象 （也就是直到其它代码通过调用 `getInstance()` 方法请求该对象时）。这通常称为迟滞创建，在很多情况下可以帮助提高代码效率。

请勿在应用程序中使用过多或过少类文件，因为这样会导致类文件设计不当，对应用程序的性能或工作流程不利。在其它地方（如时间轴）应始终尝试使用类文件，而不是编写代码；但是应避免创建很多类却只具有很少的功能或创建很少的类却要处理很多功能。这两种情况都是设计不当的表现。

使用类成员

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就增加一次。

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` 的值都增加 `1`，并在 “输出”面板中显示 `widgetCount` 的当前值。

若要使用类变量创建实例计数器，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 在 “脚本”窗口中键入以下代码：

```
class Widget {
  // 初始化类变量
  public static var widgetCount:Number = 0;
  public function Widget() {
    Widget.widgetCount++;
    trace("Creating widget #" + Widget.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 类的构造函数语句时，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. 选择 “文件” > “新建”，然后选择 “Flash 文档”，以创建一个新的 FLA，将其保存到与 Widget.as 相同的目录下，文件名为 **widget_test.fla**。
5. 在 widget_test.fla 中，将下面的代码键入到时间轴的第 1 帧中：

```
// 在创建类的任何实例之前，
// Widget.widgetCount 为零 (0)。
trace("Widget count at start:" + Widget.widgetCount); // 0
var widget1:Widget = new Widget(); // 1
var widget2:Widget = new Widget(); // 2
var widget3:Widget = new Widget(); // 3
trace("Widget count at end:" + Widget.widgetCount); // 3
```

6. 保存对 widget_test.fla 所做的更改。
7. 选择 “控制” > “测试影片” 对文件进行测试。

   Flash 会将以下信息显示在 “输出” 面板中：

```
Widget count at start: 0
Creating widget # 1
Creating widget # 2
Creating widget # 3
Widget count at end: 3
```

关于 getter 和 setter 方法

Getter 和 setter 方法是存取器方法，这表示它们通常上是用于更改私有类成员的公共接口。可以使用 getter 和 setter 方法来定义属性。即使在类内将它们定义为方法，仍可以在类外部将 getter 和 setter 方法作为属性来访问。类的外部属性可以与类的内部属性具有不同的名称。

使用 getter 和 setter 方法具有一些优势，如能够让您创建可以像访问属性一样访问的具有完善功能的成员。它们还允许您创建只读和只写属性。

尽管 getter 和 setter 方法非常有用，但也应注意不要过度使用，其中一个原因是，在特定情况下不利于代码维护。而且，它们提供了对类实现（如公共成员）的访问。OOP 惯例不提倡直接访问类内的属性。

编写类时，始终提倡将尽可能多的实例变量设为私有变量，并相应地添加 getter 和 setter 方法。这是因为，很多时候您可能不想让用户在您的类内更改某些变量。例如，如果您用一个私有静态方法跟踪为特定的类创建的实例数目，则您不希望用户使用代码修改该计数器。只有在调用构造函数语句时才会递增该变量。在这种情况下，可以创建一个私有实例变量，并只允许将 getter 方法用于计数器变量，这表示用户仅能够使用 getter 方法检索当前值，而不能使用 setter 方法设置新值。创建一个不带 setter 的 getter 是使类内的某些变量成为只读变量的简单方法。

使用 getter 和 setter 方法

getter 和 setter 方法的语法如下所示：

- getter 方法不能有任何参数，始终返回一个值。
- setter 方法始终带一个参数，从不返回任何值。

类通常会定义 getter 方法和 setter 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：

```
private var userName:String;
```

该类不允许其实例直接访问这个属性 （例如 user.userName = "Buster"），而改为提供两个方法 getUserName 和 setUserName，其实现如下一个示例所示。

使用 getter 和 setter 方法：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 在 “脚本”窗口中键入以下代码：

```
class Login {
  private var __username:String;
  public function Login(username:String) {
    this.__username = username;
  }
  public function getUserName():String {
    return this.__username;
  }
  public function setUserName(value:String):Void {
    this.__username = value;
  }
}
```

3. 将 ActionScript 文档保存为 **Login.as**。

   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。
4. 选择 “文件” > “新建”，然后选择 “Flash 文档”，以创建一个新的 FLA，将其保存到与 Login.as 相同的目录下，文件名为 **login_test.fla**。
5. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var user:Login = new Login("RickyM");

// 调用 getUserName() 方法
var userName:String = user.getUserName();
trace(userName); // RickyM

// 调用 setUserName() 方法
user.setUserName("EnriqueI");
trace(user.getUserName()); // EnriqueI
```

6. 选择 “控制” > “测试影片” 对文件进行测试。

   Flash 会将以下信息显示在 “输出” 面板中：

```
RickyM
EnriqueI
```

但是，如果要使用更简明的语法，可以使用隐式 getter 和 setter 方法。利用隐式 getter 和 setter 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，如下一个示例所示。

附加	隐式 getter 和 setter 方法是对 ActionScript 1.0 中 `Object.addProperty()` 方法的句法简化。

使用隐式 getter 和 setter 方法：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 在 “脚本” 窗口中键入以下代码：

```
class Login2 {
  private var __username:String;
  public function Login2(username:String) {
    this.__username = username;
  }
  public function get userName():String {
    return this.__username;
  }
  public function set userName(value:String):Void {
    this.__username = value;
  }
}
```

3. 将 ActionScript 文档保存为 **Login2.as**。

   切记，getter 方法不带任何参数。而 setter 方法必须正好有一个必要参数。setter 方法与同一作用域内的 getter 方法可以具有相同的名称。但 getter 和 setter 方法不能与其它属性同名。例如，在上面的示例代码中，您定义了名为 `userName` 的 getter 和 setter 方法；在此情况下，在同一个类中就不能再有名称也为 `userName` 的属性了。
4. 选择 “文件” > “新建”，然后选择 “Flash 文档”，以创建一个新的 FLA，将其保存到与 Login2.as 相同的目录下，文件名为 **login2_test.fla**。

5. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var user:Login2 = new Login2("RickyM");

// 调用 “获取”方法
var userNameStr:String = user.userName;
trace(userNameStr); // RickyM

// 调用 “设置”方法
user.userName = "EnriqueI";
trace(user.userName); // EnriqueI
```

与普通方法不同，调用 getter 和 setter 方法时，方法名后面不带任何括号或参量。调用 getter 和 setter 方法与调用同名属性类似。

6. 保存 Flash 文档，然后选择 “控制” > “测试影片”对文件进行测试。

Flash 会将以下信息显示在 “输出”面板中：

```
RickyM
EnriqueI
```

注意	不能在接口方法的声明中使用 getter 和 setter 方法属性。

关于动态类

将 `dynamic` 关键字添加到类定义中将指定基于指定类的对象可在运行时添加和访问动态属性。只有在真正需要此功能时才应创建动态类。

对于动态类的类型检查比对于非动态类的类型检查更为宽松，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作用域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。

有关创建动态类的信息，请参见第 191 页的 “创建动态类”。

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 `userName` 和 `age`。

若要创建不是动态的类，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 在 “脚本”窗口中键入下面的 ActionScript：

   ```
   class Person {
     public var userName:String;
     public var age:Number;
   }
   ```

   如果您在另一个脚本中创建了 Person 类的一个实例，并且尝试访问该类中不存在的某个属性，编辑器就会生成一个错误。
3. 在硬盘上将文件保存为 **Person.as**。
4. 选择 “文件” > “新建”，再选择 “Flash 文档”，以创建一个新的 FLA 文件，然后单击 “确定”。
5. 选择 “文件” > “另存为”，将文件命名为 **person_test.fla**，并将其保存在前面创建的 Person 类所在的目录中。
6. 添加下面的代码，以创建 Person 类的一个新实例 (`firstPerson`)，然后尝试为名为 `hairColor` 的属性赋值 （该属性在 Person 类中不存在）：

   ```
   var firstPerson:Person = new Person();
   firstPerson.hairColor = "blue"; // 错误。没有名称为 “hairColor”的属性。
   ```
7. 保存 Flash 文档。
8. 选择 “控制” > “测试影片”对代码进行测试。

   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。我们都不希望看到编译器错误，但是它们对程序员非常有用：好的错误消息可通过在编码过程中较早地指出错误来帮助编写正确的代码。

但在某些情况下，您可能需要在运行时添加和访问原始类定义中未定义的类属性或类方法。这可以通过 dynamic 类限定符来实现。

若要创建动态类，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 选择 “文件” > “另存为”，将文件命名为 **Person2.as**。将文件保存在硬盘上。
3. 在 “脚本” 窗口中键入以下代码：

```
dynamic class Person2 {
  public var userName:String;
  public var age:Number;
}
```

   此 ActionScript 将 `dynamic` 关键字添加到上面的示例的 Person 类中。Person2 类的实例可以添加和访问在类定义中未定义的属性和方法。

4. 保存对 ActionScript 文件所做的更改。
5. 选择 “文件” > “新建”，再选择 “Flash 文档”，创建一个新的 FLA 文件，然后单击 “确定”。
6. 选择 “文件” > “另存为”，将该新文件命名为 **person2_test.fla**。将该文件保存在 Person2.as 所在的同一目录中。
7. 键入下面的代码，以创建 Person2 类的一个新实例 (`firstPerson`)，然后尝试为名为 `hairColor` 的属性赋值 （该属性在 Person2 类中不存在）：

```
var firstPerson:Person2 = new Person2();
firstPerson.hairColor = "blue";
trace(firstPerson.hairColor); // blue
```

8. 保存对 person2_test.fla 文件所做的更改。
9. 选择 “控制” > “测试影片” 对代码进行测试。

   由于自定义 Flash 类是动态类，所以您可以在运行时 （SWF 文件播放时）向该类中添加方法和属性。测试代码时，“输出” 面板中应显示文本 `blue`。

开发应用程序时，除非必需，否则您可能不想让类成为动态类。不使用动态类原因之一是，对于动态类的类型检查比对于非动态类的类型检查更为宽松，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。

动态类的子类也是动态的，但有一种情况例外。默认情况下，MovieClip 类的子类不是动态的，即使 MovieClip 类本身是动态的。此实现使您能够对 MovieClip 类的子类进行更大程度的控制，因为您可以选择子类是否是动态的：

```
class A extends MovieClip {}         // A 不是动态类
dynamic class B extends A {}          // B 为动态类
class C extends B {}                  // C 为动态类
class D extends A {}                  // D 不是动态类
dynamic class E extends MovieClip{} // E 为动态类
```

有关子类的信息，请参见第 229 页的第 7 章 “继承”。

关于使用封装

在完美的面向对象设计中，对象被看作包含（或封装）功能的“黑匣子”。程序员应当能够在仅知道对象的属性、方法和事件（对象的编程接口）的情况下与对象进行交互，而不需知道其实现的详细信息。此方法使程序员可以在更高的抽象层次上思考，并能提供可用于构建复杂系统的组织框架。

封装是 ActionScript2.0 之所以包含诸如成员访问控制等功能的原因，这样实现的详细信息对于对象外的代码是私有的和不可见的。对象外的代码被强制与对象的编程接口进行交互，而不是与实现的详细信息交互。这种方法提供了一些重要优点；例如，只要编程接口不变，对象的创建者就可以在不更改对象外代码的情况下更改对象的实现。

在 Flash 中封装的一个示例是：将所有的成员和类变量设置为私有，并强制实现这些类的人员使用 getter 和 setter 方法访问这些变量。用这种方法执行封装可以确保：在将来需要更改变量的结构时，只需更改 getter 和 setter 函数的行为，而不用强制每个开发人员改变其访问该类的变量的方式。

下面的代码显示了可以怎样修改前面的示例中的 Person 类，怎样将其实例成员设置为私有，以及如何为私有实例成员定义 getter 和 setter 方法：

```
class Person {
  private var __userName:String;
  private var __age:Number;
  public function get userName():String {
    return this.__userName;
  }
  public function set userName(value:String):Void {
    this.__userName = value;
  }
  public function get age():Number {
    return this.__age;
  }
  public function set age(value:Number):Void {
    this.__age = value;
  }
}
```

关于在类中使用 this 关键字

在类中使用 `this` 关键字作为方法和成员变量的前缀。尽管 `this` 关键字并不是必需的，但是当一个属性或方法带有前缀时，很容易判断它是否属于某个类；如果没有这个关键字，则无法辨别该属性或方法是否属于其超类。

即使在一个类中，您也可以对静态变量和静态方法使用类名称前缀。这有助于限定所做的引用，使代码具有很好的可读性。根据所使用的编码环境，添加前缀可能还会触发代码提示。

提示	您不一定非得添加这些前缀，有些开发人员觉得这样做没有必要。 Macromedia 建议添加 `this` 关键字作为前缀，因为它可以增强可读性，并可通过提供方法和变量的上下文帮助您编写整洁的代码。

示例：编写自定义类

现在，您已经学习了类文件的基础知识及其包含的各种内容，下面将学习创建类文件的一些一般原则。本章的第一个示例介绍如何编写类并将它们打包。第二个示例演示如何将这些类文件与 FLA 文件一起使用。

小心	发布、导出、测试或调试 FLA 文件时，外部文件中的 ActionScript 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。

如第 170 页的“编写自定义类文件”中所述，一个类由两个主要部分组成：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，然后跟有一对花括号 ({})。花括号内的所有内容都是类体，如下面的示例所示：

```
class className {
  // 类体
}
```

切记：只能在外部 ActionScript 文件中定义类。例如，不能在 FLA 文件中的帧脚本中定义类。因此，需要为此示例创建一个新文件。

最基本的类声明应包括：一个 `class` 关键字，后面跟有类名称 （本例中为 Person），然后跟有一对大括号 ({})。花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。

在此示例的末尾，类文件的基本顺序如下所示：

- 文档注释
- 类声明
- 构造函数
- 类体

本章中不需要编写子类。有关继承和创建子类的更多信息，请参见第 229 页的第 7 章“继承”。

本示例包含以下主题：

硬盘上的一个范例文件演示了如何使用 XML 数据和自定义类文件创建动态菜单。该范例调用 ActionScript `XmlMenu()` 构造函数并向其传递两个参数：XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分在自定义类文件 XmlMenu.as 中。

可从硬盘上的 Samples 文件夹中找到该范例源文件 xmlmenu.fla。

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 中，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

关于创建类的一般原则

编写自定义类文件时必须遵循以下几条原则。它们可以帮助您编写正确而又形式完善的类。在以后的示例中将实践这些原则。

- 通常，每行只放置一个声明，不要在一行放置多个声明（不管类型是否相同）。声明的格式如下面的示例所示：

```
private var SKU:Number; // 产品 SKU（识别）号
private var quantity:Number; // 产品数量
```

- 当声明本地变量时要对其进行初始化，除非初始值要经过计算才能得出。有关初始化变量的信息，请参见第 201 页的 “添加方法和属性”。
- 变量在声明之后才能使用（包括在循环中）。例如，下面的代码在将循环迭代变量 (i) 用于 for 循环之前进行了预先声明：

```
var my_array:Array = new Array("one", "two", "three");
var i:Number;
for (i = 0 ; i < my_array.length; i++) {
  trace(i + " = " + my_array[i]);
}
```

- 避免使用隐藏更高级声明的局部声明。例如，不要将一个变量声明两次，如下面的示例所示：

```
// 错误的代码
var counter:Number = 0;
function myMethod() {
  var counter:Number;
  for (counter = 0; counter <= 4; counter++) {
    // 语句;
  }
}
```

  此代码在内部代码块中声明相同变量。

- 不要在一个语句中将许多变量赋为同一个值，因为这样很难阅读，如下面的 ActionScript 代码示例所示：

```
// 错误的形式
xPos = yPos = 15;
```

  或

```
// 错误的形式
class User {
  private var m_username:String, m_password:String;
}
```

- 只有在确实需要时才创建公共实例变量，或公共静态、类或成员变量。确保这些变量已明确声明为公共的，然后才能以这种方式创建它们。
- 将大多数成员变量设置为私有变量，除非有好的理由将其设置为公共变量。将成员变量设置为私有变量并且仅允许通过一小组 getter 和 setter 函数访问这些变量，从设计的角度而言这样要好得多。

关于命名类文件

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。一种好的做法是尝试始终将类名称限制为字母。

类名称必须与包含该类的 ActionScript 文件的文件名完全匹配（包括大小写）。在下面的示例中，如果创建一个名为 Rock 的类，那么包含类定义的 ActionScript 文件必须名为 Rock.as：

```
// 在文件 Rock.as 中
class Rock {
  // Rock 类体
}
```

您将在下面一节中命名和创建类定义。请参见第 197 页的 “创建和打包类文件”一节以创建、命名和打包类文件。有关命名类文件的更多信息，请参见第 658 页的“命名类和对象”。

创建和打包类文件

在本节中，您将为本示例（第 194 页的 “示例：编写自定义类”）创建、命名和打包类文件。下面各节介绍如何编写完整（但很简单）的类文件。有关包的详细信息，请参见第 165 页的 “关于包”、第 166 页的 “类和包的比较”和第 166 页的 “使用包”。

创建类文件时，您要决定在何处存储文件。在以下步骤中，为简单起见，您将在相同目录下保存类文件以及使用该类文件的应用程序 FLA 文件。但是，如果您想检查语法，还需要告诉 Flash 如何才能找到该文件。通常在创建应用程序时，应将存储应用程序和类文件的目录添加到 Flash 类路径中。有关类路径的信息，请参见第 175 页的“关于设置和修改类路径”。

类文件还称为 ActionScript (AS) 文件。可以在 Flash 创作工具中，或使用外部编辑器创建 AS 文件。有几种外部编辑器（如 Macromedia Dreamweaver 和 Macromedia Flex Builder）可用来创建 AS 文件。

注意	类的名称 (ClassA) 必须与包含该类的 AS 文件的名称 (ClassA.as) 完全匹配。这非常重要；如果这两个名称不完全匹配（包括大小写），将无法编译类。

若要创建类文件和类声明，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “Flash 文档”，创建一个新的 FLA 文档，然后单击 “确定”。
2. 选择 “文件” > “另存为”，将新文件命名为 **package_test.fla**，然后将该 Flash 文档保存到当前目录中。

   在以后的步骤中可以向此 Flash 文档添加内容。
3. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
4. 选择 “文件” > “另存为”，创建一个名为 **com** 的新子目录，然后执行以下操作：
   a. 在 com 子目录中，创建一个名为 **macromedia** 的新子目录。
   b. 在 macromedia 子目录中，创建一个名为 **utils** 的新子目录。
   c. 将当前的 ActionScript 文档保存在 utils 目录中，并将该文件命名为 **ClassA.as**。
5. 在 “脚本”窗口中键入以下代码：

   ```
   class com.macromedia.utils.ClassA {
   }
   ```

   上面的代码在 com.macromedia.utils 包中创建了一个名为 ClassA 的新类。
6. 保存 ClassA.as ActionScript 文档。
7. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
8. 选择 “文件” > “另存为”，将该新文件命名为 **ClassB.as**，然后将它保存在前面的步骤中创建的 ClassA.as 所在的目录中。
9. 在 “脚本”窗口中键入以下代码：

   ```
   class com.macromedia.utils.ClassB {
   }
   ```

   上面的代码在 com.macromedia.utils 包中创建了一个名为 ClassB 的新类。
10. 保存对 ClassA.as 和 ClassB.as 类文件所做的更改。

在编译 FLA 文件时，该文件中使用的类文件将导入到一个 SWF 文件中。在类文件中编写的代码应当具有一定的方法和顺序，以下部分将对此进行讨论。

如果您正在创建多个自定义类，请使用包对类文件进行组织。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。类名称在声明该类的文件内必须是完全限定的；也就是说，该类名称必须反映存储它的目录（包）。有关类路径的更多信息，请参见第 175 页的 “关于设置和修改类路径”。

例如，名为 com.macromedia.docs.YourClass 的类存储在 com/macromedia/docs 目录中。文件 YourClass.as 中的类声明如下所示：

```
class com.macromedia.docs.YourClass {
  // 您的类
}
```

注意 在下面一节 （第 194 页的 “示例：编写自定义类”）中您将编写反映包目录的类声明。

因此，最好在开始创建类之前就计划好您的包结构。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。

若要打包类文件，请执行以下操作：

1. 确定要使用的包名称。

   包名应该直观且易于由伙伴开发人员识别。切记，包名还应与特定的目录结构匹配。例如，com.macromedia.utils 包中的任何类都必须放在硬盘驱动器上的 com/macromedia/utils 文件夹中。

2. 选择包名之后创建需要的目录结构。

   例如，如果包命名为 com.macromedia.utils，则需要创建 com/macromedia/utils 这样一个目录结构，并将类放在 utils 文件夹中。

3. 在此包中创建的任何类都应使用 com.macromedia.utils 前缀。

   例如，如果类名称是 ClassA，则在 com/macromedia/utils/ClassA.as 类文件中完整的类名称应是 com.macromedia.utils.ClassA。

4. 以后对包结构进行更改时，切记不仅要修改目录结构，还要修改每个类文件中的包名以及每个 import 语句或对该包中的类的引用。

若要继续编写类文件，请参见第 199 页的 “编写构造函数”。

编写构造函数

在第 197 页的 “创建和打包类文件”中我们已经学习过如何编写类声明。在本章的这一部分，您将编写类文件的构造函数。

注意 在后面几节中您将学习如何编写注释、语句和声明。

构造函数是用于初始化（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，以下代码定义了一个 Person 类并实现一个构造函数。在 OOP 中，构造函数初始化类的每个新实例。

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，您创建的 Person 类包含下面的构造函数：

```
// Person 类构造函数
public function Person (uname:String, age:Number) {
  this.__name = uname;
  this.__age = age;
}
```

编写构造函数时应考虑以下几点：

- 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。
- 一个类只可以包含一个构造函数；ActionScript 2.0 中不允许重载构造函数。
- 构造函数不应具有返回类型。

在基于特定的类创建（实例化）对象时，通常也会使用术语构造函数 (constructor)。以下语句是对顶级 Array 类和自定义 Person 类的构造函数的调用：

```
var day_array:Array = new Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
  "Sat");
var somePerson:Person = new Person("Tom", 30);
```

接下来，您将添加一个称为构造函数的特殊函数。

附注

下面的练习是第 194 页的 “示例：编写自定义类”中的一部分。如果不希望进行该示例的操作，可以从 www.helpexamples.com/flash/learnas/classes/ 下载类文件。

若要将构造函数添加到类文件中，请执行以下操作：

1. 在 Flash 创作工具中打开 ClassA.as 类文件。
2. 修改现有类文件，使其与以下代码匹配（以粗体显示变化）：

   ```
   class com.macromedia.utils.ClassA {
     function ClassA() {
       trace("ClassA constructor");
     }
   }
   ```

   上面的代码定义了 ClassA 类的一个构造函数方法。此构造函数将一个简单字符串发送到 “输出”面板，这样您就能够知道类的一个新实例已创建完毕。
3. 在 Flash 创作工具中打开 ClassB.as 类文件。
4. 修改该类文件，使其与以下代码匹配（以粗体显示变化）：

   ```
   class com.macromedia.utils.ClassB {
     function ClassB() {
       trace("ClassB constructor");
     }
   }
   ```

5. 在继续进行之前保存这两个 ActionScript 文件。

若要继续编写类文件，请参见第 201 页的 “添加方法和属性”。

添加方法和属性

若要创建 ClassA 和 ClassB 类的属性，请使用 `var` 关键字来定义变量。

注意

下面的三个练习是第 194 页的 “示例：编写自定义类”中的一部分。如果不希望进行该示例的操作，可从 www.helpexamples.com/flash/learnas/classes/ 下载类文件。

若要向 ClassA 和 ClassB 类中添加属性，请执行以下操作：

1. 在 Flash 创作工具中打开 ClassA.as 和 ClassB.as。
2. 修改 ClassA.as ActionScript 文件，使其与以下代码匹配 （以粗体显示变化）：

```
class com.macromedia.utils.ClassA {
  static var _className:String;
  function ClassA() {
    trace("ClassA constructor");
  }
}
```

   上面的代码块添加一个新的静态变量 `_className`，该变量包含当前类的名称。

3. 修改 ClassB 类并添加静态变量，以便让它与上面的代码相似。
4. 在继续进行之前保存这两个 ActionScript 文件。

提示

按照惯例，类属性在类体顶部进行定义。在顶部定义类属性将使代码更易于理解，但这并非是必需的。

可以在变量声明中使用后冒号语法 （例如 `var username:String` 和 `var age:Number`）。这就是严格数据类型指定的一个示例。使用 `var variableName:variableType` 格式键入变量时，ActionScript 编译器将确保赋给该变量的任何值都与指定的类型匹配。如果导入该类的 FLA 文件中使用的数据类型不正确，编译器将抛出一个错误。有关严格数据类型指定的更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。

类的成员包括属性 （变量声明）和方法 （函数定义）。所有属性和方法都必须在类体 （大括号 [{}]）内声明和定义；否则，编译时将出现错误。有关成员的信息，请参见第 181 页的 “关于公共、私有与静态方法和属性 （成员）”。

若要向 ClassA 和 ClassB 类添加方法，请执行以下操作：

1. 在 Flash 创作工具中打开 ClassA.as 和 ClassB.as。
2. 修改 ClassA 类文件，使其与以下代码匹配（以粗体显示变化）：

```
class com.macromedia.utils.ClassA {
  static var _className:String;

  function ClassA() {
    trace("ClassA constructor");
  }
  function doSomething():Void {
    trace("ClassA - doSomething()");
  }
}
```

   以粗体显示的代码块在类中创建一个新方法，该方法将字符串显示到“输出”面板中。

3. 在 ClassA.as 中，选择“工具”>“语法检查”，检查 ActionScript 文件的语法。

   如果“输出”面板中报告有任何错误，请将脚本中的 ActionScript 与上一步骤中编写的完整代码进行比较。如果无法修复代码错误，在继续进行之前请先将完整的代码复制并粘贴到“脚本”窗口中。

4. 按照与处理 ClassA.as 相同的方法检查 ClassB.as 的语法。

   如果在“输出”面板中出现任何错误，请在继续前将完整代码复制并粘贴到“脚本”窗口中：

```
class com.macromedia.utils.ClassB {
  static var _className:String;

  function ClassB() {
    trace("ClassB constructor");
  }
  function doSomething():Void {
    trace("ClassB - doSomething()");
  }
}
```

5. 在继续进行之前保存这两个 ActionScript 文件。

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，如下面的示例所示：

```
class Person {
  var age:Number = 50;
  var username: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，以及以下顶级类的构造函数: Array、Boolean、Number、Object 和 String。

若要以内联方式初始化属性，请执行以下操作：

1. 在 Flash 创作工具中打开 ClassA.as 和 ClassB.as。
2. 修改 ClassA 类文件，使代码与以下 ActionScript 匹配 （以粗体显示变化）：

```
class com.macromedia.utils.ClassA {
  static var _className:String = "ClassA";

  function ClassA() {
    trace("ClassA constructor");
  }
  function doSomething():Void {
    trace("ClassA - doSomething()");
  }
}
```

   现有类文件与上面的代码块之间的唯一区别是：现在为静态变量 `_className` 定义了一个值 “ClassA”。

3. 修改 ClassB 类文件并添加内联属性，将值更改为 “ClassB”。
4. 在继续进行之前保存这两个 ActionScript 文件。

此规则仅适用于实例变量（复制到类各个实例中的变量），而不适用于类变量（属于该类的变量）。

以内联方式初始化数组时，对类的所有实例只创建一个数组。

若要继续编写类文件，请参见第 203 页的 “控制类中的成员访问”。

控制类中的成员访问

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。但是，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） 。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():String {
    return this.userName;
  }
  // 构造函数:
  public 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。在下面的示例中，将更改类文件中的成员访问。

注释 该练习是第 194 页的 “示例：编写自定义类”中的一部分。如果不希望进行该示例的操作，可从 www.helpexamples.com/flash/learnas/classes/ 下载类文件。

若要控制成员访问，请执行以下操作：

1. 在 Flash 创作工具中打开 ClassA.as 和 ClassB.as。
2. 修改 ClassA.as ActionScript 文件，使其内容与以下 ActionScript 匹配（以粗体显示变化）：

```
class com.macromedia.utils.ClassA {
  private static var _className:String = "ClassA";

  public function ClassA() {
    trace("ClassA constructor");
  }
  public function doSomething():Void {
    trace("ClassA - doSomething()");
  }
}
```

   上面的代码将两种方法（ClassA 构造函数和 `doSomething()` 方法）设置为公共，也就是说可以通过外部脚本访问到它们。静态变量 `_className` 被设置为私有，这意味着只能在类中（而不能从外部脚本）访问到该变量。
3. 修改 ClassB.as ActionScript 文件，添加与 ClassA 类相同的方法和属性访问。
4. 在继续进行之前保存这两个 ActionScript 文件。

ClassA 或 ClassB 的实例不能访问私有成员。例如，如果将以下代码添加到某个 FLA 文件内的时间轴的第 1 帧，将出现一个编译器错误，提示您该方法是私有成员，不能访问：

```
import com.macromedia.utils.ClassA;
var a:ClassA = new ClassA();
trace(a._className); // 错误。该成员是私有成员，不能访问。
```

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。

若要继续编写类文件，请参见第 205 页的 “创建类文档”。

创建类文档

在类和接口中使用注释是为其创建类文档的重要部分，这些文档可供其他用户使用。例如，您可能要将类文件分发到 Flash 社区，或者您可能正与一组设计人员或开发人员合作，他们将在工作中或作为您正在从事的项目的一部分使用您的类文件。文档有助于其他用户理解类的目的和起源。

在典型的类或接口文件中有两种注释：文档注释 和实现注释。可以使用文档注释来描述代码的规范，但不能描述实现。可以使用实现注释来注释掉代码，或对代码特定部分的实现加以注释。这两种注释使用的分隔符略有区别。文档注释是以 `/**` 和 `*/` 分隔的，而实现注释是以 `/*` 和 `*/` 分隔的。

注意 文档注释不是 ActionScript 2.0 中的语言构造。不过，这却是在类文件中构造可在 AS 文件中使用的注释的常见方法。

可以使用文档注释描述接口、类、方法和构造函数。应为每个类、接口或成员使用一个文档注释，并在其后紧跟着声明。

如果您有其它不适合放在文档注释中的文档信息，请使用实现注释（采用第 80 页的 “关于注释”中所述的注释块或单行注释的形式）。如果添加实现注释，应让它们紧随声明之后。

注意 不要包含与正在阅读的类没有直接关联的注释。例如，不要包含描述相应包的注释。

注意 下面的练习是第 194 页的 “示例：编写自定义类”中的一部分。如果不希望进行该示例的操作，可从 www.helpexamples.com/flash/learnas/classes/ 下载类文件。

若要创建类文件的文档，请执行以下操作：

1. 在 Flash 创作工具中打开 ClassA.as 和 ClassB.as。
2. 修改 ClassA 类文件，将新代码添加到该类文件的顶部 （以粗体显示变化）：

```
/**
  ClassA 类
  1.1 版
  6/21/2005
  版权所有 Macromedia, Inc.
 */
class com.macromedia.utils.ClassA {
  private static var _className:String = "ClassA";

  public function ClassA() {
    trace("ClassA constructor");
  }
  public function doSomething():Void {
    trace("ClassA - doSomething()");
  }
}
```

上面的代码向类文件顶部添加注释。最好为 ActionScript 和 Flash 文件添加注释，以便可以添加一些有用的信息，例如类的作者、最后一次修改日期、版权信息或任何可能会在该文件中出现的潜在问题 / 错误等。

3. 在 ClassB.as ActionScript 文件的顶部添加类似的注释，更改类名和其它任何您认为适合更改的信息。
4. 在继续进行之前保存这两个 ActionScript 文件。

您还可以在类的代码中添加块、单行注释或尾随注释。有关在代码内编写良好注释的信息，请参见第 662 页的 “编写好的注释”。有关注释的常规信息，请参见第 80 页的 “单行注释”、第 81 页的 “多行注释”和第 82 页的 “尾随注释”。

要学习如何在 SWF 文件中使用这些自定义类文件，请参见第 206 页的 “示例：在 Flash 中使用自定义类文件”。

示例：在 Flash 中使用自定义类文件

此示例使用在名为第 194 页的 “示例：编写自定义类”的示例中编写的类文件，您也可以从 www.helpexamples.com/flash/learnas/classes/ 下载这些类文件。如果完成了第 194 页的 “示例：编写自定义类”，请在硬盘上查找 ClassA.as 和 ClassB.as。

由于 ClassA 类文件的包名是 `com.macromedia.utils.ClassA`，所以需要确保以正确的目录结构保存这些类文件。在当前目录下创建名为 com 的子文件夹。在 com 文件夹中，添加一个名为 macromedia 的新文件夹。在 macromedia 文件夹中添加名为 utils 的第三个（最后一个）子目录。将 ClassA.as 和 ClassB.as 这两个类文件保存在此 utils 文件夹中。现在您可以继续学习此示例后面的部分了。

您可以将在第 194 页的 “示例：编写自定义类”中编写的自定义类放在一个 FLA 文件中使用。在本示例中，将使用自定义类在 Flash 中创建一个小应用程序。发布文档时，类将编译到 SWF 文件中，然后各个部分就可以协同工作了。在下面的练习中，您将了解类路径的工作原理、如何在应用程序中使用类文件以及如何导入类和包。

要继续此示例，请进入第 207 页的 “导入类和包”。

导入类和包

若要引用另一个脚本中的某个类，必须将该类的包名做为类名前缀。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），则其完全限定的类名称就是其类名称。

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，亦即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 ClassName 的类，它位于您的类路径下的 com/macromedia/docs/learnAs2 包中。若要创建该类的一个实例，您可以指定全限定类名。

您还可以使用完全限定的类名称来指定变量的类型，如下面的示例所示：

```
var myInstance:com.macromedia.docs.learnAs2.ClassName = new
  com.macromedia.docs.learnAs2.ClassName();
```

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称，而不必使用其完全限定名。您还可以使用通配符 (*) 导入包中的所有类。如果使用通配符，就不需要在每次使用类时都使用完全限定的类名称。

例如，假设在一个脚本中，您使用 import 语句导入了上面的类，如下面的示例所示：

```
import com.macromedia.docs.learnAs2.util.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，如下面的示例所示：

```
var myUser:UserClass = new UserClass();
```

您可以使用通配符 (*) 导入给定包中的所有类。假设您有一个名为 com.macromedia.utils 的包，其中包含两个 ActionScript 类文件 ClassA.as 和 ClassB.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，如下面的代码所示：

```
import com.macromedia.utils.*;
```

下面的示例演示了可以在同一个脚本中直接引用上述两个类之一：

```
var myA:ClassA = new ClassA();
var myB:ClassB = new ClassB();
```

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 文件可能加载的任何 SWF 文件都不可用。

注释

下面的练习是第 206 页的 “示例：在 Flash 中使用自定义类文件”中的一部分，该练习是示例“示例：编写自定义类”的延续。如果需要 ClassA 和 ClassB，可从 www.helpexamples.com/flash/learnas/classes/ 下载其类文件。

若要导入类或包，请执行以下操作：

1. 打开名为 package_test.fla 的文件。
2. 在 “脚本”窗口中键入以下代码：

```
import com.macromedia.utils.*;
var a = new ClassA(); // ClassA 构造函数
var b = new ClassB(); // ClassB 构造函数
```

   上面的代码块首先使用通配符 (*) 导入 `com.macromedia.utils` 包中的每个类。接着，创建 ClassA 类的一个新实例，这会使构造函数方法向 “输出”面板发送一条消息。还将创建 ClassB 类的一个实例，而这将向 “输出”面板发送调试消息。

3. 在继续进行之前，保存对该 Flash 文档所做的更改。

若要继续在 Flash 文件中使用这些类文件，请参见第 208 页的 “在示例中创建类的实例”。

在示例中创建类的实例

实例是包含某个特定类的所有属性和方法的对象。例如，数组是 Array 类的实例，因此您可以将 Array 类的任何方法或属性用于任何 array 实例。或者，您可以创建自己的类（如 UserSettings），然后创建 UserSettings 类的一个实例。

我们继续探讨在第 206 页的 “示例：在 Flash 中使用自定义类文件”中开始的示例，在此示例中您修改了 FLA 文件，以导入编写的类，从而不必总是通过完全限定的名称来引用它们。

本示例（第 206 页的 “示例：在 Flash 中使用自定义类文件”）的下一个步骤是在脚本中创建 ClassA 和 ClassB 类的实例，如 package_test.fla Flash 文档中的一个帧脚本，然后将它赋给一个变量。若要创建自定义类的实例，应使用 `new` 运算符，就像创建顶级 ActionScript 类（例如，Date 或 Array 类）的实例一样。可以使用类的完全限定类名引用该类或将其导入，如第 207 页的 “导入类和包”中所示。

附注

下面的练习是第 206 页的 “示例：在 Flash 中使用自定义类文件”中的一部分，该练习是示例“示例：编写自定义类”的延续。

若要创建 ClassA 和 ClassB 类的新实例，请执行以下操作：

1. 打开名为 **package_test.fla** 的文件。
2. 在 “脚本”窗口中键入下面的粗体代码：

```
import com.macromedia.utils.*;
var a:ClassA = new ClassA(); // ClassA 构造函数
a.doSomething(); // 调用 ClassA 的 doSomething() 方法
var b:ClassB = new ClassB(); // ClassB 构造函数
b.doSomething(); // 调用 ClassB 的 doSomething() 方法
```

   在此代码示例中指定对象的数据类型使编译器能够确保您不会尝试访问自定义类中未定义的属性或方法。有关严格数据类型指定的更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。但如果您使用 `dynamic` 关键字将类声明为动态类，则上述情形并不适用。请参见第 191 页的 “创建动态类”。

3. 在继续进行之前，保存对该 FLA 文件所做的更改。

您现在应该对如何在 Flash 文档中创建和使用类有了基本认识。切记，还可以创建顶级 ActionScript 类或内置类的实例（请参见第 223 页的 “关于使用内置类”）。

若要继续在 Flash 文件中使用这些类文件，请参见第 209 页的 “将类分配给 Flash 中的元件”。

将类分配给 Flash 中的元件

您还可以将类分配给可能会在 Flash 文件中使用的元件，如舞台上的影片剪辑对象。

若要将类分配给影片剪辑元件，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 选择 “文件” > “另存为”，将文件命名为 **Animal.as**，然后将文件保存到硬盘上。
3. 在 “脚本”窗口中键入以下代码：

```
class Animal {
  public function Animal() {
    trace("Animal::constructor");
  }
}
```

   此 ActionScript 创建一个名为 Animal 的新类，该类带有一个构造函数方法可以将一个字符串发送到 “输出”面板中。

4. 保存对 ActionScript 文件所做的更改。
5. 选择 “文件” > “新建”，再选择 “Flash 文档”，创建一个新的 FLA 文件，然后单击 “确定”。

6. 选择 “文件” > “另存为”，将文件命名为 **animal_test.fla**，然后将该文件与步骤 2 中创建的 Animal.as 文件保存在同一文件夹中。
7. 选择 “插入” > “新建元件”，以启动 “创建新元件”对话框。
8. 输入 **animal** 的元件名称，然后选择 “影片剪辑”选项。
9. 单击 “创建新元件”对话框右下角的 “高级”按钮，启用更多选项。

   “高级”按钮在 “创建新元件”对话框基本模式中可用。
10. 在 “链接”部分单击 “为 ActionScript 导出”复选框。

    启用此选项使您可以在运行期间将此元件的实例动态附加到 Flash 文档中。
11. 输入 **animal_id** 的标识符值，将 “ActionScript 2.0 类”设置为 **Animal**（以与步骤 3 中指定的类名称匹配）。
12. 选择 “在第一帧导出”复选框，然后单击 “确定”以应用更改并关闭对话框。
13. 保存 Flash 文档，并选择 “控制” > “测试影片”。

    “输出”面板将显示来自 Animal 类构造函数的文本。

注意	如果需要修改 “影片剪辑”的链接属性，可以右击文档库中的元件，并从上下文菜单中选择 “属性”或 “链接”。

编译和导出类

默认情况下，SWF 文件使用的类在该 SWF 文件的第 1 帧上打包并导出。您也可以指定在一个不同的帧上打包并导出类。在某些情况下这将很有用，例如，如果 SWF 文件使用了许多类（如组件），需要很长时间来下载。如果类在第 1 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。通过指定时间轴中后面的某一帧，在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。

若要为 Flash 文档指定类的导出帧，请执行以下操作：

1. 选择“文件”>“新建”，然后选择“Flash 文档”。将新文档保存为 **exportClasses.fla**。
2. 将默认图层重命名为 **content**，将 “进度条”组件从 “组件”面板拖动到舞台上，将此实例命名为 **my_pb**。
3. 创建一个新图层，将其拖动到 content 图层之上，然后将其重新命名为 **actions**。
4. 将以下 ActionScript 代码添加到主时间轴上 actions 图层的第 1 帧中：

```
my_pb.indeterminate = true;
```

5. 在 actions 图层的第 2 帧上创建一个新的关键帧，并添加以下 ActionScript 代码：

```
var classesFrame:Number = 10;
if (_framesloaded < classesFrame) {
  trace(this.getBytesLoaded() + " of " + this.getBytesTotal() + " bytes
  loaded");
  gotoAndPlay(1);
} else {
  gotoAndStop(classesFrame);
}
```

6. 在 actions 图层的第 10 帧上创建一个新的关键帧，并添加以下 ActionScript 代码：

```
stop();
```

7. 在 content 图层的第 10 帧上创建一个新的关键帧，并将一些组件拖动到舞台上。
8. 在 “库”面板中右击每个组件（“进度条”除外），并从上下文菜单中选择 “链接”，以启动 “链接属性”对话框。
9. 在 “链接属性”对话框中，确保选择 “为 ActionScript 导出”，并取消选择 “在第一帧导出”复选框，然后单击 “确定”。
10. 选择 “文件” > “发布设置”。
11. 在 “发布设置”对话框中，选择 “Flash”选项卡。
12. 单击 “ActionScript 版本”弹出菜单旁边的 “设置”按钮，打开 “ActionScript 设置”对话框。
13. 在 “导出用于类的帧”文本框中，输入要导出类代码的帧号 （第 10 帧）。

    如果时间轴中没有指定的帧，则发布 SWF 文件时将出现一条错误消息。
14. 单击 “确定”关闭 “ActionScript 设置”对话框，然后单击 “确定”关闭 “发布设置”对话框。
15. 选择 “控制” > “测试影片”对该 Flash 文档进行测试。如果组件加载过快，请从 SWF 文件中选择 “查看” > “模拟下载”。Flash 以较慢的速度模拟 Flash 文档的下载过程，允许您在下载类文件时查看进度条组件动画。

有关 ASO 文件的更多信息，请参见第 212 页的 “使用 ASO 文件”。

使用 ASO 文件

在编译过程中，Flash 有时会在默认全局类路径目录下的 /aso 子目录下创建扩展名为 .aso 的文件（请参见第 175 页的 “关于设置和修改类路径”）。.aso 扩展名代表 ActionScript object (ASO)。Flash 将为每个被隐式或显式导入并成功编译的 ActionScript 2.0 文件生成一个 ASO 文件。该文件包含从相关的 ActionScript (AS) 文件中产生的字节码。因此，这些文件包含类文件的编译形式（字节码）。

只有在出现以下情况时 Flash 才需要重新生成 ASO 文件：

- 对应的 AS 文件已被修改。
- 那些包含了由对应的 ActionScript 文件导入或使用的定义的 ActionScript 文件已被修改。
- 对应的 ActionScript 文件中包含的 ActionScript 文件已被修改。

编译器创建 ASO 文件以便进行缓存。您可能会注意到第一个编译过程比后面的编译过程要长。这是因为只有更改过的 AS 文件才被重新编译到 ASO 文件中。对于未更改的 AS 文件，编译器直接从 ASO 文件读取已编译过的字节码而不是重新编译 AS 文件。

ASO 文件格式是仅为内部使用而开发的中间格式。它不是文档文件格式，而且不会重新进行分布。

如果您遇到 Flash 编译已编辑过的文件的早期版本的问题，请删除 ASO 文件然后重新编译。如果您计划删除 ASO 文件，请在 Flash 没有执行其它操作（如检查语法或导出 SWF）时再删除这些文件。

删除 ASO 文件：

如果希望在编辑 FLA 文件时删除 ASO 文件，请在创作环境中选择以下方法之一：

- 选择 “控制” > “删除 ASO 文件”，以删除 ASO 文件并继续进行编辑。
- 选择 “控制” > “删除 ASO 文件并测试影片”，以删除 ASO 文件并测试应用程序。

如果正在 “脚本”窗口中编辑 ActionScript 文档：

- 选择 “控制” > “删除 ASO 文件”，以删除 ASO 文件并继续进行编辑。
- 选择“控制”>“删除 ASO 文件和测试项目”，以删除 ASO 文件，然后测试应用程序。

在单个类中可以放置的代码的数量有限：导出的 SWF 文件中类定义的字节码不能大于 32,767 字节。如果字节码大于该限制，则会出现一条警告消息。

您无法预测给定类的字节码表示的大小，但 1,500 行以下的类通常不会超过这个限制。

如果类超过限制，请将某些代码移到另一个类中。一般来说，保持类的相对简短是个良好的面向对象编程惯例。

理解类和作用域

将 ActionScript 代码移动到类中时，您可能不得不更改使用 this 关键字的方式。例如，如果具有使用回调函数的类方法（如 LoadVars 类的 onLoad() 方法），将很难知道 this 关键字引用的是类还是 LoadVars 对象。在这种情况下，可能需要创建一个指向当前类的指针，如下一个示例所示。

若要理解范围和外部类文件，请执行以下操作：

1. 选择“文件”>“新建”，再选择“ActionScript 文件”，然后单击“确定”。
2. 在“脚本”窗口中键入或粘贴以下代码：

```
/**
  Product 类
  Product.as
*/
class Product {
  private var productsXml:XML;
  // 构造函数
  // targetXmlStr - 字符串，包含 XML 文件的路径
  function Product(targetXmlStr:String) {
    /* 创建对当前类的本地引用。
       即使在 XML 的 onLoad 事件处理函数内，您
       也可以引用当前类，而不仅仅是引用 XML 包。
    */
    var thisObj:Product = this;
    // 创建一个局部变量，该变量用来加载 XML 文件。
    var prodXml:XML = new XML();
    prodXml.ignoreWhite = true;
    prodXml.onLoad = function(success:Boolean) {
      if (success) {
        /* 如果 XML 加载和解析成功，
           请将该类的 productsXml 变量设置为已解析
           的 XML 文档并调用 init 函数。
        */
        thisObj.productsXml = this;
        thisObj.init();
      } else {
        /* 加载 XML 文件时出错。 */
        trace("error loading XML");
      }
    };
    // 开始加载 XML 文档。
    prodXml.load(targetXmlStr);
  }
  public function init():Void {
    // 显示 XML 包。
    trace(this.productsXml);
  }
}
```

由于您尝试引用 onLoad 处理函数内的私有成员变量，因此 this 关键字实际上引用的是 prodXml 实例，而不是您期望的 Product 类。由于这个原因，您必须创建一个指向本地类文件的指针，才能从 onLoad 处理函数直接引用该类。现在就可以将此类用于 Flash 文档了。

3. 将上面的 ActionScript 代码保存为 **Product.as**。
4. 在同一目录下创建一个名为 **testProduct.fla** 的新 Flash 文档。
5. 选择主时间轴上的第 1 帧。
6. 在 “动作”面板中键入以下 ActionScript 代码：

```
var myProduct:Product = new Product("http://www.helpexamples.com/
  crossdomain.xml");
```

7. 选择 “控制” > “测试影片”，在测试环境中测试此代码。

“输出”面板中将出现指定 XML 文档中的内容。

使用这些类时将遇到的另一种类型的作用域就是静态变量和静态函数。 static 关键字指定某个变量或函数只为某个类创建一次，而不是在该类的每个实例中都创建。您可以通过使用语法 someClassName.username 在不创建类的实例的情况下访问静态类成员。有关静态变量和静态函数的更多信息，请参见第 181 页的 “关于公共、私有与静态方法和属性（成员）”和第 186 页的 “使用类成员”。

静态变量的另一个好处是在变量的范围结束之后其值不会丢失。下面的示例演示了可以如何使用 static 关键字创建计数器以记录 Flash 创建了多少个类实例。因为 numInstances 变量是静态变量，所以该变量只为整个类创建一次，而不是为每个实例都创建。

若要使用 static 关键字，请执行以下操作：

1. 选择 “文件” > “新建”，再选择 “ActionScript 文件”，然后单击 “确定”。
2. 在 “脚本”窗口中键入以下代码：

```
class User {
  private static var numInstances:Number = 0;
  public function User() {
    User.numInstances++;
  }
  public static function get instances():Number {
    return User.numInstances;
  }
}
```

上面的代码定义了一个 User 类，用来跟踪构造函数被调用的次数。私有、静态变量 User.numInstances 将在构造函数方法中递增。

3. 将文档保存为 **User.as**。
4. 选择 “文件” > “新建”，再选择 “Flash 文档”，创建一个新的 FLA 文件，然后将该 FLA 文件与 User.as 保存在同一目录中。

5. 将下面的 ActionScript 代码键入到时间轴的第 1 帧上：

```
trace(User.instances); // 0
var user1:User = new User();
trace(User.instances); // 1
var user2:User = new User();
trace(User.instances); // 2
```

第一行代码调用静态 `instances()` getter 方法，该方法将返回私有静态变量 `numInstances` 的值。其余代码将创建 User 类的新实例，并显示 `instances()` getter 方法返回的当前值。

6. 选择 “控制” > “测试影片” 对该文档进行测试。

有关在类中使用 `this` 关键字的信息，请参见第 194 页的 “关于在类中使用 this 关键字”。

关于顶级类和内置类

除了 ActionScript 的核心语言元素和构造（例如 `for` 和 `while` 循环）以及本手册中前面所述的原始数据类型（number、string 和 Boolean）（请参见第 275 页的第 10 章 “数据和数据类型”和第 63 页的第 4 章 “语法和语言基础知识”）之外，ActionScript 还提供了若干内置类（复杂数据类型）。这些类提供了多种脚本撰写功能。在前面几章中我们已经使用了顶级类和其它内置类，这些类是 ActionScript 语言一部分，在以后的各章中我们将继续使用它们。Flash 自带了许多类，可以用于在 SWF 文件中创建交互性和功能性，甚至还可以使用它们构建复杂的应用程序。例如，可以使用 Math 类在应用程序中执行公式。或者，可以使用 BitmapData 类创建像素和脚本化动画。

第 217 页的 “顶级类”中列出的顶级类被写入到 Flash Player 中。在 “动作” 工具栏中，这些类位于 “ActionScript 2.0 类” 目录中。这些顶级类中有一些是基于 ECMAScript (ECMA-262) 第 3 版语言规范的，称为核心 ActionScript 类。核心类的示例包括 Array、Boolean、Date 和 Math 类。有关包的更多信息，请参见第 166 页的 “使用包”。

可以在硬盘上找到安装的 ActionScript 类。类文件夹位于：

- Windows：Hard Disk\Documents and Settings\ 用户 \Local Settings\Application Data\Macromedia\Flash 8\ 语言 \Configuration\Classes。
- Macintosh：Hard Disk/Users/ 用户 /Library/Application Support/Macromedia/Flash 8/ 语言 /Configuration/Classes。

有关此结构的更多信息，请查阅此目录中的 Read Me 文档。

要理解核心 ActionScript 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：客户端 JavaScript 类提供对客户端环境（Web 浏览器和网页内容）的控制，而 Flash 专用类提供对 Flash 应用程序外观和行为的运行时控制。

其余的内置 ActionScript 类是 Macromedia Flash 和 Flash Player 对象模型专用的。这些类的示例包括 Camera、MovieClip 和 LoadVars 类。还有其它一些类被组织到包（如 flash.display）中。所有这些类有时称为内置 类（可用于向应用程序中添加功能的预定义类）。

后面各节将介绍内置 ActionScript 类，并介绍使用这些内置类可执行的基本任务。有关在面向对象的编程中使用类和对象的概述，请参见第 223 页的 “关于使用内置类”。使用这些类的代码示例包含在整个《学习 Flash 中的 ActionScript 2.0》手册中。

有关语言元素（如常量、运算符和指令）的信息，请参见第 63 页的第 4 章 “语法和语言基础知识”。

有关顶级类和内置类的更多信息，请参见以下主题：

其它语言元素

除了类之外，组成 ActionScript 的还有一些其它语言元素。这些元素包括指令、常量、全局函数、全局属性、运算符和语句。有关如何使用各种语言元素的信息，请参见以下主题：

- 第 4 章 “语法和语言基础知识”
- 第 5 章 “函数和方法”

在《ActionScript 2.0 语言参考》的以下各节中可获得这些语言元素的列表：

- 编译器指令
- 常数
- 全局函数
- 全局属性
- 运算符
- 语句

顶级类

顶级类包含 ActionScript 类和全局函数，其中的许多类和函数为应用程序提供了核心功能。核心类 是从 ECMAScript 直接借用的类，包括 Array、Boolean、Date、Error、Function、Math、 Number、 Object、 String 和 System。有关各个类的更多信息，请参见下表。

注释	CustomActions 和 XMLUI 类只能在 Flash 创作环境中使用。

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 `_accProps` 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 Accessibility。
Array	Array 类表示 ActionScript 中的数组，并且所有数组对象都是该类的实例。Array 类提供用于处理数组对象的方法和属性。请参见 Array。
AsBroadcaster	提供可以添加到其它对象中的事件通知和侦听器管理功能。请参见 AsBroadcaster。
Boolean	Boolean 类是布尔值 （`true` 或 `false`）的包装。请参见 Boolean。
Button	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 Button。请注意，内置 Button 类与 Button 组件类不同，而与第 2 版组件 Button 关联。
Camera	Camera 类用于访问用户的摄像头 （如果安装了此设备）。与 Flash Communication Server 一起使用时， SWF 文件可以捕获、广播和录制来自用户摄像头的图像和视频。请参见 Camera。
Color	通过 Color 类，您可以设置影片剪辑实例的 RGB 颜色值和颜色转换，并可以在设置后检索这些值。 Color 类在 Flash Player 8 中已否决，转而使用 ColorTransform 类。有关颜色转换的信息，请参见 ColorTransform (flash.geom.ColorTransform)。
ContextMenu	ContextMenu 类可用于在运行时控制 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 `menu` 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 ContextMenu。
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 ContextMenuItem。
CustomActions	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 CustomActions。
Date	Date 类显示如何在 ActionScript 中表示日期和时间，它支持处理日期和时间的操作。 Date 类还提供了从操作系统中获取当前日期和时间的方法。请参见 Date。

类	说明
Error	Error 类包含关于脚本中出现的运行时错误的信息。通常使用 `throw` 语句来生成错误条件，然后可以使用 `try..catch..finally` 语句来处理该错误条件。请参见 Error。
Function	Function 类是所有 ActionScript 函数 （包括 ActionScript 的固有函数和您定义的函数）的类表示形式。请参见 Function。
Key	Key 类提供用于获取有关键盘和按键信息的方法和属性。请参见 Key。
LoadVars	LoadVars 类允许您在 SWF 文件和服务器之间以名称 / 值对的形式传输变量。请参见 LoadVars。
LocalConnection	LocalConnection 类用于开发 SWF 文件，这些文件无需使用 `fscommand()` 方法或 JavaScript 即可相互发送指令。请参见 LocalConnection。
Math	使用 Math 类能方便地访问常见的数学常数，该类还提供了多个常用的数学函数。 Math 类的所有属性和方法均为静态的，必须使用以下语法来调用：`Math.method(parameter)` 或 `Math.constant`。请参见 Math。
Microphone	Microphone 类用于访问用户的麦克风 （如果安装了此设备）。与 Flash Communication Server 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 Microphone。
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）。请参见 Mouse。
MovieClip	SWF 文件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 MovieClip。
MovieClipLoader	此类用于实现在 SWF、 JPEG、 GIF 和 PNG 文件加载到影片剪辑实例中时提供状态信息的侦听器回调。请参见 MovieClipLoader。
NetConnection	NetConnection 可建立本地流连接，以便从 HTTP 地址或本地文件系统播放 Flash 视频 (FLV) 文件。请参见 NetConnection。
NetStream	NetStream 类控制来自本地文件系统或 HTTP 地址的 FLV 文件的回放。请参见 NetStream。
Number	Number 类是原始数字数据类型的包装。请参见 Number。
Object	Object 类位于 ActionScript 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 Object。
PrintJob	PrintJob 类可用于打印 SWF 文件的内容，包括动态呈现的内容和多页文档。请参见 PrintJob。
Selection	利用 Selection 类可以设置和控制插入点所在的文本字段 （具有焦点的文本字段）。请参见 Selection。

类	说明
SharedObject	SharedObject 类在客户端计算机上提供了永久的本地数据存储 （类似于 cookie）。此类提供了客户端计算机上对象之间的实时数据共享。请参见 SharedObject。
Sound	Sound 类提供对 SWF 文件中的声音的控制。请参见 Sound。
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。它还报告调整舞台大小的事件。请参见 Stage。
String	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 String。
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 System。
TextField	TextField 类提供对动态和输入文本字段的控制，例如检索格式信息、调用事件处理函数和更改属性 （如 Alpha 或背景颜色）。请参见 TextField。
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 TextFormat。
TextSnapshot	TextSnapshot 对象用于访问和布置影片剪辑中的静态文本。请参见 TextSnapshot。
Video	Video 类用于播放 SWF 文件中的视频对象。可将此类用于 Flash Communication Server 以显示 SWF 文件中的实时视频流，或者用于 Flash 中以播放 Flash 视频 (FLV) 文件。请参见 Video。
XML	此类提供用于处理 XML 对象的方法和属性。请参见 XML。
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 XMLNode。
XMLSocket	XMLSocket 类可用于在服务器计算机与运行 Flash Player 的客户端之间创建永久套接字连接。客户端套接字实现了等待时间较短的数据传输，例如，实时聊天应用程序所需的快速数据传输。请参见 XMLSocket。
XMLUI	XMLUI 对象可以与用作 Flash 创作工具扩展功能 （比如 “行为”、“命令”、“效果” 和 “工具”）的自定义用户界面的 SWF 文件进行通讯。请参见 XMLUI。

flash.display 包

flash.display 包中包含可用于构建可视显示内容的 BitmapData 类。

类	说明
BitmapData	BitmapData 类可用于在文档中创建可任意调整大小的透明或不透明位图图像，并且可以在运行时以各种方式对它们进行处理。请参见 BitmapData (flash.display.BitmapData)。

flash.external 包

flash.external 包可用来与使用 ActionScript 代码的 Flash Player 容器进行通信。例如，如果您在 HTML 页中嵌入一个 SWF 文件，则该 HTML 页就是一个容器。可以与使用 ExternalInterface 类和 JavaScript 的 HTML 页进行通信。也称为外部 API。

类	说明
ExternalInterface	ExternalInterface 类是外部 API，它是一个子系统，帮助实现 ActionScript 与 Flash Player 容器 （如使用 JavaScript 的 HTML 页）或使用 Flash Player 的桌面应用程序之间的通信。请参见 ExternalInterface (flash.external.ExternalInterface)。

flash.filters 包

flash.filters 包中包含用于 Flash Player 8 中的位图滤镜效果的类。滤镜可用来为 Image 和 MovieClip 实例实现丰富的视觉效果（如模糊、斜角、发光和投影）。有关每个类的详细信息，请参见下表中提供的交叉参考。

类	说明
BevelFilter	BevelFilter 类可用来向影片剪辑实例中添加斜角效果。请参见 BevelFilter (flash.filters.BevelFilter)。
BitmapFilter	BitmapFilter 类是所有滤镜效果的基类。请参见 BitmapFilter (flash.filters.BitmapFilter)。
BlurFilter	BlurFilter 类可用来对影片剪辑实例应用模糊效果。请参见 BlurFilter (flash.filters.BlurFilter)。
ColorMatrixFilter	ColorMatrixFilter 类可用来对输入图像上每个像素的 ARGB 颜色和 alpha 值应用 4 x 5 矩阵变形。应用变形后，可以得到一组新的 ARGB 颜色和 alpha 值。请参见 ColorMatrixFilter (flash.filters.ColorMatrixFilter)。
ConvolutionFilter	ConvolutionFilter 类可用来实现矩阵盘绕滤镜效果请参见 ConvolutionFilter (flash.filters.ConvolutionFilter)。

类	说明
DisplacementMapFilter	通过 DisplacementMapFilter 类可使用指定图像 （置换图像）的像素值在空间上置换向其应用滤镜的原始实例 （影片剪辑）。请参见 DisplacementMapFilter (flash.filters.DisplacementMapFilter)。
DropShadowFilter	DropShadowFilter 类可用来向影片剪辑中添加投影效果。请参见 DropShadowFilter (flash.filters.DropShadowFilter)。
GlowFilter	GlowFilter 类可用来向影片剪辑中添加发光效果。请参见 GlowFilter (flash.filters.GlowFilter)。
GradientBevelFilter	GradientBevelFilter 类可用来对影片剪辑应用渐变斜角效果。请参见 GradientBevelFilter (flash.filters.GradientBevelFilter)。
GradientGlowFilter	GradientGlowFilter 类可用来对影片剪辑应用渐变发光效果。请参见 GradientGlowFilter (flash.filters.GradientGlowFilter)。

flash.geom 包

flash.geom 包中包含 geometry 类，如点、矩形和变形矩阵。这些类支持 BitmapData 类和位图缓存功能。有关每个类的详细信息，请参见下表中提供的交叉参考。

类	说明
ColorTransform	ColorTransform 类可用来以算术方法设置实例的 RGB 颜色值和颜色转换。设置后就可以检索这些值了。请参见 ColorTransform (flash.geom.ColorTransform)。
Matrix	表示一个转换矩阵，它确定如何将一个坐标空间的点映射到另一个坐标空间。请参见 Matrix (flash.geom.Matrix)。
Point	Point 对象表示二维坐标系统中的某个位置，其中 x 表示水平轴， y 表示垂直轴。请参见 Point (flash.geom.Point)。
Rectangle	Rectangle 类用于创建和修改矩形对象。请参见 Rectangle (flash.geom.Rectangle)。
Transform	收集有关应用于对象实例的颜色转换和坐标处理的数据。请参见 Transform (flash.geom.Transform)。

flash.net 包

flash.net 包中包含一些类，这些类可用来在用户的计算机和服务器之间上传和下载一个或多个文件。有关每个类的详细信息，请参见下表中提供的交叉参考。

类	说明
FileReference	FileReference 类可用来在用户计算机和服务器之间上载和下载一个或多个文件。请参见 FileReference (flash.net.FileReference)。
FileReferenceList	FileReferenceList 类可用来将用户的计算机中的一个或多个文件上传到服务器。请参见 FileReferenceList (flash.net.FileReferenceList)。

flash.text 包

flash.text 包中包含 TextRenderer 类，通过该类可在 Flash Player 8 中使用高级消除锯齿功能。

类	说明
TextRenderer	此类为 Flash Player 8 提供了高级消除锯齿功能。请参见 TextRenderer (flash.text.TextRenderer)。

mx.lang 包

mx.lang 包中包含 Locale 类，通过该类可使用多语言文本。

类	说明
Locale	通过此类可控制 SWF 文件中的多语言文本显示方式。请参见 Locale (mx.lang.Locale)。

System 和 TextField 包

System 包中包含 capabilities、IME 和 security 类。这些类可以处理可能会影响 Flash Player 中的应用程序的设置。有关每个类的详细信息，请参见下表中提供的交叉参考。

类	说明
capabilities	Capabilities 类可以确定承载 SWF 文件的系统和 Flash Player 的能力。这样，您就可以针对不同的格式自定义内容了。请参见 capabilities (System.capabilities)。
IME	IME 类可用来直接处理运行在客户端计算机上的 Flash Player 应用程序内的操作系统输入法编辑器 (IME)。请参见 IME (System.IME)。
security	Security 类包含的方法可指定不同域中的 SWF 文件如何相互通信。请参见 security (System.security)。

TextField 包中包含 StyleSheet 类，可以使用该类将 CSS 样式应用于文本。

类	说明
StyleSheet	使用 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。请参见 StyleSheet (TextField.StyleSheet)。

关于使用内置类

在面向对象的编程 (OOP) 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。有关类和其它面向对象的编程概念的信息，请参见以下几节：

- 第 168 页的 “面向对象的编程基础知识”
- 第 170 页的 “编写自定义类文件”

Flash 8 有许多内置类，您可以在代码中使用这些内置类（请参见第 215 页的 “关于顶级类和内置类”），利用这些内置类可以方便地向应用程序添加交互功能。若要使用由内置类定义的属性和方法，您通常需要先创建该类的一个实例 （具有静态成员的类除外）。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，如第 215 页的 “关于顶级类和内置类”中所述。

有关使用 Flash 8 内置类的更多信息，请参见以下主题：

- 第 224 页的 “关于创建内置类的新实例”
- 第 224 页的 “访问内置对象属性”
- 第 225 页的 “关于调用内置对象方法”
- 第 225 页的 “关于类 （静态）成员”
- 第 227 页的 “预加载类文件”
- 第 226 页的 “排除类”

关于创建内置类的新实例

若要创建 ActionScript 类的实例，请使用 `new` 运算符调用该类的构造函数。构造函数与类通常具有相同的名称，并且返回该类的实例，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:Sound = new Sound();
```

在某些情况下，您无需创建类的实例即可使用其属性和方法。有关更多信息，请参见第 225 页的 “关于类 （静态）成员”。

访问内置对象属性

使用点 (.) 运算符可以访问对象中的属性值。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，`my_obj` 是对象，而 `firstName` 是属性：

```
my_obj.firstName
```

下面的代码创建一个新的 Array 对象，然后显示其 `length` 属性：

```
var my_array:Array = new Array("apples", "oranges", "bananas");
trace(my_array.length); // 3
```

也可以使用数组访问运算符 (`[]`) 来访问对象的属性，例如使用数组访问运算符进行调试。下面的示例对一个对象执行循环，以显示它的各个属性。

循环对象内容：

1. 创建一个新的 Flash 文档，并将它保存为 **forin.fla**。
2. 将下面的 ActionScript 添加到主时间轴的第 1 帧中：

```
var results:Object = {firstName:"Tommy", lastName:"G", age:7, avg:0.336,
  b:"R", t:"L"};
for (var i:String in results) {
    trace("the value of [" + i + "] is: " + results[i]);
}
```

   上面的代码定义了一个名为 results 的新对象，并定义其 `firstName`、`lastName`、`age`、`avg`、`b` 和 `t` 的值。`for..in` 循环跟踪 results 对象中的各个属性并将其值输出到“输出”面板中。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

有关运算符（包括点和数组访问运算符）的更多信息，请参见第 118 页的“关于运算符”。有关方法和属性的更多信息，请参见第 141 页的第 5 章“函数和方法”。有关使用内置 MovieClip 类的属性的示例，请参见第 313 页的第 11 章“使用影片剪辑”。有关使用 TextField、String、TextRenderer 和 TextFormat 类的属性的示例，请参见第 341 页的第 12 章“使用文本和字符串”。

关于调用内置对象方法

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：

```
var my_sound:Sound = new Sound(this);
my_sound.setVolume(50);
```

有关使用内置 MovieClip 类的方法的示例，请参见第 313 页的第 11 章“使用影片剪辑”。有关使用内置 TextField、String、TextRenderer 和 TextFormat 类的方法的示例，请参见第 341 页的第 12 章“使用文本和字符串”。

关于类（静态）成员

有些内置 ActionScript 类具有类成员（静态成员）。类成员（属性和方法）是通过类名称来访问或调用的，而不是通过类的实例。因此，不创建该类的实例即可使用其属性和方法。

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:Number = Math.max(10, 20);
trace(largerNumber); // 20
```

有关 Math 类的静态方法的更多信息以及使用它们的示例，请参见《ActionScript 2.0 语言参考》中的 Math。

排除类

为减小 SWF 文件的大小，您可能要从编译过程中排除一些类，但仍能在进行类型检查时访问和使用这些类。例如，如果您正在开发一个使用多个 SWF 文件或共享库的应用程序，特别是在这些文件或共享库要访问许多相同的类时，就可以采用此方法。排除类有助于避免复制这些文件中的类。

有关排除类的更多信息，请参见以下主题：

- 第 227 页的 “预加载类文件”

若要从编译过程中排除类，请执行以下操作：

1. 创建一个新的 XML 文件。
2. 将该 XML 文件命名为 **FLA_filename_exclude.xml**，其中 FLA_filename 是去掉扩展名后 FLA 文件的名称。

   例如，如果 FLA 文件是 sellStocks.fla，则 XML 文件名必须是 sellStocks_exclude.xml。
3. 将该文件保存在 FLA 文件所在的目录中。
4. 请将下面的标签放入 XML 文件中：

```
<excludeAssets>
  <asset name="className1" />
  <asset name="className2" />
</excludeAssets>
```

   在 <asset> 标签中为 name 属性指定的值就是要从 SWF 文件排除的类的名称。根据应用程序的需要添加类名。例如，下面的 XML 文件从 SWF 文件中排除 mx.core.UIObject 和 mx.screens.Slide 类：

```
<excludeAssets>
  <asset name="mx.core.UIObject" />
  <asset name="mx.screens.Slide" />
</excludeAssets>
```

有关预加载类的信息，请参见第 227 页的 “预加载类文件”。

预加载类文件

本节介绍一些在 Flash 8 中预加载和导出类的方法（包括 Macromedia Component Architecture 第二版中的组件使用的类）。预加载 会在用户开始与 SWF 文件进行交互之前加载该文件的一些数据。使用外部类时，Flash 会在 SWF 文件的第一帧上导入类，并首先将此数据加载到 SWF 文件中。由于组件框架也将加载到 SWF 文件第一帧，因此这一点与组件类相似。在构建大型应用程序时，必须导入数据时会延长加载时间，因此必须妥善处理此数据，如下面的过程所示。

有时您可能希望进度条反映所有数据（包括类）的加载进度，由于会首先加载类，因此如果在加载进度条之前加载类，那么在创建进度条或加载动画时可能会遇到问题。因此，应在加载 SWF 文件的其它部分后，使用组件前加载类。

下面的过程演示如何更改将类加载到 SWF 文件中的帧。

为要加载到 SWF 文件中的类选择不同的帧：

1. 选择 “文件” > “发布设置”。
2. 选择 “Flash”选项卡，然后单击 “设置”按钮。
3. 在 “导出用于类的帧”文本框中，键入新帧的帧号，以确定加载类的时间。
4. 单击 “确定”。

在播放头到达选择用于加载类的帧之前不能使用任何类。例如，第 2 版组件功能中需要用到类，因此必须在执行了 “导出用于 ActionScript 2.0 类的帧”之后加载组件。如果在第 3 帧导出，那么在播放头到达第 3 帧并加载数据之前您不能使用那些类的任何相关内容。

如果要预加载使用类的文件，如第 2 版组件类，您必须预加载 SWF 文件中的组件。为此，必须将组件设置为在 SWF 文件中的不同帧导出。默认情况下，UI 组件在 SWF 文件的第 1 帧导出，因此应确保从组件的 “链接”对话框中取消选择 “在第一帧导出”。

如果不在第一帧上加载组件，您可以为 SWF 文件的第一帧创建一个自定义进度条。如果没有为在 “导出用于类的帧”文本框中指定的帧加载类，则请勿在 ActionScript 中引用任何组件或在舞台上包含任何组件。

必须在组件使用的 ActionScript 类之后导出组件。

第 7 章

继承

7

在第 6 章 “类”中，您学习了如何编写类文件，以及类如何帮助您将代码组织到外部文件中。这一章也介绍了可以如何将类文件组织到相关的包中。本章主要介绍如何编写更高级的类以扩展现有类的功能。这是一个非常有用的主题，因为您可能会发现您需要扩展自己的自定义类或现有类，以便添加新的方法和属性。

有关继承的更多信息，请参见第 229 页的 “关于继承”。有关方法和属性的更多信息，请参见第 141 页的第 5 章 “函数和方法”。

有关继承的更多信息，请参见以下主题：

关于继承

在第 6 章 “类”中，您已经看到了可以怎样创建类文件以创建自己的自定义数据类型。学习怎样创建自定义类文件部分向您介绍了怎样将代码从时间轴移入外部文件。通过将代码移入外部文件，将使编辑代码变得更为容易。您既然已熟悉了创建自己的自定义类的基础知识，现在，您将学习一种面向对象的编程 (OOP) 技术，叫做创建子类 或扩展类，通过该技术可以基于现有的类创建新类。

OOP 的好处之一就是可以创建类的子类。子类继承超类 的所有属性和方法。例如，如果扩展 MovieClip 类（或为其创建子类），就是创建一个自定义类来扩展 MovieClip 类。子类继承 MovieClip 类的所有属性和方法。或者还可以创建一组从自定义超类扩展的类。例如，Lettuce 类可以扩展自 Vegetable 超类。

子类一般定义一些可以在应用程序中使用的更多方法和属性，因此说它是对超类的扩展。子类还可以重写（自己定义）从超类继承的方法。如果某个子类重写了从其超类继承的方法，则您不能再在该子类中访问对应的超类定义。以上规则的唯一例外就是，在子类的构造函数内部可以使用 `super` 语句调用超类的构造函数。有关重写的更多信息，请参见第 233 页的 “重写方法和属性”。

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 子类。使用子类可以重复使用代码，从而您不必重新创建两个类共有的任何代码，而只需对现有类加以扩展。另一个子类（比如 Siamese 类）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。

继承和创建子类在较大应用程序中非常有用，因为使用它们可创建一系列可以共享功能的相关类。例如，可以创建一个 Employee 类来定义公司内一般员工的基本方法和属性。然后可以创建一个叫做 Contractor 的新类，它扩展了 Employee 类并继承了其所有的方法和属性。Contractor 类可以添加自己特定的方法和属性，还可以重写在 Employee 超类中定义的方法和属性。然后可以创建一个叫做 Manager 的新类，该类也扩展了 Employee 类并定义了更多方法和属性，如，`hire()`、`fire()`、`raise()` 和 `promote()`。甚至还可以扩展子类（如 Manager）和创建一个叫做 Director 的新类，后者也可以添加新方法或重写现有的方法。

每次扩展现有类时，新类都将继承子类的所有当前方法和属性。如果每个类不相关，则必须在每一个独立类文件中编写每种方法和属性，即使在同级类中的功能相同时也是如此。如果相似的逻辑在多个文件中改变，则不仅要花费太多的时间进行编码，而且还要调试应用程序和维护项目。

在 ActionScript 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或者扩展一个接口。有关使用 `extends` 关键字的更多信息，请参见第 230 页的 “关于在 Flash 中编写子类”和第 231 页的 “关于编写子类”。有关 `extends` 关键字的其它信息，请参见《ActionScript 2.0 语言参考》中的 extends 语句。

关于在 Flash 中编写子类

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。您可以扩展自己的自定义类，就像扩展许多核心和 Flash Player ActionScript 类那样。不能扩展 TextField 类或静态类，例如 Math、Key 和 Mouse 类。

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，请使用下面的语法：

```
class SubClass extends SuperClass {}
```

在 SubClass 中指定的类将继承 SuperClass 中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），如下所示：

```
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。

有关扩展类的更多信息，请参见以下主题：

- 第 231 页的 “关于编写子类”
- 第 233 页的 “重写方法和属性”

有关私有成员的更多信息，请参见第 181 页的 “关于公共、私有与静态方法和属性（成员）”。有关创建子类的示例，请参见第 232 页的 “示例：扩展 Widget 类”。

关于编写子类

下面的代码定义了自定义类 JukeBox，它是对 Sound 类的扩展。它定义一个名为 song_arr 的数组和一个名为 playSong() 的方法。此方法播放歌曲并调用继承自 Sound 类的 loadSound() 方法。

```
class JukeBox extends Sound {
  public var song_arr:Array = new Array("beethoven.mp3", "bach.mp3",
  "mozart.mp3");
  public function playSong(songID:Number):Void {
    super.loadSound(song_arr[songID], true);
  }
}
```

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直接超类的构造函数的调用（不带参数），并将此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）在 ActionScript2.0 中是不允许的。但是，如果使用单个 extends 语句，则类便可以有效地从多个类继承，如下面的示例所示：

```
// 不允许
class C extends A, B {} // ** 错误：一个类不能扩展多个类。

// 允许
class B extends A {}
class C extends B {}
```

您还可以使用接口来实现形式有限的多重继承。有关接口的更多信息，请参见第 241 页的第 8 章 “接口”。有关创建子类的示例，请参见第 232 页的 “示例：扩展 Widget 类”。有关 super 的其它信息，请参见《ActionScript 2.0 语言参考》中 super 语句。

示例：扩展 Widget 类

类成员将传播到定义这些成员的超类的子类。下一个示例演示了可以怎样创建 Widget 类，并通过编写一个名为 SubWidget 的类来扩展它（创建子类）。

若要创建 Widget 类和 SubWidget 子类，请执行以下操作：

1. 创建一个新的 ActionScript 文件，并保存为 **Widget.as**。
2. 在新文档中添加以下代码：

```
class Widget {
  public static var widgetCount:Number = 0;
  public function Widget() {
    Widget.widgetCount++;
  }
}
```

3. 保存对 ActionScript 文件的更改。
4. 创建一个新的 ActionScript 文件，并在 Widget 类所在的同一目录下将其保存为 **SubWidget.as**。
5. 在 SubWidget.as 中，将下面的代码键入到“脚本”窗口中：

```
class SubWidget extends Widget {
  public function SubWidget() {
    trace("Creating subwidget #" + Widget.widgetCount);
  }
}
```

6. 保存对 SubWidget.as 的更改。
7. 创建一个新的 FLA 文件，并在上述 ActionScript 类文件所在的目录下将其保存为 **subWidgetTest.fla**。
8. 在 subWidgetTest.fla 文件中，将下面的代码键入到主时间轴的第 1 帧中：

```
var sw1:SubWidget = new SubWidget();
var sw2:SubWidget = new SubWidget();
trace("Widget.widgetCount = " + Widget.widgetCount);
trace("SubWidget.widgetCount = " + SubWidget.widgetCount);
```

   上面的代码创建了 SubWidget 类的两个实例：sw1 和 sw2。对 `SubWidget` 构造函数的每个调用都输出静态 `Widget.widgetCount` 属性的当前值。由于 SubWidget 类是 Widget 类的子类，因此可以通过 SubWidget 类来访问 `widgetCount` 属性。编译器会在字节码（而非 ActionScript 文件中）将引用重写为 `Widget.widgetCount`。如果试图不通过 Widget 或 SubWidget 类的实例（如 sw1 或 sw2）来访问静态 `widgetCount` 属性，编译器会引发错误。
9. 保存对文档的更改。

10. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

“输出” 面板将显示以下输出结果：

```
Creating subwidget #1
Creating subwidget #2
Widget.widgetCount = 2
SubWidget.widgetCount = 2
```

之所以会看到此输出结果是因为，即便从未显式调用 Widget 类的构造函数，SubWidget 类的构造函数也会为您调用该函数。这会引起 Widget 类的构造函数递增 Widget 类的静态 `widgetCount` 变量。

ActionScript 2.0 编译器可以在类定义中解析静态成员引用。

如果未指定 `Widget.widgetCount` 属性的类名称，而只是引用 `widgetCount`，则 ActionScript 2.0 编译器将解析对 `Widget.widgetCount` 的引用并正确导出该属性。类似地，如果将该属性作为 `SubWidget.widgetCount` 引用，编译器会将引用改写 （在字节码中，而不是在 AS 文件中）为 `Widget.widgetCount`，因为 SubWidget 是 Widget 类的子类。

小心	如果试图使用 sw1 或 sw2 实例通过 Widget 类访问静态 `widgetCount` 变量， Flash 将生成一个错误，指示只能通过类直接访问静态成员。

为了获得最好的代码可读性，Macromedia 建议在代码中始终使用对静态成员变量的显式引用，如上一个示例所示。使用显式引用意味着可以轻松地确定静态成员的定义驻留在哪里。

重写方法和属性

在子类扩展超类时，子类继承超类的所有方法和属性。使用类和扩展类的优势之一是，不仅能够向现有类提供新功能，而且还能够修改现有的功能。例如，我们看一看在第 232 页的 “示例：扩展 Widget 类”中创建的 Widget 类。可以在超类 (Widget) 中创建一个新方法，然后要么在子类 (SubWidget) 中重写该方法，要么直接使用从 Widget 类中继承的方法。下面的示例演示了可以如何重写类中现有的方法。

若要在子类中重写方法，请执行以下操作：

1. 创建一个新的 ActionScript 文档，并将该文档保存为 **Widget.as**。
2. 在 Widget.as 中，将以下 ActionScript 代码键入到 “脚本” 窗口中。

提示	如果已在前面的示例中创建了 Widget 类，则可通过添加 doSomething() 方法修改现有的代码，如下所示：

```
class Widget {
  public static var widgetCount:Number = 0;
  public function Widget() {
    Widget.widgetCount++;
  }
  public function doSomething():Void {
    trace("Widget::doSomething()");
  }
}
```

3. 保存对 ActionScript 文档的更改。

   Widget 类现在将定义一个构造函数和一个叫做 `doSomething()` 的公共方法。

4. 创建一个名为 **SubWidget.as** 的新 ActionScript 文件，并保存在 Widget.as 所在的同一目录中。

> **提示** 如果在第 232 页的 “示例：扩展 Widget 类”中已创建了 SubWidget 类，就可以直接使用此文件。

5. 在 SubWidget.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class SubWidget extends Widget {
  public function SubWidget() {
    trace("Creating subwidget # " + Widget.widgetCount);
    doSomething();
  }
}
```

6. 保存对 SubWidget.as 的更改。

   注意，SubWidget 类的构造函数将调用在超类中定义的 `doSomething()` 方法。

7. 创建一个新的 Flash 文档，并在 ActionScript 文档所在的同一目录下将其保存为 **subWidgetTest.fla**。

8. 在 subWidgetTest.fla 中，将下面的 ActionScript 键入到主时间轴的第 1 帧上：

```
var sw1:SubWidget = new SubWidget();
var sw2:SubWidget = new SubWidget();
```

9. 保存对 Flash 文档的更改。

10. 选择 “控制” > “测试影片”对该 Flash 文档进行测试。您应在 “输出”面板中看到以下输出结果：

```
Creating subwidget # 1
Widget::doSomething()
Creating subwidget # 2
Widget::doSomething()
```

    此输出结果显示，SubWidget 类的构造函数调用了其超类 (Widget) 的构造函数，这递增了静态 `widgetCount` 属性。SubWidget 的构造函数将跟踪超类的静态属性并调用 `doSomething()` 方法，后者继承自超类。

11. 打开 SubWidget 类并添加一个名为 doSomething() 的新方法。修改类以便让它与以下代码匹配 （添加粗体显示的代码）：

```
class SubWidget extends Widget {
  public function SubWidget() {
    trace("Creating subwidget # " + Widget.widgetCount);
    doSomething();
  }
  public function doSomething():Void {
    trace("SubWidget::doSomething()");
  }
}
```

12. 保存对类文件的更改，然后再次打开 subwidgetTest.fla。

13. 选择“控制”>“测试影片”对文件进行测试。您应在“输出”面板中看到以下输出结果:

```
Creating subwidget # 1
SubWidget::doSomething()
Creating subwidget # 2
SubWidget::doSomething()
```

以上输出结果显示，SubWidget 类的构造函数中的 doSomething() 方法正在调用当前类 （而不是超类）中的 doSomething() 方法。

再次打开 SubWidget 类，修改 SubWidget 类的构造函数以调用超类的 doSomething() 方法 （添加以粗体显示的代码）：

```
  public function SubWidget() {
    trace("Creating subwidget # " + Widget.widgetCount);
    super.doSomething();
  }
```

正如所演示的那样，可以添加 super 关键字以调用超类的 doSomething() 方法，而非当前类中的 doSomething() 方法。有关 super 的其它信息，请参见《ActionScript 2.0 语言参考》中的 super 条目。

14. 保存 SubWidget 类文件和修改过的构造函数，选择 “控制” > “测试影片”以重新发布 Flash 文档。

“输出”面板中将显示 Widget 类的 doSomething() 方法的内容。

在应用程序中使用多态

面向对象的编程允许使用一种名为多态的技术来表达各个类之间的差异，使用这种技术，类可以重写其超类的方法并定义这些方法的专用实现。

例如，您可以从具有 play() 和 sleep() 方法、名为 Mammal 的类开始。然后您可以创建 Cat、Monkey 和 Dog 子类来扩展 Mammal 类。这些子类重写 Mammal 类的 play() 方法，来反映那些特定种类的动物的习性。Monkey 实现悬挂在树上的 play() 方法； Cat 实现对线球猛扑的 play() 方法； Dog 实现捡回球的 play() 方法。因为各个动物的 sleep() 功能是相似的，所以您可以使用超类实现。下面的过程用 Flash 演示了此示例。

若要在应用程序中使用多态，请执行以下操作：

1. 创建一个新的 ActionScript 文档，并保存为 **Mammal.as**。

   此文档是在以下几个步骤中将要创建的几个不同动物类的基类。

2. 在 Mammal.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class Mammal {
  private var _gender:String;
  private var _name:String = "Mammal";

  // 构造函数
  public function Mammal(gender:String) {
    this._gender = gender;
  }

  public function toString():String {
    return "[object " + speciesName + "]";
  }
  public function play():String {
    return "Chase another of my kind.";
  }
  public function sleep():String {
    return "Close eyes.";
  }

  public function get gender():String {
    return this._gender;
  }
  public function get speciesName():String {
    return this._name;
  }
  public function set speciesName(value:String):Void {
    this._name = value;
  }
}
```

上述类定义两个私有变量 _gender 和 _name，它们分别用来存储动物的性别和哺乳动物类型。接下来，定义了 Mammal 构造函数。该构造函数使用单个参数 gender 来设置前面定义的私有 gender 变量它还指定了以下三个附加的公共方法：toString()、play() 和 sleep() 中的每一个都将返回字符串对象。最后的三种方法是哺乳动物的 _gender 和 _name 属性的 getter 和 setter 方法。

3. 保存该 ActionScript 文档。

   此类将作为即将创建的 Cat、Dog 和 Monkey 类的超类。可以使用 Mammal 类的 toString() 方法来显示任何 Mammal 实例（或任何扩展了 Mammal 类的实例）的字符串表示形式。

4. 创建一个新的 ActionScript 文件，并在您在步骤 1 中创建的 Mammal.as 类所在的目录下将其保存为 **Cat.as**。

5. 在 Cat.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class Cat extends Mammal {
  // 构造函数
  public function Cat(gender:String) {
    super(gender);
    speciesName = "Cat";
  }

  public function play():String {
    return "Pounce a ball of yarn.";
  }
}
```

   请注意，您正在重写 Mammal 超类中的 play() 方法。Cat 类只定义两个方法：构造函数和 play() 方法。因为 Cat 类是对 Mammal 类的扩展，所以 Mammal 类的方法和属性都由 Cat 类继承了。有关重写的更多信息，请参见第 233 页的 “重写方法和属性”。

6. 保存对 ActionScript 文档的更改。

7. 创建新的 ActionScript 文档，并在上述两个类文件所在的同一目录下将其保存为 **Dog.as**。

8. 在 Dog.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class Dog extends Mammal {
  // 构造函数
  public function Dog(gender:String) {
    super(gender);
    speciesName = "Dog";
  }

  public function play():String {
    return "Fetch a stick.";
  }
}
```

请注意，Dog 类的构造与 Cat 类非常相似，只是更改了几个值。而且，Dog 类扩展了 Mammal 类，并继承了它所有的方法和属性。Dog 构造函数具有单个参数 gender，并将它传递给 Dog 类的父类 Mammal。`speciesName` 变量也被重写并设置为字符串 `Dog`。还重写了来自父类的 `play()` 方法。

9. 保存对 ActionScript 文档的更改。

10. 在其它文件所在的目录中创建另外一个 ActionScript 文档，并保存为 **Monkey.as**。

11. 在 Monkey.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class Monkey extends Mammal {
  // 构造函数
  public function Monkey(gender:String) {
    super(gender);
    speciesName = "Monkey";
  }

  public function play():String {
    return "Swing from a tree.";
  }
}
```

与上述两个类 Cat 和 Dog 一样，Monkey 类是对 Mammal 类的扩展。Monkey 类的构造函数调用 Mammal 类的构造函数，将 gender 传递给 Mammal 的构造函数，并将 `speciesName` 设置为字符串 `Monkey`。Monkey 类还将重写 `play()` 方法的行为。

12. 保存对 ActionScript 文档的更改。

13. 在创建 Mammal 类的三个子类后，现在要创建一个名叫 **mammalTest.fla** 的新 Flash 文档。

14. 在 mammalTest.fla 中，将下面的 ActionScript 代码键入到主时间轴的第 1 帧中：

```
var mammals_arr:Array = new Array();
this.createTextField("info_txt", 10, 10, 10, 450, 80);
info_txt.html = true;
info_txt.multiline = true;
info_txt.border = true;
info_txt.wordWrap = true;

createMammals()
createReport()

function createMammals():Void {
  mammals_arr.push(new Dog("Female"));
  mammals_arr.push(new Cat("Male"));
  mammals_arr.push(new Monkey("Female"));
  mammals_arr.push(new Mammal("Male"));
}

function createReport():Void {
  var i:Number;
  var len:Number = mammals_arr.length;
  // 使用 Tab 停靠位以 4 列 HTML 文本显示 Mammal 信息。
  info_txt.htmlText = "<textformat tabstops='[110, 200, 300]'>";
  info_txt.htmlText += "<b>Mammal\tGender\tSleep\tPlay</b>";
  for (i = 0; i < len; i++) {
    info_txt.htmlText += "<p>" + mammals_arr[i].speciesName
        + "\t" + mammals_arr[i].gender
        + "\t" + mammals_arr[i].sleep()
        + "\t" + mammals_arr[i].play() + "</p>";
    // trace 语句将调用 Mammal.toString() 方法。
    trace(mammals_arr[i]);
  }
  info_txt.htmlText += "</textformat>";
}
```

mammalTest.fla 代码比前面的类稍微复杂一些。首先它将导入这三个动物类。

15. 保存 Flash 文档，然后选择 “控制” > “测试影片”对该文档进行测试。

您将看到 Mammal 信息显示在 “舞台”上的文本字段中，而且 “输出”面板中显示了以下文本：

```
[object Dog]
[object Cat]
[object Monkey]
[object Mammal]
```

第 8 章

接口

8

在面向对象的编程 (OOP) 中，接口是一个文档，使用它可以声明 （但不是定义）必须出现在类中的方法。当您在开发团队中工作，或者在 Flash 中构建较大的应用程序时，接口在开发过程中非常有用。通过接口，开发人员可以方便地确定 ActionScript 类中的基本方法。在开发人员使用每个接口时，这些方法都必须已实现。

本章将展示几个示例接口，在结束本章时您将能够构建自己的接口文件。如果您对构建类不熟悉，则必须先阅读第 6 章 “类”，然后才能尝试使用本章中的教程和示例。

有关使用接口的更多信息，请参见以下主题：

关于接口

在面向对象的编程中，接口类似于尚未实现 （定义）其方法的类，也就是说，若类未实现方法，就会与接口一样，不会 “做”任何事情。所以说，接口由 “空”方法组成。于是，另一个类可以实现该接口所声明的方法。在 ActionScript 中，接口和对象之间的区别仅在于编译时错误检查和语言规则的强制执行上。

接口不是类；但是由于接口是抽象的，因此在运行时在 ActionScript 中并非总是如此。ActionScript 接口在运行时确实存在以便允许进行类型转换（将现有数据类型更改为不同类型）。 ActionScript 2.0 对象模型不支持多重继承。因此，一个类只能继承自一个父类。此父类可以是核心类或 Flash Player 类，也可以是用户定义 （自定义）的类。可以使用接口来实现有限形式的多重继承，通过多重继承，一个类可以从多个类继承。

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。和 Java 一样， ActionScript 2.0 不允许一个类直接扩展多个类，但允许一个类扩展一个类并实现多个接口。这样，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。

也可将接口看作是用于将两个若没有接口便没有任何关系的类关联起来的 “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。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。任何实现此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。接口就像一个所有类必须遵守的通信协议。

一种实现方式是创建一个类，定义所有这些方法，然后让每个类都从这个超类扩展或继承。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。

ActionScript 接口在运行时确实存在以便允许进行类型转换；请参见第 310 页的第 10 章 “关于转换对象”。接口既不是对象也不是类，但其工作流程类似于使用类。有关类工作流程的更多信息，请参见第 170 页的 “编写自定义类文件”。有关使用接口创建应用程序的教程，请参见第 248 页的 “示例：使用接口”。

有关使用接口的更多信息，请参见以下部分：

关于 interface 关键字

interface 关键字定义接口。接口与类相似，但也有以下重要差异：

- 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。
- 在接口定义中只允许有公共成员，不允许有静态成员和类成员。
- 在接口定义中不允许有 get 和 set 语句。
- 若要使用 interface 关键字，必须在 FLA 文件的 “发布设置”对话框的 “Flash”选项卡中指定 ActionScript 2.0 和 Flash Player 6 或更高版本。

仅支持在外部脚本文件中使用 interface 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。

关于命名接口

接口名称的第一个字母为大写，这与类名称相同。接口名称通常是形容词，如 `Printable`。下面的接口名称 `IEmployeeRecords` 使用大写的首字母和带有混合大小写的相连接的单词

```
interface IEmployeeRecords {}
```

注意 有些开发人员使用大写 “I”开头的接口名称以便将接口名称与类名称区分开来。这是一种好的做法，它让您可以快速区分接口和常规类。

有关命名约定的更多信息，请参见第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。

定义和实现接口

创建接口的过程与创建类相同。与类相同，只能在外部 ActionScript 文件中定义接口。创建接口的工作流程至少包括以下步骤：

- 在外部 ActionScript 类文件中定义一个接口。
- 将接口文件保存到指定的类路径目录 （Flash 查找类的位置）中或保存在应用程序的 FLA 文件所在的目录中
- 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），或基于原始接口创建一个子接口
- 在外部脚本文件中创建一个实现接口的类

接口的声明使用 `interface` 关键字，后面跟有接口名，然后跟有定义接口体的一对花括号 (`{}`)，如下面的示例所示：

```
interface IEmployeeRecords {
  // 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。

有关构造类和接口的相关约定的更多信息，请参见第 651 页的第 19 章 “ActionScript 2.0 的最佳做法和编码约定”。有关创建使用接口的应用程序的教程，请参见第 248 页的 “示例：使用接口”。

例如，以下代码声明一个名为 `IMyInterface` 的接口，其中包含两个方法 `method_1()` 和 `method_2()`。第一个方法 `method1()` 不具有任何参数，而且返回类型指定为 Void（表示它不返回值）。第二个方法 `method2()` 具有一个类型为 String 的参数，其返回类型指定为 Boolean。

若要创建简单接口，请执行以下操作：

1. 创建一个新的 ActionScript 文件，并保存为 **IMyInterface.as**。
2. 将以下 ActionScript 代码键入到 “脚本”窗口中：

```
interface IMyInterface {
  public function method1():Void;
  public function method2(param:String):Boolean;
}
```

3. 保存对 ActionScript 文件的更改。

   为了在应用程序内使用该接口，需要首先创建一个实现此新接口的类。

4. 创建一个新的 ActionScript 文件，并将其保存到 IMyInterface.as 所在的同一目录下，文件名为 **MyClass.as**。
5. 在 MyClass 类文件中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class MyClass {
}
```

   为了指示自定义类 (MyClass) 使用接口 (IMyInterface)，需要使用 `implements` 关键字，该关键字指定类必须定义在实现的接口中声明的所有方法。

6. 修改 MyClass.as 中的 ActionScript 代码 （添加粗体代码），以便让它与以下代码片断匹配：

```
class MyClass implements IMyInterface {
}
```

   将 `implements` 关键字放在类名称之后。

7. 单击 “检查语法”按钮。

   Flash 将在 “输出”面板中显示一个错误，指出 MyClass 必须实现接口 IMyInterface 中的方法 X。之所以看到此错误信息是因为，任何扩展接口的类都必须定义接口文档中列出的每一个方法。

8. 再次修改 MyClass 文档 （添加粗体代码），编写 `method1()` 和 `method2()` 方法的 ActionScript 代码，如下面的代码片断所示：

```
class MyClass implements IMyInterface {
  public function method1():Void {
    // ...
  };
  public function method2(param:String):Boolean {
    // ...
    return true;
  }
}
```

9. 保存 MyClass.as 文档，然后单击 “检查语法”。

“输出”面板上将不再显示任何错误消息或警告，因为现在已经定义了这两个方法。

您创建的类文件不限于在接口文件中定义的公共方法。接口文件仅列出了必须实现的最低限度的方法，以及这些方法的属性和返回类型。实现特定接口的类几乎总是包括其它一些方法、变量和 getter 和 setter 方法。

接口文件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花括号。例如，下面的接口无法编译：

```
interface IBadInterface {
  // 编译器错误。不允许在接口中声明变量。
  public var illegalVar:String;

  // 编译器错误。接口中不允许有函数体。
  public function illegalMethod():Void {
  }

  // 编译器错误。接口中不允许有私有方法。
  private function illegalPrivateMethod():Void;

  // 编译器错误。接口中不允许有 Getter/setter。
  public function get illegalGetter():String;
}
```

有关介绍如何创建复杂接口的教程，请参见第 248 页的 “示例：使用接口”。

接口的命名规则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 197 页的 “关于命名类文件”。

创建作为数据类型的接口

与类相似，接口也定义一种新的数据类型。可以将实现接口的任何类视为由接口定义的类型。这有助于确定给定的对象是否实现了给定的接口。例如，请考虑在下面的示例中创建的接口 IMovable。

若要创建作为数据类型的接口，请执行以下操作：

1. 创建一个新的 ActionScript 文档，在硬盘上将它保存为 **IMovable.as**。
2. 在 IMovable.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
interface IMovable {
  public function moveUp():Void;
  public function moveDown():Void;
}
```

3. 保存对 ActionScript 文件的更改。

4. 创建一个新的 ActionScript 文档，并将其保存到与 IMovable.as 相同的目录中，文件名称为 **Box.as**。

   在此文档中，创建一个 Box 类，以实现在上一步创建的 IMovable 接口。

5. 在 Box.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class Box implements IMovable {
  public var xPos:Number;
  public var yPos:Number;

  public function Box() {
  }

  public function moveUp():Void {
    trace("moving up");
    // 方法定义
  }
  public function moveDown():Void {
    trace("moving down");
    // 方法定义
  }
}
```

6. 保存对 ActionScript 文档的更改。

7. 创建一个名为 **boxTest.fla** 的新 Flash 文档，然后将它保存在上述两个 ActionScript 文档所在的同一目录中。

8. 选择时间轴中的第 1 帧，打开 ActionScript 编辑器，然后将下面的 ActionScript 代码键入到 “动作”面板 （或 “脚本”窗口）中：

```
var newBox:Box = new Box();
```

   此 ActionScript 代码将创建 Box 类的一个实例，您将把这个实例声明为 Box 类型的变量。

9. 保存对 Flash 文档所做的更改，然后选择“控制”>“测试影片”对 SWF 文件进行测试。

   在 Flash Player 7 和更高版本中，可在运行时将表达式转换为接口类型或其它数据类型。与 Java 接口不同，ActionScript 接口存在于运行时（在运行时允许类型转换）。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。有关类型转换的更多信息，请参见第 310 页的第 10 章 “关于转换对象”。

10. 将以下代码添加到 boxTest.fla 中的 ActionScript 代码的结尾：

```
if (IMovable(newBox) != null) {
  newBox.moveUp();
} else {
  trace("box instance is not movable");
}
```

此 ActionScript 代码将检查 `newBox` 实例在对对象调用 `moveUp()` 方法之前是否实现了 IMovable 接口。

11. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对 SWF 文件进行测试。

由于 Box 实例实现了 IMovable 接口，因此将调用 `Box.moveUp()` 方法，并且在 “输出” 面板上将出现文本 “moving up”。

有关转换的更多信息，请参见第 310 页的第 10 章 “关于转换对象”。

了解继承和接口

您可以使用 `extends` 关键字创建接口的子类。对于要为其扩展现有接口 （或创建现有接口的子类）和添加附加方法的较大项目来说，这非常有用。这些方法必须由实现该接口的任何类定义。

扩展接口时需要考虑的一个事项是，如果多个接口文件声明具有相同名称但却具有不同参数或返回类型的函数，您就会在 Flash 中收到错误消息。

下面的示例演示了可以如何使用 `extends` 关键字创建接口文件的子类。

若要扩展接口，请执行以下操作：

1. 创建一个新的 ActionScript 文件，并保存为 **Ia.as**。
2. 在 Ia.as 中，将以下 ActionScript 代码键入到 “脚本” 窗口中：

```
interface Ia {
  public function f1():Void;
  public function f2():Void;
}
```

3. 保存对 ActionScript 文件的更改。
4. 创建新的 ActionScript 文件，并在您在步骤 1 中创建的 Ia.as 文件所在的同一文件夹中将其保存为 **Ib.as**。
5. 在 Ib.as 中，将以下 ActionScript 代码键入到 “脚本” 窗口中：

```
interface Ib extends Ia {
  public function f8():Void;
  public function f9():Void;
}
```

6. 保存对 ActionScript 文件的更改。

7. 创建新的 ActionScript 文件，并在上述两个文件所在的同一目录下将其保存为 **ClassA.as**。
8. 在 ClassA.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class ClassA implements Ib {
  // f1() 和 f2() 均在接口 Ia 中定义。
  public function f1():Void {
  }
  public function f2():Void {
  }

  // f8() 和 f9() 在接口 Ib 中定义，该接口对 Ia 进行了扩展。
  public function f8():Void {
  }
  public function f9():Void {
  }
}
```

9. 保存您的类文件，单击 “脚本”窗口上方的 “检查语法”按钮。

   只要定义了所有四个方法并且这四个方法都与其各自的接口文件中的定义匹配，Flash 就不会生成任何错误消息。

附加	尽管可以使用类实现任意数量的接口，但在 ActionScript 2.0 中，类只能对一个类进行扩展。

如果您想让 ClassA 类实现上例中的多个接口，只需用逗号将接口分开即可。或者，如果某个类扩展了超类并实现了多个接口，则可以使用类似于下面的代码：

```
class ClassA extends ClassB implements Ib, Ic, Id {...}.
```

示例：使用接口

在本示例中，将创建一个可以在许多不同的类之间重复使用的简单接口。

若要构建接口，请执行以下操作：

1. 创建一个新的 ActionScript 文件，并保存为 **IDocumentation.as**。
2. 在 IDocumentation.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
interface IDocumentation {
  public function downloadUpdates():Void;
  public function checkForUpdates():Boolean;
  public function searchHelp(keyword:String):Array;
}
```

3. 保存对 ActionScript 接口文件所做的更改。
4. 在 IDocumentation.as 文件所在的同一目录中创建一个新的 ActionScript 文件，并将此新文件保存为 **FlashPaper.as**。

5. 在 FlashPaper.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
class FlashPaper implements IDocumentation {
}
```

6. 保存对 ActionScript 文件所做的更改。

7. 对您的 ActionScript 类单击 “检查语法”按钮。

   您将看到类似下面的错误消息：

```
**Error** path\FlashPaper.as:Line 1:The class must implement method
  'checkForUpdates' from interface 'IDocumentation'.

     class FlashPaper implements IDocumentation {

Total ActionScript Errors:1  Reported Errors: 1
```

   出现此错误的原因在于，当前的 FlashPaper 类未定义 IDocumentation 接口中定义的任何公共方法。

8. 再次打开 FlashPaper.as 类文件，修改现有的 ActionScript 代码，以便让它与以下代码匹配：

```
class FlashPaper implements IDocumentation {
  private static var __version:String = "1,2,3,4";
  public function downloadUpdates():Void {
  };
  public function checkForUpdates():Boolean {
    return true;
  };
  public function searchHelp(keyword:String):Array {
    return []
  };
}
```

9. 保存对 ActionScript 文件的更改，然后再次单击 “检查语法”。

   这一次在 “输出”面板中不会显示任何错误。

可以向 FlashPaper 类文件添加任意数量的静态、公共或私有变量或方法。接口文件只定义必须出现在实现该接口的任何类内的最低限度的方法。

10. 再次打开 IDocumentation 接口文档，添加下面这行粗体代码（在 `searchHelp()` 方法的下面）：

```
interface IDocumentation {
  public function downloadUpdates():Void;
  public function checkForUpdates():Boolean;
  public function searchHelp(keyword:String):Array;
  public function addComment(username:String, comment:String):Void;
}
```

11. 保存对接口文件的更改，然后重新打开 FlashPaper.as 文档。

12. 单击 “检查语法”按钮，将会看到 “输出”面板中出现一条新错误消息：

```
**Error** path\FlashPaper.as:Line 1:The class must implement method
  'addComment' from interface 'IDocumentation'.

    class FlashPaper implements IDocumentation {

Total ActionScript Errors:1  Reported Errors: 1
```

之所以会看到上述错误，是因为 FlashPaper.as 类文件已不再定义您在接口文件中列出的所有类。为修复此错误，必须要么向 FlashPaper 类添加 `addComment()` 方法，要么从 IDocumentation 接口文件中删除该方法的定义。

13. 在 FlashPaper 类中添加以下方法：

```
public function addComment(username:String, comment:String):Void {
  /* 将参数发送到服务器端页面，该页面会将注释插入到数据库中。 */
}
```

14. 保存对 FlashPaper.as 的更改，单击“检查语法”按钮，这次应该不会再收到任何错误了。

在以上部分中您基于 IDocumentation 接口文件创建了一个类。在本部分您将创建一个新类，它也将实现 IDocumentation 接口，不过它会添加一些其它方法和属性。

本教程将演示使用接口的有效性，因为如果您想创建扩展 IDocumentation 接口的另外一个类，使用接口可以方便地确定新类中所需的方法。

示例：创建复杂接口

以下示例显示用于定义和实现接口的若干方法。在此教程中，您将学习如何创建简单的接口文件，如何编写实现多个接口的类，以及如何让接口扩展其它接口以创建更为复杂的数据结构。

若要创建复杂接口，请执行以下操作：

1. 创建一个新的 ActionScript 文档，将其保存为 **InterfaceA.as**。
2. 创建一个名为 **complexInterface** 的新文件夹，并将 InterfaceA.as 保存到此目录中。

   应将为此教程创建的所有文件都保存到此目录中。
3. 在 Interface.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
// 文件名: InterfaceA.as
interface InterfaceA {
  public function k():Number;
  public function n(z:Number):Number;
}
```

4. 保存 ActionScript 文档，然后创建一个名为 **ClassB.as** 的新 ActionScript 文档，将它保存在 complexInterface 目录中。

   ClassB.as 将实现您前面创建的 InterfaceA 接口。

5. 在 ClassB.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
// 文件名: ClassB.as
class ClassB implements InterfaceA {
  public function k():Number {
    return 25;
  }
  public function n(z:Number):Number {
    return (z + 5);
  }
}
```

6. 保存对 ClassB.as 文档的更改，然后创建一个新的 Flash 文档，在 complexInterface 目录中将它保存为 **classbTest.fla**。

   此类文件将测试您前面创建的 ClassB 类。

7. 在 classbTest.fla 中，将下面的 ActionScript 代码键入到时间轴的第 1 帧上：

```
// 文件名: classbTest.fla
import ClassB;
var myB:ClassB = new ClassB();
trace(myB.k()); // 25
trace(myB.n(7)); // 12
```

8. 保存对 Flash 文档所做的更改，然后选择“控制”>“测试影片”对 Flash 文档进行测试。“输出”面板上将显示两个数字 25 和 12，它们是 ClassB 类中 `k()` 和 `n()` 方法的结果。

9. 创建一个新的 ActionScript 文件，在 complexInterface 目录中将它保存为 **ClassC.as**。

   此类文件将实现在步骤 1 中创建的 InterfaceA 接口。

10. 在 ClassC.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
// 文件名: ClassC.as
class ClassC implements InterfaceA {
  public function k():Number {
    return 25;
  }
  // ** 错误 ** 该类也必须从接口 “InterfaceA”实现方法 “n”。
}
```

    如果对 ClassC 类文件单击 “检查语法”按钮，Flash 将在 “输出”面板中显示一条错误消息，指示当前的类必须实现 InterfaceA 接口中定义的 `n()` 方法。在创建实现接口的类时，为接口中的每个条目都定义方法是非常重要的。

11. 创建一个新的 ActionScript 文档，在 complexInterface 目录中将它保存为 **InterfaceB.as**。

12. 在 InterfaceB.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
// 文件名: InterfaceB.as
interface InterfaceB {
  public function o():Void;
}
```

13. 保存对 InterfaceB.as 文档的更改，然后创建一个新的 ActionScript 文档，在 complexInterface 目录中将它保存为 **ClassD.as**。

    此类将实现在前面步骤中创建的 InterfaceA 接口和 InterfaceB 接口。ClassD 类必须包含每个接口文件中列出的每个方法的方法实现。

14. 在 ClassD.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
// 文件名: ClassD.as
class ClassD implements InterfaceA, InterfaceB {
  public function k():Number {
    return 15;
  }
  public function n(z:Number):Number {
    return (z * z);
  }
  public function o():Void {
    trace("o");
  }
}
```

15. 保存对 ClassD.as 文件的更改，然后创建一个新的 Flash 文档，并将其保存为 **classdTest.fla**。

    此 Flash 文档将测试上面创建的 ClassD 类。

16. 在 classdTest.fla 中，将下面的 ActionScript 代码添加到时间轴的第 1 帧上：

```
// 文件名: classdTest.fla
import ClassD;
var myD:ClassD = new ClassD();
trace(myD.k()); // 15
trace(myD.n(7)); // 49
myD.o(); // o
```

17. 保存对 classdTest.fla 文档所做的更改，然后选择 “控制” > “测试影片”对该文件进行测试。

    在 “输出”面板中应该显示值 15、49 和字母 o。这些值是分别是 `ClassD.k()` 方法、`ClassD.n()` 和 `ClassD.o()` 方法的结果。

18. 创建一个新的 ActionScript 文档，将其保存为 **InterfaceC.as**。

    此接口将扩展前面创建的 InterfaceA 接口，并将添加新的方法定义。

19. 在 InterfaceC.as 中，将以下 ActionScript 代码键入到 “脚本”窗口中：

```
// 文件名: InterfaceC.as
interface InterfaceC extends InterfaceA {
  public function p():Void;
}
```

20. 保存对 ActionScript 文件的更改，然后创建一个新的 ActionScript 文件，并在 complexInterface 目录中将它保存为 **ClassE.as**。

此类将实现两个接口，InterfaceB 和 InterfaceC。

21. 在 ClassE.as 中，将以下 ActionScript 代码键入到“脚本”窗口中：

```
// 文件名: ClassE.as
class ClassE implements InterfaceB, InterfaceC {
  public function k():Number {
    return 15;
  }
  public function n(z:Number):Number {
    return (z + 5);
  }
  public function o():Void {
    trace("o");
  }
  public function p():Void {
    trace("p");
  }
}
```

22. 保存对 ActionScript 文档的更改，然后创建一个新的 Flash 文档，在 complexInterface 目录中将它保存为 **classeTest.fla**。

23. 在 classeTest.fla 中，将下面的 ActionScript 代码键入到时间轴的第 1 帧上：

```
// 文件名: classeTest.fla
import ClassE;
var myE:ClassE = new ClassE();
trace(myE.k()); // 15
trace(myE.n(7)); // 12
myE.o(); // o
myE.p(); // p
```

24. 保存 Flash 文档，然后选择“控制”>“测试影片”对 SWF 文件进行测试。

“输出”面板中将显示值 15、12、o 和 p。这些值是从 ClassE.k()、ClassE.n()、ClassE.o() 和 ClassE.p() 方法中返回的值。因为 ClassE 类实现了 InterfaceB 和 InterfaceC 接口，所以必须定义来自这两个接口文件的每个方法。虽然 InterfaceB 和 InterfaceC 接口只定义 o() 和 p() 方法，但 InterfaceC 仍扩展了 InterfaceA。这意味着还必须实现它定义的所有方法，即 k() 和 n()。

第 9 章

处理事件

9

事件 是 SWF 文件播放时发生的动作。例如，鼠标单击或按键之类的事件称作用户事件，因为它是由于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 ActionScript 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。

用 ActionScript 处理事件的方式有多种：

- 第 256 页的 “使用事件处理函数方法”
- 第 258 页的 “使用事件侦听器”
- 第 262 页的 “使用按钮和影片剪辑事件处理函数”，具体说就是 on 处理函数和 onClipEvent 处理函数。
- 第 266 页的 “从组件实例广播事件”

将事件处理函数和 loadMovie （MovieClip.loadMovie 方法）一起使用，其结果是无法预知的。如果使用 `on()` 将事件处理函数附加到按钮，或是使用诸如 onPress （MovieClip.onPress 处理函数）的事件处理函数方法创建动态处理函数，然后调用 `loadMovie()`，则在加载新内容之后，事件处理函数将不再可用。然而，如果使用 onClipEvent 处理函数或 on 处理函数将事件处理函数附加到影片剪辑，然后对该影片剪辑调用 `loadMovie()`，则在加载新内容之后，事件处理函数将仍然可用。

有关处理事件的更多信息，请参见以下各节：

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，MovieClip 类定义 `onPress` 事件处理函数，只要按下鼠标就对影片剪辑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。

以下 ActionScript 类是用于定义事件处理函数的类的示例：Button、ContextMenu、ContextMenuItem、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《ActionScript 2.0 语言参考》中各个类的相应条目。每个事件处理函数的标题中都添加了 handler 这个单词。

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 () {
  // 此处是您的代码，对事件作出反应。
}
```

例如，假设您在舞台上具有一个名为 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当前时间轴中的下一帧：

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

分配函数引用 在上面的代码中，nextFrame() 函数被分配给 onPress 的事件处理函数。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，以后再定义该函数。如下例所示：

```
// 将一个函数引用分配给按钮的 onPress 事件处理函数。
next_btn.onPress = goNextFrame;

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

请注意，在下面的示例中您将函数引用（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;
```

接收传递参数 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不再具有键盘焦点的文本字段：

```
this.createTextField("my_txt", 99, 10, 10, 200, 20);
my_txt.border = true;
my_txt.type = "input";
this.createTextField("myOther_txt", 100, 10, 50, 200, 20);
myOther_txt.border = true;
myOther_txt.type = "input";
myOther_txt.onSetFocus = function(my_txt:TextField) {
  my_txt.text = "I just lost keyboard focus";
};
```

运行时对象的事件处理函数 您也可以为用于运行时创建的对象的事件处理函数分配函数。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：

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

有关更多信息，请参见第 321 页的 “在运行时创建影片剪辑”。

覆盖事件处理函数方法 通过创建可扩展 ActionScript 类的类，您可以用编写的函数覆盖事件处理函数方法。可以在新的子类中定义事件处理函数，然后通过将扩展类所在库中的所有元件链接到此新子类，可以为各种对象重复使用事件处理函数。下面的代码使用一个降低影片剪辑透明度的函数覆盖了 MovieClip 类的 onPress 事件处理函数：

```
// FadeAlpha 类 -- 单击影片剪辑时设置透明度。
class FadeAlpha extends MovieClip {
  function onPress() {
    this._alpha -= 10;
  }
}
```

有关扩展 ActionScript 类和链接库内元件的具体说明，请参见第 209 页的 “将类分配给 Flash 中的元件”中的示例。有关编写和使用自定义类的信息，请参见第 6 章 “类”，

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，或者注册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。

事件的侦听器 - 广播器模型与事件处理函数方法不同，它允许多个代码片断互不冲突地侦听同一事件。不使用侦听器 / 广播器模型的事件模型 （如 XML.onLoad()）在不同的代码片断侦听同一事件时可能会有问题；不同的代码片断在控制唯一的 XML.onLoad 回调函数引用时会产生冲突。利用侦听器 / 广播器模型，您可以轻松地为同一事件添加侦听器而不用担心代码瓶颈。

以下 ActionScript 类能够广播事件：Key、Mouse、MovieClipLoader、Selection、Stage 和 TextField。要查看类可用的侦听器，请参见 《ActionScript 2.0 语言参考》中各个类的相应条目。

有关事件侦听器的更多信息，请参见以下主题：

- 第 259 页的 “事件侦听器模型”
- 第 260 页的 “事件侦听器示例”

Stage 类可以广播事件。在硬盘上的 Samples 文件夹中可以找到范例源文件 stagesize.fla。此范例演示了在调整浏览器窗口大小时，`Stage.scaleMode` 属性如何影响 `Stage.width` 和 `Stage.height` 的值。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\StageSize。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/StageSize。

事件侦听器模型

事件侦听器的事件模型类似于事件处理函数的模型（请参见第 256 页的 “使用事件处理函数方法”），但有两个主要差别：

- 事件处理函数分配给侦听器对象，而不是广播事件的对象。
- 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。

以下代码概要介绍了事件侦听器模型：

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

代码由 *listenerObject* 对象开始，该对象含有一个属性 *eventName*。侦听器对象可以是任何对象，例如舞台上的现有对象、影片剪辑或按钮实例，或者可以是任何 ActionScript 类的实例。例如，某个自定义影片剪辑可以为舞台侦听器实现侦听器方法。您甚至可以让一个对象侦听多种类型的侦听器。

eventName 属性是一个发生于 *broadcasterObject* 之上的事件，broadcasterObject 接着会将该事件广播给 *listenerObject*。可以向一个事件广播器注册多个侦听器。

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，您对广播器对象调用 `addListener()` 方法，将侦听器对象传递给 addListener() 方法。

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 `removeEventListener()` 方法，向它传递要删除事件的名称以及侦听器对象。

```
broadcasterObject.removeListener(listenerObject);
```

事件侦听器示例

以下示例显示如何使用 Selection 类中的 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，然后在属性检查器中，从 “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。您可以继续创建输入文本字段。
4. 选择 “时间轴中的第 1 帧”，然后打开 “动作”面板 （“窗口” > “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：

```
// 创建侦听器对象 focusListener。
var focusListener:Object = new Object();
// 为侦听器对象定义函数。
focusListener.onSetFocus = function(oldFocus_txt:TextField,
  newFocus_txt:TextField) {
  oldFocus_txt.border = false;
  newFocus_txt.border = true;
}
```

   此代码创建一个名为 focusListener 的对象，用以定义 onSetFocus 属性并将一个函数分配给该属性。该函数采用两个参数：对不具有焦点的文本字段的引用和对具有焦点的文本字段的引用。该函数将不具有焦点的文本字段的 border 属性设置为 false，将具有焦点的文本字段的 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：

```
// 向广播器注册 focusListener。
Selection.addListener(focusListener);
```

7. 测试应用程序（“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。

对组件使用事件侦听器

使用组件时，事件侦听器语法会稍有不同。组件生成事件，您必须使用侦听器对象或自定义函数专门侦听这些事件。

下面的示例演示了您可以如何使用事件侦听器来监视动态加载图像的下载进度。

若要侦听 Loader 组件事件，请执行以下操作：

1. 将 Loader 组件的一个实例从 “组件”面板中拖放到舞台。
2. 选择 “loader”，然后在 “属性”检查器的 “实例名称”文本框中键入 **my_ldr**。
3. 将下面的代码添加到时间轴中的第 1 帧：

```
System.security.allowDomain("http://www.helpexamples.com");

var loaderListener:Object = new Object();
loaderListener.progress = function(evt_obj:Object):Void {
  trace(evt_obj.type); // progress
  trace("\t" + evt_obj.target.bytesLoaded + " of " +
  evt_obj.target.bytesTotal + " bytes loaded");
}
loaderListener.complete = function(evt_obj:Object):Void {
  trace(evt_obj.type); // complete
}

my_ldr.addEventListener("progress", loaderListener);
my_ldr.addEventListener("complete", loaderListener);
my_ldr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

   此 ActionScript 代码定义一个名为 `loaderListener` 的侦听器对象，以侦听以下两个事件：`progress` 和 `complete`。当调度其中的任何一个事件时，都将执行其代码，并在“输出”面板中显示调试文本 （如果您使用创作工具测试 SWF 文件）。

   接下来您指示 `my_ldr` 实例侦听这两个指定事件中的每个事件 （`progress` 和 `complete`），并指定调动事件时要执行的侦听器对象或函数。最后，将调用 `Loader.load()` 方法，以触发图像的下载。

4. 选择 “控制” > “测试影片”对 SWF 文件进行测试。

   该图像将下载到舞台上的 Loader 实例中，然后将在 “输出”面板中显示几条消息。根据下载的图像的大小和图像是否已缓存到了用户的本机系统，`progress` 事件可能会被多次调度，而 complete 事件只在图像完全下载后才调度一次。

   在使用组件和调度事件时，语法会与上述示例中的事件侦听器稍有不同。特别需要指出的是，必须使用 `addEventListener()` 方法，而不是调用 `addListener()`。其次，还必须指定要侦听的具体事件，并指定侦听器对象或函数。

可以使用自定义函数，而不是像第 261 页的 “对组件使用事件侦听器”中第一个步骤中那样使用侦听器对象。上述示例中的代码可以重写为：

```
System.security.allowDomain("http://www.helpexamples.com");

my_ldr.addEventListener("progress", progressListener);
my_ldr.addEventListener("complete", completeListener);
my_ldr.load("http://www.helpexamples.com/flash/images/image1.png");

function progressListener(evt_obj:Object):Void {
  trace(evt_obj.type); // progress
  trace("\t" + evt_obj.target.bytesLoaded + " of " +
  evt_obj.target.bytesTotal + " bytes loaded");
}
function completeListener(evt_obj:Object):Void {
  trace(evt_obj.type); // complete
}
```

注意

在上述示例中，始终要在调用 Loader.load() 方法之前添加事件侦听器。如果您在指定事件侦听器之前调用了 Loader.load() 方法，则加载有可能会在完全定义事件侦听器之前完成。这意味着虽然会显示内容，但却可能捕获不到 complete 事件。

使用按钮和影片剪辑事件处理函数

可以使用 onClipEvent() 和 on() 事件处理函数直接将事件处理函数附加到舞台上的按钮或影片剪辑实例。onClipEvent() 事件处理函数广播影片剪辑事件，而 on() 事件处理函数处理按钮事件。

若要将事件处理函数附加到某个按钮或影片剪辑实例，先单击舞台上的该按钮或影片剪辑实例，使它获得焦点，然后再在 “动作”面板中输入代码。“动作”面板的标题反映了代码将要附加到按钮或影片剪辑：“动作”面板 — 按钮或 “动作”面板 — 影片剪辑。有关使用附加于按钮或影片剪辑实例的代码的指导，请参见第 664 页的 “将代码附加到对象”。

注意

不要将按钮和影片剪辑事件处理函数与组件事件 （如 SimpleButton.click、UIObject.hide 和 UIObject.reveal）相混淆，组件事件必须附加到组件实例，对此我们会在 《使用组件》中进行讨论。

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 256 页的“使用事件处理函数方法”和第 258 页的“使用事件侦听器”。）

注意

不建议附加 onClipEvent() 和 on() 处理函数。而是应该按本手册所述将代码放在帧脚本或类文件中。有关更多信息，请参见第 256 页的 “使用事件处理函数方法”和第 664 页的 “将代码附加到对象”。

有关按钮和影片剪辑事件处理函数的更多信息，请参见以下主题：

与事件处理函数方法一同使用 on 和 onClipEvent

在某些情况下，您可以使用不同的技术无冲突地处理事件。使用 on() 和 onClipEvent() 方法并不与使用您自己定义的事件处理函数方法相冲突。

例如，假设 SWF 文件中有一个按钮；该按钮可以具有一个通知 SWF 文件播放的 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。根据想要在何时调用何种事件，您可以分别使用 on() 和 onClipEvent() 方法、事件处理函数方法或是同时使用事件处理的两种技术。

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 268 页的 “事件处理函数的作用域”。

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 266 页的 “创建具有按钮状态的影片剪辑”。有关为 on() 和 onClipEvent() 指定事件的信息，请参见第 264 页的 “为 on 或 onClipEvent 方法指定事件”。

使用 on 处理函数和 onPress 事件处理函数：

1. 创建一个新的 Flash 文档，并将它保存为 **handlers.fla**。
2. 选择 “矩形工具”并在舞台上绘制一个大的正方形。
3. 选择 “选择工具”，在舞台上双击该正方形，按 F8 启动 “转换为元件”对话框。
4. 输入框的元件名称，将类型设置为 “影片剪辑”并单击 “确定”。
5. 为舞台上的影片剪辑提供一个名为 **box_mc** 的实例。
6. 将以下 ActionScript 直接添加到舞台上的影片剪辑元件上：

```
on (press) {
  trace("on (press) {...}");
}
```

7. 将下面的 ActionScript 添加到主时间轴中的第 1 帧中：

```
box_mc.onPress = function() {
  trace("box_mc.onPress = function() {...};");
};
```

8. 选择 “控制” > “测试影片”对该 Flash 文档进行测试。

   单击舞台上的影片剪辑元件时，以下输出将发送到 “输出”面板：

```
on (press) {...}
box_mc.onPress = function() {...};
```

注意 不建议附加 `onClipEvent()` 和 `on()` 处理函数。而是应该按本手册所述将代码放在帧脚本或类文件中。有关更多信息，请参见第 256 页的 “使用事件处理函数方法”和第 664 页的 “将代码附加到对象”。

为 on 或 onClipEvent 方法指定事件

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。有关 `on()` 和 `onClipEvent()` 事件处理函数所支持的事件的完整列表，请参见《ActionScript 2.0 语言参考》中的 `on` 处理函数和 `onClipEvent` 处理函数。

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 ActionScript。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");
}
```

您还可以使用 `on()` 处理函数添加按键事件。例如，在键盘上按下数字 3 后，以下代码将跟踪字符串。选择一个按钮或影片剪辑实例，并向 “动作”面板添加以下代码：

```
on (keyPress "3") {
  trace("You pressed 3")
}
```

或者，如果要在用户按下 Enter 键后进行跟踪，可以使用以下代码格式。选择一个按钮或影片剪辑实例，向 “动作”面板添加以下代码：

```
on (keyPress "<Enter>") {
  trace("Enter Pressed");
}
```

选择 “控制” > “测试影片”，按下 Enter 键查看到 “输出”面板的字符串跟踪。如果什么都没跟踪到，选择 “控制” > “禁用快捷键”并重试。有关向应用程序添加按键互动性的更多信息，请参见 Key。

注意 不建议附加 onClipEvent() 和 on() 处理函数。而是应该按本手册所述将代码放在帧脚本或类文件中。有关更多信息，请参见第 256 页的 “使用事件处理函数方法”和第 664 页的 “将代码附加到对象”。

将多个处理函数附加或分配到一个对象

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。

```
on (press) {
  this.unloadMovie()
}
onClipEvent (load) {
  trace("I've loaded");
}
onClipEvent (unload) {
  trace("I've unloaded");
}
```

注意 不建议附加 onClipEvent() 和 on() 处理函数。而是应该按本手册所述将代码放在帧脚本或类文件中。有关更多信息，请参见第 256 页的 “使用事件处理函数方法”和第 664 页的 “将代码附加到对象”。

若要使用放置在时间轴上的代码将多个处理函数附加到一个对象，请参见以下示例。代码将 onPress 和 onRelease 处理函数附加到影片剪辑实例。

将多个处理函数分配给一个对象：

1. 创建一个新的 Flash 文档并将它命名为 **assignMulti.fla**。
2. 在时间轴中选择第 1 帧，在 “动作”面板中，添加下面的代码：

```
this.createEmptyMovieClip("img_mc", 10);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
     target_mc.onPress = function() {
          target_mc.startDrag();
     };
     target_mc.onRelease = function() {
          target_mc.stopDrag();
     };
}
```

```
mclListener.onLoadError = function(target_mc:MovieClip) {
     trace("error downloading image");
}
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

3. 选择 “控制” > “测试影片” 来测试该文档。

   该图像将加载到 `img_mc` 实例中，使用 `onPress()` 和 `onRelease()` 事件处理函数可以围绕舞台拖动该图像。

从组件实例广播事件

对于任何组件实例，您都可以指定如何处理事件。对组件事件的处理不同于对从本机 ActionScript 对象广播的事件的处理。

有关更多信息，请参见 《使用组件》中的 “处理组件事件”。

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。您还可以通过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。若要指定影片剪辑使用的点击区域，请使用 hitArea （MovieClip.hitArea 属性）。

若要创建影片剪辑中的按钮状态，请执行以下操作：

1. 创建一个新的 Flash 文档，并将它保存为 **mcbutton.fla**。
2. 使用 “矩形工具” 在舞台上绘制一个小矩形 （大约 100 像素宽， 20 像素高）。
3. 使用 “选择工具” 在舞台上双击该形状，按 F8 启动 “转换为元件” 对话框。
4. 输入元件名称 **mcbutton**，将元件类型设置为影片剪辑，然后单击 “确定”。
5. 在舞台上双击该影片剪辑元件，进入元件 - 编辑模式。
6. 在该影片剪辑的时间轴上创建一个新图层，并将它重命名为 **labels**。

7. 在属性检查器中输入帧标签 _up。
8. 在默认图层和标签图层之上创建一个新图层。
9. 将新图层重命名为 **actions**，并将以下 ActionScript 添加到影片剪辑的时间轴中的第 1 帧：

   stop();
10. 选择所有三个图层的第 10 帧，然后选择 “插入” > “时间轴” > “关键帧”。
11. 在动作图层的第 10 帧上添加 **stop()** 动作，并在标签图层的第 10 帧上添加帧标签 **_over**。
12. 选择第 10 帧上的矩形，并使用属性检查器选择另一种填充颜色。
13. 在所有三个图层的第 20 帧上创建新的关键帧，并在属性检查器中添加帧标签 **_down**。
14. 在第 20 帧中修改矩形的颜色，那样三个按钮状态就可以具有不同的颜色。
15. 返回到主时间轴。
16. 要令影片剪辑响应鼠标事件，请执行以下操作之一：
    - 如第 262 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 256 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）。
17. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

    将鼠标指针移到舞台上的影片剪辑实例上，影片剪辑将自动进入影片剪辑的 _over 状态。单击该影片剪辑实例，播放头将自动进入影片剪辑的 _down 状态。

事件处理函数的作用域

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。如果您在新的 ActionScript 类中定义事件处理函数，则范围也取决于您定义事件处理函数的方式。本部分包含了 ActionScript 1.0 和 ActionScript 2.0 的示例。

ActionScript 1.0 示例 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 ActionScript 函数）定义本地变量作用域，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 的父剪辑时间轴:
clip_mc.onPress = function () {
  var shoeColor; // 本地函数变量
  shoeColor = "blue";
}
// 附加到 clip_mc 的 on() 处理函数:
on (press) {
  var shoeColor; // 无本地变量作用域
  shoeColor = "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。在第二个示例中，因为 on() 处理函数没有定义本地变量作用域，所以变量的作用域为影片剪辑 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）是在包含该按钮实例的时间轴的作用域内调用的。

例如，以下 on() 事件处理函数将产生不同的结果，这取决于它是附加到影片剪辑对象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。
}
```

如果附加到按钮对象，play() 函数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 on(press) 处理函数附加到影片剪辑对象，play() 函数调用将应用于包含该处理函数的影片剪辑。如果将以下代码附加到影片剪辑上，它将会播放其父时间轴：

```
// 附加到影片剪辑。
on (press) {
  _parent.play(); // 播放父时间轴。
}
```

在事件处理函数或事件侦听器定义内，相同的 play() 函数将应用于包含该函数定义的时间轴。例如，假设您在包含影片剪辑实例 my_mc 的时间轴上声明了以下事件处理函数方法 my_mc.onPress：

```
// 时间轴上定义的函数
my_mc.onPress = function () {
  play(); // 播放在其上定义函数的时间轴。
};
```

若要播放定义 onPress 事件处理函数的影片剪辑，请使用 this 关键字显式引用该剪辑，如下所示：

```
// 根时间轴上定义的函数
my_mc.onPress = function () {
  this.play(); // 播放 my_mc 剪辑的时间轴。
};
```

然而，如果相同的代码放到按钮实例的根时间轴上，则将播放根时间轴：

```
my_btn.onPress = function () {
  this.play(); // 播放根时间轴
};
```

有关 this 关键字在事件处理函数中的作用域的更多信息，请参见第 271 页的“this 关键字的作用域”。

ActionScript 2.0 示例 下面的 TextLoader 类用于加载文本文件，并在文件加载成功后显示一些文本。

```
// TextLoader.as
class TextLoader {
  private var params_lv:LoadVars;
  public function TextLoader() {
    params_lv = new LoadVars();
    params_lv.onLoad = onLoadVarsDone;
    params_lv.load("http://www.helpexamples.com/flash/params.txt");
  }
  private function onLoadVarsDone(success:Boolean):Void {
    _level0.createTextField("my_txt", 999, 0, 0, 100, 20);
    _level0.my_txt.autoSize = "left";
    _level0.my_txt.text = params_lv.monthNames; // undefined
  }
}
```

这段代码无法正确工作，因为在事件处理函数的作用域上存在相关问题，而且 this 的所指也有混淆，无法区分指的是 onLoad 事件处理函数还是类本身。在本示例中，原本预期会出现的行为会在 TextLoader 对象的作用域内调用 onLoadVarsDone() 方法；但实际上是在 LoadVars 对象的作用域中调用的，这是因为该方法已经从 TextLoader 对象中提取出来然后转接到 LoadVars 对象上。在文本文件成功加载后， LoadVars 对象会接着调用 this.onLoad 事件处理函数，然后调用 onLoadVarsDone() 函数，而此时的 this 设置为 LoadVars，而不是 TextLoader。params_lv 对象在调用时位于 this 作用域中，即使 onLoadVarsDone() 函数按照引用依赖于 params_lv 对象，情况也是如此。因此，onLoadVarsDone() 函数需要的是并不存在的 params_lv.params_lv 实例。

要在 TextLoader 对象的范围内正确地调用 onLoadVarsDone() 方法，可以使用以下策略：使用函数文本创建一个调用所需函数的匿名函数。owner 对象在匿名函数的范围内仍然可见，所以可以用它来查找执行调用的 TextLoader 对象。

```
// TextLoader.as
class TextLoader {
  private var params_lv:LoadVars;
  public function TextLoader() {
    params_lv = new LoadVars();
    var owner:TextLoader = this;
    params_lv.onLoad = function (success:Boolean):Void {
      owner.onLoadVarsDone(success);
    }
    params_lv.load("http://www.helpexamples.com/flash/params.txt");
  }
  private function onLoadVarsDone(success:Boolean):Void {
    _level0.createTextField("my_txt", 999, 0, 0, 100, 20);
    _level0.my_txt.autoSize = "left";
    _level0.my_txt.text = params_lv.monthNames; // 一月、二月、三月……
  }
}
```

this 关键字的作用域

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 my_mc：

```
// 附加到主时间轴的 onPress() 事件处理函数:
my_mc.onPress = function () {
  trace(this); // _level0.my_mc
}
```

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，如以下代码所示：

```
// 已附加到主时间轴上名为 my_mc 的影片剪辑
on (press) {
  trace(this); // _level0.my_mc
}
```

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，如以下代码所示：

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

使用 Delegate 类

Delegate 类允许您在特定作用域内运行函数。提供了这个类之后，您就可以将同一个事件调度给两个不同的函数（请参见《使用组件》中的“将事件委托给函数”），并可以在包含类的作用域内调用函数。

当将一个函数作为参数传递给 EventDispatcher.addEventListener() 时，会在广播器组件实例的作用域内调用该函数，而不是在声明该函数的对象的作用域内调用（请参见《使用组件》中的“委托函数的范围”）。您可以使用 Delegate.create() 在声明对象的作用域内调用函数。

以下示例演示了侦听 Button 组件实例的事件的三种方法。使用不同的方法向 Button 组件实例添加事件侦听器，会使事件在不同作用域内调度。

使用 Delegate 类侦听事件：

1. 创建一个新的 Flash 文档，并将它保存为 **delegate.fla**。
2. 将一个 Button 组件从“组件”面板的 User Interface 文件夹拖到库中。

   在以后的步骤中，使用 ActionScript 将该按钮实例添加或放置在舞台上。
3. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

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

function clickHandler(eventObj:Object):Void {
  trace("[" + eventObj.type + "] event on " + eventObj.target + "
  instance.");
  trace("\t this -> " + this);
}

var buttonListener:Object = new Object();
buttonListener.click = function(eventObj:Object):Void {
  trace("[" + eventObj.type + "] event on " + eventObj.target + "
  instance.");
  trace("\t this -> " + this);
};

this.createClassObject(Button, "one_button", 10, {label:"One"});
one_button.move(10, 10);
one_button.addEventListener("click", clickHandler);

this.createClassObject(Button, "two_button", 20, {label:"Two"});
two_button.move(120, 10);
two_button.addEventListener("click", buttonListener);

this.createClassObject(Button, "three_button", 30, {label:"Three"});
three_button.move(230, 10);
three_button.addEventListener("click", Delegate.create(this,
  clickHandler));
```

前面的代码分为六个部分（各部分之间用空行分隔）。第一部分导入 Button 类（针对 Button 组件）和 Delegate 类。代码的第二部分定义用户单击某些按钮时调用的函数。代码的第三部分创建用作事件侦听器的对象，该对象侦听单个 `click` 事件。

代码剩下的三部分可单独在舞台上创建一个新的 Button 组件实例，重新放置实例，以及添加 `click` 事件的事件侦听器。第一个按钮添加 `click` 事件的事件侦听器，并直接传递对 `click` 处理函数的引用。第二个按钮添加 `click` 事件的事件侦听器，并传递对包含 `click` 事件的处理函数的侦听器对象的引用。最后，第三个函数添加 `click` 事件的事件侦听器，使用 Delegate 类在 `this` 作用域内调度 click 事件（其中 `this` 等于 `_level0`），并传递对 `click` 处理函数的引用。

4. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。
5. 单击舞台上的每个按钮实例，查看处理该事件的作用域。
   a. 单击舞台上的第一个按钮，在 “输出” 面板中跟踪以下文本：

      ```
      [click] event on _level0.one_button instance.
         this -> _level0.one_button
      ```

      单击 one_button 实例时，this 作用域引用该按钮实例本身。
   b. 单击舞台上的第二个按钮，在 “输出” 面板中跟踪以下文本：

      ```
      [click] event on _level0.two_button instance.
         this -> [object Object]
      ```

      单击 two_button 实例时，this 作用域引用 buttonListener 对象。
   c. 单击舞台上的第三个按钮，在 “输出” 面板中跟踪以下文本：

      ```
      [click] event on _level0.three_button instance.
         this -> _level0
      ```

      单击 three_button 实例时，this 作用域引用在 Delegate.create() 方法调用中指定的作用域，在这个示例中为 _level0。

第 10 章

数据和数据类型

10

有几章内容概括介绍和演示了 ActionScript 的一些基本概念，本章是其中的第一章。您将通过实践一些基本编码技术来学习如何创建复杂的应用程序。在本章中，您还将学习如何在 FLA 文件中使用数据，以及可以使用哪些类型的数据。在下一章第 4 章 “语法和语言基础知识”中，您将学习如何使用 ActionScript 语法和表单语句。接下来，第 5 章 “函数和方法”将演示如何在 ActionScript 语言中使用函数和方法。

有关数据和数据类型的更多信息，请参见以下各节：

关于数据

数据 是指可以在 Flash 中操作的数字、字符串和其它信息。创建应用程序或 Web 站点时，使用数据通常是必不可少的。在创建高级图形和脚本生成的动画时也需要使用数据，而且您可能必须通过操作所使用的值来实现效果。

在 Flash 中可以用变量 来定义数据，或者可以使用 XML、Web 服务、内置 ActionScript 类等从外部文件或站点中加载数据。可以将数据存储在数据库中，然后在 SWF 文件中以几种形式表示该信息。这可能包括在文本字段或组件中显示信息，或在影片剪辑实例中显示图像。

最常见的一些数据类型包括字符串型 （字符的序列，例如名称和文本段）、数字型、对象型（如影片剪辑）、布尔型 （`true` 和 `false`）等。在本章中，您还将学习 Flash 中的数据类型以及如何使用它们。

有关数据类型的信息，请参见第 276 页的 “关于数据类型”。有关变量的信息，请参见第 288 页的 “关于变量”。

关于数据类型

数据类型 描述一个数据片段，以及可以对其执行的各种操作。数据存储在变量中。在创建变量、对象实例和函数定义时，您使用数据类型来指定要使用的数据的类型。在编写 ActionScript 时可以使用多种不同的数据类型。

ActionScript 2.0 定义了几种常用的数据类型。数据类型描述变量或 ActionScript 元素可以包含的值的种类。指定了数据类型的变量仅能包含该数据类型值的集合中的一个值。有关变量的信息，请参见第 288 页的 “关于变量”。

ActionScript 具有多种基本数据类型，您在应用程序中可能会频繁地用到它们。有关更多信息，请参见第 277 页的 “关于原始和复杂数据类型”中的表。

ActionScript 还有核心类，例如 Array 和 Date，它们可以被视为复杂或引用数据类型。有关复杂和引用数据类型的更多信息，请参见第 277 页的 “关于原始和复杂数据类型”。另外，所有数据类型和类在《ActionScript 2.0 语言参考》中都有完整的定义。

您还可以为您的应用程序创建自定义类。使用 class 声明定义的任何类也被视为数据类型。有关核心类和其它内置类的更多信息，请参见第 215 页的 “关于顶级类和内置类”。有关创建自定义类的更多信息，请参见第 163 页的第 6 章 “类”。

在 ActionScript 2.0 中，可以在声明变量时为它们指定数据类型。您指定的数据类型可以是任何核心类型，也可以代表您创建的自定义类。有关更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。

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 `instanceof` 和 `typeof` 运算符实现这一目的（请参见第 288 页的 “关于确定数据类型”）。

您可以使用以下转换函数之一在运行时将一种数据类型转换为另一种数据类型：`Array()`、`Boolean()`、`Number()`、`Object()` 和 `String()`。

在硬盘上的 Samples 文件夹中可以找到范例源文件 datatypes.fla，该文件演示了如何在应用程序中使用数据类型。

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DataTypes。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DataTypes。

关于原始和复杂数据类型

可以将所有不同的数据类型值划分为两个主要的类别：原始 或复杂。

原始值 （即原始数据类型）是 ActionScript 在最低抽象层存储的值，这意味着对原始数据类型的操作比对复杂数据类型的操作往往会更快、更高效。以下数据类型都定义了一个原始值集合（可能包含一个原始值，也可能包含多个）：Boolean、null、Number、String 和 undefined。

复杂值 （或复杂数据类型）不是原始值，但它引用原始值。通常，我们称之为引用 数据类型。复杂值属于 Object 数据类型，或者说是基于 Object 数据类型的一种数据类型。定义复杂值的集合的数据类型包括：Array、Date、Error、Function 和 XML。有关这些复杂数据类型的更多信息，请参见 《ActionScript 2.0 语言参考》中的相应条目。

包含原始数据类型的变量与包含复杂类型的变量在某些情况下的行为是不同的。有关更多信息，请参见第 306 页的 “在项目中使用变量”。

ActionScript 具有以下基本数据类型，您可以在应用程序中使用它们：

数据类型	说明
Boolean	原始。Boolean 数据类型包括两个值：`true` 和 `false`。对于此类变量，其它任何值都是无效的。已经声明但尚未初始化的布尔变量的默认值是 `false`。有关更多信息，请参见第 278 页的 “Boolean 数据类型”。
MovieClip	复杂。 MovieClip 数据类型允许您使用 MovieClip 类的方法控制影片剪辑元件。有关更多信息，请参见第 279 页的 “MovieClip 数据类型”。
null	原始。null 数据类型包含一个值，即 `null`。此值意味着“没有值”，即没有数据。在很多种情况下，您可以指定 `null` 值，以指示某个属性或变量尚未赋值。null 数据类型是所有定义复杂数据类型的类的默认数据类型。此规则的例外情况是 Object 类，其默认值为 `undefined`。有关更多信息，请参见第 281 页的 “null 数据类型”。
Number	原始。此数据类型可以表示整数、无符号整数和浮点数。若要存储浮点数，数字中应该包括一个小数点。若没有小数点，数字将被存储为整数。Number 数据类型可以存储从 `Number.MAX_VALUE` （最大值）到 `Number.MIN_VALUE` （最小值）之间的值。有关更多信息，请参见 《ActionScript 2.0 语言参考》和第 281 页的 “Number 数据类型”。
Object	复杂。Object 数据类型是由 Object 类定义的。在 ActionScript 中，Object 类可用作所有类定义的基类，它可以使对象包含对象 （嵌套对象） 。有关更多信息，请参见第 282 页的 “Object 数据类型”。
String	原始。 String 数据类型表示 16 位字符的序列，可能包括字母、数字和标点符号。字符串存储为 Unicode 字符，使用 UTF-16 格式。对字符串值的操作返回字符串的一个新的实例。有关更多信息，请参见第 282 页的 “String 数据类型”。

数据类型	说明
undefined	原始。undefined 数据类型包含一个值：undefined。这是 Object 类实例的默认值。您只能为属于 Object 类的变量指定 undefined 这一值。有关更多信息，请参见第 283 页的 “undefined 数据类型”。
Void	复杂。Void 数据类型仅包含一个值：void。可以对不返回值的函数指定此数据类型。Void 是一个复杂数据类型，它引用原始 Void 数据类型。有关更多信息，请参见第 284 页的 “Void 数据类型”。

在硬盘上的 Samples 文件夹中可以找到范例源文件 datatypes.fla，该文件演示了如何在应用程序中使用数据类型。

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DataTypes。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DataTypes。

Boolean 数据类型

布尔值是 true 或 false 中的一个。ActionScript 也会在适当时将值 true 和 false 转换为 1 和 0。布尔值经常与 ActionScript 语句中通过比较来控制脚本流的逻辑运算符一起使用。

下面的示例将一个文本文件加载到一个 SWF 文件中，如果文本文件加载不正确，则在 “输出” 面板中显示一条消息，如果成功加载，则显示各个参数。有关详细信息，请参见代码示例中的注释。

```
var my_lv:LoadVars = new LoadVars();
//success 是一个布尔值
my_lv.onLoad = function(success:Boolean) {
  // 如果 success 为 true，则输出 monthNames
  if (success) {
    trace(my_lv.monthNames);
  // 如果 success 为 false，则输出消息
  } else {
    trace("unable to load text file");
  }
};
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

下面的示例检查用户是否在两个 TextInput 组件实例中输入了值。创建两个布尔变量 userNameEntered 和 isPasswordCorrect，如果这两个变量的求值结果都等于 true，则将一条欢迎消息赋值给 titleMessage String 变量。

```
// 在舞台上添加两个 TextInput 组件、一个 Label 和一个 Button 组件。
// 三个组件实例都是严格数据类型
var userName_ti:mx.controls.TextInput;
var password_ti:mx.controls.TextInput;
var submit_button:mx.controls.Button;
var welcome_lbl:mx.controls.Label;

// 隐藏标签
welcome_lbl.visible = false;

// 创建一个侦听器对象，它与 Button 组件结合使用。
// 单击 Button 时，检查用户名和密码。
var btnListener:Object = new Object();
btnListener.click = function(evt:Object) {
  // 检查用户是否在 TextInput 中输入了至少一个字符
  // 实例和返回布尔值 true/false。
  var userNameEntered:Boolean = (userName_ti.text.length > 0);
  var isPasswordCorrect:Boolean = (password_ti.text == "vertigo");
  if (userNameEntered && isPasswordCorrect) {
    var titleMessage:String = "Welcome " + userName_ti.text + "!";
    welcome_lbl.text = titleMessage;
    // 显示标签
    welcome_lbl.visible = true;
  }
};
submit_button.addEventListener("click", btnListener);
```

有关更多信息，请参见第152页的“在 Flash 中使用函数”和第135页的“关于逻辑运算符”。

MovieClip 数据类型

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。

调用 MovieClip 类的方法时不使用构造函数。可以在舞台上创建一个影片剪辑实例，或者动态地创建一个实例。然后只需使用点 (.) 运算符调用 MovieClip 类的方法。

在舞台上使用影片剪辑　下面的示例为舞台上不同的影片剪辑实例调用 startDrag() 和 getURL() 方法：

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

第二个示例返回舞台上名为 my_mc 的影片剪辑的宽度。所针对的实例必须是影片剪辑，返回的值必须是数字值。

```
function getMCWidth(target_mc:MovieClip):Number {
  return target_mc._width;
}
trace(getMCWidth(my_mc));
```

动态创建影片剪辑 当您想要避免在舞台上手动创建影片剪辑或想避免从库中手动附加影片剪辑时，使用 ActionScript 动态地创建影片剪辑非常有用。例如，您可以创建一个图库，里面存放大量您想要在舞台上组织的缩略图。使用 MovieClip.createEmptyMovieClip() 允许您完全使用 ActionScript 创建应用程序。

要动态地创建影片剪辑，请使用 MovieClip.createEmptyMovieClip()，如下面的示例所示：

```
// 创建一个影片剪辑以放置该容器。
this.createEmptyMovieClip("image_mc", 9);
// 将图像加载到 image_mc 中。
image_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

第二个示例创建一个名为 square_mc 的影片剪辑，该影片剪辑使用 Drawing API 绘制一个矩形。其中添加了事件处理函数以及 MovieClip 类的 startDrag() 和 stopDrag() 方法，以使矩形可拖动。

```
this.createEmptyMovieClip("square_mc", 1);
square_mc.lineStyle(1, 0x000000, 100);
square_mc.beginFill(0xFF0000, 100);
square_mc.moveTo(100, 100);
square_mc.lineTo(200, 100);
square_mc.lineTo(200, 200);
square_mc.lineTo(100, 200);
square_mc.lineTo(100, 100);
square_mc.endFill();
square_mc.onPress = function() {
  this.startDrag();
};
square_mc.onRelease = function() {
  this.stopDrag();
};
```

有关更多信息，请参见第 313 页的第 11 章“使用影片剪辑”和《ActionScript 2.0 语言参考》中的 MovieClip 条目。

null 数据类型

null 数据类型只有一个值，即 null。此值意味着没有值，即缺少数据。在很多种情况下，您可以指定 null 值，以指示某个属性或变量尚未赋值。例如，您可以在以下情况下指定 null 值：

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

如果尚未设置值，一些方法和函数会返回 null。下面的示例演示了如何使用 null 来测试表单字段当前是否拥有表单焦点：

```
if (Selection.getFocus() == null) {
  trace("no selection");
}
```

Number 数据类型

Number 数据类型是双精度浮点数。数字对象的最小值大约为 5e-324，最大值大约为 1.79E+308。

您可以使用加 (+)、减 (-)、乘 (*)、除 (/)、求模 (%)、递增 (++) 和递减 (--) 等算术运算符来操作数字。有关更多信息，请参见第 129 页的 “使用数值运算符”。

您也可使用内置的 Math 和 Number 类的方法来操作数字。有关这些类的方法和属性的更多信息，请参见 《ActionScript 2.0 语言参考》中的 Math 和 Number 条目。

下面的示例使用 Math 类的 sqrt() （平方根）方法返回数字 100 的平方根：

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

下面的示例输出一个 10 到 17 （含 10 和 17）之间的随机整数：

```
var bottles:Number = 0;
bottles = 10 + Math.floor(Math.random() * 7);
trace("There are " + bottles + " bottles");
```

下面的示例查找加载的 intro_mc 影片剪辑的百分比，并将百分比表示为整数：

```
var percentLoaded:Number = Math.round((intro_mc.getBytesLoaded() /
  intro_mc.getBytesTotal()) * 100);
```

Object 数据类型

对象是属性的集合。属性 是用于描述对象的特性。例如，对象 （如影片剪辑）的透明度是描述其外观的一个特性。因此，_alpha（透明度）是一个属性。每个属性都有名称和值。属性的值可以是任何 Flash 数据类型，甚至可以是 Object 数据类型。这样就可以使对象包含对象 （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
```

ActionScript 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 ActionScript 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、年龄和电话号码，球的速度，购物车中货物的名称，已加载的帧数，或用户上次所按的键。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。

下面的 ActionScript 代码显示了使用自定义对象组织信息的示例。它创建了一个名为 user 的新对象并创建了三个属性，name、age 和 phone，这些属性分别是 String 和 Number 数据类型。

```
var user:Object = new Object();
user.name = "Irving";
user.age = 32;
user.phone = "555-1234";
```

有关更多信息，请参见第 194 页的 “示例：编写自定义类”。

String 数据类型

字符串是诸如字母、数字和标点符号等字符的序列。在 ActionScript 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。

使用字符串类型的常见方法是将一个字符串指定给一个变量。例如，在下面的语句中，"L7" 是指定给变量 favoriteBand_str 的字符串：

```
var favoriteBand_str:String = "L7";
```

可以使用加法 (+) 运算符连接 或合并两个字符串。 ActionScript 将字符串开头或结尾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：

```
var greeting_str:String = "Welcome, " + firstName;
```

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 ActionScript 中，还有一些只能用特殊的转义序列才能表示的字符。下表列出了所有 ActionScript 转义符：

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

ActionScript 中的字符串是不可改变的，就像在 Java 中一样。任何修改字符串的操作都会返回一个新字符串。

String 类是一个内置的 ActionScript 类。有关 String 类的方法和属性的信息，请参见《ActionScript 2.0 语言参考》中的 String 条目。

undefined 数据类型

undefined 数据类型有一个值，即 undefined，并会自动指定给尚未赋值的变量，可以通过代码或用户交互进行指定。

值 undefined 是自动指定的；与 null 不同，您无需将 undefined 指定给变量或属性。您使用 undefined 数据类型检查是否已设置或定义某个变量。此数据类型允许您编写只在应用程序运行时执行的代码，如下面的示例所示：

```
if (init == undefined) {
  trace("initializing app");
  init = true;
}
```

如果您的应用程序有多个帧，则代码不会执行第二次，因为 init 变量不再是未定义的了。

Void 数据类型

Void 数据类型有一个值 `void`，用来在函数定义中指示函数不返回值，如下面的示例所示：

```
// 创建返回类型为 Void 的函数
function displayFromURL(url:String):Void {}
```

关于指定数据类型和严格数据类型指定

Flash 中使用变量来在代码中保存值。在创建变量时可以显式声明变量的对象类型，这称作严格数据类型指定。

如果不显式地将项目定义为具有数字、字符串或其它数据类型，则在运行时，Flash Player 将尝试在为项目赋值时确定该项目的数据类型。如果您为变量赋值（如下面的示例所示），Flash Player 会在运行时计算运算符右边元素的值，并确定它是否为 Number 数据类型：

```
var x = 3;
```

因为没有使用严格数据类型指定声明 `x`，所以编译器无法确定其类型。对于编译器而言，变量 x 可以具有任何类型的值。（请参见第 285 页的 “指定数据类型”。）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 String。

当表达式需要转换但没有严格指定变量的类型时，ActionScript 总是自动转换原始数据类型（如 Boolean、Number、String、null 或 undefined）。

严格数据类型指定在编译时带来了几方面好处。声明数据类型（严格数据类型指定）有助于防止代码中出现错误，或者可以在编译时诊断代码中的错误。要使用严格数据类型指定来声明变量，请使用下面的格式：

```
var variableName:datatype;
```

术语	严格数据类型指定有时称为对变量进行强类型指定。

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在编译时查找代码中的错误，以及避免为现有变量指定类型错误的数据。在创作期间，严格数据类型指定会激活 ActionScript 编辑器中的代码提示（但您仍应为可视元素使用实例名称后缀）。

使用严格数据类型指定有助于确保您不会因为疏忽而为变量指定错误的值类型。Flash 将在编译时检查类型指定不匹配错误，如果使用了错误的值类型，就会显示一条错误消息。因此，使用严格类型指定还有助于确保您不会尝试访问不是对象类型一部分的属性或方法。严格数据类型指定意味着 ActionScript 编辑器会自动显示对象的代码提示。

有关创建变量的更多信息，请参见第 288 页的 “关于变量”。有关命名变量的信息，请参见第 293 页的 “关于命名变量”。有关指定数据类型以及可以指定的类型的更多信息，请参见第 285 页的 “指定数据类型”。

在硬盘上的 Samples 文件夹中可以找到范例源文件 datatypes.fla，该文件演示了如何在应用程序中使用数据类型。

- 在 Windows 中，浏览到启动驱动器 \Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DataTypes。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DataTypes。

指定数据类型

只要定义变量，就需要指定数据类型 -- 无论是使用 var 关键字声明变量，创建函数参数，设置函数返回类型，还是定义在 for 或 for..in 循环中使用的变量。要指定数据类型，请使用后冒号语法，这意味着变量名后面需要跟一个冒号，然后是数据类型：

```
var my_mc:MovieClip;
```

有多种可能的数据类型，包括本机数据类型（如 Number、String、Boolean）或 Flash Player 8 中包括的内置类（如 BitmapData、FileReference），甚至还有您或其他开发人员编写的自定义类。需要指定的最常见的数据类型是内置数据类型，例如，Number、String、Boolean、Array 或 Object。下面的代码示例中显示了这些数据类型。

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，如下面的示例所示：

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

// 严格指定参数的类型
function welcome(firstName:String, age:Number) {
}

// 严格指定参数和返回值的类型
function square(myNum:Number):Number {
  var squared:Number = myNum * myNum;
  return squared;
}
```

您可以根据内置类（Button、Date 等）以及您创建的类和接口来声明对象的数据类型。在下面的示例中，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 myStudent:Student = new Student();
```

对于此示例，假设您键入以下代码：

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

// 在 FLA 文件中
var studentMaryLago:Student = new Student();
studentMaryLago.status = "enrolled"; /* 赋值语句中类型不匹配：在需要 Boolean 类型的
  地方发现 String 类型。 */
```

当 Flash 编译此脚本时，会生成 “类型不匹配”错误，因为该 SWF 文件需要布尔值。

如果要编写一个没有返回类型的函数，可以将该函数的返回类型指定为 Void。或者，如果要创建到函数的快捷方式，可以将 Function 数据类型指定给新的变量。要指定对象类型为 Function 或 Void，请参见下面的示例：

```
function sayHello(name_str:String):Void {
  trace("Hello, " + name_str);
}
sayHello("world"); // Hello, world
var greeting:Function = sayHello;
greeting("Augustus"); // Hello, Augustus
```

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 会自动显示代码提示。有关更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。

使用 ActionScript 1.0 发布的文件在编译时不考虑严格数据类型的指定，因此为严格指定了类型的变量分配类型错误的值时，不会生成编译器错误。

```
var myNum:String = "abc";
myNum = 12;
/* 在 ActionScript 1.0 中不会出现任何错误，但是在 ActionScript 2.0 中会出现类型不匹配
  错误 */
```

其原因是，为 ActionScript 1.0 发布文件时， Flash 以斜杠语法（而不是严格类型指定）解释像 `var myNum:String = "abc"` 这样的语句。（ActionScript 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。

使用 ActionScript 2.0 发布的文件可以根据需要使用数据类型指定。因此，如果在代码中执行严格数据类型指定，请确保 “发布设置”设置为 ActionScript 2.0。可以使用以下方法指定 “发布设置”和定义要用于发布文件的 ActionScript 版本：从主菜单中修改发布设置（“文件” > “发布设置”），或单击属性检查器中的 “设置”按钮 （确保没有选中任何实例） 。要使用特定版本的 ActionScript 或 Flash Player，请在 “发布设置”对话框中选择 “Flash”选项卡，并从 “ActionScript 版本”弹出菜单中做出一项选择。

有关类型检查的信息，请参见第 287 页的 “关于类型检查”。

关于类型检查

类型检查 是指验证变量和表达式的类型是否兼容。因此，Flash 检查您为变量指定的类型是否与赋给它的值相匹配。有关严格数据类型和指定数据类型的更多信息，请参见第 284 页的“关于指定数据类型和严格数据类型指定”和第 285 页的 “指定数据类型”。

类型检查可以在编译时或运行时执行。如果使用严格数据类型指定，则在编译时进行类型检查。因为 ActionScript 是一种动态指定类型的语言，所以 ActionScript 在运行时也可以进行类型检查。

例如，以下代码没有指定参数 `xParam` 的数据类型。在运行时，可以使用该参数保存一个 Number 类型的值，然后使用该参数保存 `String` 类型的值。`dynamicTest()` 函数然后使用 `typeof` 运算符测试该参数是 String 类型还是 Number 类型。

```
function dynamicTest(xParam) {
  if (typeof(xParam) == "string") {
    var myStr:String = xParam;
    trace("String: " + myStr);
  } else if (typeof(xParam) == "number") {
    var myNum:Number = xParam;
    trace("Number: " + myNum);
  }
}
dynamicTest(100);
dynamicTest("one hundred");
```

无需在 ActionScript 中显式添加数据类型信息。 ActionScript 编译器允许在编译时使用不存在的属性和调用不存在的方法。这使您可以在运行时动态创建属性和指定方法。

动态类型检查提供的灵活性的一个例子是，可以在编译时使用未知的属性和方法。因为代码的限制性更小，所以，这会给某些编码情形带来好处。例如，下面的代码创建了一个名为 `runtimeTest()` 的函数，它调用一个方法并返回一个属性，这两个事件对编译器来说都是未知的。代码不会生成编译时错误，但是如果属性或方法在运行时不能访问，就会出现一个运行时错误。

```
function runtimeTest(myParam) {
  myParam.someMethod();
  return myParam.someProperty;
}
```

关于确定数据类型

当测试和调试程序时，您可能会发现看起来与不同项目的数据类型相关的问题。或者，如果您使用没有明确地与一种数据类型相关联的变量，您可能会发现，了解给定变量的数据类型是非常有用的。使用 ActionScript，可以确定一个项目的数据类型。可以使用 typeof 运算符返回关于数据的信息。

使用 typeof 运算符可获取数据类型，但是，请记住 typeof 不返回实例属于哪个类的信息。

下面的示例说明如何使用 typeof 运算符来返回所跟踪的对象的种类：

```
// 创建 LoadVars 类的新实例。
var my_lv:LoadVars = new LoadVars();

/* typeof 运算符不指定类，只指定 my_lv 是一个对象 */
var typeResult:String = typeof(my_lv);
trace(typeResult); // 对象
```

在此示例中，创建了一个名为 myName 的新的 String 变量，然后将它转换为 Number 数据类型：

```
var myName:String = new String("17");
trace(myName instanceof String); // true
var myNumber:Number = new Number(myName);
trace(myNumber instanceof Number); // true
```

有关这些运算符的更多信息，请参见《ActionScript 2.0 语言参考》中的 typeof 运算符和 instanceof 运算符。有关测试和调试的更多信息，请参见第 633 页的第 18 章“调试应用程序”。有关继承和接口的更多信息，请参见第 229 页的第 7 章“继承”。有关类的更多信息，请参见第 163 页的第 6 章“类”。

关于变量

变量 是保存信息的容器。下面的 ActionScript 显示了 ActionScript 中变量的样子：

```
var myVariable:Number = 10;
```

此变量保存一个数字值。在以上代码中，使用 :Number 指定该变量保存的值的类型，这称为数据类型指定。有关数据类型指定的更多信息，请参见第 284 页的“关于指定数据类型和严格数据类型指定”和第 285 页的“指定数据类型”。

容器（用变量名表示）在 ActionScript 中始终不变，但内容（值）可以更改。在一个脚本中，可以任意次更改变量的值。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，或者计算某个条件是 true 还是 false。在 SWF 文件播放时您可能需要变量不断更新，例如在 Flash 游戏中玩家的分数不断变化。在 SWF 文件中创建和处理用户交互时，必须使用变量。

首次声明变量时，最好为该变量赋一个值。指定一个初始值称为初始化 该变量，通常是在时间轴上的第一个帧中或 SWF 文件开始播放时加载的类中完成。变量有许多不同的种类，它们受作用域的影响。有关不同种类的变量及作用域的更多信息，请参见第 297 页的 “关于变量和作用域”。

提示 初始化变量有助于在播放 SWF 文件时跟踪和比较变量的值。

注意 Flash Player 7 及更高版本计算未初始化变量的方式与 Flash Player 6 及较低版本不同。如果您已为 Flash Player 6 编写了脚本，并计划为 Flash Player 7 或更高版本编写或移植脚本，就应了解这些区别，以避免出现意外行为。

变量可以保存不同类型的数据；有关更多信息，请参见第 276 页的 “关于数据类型”。变量包含的数据类型影响着脚本中分配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 （String 类型）、用户名 （String 类型）、数学运算的结果 （Number 类型）、事件发生的次数 （Number 类型），以及用户是否单击了某个特定的按钮 （Boolean 类型），等等。每个 SWF 文件和对象实例 （例如影片剪辑）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。

要查看变量的值，请使用 `trace()` 语句向 “输出”面板发送值。然后，在测试环境中测试 SWF 文件时，值显示在 “输出”面板中。例如，`trace(hoursWorked)` 会在测试环境中将变量 `hoursWorked` 的值发送给 “输出”面板。也可在测试环境中的调试器中检查和设置变量值。

有关变量的更多信息，请参见以下主题：

- 第 290 页的 “关于声明变量”
- 第 290 页的 “关于赋值”
- 第 293 页的 “关于命名变量”
- 第 294 页的 “在应用程序中使用变量”
- 第 297 页的 “关于变量和作用域”
- 第 290 页的 “关于默认值”
- 第 292 页的 “关于运算符和变量”
- 第 301 页的 “关于加载变量”
- 第 306 页的 “在项目中使用变量”

关于声明变量

可以在时间轴中的帧上声明变量，直接在对象上声明，或在一个外部类文件中声明。

应使用 var 关键字定义变量并遵守变量命名约定。可以声明一个名为 firstName 的变量，如下面的示例中所示：

```
var firstName:String;
```

在声明变量时，应指定变量的数据类型。此例中将 firstName 变量指定为 String 数据类型。有关指定数据类型的更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。

关于默认值

默认值 是设置变量的值之前变量中包含的值。首次设置变量的值实际上就是初始化 变量。如果您声明了一个变量，但是没有设置它的值，则该变量便处于未初始化状态。未初始化的变量的默认值为 undefined。有关创建和使用变量的更多信息，请参见第 288 页的 “关于变量”。

关于赋值

可以定义一个值 作为变量的当前内容。该值可以是字符串、数字、数组、对象、 XML、日期，甚至可以是您创建的自定义类。请记住，在 Flash 中应使用 var 关键字声明变量。声明变量时，还可以指定变量的数据类型。您也可以为变量赋值，只要所赋的值与为变量指定的数据类型匹配即可。

下面的示例显示了可以如何创建名为 catName 的变量：

```
var catName:String;
```

声明了变量之后，便可以为其赋值。可以在上一行 ActionScript 后面添加下面这一行：

```
catName = "Pirate Eye";
```

注意	因为 Pirate Eye 是一个字符串，所以其值需要置于直引号 （引号）中。

此示例将 Pirate Eye 这一值赋予变量 catName。在声明变量时，还可以为它赋值，而不必等声明以后再赋值 （如上面的示例中所示）。在声明 catName 变量时即可以设置此变量，如下面的示例中所示：

```
var catName:String = "Pirate Eye";
```

如果要在测试环境中显示 catName 变量的值，可以使用 trace() 语句。此语句向“输出”面板发送值。可以使用以下 ActionScript 跟踪 catName 变量的值，您会看到实际值不包括引号：

```
var catName:String = "Pirate Eye";
trace(catName); // Pirate Eye
```

请记住，您赋的值必须与为该变量指定的数据类型相匹配（在此例中为字符串）。如果以后尝试将一个数字赋予 catName 变量，例如 catName = 10，在测试 SWF 文件时您会发现“输出”面板中出现以下错误：

```
Type mismatch in assignment statement: found Number where String is required.
```

此错误指出您在试图为指定的变量设置错误的数据类型。

在赋予变量一个数字值时，不需要使用引号，如下面的代码中所示：

```
var numWrinkles:Number = 55;
```

如果以后要在代码中更改 numWrinkles 的值，可以使用以下 ActionScript 为变量赋予一个新值：

```
numWrinkles = 60;
```

在为现有的变量重新赋予新值时，无需使用 var 关键字或定义变量的数据类型（在此例中为 :Number）。

如果值是数字或布尔值（true 或 false），该值不用使用直引号（引号）。下面的代码片段中显示了数字和布尔值的示例：

```
var age:Number = 38;
var married:Boolean = true;
var hasChildren:Boolean = false;
```

在上面的示例中，变量 age 包含整数（非小数）值，但是您还可以使用小数或浮点值，如 38.4。Boolean 变量（例如 married 或 hasChildren）只有两个可能的值，即 true 或 false。

如果要创建一个数组并为它赋值，格式稍微有些不同，如以下代码所示：

```
var childrenArr:Array = new Array("Pylon", "Smithers", "Gil");
```

有一种使用数组访问运算符创建数组的备选（简化）语法，这种语法使用了方括号 ([])。您可以按如下方式重写以上示例：

```
var childrenArr:Array = ["Pylon", "Smithers", "Gil"];
```

有关创建数组和数组访问运算符的更多信息，请参见第 107 页的“关于数组”和第 68 页的“关于使用点语法将实例设定为目标”。

类似地，您可以创建名为 myObj 的新对象。可以使用以下两种方法中的任意一种创建新对象。第一种通过编码创建数组的方法（较长）如下所示：

```
var myObj:Object = new Object();
myObj.firstName = "Steve";
myObj.age = 50;
myObj.childrenArr = new Array("Mike", "Robbie", "Chip");
```

第二种通过编码创建 myObj 数组的简化方法如下所示：

```
var myObj:Object = {firstName:"Steve", age:50, childrenArr:["Mike",
  "Robbie", "Chip"]};
```

如在此示例中所看到的，使用简化方法可以节省大量的键入任务和时间，尤其是在定义对象的实例时。熟悉此备选语法是非常重要的，因为，在结组工作或者在使用从某个地方（例如从 Internet 上或其它书中）找到的第三方 ActionScript 代码时可能会遇到此语法。

注意 并非所有变量都需要显式定义。一些变量是 Flash 自动创建的。例如，要查找舞台的尺寸，应该使用以下两个预定义变量的值：Stage.width 和 Stage.height。

关于运算符和变量

您可能想知道代码中的数学符号。这些符号在 ActionScript 中称为运算符。运算符从一个或多个值中计算出一个新值，您可以在代码中使用运算符来为变量赋值。应使用等于 (=) 运算符为变量赋值：

```
var username:String = "Gus";
```

另一个示例是加法 (+) 运算符，用于将两个或更多个数字值相加，从而产生一个新值。如果您对两个或多个字符串值使用 + 运算符，这些字符串将连接在一起。运算符处理的值称为操作数。

在赋值时，可以使用运算符为变量定义一个值。例如，下面的脚本使用赋值运算符将值 7 赋予变量 numChildren：

```
var numChildren:Number = 7;
```

如果要更改变量 numChildren 的值，请使用以下代码：

```
numChildren = 8;
```

注意 不需要使用 var，因为该变量以前已经定义过。

有关在 ActionScript 中使用运算符的更多信息，请参见第 118 页的 “关于运算符”。

关于命名变量

在开始为变量命名时要小心，因为尽管它们几乎可以用任意名称，但是仍然有一些规则需要遵循。变量名必须遵守下面的规则：

- 变量必须是一个标识符。

注意	标识符 是变量、属性、对象、函数或方法的名称。标识符的第一个字符必须为字母、下划线 (_) 或美元符号 ($)。其后的字符可以是数字、字母、下划线或美元符号。

- 变量不能是关键字或 ActionScript 文本，例如 `true`、`false`、`null` 或 `undefined`。有关文本的更多信息，请参见第 79 页的 “关于文本”。
- 变量在其作用域内必须是唯一的 （请参见第 297 页的 “关于变量和作用域”）。
- 变量不能是 ActionScript 语言中的任何元素，例如类名称。

如果在命名变量时不遵守规则，可能会遇到语法错误或意外的结果。在下面的示例中，如果将一个变量命名为 `new`，然后测试文档， Flash 将生成一个编译器错误：

```
// 此代码可正常运行。
var helloStr:String = new String();
trace(helloStr.length); // 0
// 但如果您为变量提供与内置类相同的名称 ...
var new:String = "hello"; // 错误: 应该用标识符
var helloStr:String = new String();
trace(helloStr.length); // undifined
```

ActionScript 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，可以严格指定变量类型。代码提示提供了工具提示样式的语法提示，并提供一个帮您更快地编写代码的弹出菜单。

例如，键入以下代码：

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

您一在 “动作”面板中键入句点 (.)， Flash 就会显示可用于 Array 对象的方法和属性的列表。

如想了解变量命名方面的建议编码约定，请参见第 655 页的 “命名变量”。

在应用程序中使用变量

在本节中，您将在 ActionScript 的简短代码片断中使用变量。您需要在脚本中声明和初始化变量，然后才能在表达式中使用它。表达式是操作数和运算符的组合，表示一个值。例如，在表达式 `i+2` 中，`i` 和 `2` 是操作数，而 `+` 是运算符。

如果在表达式中使用变量之前没有对它进行初始化，则变量就处于未定义状态，因而会导致意外结果。有关编写表达式的更多信息，请参见第 63 页的第 4 章“语法和语言基础知识”。

如果使用未定义的变量（如下面的示例所示），在 Flash Player 7 和更高的版本中，该变量的值将为 `NaN`，并且您的脚本可能产生意外的结果：

```
var squared:Number = myNum * myNum;
trace(squared); // NaN
var myNum:Number = 6;
```

在下面的示例中，声明和初始化变量 `myNum` 的语句放在前面，因此 `squared` 就可以替换为一个值：

```
var myNum:Number = 6;
var squared:Number = myNum * myNum;
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，如下所示。

比较向函数传递未定义变量和已定义变量的情形：

1. 将 Button 组件从“组件”面板拖到舞台上。
2. 打开属性检查器，并在“实例名称”文本框中键入 **bad_button**。
3. 将下面的代码键入到时间轴中的第 1 帧上。

```
// 无效
function badClickListener(evt:Object):Void {
  getURL(targetUrl);
  var targetUrl:String = "http://www.macromedia.com";
}
bad_button.addEventListener("click", badClickListener);
```

4. 选择“控制”>“测试影片”，会发现按钮无效（没有打开 Web 页）。
5. 将另一个 Button 组件拖动到舞台上。选择该按钮。
6. 打开属性检查器，并在“实例名称”文本框中键入 **good_button**。
7. 将下面的 ActionScript 添加到时间轴上的第 1 帧上（跟在您前面添加的 ActionScript 之后）：

```
// 有效
function goodClickListener(evt:Object):Void {
  var targetUrl:String = "http://www.macromedia.com";
  getURL(targetUrl);
}
good_button.addEventListener("click", goodClickListener);
```

8. 选择 “控制” > “测试影片”，单击添加到舞台上的第二个按钮。

该按钮正常打开该 Web 页。

变量包含的数据类型会影响如何以及何时更改变量的值。原始数据类型 （例如字符串和数字）是按值进行传递 的，这意味着使用的是变量的当前值，而非对该值的引用。复杂数据类型的例子包括 Array 和 Object 数据类型。

在下面的示例中，将 myNum 设置为 15 并将该值复制到 otherNum 中。当您将 myNum 更改为 30（在代码的第 3 行）时，otherNum 的值仍然是 15，因为 otherNum 不会到 myNum 中查找它的值。otherNum 变量包含它接收到的 myNum 的值 （在代码的第 2 行）。

在 ActionScript 中使用变量：

1. 创建一个新的 Flash 文档，并将它保存为 **var_example.fla**。
2. 在时间轴中选择第 1 帧，在 “动作”面板中输入下面的代码：

```
var myNum:Number = 15;
var otherNum:Number = myNum;
myNum = 30;
trace(myNum); // 30
trace(otherNum); // 15
```

当您将 myNum 更改为 30 （在代码的第 3 行）时，otherNum 的值仍然是 15，因为 otherNum 不会到 myNum 中查找它的值。otherNum 变量包含它接收到的 myNum 的值（在代码的第 2 行）。

3. 选择 “控制” > “测试影片”，查看 “输出”面板中显示的值。
4. 现在将下面的 ActionScript 添加到第 2 步中添加的代码后面：

```
function sqr(myNum:Number):Number {
  myNum *= myNum;
  return myNum;
}
var inValue:Number = 3;
var outValue:Number = sqr(inValue);
trace(inValue); // 3
trace(outValue); // 9
```

在此代码中，变量 inValue 包含一个原始值 3，因此该值会传递给 sqr() 函数，而返回值为 9。变量 inValue 的值不会更改，尽管函数中的 myNum 值会发生更改。

5. 选择 “控制” > “测试影片”，查看 “输出”面板中显示的值。

Object 数据类型可以包含大量复杂的信息，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。

有关按引用传递变量的信息，请参见第 296 页的 “按引用传递变量”。

按引用传递变量

因为数组和 Object 数据类型包含对值的引用而不是包含实际的值，因此在使用数组和对象时要小心。

下面的示例显示了如何按引用传递对象。在创建数组的副本时，实际上创建的只是对数组内容的引用（或别名）的副本。编辑第二个数组中的内容时，将修改第一个和第二个数组的内容，因为它们都指向同一个值。

按引用传递对象：

1. 选择“文件”>“新建”，然后选择“Flash 文档”创建一个新的 FLA 文件，并将它保存为 **copybyref.fla**。
2. 在时间轴中选择第 1 帧，在“动作”面板中键入下面的代码：

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

3. 选择“控制”>“测试影片”对 ActionScript 进行测试。

   此 ActionScript 创建了一个名为 `myArray` 的 Array 对象，它包含两个元素。创建变量 `newArray` 并将引用传递给 `myArray`。当将 `myArray` 的第二个元素更改为 `jack` 时，它将影响引用它的每个变量。`trace()` 语句将 `tom,jack` 发送到“输出”面板。

Flash 使用基于零的索引，这意味着 0 是数组中的第一项，1 是第二项，依此类推。

在下面的示例中，`myArray` 包含一个 Array 对象，因此您可以按引用将该数组传递给函数 `zeroArray()`。函数 `zeroArray()` 会将 Array 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。

按引用传递数组：

1. 选择“文件”>“新建”，然后选择“Flash 文档”创建一个新的 FLA 文件，并将它保存为 **arraybyref.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
function zeroArray (theArr:Array):Void {
  var i:Number;
  for (i = 0; i < theArr.length; i++) {
    theArr[i] = 0;
  }
}
```

```
var myArr:Array = new Array();
myArr[0] = 1;
myArr[1] = 2;
myArr[2] = 3;
trace(myArr); // 1,2,3
zeroArray(myArr);
trace(myArr); // 0,0,0
```

3. 选择 “控制” > “测试影片” 对 ActionScript 进行测试。

   此 ActionScript 中的第一个 `trace()` 语句显示 `myArray` 数组的原始内容 (`1,2,3`)。在调用了 `zeroArray()` 函数并传递一个对 `myArray` 数组的引用之后，该数组的各个值都将被覆盖并被设置为 0。后面的 `trace()` 语句显示 `myArray` 数组的新内容 (`0,0,0`)。因为是按引用 （而不是按值）传递数组的，所以在 `zeroArray()` 函数中不需要返回数组的更新内容。

有关数组的更多信息，请参见第 63 页的第 4 章 “语法和语言基础知识”。

关于变量和作用域

变量的作用域是指识别 （定义）变量的区域和可以引用变量的区域。变量在其中被识别的区域可能在某个时间轴中或在某个函数内部，也可能在整个应用程序中是全局已知的。有关作用域的更多信息，请参见第 72 页的 “关于作用域和目标设定”。

在使用 ActionScript 开发 Flash 应用程序时，了解变量作用域是非常重要的。作用域不仅表示可以在什么时间和位置引用变量，还表示一个特定的变量可以在应用程序中存在多长时间。在函数体中定义变量时，一旦指定的函数结束，变量就不再存在了。如果您试图在错误的作用域内引用对象或引用已过期的变量，您的 Flash 文档中将会出错，这将导致出现意外情况或使功能受到破坏。

在 ActionScript 中有三种类型的变量作用域：

- 全局变量和函数对于文档中的每个时间轴和作用域均可见。因此，全局变量是在代码的所有区域中定义的。
- 时间轴变量可用于该时间轴上的任何脚本。
- 本地变量在声明它们的函数体 （由大括号界定）内可用。因此，本地变量仅是在代码的一部分中定义的。

有关使用作用域和变量的准则，请参见第 72 页的第 4 章 “关于作用域和目标设定”。

注意

您创建的 ActionScript 2.0 类支持公共、私有和静态变量作用域。有关更多信息，请参见第 183 页的 “关于类成员”和第 203 页的 “控制类中的成员访问”。

您不能严格指定全局变量的类型。有关信息和解决方法，请参见第 298 页的 “全局变量”。

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作用域而言都是可见的。要声明（或创建）具有全局作用域的变量，请在变量名前使用 _global 标识符，而不要使用 var = 语法。例如，以下代码创建全局变量 myName：

```
var _global.myName = "George"; // 全局变量的错误语法
_global.myName = "George"; // 全局变量的正确语法
```

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作用域内时对该全局变量不具有访问权限，如下面的示例所示：

```
_global.counter = 100; // 声明全局变量
trace(counter); // 访问全局变量并显示 100
function count():Void {
  for (var counter:Number = 0; counter <= 2; counter++) { // 本地变量
    trace(counter); // 访问本地变量并显示 0 到 2
  }
}
count();
trace(counter); // 访问全局变量并显示 100
```

此示例只是表明在 count() 函数作用域内未访问该全局变量。但是，如果加上一个 _global 前缀，您就可以访问具有全局作用域的变量了。例如，如果为 counter 加上前缀 _global，就可以访问它，如下面的代码所示：

```
trace(_global.counter);
```

您不能为在 _global 作用域中创建的值指定严格的数据类型，因为在指定数据类型时必须使用 var 关键字。例如，您不能执行以下操作：

```
_global.foo:String = "foo";  // 语法错误
var _global.foo:String = "foo"; // 语法错误
```

通过从独立安全域加载的 SWF 文件访问全局变量时，Flash Player 第 7 版以及更高版本的安全性“沙箱”将强行施加限制。有关更多信息，请参见第 603 页的第 17 章“了解安全性”。

时间轴变量

时间轴变量可用于该特定时间轴上的任何脚本。要声明时间轴变量，请使用 var 语句并在该时间轴中的任一帧上初始化这些变量。该变量可以用于该帧和其后的所有帧，如下面的示例所示。

在文档中使用时间轴变量：

1. 创建一个新的 Flash 文档并将它命名为 **timelinevar.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

   ```
   var myNum:Number = 15; /* 在第 1 帧中初始化，这样它可用于所有帧 */
   ```
3. 选择时间轴上的第 20 帧。

4. 选择 “插入” > “时间轴” > “空白关键帧”。
5. 选择了新的关键帧之后，在 “动作” 面板中键入以下 ActionScript：

```
trace(myNum);
```

6. 选择 “控制” > “测试影片” 对该新文档进行测试。

   大约一秒钟之后，在 “输出” 面板中出现值 15。因为默认情况下 Flash 文档是循环播放的，所以每次播放头到达时间轴上的第 20 帧时，值 15 就会在 “输出” 面板中再次出现。要停止循环动作，请在 `trace()` 语句之后添加 `stop();`。

必须首先声明时间轴变量，然后再尝试在脚本中访问它。例如，如果将代码 `var myNum:Number = 15;` 放置在第 20 帧中，则附加到第 20 帧之前的任何帧上的脚本都无法访问 `myNum`，它们都是未定义的，而不包含值 `15`。

本地变量

在函数体内使用 `var` 语句时，声明的是本地变量。在函数块（又称函数定义）内声明一个本地变量时，该变量是在该函数块的作用域内定义的，因而在该函数块结束时会过期。因此，本地变量仅存在于该函数中。

例如，如果在名为 `localScope` 的函数中声明一个名为 `myStr` 的变量，该变量在该函数外部将不可用。

```
function localScope():Void {
  var myStr:String = "local";
}
localScope();
trace(myStr); // Undefined，因为 myStr 不是在全局作用域内定义的
```

如果用于本地变量的变量名已经被声明为时间轴变量，则当本地变量在作用域内时，本地定义优先于时间轴定义。时间轴变量在该函数外部仍然存在。例如，下面的代码创建了一个名为 `str1` 的时间轴字符串变量，然后在 `scopeTest()` 函数中创建一个具有相同名称的本地变量。该函数中的 `trace` 语句生成该变量的本地定义，而函数外部的 `trace` 语句生成变量的时间轴定义。

```
var str1:String = "Timeline";
function scopeTest():Void {
  var str1:String = "Local";
  trace(str1); // 本地
}
scopeTest();
trace(str1); // 时间轴
```

在下一个示例中您可以看到，某些变量仅在特定函数的生命期内有效，如果尝试在该函数的作用域外引用该变量，就会生成错误。

在应用程序中使用本地变量：

1. 创建一个新的 Flash 文档。
2. 打开“动作”面板（“窗口”>“动作”），在时间轴的第 1 帧上添加下面的 ActionScript：

```
function sayHello(nameStr:String):Void {
  var greetingStr:String = "Hello, " + nameStr;
  trace(greetingStr);
}
sayHello("world"); // Hello, world
trace(nameStr); // undefined
trace(greetingStr); // undefined
```

3. 选择 “控制” > “测试影片”来测试该文档。

   Flash 在“输出”面板中显示字符串“Hello, world”，并显示 `nameStr` 和 `greetingStr` 的值为 `undefined`，因为这些变量在当前的作用域内不再可用。仅能在 `sayHello` 函数执行时引用 `nameStr` 和 `greetingStr`。函数退出后，这些变量将不再存在。

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

```
var myArr:Array = new Array();
function initArray(arrayLength:Number):Void {
  var i:Number;
  for(i = 0; i < arrayLength; i++) {
    myArr[i] = i + 1;
  }
}
trace(myArr); // <blank>
initArray(3);
trace(myArr); // 1,2,3
trace(i); // undefined
```

注意

下面的 `for` 循环语法也很常见：`for (var i:Number = 0; i < arrayLength; i++) {...}`。

因为变量 `i` 不是在主时间轴中定义的，所以此示例在 Flash 测试环境中显示 `undefined`。它仅存在于 `initArray()` 函数中。

可以使用本地变量防止出现名称冲突，名称冲突可能会导致应用程序出现意外结果。例如，如果将 `age` 用作本地变量，可以用它在一个上下文中存储一个人的年龄，而在另一个上下文中存储此人的孩子的年龄。因为是在不同的作用域中使用这些变量的，因此在这种情况下不会有冲突。

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。只能在本地变量的代码块中对它进行更改。如果函数中的表达式使用全局变量，则该函数外部的代码或事件也可以更改它的值，这样也更改了该函数。

可以在声明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 284 页的 “关于指定数据类型和严格数据类型指定”。

关于加载变量

在以下几部分中，您将从服务器上以不同的方法加载变量，或将变量从 HTML 代码中的 URL 字符串或 FlashVars（可以使用 FlashVars 将变量传递到 Flash 中）中加载到一个文档中。这些练习演示了可以用几种方法来使用 SWF 文件外部的变量。

在第 565 页的第 16 章“使用外部数据”中可以找到有关加载变量（例如名称 / 值对）的更多信息。

在 SWF 文件中可以以不同的方式使用变量，具体取决于您需要用变量做什么。有关更多信息，请参见以下主题：

- 第 301 页的 “使用 URL 中的变量”
- 第 304 页的 “在应用程序中使用 FlashVars”
- 第 305 页的 “从服务器中加载变量”

使用 URL 中的变量

当您在 Flash 中开发应用程序或简单的示例时，可能需要将值从 HTML 页传递到 Flash 文档中。传递的值有时称为查询字符串 或 URL 编码变量。例如，如果要在 Flash 中创建一个菜单， URL 变量是非常有用的。可以将菜单初始化为默认情况下显示正确的导航。或者，可以在 Flash 中构建一个图像查看器并定义要在 Web 站点上显示的默认图像。

在文档中使用 URL 变量：

1. 创建一个 Flash 文档并将它命名为 **urlvariables.fla**。
2. 选择 “文件” > “另存为”，然后将文档保存到桌面上。
3. 在时间轴中选择第 1 帧，在 “动作”面板中，添加下面的代码：

```
this.createTextField("myTxt", 100, 0, 0, 100, 20);
myTxt.autoSize = "left";
myTxt.text = _level0.myURL;
```

4. 选择 “控制” > “测试影片”，在 Flash Player 中对 SWF 文件进行测试。

   文本字段显示 `undefined`。如果要在继续之前确保正确定义变量，需要检查 Flash 中是否存在变量。可以通过查看它们是否为未定义来实现此目的。

5. 要检查变量是否已定义，请修改在步骤 3 中添加到 “动作”面板中的 ActionScript 使它与下面的代码匹配。添加以**粗体**显示的代码：

```
this.createTextField("myTxt", 100, 0, 0, 100, 20);
myTxt.autoSize = "left";
if (_level0.myURL == undefined) {
  myTxt.text = "myURL is not defined";
} else {
  myTxt.text = _level0.myURL;
}
```

在发布 Flash 文档时，默认情况下会在 SWF 文件所在的目录下创建一个 HTML 文档。如果没有创建 HTML 文件，请选择 “文件” > “发布设置”，确保选择了 “格式”选项卡中的 HTML。然后再次发布文档。

以下代码演示了文档中负责将 Flash 文档嵌入到 HTML 页中的 HTML。需要查看此 HTML 以了解在下一步中（在其中为 URL 变量增加额外的代码）URL 变量是如何起作用的。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=8,0,0,0" width="550" height="400" id="urlvariables"
  align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="urlvariables.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<embed src="urlvariables.swf" quality="high" bgcolor="#ffffff"
  width="550" height="400" name="urlvariables" align="middle"
  allowScriptAccess="sameDomain" type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

6. 要将变量从生成的 HTML 文档中传递给 Flash 文档，可以在路径和文件名 (urlvariables.swf) 之后传递变量。将**粗体文本**添加到在桌面上生成的 HTML 文件中。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=8,0,0,0" width="550" height="400" id="urlvariables"
  align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="urlvariables.swf?myURL=http://
  weblogs.macromedia.com" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<embed src="urlvariables.swf?myURL=http://weblogs.macromedia.com"
  quality="high" bgcolor="#ffffff" width="550" height="400"
  name="urlvariables" align="middle" allowScriptAccess="sameDomain"
  type="application/x-shockwave-flash" pluginspage="http://
  www.macromedia.com/go/getflashplayer" />
</object>
```

7. 如果要将多个变量传递给 Flash，需要用“and”字符 (&) 分隔名称 / 值对。从步骤 6 中查找以下代码：

```
?myURL=http://weblogs.macromedia.com
```

用以下文本将其替换掉：

```
?myURL=http://weblogs.macromedia.com&myTitle=Macromedia+News+Aggregator
```

请注意，需要对 object 标签和 embed 标签做同样的更改，以保持所有浏览器之间的一致性。您可能会注意到单词被 + 号分隔开了。单词用这种方式隔开是因为值是 URL 编码的，+ 号表示单个空格。

注意 有关常见 URL 编码的特殊字符的列表，请参见 Flash 技术说明，URL 编码：从文本文件中读取特殊字符。

因为“and”字符 (&) 用作不同名称 / 值对的分隔符，所以如果传递的值包含“and”字符，可能会出现意外的结果。考虑到名称 / 值对的性质和语法分析，如果您将以下值传递到 Flash，

```
my.swf?name=Ben+&+Jerry&flavor=Half+Baked
```

Flash 将在根作用域中构建以下变量（和值）：

```
'name': 'Ben ' (note space at end of value)
' Jerry': '' (note space at beginning of variable name and an empty value)
'flavor': 'Half Baked'
```

为了避免这种情况，您需要将名称 / 值对中的“and”(&) 字符转义 为 URL 编码的等效字符 (%26)。

8. 打开 urlvariables.html 文档，并查找以下代码：

```
?myURL=http://weblogs.macromedia.com&myTitle=Macromedia+News+Aggregator
```

用以下代码替换该代码：

```
?myURL=Ben+%26+Jerry&flavor=Half+Baked
```

9. 保存修改的 HTML，再次测试 Flash 文档。

您会看到 Flash 创建了以下名称 / 值对。

```
'name': 'Ben & Jerry'
'flavor': 'Half Baked'
```

注意 所有浏览器都将支持长达 64K（65535 字节）的字符串长度。为了能在所有浏览器中使用，必须在 object 和 embed 标签上指定 FlashVars。

在应用程序中使用 FlashVars

使用 FlashVars 将变量传递到 Flash 中类似于随 HTML 代码中的 URL 传递变量。使用 FlashVars，不用在文件名后面传递变量，变量是在单独的 `param` 标签以及在 `embed` 标签中传递的。

在文档中使用 FlashVars：

1. 创建一个新的 Flash 文档并将它命名为 **myflashvars.fla**。
2. 选择“文件”>“发布设置”，确保选择了该 HTML，然后单击“确定”关闭对话框。
3. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
this.createTextField("myTxt", 100, 0, 0, 100, 20);
myTxt.autoSize = "left";
if (_level0.myURL == undefined) {
  myTxt.text = "myURL is not defined";
} else {
  myTxt.text = _level0.myURL;
}
```

> **附注** 默认情况下，HTML 代码发布到 myflashvars.fla 文件所在的位置。

4. 选择“文件”>“发布”来发布 SWF 和 HTML 文件。
5. 打开包含发布的文件的目录（硬盘上保存 myflashvars.fla 的位置）并在 HTML 编辑器（如 Dreamweaver 或 Notepad）中打开该 HTML 文档（默认情况下为 myflashvars.html）。
6. 添加下面以**粗体**显示的代码，使您的 HTML 文档与以下各项匹配：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=8,0,0,0" width="550" height="400" id="myflashvars"
  align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="myflashvars.swf" />
<param name="FlashVars" value="myURL=http://weblogs.macromedia.com/">
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<embed src="myflashvars.swf" FlashVars="myURL=http://
  weblogs.macromedia.com/" quality="high" bgcolor="#ffffff" width="550"
  height="400" name="myflashvars" align="middle"
  allowScriptAccess="sameDomain" type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

此代码传递名为 myURL 的单个变量，该变量中包含字符串 http://weblogs.macromedia.com。加载 SWF 文件时，在 _level0 作用域内创建一个名为 myURL 的属性。使用 FlashVars 或将变量随 URL 传递的优点之一是，在 SWF 文件加载时变量在 Flash 中便随即可用。这意味着无须编写任何函数来检查变量是否已完成加载，而在使用 LoadVars 或 XML 加载变量时需要编写函数进行检查。

7. 保存对 HTML 文档所做的更改，然后关闭它。
8. 双击 myflashvars.html 对应用程序进行测试。

   文本 http://weblogs.macromedia.com（HTML 文件中的一个变量）出现在 SWF 文件中。

注意	所有浏览器都将支持长达 64K（65,535 字节）的字符串长度。为了能在所有浏览器中使用，必须在 object 和 embed 标签上指定 FlashVars。

从服务器中加载变量

有几种方法可以将变量从外部源（例如，文本文件、XML 文档等）中加载到 Flash 中。在第 565 页的第 16 章“使用外部数据”中可以找到有关加载变量（包括名称 / 值对）的更详细的信息。

在 Flash 中，可以使用 LoadVars 类方便地加载变量，如下面的示例中所示。

从服务器中加载变量：

1. 创建一个新的 Flash 文档。
2. 在时间轴中选择第 1 帧，在“动作”面板中，添加下面的 ActionScript：

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean):Void {
  if (success) {
    trace(this.dayNames); // Sunday,Monday,Tuesday,...
  } else {
    trace("Error");
  }
}
my_lv.load("http://www.helpexamples.com/flash/params.txt");
```

   此代码从远程服务器上加载一个文本文件并分析它的名称 / 值对。

提示	如果要了解变量的格式，请在浏览器中下载或查看文本文件 (http://www.helpexamples.com/flash/params.txt)。

3. 选择“控制”>“测试影片”来测试该文档。

   如果文件加载成功，就会调用 complete 事件，并且“输出”面板中显示 dayNames 的值。如果无法下载该文本文件，success 参数就被设置为 false，同时“输出”面板中显示文本 Error。

在项目中使用变量

在使用 Flash 生成动画或应用程序时，在项目中不需要使用任何种类的变量的情形是很少见的。例如，如果要构建一个登录系统，可能需要使用变量来确定用户名和密码是否有效，或者它们是否根本没有被填写。

在第 565 页的第 16 章“使用外部数据”中可以找到有关加载变量（例如名称 / 值对）的更多信息。

在下面的示例中，将使用变量来存储使用 Loader 类加载的图像的路径，使用一个用于 Loader 类的实例的变量，并使用几个根据文件是否成功加载而调用的函数。

在项目中使用变量：

1. 创建一个新的 Flash 文档，并将它保存为 **imgloader.fla**。
2. 在时间轴中选择第 1 帧，在“动作”面板中，添加下面的 ActionScript：

```
/* 指定默认图像以备未用 FlashVars 传递值时使用。 */
var imgUrl:String = "http://www.helpexamples.com/flash/images/
  image1.jpg";
if (_level0.imgURL != undefined) {
  // 如果已指定了图像，将覆盖默认值。
  imgUrl = _level0.imgURL;
}

this.createEmptyMovieClip("img_mc", 10);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip):Void {
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
}
mclListener.onLoadError = function(target_mc:MovieClip):Void {
  target_mc.createTextField("error_txt", 1, 0, 0, 100, 20);
  target_mc.error_txt.autoSize = "left";
  target_mc.error_txt.text = "Error downloading specified image;\n\t" +
  target_mc._url;
}
var myMCL:MovieClipLoader = new MovieClipLoader();
myMCL.addListener(mclListener);
myMCL.loadClip(imgUrl, img_mc);
```

代码的第一行指定了要动态加载到 Flash 文档中的图像。接下来，检查是否使用 FlashVars 或 URL 编码的变量为 `imgURL` 指定了一个新值。如果已指定了一个新值，默认图像的 URL 将被该新值覆盖。有关使用 URL 变量的信息，请参见第 301 页的“使用 URL 中的变量”。有关 FlashVars 的信息，请参见第 304 页的“在应用程序中使用 FlashVars”。

代码中接下来的两行定义了 MovieClip 实例，和一个用于将来的 MovieClipLoader 实例的 Listener 对象。 MovieClipLoader 的 Listener 对象定义了两个事件处理函数，`onLoadInit` 和 `onLoadError`。在图像加载成功并在舞台上进行了初始化时，或在图像加载失败时，将调用这两个处理函数。然后您可以创建一个 MovieClipLoader 实例，使用 `addListener()` 方法将前面定义的 listener 对象添加到该 MovieClipLoader 中。最后，在调用 `MovieClipLoader.loadClip()` 方法（该方法指定要加载的图像文件和该图像要加载到的目标影片剪辑）时，将下载并触发图像。

3. 选择“控制”>“测试影片”来测试该文档。

   因为您是在创作工具中测试 Flash 文档，`imgUrl` 的任何值都不会通过 FlashVars 或随 URL 传递，因此将显示默认图像。

4. 保存该 Flash 文档，并选择“文件”>“发布”将该文件发布为 SWF 和 HTML 文档。

> **注意** 确保在“发布设置”对话框中同时选中了“Flash”和“HTML”。选择“文件”>“发布设置”，然后单击“格式”选项卡。然后，选择这两个选项。

5. 如果您在 Flash 工具（选择“控制”>“测试影片”）或在本地浏览器（“文件”>“发布预览”>“HTML”）中测试文档，您将看到图像在舞台上在垂直和水平方向上自动居中显示。

6. 在编辑器（例如 Dreamweaver 或记事本）中编辑生成的 HTML 文档，并修改默认的 HTML 使它与以下文本匹配：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=8,0,0,0" width="550" height="400" id="urlvariables"
  align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="urlvariables.swf" />
<param name="FlashVars" value="imgURL=http://www.helpexamples.com/flash/
  images/image2.jpg">
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<embed src="urlvariables.swf" quality="high" FlashVars="imgURL=http://
  www.helpexamples.com/flash/images/image2.jpg" bgcolor="#ffffff"
  width="550" height="400" name="urlvariables" align="middle"
  allowScriptAccess="sameDomain" type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

7. 测试 HTML 文档以查看所做的更改。您在 HTML 代码中指定的图像出现在 SWF 文件中。

   要修改此示例以使用自己的图像，应该修改 FlashVars 值（双引号中的字符串）。

用对象组织数据

您可能已经习惯了放置在舞台上的对象。例如，舞台上可能有一个 MovieClip 对象，并且该对象内包含其它影片剪辑。在放置到舞台上时，文本字段、影片剪辑和按钮通常被称为对象。

在 ActionScript 中，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。内置对象来自在 ActionScript 中预定义的类。例如，内置的 Date 类可以提供用户的计算机上的系统时钟的信息。可以使用内置的 LoadVars 类将变量加载到 SWF 文件中。

还可以使用 ActionScript 创建对象和类。可以创建一个对象来保存数据的集合，例如一个人的姓名、地址和电话号码。可以创建一个对象来保存图像的颜色信息。用对象组织数据有助于更好地组织 Flash 文档。有关创建自定义类以保存方法和属性集合的一般信息，请参见第 170 页的 “编写自定义类文件”。有关内置类和自定义类的详细信息，请参见第 163 页的第 6 章 “类”。

在 ActionScript 中创建一个对象有多种方法。下一示例用两种不同的方法创建了简单对象，然后循环访问这些对象的内容。

在 Flash 中创建简单对象：

1. 创建一个新的 Flash 文档，并将它保存为 **simpleObjects.fla**。
2. 在时间轴中选择第 1 帧，在 “动作” 面板中，键入下面的 ActionScript：

```
// 第一种方法
var firstObj:Object = new Object();
firstObj.firstVar = "hello world";
firstObj.secondVar = 28;
firstObj.thirdVar = new Date(1980, 0, 1); // 1980 年 1 月 1 日
```

   此代码是创建简单对象的一种方法，它创建了一个新的对象实例并在该对象中定义了几个属性。

3. 现在，在步骤 2 中输入的代码后面输入下面的 ActionScript：

```
// 第二种方法
var secondObj:Object = {firstVar:"hello world", secondVar:28,
  thirdVar:new Date(1980, 0, 1)};
```

   这是创建对象的另一种方法。两个对象是等效的。以上代码创建一个新的对象，并使用对象简化记号初始化了一些属性。

4. 要循环访问前面的每一个对象并显示对象的内容，请将以下 ActionScript 添加到时间轴的第 1 帧上 （在已经输入的代码之后）：

```
var i:String;
for (i in firstObj) {
  trace(i + ": " + firstObj[i]);
}
```

5. 选择 “控制” > “测试影片”，“输出”面板中出现以下文本：

```
firstVar: hello world
secondVar: 28
thirdVar: Tue Jan 1 00:00:00 GMT-0800 1980
```

也可以使用数组来创建对象。可以不使用一系列变量（例如 firstname1、firstname2 和 firstname3）来表示变量的集合，而是用一个对象的数组来表示此同一数据。下面将演示这一做法。

使用数组创建对象：

1. 创建一个新的 Flash 文档，并将它保存为 **arrayObject.fla**。
2. 在时间轴中选择第 1 帧，在 “动作”面板中，键入下面的 ActionScript：

```
var usersArr:Array = new Array();
usersArr.push({firstname:"George"});
usersArr.push({firstname:"John"});
usersArr.push({firstname:"Thomas"});
```

将变量组织到数组和对象中的好处是，可以更方便地循环访问变量并查看值，如以下步骤中所示。

3. 在步骤 2 中添加的 ActionScript 之后键入以下代码：

```
var i:Number;
for (i = 0; i < usersArr.length; i++) {
  trace(usersArr[i].firstname); // George, John, Thomas
}
```

4. 选择 “控制” > “测试影片”，“输出”面板中将显示以下文本：

```
George
John
Thomas
```

下面的示例介绍了循环访问对象的另一种方法。在此示例中，将创建一个对象并使用 for..in 循环对其执行循环，每个属性都出现在 “输出”面板中：

```
var myObj:Object = {var1:"One", var2:"Two", var3:18, var4:1987};
var i:String;
for (i in myObj) {
  trace(i + ": " + myObj[i]);
}
// 输出以下结果：
/*
    var1:One
    var2:Two
    var3: 18
    var4: 1987
*/
```

有关创建 for 循环的信息，请参见第 102 页的第 4 章 “使用 for 循环”。有关 for...in 循环的信息，请参见第 103 页的 “使用 for..in 循环”。有关对象的更多信息，请参见第 163 页的第 6 章 “类”。

关于转换

ActionScript 2.0 允许将一种数据类型转换为另一种数据类型。将对象转换为不同的类型意味着将对象或变量中保存的值转换为不同的类型。

类型转换的结果因涉及的数据类型而异。要将对象转换为不同的类型，请用小括号 (()) 括起对象名并在它前面加上新类型的名称。例如，以下代码取一个布尔值并将它转换为一个整数。

```
var myBoolean:Boolean = true;
var myNumber:Number = Number(myBoolean);
```

有关转换的更多信息，请参见以下主题：

- 第 310 页的 “关于转换对象”

关于转换对象

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，如果转换在运行时失败，该函数调用将返回 `null`（仅在为 Flash Player 7 或更高版本发布的文件中会出现这种情况；为 Flash Player 6 发布的文件不为失败的转换提供运行时支持）。如果转换成功，该函数调用将返回原对象。不过，编译器无法确定转换在运行时是否会失败，也不会在失败时生成编译时错误。

下面的代码显示了一个示例：

```
// Both the Cat and Dog classes are subclasses of the Animal class
function bark(myAnimal:Animal) {
  var foo:Dog = Dog(myAnimal);
  foo.bark();
}
var curAnimal:Animal = new Dog();
bark(curAnimal); // 起作用
curAnimal = new Cat();
bark(curAnimal); // 不起作用
```

在此示例中，您向编译器声明了 foo 是一个 Dog 对象，因此，编译器认为 foo.bark(); 是一个合法语句。但是，编译器不知道该转换将失败（即，您尝试将一个 Cat 对象转换为 Animal 类型），因此不会发生任何编译时错误。不过，如果在脚本中加入一条检查语句，检查转换是否成功，在运行时便能够发现转换错误，如下面的示例中所示。

```
function bark(myAnimal:Animal) {
  var foo:Dog = Dog(myAnimal);
  if (foo) {
    foo.bark();
  }
}
```

您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，则转换将成功。如果不是，转换将失败。

提示	转换为 null 或未定义会返回 undefined。

不能使用转换运算符覆盖具有相应全局转换函数的同名原始数据类型。这是因为全局转换函数的优先级高于转换运算符。例如，您无法转换成 Array，因为 Array() 转换函数的优先级高于转换运算符。

此示例定义了两个字符串变量（firstNum 和 secondNum），并将它们加在了一起。初始结果是数字被连接在一起，而不是加在一起，因为它们是 String 数据类型。第二个 trace 语句在执行加法之前将两个数字转换为 Number 数据类型，从而得到正确的结果。在使用通过 XML 或 FlashVars 加载的数据时数据转换是非常重要的，如下面的示例中所示：

```
var firstNum:String = "17";
var secondNum:String = "29";
trace(firstNum + secondNum); // 1729
trace(Number(firstNum) + Number(secondNum)); // 46
```

有关数据转换函数的更多信息，请参见《ActionScript 2.0 语言参考》中每个转换函数的条目：Array 函数、Boolean 函数、Number 函数、Object 函数和 String 函数。

第 11 章

使用影片剪辑

11

影片剪辑就像自包含的 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。

您可以命名影片剪辑实例，以将它们唯一地标识为可使用 ActionScript 控制的对象。当您向影片剪辑实例提供实例名称 时，该实例名称即将其标识为 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 Basic 8 和 Macromedia Flash Professional 8 中基于组件的体系结构 的基础。事实上，“组件”面板（“窗口”>“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。

有关使用 Drawing API（MovieClip 类中的绘画方法）、滤镜、混合、脚本动画等的信息，请参见第 13 章 “动画、滤镜和绘画”。

有关影片剪辑的更多信息，请参见以下主题：

关于通过 ActionScript 控制影片剪辑

您可以使用全局 ActionScript 函数或 MovieClip 类的方法对影片剪辑执行任务。某些 MovieClip 类中的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。

下面的示例显示了使用方法和使用函数之间的差异。每个语句都复制实例 `my_mc`，将新剪辑命名为 `new_mc`，然后将它放在第 5 层。

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

当函数和方法提供相似的行为时，您可以选择使用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 ActionScript 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。

若要使用方法，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的实例的目标路径。例如，在下面的脚本中，`startDrag()` 以代码所在的实例为目标并使该实例可拖动：

```
my_mc.onPress = function() {
  startDrag(this);
};
my_mc.onRelease = function() {
  stopDrag();
};
```

以下函数针对影片剪辑：`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，它们没有等同的函数：`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()` 和 `MovieClip.swapDepths()`。

有关这些函数和方法的更多信息，请参见《ActionScript 2.0 语言参考》中的相应条目。

有关 Flash 中脚本动画的示例，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

在硬盘上可以找到图库应用的范例。这些文件提供了示例，说明如何在向 SWF 文件加载范例源文件时使用 ActionScript 动态控制影片剪辑，其中包括脚本动画。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

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 ActionScript 对象（如 Array、Color 和 Sound），而不仅仅是影片剪辑。

`with` 语句使用影片剪辑作为参数。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，对象 `donut.hole` 将传递至 `with` 语句以更改 `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。前面的代码等同于以下示例：

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

前面的代码也等同于以下示例：

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

加载和卸载 SWF 文件

若要播放其它 SWF 文件而不必关闭 Flash Player，或若要切换 SWF 文件而不必加载另一个 HTML 页面，可以使用以下选项之一：

- 全局 `loadMovie()` 函数或 MovieClip 类的 `loadMovie()` 方法。
- MovieClipLoader 类的 `loadClip()` 方法。有关 MovieClipLoader 类的更多信息，请参见《ActionScript 2.0 语言参考》中的 `MovieClipLoader`。

也可以使用 `loadMovie()` 方法将变量发送给 CGI 脚本，该脚本会生成一个 SWF 文件作为它的 CGI 输出。例如，您可以使用此过程根据影片剪辑中指定的变量加载动态 SWF 或图像文件。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。 Flash 影片加载完毕后，您就可以更改这些属性了。

`unloadMovie()` 方法删除以前由 `loadMovie()` 方法加载的 SWF 文件。用 `unloadMovie()` 显式卸载 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。某些情况下，将影片剪辑的 `_visible` 属性设置为 `false` 比卸载剪辑更有效。如果要在以后重复使用该剪辑，可将 `_visible` 属性设置为 `false`，在必要时再设置为 `true`。

使用 `loadMovie()` 可以执行以下的任何任务：

- 通过将 `loadMovie()` 函数放置在按顺序加载和卸载 SWF 横幅广告文件的容器 SWF 文件中，播放由 SWF 文件构成的横幅广告序列。
- 开发一个分支接口，其链接可让用户在用于显示站点内容的多个 SWF 文件中进行选择。
- 构建一个导航界面，其导航控件位于级别 0，用于将内容加载到其它级别。与在浏览器中加载新的 HTML 页面相比，将内容加载到其它级别有助于实现页面内容之间的更平滑过渡。

有关加载 SWF 文件的更多信息，请参见第 529 页的 “加载外部 SWF 和图像文件”。

有关更多信息，请参见以下主题：

- 第 317 页的 “指定加载的 SWF 文件的根时间轴”
- 第 318 页的 “将图像文件加载到影片剪辑中”

指定加载的 SWF 文件的根时间轴

_root ActionScript 属性指定或包含对 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。

在下面的示例中，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");
my_btn.onRelease = function():Void {
  trace(_root.userName);
};
```

在下面的示例中，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 contents.swf 中:
_root.userName = "Mary";
```

contents.swf 加载到 container.swf 中的影片剪辑之后，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°±，而不是 °×Tim°±。这将导致 container.swf（以及 contents.swf）中的代码出现问题。

若要强制 _root 始终计算所加载的 SWF 文件的时间轴，而不是实际的根时间轴，请使用 _lockroot 属性。可通过加载 SWF 文件或已加载的 SWF 文件来设置此属性。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 的作者可在主时间轴的第 1 帧上放置下面的代码：

```
// 添加至 container.swf 中的第 1 帧:
target_mc._lockroot = true;
```

此步骤确保所有对 contents.swf 或任何加载到 target_mc 中的 SWF 文件中的 _root 的引用都将引用其自己的时间轴，而不是 container.swf 的实际根时间轴。现在，单击该按钮时，就会显示 “Tim”。

或者， contents.swf 的作者可以将下面的代码添加到其主时间轴中：

```
// 添加至 contents.swf 中的第 1 帧:
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 _lockroot （MovieClip._lockroot 属性）。

将图像文件加载到影片剪辑中

您可以使用 loadMovie() 函数或同名的 MovieClip 方法将图像文件加载到影片剪辑实例中。还可以使用 loadMovieNum() 函数将图像文件加载到某个级别。

当您将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。因为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，图像的左上角将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《ActionScript 2.0 语言参考》中的 loadMovie 函数、loadMovie (MovieClip.loadMovie 方法) 和 loadMovieNum 函数以及第 529 页的 “加载外部 SWF 和图像文件”。

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：

```
my_mc._rotation = 45;
```

上面的语句等同于下面使用 setProperty() 函数的代码：

```
setProperty("my_mc", _rotation, 45);
```

有些属性称为只读属性，其值只能读取而不能设置。（这些属性在它们的《ActionScript 2.0 语言参考》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_mc 的 _alpha 属性，该实例是 car_mc 实例的子级：

```
car_mc.wheel_mc._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 my_mc 实例的 _x 属性设置为该值：

```
this.onEnterFrame = function() {
  my_mc._x = _root._xmouse;
};
```

上面的语句等同于下面使用 getProperty() 函数的代码：

```
this.onEnterFrame = function() {
  my_mc._x = getProperty(_root, _xmouse);
};
```

_x、_y、_rotation、_xscale、_yscale、_height、_width、_alpha 和 _visible 属性受影片剪辑的父级的变形影响，并可使影片剪辑及其子级变形。_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《ActionScript 2.0 语言参考》中 MovieClip 类的属性摘要。

有关 Flash 中脚本动画的示例，可以在硬盘上的 Samples 文件夹下找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

在硬盘上可以找到图库应用的范例。这些文件提供了示例，说明如何在向 SWF 文件加载图像文件时使用 ActionScript 动态控制影片剪辑，其中包括脚本动画。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

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。SWF 文件中，同一时间只有一个影片剪辑是可拖动的。

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已被拖到一个特定的影片剪辑（如“trash can”影片剪辑）上，然后触发另一个动作，如下例所示：

```
// 拖动一部分垃圾。
garbage_mc.onPress = function() {
  this.startDrag(false);
};
// 垃圾拖动到 trashcan 上时，使垃圾不可见。
garbage_mc.onRelease = function() {
  this.stopDrag();
  // 使用 eval 将斜杠记号转换为点记号。
  if (eval(this._droptarget) == trashcan_mc) {
    garbage_mc._visible = false;
  }
};
```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `startDrag 函数`或 `startDrag (MovieClip.loadMovie 方法)`。

在硬盘上可以找到图库应用的范例。这些文件提供了示例，说明如何在向 SWF 文件加载图像文件时使用 ActionScript 动态控制影片剪辑，其中包括将各个影片剪辑设为可拖动的。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

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，还可以在运行时通过以下方式创建影片剪辑实例：

- 第 322 页的 “创建空白影片剪辑”
- 第 323 页的 “复制或删除影片剪辑”
- 第 323 页的 “将影片剪辑元件附加到舞台”

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。它还允许您覆盖位于同一深度的影片剪辑。（请参见第 326 页的 “管理影片剪辑的深度”。）

在硬盘上可以找到图库应用的范例。这些文件提供了示例，说明如何在向 SWF 文件加载图像文件时使用 ActionScript 动态控制影片剪辑，其中包括在运行时创建影片剪辑。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

有关在运行时创建和删除多个影片剪辑的示例源文件，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

有关更多信息，请参见以下主题：

- 第 322 页的 “创建空白影片剪辑”
- 第 323 页的 “复制或删除影片剪辑”
- 第 323 页的 “将影片剪辑元件附加到舞台”

创建空白影片剪辑

若要在舞台上创建一个新的空白影片剪辑，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，然后激活 `loadMovie()` 将一个外部 JPEG 文件加载到自身中：

```
this.createEmptyMovieClip("canvas_mc", 10);
canvas_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

如下面的示例所示，您可以将图像 image2.jpg 加载到影片剪辑中，然后使用 `MovieClip.onPress()` 方法将该图像用作按钮。使用 `loadMovie()` 加载图像会用该图像替换影片剪辑，但不会让您访问影片剪辑方法。若要访问影片剪辑方法，您必须创建一个空白父级影片剪辑以及一个容器子级影片剪辑。将图像加载到容器中，然后将事件处理函数放置在父级影片剪辑上。

```
// 创建一个父影片剪辑以放置该容器。
this.createEmptyMovieClip("my_mc", 0);

// 在 “my_mc” 内创建一个子影片剪辑。
// 这是图像将替换的影片剪辑。
my_mc.createEmptyMovieClip("container_mc",99);

// 使用 MovieClipLoader 加载图像。
var my_mcl:MovieClipLoader = new MovieClipLoader();
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image2.jpg",
  my_mc.container_mc);

// 将事件处理函数放置在父影片剪辑 my_mc 上。
my_mc.onPress = function():Void {
  trace("It works");
};
```

有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `createEmptyMovieClip (MovieClip.createEmptyMovieClip 方法)`。

有关在运行时创建和删除多个影片剪辑的示例源文件，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

复制或删除影片剪辑

若要直接复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）。重复的影片剪辑始终从第 1 帧开始，即使原始影片剪辑在被直接复制时位于其它帧也如此，并且，重复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，那么直接复制的影片剪辑也会被删除。

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 duplicateMovieClip 函数和 removeMovieClip 函数。

有关在运行时创建和删除多个影片剪辑的示例源文件，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

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() 方法。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 ActionScript 从库中附加一个影片剪辑元件，必须为 ActionScript 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 ActionScript 而导出的所有影片剪辑都将在包含它们的 SWF 文件的第 1 帧之前加载。这可能会造成在第 1 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 1 帧之前加载该内容。如果没有将该元素添加到第 1 帧中，则必须将它的实例包含在 SWF 文件的其它某个帧中；否则将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”面板弹出菜单中选择 “链接”。

   随即出现 “链接属性”对话框。
4. 对于 “链接”，选择 “为 ActionScript 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。

   您还可以将 ActionScript 的类分配给影片剪辑元件。这使影片剪辑可以继承指定类的方法和属性。（请参见第 338 页的 “将类分配给影片剪辑元件”。）

6. 如果不想在第 1 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第 10 帧上或第 10 帧之前。

7. 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. 如上例所述，将链接标识符分配给影片剪辑库元件。
2. 在 “动作”面板 （“窗口” > “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 **this**。

4. 在 “动作”工具箱（位于 “动作”面板左侧）中，选择 “ActionScript 2.0 类” > “影片” > “MovieClip” > “方法”，然后选择 `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`，输入直接复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，如下例所示：

     ```
     this.attachMovie("calif_id", "california_mc", 10);
     ```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `attachMovie (MovieClip.attachMovie 方法 )`。

将参数添加到动态创建的影片剪辑中

使用 MovieClip.attachMovie() 和 MovieClip.duplicateMovie() 动态创建或复制影片剪辑时，可以用另一对象中的参数填充影片剪辑。attachMovie() 和 duplicateMovie() 的 *initObject* 参数允许动态创建的影片剪辑接收剪辑参数。

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 attachMovie（MovieClip.attachMovie 方法）和 duplicateMovieClip（MovieClip.duplicateMovieClip 方法）。

使用指定对象的参数填充动态创建的影片剪辑：

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 duplicateMovie() 的语法：
  myMovieClip.duplicateMovie(*idName*, *newName*, *depth* [, *initObject*]);

initObject 参数指定您要使用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。
2. 在 “元件名称”文本框中键入 **dynamic_mc**，然后选择 “影片剪辑”行为。
3. 在该元件内部，使用名为 **name_txt** 的实例在舞台上创建一个动态文本字段。

   确保此文本字段位于注册点下方的右侧。
4. 选择影片剪辑的时间轴的第 1 帧，然后打开 “动作”面板 （“窗口” > “动作”）。
5. 创建一个名为 **name_str** 的新变量，然后将其值分配给 name_txt 的 text 属性，如下面的示例所示：

   ```
   var name_str:String;
   name_txt.text = name_str;
   ```
6. 选择 “编辑” > “编辑文档”返回到主时间轴。
7. 在库中选择影片剪辑元件，然后从 “库”弹出菜单中选择 “链接”。

   随即出现 “链接属性”对话框。
8. 选择 “为 ActionScript 导出”选项和 “在第一帧导出”。
9. 在 “标识符”文本框中键入 **dynamic_id**，然后单击 “确定”。

10. 选择主时间轴的第 1 帧并将以下代码添加到 “动作”面板的 “脚本”窗格中：

```
/* 附加一个影片剪辑并将其移动到 x 和 y 坐标均为 50 的位置 */
this.attachMovie("dynamic_id", "newClip_mc", 99, {name_str:"Erick",
  _x:50, _y:50});
```

11. 对 Flash 文档进行测试 （“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。

在硬盘上可以找到图库应用的范例。这些文件提供了示例，说明如何在向 SWF 文件加载图像文件时使用 ActionScript 动态控制影片剪辑，其中还包括在运行时创建影片剪辑。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

有关在运行时创建和删除多个影片剪辑的示例源文件，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

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，该值用来确定它是呈现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。使用 attachMovie (MovieClip.attachMovie 方法)、duplicateMovieClip (MovieClip.duplicateMovieClip 方法) 或 createEmptyMovieClip (MovieClip.createEmptyMovieClip 方法) 在运行时创建影片剪辑时，始终指定新剪辑的深度作为方法参数。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", "clip1_mc", 10);
```

此示例在 container_mc 的 z 顺序空间中创建一个深度为 10 的新影片剪辑。

下面的代码将两个新影片剪辑附加到 container_mc。第一个名为 clip1_mc 的剪辑将呈现在 clip2_mc 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", "clip1_mc", 10);
container_mc.attachMovie("symbolID", "clip2_mc", 15);
```

影片剪辑的深度值范围为 -16384 至 1048575。如果在已经存在影片剪辑的深度上创建或附加新的影片剪辑，则新建或附加的剪辑将覆盖现有内容。若要避免此问题，请使用 MovieClip.getNextHighestDepth() 方法，但是不要对使用不同深度管理系统的组件使用此方法。而是对组件实例使用 “DepthManager 类”。

MovieClip 类提供了几种管理影片剪辑深度的方法；有关更多信息，请参见《ActionScript 2.0 语言参考》中的 getNextHighestDepth (MovieClip.getNextHighestDepth 方法)、getInstanceAtDepth (MovieClip.getInstanceAtDepth 方法)、getDepth (MovieClip.getDepth 方法) 和 swapDepths (MovieClip.swapDepths 方法)。

有关影片剪辑深度的更多信息，请参见以下主题：

- 第 327 页的 “确定下一个最大的可用深度”
- 第 328 页的 “确定处于特定深度的实例”
- 第 328 页的 “确定实例的深度”
- 第 328 页的 “交换影片剪辑的深度”

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。

下面的代码在名为 file_mc 的根时间轴上附加一个新影片剪辑，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名为 edit_mc 新影片剪辑。

```
this.attachMovie("menuClip","file_mc", 10, {_x:0, _y:0});
trace(file_mc.getDepth()); // 10
var nextDepth:Number = this.getNextHighestDepth();
this.attachMovie("menuClip", "edit_mc", nextDepth, {_x:200, _y:0});
trace(edit_mc.getDepth()); // 11
```

在本例中，名为 nextDepth 的变量包含值 11，因为这是影片剪辑 edit_mc 的下一个最大的可用深度。

不要对组件使用 MovieClip.getNextHighestDepth()，而要使用深度管理器。有关更多信息，请参见《组件语言参考》中的 “DepthManager 类”。有关 MovieClip.getNextHighestDepth() 的更多信息，请参见 getNextHighestDepth (MovieClip.getNextHighestDepth 方法)。

若要获得当前占用的最大深度，从 getNextHighestDepth() 返回的值中减 1 即可，如下一节所示。

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（当前）最大深度的影片剪辑。

```
var highestOccupiedDepth:Number = this.getNextHighestDepth() - 1;
var instanceAtHighestDepth:MovieClip =
  this.getInstanceAtDepth(highestOccupiedDepth);
```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 getInstanceAtDepth (MovieClip.getInstanceAtDepth 方法)。

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 (var item:String in _root) {
  var obj:Object = _root[item];
  if (obj instanceof MovieClip) {
    var objDepth:Number = obj.getDepth();
    trace(obj._name + ":" + objDepth)
  }
}
```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 getDepth (MovieClip.getDepth 方法)。

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()。以下示例演示了如何在运行时交换两个影片剪辑实例的深度。

交换影片剪辑的深度：

1. 创建一个名为 **swap.fla** 的新 Flash 文档。
2. 在舞台上绘制一个蓝色的圆形。
3. 选择所绘蓝色圆形，然后选择“修改”>“转换为元件”。
4. 选择“影片剪辑”选项，然后单击“确定”。
5. 选择舞台上的实例，然后在属性检查器的“实例名称”文本框中键入 **first_mc**。
6. 在舞台上绘制一个红色的圆形，然后选择“修改”>“转换为元件”。

7. 选择 “影片剪辑”选项，然后单击 “确定”。
8. 选择舞台上的实例，然后在属性检查器的 “实例名称”文本框中键入 **second_mc**。
9. 拖动这两个实例，使它们在舞台上有一点重叠。
10. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
first_mc.onRelease = function() {
      this.swapDepths(second_mc);
   };
   second_mc.onRelease = function() {
      this.swapDepths(first_mc);
   };
```

11. 选择 “控制” > “测试影片”对该文档进行测试。

    单击舞台上的实例时，它们将交换深度。您将看到这两个实例更改两个剪辑的位置，即位于底层的将位于上层。

有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `swapDepths (MovieClip.swapDepths 方法)`。

关于使用 ActionScript 缓存和滚动影片剪辑

如果 Flash 中的设计尺寸增大，无论创建的是应用程序还是复杂的脚本动画，都需要考虑性能和优化。如果您的内容保持为静态（如矩形影片剪辑），Flash 将不会优化该内容。因此，在更改矩形影片剪辑的位置时， Flash 将在 Flash Player 7 及早期版本中重绘整个矩形。

在 Flash Player 8 中，可以缓存指定的影片剪辑和按钮以提高 SWF 文件的性能。影片剪辑或按钮是一个表面，实际上是实例的矢量数据的位图版本，矢量数据是一种在 SWF 文件的过程不需要有太多更改的数据。因此，打开了缓存的实例在播放 SWF 文件的过程中不会不断地重绘，这样 SWF 文件就能快速地呈现。

注意	可以更新矢量数据，这时将重新创建表面。因此，缓存在表面中的矢量数据不需要在整个 SWF 文件中保持一样。

可以使用 ActionScript 来启用缓存或滚动和控制背景。可以使用属性检查器对影片剪辑实例启用缓存。若要在不使用 ActionScript 的情况下缓存影片剪辑或按钮，可以在属性检查器中选择 “使用运行时位图缓存”选项。

下表包含了对影片剪辑实例的新属性的简要说明：

属性	说明
cacheAsBitmap	使影片剪辑实例缓存其自身的位图表示。 Flash 为该实例创建一个 surface 对象，该对象是一个缓存的位图，而不是矢量数据。如果要更改影片剪辑的范围，则表面会重新构建而不是重新调整。有关更多信息和示例，请参见第 333 页的“缓存影片剪辑”。
opaqueBackground	使您可以指定不透明影片剪辑实例的背景颜色。如果将此属性设置为数值，则影片剪辑实例将具有一个不透明（非透明）的表面。不透明位图不具有 Alpha 通道（透明度），可以更快地呈现。有关更多信息和示例，请参见第 335 页的“设置影片剪辑的背景”。
scrollRect	使您可以快速滚动影片剪辑内容并可以用一个窗口查看更大的内容。将裁切影片剪辑内容，且实例会按指定的宽度、高度和滚动偏移而滚动。这样，用户可以快速滚动影片剪辑内容，并用一个窗口显示比 “舞台” 区域更大的内容。显示在实例中的文本字段和复杂内容可以更快速地滚动，因为 Flash 不需要重新生成整个影片剪辑矢量数据。有关更多信息和示例，请参见 `scrollRect (MovieClip.scrollRect 属性)`。

这三个属性互相独立，但是，当对象被缓存为位图时，`opaqueBackground` 和 `scrollRect` 属性的作用最佳。将 `cacheAsBitmap` 设置为 `true` 时，您只能看到 `opaqueBackground` 和 `scrollRect` 属性的性能优点。

若要创建也可以滚动的表面，必须设置影片剪辑实例的 `cacheAsBitmap` 和 `scrollRect` 属性。表面可以嵌套在其它表面之内。表面将把位图复制到它的父表面上。

有关 Alpha 通道遮罩（要求将 `cacheAsBitmap` 属性设置为 `true`）的信息，请参见第 337 页的 “关于 Alpha 通道遮罩”。

注意

不能将缓存直接应用于文本字段。需要将文本放在影片剪辑中以利用此功能。有关示例，请参见 Flash install directory\Samples and Tutorials\Samples\ActionScript\FlashType 中的范例文件。

您可以找到对应的范例源文件，该文件说明了如何对实例应用位图缓存。可以在硬盘上的 Samples 文件夹中找到名为 cacheBitmap.fla 的文件。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\CacheBitmap。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/CacheBitmap。

您也可以找到对应的范例源文件，该文件说明了如何对滚动文本应用位图缓存。可以在硬盘上的 Samples 文件夹中找到范例源文件 flashtype.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

何时启用缓存

对影片剪辑启用缓存可以创建一个表面，这有多种优势，如有助于更快速地呈现复杂的矢量动画。有几种情形需要启用缓存。您可能总希望通过启用缓存来提高 SWF 文件的性能；但是，在某些情况下启用缓存不能提高性能，甚至还会降低性能。本节介绍在哪些情形下应使用缓存，以及何时使用常规影片剪辑。

缓存数据的总体性能取决于实例的矢量数据的复杂程度、要更改的数据量，以及是否设置了 `opaqueBackground` 属性。如果要更改的范围较小，则使用表面和使用矢量数据的差异微乎其微。在部署应用程序之前您可能需要实际测试一下这两种情形。

有关 Alpha 通道遮罩（要求将 `cacheAsBitmap` 属性设置为 `true`）的信息，请参见第 337 页的 “关于 Alpha 通道遮罩”。

何时使用位图缓存

在以下典型情形中启用位图缓存您可能会看到明显的好处。

复杂背景图像 包含矢量数据的详细的复杂背景图像的应用程序（可能是应用了跟踪位图命令的图像，也可能是在 Adobe Illustrator 中创建的图片）。您可能会在背景上设计动画字符，这样会降低动画速度，因为背景需要持续地重新生成矢量数据。若要提高性能，可以选择内容，将其存储到影片剪辑中，然后将 `opaqueBackground` 属性设置为 `true`。背景将呈现为位图，可以迅速地重新绘制，以便更快地播放动画。

滚动文本字段 在滚动文本字段中显示大量文本的应用程序。可以将文本字段放置在通过滚动框（`scrollRect` 属性）设置为可滚动的影片剪辑中。这可以使指定的实例进行快速像素滚动。当用户滚动影片剪辑实例的时候，Flash 把滚动过的像素移向上方，生成新出现的区域，而不是重新生成整个文本字段。

窗口系统 具有重叠窗口的复杂系统的应用程序。每个窗口都可以打开或关闭（例如，Web 浏览器窗口）。如果将每个窗口标记为一个表面（将 `cacheAsBitmap` 属性设置为 `true`），则各个窗口将隔离开来并进行缓存。用户可以拖动窗口使其互相重叠，每个窗口无需重新生成矢量内容。

所有这些情形通过优化矢量图形可以提高应用程序的响应能力和互动性。

您可以找到对应的范例源文件，该文件说明了如何对实例应用位图缓存。可以在硬盘上的 Samples 文件夹中找到名为 cacheBitmap.fla 的文件。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\CacheBitmap。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/CacheBitmap。

您也可以找到对应的范例源文件，该文件说明了如何对滚动文本应用位图缓存。可以在硬盘上的 Samples 文件夹中找到范例源文件 flashtype.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

何时避免使用位图缓存

滥用此功能会给 SWF 文件带来负面影响。在开发使用表面的 FLA 文件时，要牢记以下原则：

- 不要过度使用表面 （启用了缓存的影片剪辑） 。每个表面使用的内存都比常规影片剪辑多，这意味着只能在需要提高呈现性能时启用表面。

  缓存的位图使用的内存比常规影片剪辑实例多很多。例如，如果舞台上的影片剪辑大小为 250 x 250 像素，对它进行缓存时可能会使用 250 KB 内存，如果它是常规（未缓存的）影片剪辑实例，则使用 1 KB 内存。
- 避免缩小到缓存的表面。如果过度使用位图缓存，尤其是在缩小内容时，将使用大量内存 （请参见上一段落）。
- 对主要为静态 （非动画）的影片剪辑实例使用表面。可以拖放或移动实例，但实例的内容不应动起来或更改太多。例如，如果旋转或转换实例，实例将在表面和矢量数据之间改变，这种情况难于处理，并会对 SWF 文件产生负面影响。
- 如果将表面和矢量数据混在一起，将增加 Flash Player （有时还有计算机）需要处理的工作量。尽可能将表面归为一组，例如，创建窗口应用程序时。

缓存影片剪辑

若要缓存影片剪辑实例，需要将 cacheAsBitmap 属性设置为 true。在将 cacheAsBitmap 属性设置为 true 后，您可能会注意到，影片剪辑实例将自动贴紧至整个坐标像素。在测试 SWF 文件时，还会注意到，任何复杂的矢量动画都将以快得多的速度呈现出来。

即便是将 cacheAsBitmap 设置为 true，如果出现以下一种或多种情形，也将不创建表面（缓存的位图）：

- 位图高度或宽度超过 2880 像素。
- 位图无法分配（内存不足错误）。

缓存影片剪辑：

1. 创建一个新的 Flash 文档，并将文件命名为 **cachebitmap.fla**。
2. 在属性检查器（“窗口” > “属性” > “属性”）的 fps 文本框中键入 **24**。
3. 创建或向 FLA 文件导入一个复杂的矢量图形。

   在以下目录中此示例的完成源文件中可以找到一个复杂的矢量图形：

   - 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\CacheBitmap。
   - 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/CacheBitmap。
4. 选择矢量图形，然后选择 “修改” > “转换为元件”。
5. 在 “名称”文本框中键入 **star**，然后单击 “高级”（如果对话框尚未展开）。
6. 选择 “为 ActionScript 导出”（同时选择 “在第一帧导出”）。
7. 在 “标识符”文本框中键入 **star_id**。
8. 单击 “确定”创建影片剪辑元件和 Star 的链接标识符。
9. 在时间轴中选择第 1 帧，然后将下面的 ActionScript 添加到 “动作”面板中：

```
import mx.transitions.Tween;

var star_array:Array = new Array();
for (var i:Number = 0; i < 20; i++) {
  makeStar();
}
function makeStar():Void {
  var depth:Number = this.getNextHighestDepth();
  var star_mc:MovieClip = this.attachMovie("star_id", "star" + depth,
  depth);
  star_mc.onEnterFrame = function() {
    star_mc._rotation += 5;
  }
```

```
    star_mc._y = Math.round(Math.random() * Stage.height - star_mc._height
    / 2);
    var star_tween:Tween = new Tween(star_mc, "_x", null, 0, Stage.width,
    (Math.random() * 5) + 5, true);
    star_tween.onMotionFinished = function():Void  {
      star_tween.yoyo();
    };
    star_array.push(star_mc);
  }
  var mouseListener:Object = new Object();
  mouseListener.onMouseDown = function():Void {
    var star_mc:MovieClip;
    for (var i:Number = 0; i < star_array.length; i++) {
      star_mc = star_array[i];
      star_mc.cacheAsBitmap = !star_mc.cacheAsBitmap;
    }
  }
  Mouse.addListener(mouseListener);
```

10. 选择 “控制” > “测试影片” 来测试该文档。

11. 单击舞台上的任意处，启用位图缓存。

    您将注意到，动画从每秒显示 1 帧的动画变为实例来回穿过舞台的平滑动画。单击舞台后，`cacheAsBitmap` 的设置将在 `true` 和 `false` 之间进行切换。

如果将缓存切换为关和闭，如上个示例所演示的，将释放缓存的数据。也可以将此代码应用于 Button 实例。请参见 《ActionScript 2.0 语言参考》中的 `cacheAsBitmap` (`Button.cacheAsBitmap` 属性)。

有关滚动影片剪辑的示例，请参见 《ActionScript 2.0 语言参考》中的 `scrollRect` (`MovieClip.scrollRect` 属性)。有关 Alpha 通道遮罩 （要求将 `cacheAsBitmap` 属性设置为 `true`）的信息，请参见第 337 页的 “关于 Alpha 通道遮罩”。

您可以找到对应的范例源文件，该文件说明了如何对实例应用位图缓存。可以在硬盘上的 Samples 文件夹中找到名为 cacheBitmap.fla 的文件。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\CacheBitmap。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/CacheBitmap。

您也可以找到对应的范例源文件，该文件说明了如何对滚动文本应用位图缓存。可以在硬盘上的 Samples 文件夹中找到范例源文件 flashtype.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

设置影片剪辑的背景

可以为影片剪辑设置不透明背景。例如，如果背景包括复杂矢量图片，则可以将 opaqueBackground 属性设置为特定的颜色（通常与舞台颜色相同）。然后可将背景视作位图，这样有助于优化性能。

当您将 cacheAsBitmap 设置为 true 并将 opaqueBackground 属性设置为特定颜色时，opaqueBackground 属性可以使内部位图不透明并提高呈现速度。如果不将 cacheAsBitmap 设置为 true，opaqueBackground 属性将向影片剪辑实例添加一个矢量方形。不会自动创建位图。

下面的示例演示了如何设置影片剪辑的背景来优化性能。

设置影片剪辑的背景：

1. 创建一个名为 **background.fla** 的新 Flash 文档。
2. 在舞台上绘制一个蓝色的圆形。
3. 选择所绘蓝色圆形，然后选择“修改”>“转换为元件”。
4. 选择“影片剪辑”选项，然后单击“确定”。
5. 选择舞台上的实例，然后在属性检查器的“实例”名称文本框中键入 **my_mc**。
6. 在时间轴中选择第 1 帧，然后在“动作”面板中键入下面的代码：

```
/* When you set cacheAsBitmap, the internal bitmap is opaque and renders
  faster. */
my_mc.cacheAsBitmap = true;
my_mc.opaqueBackground = 0xFF0000;
```

7. 选择“控制”>“测试影片”对该文档进行测试。

   影片剪辑将显示在舞台上，背景为您指定的颜色。

有关此属性的更多信息，请参见《ActionScript 2.0 语言参考》中的 opaqueBackground (MovieClip.opaqueBackground 属性)。

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 ActionScript 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。

若要创建遮罩，请执行以下操作：

1. 使用 “矩形”工具在舞台上创建一个正方形。
2. 选择该正方形，并按 F8 将其转换为影片剪辑。

   此实例即为您的遮罩。
3. 在属性检查器的 “实例名称”文本框中，键入 **mask_mc**。

   在充当遮罩的影片剪辑的所有不透明 （非透明）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”面板尚未打开，请打开该面板 （“窗口” > “动作”）。
6. 在 “动作”面板中输入以下代码：

```
System.security.allowDomain("http://www.helpexamples.com");

this.createEmptyMovieClip("img_mc", 10);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip):Void {
  target_mc.setMask(mask_mc);
}
var my_mcl:MovieClipLoader = new MovieClipLoader();
my_mcl.addListener(mclListener);
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

7. 选择 “控制” > “测试影片”对该文档进行测试。

   外部 JPEG 图像将在运行时加载到 SWF 文件中，并被您刚才在舞台上绘制的形状遮罩起来。

有关详细信息，请参见 《ActionScript 2.0 语言参考》中的 `setMask (MovieClip.setMask 方法)`。

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 (6.0.40.0) 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。这就是说，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，则出现在 SWF 文件中的遮罩将呈现为该遮罩矩形边框的形状，而不是该遮罩本身的形状。

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。

关于 Alpha 通道遮罩

如果遮罩影片剪辑和被遮罩的影片剪辑都使用位图缓存，则支持 Alpha 通道遮罩。对遮罩的支持还使您可以对遮罩使用应用于被遮罩影片剪辑本身之外的滤镜。

若要查看 Alpha 遮罩的示例，请从 www.macromedia.com/go/flash_samples_cn 下载 Alpha 遮罩范例文件。

在此范例文件中，遮罩是一个椭圆 (`oval_mask`)，透明度为 50% 并应用了模糊滤镜。被遮罩的剪辑 (`flower_maskee`) 的透明度为 100%，且没有应用滤镜。两种影片剪辑都在属性检查器中应用了运行时位图缓存。

在 “动作”面板中，以下代码置于时间轴的第 1 帧上：

```
flower_maskee.setMask(oval_mask);
```

测试文档 （“控制” > “测试影片”）时，通过使用遮罩混合被遮罩剪辑的透明度。

注释 遮罩层不支持 Alpha 通道遮罩。必须使用 ActionScript 代码来应用遮罩，而且还需使用运行时位图缓存。

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。ActionScript 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。有关处理影片剪辑事件的更多信息，请参见第 9 章 “处理事件”。

将类分配给影片剪辑元件

使用 ActionScript 2.0，您可以创建一个类来扩展内置 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 ActionScript 2.0 的更多信息，请参见第 194 页的 “示例：编写自定义类”。）

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。

若要创建影片剪辑子类，请执行以下操作：

1. 创建一个名为 **BallTest** 的新目录。
2. 选择 “文件” > “新建”，然后从文档类型列表中选择 ActionScript 文件以创建新的 ActionScript 文件。
3. 在脚本文件中输入以下代码：

```
// MoveRight 类    单击时将剪辑向右移动 20 个像素
class MoveRight extends MovieClip {
  public 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”。
4. 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_mc**。
5. 如果尚未显示链接的选项，选择 “高级”显示这些选项。
6. 选择 “为 ActionScript 导出”选项，然后在 “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. 对 Flash 文档进行测试（“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。

如果为一个类创建组件属性，并希望影片剪辑继承这些组件属性，您还需要执行一个步骤：在“库”面板中选择影片剪辑元件，从“库”弹出菜单中选择“组件定义”，然后在“类”框中输入新的类名称。

初始化类属性

在“将类分配给影片剪辑元件”中的第二个步骤内显示的示例中，您已经在创作时向舞台添加了 Ball 元件实例。正如在第 325 页的“将参数添加到动态创建的影片剪辑中”中所讨论的，您可以使用 `attachMovie()` 和 `duplicateMovie()` 的 *`initObject`* 参数为在运行时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 MoveRight 类的变体（请参见第 338 页的“将类分配给影片剪辑元件”）。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。

将参量传递给自定义类：

1. 创建一个新的 ActionScript 文档，并将它保存为 **MoveRightDistance.as**。
2. 在“脚本”窗口中键入下面的 ActionScript：

```
// 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
  // 按下鼠标时将剪辑移动多少像素。
  var distance:Number;
  function onPress() {
    this._x += this.distance;
  }
}
```

3. 保存进度。
4. 创建一个新的 Flash 文档，并使用 **MoveRightDistance.fla** 这个名称将它保存到类文件所在的目录中。
5. 创建一个包含矢量形状（例如椭圆）的影片剪辑元件，然后从舞台删除所有内容。

   您只需要此示例的库中的一个影片剪辑元件。
6. 在“库”面板中，右击 (Windows) 或按住 Control 单击 (Macintosh) 该元件并从上下文菜单中选择“链接”。

7. 将链接标识符 **Ball** 分配给元件。
8. 在 “AS 2.0 类”文本框中键入 **MoveRightDistance**。
9. 将下面的代码添加到时间轴中的第 1 帧：

```
this.attachMovie("Ball", "ball50_mc", 10, {distance:50});
this.attachMovie("Ball", "ball125_mc", 20, {distance:125});
```

   此代码在 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 `ball50_mc`）每次单击时移动 50 个像素；第二个实例（名为 `ball125_mc`）每次单击时移动 125 个像素。
10. 选择 “控制” > “测试影片”对 SWF 文件进行测试。

第 12 章

使用文本和字符串

12

使用 Macromedia Flash Professional 8 或 Macromedia Flash Basic 8 创建的许多应用程序、演示文稿或图形都会包括某种文本。可以使用多种不同的文本。可以在布局中使用静态文本，而将动态文本用于篇幅较长的文本。或者，可以使用输入文本来捕获用户输入，或在图像中使用文本作为背景设计。可以使用 Flash 创作工具或使用 ActionScript 创建文本字段。

显示文本的一种方法是在字符串在舞台上加载并显示之前使用代码操作字符串的显示方式。在应用程序中可以通过多种方法使用字符串，例如，将字符串发送给服务器并检索响应，分析数组中的字符串，或者验证用户键入到文本字段中的字符串。

本章将介绍在应用程序中使用文本和字符串的一些方法，重点介绍使用代码来操作文本。

下面的列表就本章中所用术语加以说明。

锯齿 与消除锯齿文本 （请参见下面的定义）不同，带有锯齿的文本不使用颜色变化使其锯齿边缘更为平滑。

消除锯齿 使用消除锯齿可使文本变得平滑，使显示在屏幕上的字符的边缘表现出较小的锯齿状。 Flash 中的 “消除锯齿”选项可以沿像素边界对齐文本轮廓，使文本更清楚，并且对于清晰呈现较小字体尤为有效。

字符 字符是指组成字符串的字母、数字和标点符号。

设备字体 设备字体是 Flash 中未嵌入到 SWF 文件中的特殊字体。相反， Flash Player 会使用本地计算机上与设备字体最相近的任何字体。由于未嵌入字体轮廓，因此与使用嵌入字型相比 SWF 文件大小更小。然而，由于未嵌入设备字体，因此使用这些字体创建的文本在未安装与设备字体相对应的字体的计算机系统上不能显示出预期效果。 Flash 包含三种设备字体：_sans（类似于 Helvetica 和 Arial 字体）、_serif（类似于 Times Roman 字体）和 _typewriter （类似于 Courier 字体）。

字体 具有类似字型、样式和大小的字符集。

字符串 字符序列。

文本 可在文本字段中或用户界面组件中显示的一个或多个字符串系列。

文本字段　舞台上的一种可见元素，用于向用户显示文本。与 HTML 中的输入文本字段或文本区域表单控件类似，Flash 允许将文本字段设置为可编辑（只读），允许 HTML 格式设置，启用多行支持，密码屏蔽或将 CSS 样式表应用于 HTML 格式的文本。

文本格式设置　可对文本字段或其中某些字符的格式进行设置。可应用于文本的一些文本格式设置选项的示例有：对齐、缩进、粗体、颜色、字号、边距宽度，斜体和字母间距。

有关文本的更多信息，请参见以下主题：

关于文本字段

动态或输入文本字段是一个 TextField 对象（TextField 类的实例）。当您在创作环境中创建文本字段时，可以在属性检查器中给它指定一个实例名称。可以在 ActionScript 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格式。

可以使用用户界面创建几种文本字段，也可以使用 ActionScript 创建文本字段。在 Flash 中可以创建以下几种文本字段：

静态文本　使用静态文本可显示无需更改的字符、少量的文本，或者显示在大多数计算机上不可用的特殊字体。还可以通过嵌入动态文本字段字符显示不常见的字体。

动态文本　需要显示在运行时更新或更改的字符时，可使用动态文本字段。而且，您还可以将文本加载到动态文本字段中。

输入文本　在需要捕获用户输入时可使用输入文本字段。用户可以将内容键入到这些文本字段中。

文本组件 可以使用 TextArea 或 TextInput 组件来显示或捕获应用程序中的文本。TextArea 组件类似于带有内置滚动条的动态文本字段。TextInput 组件类似于输入文本字段。这两个组件除了与文本字段等效的功能外还有其它功能；但是它们会增大应用程序的文件大小。

注释 所有的文本字段都支持 Unicode。有关 Unicode 的信息，请参见第 404 页的 “关于字符串和 String 类”。

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 344 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 648 页的 “关于显示文本字段属性以进行调试”。

ActionScript 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 375 页的 “使用 TextFormat 类”）。Flash Player 还支持部分 HTML 标签，您可以使用这些 HTML 标签设置文本格式（请参见第 390 页的 “使用 HTML 格式的文本”）。Flash Player 7 和更高版本支持 `img` 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 394 页的 “图像标签”）。

在 Flash Player 7 和更高版本中，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 样式设置内置 HTML 标签的样式，定义新的格式标签或应用样式。有关使用 CSS 的更多信息，请参见第 377 页的 “使用层叠样式表设置文本格式”。

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）。在 Flash Player 8 中，还可以动态加载 PNG、GIF 和渐进式 JPEG 图像（Flash Player 7 不支持渐进式 JPEG 图像）。文本在嵌入的媒体旁自动换行，这非常类似于 HTML 页中的文本在嵌入的媒体旁自动换行。有关更多信息，请参见第 394 页的 “图像标签”。

有关用来比较文本、字符串及更多内容的术语的信息，请参见本章的介绍部分第 341 页的 “使用文本和字符串”。

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输入（可编辑）文本字段。使用此类的方法和属性可以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。您可以动态地获取和设置属性并调用文本字段的方法。

若要使用 ActionScript 控制动态或输入文本字段，必须在属性检查器中为该文本字段指定一个实例名称。然后，您可以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 347 页的“在运行时创建文本字段”。

有关使用 TextField 类的更多信息，请参见以下主题：

- 第 344 页的“在运行时将文本分配到文本字段”
- 第 346 页的“关于文本字段实例和变量名称”

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

在运行时将文本分配到文本字段

使用 Flash 构建应用程序时，可能需要从外部源加载文本，如文本文件、XML 文件或远程 Web 服务。Flash 可提供多种控制在舞台上创建和显示文本的方式，如支持 HTML 格式的文本、纯文本、XML 格式文本和外部样式表。还可以使用 ActionScript 定义样式表。

若要将文本分配到文本字段，可以使用 TextField.text 或 TextField.htmlText 属性。如果您在属性检查器的变量文本字段中输入了一个值，则可以通过创建具有指定名称的变量向该文本字段分配值。如果您在 Flash 文档中使用 Macromedia Components Architecture 第二版，则还可以通过创建组件绑定来分配值。

下面的练习在运行时将文本分配到文本字段。

若要在运行时将文本分配到文本字段，请执行以下操作：

1. 使用文本工具在舞台上创建一个文本字段。
2. 选中文本字段后，在属性检查器（“窗口”>“属性”>“属性”）中，从“文本类型”弹出菜单中选择“输入文本”，并在“实例名称”文本框中输入 **headline_txt**。

   实例名称只能由字母、数字、下划线 (_) 和美元符号 ($) 组成。
3. 在时间轴中选择第 1 帧，然后打开“动作”面板（“窗口”>“动作”）。
4. 在“动作”面板中键入以下代码：

```
headline_txt.text = "New articles available on Developer Center";
```

5. 选择“控制”>“测试影片”，对该 Flash 文档进行测试。

还可以使用 ActionScript 创建文本字段，然后将文本分配到该文本字段。在时间轴的第 1 帧上键入下面的 ActionScript：

```
this.createTextField("headline_txt", this.getNextHighestDepth(), 100, 100,
  300, 20);
headline_txt.text = "New articles available on Developer Center";
```

此代码创建一个实例名称为 `headline_txt` 的新文本字段。该文本字段是在下一个最大的深度处创建，即 x 和 y 坐标分别为 100、100，文本字段宽度为 200 像素、高度为 20 像素。测试 SWF 文件（“控制”>“测试影片”）时，舞台上将出现文本“New articles available on Developer Center”。

创建 HTML 格式的文本字段：

使用以下两个步骤之一启用文本字段的 HTML 格式：

- 选择一个文本字段，并在属性检查器中单击“用 HTML 来格式化文本”。
- 使用 ActionScript 将文本字段的 `html` 属性设置为 `true`（请参见下面的代码范例）。

若要使用 ActionScript 将 HTML 格式应用于文本字段，请在时间轴的第 1 帧上键入以下 ActionScript：

```
this.createTextField("headline_txt", this.getNextHighestDepth(), 100, 100,
  300, 20);
headline_txt.html = true;
headline_txt.htmlText = "New articles available on <i>Developer Center</
  i>.";
```

上面的代码动态创建一个新文本字段，启用 HTML 格式并在舞台上显示“New articles available on Developer Center”，其中“Developer Center”以斜体显示。

在舞台上将 HTML 格式的文本用于文本字段时，必须为该文本字段的 `htmlText` 属性（而不是 text 属性）分配文本。

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

关于文本字段实例和变量名称

在属性检查器的“实例名称”文本框中，必须为文本字段指定实例名称，以在该文本字段上调用方法并获取和设置属性。

在属性检查器的“变量”文本框中，您可以为某个动态或输入文本字段分配一个变量名称。然后您可以为该变量分配一个值。此功能现在已否决，但在为较早版本的 Flash Player（如 Flash Player 4）创建应用程序时，您可能要使用此功能。在面向较新的播放器时，应该使用文本字段的实例名称和 ActionScript 来将该文本字段中的文本设定为目标。

但是，不要将文本字段的实例名称与其变量名称混淆。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；不是对对象的引用。

例如，如果为某个文本字段分配了变量名称 `myTextVar`，您就可以使用下面的代码设置该文本字段的内容：

```
var myTextVar:String = "This is what will appear in the text field";
```

但是，您不能使用变量名称 `myTextVar` 设置文本字段的 `text` 属性。您必须使用实例名称，如下面的代码所示：

```
// 这样不起作用。
myTextVar.text = "A text field variable is not an object reference";

// 对实例名称为“myField”的输入文本字段，这将有效。
myField.text = "This sets the text property of the myField object";
```

如果面向的 Flash Player 版本支持 TextField 类，则可使用 `TextField.text` 属性控制文本字段的内容。这将降低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行为。

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

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。

若要使用 ActionScript 动态创建文本字段，请执行以下操作：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”创建一个新的 FLA 文件。
2. 在时间轴的第 1 帧上键入下面的 ActionScript：

   ```
   this.createTextField("test_txt", 10, 0, 0, 300, 100);
   ```

   此代码在 (0, 0) 点深度 （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。
3. 若要访问新创建的文本字段的方法和属性，请使用在 `createTextField()` 方法第 1 个参数中指定的实例名称。

   例如，下面的代码创建一个名为 `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，该文本字段可以根据插入文本的长度进行调整。然后，该代码使用文本字段的 `text` 属性分配文本：

   ```
   test_txt.multiline = true;
   test_txt.wordWrap = true;
   test_txt.autoSize = "left";
   test_txt.text = "Create new text fields with the
     MovieClip.createTextField() method.";
   ```
4. 选择 “控制” > “测试影片”，查看该文本字段。

   在运行时创建的文本将出现在舞台上。

可以使用 `TextField.removeTextField()` 方法删除使用 `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《ActionScript 2.0 语言参考》中的 createTextField （MovieClip.createTextField 方法）和 removeTextField （TextField.removeTextField 方法）。

注意

某些 TextField 属性 （如 `_rotation`）在运行时创建文本字段时不可用。只有文本字段使用嵌入字体时才能对其进行旋转。请参见第 358 页的 “嵌入字体元件：”。

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

关于操作文本字段

可以用多种方式操作在 FLA 文件中创建的文本字段。只要您在属性检查器中为文本字段指定了实例名称，或者可以使用代码指定一个实例名称 （如果您使用代码创建字段），就可以操作文本字段了。下面的简单示例创建了一个文本字段，将文本赋予该文本字段，并更改了该字段的 border 属性：

```
this.createTextField("pigeon_txt", this.getNextHighestDepth(), 100, 100,
  200, 20);
pigeon_txt.text = "I like seeds";
pigeon_txt.border = true;
```

有关 TextField 类中属性的完整列表，请参见 《ActionScript 2.0 语言参考》。

有关如何处理文本字段的示例，请参见以下各部分：

- 第 348 页的 “更改文本字段的位置”
- 第 349 页的 “在运行时更改文本字段的尺寸”

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

更改文本字段的位置

可以在运行时在舞台上更改文本字段的位置。您需要为该文本字段的 _x 和 _y 属性设置新值，如下例所示。

使用 ActionScript 重新定位文本字段：

1. 创建一个新的 FLA 文件，并将其另存为 **positionText.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 0, 0, 300, 200);
my_txt.border = true;
my_txt.text = "Hello world";
my_txt._x = (Stage.width - my_txt._width) / 2;
my_txt._y = (Stage.height - my_txt._height) / 2;
```

3. 保存该 Flash 文档，然后选择 “控制” > “测试影片” 以查看该文本字段是否位于舞台的中心。

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

在运行时更改文本字段的尺寸

您可能需要在运行时而不是在创作环境中动态获取或设置文本字段的尺寸。下一个示例在时间轴上创建一个文本字段，并将其初始尺寸设置为 100 像素宽乘以 21 像素高。然后，将该文本字段调整为 300 像素宽乘以 200 像素高，并将它重新定位到舞台中央。

使用 ActionScript 调整文本字段的大小：

1. 创建一个新的 Flash 文档，并将其另存为 **resizeText.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 0, 0, 100, 21);
my_txt.border = true;
my_txt.multiline = true;
my_txt.text = "Hello world";
my_txt.wordWrap = true;
my_txt._width = 300;
my_txt._height = 200;
my_txt._x = (Stage.width - my_txt._width) / 2;
my_txt._y = (Stage.height - my_txt._height) / 2;
```

3. 保存该 Flash 文档，然后选择 “控制” > “测试影片” 以在创作环境中查看结果。

上面的示例在运行时将动态创建的文本字段调整为 300 像素乘以 200 像素，但是当您从外部网站加载内容且不确定将返回多少内容时，此技术可能不能满足您的要求。所幸，Flash 包含 `TextField.autoSize()` 方法，可以用它来自动调整文本字段以适合其内容。下面的示例演示了在将文本添加到文本字段之后，可以如何使用 `TextField.autoSize()` 属性来调整文本字段。

根据内容自动调整文本字段：

1. 创建一个新的 Flash 文档，并将其另存为 **resizeTextAuto.fla**。
2. 将下面的代码添加到主时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 10, 10, 160, 120);
my_txt.autoSize = "left";
my_txt.border = true;
my_txt.multiline = true;
my_txt.text = "Lorem ipsum dolor sit amet, consectetur adipisicing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
  enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
  aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
  in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
  Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
  officia deserunt mollit anim id est laborum.";
my_txt.wordWrap = true;
```

> **注意** 将此代码直接从某些版本的 “Flash 帮助”粘贴到 “动作”面板时，长文本字符串中可能包含换行符。此时不会对代码进行编译。如果遇到这种情况，请在 “动作”面板的弹出菜单上启用 “隐藏字符”，然后删除长文本字符串中的换行符。

3. 保存该 Flash 文档，然后选择“控制”>“测试影片”以在创作环境中查看该 Flash 文档。

   Flash 将沿垂直方向调整文本字段，以便可以显示所有内容，而不会被文本字段边界裁切。如果将 `my_txt.wordWrap` 属性设置为 `false`，则文本字段将沿水平方向调整以容纳文本。

   若要对自动调整大小的文本字段强制规定一个最大高度（以便文本字段的高度不会超出舞台边界），请使用以下代码。

```
if (my_txt._height > 160) {
  my_txt.autoSize = "none";
  my_txt._height = 160;
}
```

必须添加一些滚动功能（如滚动条）以便允许用户查看其余的文本内容。此外，还可以在文本上滚动鼠标指针；在测试此代码时完全可以采用此方法。

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 textfieldsA.fla 和 textfieldsB.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\TextFields。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/TextFields。

关于将文本和变量加载到文本字段

将文本加载到 Flash 文档的方法有多种，其中包括（但不限于）使用 FlashVars、LoadVars、XML 或 Web 服务。也许将文本传递到 Flash 文档中的最简单方法就是使用 FlashVars 属性，此方法通过 HTML 代码中用于将 SWF 文件嵌入到 HTML 页面中的 `object` 和 `embed` 标签将简短的字符串文本传递到 Flash 文档中。将文本或变量加载到 Flash 文档的另一个简单方法就是使用 LoadVars 类，该类可从文本文件中加载较大的文本块或加载一系列 URL 编码的变量。

正如您在本节中上面的示例中所看到的，将文本加载到 SWF 文件的某些方法比其它方法更加简便。但是，当您从外部站点上收集数据时，可能无法选择要加载的数据的格式。

每种向 SWF 文件加载数据和 / 或向 （从） SWF 文件发送数据的方法都各有利弊。XML、Web 服务和 Flash Remoting 是加载外部数据的功能最全面的方法，但是它们也是最难掌握的。有关 Flash Remoting 的信息，请参见 www.macromedia.com/support/flashremoting。

如第 352 页的 “使用 FlashVars 加载并显示文本”和第 353 页的 “使用 LoadVars 加载并显示文本”中所示， FlashVars 和 LoadVars 比较简单，但在可以加载的数据的类型和格式方面有很多限制。此外，在发送和加载数据时还必须要遵守安全性限制。有关安全性的信息，请参见第 17 章 “了解安全性”。有关加载外部数据的更多信息，请参见第 16 章 “使用外部数据”。

以下各部分将介绍向文档中加载文本和变量的几种不同的方法：

- 第 352 页的 “使用 FlashVars 加载并显示文本”
- 第 353 页的 “使用 LoadVars 加载并显示文本”
- 第 354 页的 “使用 LoadVars 加载变量”
- 第 355 页的 “从 XML 文档中加载并显示文本”

您可以找到演示如何通过 ActionScript 使用文本字段的范例源文件。该源文件名为 loadText.fla 和 formattedText.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\LoadText。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/LoadText。

除了位图缓存之外，您还可以找到加载文本并应用消除锯齿格式的源文件。该源文件名为 flashtype.fla，位于硬盘上的 Samples 文件夹中：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

使用 FlashVars 加载并显示文本

使用 FlashVars 非常简单，但是需要将 SWF 文件随 HTML 文档一起发布。您修改了生成的 HTML 代码并在 `object` 和 `embed` 标签中包括了 FlashVars 属性。然后可以通过在 Web 浏览器中查看修改的 HTML 文档对 Flash 文档进行测试。

若要使用 FlashVars 将变量从 HTML 中传递到 Flash 文档，请执行以下操作：

1. 创建一个新的 Flash 文档，并将其另存为 **flashvars.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 10, 10, 100, 21);
my_txt.text = _level0.username;
```

3. 保存该 Flash 文档，然后选择 “文件” > “发布”以生成 HTML 和 SWF 文件。

> **提示** 默认情况下，HTML 文档将发布到 FLA 文件所在的同一目录中。如果未发布 HTML 文档，请选择 “文件” > “发布设置”，然后选择 “格式”选项卡。请确保选择了 HTML。

4. 在文本或 HTML 编辑器中打开 flashvars.html 文档。
5. 在 HTML 文档中，修改 `object` 标签中的代码以与下面的代码匹配。

   需要添加的代码以**粗体**显示。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
  swflash.cab#version=8,0,0,0" width="550" height="400" id="flashvars"
  align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="flashvars.swf" />
<param name="FlashVars" value="username=Thomas" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<embed src="flashvars.swf" FlashVars="username=Thomas" quality="high"
  bgcolor="#ffffff" width="550" height="400" name="flashvars"
  align="middle" allowScriptAccess="sameDomain" type="application/x-
  shockwave-flash" pluginspage="http://www.macromedia.com/go/
  getflashplayer" />
</object>
```

6. 保存对该 HTML 文档所做的更改。
7. 在 web 浏览器中打开修改过的 HTML。

   SWF 文件将在舞台上动态创建的文本字段中显示名称 “Thomas”。

有关安全性的信息，请参见第 17 章 “了解安全性”

使用 LoadVars 加载并显示文本

您还可以使用 LoadVars 类将内容加载到 SWF 文件中，此方法将加载来自同一服务器上的外部文件中的文本或变量，甚至来自不同服务器上的内容。下一个示例演示了如何动态创建一个文本字段并用远程文本文件的内容填充该字段。

使用 LoadVars 以外部文本填充文本字段：

1. 创建一个新的 Flash 文档，并将其另存为 **loadvarsText.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 10, 10, 320, 100);
my_txt.autoSize = "left";
my_txt.border = true;
my_txt.multiline = true;
my_txt.wordWrap = true;

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function (src:String):Void {
  if (src != undefined) {
    my_txt.text = src;
  } else {
    my_txt.text = "Unable to load external file.";
  }
}
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

   上面的代码片段中的第一个代码块将在舞台上创建一个新的文本字段，并启用多行和自动换行。第二个代码块定义一个新的 LoadVars 对象，该对象用于从远程 Web 服务器加载文本文件 (lorem.txt) 并将其内容显示在前面创建的 my_txt 文本字段中。

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该 SWF 文件进行测试。

   在短暂的延迟之后， Flash 会将远程文件的内容显示在舞台上的文本字段中。

有关安全性的信息，请参见第 17 章 “了解安全性”

使用 LoadVars 加载变量

LoadVars 类还允许您加载 URL 编码格式的变量，这类似于在 Web 浏览器中使用查询字符串传递变量。下面的示例演示了如何将远程文本文件加载到 SWF 文件中并显示其变量 monthNames 和 dayNames。

使用 LoadVars 从文本文件加载变量：

1. 创建一个新的 Flash 文档，并将其另存为 **loadvarsVariables.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 10, 10, 320, 100);
my_txt.autoSize = "left";
my_txt.border = true;
my_txt.multiline = true;
my_txt.wordWrap = true;

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onLoad = function (success:Boolean):Void {
  if (success) {
    my_txt.text = "dayNames: " + lorem_lv.dayNames + "\n\n";
    my_txt.text += "monthNames: " + lorem_lv.monthNames;
  } else {
    my_txt.text = "Unable to load external file.";
  }
}
/* contents of params.txt:
  &monthNames=January,February,...&dayNames=Sunday,Monday,...
*/
lorem_lv.load("http://www.helpexamples.com/flash/params.txt");
```

3. 保存 Flash 文档，然后从主菜单中选择“控制”>“测试影片”。

   因为您所使用的是 LoadVars.onLoad() 方法（而不是 LoadVars.onData()），所以 Flash 将分析出变量并在 LoadVars 对象实例内创建变量。外部文本文件包含两个变量，monthNames 和 dayNames，它们都包含一些字符串。

有关安全性的信息，请参见第 17 章“了解安全性”。

从 XML 文档中加载并显示文本

XML 数据是在 Internet 上分发内容的一种常用方法，很大程度上是因为它是用于组织和分析数据的一种被普遍接受的标准。因此，XML 是与 Flash 之间收发数据的绝佳选择；然而，与使用 LoadVars 和 FlashVars 加载数据并显示文本相比，XML 更不易掌握。

从外部 XML 文档中将文本加载到 Flash：

1. 创建一个新的 Flash 文档，并将其另存为 **xmlReviews.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
this.createTextField("my_txt", 10, 10, 10, 320, 100);
my_txt.autoSize = "left";
my_txt.border = true;
my_txt.multiline = true;
my_txt.wordWrap = true;

var reviews_xml:XML = new XML();
reviews_xml.ignoreWhite = true;
reviews_xml.onLoad = function (success:Boolean):Void {
  if (success) {
    var childItems:Array = reviews_xml.firstChild.childNodes;
    for (var i:Number = 0; i < childItems.length; i++) {
      my_txt.text += childItems[i].firstChild.firstChild.nodeValue +
"\n";
    }
  } else {
    my_txt.text = "Unable to load external file.";
  }
}
reviews_xml.load("http://www.helpexamples.com/flash/xml/reviews.xml");
```

   上面的代码片段中的第一个代码块将在舞台上创建一个新的文本字段。此文本字段用于显示以后加载的 XML 文档的各个部分。第二个代码块负责创建将用于加载 XML 内容的 XML 对象。当日期完全加载并经过 Flash 分析之后，将立刻调用 `XML.onLoad()` 事件处理函数，并在文本字段中显示 XML 包的内容。

3. 保存 Flash 文档，然后选择“控制”>“测试影片”对该 SWF 文件进行测试。

   Flash 将在舞台上的文本字段中显示以下输出结果：

```
Item 1
Item 2
...
Item 8
```

有关安全性的信息，请参见第 17 章 “了解安全性”

使用字体

字体是具有类似字型、样式和大小的字符集。无论您使用 Flash Basic 8 或 Flash Professional 8 构建何种内容，在 Flash 应用程序中都可能会用到至少使用一种或两种字体的文本。如果要生成动画，而且不确定最终用户的系统上是否安装了特定字体，则需要了解字体嵌入的基础知识。

以下各部分将介绍如何嵌入字符、整个字体、共享字体以及在 Flash 8 中使用字体的其它技术。

有关字体的更多信息，请参见以下主题：

- 第 357 页的 “嵌入字符”
- 第 358 页的 “嵌入字体”
- 第 360 页的 “创建自定义字符集”
- 第 362 页的 “对嵌入字体使用 TextField 方法”
- 第 363 页的 “关于共享字体”

下面的示例演示如何在 Flash 文档中添加和删除嵌入字符和字符集。

添加和删除嵌入字符和字符集：

1. 创建一个新的 Flash 文档，并将其另存为 **embedding.fla**。
2. 使用文本工具在舞台上创建一个动态文本字段。
3. 单击 “嵌入”启动 “字符嵌入”对话框。
4. 使用鼠标指针单击选择要嵌入的特定字符集。

   若要选择多个字符集，可以在用鼠标指针选择项时使用 Shift 或 Control 键。若要选择连续排列的字符集，请通过鼠标指针选择一个字符集，按住 Shift，然后再单击另一个字符集。使用 Shift 可选择两个选定字符集之间的所有字符集。若要选择多个非连续排列的字符集，请在选择字符集的同时按住 Control 键。还可以通过在选择一个字符集后按住鼠标键并拖动鼠标经过多个字符集快速选择多个字符集。
5. 若要删除以前添加的特定字符集，请按住 Control 键并单击鼠标指针取消选择该字符集。
6. 若要删除所有选定字符集以及 “包含这些字符”文本输入字段中的任何指定字符，请单击 “不要嵌入”。

   “不要嵌入”可清除以前指定的任何单个字符或字符集。

在 “字符嵌入”对话框中单击 “不要嵌入”将直接删除以前选择的任何指定嵌入字符和字符集，而不再进行删除确认。

嵌入字符

如果正在使用嵌入字体并且明确知道所需的字符，您就可以仅嵌入所需字符而不是包含其它未用的字体轮廓，从而减少文件大小。若要在文本字段中嵌入某些字符而不是嵌入整个字符集，请使用 “字符嵌入”对话框指定要嵌入的特定字符。

嵌入用于文本字段的特定字符：

1. 创建一个新的 Flash 文档，并将其另存为 **charembed.fla**。
2. 使用文本工具在舞台上创建一个文本字段，并将文本字段的文本类型设置为动态或输入。
3. 在舞台上选中该文本字段，在属性检查器中单击 “嵌入”，打开 “字符嵌入”对话框。

   通过 “字符嵌入”对话框可设置要嵌入 Flash 文档的字符集（以及每个字符集的字型数），指定要嵌入的特定字符，并通知您嵌入此文本字段的字型总数。
4. 将字符串 **hello world** 键入 “包含这些字符”文本框。

   该对话框通知您嵌入此文本字段的字型总数为 8 个。尽管字符串 “hello world”包含 11 个字符，但是 Flash 只嵌入唯一字型，因此字母 l 和 o 都只嵌入一次，而不是多次。
5. 单击 “确定”应用更改并返回文档。
6. 使用文本工具在舞台上创建一个新的文本字段。
7. 在属性检查器中将文本字段的文本类型设置为动态。
8. 将字符串 **hello world** 键入舞台上的文本字段。
9. 在属性检查器中单击 “嵌入”，再次打开 “字符嵌入”对话框。
10. 单击 “自动填充”将自动填充 “包含这些字符”文本框。

    将看到字符串“helo wrd”。不必通知 Flash 要包含的字符，Flash 可以为您确定指定文本字段中的所有唯一字符。

    提示 只有文本字段包含舞台上的文本时，Flash 才可以自动确定嵌入的字符。如果使用 ActionScript 填充文本字段，则必须为该文本字段指定要嵌入的字符。
11. 单击 “确定”。

嵌入字体

在嵌入字体时，Flash 会将所有字体信息存储在 SWF 文件中，因此，即使用户计算机上没有安装的字体也可以正确显示。如果您在 FLA 文件中所用的某种字体在用户的系统上没有安装，而且您没有把该字体嵌入到 SWF 文件中，则 Flash Player 将自动选择一个替换字体进行显示。

注释	只有在使用动态或输入文本字段时才需要嵌入字体。如果使用静态文本字段，则不必嵌入字体。

嵌入字体元件：

1. 选择 “窗口” > “库” 来打开当前的 FLA 文件的库。

   打开要向其中添加字体元件的库。
2. 从库的弹出菜单（位于 “库” 面板右上角）中选择 “新建字型”。
3. 在 “字体元件属性” 对话框的 “名称” 文本框中键入字体元件的名称。
4. 从 “字体” 菜单中选择一种字体，或者将字体名称键入到 “字体” 文本框中。
5. 如果您要对字体应用样式，请选择 “粗体”、“斜体” 或 “锯齿文本”。
6. 输入要嵌入的字体大小，然后单击 “确定” 应用更改并返回文档。

   现在，字体出现在当前的文档库中。

将某种字体嵌入到库中之后，就可以将它用于舞台上的文本字段了。

在 Flash 文档中使用嵌入字体元件：

1. 按照第 358 页的 “嵌入字体”过程中的步骤将字体嵌入到库中。
2. 使用 “文本” 工具在舞台上创建一个文本字段。
3. 在文本字段中键入某个文本。
4. 选择该文本字段，打开属性检查器。
   a. 将该文本字段设置为单行。
   b. 使用 “字体” 下拉菜单选择嵌入字体的名称。

   嵌入字体的字体名称后面有一个星号 (*)。

5. 在属性检查器中单击 “嵌入”启动 “字符嵌入”对话框。

“字符嵌入”对话框允许您选择要想为选定的文本字段嵌入的单个字符或字符集。若要指定将要嵌入的字符，请在该对话框中将这些字符键入文本框，或单击 “自动填充”使用文本字段中的当前唯一字符自动填充该文本字段。如果不确定需要哪些字符 （例如，由于文本是从外部文件或 Web 服务中加载），则可以选择要嵌入的整个字符集，如大写字母 [A..Z]、小写字母 [a..z]、数字 [0..9]、标点符号 [!@#%...] 以及用于几种不同语言的字符集。

注意	由于 Flash 需要为使用的每个字符集存储所有字体信息，因此您选择的每个字符集都会增加 SWF 文件的最终大小。

6. 选择要嵌入的单个字符或字符集，然后单击 “确定”应用更改并返回到您的文档。
7. 选择 “控制” > “测试影片”以在创建环境中测试 Flash 文档。

嵌入字体将在舞台上的文本字段中显示。为了正确测试字体是否已经嵌入，可能需要在没有安装嵌入字体的另一台计算机上进行测试。

或者，还可以为使用嵌入字体的文本字段设置 `TextField._alpha` 或 `TextField._rotation` 属性，因为这些属性仅对嵌入字体有效 （请参见以下步骤）。

8. 关闭 SWF 文件，返回到创建工具。
9. 在舞台上选择文本字段，打开属性检查器。
   a. 将文本字段的文本类型设置为 “动态文本”。
   b. 在 “实例名称”文本框中键入 **font_txt**。
10. 将下面的代码添加到时间轴中的第 1 帧：

```
font_txt._rotation = 45;
```

11. 再次选择 “控制” > “测试影片”以便在创建环境中查看所做更改。

嵌入字体将顺时针旋转 45°，仍可以看到文本，因为它已嵌入到了 SWF 文件中。

小心	如果没有将某种字体嵌入到 Flash 文档中，并且 Flash Player 自动在用户计算机上选择一种替换字体，则 TextField.font 属性将返回 FLA 内使用的原始字体，而不是替换字体的名称。

注意	如果在文本字段中使用具有多种样式的嵌入字体，则必须嵌入要使用的样式。例如，如果正在使用一种名为 Times 的嵌入字体，然后又希望文字显示为斜体，则必须确保同时嵌入正常和斜体字符外形。否则，该文本将不会在文本字段中显示。

创建自定义字符集

除了使用 Flash 默认字符集之外，还可以创建自己的字符集并将其添加到“字符嵌入”对话框。例如，您可能需要某些字段包含 Extended Latin，支持各种强调符。但是您可能不需要数字和标点符号，或者您可能只需要大写字符。您可以创建仅包含所需字符的自定义字符集，而不是嵌入整个字符集。这种方法不会存储不需要的字符的额外字体信息，因此可以使 SWF 文件保持最小。

若要创建自定义字符集，您必须编辑 UnicodeTable.xml 文件，该文件位于 C:\Program Files\Macromedia\Flash 8\<language>\First Run\FontEmbedding\ 目录。该文件定义默认字符集及其包含的字符范围和字符。

在创建自定义字符集之前，应理解必需的 XML 结构。以下 XML 节点定义大写 [A..Z] 字符集：

```
<glyphRange name="Uppercase [A..Z] " id="1" >
  <range min="0x0020" max ="0x0020" />
  <range min="0x0041" max ="0x005A" />
</glyphRange>
```

请注意，`glyphRange` 节点包含 `name`、`Uppercase [A..Z]` 和 `id`。`glyphRange` 节点可具有任意数量的 range 子节点。 range 可以是前面代码片段中所示的单个字符，如 `0x0020`（空格字符），也可以是一些字符范围，如第二个 range 子节点。若要仅嵌入单个字符，请将 `min` 值和 `max` 值设置为相同的 Unicode 字符值。

XML `glyphRange` 节点的另一个示例是 `Numerals [0..9]` 节点：

```
<glyphRange name="Numerals [0..9] " id="3" >
  <range min="0x0030" max ="0x0039" />
  <range min="0x002E" max ="0x002E" />
</glyphRange>
```

此字符范围包含 Unicode 值 0x0030 (0) 到 0x0039 (9)，以及 0x002E (.)。

在创建自定义字符集之前，需要了解字符及其相应的 Unicode 值。查找 Unicode 值的最佳位置是 Unicode Standards 网站 www.unicode.org，该网站包含面向数十种语言的 Unicode 字符代码图表。

小心 若要添加自定义字符集，需要编辑 Flash 安装文件夹中的 XML 文件。在编辑此文件之前，应制作备份副本，以备还原为原始 Unicode 表时使用。

小心 Macromedia 建议不要修改随 Flash 安装的现有字符集，而应创建包含所需字符和标点符号的自定义字符集。

创建和使用自定义字符集：

1. 使用 XML 或文本编辑器（如 Notepad 或 TextEdit）打开 UnicodeTable.xml 文档，该文档位于 <Flash install directory>\<language>\First Run\FontEmbedding\ 目录。

   注意 请不要忘记保存此文档备份，以备还原为随 Flash 安装的原始文件时使用。

2. 滚动到 XML 文档底部，并在结束 `</fontEmbeddingTable>` 节点之前直接添加以下 XML 代码：

```
<glyphRange name="Uppercase and Numerals [A..Z,0..9] " id="100" >
  <range min="0x0020" max ="0x0020" />
  <range min="0x002E" max ="0x002E" />
  <range min="0x0030" max ="0x0039" />
  <range min="0x0041" max ="0x005A" />
</glyphRange>
```

3. 保存对 UnicodeTable.xml 所做更改。

   如果已打开 Flash，则必须重新启动该应用程序才能使用新的字符集。

4. 打开或重新启动 Flash，然后创建新的 Flash 文档。
5. 使用文本工具在舞台上添加一个新的 TextField 实例。
6. 在属性检查器中将 TextField 的文本类型设置为动态，然后单击 “嵌入字符选项”打开 “字符嵌入”对话框。
7. 滚动到 “字符嵌入”对话框底部，然后选择新的自定义字符集，大写和数字 [A..Z,0..9]（38 个字型）。
8. 选择任何其它字符集，然后单击 “确定”。

   如果选择自定义字符集，大写和数字 [A..Z,0..9]，以及默认大写 [A..Z] 或数字 [0..9] 字符集，请注意嵌入的字型总数不会改变。原因是自定义字符集中包含了所有大写字符，而 Flash 不包含重复字符，这样可使文件大小保持最小。如果选择标点符号字符集（包含 52 个字型）以及自定义字符集（包含 38 个字型），则 Flash 将只存储 88 个而不是 90 个字型的信息。这是因为自定义字符集中已包含了其中两个重复的字符（空格和句点）。

   提示 字符集在 “字符嵌入”对话框中的位置由其在 XML 文档中的位置决定。通过在 XML 文件中移动 <glyphRange> 包，可以重新排列字符集顺序，包括自定义字符集。

对嵌入字体使用 TextField 方法

TextField 类的方法为应用程序提供了有用的功能。例如，可以使用 ActionScript 控制文本字段的粗细，如下例所示。

若要使用 ActionScript 设置文本字段的粗细，请执行以下操作：

1. 创建一个新的 Flash 文档，并将其另存为 **textfieldThickness.fla**。
2. 打开 “库”面板，从弹出菜单 （位于 “库”面板的右上角）中选择 “新建字型”。

   随即打开 “字体元件属性”对话框。此对话框允许您选择要嵌入到 SWF 文件中的一种字体 （包括字体样式和字体大小） 。还可以指定在文档库和属性检查器中的字体下拉菜单中显示的字体名称 （如果您在舞台上选择了一个文本字段）。

   a. 从 “字体”下拉菜单中选择 “Times New Roman”字体。
   b. 确保取消选择了 “粗体”和 “斜体”选项。
   c. 将大小设置为 30 像素。
   d. 输入字体名称 **Times (embedded)**
   e. 单击 “确定”。
3. 在库中，右击字体元件，然后从上下文菜单中选择 “链接”。

   Flash 将打开 “链接属性”对话框。
4. 选择 “为 ActionScript 导出”和 “在第一帧导出”选项，并单击 “确定”。
5. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
// 1
this.createTextField("thickness_txt", 10, 0, 0, Stage.width, 22);
this.createTextField("lorem_txt", 20, 0, 20, Stage.width, 0);
lorem_txt.autoSize = "left";
lorem_txt.embedFonts = true;
lorem_txt.antiAliasType = "advanced";
lorem_txt.text = "Lorem ipsum dolor sit amet, consectetur adipisicing
  elit, sed do eiusmod tempor incididunt ut labore et dolore magna
  aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
  laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
  in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
  pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa
  qui officia deserunt mollit anim id est laborum.";
lorem_txt.wordWrap = true;

// 2
var style_fmt:TextFormat = new TextFormat();
style_fmt.font = "Times (embedded)";
style_fmt.size = 30;
lorem_txt.setTextFormat(style_fmt);

// 3
```

```
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  //TextField.thickness 值的范围可以是从 -200 到 +200。
  lorem_txt.thickness = Math.round(_xmouse * (400 / Stage.width) - 200);
  thickness_txt.text = "TextField.thickness = " + lorem_txt.thickness;
};
Mouse.addListener(mouseListener);
```

第一个代码块创建两个文本字段 thickness_txt 和 lorem_txt，并将它们放置在舞台上。lorem_txt 文本字段将其 embedFonts 属性设置为 true，并用文本块填充该文本字段。

第二个代码块使用 Times New Roman 字型定义一种文本格式，将字体大小设置为 30 像素，并将该文本格式应用于 lorem_txt 文本字段。

第三个（也就是最后一个）代码块定义一个鼠标侦听器并将它分配给 onMouseMove 事件。当鼠标指针在舞台上沿水平方向移动时，TextField.thickness 属性将根据 _xmouse 的当前值在 -200 到 +200 之间变化。

6. 保存对 FLA 文件所做的更改。
7. 选择“控制”>“测试影片”对 Flash 文档进行测试。

   当向舞台左侧移动鼠标指针时，字体粗细值将减少。当向舞台右侧移动鼠标指针时，字体粗细值将增加。

关于共享字体

若要将一种字体作为共享库项使用，可以在“库”面板中创建字体元件，然后给该字体元件分配以下属性：

- 一个标识符字符串
- 一个用于公布包含该字体元件的文档的 URL

由此，您可以链接到该字体并在 Flash 应用程序中使用它，而不用将字体存储在 FLA 文件中。

关于字体呈现和消除锯齿文本

Flash 中的字体呈现控制文本在 SWF 文件中的显示方式；即文本在运行时的呈现（或者绘制）方式。Flash Player 8 中使用的高级字体呈现技术称为 FlashType。FlashType 使用了高级呈现技术，有助于文本在以小字体到正常大小字体显示时均能清晰易读，如将高级消除锯齿应用于文本字段时。本部分的后面内容将对此技术进行更为详细的介绍。

消除锯齿使您能够对文本进行平滑处理，以便显示在屏幕上的字符的边缘表现出较小的锯齿状，在要使用较小的文本大小显示文本时此功能尤其有帮助。文本的“消除锯齿”选项通过沿像素边界对齐文本轮廓使字符更加清晰易读，此选项对于更清晰地呈现小字体尤其有效。可以对应用程序中的每个文本字段（而不是单个字符）应用消除锯齿。

如果用户具有 Flash Player 7 或更高版本，则消除锯齿可用于静态、动态和输入文本。如果用户具有 Flash Player 的早期版本，则此选项只能用于静态文本。Flash Player 8 中可以使用高级消除锯齿选项。

Flash Basic 8 和 Flash Professional 8 包含大为改进的字体光栅化处理和呈现技术（称为 FlashType），用于使用消除锯齿的字体。Flash 8 包含五种字体呈现方法，仅在发布面向 Flash Player 8 的 SWF 文件时全部适用。如果发布的文件要在 Flash Player 7 或早期版本中使用，则只能对文本字段可使用“动画消除锯齿”选项。

FlashType 是一种高质量的字体呈现技术，可以通过使用 Flash 8 创作工具或 ActionScript 启用该技术。通过 FlashType 技术可以呈现具有高质量输出的小字号字型，同时具有更多控制。您可以将 FlashType 应用于静态、动态和输出文本字段的嵌入字体呈现。改进的功能使得嵌入文本在显示时可以具有与设备文本相同的品质级别，并且在不同的平台上字体的显示效果相同。

Flash Player 8 提供的字体呈现方法有“设备字体”、“位图字体”（关闭消除锯齿）、“动画消除锯齿”、“可读性消除锯齿”和“自定义消除锯齿”，通过这些字体呈现方法可以定义粗细和清晰度的自定义值。有关这些选项的更多信息，请参见第 365 页的“Flash 中的字体呈现选项”。

注释	在 Flash 8 中打开现有的 FLA 文件时，文本不会自动更新到“可读性消除锯齿”选项；必须选择每个文本字段并手动更改消除锯齿设置才能利用 FlashType 呈现技术。

高级和自定义消除锯齿功能支持以下各项：

- 缩放和旋转文本
- 所有字体（正常、粗体或斜体）最大为 255 磅。
- 文件导出为最常见的格式（如 JPEG 或 GIF 文件）。

高级和自定义消除锯齿功能不支持以下各项：

- Flash Player 7 或更早版本
- 倾斜或翻转文本
- 打印
- 文件导出为 PNG 文件格式

注释 在动画播放文本时，播放器将关闭高级消除锯齿以改善文本在移动时的外观。动画结束后，消除锯齿会重新打开。

硬盘上的一个范例文件演示了如何在应用程序中应用并操作消除锯齿文本。使用 FlashType 呈现技术可创建清晰易读的小字号文本。此范例还演示了在使用 `cacheAsBitmap` 属性时如何快速平滑滚动文本字段。

该范例源文件 flashtype.fla 位于硬盘上的 Samples 文件夹中。

在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。

在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

Flash 中的字体呈现选项

Flash 8 中有 5 种不同的字体呈现选项。若要选择一个选项，请选择文本字段并打开属性检查器。从 “字体呈现方法”弹出菜单中选择一个选项。

设备字体 此选项生成一个较小的 SWF 文件。此选项使用最终用户计算机上当前安装的字体来呈现文本。

位图文本（关闭消除锯齿） 此选项生成明显的文本边缘，没有消除锯齿。因为此选项生成的 SWF 文件中包含字体轮廓，所以生成一个较大的 SWF 文件。

动画消除锯齿 此选项生成可滑顺畅进行动画播放的消除锯齿文本。因为在文本动画播放时没有应用对齐和消除锯齿，所以在某些情况下，文本动画还可以更快地播放。在使用带有许多字母的大字体或缩放字体时，可能看不到性能上的提高。因为此选项生成的 SWF 文件中包含字体轮廓，所以生成一个较大的 SWF 文件。

可读性消除锯齿 此选项使用高级消除锯齿引擎。此选项提供了品质最高的文本，具有最易读的文本。因为此选项生成的文件中包含字体轮廓，以及特定的消除锯齿信息，所以生成最大的 SWF 文件。

自定义消除锯齿 此选项与 “可读性消除锯齿”选项相同，但是可以直观地操作消除锯齿参数，以生成特定外观。此选项在为新字体或不常见的字体生成最佳的外观方面非常有用。

有关如何在 ActionScript 中使用消除锯齿的示例，请参见第 366 页的 “使用 ActionScript 设置消除锯齿”。

关于连续笔触调整

FlashType 字体呈现技术采用距离字段的内在属性提供连续笔触调整 (CSM)；例如，文本笔触粗细和边缘清晰度的连续调整。CSM 使用两个呈现参数控制自适应采样距离字段 (ADF) 距离到字型密度值的映射。这些参数的最优值非常主观；可由用户首选项、照明条件、显示属性、字型、前景色和背景色以及磅值决定。将 ADF 距离映射到密度值的函数具有一个外侧截止（低于该值时将值设置为 0）和一个内部截止（高于该值时将值设置为最大密度值，如 255）。

使用 ActionScript 设置消除锯齿

Flash 8 提供两种消除锯齿类型：正常和高级。高级消除锯齿仅适用于 Flash Player 8 和更高版本，并且只有将字体嵌入到库中且将字段的 `embedFonts` 属性设置为 `true` 时才可用。对于 Flash Player 8，使用 ActionScript 创建的文本字段的默认设置为正常。

若要设置 `TextField.antiAliasType` 属性的值，请使用以下字符串值：

正常 应用常规文本消除锯齿。这与 Flash Player 在第 7 版和更低版本中使用的消除锯齿类型匹配。

高级 应用高级消除锯齿以提高文本可读性，该类型适用于 Flash Player 8。通过高级消除锯齿，能够以很小的尺寸、很高的品质呈现字型。它最适合在具有大量小字号文本的应用程序中使用。

注意	Macromedia 不建议对大于 48 磅的字体使用高级消除锯齿。

若要使用 ActionScript 设置消除锯齿文本，请参见下面的示例。

使用高级消除锯齿：

1. 创建一个新的 Flash 文档，并将其另存为 **antialiastype.fla**。
2. 在舞台上创建两个影片剪辑，其实例名称为 **normal_mc** 和 **advanced_mc**。

   您将使用这些影片剪辑在两种消除锯齿类型之间进行切换：正常和高级。
3. 打开 “库”面板并从 “库”面板右上角的弹出菜单中选择 “新建字型”。

   随即打开 “字体元件属性”对话框，在该对话框中可选择要嵌入到 SWF 文件中的字体（包括字体样式和字体大小）。还可以指定在文档库和属性检查器中的字体下拉菜单中显示的字体名称 （如果您在舞台上选择了一个文本字段）。
   a. 从 “字体”下拉菜单中选择 “Arial”字体。
   b. 请确保未选中 “粗体”和 “斜体”选项。
   c. 将大小设置为 10 像素。
   d. 输入字体名称 **Arial-10 (embedded)**。
   e. 单击 “确定”。

4. 在库中，右击字体元件，然后从上下文菜单中选择 “链接”。

   即可出现 “链接属性”对话框。

5. 选中 “为 ActionScript 导出”和 “在第一帧导出”选项，输入链接标识符 **Arial-10**，然后单击 “确定”。

6. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var text_fmt:TextFormat = new TextFormat();
text_fmt.font = "Arial-10";
text_fmt.size = 10;

this.createTextField("my_txt", 10, 20, 20, 320, 240);
my_txt.autoSize = "left";
my_txt.embedFonts = true;
my_txt.selectable = false;
my_txt.setNewTextFormat(text_fmt);
my_txt.multiline = true;
my_txt.wordWrap = true;

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function(src:String) {
  if (src != undefined) {
    my_txt.text = src;
  } else {
    my_txt.text = "unable to load text file.";
  }
};
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");

normal_mc.onRelease = function() {
  my_txt.antiAliasType = "normal";
};
advanced_mc.onRelease = function() {
  my_txt.antiAliasType = "advanced";
};
```

前面的代码分为四个关键区域。第一个代码块创建一个新的 TextFormat 对象，该对象指定用于即将创建的文本字段的字体和字体大小。指定的字体 Arial-10 是在上一步骤中嵌入的字体元件的链接标识符。

第二个代码块创建一个新的文本字段，其实例名称为 `my_txt`。为了正确嵌入字体，必须将该文本字段实例的 `embedFonts` 设置为 `true` 。此代码还将新文本字段的文本格式设置为前面创建的 TextFormat 对象。

第三个代码块定义一个 LoadVars 实例，该实例使用外部文本文件的内容对舞台上的文本字段进行填充。文档完全加载完毕之后（但未分析），整个文件内容将复制到 `my_txt.text` 属性，以便在舞台上显示。

最后一个代码块，即第四个代码块为 `normal_mc` 影片剪辑和 `advanced_mc` 影片剪辑定义 `onRelease` 事件处理函数。用户单击和释放这些选项之一时，舞台上的文本字段的消除锯齿类型也将改变。

7. 保存对 FLA 文件所做的更改。
8. 选择“控制”>“测试影片”对 Flash 文档进行测试。
9. 单击舞台上的 `advanced_mc` 影片剪辑。

   单击影片剪辑可将消除锯齿类型从正常（默认）切换到高级。处理具有较小字体大小的文本字段时，将消除锯齿设置为高级可大大提高文本可读性。

提示	使用高级消除锯齿时，即使字号很小，也能达到极高的呈现品质。它最适合具有大量小字号文本的应用程序。Macromedia 不建议对大于 48 磅的字体使用高级消除锯齿。

有关对消除锯齿文本进行格式设置的信息，请参见第 373 页的“使用网格固定类型”和第 371 页的“关于设置消除锯齿文本的格式”。

硬盘上的一个范例文件演示了如何在应用程序中应用和操作消除锯齿文本。使用 FlashType 呈现技术可创建非常清晰易读的小字号文本。此范例还演示了在使用 `cacheAsBitmap` 属性时如何快速平滑滚动文本字段。

该范例源文件 flashtype.fla 位于硬盘上的 Samples 文件夹中。

在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。

在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

设置字体表

如果创建用于 SWF 文件或分发给 Flash 开发人员的字体，可能需要设置字体表来控制字体在舞台上的呈现方式。

高级消除锯齿使用自适应采样距离字段 (ADF) 表示确定字型 （字符）的轮廓。 Flash 使用两个值：

- 一个外侧截止值，低于该值时密度设置为 0。
- 一个内侧截止值，高于该值时密度设置为最大密度值，如 255。

在这两个截止值之间，映射函数是范围从外侧截止处的 0 到内侧截止处最大密度的线性曲线。

调整外侧截止值和内侧截止值会影响笔触粗细和边缘清晰度。这两个参数之间的间距相当于典型消除锯齿方法的滤镜半径的两倍；较窄的间距提供的边缘更清晰，而较宽的间距提供更柔滑、经过更多过滤的边缘。在间距为 0 时，生成的密度图像为双层位图。在间距非常宽时，生成的密度图像具有类似水彩画的边缘。

通常，对于小磅值，用户首选清晰的、高对比边缘；对于动画文本和较大的磅值，用户首选较柔滑的边缘。

外侧截止通常为负值，内侧截止为正值，其中点在 0 附近。对这些参数进行调整将中点向负无穷移动将增加笔触粗细；而将中点向正无穷移动则将减少笔触粗细。

提示	外侧截止应该始终小于或等于内侧截止。

Flash Player 包括 10 种基本字体的高级消除锯齿设置；对于这些字体，仅为从 6 到 20 的字体大小提供高级消除锯齿设置。对于这些字体， 6 之下的所有字体将使用 6 的设置， 20 之上的所有大小都使用 20 的设置。其它字体映射到提供的字体数据。通过 `setAdvancedAntialiasingTable()` 方法，可以为其它字体和字体大小设置自定义消除锯齿数据，或覆盖所提供字体的默认设置。有关创建消除锯齿表的更多信息，请参见下面的示例：

为嵌入字体创建高级消除锯齿表：

1. 创建一个新的 Flash 文档，并将其另存为 **advancedaatable.fla**。
2. 从 “库”面板弹出菜单中选择 “新建字型”。
3. 从 “字体”弹出菜单中选择 “Arial”，然后将字体大小设置为 **32** 磅。
4. 选择 “粗体”和 “斜体”选项。
5. 在 “名称”文本框中输入字体名称 **Arial (embedded)**，然后单击 “确定”。
6. 右击 (Windows) 或按住 Control 单击 (Macintosh) 库中的字体元件，然后选择“链接”。

7. 在 “链接属性”对话框中：
   a. 在 “标识符”文本框中键入 **Arial-embedded**。
   b. 选中 “为 ActionScript 导出”和 “在第一帧导出”。
   c. 单击 “确定”。
8. 在主时间轴中选择第 1 帧，在 “动作”面板中，添加下面的 ActionScript：

```
import flash.text.TextRenderer;
var arialTable:Array = new Array();
arialTable.push({fontSize:16.0, insideCutoff:0.516,
  outsideCutoff:0.416});
arialTable.push({fontSize:32.0, insideCutoff:2.8, outsideCutoff:-2.8});
TextRenderer.setAdvancedAntialiasingTable("Arial", "bolditalic", "dark",
  arialTable);

var my_fmt:TextFormat = new TextFormat();
my_fmt.align = "justify";
my_fmt.font = "Arial-embedded";
my_fmt.size = 32;

this.createTextField("my_txt", 999, 10, 10, Stage.width-20, Stage.height-
  20);
my_txt.antiAliasType = "advanced";
my_txt.embedFonts = true;
my_txt.multiline = true;
my_txt.setNewTextFormat(my_fmt);
my_txt.sharpness = 0;
my_txt.thickness = 0;
my_txt.wordWrap = true;

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function(src:String):Void {
  if (src != undefined) {
    my_txt.text = src + "\n\n" + src;
  } else {
    trace("error downloading text file");
  }
};
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

前面的代码分为四个部分。第一部分代码导入 TextRenderer 类并为两种大小不同的 Arial 字体定义一个新的消除锯齿表。第二部分代码定义一个新的 TextFormat 对象，通过该对象可将文本格式应用于文本字段（在下一部分代码中创建）。第三部分代码创建一个新的文本字段（实例名称为 my_txt），启用高级消除锯齿，应用文本格式对象（前面创建）并启用多行文本和自动换行。最后一部分代码定义一个 LoadVars 对象，该对象用于从外部文本文件加载文本，以及填充舞台上的文本字段。

9. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

   从远程服务器加载文本之后，Flash 将在文本字段显示一些文本，您将看到应用于文本字段的高级消除锯齿表属性。当前 `insideCutoff` 和 `outsideCutoff` 值会使舞台上的嵌入字体看起来会稍微有点模糊效果。

关于文本布局和格式设置

通过使用 ActionScript 可以控制文本布局和格式设置。除了其它形式的格式设置，如样式表（请参见第 377 页的 “使用层叠样式表设置文本格式”）和 HTML 文本（请参见第 390 页的 “使用 HTML 格式的文本”）之外，TextFormat 类还可对文本在运行时的显示方式提供多种控制。

在 SWF 文件中使用消除锯齿文本时还可以通过使用 ActionScript 控制字符在网格上的固定方式。这有助于控制字符在运行时的外观。有关如何在应用程序中使用网格固定类型的示例，请参见第 373 页的 “使用网格固定类型”。

有关文本字段的一般信息，请参见第 342 页的 “关于文本字段”。有关设置文本格式的信息，请参见第 371 页的 “关于设置消除锯齿文本的格式”。有关 TextFormat 类的更多信息，请参见《ActionScript 2.0 语言参考》中的第 375 页的 “使用 TextFormat 类”和 `TextFormat`。

有关使用 TextFormat 类进行文本布局和文本格式设置的更多信息，请参见以下各部分：

关于设置消除锯齿文本的格式

Flash 8 引入了两个新的属性，可在启用高级消除锯齿时设置文本字段的格式：`sharpness` 和 `thickness`。清晰度是指应用于文本字段实例的锯齿量。清晰度值较高可使嵌入字体边缘锯齿更显示并且更清晰。将清晰度设置为较低的值可使字体看起来更为柔滑，更为模糊。对于文本字段而言，设置字体粗细与启用粗体格式类似。粗细值越高，则字体粗体效果越明显。

下面的示例动态加载文本文件并在舞台上显示文本。沿 x 轴移动鼠标指针可将清晰度设置在 -400 和 400 之间。沿 y 轴移动鼠标指针可将粗细设置在 -200 和 200 之间。

修改文本字段的清晰度和粗细：

1. 创建一个新的 Flash 文档，并将其另存为 **sharpness.fla**。
2. 从 “库”面板右上角的弹出菜单中选择 “新建字型”。
3. 从 “字体”下拉菜单中选择 “Arial”并将字体大小设置为 24 磅。
4. 在 “名称”文本框中输入字体名称 **Arial-24 (embedded)**，然后单击 “确定”。
5. 右击库中的字体元件，然后选择 “链接”打开 “链接属性”对话框。
6. 将链接标识符设置为 Arial-24，选中 “为 ActionScript 导出”和 “在第一帧导出”复选框，然后单击 “确定”。
7. 将下面的代码添加到主时间轴中的第 1 帧：

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.size = 24;
my_fmt.font = "Arial-24";

this.createTextField("lorem_txt", 10, 0, 20, Stage.width, (Stage.height -
  20));
lorem_txt.setNewTextFormat(my_fmt);
lorem_txt.text = "loading...";
lorem_txt.wordWrap = true;
lorem_txt.autoSize = "left";
lorem_txt.embedFonts = true;
lorem_txt.antiAliasType = "advanced";

this.createTextField("debug_txt", 100, 0, 0, Stage.width, 20);
debug_txt.autoSize = "left";
debug_txt.background = 0xFFFFFF;

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function(src:String) {
  lorem_txt.text = src;
}
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  lorem_txt.sharpness = (_xmouse * (800 / Stage.width)) - 400;
  lorem_txt.thickness = (_ymouse * (400 / Stage.height)) - 200;
  debug_txt.text = "sharpness=" + Math.round(lorem_txt.sharpness) +
    ", thickness=" + Math.round(lorem_txt.thickness);
};
Mouse.addListener(mouseListener);
```

此 ActionScript 代码可分为五个关键部分。第一部分代码定义一个新的 TextFormat 实例，该实例将应用于动态创建的文本字段。下面两部分代码在舞台上创建两个新文本字段。第一个文本字段 lorem_txt 应用前面创建的自定义文本格式设置对象，启用嵌入字体，并将 antiAliasType 属性设置为 true。第二个文本字段 debug_txt 显示 lorem_txt 文本字段的当前清晰度和粗细值。第四部分代码创建一个 LoadVars 对象，该对象负责加载外部文本文件并填充 lorem_txt 文本字段。最后一部分代码，即第五部分代码定义在舞台上移动鼠标指针时调用的鼠标侦听器。根据鼠标指针在舞台上的当前位置计算 sharpness 和 thickness 的当前值。sharpness 和 thickness 属性是为 lorem_txt 文本字段设置的，其当前值显示在 debug_txt 文本字段中。

8. 选择 “控制” > “测试影片” 来测试该文档。

   沿 x 轴移动鼠标指针以更改文本字段的清晰度。从左到右移动鼠标指针以增加清晰度并使锯齿效果更明显。沿 y 轴移动鼠标指针以更改文本字段的粗细。

有关在 SWF 文件中使用消除锯齿文本的更多信息，请参见第 366 页的 “使用 ActionScript 设置消除锯齿”、第 365 页的 “Flash 中的字体呈现选项” 和第 373 页的 “使用网格固定类型”。

硬盘上的一个范例文件演示了如何在应用程序中应用和操作消除锯齿文本。使用 FlashType 呈现技术可创建清晰易读的小字号文本。此范例还演示了在使用 cacheAsBitmap 属性时如何快速平滑地滚动文本字段。

该范例源文件 flashtype.fla 位于硬盘上的 Samples 文件夹中。

在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。

在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

使用网格固定类型

在文本字段上使用高级消除锯齿时，有三种网格固定类型可供使用：

无 不指定任何网格固定类型。不强制根据像素网格调整字型中的水平线和垂直线。通常，此设置非常适合动画和大字体文本。

像素 指定粗水平线和垂直线适合像素网格。此设置仅适用于左对齐文本字段。该设置通常能为左对齐文本提供最佳可读性。

子像素 指定粗水平线和垂直线适合 LCD 监视器上的子像素网格。通常，子像素设置非常适合右对齐和居中对齐的动态文本，有时，为了在动画与文本品质之间达到一种平衡，也可使用该设置。

下面的示例演示如何通过使用 ActionScript 在文本字段上设置网格固定类型。

在文本字段上设置网格固定类型：

1. 创建一个新的 Flash 文档，并将其另存为 **gridfittype.fla**。
2. 从 “库”面板右上角的弹出菜单中选择 “新建字型”。
3. 从 “字体”下拉菜单中选择 “Arial”字体，并将字体大小设置为 10 磅。
4. 在 “名称”文本框键入字体名称 **Arial-10 (embedded)**，然后单击 “确定”。
5. 右击库中的字体元件，然后选择 “链接”打开 “链接属性”对话框。
6. 将链接标识符设置为 Arial-10，然后选中 “为 ActionScript 导出”和 “在第一帧导出”复选框。
7. 单击 “确定”。
8. 将下面的代码添加到主时间轴中的第 1 帧：

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.size = 10;
my_fmt.font = "Arial-10";
var h:Number = Math.floor(Stage.height / 3);

this.createTextField("none_txt", 10, 0, 0, Stage.width, h);
none_txt.antiAliasType = "advanced";
none_txt.embedFonts = true;
none_txt.gridFitType = "none";
none_txt.multiline = true;
none_txt.setNewTextFormat(my_fmt);
none_txt.text = "loading...";
none_txt.wordWrap = true;

this.createTextField("pixel_txt", 20, 0, h, Stage.width, h);
pixel_txt.antiAliasType = "advanced";
pixel_txt.embedFonts = true;
pixel_txt.gridFitType = "pixel";
pixel_txt.multiline = true;
pixel_txt.selectable = false;
pixel_txt.setNewTextFormat(my_fmt);
pixel_txt.text = "loading...";
pixel_txt.wordWrap = true;

this.createTextField("subpixel_txt", 30, 0, h*2, Stage.width, h);
subpixel_txt.antiAliasType = "advanced";
subpixel_txt.embedFonts = true;
subpixel_txt.gridFitType = "subpixel";
subpixel_txt.multiline = true;
subpixel_txt.setNewTextFormat(my_fmt);
subpixel_txt.text = "loading...";
subpixel_txt.wordWrap = true;

var lorem_lv:LoadVars = new LoadVars();
lorem_lv.onData = function(src:String):Void {
```

```
    if (src != undefined) {
      none_txt.text = "[antiAliasType=none]\n" + src;
      pixel_txt.text = "[antiAliasType=pixel]\n" + src;
      subpixel_txt.text = "[antiAliasType=subpixel]\n" + src;
    } else {
      trace("unable to load text file");
    }
};
lorem_lv.load("http://www.helpexamples.com/flash/lorem.txt");
```

上面的 ActionScript 代码可分为五个部分。第一部分定义一个新的文本格式对象，该对象指定两个属性 `size` 和 `font`。`font` 属性是指当前文档库中的字体元件的链接标识符。第二、第三、第四部分代码分别在舞台上创建一个新的动态文本字段并设置一些常见属性：`antiAliasType`（必须设置为高级）、`embedFonts`（设置为 `true`）、`multiline` 和 `wordWrap`。每部分代码还应用在前面部分代码中创建的文本格式对象，并将网格固定类型设置为正常、像素或子像素。最后一部分代码，即第五部分代码创建一个 LoadVars 实例，该实例将外部文本文件的内容加载到用代码创建的各个文本字段中。

9. 保存该文档，然后选择“控制”>“测试影片”对该 SWF 文件进行测试。

   每个文本字段均应使用值“`loading...`”进行初始化。外部文本文件加载成功之后，每个文本字段将使用不同的网格固定类型显示一些设置格式的范例文本。

提示	FlashType 呈现技术仅在 0 度旋转使用网格固定。

使用 TextFormat 类

您可以使用 TextFormat 类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()` 方法可用来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 ActionScript 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《ActionScript 2.0 语言参考》中的 setTextFormat（TextField.setTextFormat 方法）和 setNewTextFormat（TextField.setNewTextFormat 方法）。

若要通过 TextFormat 类对文本字段进行格式设置，请执行以下操作：

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。

   在舞台上的文本字段中键入一些文本，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” 弹出菜单中选择 “动态”，然后从 “行类型” 弹出菜单中选择 “多行”。

3. 在时间轴中选择第 1 帧，然后打开 “动作” 面板 （“窗口” > “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 `size` 属性设置为 24：

   ```
   // 创建一个 TextFormat 对象。
   var txt_fmt:TextFormat = new TextFormat();
   // 指定段落和字符格式。
   txt_fmt.bold = true;
   txt_fmt.italic = true;
   txt_fmt.size = 24;
   ```

5. 使用 `TextField.setTextFormat()` 将 TextFormat 对象应用于在第 1 步中创建的文本字段：

   ```
   myText_txt.setTextFormat(txt_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，可将格式设置应用于单个字符或字符组。例如，下面的代码将粗体、斜体和 24 磅格式设置应用于您在文本字段中输入的前三个字符：

   ```
   myText_txt.setTextFormat(0, 3, txt_fmt);
   ```

   有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 setTextFormat （TextField.setTextFormat 方法）。

6. 选择 “控制” > “测试影片” 对应用程序进行测试。

有关使用 TextFormat 类的更多信息，请参见以下主题：

- 第 377 页的 “新文本字段的默认属性”
- 第 377 页的 “使用层叠样式表设置文本格式”

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：

```
align = "left"
blockIndent = 0
bold = false
bullet = false
color = 0x000000
font = "Times New Roman" (default font is Times on Mac OS X)
indent = 0
italic = false
kerning = false
leading = 0
leftMargin = 0
letterSpacing = 0
rightMargin = 0
size = 12
tabStops = [] (empty array)
target = ""
underline = false
url = ""
```

注释	Mac OS X 中的默认字体属性是 Times。

有关 TextFormat 方法及其描述的完整列表，请参见《ActionScript 2.0 语言参考》中的 `TextFormat`。

使用层叠样式表设置文本格式

层叠样式表 (CSS) 样式是使用可应用于 HTML 或 XML 文档的文本样式的一种方式。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，下面的样式定义名为 `bodyText` 的选择器：

```
.bodyText {
  text-align: left
}
```

可以创建重新定义 Flash Player 使用的内置 HTML 格式设置标签（如 `<p>` 和 `<li>`）的样式。还可以使用 `<p>` 或 `<span>` 标签的 `class` 属性创建应用于特定 HTML 元素的样式 classes，或定义新标签。

您可以使用 TextField.StyleSheet 类处理文本样式表。虽然 TextField 类可以在 Flash Player 6 中使用，但是 TextField.StyleSheet 类要求 SWF 文件面向 Flash Player 7 或更高版本。您可以从外部 CSS 文件加载样式或使用 ActionScript 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。

使用样式表涉及以下的三个基本步骤：

- 从 TextField.StyleSheet 类创建样式表对象（有关更多信息，请参见《ActionScript 2.0 语言参考》中的 StyleSheet (TextField.StyleSheet)）。
- 通过从外部 CSS 文件加载样式或使用 ActionScript 创建新的样式，将样式添加到样式表对象中。
- 将样式表分配到包含 HTML 或 XML 格式文本的 TextField 对象。

有关更多信息，请参见以下主题：

- 第 379 页的 “支持的 CSS 属性”
- 第 380 页的 “创建样式表对象”
- 第 380 页的 “加载外部 CSS 文件”
- 第 382 页的 “使用 ActionScript 创建新样式”
- 第 382 页的 “将样式应用于 TextField 对象”
- 第 383 页的 “将样式表应用于 TextArea 组件”
- 第 384 页的 “合并样式”
- 第 384 页的 “使用样式类”
- 第 385 页的 “设置内置 HTML 标签的样式”
- 第 385 页的 “在 HTML 中使用样式的示例”
- 第 388 页的 “使用样式定义新标签”
- 第 388 页的 “在 XML 中使用样式的示例”

硬盘上的 Samples 文件夹中有一个范例源文件 formattedText.fla，该源文件演示如何将 CSS 格式设置应用于在运行时加载到 SWF 文件中的文本。

在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\LoadText。

在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/LoadText。

支持的 CSS 属性

Flash Player 支持原 CSS1 规范 (www.w3.org/TR/REC-CSS1) 中的部分属性。下表显示支持的 CSS 属性和值，以及它们对应的 ActionScript 属性名称。（每个 ActionScript 属性名称都来自相应的 CSS 属性名称；省略连字符并将连字符后的字符变成大写。）

CSS 属性	ActionScript 属性	用法和支持的值
text-align	textAlign	可以识别的值包括 `left`、`center`、`right` 和 `justify`。
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`。
kerning	kerning	可以识别的值包括 `true` 和 `false`。
font-style	fontStyle	可以识别的值包括 `normal` 和 `italic`。
letterSpacing	letterSpacing	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-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（如 `blue`）。颜色以下面的格式写入：`#FF0000`。

创建样式表对象

在 ActionScript 中，CSS 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，请调用 TextField.StyleSheet 类的构造函数：

```
var newStyle:TextField.StyleSheet = new TextField.StyleSheet();
```

若要将样式添加到样式表对象中，可以将外部 CSS 文件加载到对象中，也可以在 ActionScript 中定义样式。请参见第 380 页的 “加载外部 CSS 文件”和第 382 页的 “使用 ActionScript 创建新样式”。

硬盘上的 Samples 文件夹中有一个 formattedText.fla 范例源文件，该源文件演示如何将 CSS 格式设置应用于在运行时加载到 SWF 文件中的文本。

在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\LoadText。

在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/LoadText。

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 CSS 编辑器中，创建一个新的文件。
2. 将以下样式定义添加到该文件中：

```
.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，创建一个新 FLA 文件。
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” > “动作”）。

7. 将以下代码添加到 “动作”面板中：

```
var styles:TextField.StyleSheet = new TextField.StyleSheet();
styles.onLoad = function(success:Boolean):Void {
  if (success) {
    // 显示样式名称。
    trace(this.getStyleNames());
  } else {
    trace("Error loading CSS file.");
  }
};
styles.load("styles.css");
```

注意 在上面的代码片段中，`this.getStyleNames()` 是指您在 ActionScript 的第一行中构造的 `styles` 对象。

8. 将 FLA 文件保存到包含 styles.css 的同一个目录中。

9. 测试 Flash 文档 （“控制” > “测试影片”）。

   您应该可以在 “输出”面板中看到两个样式的名称：

   ```
   .bodyText,.headline
   ```

   如果在 “输出”面板中看到了 “加载 CSS 文件时出错。”，则请确保 FLA 文件和 CSS 文件在同一个目录中并且正确键入了 CSS 文件的名称。

与通过网络加载数据的其它所有 ActionScript 方法一样，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 620 页的 “SWF 文件之间的跨域和子域访问”。）有关在 Flash 中使用 CSS 的更多信息，请参见 《ActionScript 2.0 语言参考》中的 StyleSheet (TextField.StyleSheet)。

硬盘上的 Samples 文件夹中有一个 formattedText.fla 范例源文件，该源文件演示如何将 CSS 格式设置应用于在运行时加载到 SWF 文件中的文本。

在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\LoadText。

在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/LoadText。

使用 ActionScript 创建新样式

可以通过 ActionScript 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，下面的代码创建一个名为 `styles` 的样式表对象，该对象定义两个样式，这两个样式与您已经导入的样式相同（请参见第 380 页的 “加载外部 CSS 文件”）：

```
var styles:TextField.StyleSheet = 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 TextField 对象，请将样式表对象分配到文本字段的 `styleSheet` 属性。

```
textObj_txt.styleSheet = styles;
```

注意 请不要将 `TextField.styleSheet` 属性和 TextField.StyleSheet 类混淆。大写字母指示其区别。

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。

将样式表应用于 TextArea 组件

要将样式表应用于 TextArea 组件，您可以创建一个样式表对象，并使用 TextField.StyleSheet 类为其分配 HTML 样式。然后为 TextArea 组件的 `styleSheet` 属性分配样式表。

下面的示例创建了一个样式表对象 `styles`，并将其分配给 `myTextArea` 组件实例。

在 TextArea 组件中使用样式表：

1. 创建一个新的 Flash 文档，并将其保存为 **textareastyle.fla**。
2. 将 TextArea 组件从 “组件” 面板的 “用户界面” 文件夹拖动到舞台上，并为其指定实例名称 **myTextArea**。
3. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
// Create a new style sheet object and set styles for it.
var styles:TextField.StyleSheet = new TextField.StyleSheet();
styles.setStyle("html", {fontFamily:'Arial,Helvetica,sans-serif',
            fontSize:'12px',
            color:'#0000FF'});
styles.setStyle("body", {color:'#00CCFF',
            textDecoration:'underline'});
styles.setStyle("h1",{fontFamily:'Arial,Helvetica,sans-serif',
            fontSize:'24px',
            color:'#006600'});

/* Assign the style sheet object to myTextArea component. Set html
  property to true, set styleSheet property to the style sheet object. */
myTextArea.styleSheet = styles;
myTextArea.html = true;

var myVars:LoadVars = new LoadVars();
// Define onData handler and load text to be displayed.
myVars.onData = function(myStr:String):Void {
  if (myStr != undefined) {
    myTextArea.text = myStr;
  } else {
    trace("Unable to load text file.");
  }
};
myVars.load("http://www.helpexamples.com/flash/myText.htm");
```

上面的代码块创建了一个新的 TextField.StyleSheet 实例，该实例定义三种样式：用于 `html`、`body` 和 `h1` HTML 标签。接着，将该样式表对象应用于 TextArea 组件并启用 HTML 格式设置。其余的 ActionScript 定义了一个 LoadVars 对象，该对象加载一个外部 HTML 文件并使用加载的文本填充文本区域。

4. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

合并样式

Flash Player 中的 CSS 样式是添加式的：也就是说，当嵌套样式时，每层嵌套都可以提供样式信息，这些信息共同形成最终的格式设置。

下面的示例显示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
<emphasized>emphatic</emphasized> 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”（并非真正的 ActionScript 2.0 类），可以使用 `<p>` 或 `<span>` 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 `<span>` 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，可以将这些样式应用于 `<p>` 和 `<span>` 标签，如下面的代码片段所示：

```
<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 390 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，下面的代码定义内置 <p> HTML 标签的样式。该标签的所有实例均按样式规则指定的方式进行样式设置。

```
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	当鼠标在链接上方时应用于 <a> 锚标签。此样式在所有 a 和 a:link 样式之后应用。鼠标从链接上方移开之后，a:hover 样式将从该链接中删除。
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。松开鼠标按钮之后，a:active 样式将从该链接中删除。

在 HTML 中使用样式的示例

本节介绍将样式用于 HTML 标签的示例。您可以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。然后，您可以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. 使用您常用的文本或 CSS 编辑器新建一个文件。
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. 在文本或 HTML 编辑器中创建一个新的文本文件，并另存为 **myText.htm**。

   在文件中添加下面的文本：

```
<p class='headline'>Flash adds FlashType rendering technology!</
  p><p><span class='byline'>San Francisco, CA</span>--Macromedia Inc.
  announced today a new version of Flash that features a brand new font
  rendering technology called FlashType, most excellent at rendering
  small text with incredible clarity and consistency across platforms.
  For more information, visit the <a href='http://
  www.macromedia.com'>Macromedia Flash web site.</a></p>
```

注意	如果要复制和粘贴此文本字符串，请确保删除所有可能已添加到该文本字符串中的换行符。

5. 在 Flash 创作工具中创建一个新的 Flash 文档。
6. 在时间轴 （“窗口” > “时间轴”）中选择图层 1 中的第 1 帧。

7. 打开 “动作”面板 （“窗口” > “动作”），将下面的代码添加到 “动作”面板：

```
this.createTextField("news_txt", 99, 50, 50, 450, 300);
news_txt.border = true;
news_txt.html = true;
news_txt.multiline = true;
news_txt.wordWrap = true;
// Create a new style sheet and LoadVars object.
var myVars_lv:LoadVars = new LoadVars();
var styles:TextField.StyleSheet = new TextField.StyleSheet();
// Location of CSS and text files to load.
var txt_url:String = "myText.htm";
var css_url:String = "html_styles.css";
// Define onData handler and load text to display.
myVars_lv.onData = function(src:String):Void {
  if (src != undefined) {
    news_txt.htmlText = src;
  } else {
    trace("Unable to load HTML file");
  }
};
myVars_lv.load(txt_url);

// Define onLoad handler and Load CSS file.
styles.onLoad = function(success:Boolean):Void {
  if (success) {
    //* 如果样式表正确加载，
       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 = styles;
    news_txt.text = storyText;
  } else {
    trace("Unable to load CSS file.");
  }
};
styles.load(css_url);
```

> **提示** 在此 ActionScript 中，您将从一个外部文件加载文本。有关加载外部数据的信息，请参见第 15 章 “使用图像、声音和视频”。

8. 将该文件另存为 **news_html.fla** 并保存到在第 3 步中创建的 CSS 文件所在的同一个目录中。
9. 选择 “控制” > “测试影片”查看自动应用于 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，方式与使用内置 HTML 标签相同。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，下面的样式表创建新样式 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
}
```

然后，可以使用下面的 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>
```

在 XML 中使用样式的示例

在本部分中，您将创建一个具有 XML 格式文本的 FLA 文件。您将使用 ActionScript 创建一个样式表，而不是从 CSS 文件导入样式（如第 385 页的 “在 HTML 中使用样式的示例”所示）

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” > “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 “行类型”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，请打开“动作”面板（“窗口”>“动作”）并将以下代码添加到“动作”面板：

```
var styles:TextField.StyleSheet = new TextField.StyleSheet();
styles.setStyle("mainBody", {
  color:'#000000',
  fontFamily:'Arial,Helvetica,sans-serif',
  fontSize:'12',
  display:'block'
});
styles.setStyle("title", {
  color:'#000000',
  fontFamily:'Arial,Helvetica,sans-serif',
  fontSize:'18',
  display:'block',
  fontWeight:'bold'
});
styles.setStyle("byline", {
  color:'#666600',
  fontWeight:'bold',
  fontStyle:'italic',
  display:'inline'
});
styles.setStyle("a:link", {
  color:'#FF0000'
});
styles.setStyle("a:hover", {
  textDecoration:'underline'
});
```

此代码创建一个名为 `styles` 的新样式表对象，该对象通过使用 `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. 若要创建要分配给文本字段的 XML 文本，请打开文本编辑器并将以下文本输入到新文档中：

```
<story><title>Flash now has FlashType</title><mainBody><byline>San
  Francisco, CA</byline>--Macromedia Inc. announced today a new version
  of Flash that features the new FlashType rendering technology. For more
  information, visit the <a href="http://www.macromedia.com">Macromedia
  Flash website</a></mainBody></story>
```

注意 如果要复制和粘贴此文本字符串，请确保删除所有可能已添加到该文本字符串中的换行符。从“动作”面板中的弹出菜单选择“隐藏字符”，查看并删除所有额外的换行符。

8. 将该文本文件另存为 **story.xml**。

9. 在 Flash 中，将以下代码添加到 “动作”面板中步骤 6 中的代码之后。

   此代码将加载 story.xml 文档，将样式表对象分配到文本字段的 `styleSheet` 属性，并将 XML 文本分配到文本字段：

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean):Void {
  if (success) {
    news_txt.styleSheet = styles;
    news_txt.text = my_xml;
  } else {
    trace("Error loading XML.");
  }
};
my_xml.load("story.xml");
```

   注意 在此 ActionScript 中，您将从外部文件加载 XML 数据。有关加载外部数据的信息，请参见第 15 章 “使用图像、声音和视频”。

10. 将该文件另存为 **news_xml.fla**，并与 story.xml 在同一文件夹中。

11. 运行 SWF 文件（“控制”>“测试影片”），查看自动应用于文本字段中的文本的样式。

使用 HTML 格式的文本

Flash Player 支持部分标准 HTML 标签（如 `<p>` 和 `<li>`），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 `<img>` 标签，该标签允许您在文本字段中嵌入图像文件（JPEG、GIF、PNG）、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 399 页的 “关于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 375 页的 “使用 TextFormat 类”。

有关 HTML 格式的文本的更多信息，请参见以下主题：

- 第 391 页的 “使用 HTML 格式文本必需的属性和语法”
- 第 392 页的 “关于支持的 HTML 标签”
- 第 398 页的 “关于支持的 HTML 实体”
- 第 399 页的 “关于在文本字段中嵌入图像、 SWF 文件和影片剪辑”

使用 HTML 格式文本必需的属性和语法

若要在文本字段中使用 HTML，您必须在属性检查器中或使用 ActionScript 来设置该文本字段的多个属性：

- 通过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`，启用文本字段的 HTML 格式设置。
- 若要使用 HTML 标签（如 `<p>`、`<br>` 和 `<img>`），您必须通过选择属性检查器中的“多行”选项或将文本字段的 `multiline` 属性设置为 `true`，以使文本字段成为多行文本字段。
- 在 ActionScript 中，将 `TextField.htmlText` 值设置为要显示的 HTML 格式文本字符串。

例如，下面的代码为名为 `headline_txt` 的文本字段启用 HTML 格式设置，然后将一些 HTML 分配到该文本字段：

```
this.createTextField("headline_txt", 1, 10, 10, 500, 300);
headline_txt.html = true;
headline_txt.wordWrap = true;
headline_txt.multiline = true;
headline_txt.htmlText = "<font face='Times New Roman' size='25'>This is how
  you assign HTML text to a text field.</font><br>It's very useful.</br>";
```

若要正确地呈现 HTML，必须使用正确的语法。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 无法 正确呈现下面的 HTML 代码片段，因为分配给 `align` 属性的值 (`left`) 未包含在引号中：

```
this.createTextField("myField_txt", 10, 10, 10, 400, 200);
myField_txt.html = true;
myField_txt.htmlText = "<p align=left>This is left-aligned text</p>";
```

如果将属性值包含在双引号中，则必须对引号进行转义 处理 (`\"`)。采用以下两种方法之一均可：

```
myField_txt.htmlText = "<p align='left'>This uses single quotes</p>";
myField_txt.htmlText = "<p align=\"left\">This uses escaped double quotes</
  p>";
myField_txt.htmlText = '<p align="left">This uses outer single quotes</p>';
myField_txt.htmlText = '<p align=\'left\'>This uses escaped single 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 ActionScript 中分配文本字符串时才需执行此操作。

关于支持的 HTML 标签

本部分列出 Flash Player 支持的内置 HTML 标签。还可以使用 CSS 创建新的样式和标签；请参见第 377 页的 “使用层叠样式表设置文本格式”。

有关支持的 HTML 标签的更多信息，请参见以下主题：

锚标签

<a> 标签创建超文本链接并支持以下属性：

- `href` 一个最多可达 128 个字符的字符串，用于指定要加载到浏览器的页面的 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）。URL 的绝对引用的一个示例是 `http://www.macromedia.com`；相对引用的一个示例是 `/index.html`。
- `target` 指定要加载页面的目标窗口名称。选项包括 `_self`、`_blank`、`_parent` 和 `_top`。`_self` 选项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。

例如，以下 HTML 代码创建链接 “Go home”，该链接将在新浏览器窗口中打开 www.macromedia.com：

```
urlText_txt.htmlText = "<a href='http://www.macromedia.com'
  target='_blank'>Go home</a>";
```

可以使用特殊 `asfunction` 协议使该链接执行 SWF 文件中的 ActionScript 函数，而不是打开一个 URL。有关 `asfunction` 协议的更多信息，请参见 《ActionScript 2.0 语言参考》中的 asfunction 协议。

您还可以使用样式表为锚记标签定义 `a:link`、`a:hover` 和 `a:active` 样式。请参阅第 385 页的 “设置内置 HTML 标签的样式”。

注释	绝对 URL 必须以 `http://` 为前缀，否则 Flash 会将其视为相对 URL。

粗体标签

<b> 标签将文本呈现为粗体，如下面的示例所示：

```
text3_txt.htmlText = "He was <b>ready</b> to leave!";
```

粗体字体必须可用于显示文本的字体。

换行标签

 标签将在文本字段中创建一个换行符。若要使用此标签，您必须将文本字段设置为多行文本字段。

在下面的示例中，换行符位于句子中间：

```
this.createTextField("text1_txt", 1, 10, 10, 200, 100);
text1_txt.html = true;
text1_txt.multiline = true;
text1_txt.htmlText = "The boy put on his coat.<br />His coat was <font
  color='#FF0033'>red</font> plaid.";
```

字体标签

<font> 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，下面的 HTML 代码创建红色文本：

  ```
  myText_txt.htmlText = "<font color='#FF0000'>This is red text</font>";
  ```

- `face` 指定要使用的字体的名称。如下面的示例所示，您可以指定一个逗号分隔的字体名称的列表，这种情况下 Flash Player 选择第一个可用的字体：

  ```
  myText_txt.htmlText = "<font face='Times, Times New Roman'>Displays as
    either Times or Times New Roman...</font>";
  ```

  如果用户的计算机系统上没有安装指定字体，或字体未嵌入到 SWF 文件中，则 Flash Player 将选择替代字体。

  有关在 Flash 应用程序中嵌入字体的更多信息，请参见 《ActionScript 2.0 语言参考》中的 embedFonts（TextField.embedFonts 属性）和 《使用 Flash》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），如下面的示例所示：

  ```
  myText_txt.htmlText = "<font size='24' color='#0000FF'>This is blue, 24-
    point text</font>";
  ```

  您也可以使用相对磅值来代替像素大小，如 +2 或 -4。

图像标签

<img> 标签允许您将外部图像文件（JPEG、GIF、PNG）、SWF 文件和影片剪辑嵌入到文本字段中和 TextArea 组件实例中。在文本字段或组件中，文本在嵌入的图像旁自动换行。若要使用该标签，必须将动态或输入文本字段设置为多行和文本换行。

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”菜单中选择“多行”。
- 对于在运行时使用 createTextField（MovieClip.createTextField 方法）创建的文本字段，请将新文本字段实例的 multiline（TextField.multiline 属性）和 multiline（TextField.multiline 属性）设置为 `true`。

<img> 标签具有一个必需属性 `src`，该属性指定图像文件、SWF 文件的路径或库中影片剪辑元件的链接标识符。所有其它属性都是可选的。

<img> 标签支持以下属性：

- `src` 指定图像或 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、GIF、PNG 和 SWF 文件）只有在完全下载之后才能显示。
- `id` 指定包含嵌入的图像文件、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 ActionScript 控制嵌入的内容。
- `width` 所插入的图像、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。有效值为 `left` 和 `right`。默认值是 `left`。
- `hspace` 指定图像周围不显示任何文本时的水平空间量。默认值是 8。
- `vspace` 指定图像周围不显示任何文本的垂直空间量。默认值是 8。

有关使用 <img> 标签的更多信息和示例，请参见第 399 页的“关于在文本字段中嵌入图像、SWF 文件和影片剪辑”。

斜体标签

<i> 标签将标签中的文本显示为斜体，如下面的代码所示：

```
That is very <i>interesting</i>.
```

此代码示例将以如下方式呈现：

That is very interesting.

斜体字体必需可用于所使用的字体。

列表项标签

 标签在所包含的文本前放置项目符号，如下面的代码所示：

```
Grocery list:
<li>Apples</li>
<li>Oranges</li>
<li>Lemons</li>
```

此代码示例将以如下方式呈现：

Grocery list:

- Apples
- Oranges
- Lemons

注释 Flash Player 不能识别有序和无序列表 （ 和 <ul> 标签），所以这些标签无法修改列表的呈现方式。所有的列表项都有项目符号。

段落标签

<p> 标签创建一个新段落。若要使用此标签，您必须将文本字段设置为多行文本字段。

<p> 标签支持以下属性：

- `align` 指定段落内的文本对齐方式；有效值为 `left`、`right`、`justify` 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 384 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。

```
this.createTextField("myText_txt", 1, 10, 10, 400, 100);
myText_txt.html = true;
myText_txt.multiline = true;
myText_txt.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:TextField.StyleSheet = new TextField.StyleSheet();
myStyleSheet.setStyle(".blue", {color:'#99CCFF', fontSize:18});
this.createTextField("test_txt", 10, 0, 0, 300, 100);
test_txt.html = true;
test_txt.styleSheet = myStyleSheet;
test_txt.htmlText = "<p class='blue'>This is some body-styled text.</
  p>.";
```

Span 标签

 标签只可用于 CSS 文本样式。（有关更多信息，请参见第 377 页的“使用层叠样式表设置文本格式”。）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 384 页的 “使用样式类”。

文本格式标签

<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《ActionScript 2.0 语言参考》中的 blockIndent（TextFormat.blockIndent 属性）。）
- indent 指定从左边缘到段落中第一个字符的缩进；对应于 TextFormat.indent。允许使用负整数。（请参见《ActionScript 2.0 语言参考》中的 indent（TextFormat.indent 属性）。）
- leading 指定行与行之间的前导量（垂直间距）；对应于 TextFormat.leading。允许使用负整数。（请参见《ActionScript 2.0 语言参考》中的 leading（TextFormat.leading 属性）。）
- leftmargin 指定段落的左边距（以磅值为单位）；对应于 TextFormat.leftMargin。（请参见《ActionScript 2.0 语言参考》中的 leftMargin（TextFormat.leftMargin 属性）。）
- rightmargin 指定段落的右边距（以磅值为单位）；对应于 TextFormat.rightMargin。（请参见《ActionScript 2.0 语言参考》中的 rightMargin（TextFormat.rightMargin 属性）。）
- 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《ActionScript 2.0 语言参考》中的 tabStops（TextFormat.tabStops 属性）。）

下面具有粗体行标题的数据表是下面过程中的代码示例的结果：

Name	Age	Occupation
Rick	33	Detective
AJ	34	Detective

若要使用 Tab 键停靠位创建具有格式的数据表，请执行以下操作：

1. 创建一个新的 Flash 文档，并将其另存为 **tabstops.fla**。
2. 在时间轴中，选择图层 1 上的第 1 帧。
3. 打开 “动作”面板 （“窗口” > “动作”），然后在 “动作”面板中输入以下代码：

```
// 创建新的文本字段。
this.createTextField("table_txt", 99, 50, 50, 450, 100);
table_txt.multiline = true;
table_txt.html = true;
// 创建以制表符分隔的列标题 （格式为粗体）。
var rowHeaders:String = "<b>Name\tAge\tOccupation</b>";

// 创建带有数据的行。
var row_1:String = "Rick\t33\tDetective";
var row_2:String = "AJ\t34\tDetective";

// 设置两个 Tab 键停靠位，分别为 50 和 100 磅。
table_txt.htmlText = "<textformat tabstops='[50,100]'>";
table_txt.htmlText += rowHeaders;
table_txt.htmlText += row_1;
table_txt.htmlText += row_2 ;
table_txt.htmlText += "</textformat>";
```

   使用 Tab 字符转义序列 (`\t`) 时会在表中每列之间添加制表符。可以使用 += 运算符追加文本。

4. 选择 “控制” > “测试影片”查看设置了格式的表。

下划线标签

`<u>` 标签为标签中的文本添加下划线，如下面的代码所示：

```
This is <u>underlined</u> text.
```

此代码将以如下方式呈现：

This is underlined text.

关于支持的 HTML 实体

HTML 实体有助于在 HTML 格式文本字段中显示某些字符，因此它们不会被解释为 HTML。例如，使用小于号 (<) 和大于号 (>) 将 HTML 标签括起来，如 `<img>` 和 `<span>`。若要在 Flash 中的 HTML 格式文本字段中显示小于号或大于号字符，您需要将 HTML 实体替换为这些字符。下面的 ActionScript 在舞台上创建一个 HTML 格式文本字段，并使用 HTML 实体显示字符串 “<b>”但不会粗体显示文本：

```
this.createTextField("my_txt", 10, 100, 100, 100, 19);
my_txt.autoSize = "left";
my_txt.html = true;
my_txt.htmlText = "The &lt;b&gt; tag makes text appear <b>bold</b>.";
```

在运行时，上面的代码示例在 Flash 中将在舞台上显示以下文本：

The <b> tag makes text appear **bold**.

除了大于号和小于号之外， Flash 还可以识别下表中列出的其它 HTML 实体。

实体	说明
`<`	< （小于）
`>`	> （大于）
`&`	& （和）
`&quot;`	" （双引号）
`&apos;`	' （撇号，单引号）

Flash 还支持显式字符代码，如 `&#39;` (ampersand - ASCII) 和 `&#x0026;` (ampersand - Unicode)。

下面的 ActionScript 演示如何使用 ASCII 或 Unicode 字符代码嵌入代字号 (~) 字符：

```
this.createTextField("my_txt", 10, 100, 100, 100, 19);
my_txt.autoSize = "left";
my_txt.html = true;
my_txt.htmlText = "&#126;"; // 代字号 (ASCII)
my_txt.htmlText += "\t"
my_txt.htmlText += "&#x007E;"; // 代字号 (Unicode)
```

关于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 <img> 标签在动态和输入文本字段中嵌入图像文件 （JPEG、 GIF、 PNG）、 SWF 文件和影片剪辑，以及 TextArea 组件实例。（有关 <img> 标签属性的完整列表，请参见第 394 页的 “图像标签”。）

Flash 以完全大小显示文本字段中嵌入的媒体。若要指定嵌入的媒体的尺寸，请使用 <img> 标签的 height 和 width 属性。（请参见第 401 页的 “关于指定高度和宽度值”。）

通常情况下，文本字段中嵌入的图像显示在 <img> 标签后的行上。但是，如果 <img> 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 和图像文件

若要将图像或 SWF 文件嵌入到文本字段中，请在 <img> 标签的 src 属性中指定图像（GIF、 JPEG 或 PNG）或 SWF 文件的绝对或相对路径。例如，以下代码插入一个 GIF 文件，该文件与 SWF 文件位于同一个目录中 （相对地址，联机或脱机）。

在文本字段中嵌入图像：

1. 创建一个新的 Flash 文档，并将其另存为 **embedding.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
this.createTextField("image1_txt", 10, 50, 50, 450, 150);
image1_txt.html = true;
image1_txt.htmlText = "<p>Here's a picture from my vacation:<img
  src='beach.gif'>";
```

   上面的代码在舞台上创建一个新的动态文本字段，启用 HTML 格式设置，并向该文本字段添加一些文本和本地图像。

3. 在上一步骤中添加的代码下面添加以下 ActionScript：

```
this.createTextField("image2_txt", 20, 50, 200, 400, 150);
image2_txt.html = true;
image2_txt.htmlText = "<p>Here's a picture from my garden:<img
  src='http://www.helpexamples.com/flash/images/image2.jpg'>";
```

   还可以使用绝对地址插入图像。上面的代码插入一个 JPEG 文件，该文件位于服务器上的一个目录中。包含此代码的 SWF 文件可能在您的硬盘上，也可能在服务器上。

4. 保存该文档，然后选择 “控制” > “测试影片”对该文档进行测试。

   上面的文本字段应具有一句文本，很可能是 “输出”面板中的错误消息，说明 Flash 无法在当前目录中定位名为 beach.gif 的文件。下面的文本字段应具有一句文本以及一个从远程服务器加载的花朵图像。

   将 GIF 图像复制到与 FLA 所在的相同目录，并将该图像重命名为 beach.gif，然后选择 “控制” > “测试影片”，重新测试该 Flash 文档。

注释	使用绝对 URL 时，必须确保 URL 以 `http://` 为前缀。

嵌入影片剪辑元件

若要在文本字段中嵌入影片剪辑元件，请为 `<img>` 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 323 页的 “将影片剪辑元件附加到舞台”。）

例如，下面的代码将一个链接标识符为 `symbol_ID` 的影片剪辑元件插入到实例名称为 `textField_txt` 的动态文本字段中。

将影片剪辑嵌入到文本字段中：

1. 创建一个新的 Flash 文档，并将其另存为 **embeddedmc.fla**。
2. 在舞台上绘制新的形状，或者选择 “文件” > “导入” > “导入到舞台”，并选择一个大约 100 像素宽乘以 100 像素高的图像。
3. 在舞台上选择该形状或前面步骤中导入的图像，然后按 F8 打开 “转换为元件”对话框，对其进行转换。
4. 将行为设置为影片剪辑，并输入描述性的元件名称。选择注册点网格左上的方型，然后单击 “高级”切换到高级模式 （如果尚未切换）。
5. 选中 “为 ActionScript 导出”和 “在第一帧导出”复选框。
6. 在 “标识符”文本框中输入链接标识符 **img_id**，然后单击 “确定”。
7. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
this.createTextField("textField_txt", 10, 0, 0, 300, 200);
textField_txt.html = true;
textField_txt.htmlText = "<p>Here's a movie clip symbol:<img
  src='img_id'>";
```

   为了正确完整地显示嵌入的影片剪辑，其元件的注册点应该位于点 (0,0)。

8. 保存对 Flash 文档的更改。
9. 选择 “控制” > “测试影片”对该 Flash 文档进行测试。

关于指定高度和宽度值

如果为 <img> 标签指定了 width 和 height 属性，则将在文本字段中为图像文件、SWF 文件或影片剪辑保留空间。图像或 SWF 文件完全下载之后，将显示在保留空间中。Flash 将根据为 height 和 width 指定的值来增大或缩小媒体。必须为 height 和 width 属性都输入值才能缩放该图像。

如果未指定 height 和 width 值，则不会为嵌入的媒体保留空间。图像或 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。

提示	如果您动态地将图像加载到包含文本的文本字段中，最好指定原始图片的宽度和高度，这样文本就可以在为图像保留的空间周围正确地自动换行。

使用 ActionScript 控制嵌入的媒体

Flash 为每个 <img> 标签创建一个新的影片剪辑并在 TextField 对象中嵌入该影片剪辑。<img> 标签的 id 属性允许您将实例名称分配到创建的影片剪辑。这允许您使用 ActionScript 控制该影片剪辑。

Flash 创建的影片剪辑作为子级影片剪辑添加到包含该图像的文本字段中。

例如，下面的示例在文本字段中嵌入一个 SWF 文件。

在文本字段中嵌入一个 SWF 文件：

1. 创建一个新的 Flash 文档。
2. 将文档的舞台大小调整为 100 像素乘以 100 像素。
3. 使用矩形工具在舞台上绘制一个红色矩形。
4. 通过使用属性检查器将该方型大小调整为 80 像素乘以 80 像素，然后将该形状移动到舞台中央。
5. 在时间轴上选择第 20 帧，然后按 F7 键（Windows 或 Macintosh）插入一个新的空关键帧。
6. 使用椭圆工具在舞台上第 20 帧绘制一个蓝色的圆形。
7. 通过使用属性检查器将该圆形大小调整为 80 像素乘以 80 像素，然后将其移动到舞台中央。
8. 单击第 1 帧和第 20 帧之间的空白帧，在属性检查器中将补间类型设置为 “形状”。
9. 将当前文档另存为 **animation.fla**。
10. 选择 “控制” > “测试影片”来预览动画。

    SWF 文件创建在与 FLA 相同的目录中。为使此练习正常工作，需要生成 SWF 文件以便将其加载到单独的 FLA 文件中。

11. 创建一个新的 FLA 文件，并将其另存为 **animationholder.fla**。

    将该文件保存在与上面创建的 animation.fla 文件相同的文件夹中。

12. 将以下 ActionScript 代码添加到主时间轴的第一帧中：

```
this.createTextField("textField_txt", 10, 0, 0, 300, 200);
textField_txt.html = true;
textField_txt.htmlText = "Here's an interesting animation: <img
  src='animation.swf' id='animation_mc'>";
```

    在本例中，新建影片剪辑的完全限定路径为 `textField_txt.animation_mc`。

13. 保存对 Flash 文档的更改，然后选择 “控制” > “测试影片”，预览文本字段中的动画。

若要在文本字段中播放 SWF 文件时控制该文件，请完成下一练习。

控制在文本字段中播放的 SWF 文件：

1. 按照 第 401 页的 “使用 ActionScript 控制嵌入的媒体” 下第一个过程中的步骤操作。
2. 在舞台上创建一个按钮实例，在属性检查器中为其分配实例名称 **stop_btn** 。
3. 在主时间轴的第 1 帧现有代码下面，添加以下 ActionScript 代码：

```
stop_btn.onRelease = function() {
  textField_txt.animation_mc.stop();
};
```

4. 选择 “控制” > “测试影片” 对应用程序进行测试。

    现在，只要单击 `stop_btn` 按钮实例，文本字段中嵌套的动画的时间轴就会停止。

有关将嵌入媒体制成超链接的信息，请参见第 402 页的 “关于使用嵌入的媒体创建超文本链接”。

关于使用嵌入的媒体创建超文本链接

若要通过嵌入的图像文件、 SWF 文件或影片剪辑创建超文本链接，请将 `<img>` 标签括在 `<a>` 标签中：

```
textField_txt.htmlText = "Click the image to return home<a
  href='home.htm'><img src='home.jpg'></a>";
```

当鼠标指针位于包含在 `<a>` 标签中的图像、 SWF 文件或影片剪辑上时，鼠标指针会变为 “手指形” 图标，这与标准超文本链接的处理方式是相同的。交互动作 （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。

有关嵌入媒体的信息，请参见第 402 页的 “关于使用嵌入的媒体创建超文本链接”。

示例：创建滚动文本

在 Flash 中创建滚动文本的方法有多种。通过选择 “文本”菜单或上下文菜单中的 “可滚动”选项或按住 Shift 键双击文本字段句柄，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` 属性在文本字段中控制垂直滚动，使用 `hscroll` 和 `maxhscroll` 属性在文本字段中控制水平滚动。`scroll` 和 `hscroll` 属性分别指定当前垂直和水平滚动位置；您可以对这些属性进行读写操作。`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。

TextArea 组件提供了一种简便的方法，可以通过撰写最少的脚本来创建滚动文本字段。有关更多信息，请参见《组件语言参考》中的 “TextArea 组件”。

创建可滚动动态文本字段：

执行以下操作之一：

- 按住 Shift 键双击动态文本字段上的手柄。
- 使用选择工具选择动态文本字段，然后选择 “文本” > “可滚动”。
- 用选择工具选择动态文本字段。右击 (Windows) 或按住 Control 键单击 (Macintosh) 动态文本字段，然后选择 “文本” > “可滚动”。

使用滚动属性创建滚动文本：

1. 执行以下操作之一：
   - 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_txt**。
   - 使用 ActionScript，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_txt** 作为该方法的参数。

注意	如果不 动态地将文本加载到 SWF 文件中，则从主菜单中选择 “文本” > “可滚动”。

2. 创建一个向上按钮和一个向下按钮，或选择 “窗口” > “公用库” > “按钮”，然后将按钮拖动到舞台上。

   您将使用这些按钮来上下滚动文本。
3. 选择舞台中的向下按钮，并在 “实例名称”文本框中键入 **down_btn**。
4. 选择舞台中的向上按钮，并在 “实例名称”文本框中键入 **up_btn**。
5. 在时间轴上选择第 1 帧，然后在 “动作”面板 （“窗口” > “动作”）中输入下面的代码，以在文本字段中向下滚动文本：

```
down_btn.onPress = function() {
  textField_txt.scroll += 1;
};
```

6. 在第 5 步中的 ActionScript 之后，输入下面的代码，以向上滚动文本：

```
up_btn.onPress = function() {
  textField_txt.scroll -= 1;
};
```

可以用向上和向下按钮滚动加载到 `textField_txt` 文本字段的任何文本。

关于字符串和 String 类

在编程语言中，字符串是字符的有序序列。在 Flash 文档和类文件中经常要用字符串来在应用程序中显示文本，如在文本字段内。而且，还可以将值存储为字符串，以在应用程序中用于多种目的。可以通过在字符数据两端加上引号直接将字符串放到 ActionScript 代码中。有关创建字符串的更多信息，请参见第 411 页的 “创建字符串”。有关使用文本字段的信息，请参见第 344 页的 “使用 TextField 类”。

可以将每个字符与特定的字符代码相关联，还可以选用该代码来显示文本。例如，字符“A”用 Unicode 字符代码 0041 或用 ASCII（美国标准信息交换码）代码 65 表示。有关字符代码和代码图表的更多信息，请参见 www.unicode.org/charts。如您所看到的，Flash 文档中的字符串表示方法主要取决于所选的字符集以及字符的编码方式。

字符编码 是指一种代码或方法，用以将一种语言中的一个字符集表示为表示码（如数值）。字符代码（在上一段中已提到）是一个映射值表（如 ASCII 表，其中 A 等于 65）。编码方法将在计算机程序中对其进行解密。

例如，英语中的每个字母在一种字符编码中都有一个表示性的数值代码。 ASCII 将每个字母、数字和一些符号编码为每个整数的 7 位二进制版本。ASCII 是由 95 个可打印字符和大量控制字符组成的字符集； ASCII 由计算机用于表示文本。

与 ASCII 类似， Unicode 是另一种为字母表中的每个字母关联一个代码的方法。由于 ASCII 不能支持大字符集（如汉语），因此 Unicode Standard 成为对语言进行编码的重要标准。Unicode 是可表示任何语言集的字符集标准。它是一种标准，旨在帮助在多种语言中进行开发。字符代码指定它表示什么字符，而标准则试图提供一种通用的方法，以编码任何一种语言中的字符。字符串可以在任何计算机系统、平台或使用的软件上显示。然后，由涉及的程序（如 Flash 或 Web 浏览器）来显示字型（其可视的外观）。

多年以来， Unicode 支持的字符数已经扩展以便为更多（和更大）的语言提供支持。字符编码被称为 “Unicode 转换格式 (UTF)”和 “通用字符集 (UCS)”，其中包括 UTF-8、UTF-16 和 UTF-32。UTF 编码中的数字代表一个单元中的位数，而 UCS 编码中的数字代表字节数。

- UTF-8 是用于交换文本的标准编码，如联机电子邮件系统。 UTF 是一个 8 位系统。
- UTF-16 通常用于内部处理。

字符串在应用程序中的长度各异。您可以确定字符串的长度，不过，此长度可能会因您所用的语言而异。而且，在字符串的末尾可能会看到终止字符，此空字符没有值。此终止字符不是实际字符，但是可用它来判断字符串何时结束。例如，如果使用套接字连接，则可以通过观察终止字符来确定字符串是否已结束（如在聊天程序中）。

硬盘上的 Samples 文件夹中有一个范例源文件 strings.fla。此文件演示了如何构建简单的字处理程序，用于比较和检索字符和子字符串选择。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Strings。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Strings。

有关字符串和 String 类的更多信息，请参见以下主题：

- 第 405 页的 “关于 “字符串” 面板”
- 第 406 页的 “使用 Locale 类”
- 第 408 页的 “使用输入方法编辑器”
- 第 410 页的 “关于 String 类”
- 第 411 页的 “创建字符串”
- 第 412 页的 “关于转义字符”
- 第 413 页的 “分析和比较字符串中的字符”
- 第 416 页的 “转换和连接字符串”
- 第 419 页的 “返回子字符串”

关于 “字符串” 面板

“字符串” 面板允许您创建和更新多语言内容。您可以为涵盖多种语言的文本字段指定内容，并让 Flash 根据运行 Flash Player 的计算机的语言自动确定应使用何种语言显示该内容。

有关 “字符串” 面板以及如何在应用程序中使用 “字符串” 面板的一般信息，请参见《使用 Flash》中的以下主题：

- 第 305 页的 “用 “字符串” 面板创作多语言文本”
- 第 308 页的 “关于编辑 “字符串” 面板中的字符串”
- 第 313 页的 “在 “字符串” 面板或 XML 文件中翻译文本”
- 第 313 页的 “将 XML 文件导入到 “字符串” 面板”

您可以使用 Locale 类控制多语言文本的显示方式。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 第 406 页的 “使用 Locale 类” 和 `Locale (mx.lang.Locale)`。

使用 Locale 类

通过 Locale 类 (mx.lang.Locale) 可控制在运行时多语言文本在 Flash 应用程序中的显示方式。通过 “字符串”面板，您可以在动态文本字段中使用字符串 ID（而不是字符串），这可允许您创建一个 SWF 文件，以显示从特定语言的 XML 文件加载的文本。您可以使用以下方法显示在 XML 本地化交换文件格式 (XLIFF) 文件中包含的特定语言的字符串。

运行时自动 Flash Player 用 XML 文件中的字符串替换字符串 ID，该文件与 `language`（`capabilities.language` 属性）返回的默认系统语言代码匹配。

手动使用舞台语言 在编译 SWF 文件时由字符串替换字符串 ID，但不能由 Flash Player 更改字符串 ID。

在运行时使用 ActionScript 可以通过使用 ActionScript 在运行时控制字符串 ID 替换。利用此选项，可以控制字符串 ID 替换的时间和语言。

如果要通过使用 ActionScript 替换字符串 ID，您可以使用 Locale 类的属性和方法在应用程序在 Flash Player 中播放时对其进行控制。有关如何使用 Locale 类的演示，请参见下面的过程。

使用 Locale 类创建多语言站点：

1. 创建一个新的 Flash 文档，并将其另存为 **locale.fla**。
2. 打开 “字符串”面板（“窗口” > “其它面板” > “字符串”），然后单击 “设置”。
3. 选择两种语言 en（英语）和 fr（法语），然后单击 “增加”，将所选语言添加到 “活动语言”面板。
4. 选择 “运行时通过 ActionScript”选项，将默认运行时语言设置为 “法语”，然后单击 “确定”。
5. 从 “组件”面板（“窗口” > “组件”）的 “User Interface”文件夹将一个组合框组件拖到舞台上，并为其指定实例名称 **lang_cb**。
6. 使用文本工具在舞台上创建一个动态文本字段，并为其指定实例名称 **greeting_txt**。
7. 在舞台上选择该文本字段，在 “字符串”面板的 “ID”文本框中键入字符串标识符 **greeting**，然后单击 “应用”。

   您将注意到 Flash 将字符串 “greeting”转换为 “IDS_GREETING”。
8. 在 “字符串”面板网格中，在 en 列中键入字符串 **hello**。
9. 在 fr 列中键入字符串 **bonjour**。

   使用 `lang_cb` 组合框更改舞台上的语言时将使用这些字符串。

10. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
import mx.lang.Locale;
Locale.setLoadCallback(localeListener);
lang_cb.dataProvider = Locale.languageCodeArray.sort();
lang_cb.addEventListener("change", langListener);
greeting_txt.autoSize = "left";
Locale.loadLanguageXML(lang_cb.value);

function langListener(eventObj:Object):Void {
  Locale.loadLanguageXML(eventObj.target.value);
}
function localeListener(success:Boolean):Void {
  if (success) {
    greeting_txt.text = Locale.loadString("IDS_GREETING");
  } else {
    greeting_txt.text = "unable to load language XML file.";
  }
}
```

上面的 ActionScript 分为两个部分。第一个部分代码导入 Locale 类并指定一个回调侦听器，只要下载完成一个语言 XML 文件就将调用该侦听器。接着，使用可用语言的排序数组填充 `lang_cb` 组合框。只要 `lang_cb` 值发生改变，Flash 的事件调度程序就会触发 `langListener()` 函数，加载特定语言的 XML 文件。第二部分代码定义了两个函数 `langListener()` 和 `localeListener()`。只要用户更改了 `lang_cb` 组合框的值，就调用第一个函数 `langListener()`。只要某个语言 XML 文件完成加载就将调用第二个函数 `localeListener()`。该函数检查加载是否成功，如果加载成功，则将 `greeting_txt` 实例的 `text` 属性设置为所选语言对应的 “greeting”。

11. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

提示	所用的 XML 文件必须使用 “XML 本地化交换文件格式” (XLIFF)。

小心	Locale 类不是 Flash Player 的一部分，因此与 《ActionScript 2.0 语言参考》中的其它类不同。由于此类安装在 Flash Authoring 类路径中，因此将自动编译到 SWF 文件中。由于 Locale 类必须编译到 SWF 文件，因此使用该类可稍微增加 SWF 文件的大小。

有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `Locale (mx.lang.Locale)`。

使用输入方法编辑器

通过输入方法编辑器 (IME)，用户可以键入亚洲语言（如中文、日语和朝鲜语）中的非 ASCII 文本字符。通过 ActionScript 中的 IME 类，您可以直接在运行在客户端计算机的 Flash Player 应用程序中对操作系统的 IME 进行操作。

使用 ActionScript 可以确定以下内容：

- 用户计算机上是否安装了 IME。
- 用户计算机上是否启用了 IME。
- 当前 IME 使用的转换模式。

IME 类可以确定当前 IME 使用的转换模式：例如，如果日语 IME 为活动状态，则可以使用 `System.IME.getConversionMode()` 方法确定其转换模式是否为平假名、片假名（等等）。可以通过 `System.IME.setConversionMode()` 方法进行设置。

注意 目前还无法分辨活动 IME（如果有），或更改 IME（例如将英语更改为日语，或将朝鲜语更改为汉语）。

根据用户的操作系统，您还可以通过在运行时使用应用程序禁用或启用 IME，并执行其它功能。可以通过使用 `System.capabilities.hasIME` 属性检查系统是否具有 IME。下一个示例演示如何确定用户是否安装并启用了 IME。

确定用户是否安装并启用了 IME：

1. 创建一个新的 Flash 文档，并将其另存为 **ime.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
if (System.capabilities.hasIME) {
  if (System.IME.getEnabled()) {
    trace("You have an IME installed and enabled.");
  } else {
    trace("You have an IME installed but not enabled.");
  }
} else {
  trace("Please install an IME and try again.");
}
```

   上面的代码首先检查当前系统是否安装了 IME。如果安装了 IME，则 Flash 将检查当前是否启用了该 IME。

3. 选择“控制”>“测试影片”来测试该文档。

   “输出”面板中将显示一条消息，表明当前是否安装并启用了 IME。

还可以在运行时在 Flash 中使用 IME 类启用和禁用 IME。下面的示例要求您系统中安装了 IME。有关在特定平台上安装 IME 的更多信息，请参见以下链接：

- www.microsoft.com/globaldev/default.mspx
- http://developer.apple.com/documentation/
- http://java.sun.com

您可以在播放 SWF 文件时启用和禁用 IME，如下面的示例所示。

在运行时启用和禁用输入方法编辑器：

1. 创建一个新的 Flash 文档，并将其另存为 **ime2.fla**。
2. 在舞台上创建两个按钮元件实例，并为其分配实例名称 **enable_btn** 和 **disable_btn**。
3. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
checkIME();

var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "_sans";

this.createTextField("ime_txt", 10, 100, 10, 320, 240);
ime_txt.border = true;
ime_txt.multiline = true;
ime_txt.setNewTextFormat(my_fmt);
ime_txt.type = "input";
ime_txt.wordWrap = true;

enable_btn.onRelease = function() {
  System.IME.setEnabled(true);
};
disable_btn.onRelease = function() {
  System.IME.setEnabled(false);
};

function checkIME():Boolean {
  if (System.capabilities.hasIME) {
    if (System.IME.getEnabled()) {
      trace("You have an IME installed and enabled.");
      return true;
    } else {
      trace("You have an IME installed but not enabled.");
      return false;
    }
  } else {
    trace("Please install an IME and try again.");
    return false;
  }
}
```

上面的代码分为五个部分。第一部分调用 checkIME() 方法，如果系统安装或启用了 IME，则该方法将在 “输出”面板中显示一条消息。第二部分定义一个自定义文本格式对象，该对象将字体设置为 _sans。第三部分创建一个输入文本字段并应用自定义文本格式。第四部分为上面步骤中创建的 enable_btn 和 disable_btn 实例创建一些事件处理函数。最后一部分，即第五部分代码定义自定义 checkIME() 函数，该函数检查当前系统是否安装了 IME，如果是，则检查该 IME 是否启用。

4. 保存该 FLA 文件，然后选择 “控制” > “测试影片”对该文档进行测试。

测试	该示例要求您系统中安装了 IME。有关安装 IME 的信息，请参见此示例前面的链接。

在舞台上的输入文本字段中键入一些文本。将 IME 切换为不同语言，并再次在此输入文本字段中键入。Flash Player 使用新 IME 输入字符。如果在舞台上单击 disable_btn 按钮，则 Flash 将还原为使用原来的语言并忽略当前的 IME 设置。

有关 System.capabilities.hasIME 的信息，请参见 《ActionScript 2.0 语言参考》中的 hasIME (capabilities.hasIME 属性)。

关于 String 类

在核心 ActionScript 语言中字符串还是一个类和一种数据类型。字符串数据类型表示 16 位字符的序列，可能包括字母、数字和标点符号。字符串存储为 Unicode 字符，使用 UTF-16 格式。对字符串值的操作返回字符串的一个新的实例。用字符串数据类型声明的变量的默认值是 null。

有关字符串、数据和值的更多信息，请参见第 10 章 “数据和数据类型”

String 类包含使您能够使用文本字符串的方法。在使用多个对象时字符串是非常重要的，而在将字符串用于多个对象（如 TextField、XML、ContextMenu 和 FileReference 实例）时本章中介绍的方法是非常有用的。

String 类是字符串基元数据类型的包装，提供用于操作基元字符串值的方法和属性。可以通过 String() 函数将任何对象的值转换为字符串。String 类的所有方法（concat()、fromCharCode()、slice() 和 substr() 除外）都是通用方法，这意味着在这些方法执行其操作前，这些方法都将调用 toString() 函数，并且可以对其它非 String 对象使用这些方法。

由于所有字符串索引都是从零开始，因此任何字符串 myStr 的最后一个字符的索引都是 myStr.length - 1。

硬盘上的 Samples 文件夹中有一个范例源文件 strings.fla。此文件演示了如何构建简单的字处理程序，用于比较和检索字符和子字符串选择。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Strings。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Strings。

创建字符串

可以使用构造函数方法 `new String()` 或者使用字符串值调用 String 类的任何方法。如果您指定了一个字符串，则 ActionScript 解释程序会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。还可以将 `String.length` 属性用于字符串。

请不要将字符串文本 和 String 对象 相混淆。有关字符串和 String 对象的更多信息，请参见第 79 页的第 4 章 “关于文本”。

在下面的示例中，此行代码将创建字符串 `firstStr`。若要声明字符串，请使用单直引号 (') 或双直引号 (") 分隔符。

创建和使用字符串：

1. 创建一个新的 Flash 文档，并将其另存为 **strings.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

   ```
   var firstStr:String = "foo";
   var secondStr:String = new String("foo");
   trace(firstStr == secondStr); // true
   var thirdStr:String;
   trace(thirdStr); // 未定义
   ```

   此代码定义 3 个 String 对象，一个使用字符串，一个使用 `new` 运算符，另一个没有初始值。通过使用等号 (==) 运算符可对字符串进行比较，如第三行代码所示。在引用变量时，只有在定义了变量时才能指定数据类型。
3. 选择 “控制” > “测试影片” 来测试该文档。

除非您确实需要使用 String 对象，否则请使用字符串。有关字符串和 String 对象的更多信息，请参见第 79 页的第 4 章 “关于文本”。

若要在单字符串中使用单直引号 (') 和双直引号 (") 分隔符，请使用反斜杠字符 (\) 来转义字符。下面的两个字符串是等效的：

```
var firstStr:String = "That's \"fine\"";
var secondStr:String = 'That\'s "fine"';
```

有关在设置中使用反斜杠字符的更多信息，请参见第 412 页的 “关于转义字符”。

请记住，不能在 ActionScript 代码中使用 “弯曲引号” 或 “特殊引号” 字符；它们与可在代码中使用的直引号 (') and (") 不同。当将来自其它源 （如 Web 或 Word 文档）的文本粘贴到 ActionScript 时，请务必使用直引号分隔符。

硬盘上的 Samples 文件夹中有一个范例源文件 strings.fla。此文件演示了如何构建简单的字处理程序，用于比较和检索字符和子字符串选择。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Strings。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Strings。

关于转义字符

可以使用反斜杠转义字符 (\) 在字符串中定义其它字符。

转义序列	说明
\b	退格符。
\f	换页符。
\n	换行符。
\r	回车。
\t	制表符。
\u*nnnn*	Unicode 字符，字符代码由十六进制数字 nnnn 指定。例如，/u263a 是一个微笑字符。
\x*nn*	ASCII 字符，字符代码由十六进制数字 nn 指定。
\'	单引号。
\"	双引号。
\\	单反斜杠字符。

有关字符串的更多信息，请参见第 79 页的第 4 章 “关于文本” 和第 411 页的 “创建字符串”。

分析和比较字符串中的字符

字符串中的每个字符在字符串中都有一个索引位置（整数）。第一个字符的索引位置为 0。例如，在字符串 `yellow` 中，字符 `y` 位置为 0，而字符 `w` 位置为 5。

每个字符串都有 `length` 属性，其值等于字符串中的字符数：

```
var companyStr:String = "macromedia";
trace(companyStr.length); // 10
```

空字符串和 null 字符串的长度都为 0：

```
var firstStr:String = new String();
trace(firstStr.length); // 0

var secondStr:String = "";
trace(secondStr.length); // 0
```

如果字符串不包含任何值，则将其长度设置为 undefined：

```
var thirdStr:String;
trace(thirdStr.length); // undefined
```

警告	如果字符串包含一个空字节字符 (\0)，则该字符串值将被截断。

还可以使用字符代码来定义字符串。有关字符代码和字符编码的更多信息，请参见第 404 页的 “关于字符串和 String 类”。

下面的示例创建一个名为 `myStr` 的变量，并基于传递到 `String.fromCharCode()` 方法的 ASCII 值设置字符串值：

```
var myStr:String =
  String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
trace(myStr); // hello world!
```

上面的代码中的 `fromCharCode()` 方法列出的每个数字表示一个字符。例如，ASCII 值 104 表示一个小写的 h，而 ASCII 值 32 表示空格字符。

还可以使用 `String.fromCharCode()` 方法转换 Unicode 值，尽管 Unicode 值必须从十六进制转换为十进制值，如下面的 ActionScript 所示：

```
// Unicode 0068 == "h"
var letter:Number = Number(new Number(0x0068).toString(10));
trace(String.fromCharCode(letter)); // h
```

可以检查字符串中各个位置上的字符，如下例所示。

循环一个字符串：

1. 创建一个新的 Flash 文档。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var myStr:String = "hello world!";
for (var i:Number = 0; i < myStr.length; i++) {
  trace(myStr.charAt(i));
}
```

3. 选择 “控制” > “测试影片” 预览 Flash 文档。您应看到输出到 “输出” 面板的每个字符都占一行。
4. 修改现有 ActionScript 代码，使其跟踪每个字符的 ASCII 值：

```
var myStr:String = "hello world!";
for (var i:Number = 0; i < myStr.length; i++) {
  trace(myStr.charAt(i) + " - ASCII=" + myStr.charCodeAt(i));
}
```

5. 保存当前的 Flash 文档，然后选择 “控制” > “测试影片” 预览该 SWF 文件。

   运行此代码时，“输出” 面板中将显示以下内容：

```
h - ASCII=104
e - ASCII=101
l - ASCII=108
l - ASCII=108
o - ASCII=111
  - ASCII=32
w - ASCII=119
o - ASCII=111
r - ASCII=114
l - ASCII=108
d - ASCII=100
! - ASCII=33
```

提示 还可以通过使用 String.split() 方法并输入一个 空字符串 （""）作为分隔符将字符串分割为字符数组；例如，`var charArray:Array = myStr.split("");`

可以使用运算符操作字符串。有关将运算符与字符串配合使用的信息，请参见第 124 页的 “关于对字符串使用运算符”。

还可以将这些运算符与条件语句 （如 `if` 和 `while`）配合使用。下面的示例将使用运算符和字符串来进行比较。

若要比较两个字符串，请执行以下操作：

1. 创建一个新的 Flash 文档，并将其另存为 **comparestr.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2) {
  trace("Uppercase letters sort first.");
}
```

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该 SWF 文件进行测试。

可以使用等号 (==) 和不等号 (!=) 运算符将字符串与其它类型的对象进行比较，如下例所示。

将字符串与其它数据类型比较：

1. 创建一个新的 Flash 文档，并将其另存为 **comparenum.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var myStr:String = "4";
var total:Number = 4;
if (myStr == total) {
  trace("Types are converted.");
}
```

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该 SWF 文件进行测试。

   比较两个不同的数据类型时 （如字符串和数字）， Flash 将尝试转换数据类型以便进行比较。

可以使用全等 (===) 和全不等 (!==) 运算符来检查这两个比较对象是否属于同一类型：下面的示例使用严格比较运算符，确保 Flash 在尝试比较值时不会尝试转换数据类型。

强制严格数据类型比较：

1. 创建一个新的 Flash 文档，并将其另存为 **comparestrict.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var str1:String = "4";
var str2:String = "5";
var total:Number = 4;
if (str1 !== total) {
  trace("Types are not converted.");
}
if (str1 !== str2) {
  trace("Same type, but the strings don't match.");
}
```

3. 保存 Flash 文档，并选择 “控制” > “测试影片”。

有关将运算符与字符串配合使用的更多信息，请参见第124页的 “关于对字符串使用运算符”。

硬盘上的 Samples 文件夹中有一个范例源文件 strings.fla。此文件演示了如何构建简单的字处理程序，用于比较和检索字符和子字符串选择。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Strings。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Strings。

转换和连接字符串

使用 `toString()` 方法可以将许多对象转换为字符串。大多数内置对象都有一个 `toString()` 方法用于此目的：

```
var n:Number = 0.470;
trace(typeof(n.toString())); // 字符串
```

当将加法 (+) 运算符用于字符串和非字符串实例的组合时，不需要使用 `toString()` 方法。有关连接的详细信息，请参见本部分中的第二个过程。

`toLowerCase()` 方法和 `toUpperCase()` 方法可将字符串中的英文字母转换为相应的小写和大写。下面的示例演示了如何将字符串从小写字符转换为大写字符。

将字符串从小写转换为大写：

1. 创建一个新的 Flash 文档，并将其另存为 **convert.fla**。
2. 将下面的代码输入到时间轴中的第 1 帧上：

```
var myStr:String = "Dr. Bob Roberts, #9.";
trace(myStr.toLowerCase()); // dr. bob roberts, #9.
trace(myStr.toUpperCase()); // DR. BOB ROBERTS, #9.
trace(myStr); // Dr. Bob Roberts, #9.
```

3. 保存 Flash 文档并选择 “控制” > “测试影片”。

提示 执行完这些方法后，源字符串仍保持不变。要转换源字符串，请使用下列方法：

```
myStr = myStr.toUpperCase();
```

连接字符串时，要将两个字符串按顺序连接成一个字符串。例如，可以使用加法 (+) 运算符来连接两个字符串。下面的示例说明了如何连接两个字符串。

连接两个字符串：

1. 创建一个新的 Flash 文档，并将其另存为 **concat.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
var str1:String = "green";
var str2:String = str1 + "ish";
trace(str2); // greenish
//
var str3:String = "yellow";
str3 += "ish";
trace(str3); // yellowish
```

   上面的代码演示了两个连接字符串的方法。第一个方法使用加法 (+) 运算符连接 `str1` 字符串与 `"ish"` 字符串。第二个方法使用加法赋值 (+=) 运算符将字符串 `"ish"` 与 `str3` 的当前值连接起来。

3. 保存 Flash 文档，并选择 “控制” > “测试影片”。

还可以使用 String 类的 `concat()` 方法来连接字符串。下面的示例中演示了此方法。

使用 concat() 方法连接两个字符串：

1. 创建一个新的 Flash 文档，并将其另存为 **concat2.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
trace(str4); // Bonjour from Paris
```

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

如果使用加法 (+) 运算符（或加法赋值 (+=) 运算符）连接一个字符串和一个非字符串对象，那么，为了计算表达式的值，ActionScript 会自动将非字符串对象转换为字符串。下面的代码示例演示了此转换过程：

```
var version:String = "Flash Player ";
var rel:Number = 8;
version = version + rel;
trace(version); // Flash Player 8
```

不过，可以使用小括号强制加法 (+) 运算符进行算术计算，如下面的 ActionScript 代码所示：

```
trace("Total: $" + 4.55 + 1.46); // 总计: $4.551.46
trace("Total: $" + (4.55 + 1.46)); // 总计: $6.01
```

可以使用 `split()` 方法来创建字符串的子字符串数组，该数组由分隔符分隔。例如，可以将逗号分隔或 tab 分隔的字符串分为多个字符串。

例如，下面的代码演示了如何通过将 “and” (`&`) 字符用作分隔符将一个数组分割为多个子字符串。

创建以分隔符隔开的子字符串数组：

1. 创建一个新的 Flash 文档，并将其另存为 **strsplit.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var queryStr:String = "first=joe&last=cheng&title=manager&startDate=3/6/
  65";
var params:Array = queryStr.split("&", 2);
trace(params); // first=joe,last=cheng
    /* params is set to an array with two elements:
       params[0] == "first=joe"
       params[1] == "last=cheng"
    */
```

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

提示 split() 方法的第二个参数定义数组的最大大小。如果不希望限制由 split() 方法创建的数组的大小，则可以忽略第二个参数。

提示 分析查询字符串 （由 & 和 = 字符分隔的字符串）的最简单方法就是使用 LoadVars.decode() 方法，如下面的 ActionScript 所示：

```
var queryStr:String = "first=joe&last=cheng&title=manager&startDate=3/6/
  65";
var my_lv:LoadVars = new LoadVars();
my_lv.decode(queryStr);
trace(my_lv.first); // joe
```

有关将运算符与字符串配合使用的更多信息，请参见第124页的 “关于对字符串使用运算符”。

硬盘上的 Samples 文件夹中有一个范例源文件 strings.fla。此文件演示了如何构建简单的字处理程序，用于比较和检索字符和子字符串选择。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Strings。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Strings。

返回子字符串

String 类的 substr() 和 substring() 方法非常相似。都返回字符串的子字符串并且都有两个参数。在这两个方法中，第一个参数是给定字符串中起始字符的位置。不过，在 substr() 方法中，第二个参数是要返回的子字符串的长度，而在 substring() 方法中，第二个参数是子字符串的末尾 字符的位置（未包含在返回的字符串中）。此示例显示了这两种方法之间的差别：

通过字符位置查找子字符串：

1. 创建一个新的 Flash 文档，并将其另存为 **substring.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var myStr:String = "Hello from Paris, Texas!!!";
trace(myStr.substr(11,15)); // Paris, Texas!!!
trace(myStr.substring(11,15)); // Pari
```

   第一个方法 substr() 返回一个长为 15 个字符的字符串，该字符串从第 11 个字符开始。第二个方法 substring() 通过获取第 11 和第 14 索引之间的所有字符返回一个长为 4 个字符的字符串。

3. 在上一步骤中添加的代码下面添加以下 ActionScript：

```
trace(myStr.slice(11, 15)); // Pari
trace(myStr.slice(-3, -1)); // !!
trace(myStr.slice(-3, 26)); // !!!
trace(myStr.slice(-3, myStr.length)); // !!!
trace(myStr.slice(-8, -3)); // Texas
```

   slice() 方法与 substring() 方法的运行方式类似。当指定两个非负整数作为参数时，其运行方式将完全一样。然而， slice() 方法可使用负整数作为其参数。

4. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

注意	可以结合使用非负整数和负整数作为 slice() 方法的参数。

可以使用 indexOf() 和 lastIndexOf() 方法在字符串内查找匹配的子字符串，如下例所示。

查找匹配的子字符串的字符位置：

1. 创建一个新的 Flash 文档，并将其另存为 **indexof.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var myStr:String = "The moon, the stars, the sea, the land";
trace(myStr.indexOf("the")); // 10
trace(myStr.indexOf("the", 11)); // 21
```

   the 的第 1 个索引从第 10 个字符开始，因为 `indexOf()` 方法区分大小写；因此不考虑 The 的第一个实例。还可以指定 `indexOf()` 的第二个参数，指示从字符串中的哪个索引位置开始搜索。在上面的代码中，Flash 搜索第 11 个字符后出现的 the 的第一个索引。

3. 在上一步骤中添加的代码下面添加以下 ActionScript：

```
trace(myStr.lastIndexOf("the")); // 30
trace(myStr.lastIndexOf("the", 29)); // 21
```

   `lastIndexOf()` 方法查找字符串中的最后一个子字符串匹配项。例如，`lastIndexOf()` 不是从字符串开始处搜索字符或子字符串，而是末尾处开始反向搜索。与 `indexOf()` 方法类似，如果在 `lastIndexOf()` 方法中包含第二个参数，则将从该索引位置执行搜索，尽管 `lastIndexOf()` 是反向搜索字符串 （从右到左）。

4. 选择 “控制” > “测试影片” 对 Flash 文档进行测试。

提示	`indexOf()` 和 `lastIndexOf()` 方法区分大小写。

硬盘上的 Samples 文件夹中有一个范例源文件 strings.fla。此文件演示了如何构建简单的字处理程序，用于比较和检索字符和子字符串选择。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Strings。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Strings。

第 13 章

13 动画、滤镜和绘画

本章介绍如何使用 ActionScript 替代（或作为后者的补充）基于时间轴的动画（该动画使用补间动画或补间形状）向 Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 应用程序中添加动画。使用代码创建动画和特效通常可以减小完成的应用程序的文件大小，还可以提高动画本身的性能和一致性。有时，基于 ActionScript 的动画甚至可以降低工作负载: 编写代码更快，并且可以轻松地将代码一次应用于很多实例或在其它应用程序中重用。本章还演示了如何使用基础的 ActionScript 基本功能（Tween 和 TransitionManager 类、Drawing API、滤镜类和混合模式）来创建动画。

可以使用 Drawing API 来添加动画和绘画，该 API 由 MovieClip 类中的绘画方法组成。通过这些方法可以使用代码创建线条、填充和形状，而无需使用创作工具中的绘画工具。

在许多 Flash 应用程序中，滤镜和其它表达效果对于快速应用效果和创建动画也很重要。可以使用代码来添加滤镜效果、混合模式和位图图像，并为其实现动画效果。

本章包含以下几节，介绍了如何使用 ActionScript 创建动画和添加效果，以及在 ActionScript 中如何使用 Drawing API 来进行绘制:

使用 ActionScript 2.0 编写动画脚本

可以使用 ActionScript 2.0 向 Flash 应用程序中添加动画，而不使用时间轴上的补间动画或补间形状。下面各节说明了如何使用代码为实例实现动画效果，例如更改实例的透明度和外观，以及沿舞台移动实例等。

有关使用 Tween 和 TransitionManager 类对基于代码的动画进一步实现自动化的信息，请参见第 1139 页的 “TransitionManager 类”和第 1207 页的 “Tween 类”。这些类可帮助您向应用程序的影片剪辑实例中添加高级缓动方程和转变动画。在没有这些预生成类的情况下，使用 ActionScript 很难重新创建其中的许多效果，因为需要使用的代码会涉及到编写复杂的数学方程来获得效果。

有关如何为使用代码创建的绘画添加动画效果的更多信息，请参见第 482 页的 “使用 ActionScript 绘画”。

下面各节介绍了如何编写动画脚本：

有关 Flash 中的脚本动画的示例，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

可在硬盘上找到图库应用的范例。这些文件提供了相关示例，演示在向 SWF 文件中加载图像文件（其中包含脚本动画）时如何使用 ActionScript 动态地控制影片剪辑。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

关于动画和帧频

在向应用程序中添加动画时，需要考虑为 FLA 文件设置的帧频。使用动画时需要考虑帧频，因为帧频会影响 SWF 文件以及播放该文件的计算机的性能。将帧频设置得过高会导致处理器问题，特别是在使用许多资源或使用 ActionScript 创建动画时。

但是，还需要考虑帧频设置，因为该设置会影响播放动画的平滑程度。例如，在属性检查器中将动画设置为 12 帧 / 秒 (fps) 时，则该动画将每秒播放 12 帧。如果文档的帧频设置为 24 fps，则动画运行时将比 12 fps 时更为平滑。但是，设置为 24 fps 时，动画的播放速度要比 12 fps 时快得多，所以总持续时间 （秒）较短。因此，如果使用较高的帧频制作 5 秒的动画，则意味着与较低的帧频相比，需要添加更多的帧来填充这五秒动画（因此，这将使动画的总文件大小增加）。帧频为 24 fps 的 5 秒动画的文件大小通常比帧频为 12 fps 的 5 秒动画的文件大。

附加

当您使用 `onEnterFrame` 事件处理函数创建脚本动画时，该动画将以文档的帧频运行，与在时间轴上创建补间动画时相似。`onEnterFrame` 事件处理函数的备选函数是 `setInterval` （请参见《ActionScript 2.0 语言参考》中的 setInterval 函数）。不依赖于帧频，而以指定的间隔调用函数。与 `onEnterFrame` 类似，越频繁使用 `setInterval` 进行函数调用，动画所需的处理器资源就越密集。

使用可使动画在运行时平滑播放的最低可能帧频，有助于减少对最终用户的处理器的压力。尽量不要使用 30 到 40 fps 以上的帧频；帧频高时将给处理器施加很大压力，在运行时也不要过多更改动画外观，或干脆根本不更改动画外观。

而且，特别是在使用基于时间轴的动画时，请在开发过程中尽早为动画选择帧频。测试 SWF 文件时，请检查动画的持续时间以及 SWF 文件大小。帧频会对动画的速度产生极大的影响。

使用代码淡化对象

在舞台上使用影片剪辑时，您可能希望淡入淡出影片剪辑，而不是切换其 `_visible` 属性。下面的过程将演示如何使用 `onEnterFrame` 事件处理函数为影片剪辑创建动画。

通过使用代码淡化影片剪辑：

1. 创建一个名为 **fade1.fla** 的新 Flash 文档。
2. 使用绘画工具在舞台上绘制一些图形，或将图像导入舞台（“文件” > “导入” > “导入到舞台”）。
3. 在舞台上选择内容，并选择 “修改” > “转换为元件”。
4. 选择 “影片剪辑”选项并单击 “确定”，以创建元件。
5. 在舞台上选择影片剪辑实例，并在属性检查器的“实例名称”文本框中键入 **img1_mc**。

6. 在时间轴中选择第 1 帧，在 “动作”面板中添加下面的代码：

```
img1_mc.onEnterFrame = function() {
  img1_mc._alpha -= 5;
  if (img1_mc._alpha <= 0) {
    img1_mc._visible = false;
    delete img1_mc.onEnterFrame;
  }
};
```

此代码使用 onEnterFrame 事件处理函数，它可以按 SWF 文件的帧频反复调用。每秒钟调用该事件处理函数的次数取决于设置的 Flash 文档的帧频。如果帧频是 12 帧 / 秒 (fps)，则 onEnterFrame 事件处理函数每秒被调用 12 次。 同样，如果 Flash 文档的帧频是 30 fps，则事件处理函数每秒被调用 30 次。

7. 选择 “控制” > “测试影片”来测试该文档。

添加到舞台的影片剪辑将慢慢淡出。

可以通过使用 setInterval() 函数而不是 onEnterFrame 事件处理函数来修改 _alpha 属性，如下面的过程所示。

通过使用 setInterval() 函数淡化对象：

1. 创建一个名为 **fade2.fla** 的新 Flash 文档。
2. 在舞台上绘制一些图形，或将图像导入舞台 （“文件” > “导入” > “导入到舞台”）。
3. 在舞台上选择内容，并选择 “修改” > “转换为元件”。
4. 选择 “影片剪辑”选项并单击 “确定”，以创建元件。
5. 在 “舞台”上选择影片剪辑实例，并在属性检查器的 “实例名称”文本框中键入 **img1_mc**。
6. 在时间轴中选择第 1 帧，在 “动作”面板中添加下面的代码：

```
var alpha_interval:Number = setInterval(fadeImage, 50, img1_mc);
function fadeImage(target_mc:MovieClip):Void {
  target_mc._alpha -= 5;
  if (target_mc._alpha <= 0) {
    target_mc._visible = false;
    clearInterval(alpha_interval);
  }
}
```

setInterval() 函数的工作方式与 onEnterFrame 事件处理函数略有不同，因为 setInterval() 函数告知 Flash 代码应调用特定函数的准确频率。在此代码示例中，用户定义的 fadeImage() 函数每 50 毫秒被调用一次 （每秒 20 次）。 fadeImage() 函数将递减当前影片剪辑的 _alpha 属性的值。当 _alpha 值等于或小于 0 时，将清除间隔，从而引起 fadeImage() 函数停止执行。

7. 选择 “控制” > “测试影片” 来测试该文档。

添加到舞台的影片剪辑将慢慢淡出。

有关用户定义的函数的更多信息，请参见第 150 页的 “定义全局函数和时间轴函数”。有关 onEnterFrame 事件处理函数的更多信息，请参见 《ActionScript 2.0 语言参考》中的 onEnterFrame (MovieClip.onEnterFrame 处理函数)。有关 setInterval() 函数的更多信息，请参见 《ActionScript 2.0 语言参考》中的 setInterval 函数。

有关 Flash 中的脚本动画的示例，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

使用代码添加颜色和亮度效果

除了使用 ActionScript 设置 Alpha 淡化并为其添加动画外 （请参见第 423 页的 “使用代码淡化对象”），还可以使用代码 （而不是使用属性检查器中的 “滤镜” 面板）来为各种颜色和亮度效果实现动画。

下面的过程将加载 JPEG 图像并应用颜色转换滤镜，当鼠标指针沿 x 轴和 y 轴移动时，此滤镜将修改红色和绿色通道。

通过使用 ActionScript 更改对象的颜色通道：

1. 创建一个名为 **colorTrans.fla** 的新 Flash 文档。
2. 在时间轴中选择第 1 帧，在 “动作” 面板中添加下面的代码：

```
import flash.geom.Transform;
import flash.geom.ColorTransform;

var imageClip:MovieClip = this.createEmptyMovieClip("imageClip", 1);
var clipLoader:MovieClipLoader = new MovieClipLoader();
clipLoader.loadClip("http://www.helpexamples.com/flash/images/
  image1.jpg", imageClip);

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  var transformer:Transform = new Transform(imageClip);
  var colorTransformer:ColorTransform = transformer.colorTransform;
  colorTransformer.redMultiplier = (_xmouse / Stage.width) * 1;
  colorTransformer.greenMultiplier = (_ymouse / Stage.height) * 1;
  transformer.colorTransform = colorTransformer;
}
Mouse.addListener(mouseListener);
```

3. 选择 “控制” > “测试影片” 来测试文档，然后沿着舞台移动鼠标指针。

   移动鼠标时，加载的图像文件将随之转变颜色。

还可以使用 ColorMatrixFilter 类将彩色图像转换成黑白图像，如下面的过程所示。

使用 ColorMatrixFilter 类将图像更改为灰度图像：

1. 创建一个名为 **grayscale.fla** 的新 Flash 文档。
2. 在时间轴中选择第 1 帧，在 “动作” 面板中添加下面的代码：

```
import flash.filters.ColorMatrixFilter;
System.security.allowDomain("http://www.helpexamples.com");
var mcl_obj:Object = new Object();
mcl_obj.onLoadInit = function(target_mc:MovieClip):Void {
  var myElements_array:Array = [0.3, 0.59, 0.11, 0, 0,
       0.3, 0.59, 0.11, 0, 0,
       0.3, 0.59, 0.11, 0, 0,
       0, 0, 0, 1, 0];
  var myColorMatrix_filter:ColorMatrixFilter = new
  ColorMatrixFilter(myElements_array);
  target_mc.filters = [myColorMatrix_filter];
}
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mcl_obj);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

   前面的代码首先导入 ColorMatrixFilter 类，并创建一个侦听器对象，在后面的代码中，会将该对象与创建的新 MovieClipLoader 实例一起使用。接下来，将创建一个实例名称为 `img_mc` 的新影片剪辑实例，以及一个实例名称为 `img_mcl` 的新影片剪辑加载器实例。最后，源影片剪辑将加载到舞台上的 `img_mc` 影片剪辑中。成功加载图像时，将调用 onLoadInit 事件处理函数，并且会将一个 ColorMatrixFilter 附加到加载的图像中。

3. 选择 “控制” > “测试影片” 来测试该文档。

   您加载到舞台上的图像将更改为灰度图像。联机查看图像 (http://www.helpexamples.com/flash/images/image1.jpg) 以便查看图像的原始颜色。

也可以通过使用下面过程中的 ActionScript 代码来设置图像的亮度。

更改图像的亮度：

1. 创建一个名为 **brightness.fla** 的新 Flash 文档。
2. 在时间轴中选择第 1 帧，在 “动作”面板中添加下面代码：

```
import flash.filters.ColorMatrixFilter;
System.security.allowDomain("http://www.helpexamples.com/");
var mcl_obj:Object = new Object();
mcl_obj.onLoadInit = function(target_mc:MovieClip):Void {
  var myElements_array:Array = [1, 0, 0, 0, 100,
       0, 1, 0, 0, 100,
       0, 0, 1, 0, 100,
       0, 0, 0, 1, 0];
  var myColorMatrix_filter:ColorMatrixFilter = new
  ColorMatrixFilter(myElements_array);
  target_mc.filters = [myColorMatrix_filter];
}
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mcl_obj);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image2.jpg",
  img_mc);
```

   此代码块使用 MovieClipLoader 类来加载外部 JPEG。当成功加载图像时，将调用 MovieClipLoader 类的 onLoadInit 事件处理函数，并使用 ColorMatrixFilter 滤镜将图像亮度修改为 100。

3. 选择 “控制” > “测试影片”来测试该文档。

   当您对 SWF 文件进行测试时，加载到该 SWF 文件中的图像将改变其亮度。联机查看图像 (http://www.helpexamples.com/flash/images/image2.jpg) 以便查看图像的原始外观。

有关 Flash 中的脚本动画的示例，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

还可以在硬盘上找到图库应用的范例。这些文件提供了相关示例，演示在向 SWF 文件中加载图像文件 （其中包含脚本动画）时如何使用 ActionScript 动态地控制影片剪辑。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

使用代码移动对象

使用 ActionScript 移动对象与修改对象的 _alpha 属性相似，不同之处在于移动对象时修改的是 _x 或 _y 属性。

下面的过程可为动态加载的 JPEG 图像添加动画效果，并使其水平滑过舞台。

通过使用代码在舞台上移动实例：

1. 创建一个名为 **moveClip.fla** 的新 Flash 文档。
2. 在属性检查器中，将文档的帧频更改为 **24** fps。

   如果使用较高的帧频（例如 24 fps），动画要平滑得多。
3. 在时间轴中选择第 1 帧，在“动作”面板中添加下面的代码：

```
// 创建影片剪辑实例。
this.createEmptyMovieClip("img1_mc", 10);
var mcl_obj:Object = new Object();
mcl_obj.onLoadInit = function (target_mc:MovieClip):Void {
  target_mc._x = Stage.width;
  target_mc.onEnterFrame = function() {
    target_mc._x -= 3; // 将当前的 _x 位置减小 3 个像素
    if (target_mc._x <= 0) {
      target_mc._x = 0;
      delete target_mc.onEnterFrame;
    }
  };
};
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mcl_obj);
// 将图像加载到影片剪辑中
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img1_mc);
```

   此代码示例从远程 Web 服务器加载了一个外部图像，并且一旦图像完全加载，就让它沿舞台水平移动。可以不使用 onEnterFrame，而是使用 setInterval() 函数让图像动起来。
4. 选择“控制”>“测试影片”来测试该文档。

   将加载图像，然后从舞台的右侧向舞台的左上角移动。

有关使用 onEnterFrame 事件处理函数或 setInterval() 函数实现图像移动的信息，请参见第 423 页的“使用代码淡化对象”。

有关 Flash 中的脚本动画的示例，可以在硬盘上的 Samples 文件夹中找到范例源文件 animation.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Animation。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Animation。

可在硬盘上找到图库应用的范例。这些文件提供了相关示例，演示在向 SWF 文件中加载图像文件（其中包含脚本动画）时如何使用 ActionScript 动态地控制影片剪辑。在硬盘上的 Samples 文件夹中可以找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

使用代码平移图像

使用 ActionScript 可以在 Flash 文档内轻松平移大型图像。当您的图像不适合舞台大小时，或者您要创建一个动画效果并在其中从一侧向舞台的另一侧平移影片剪辑时，这一功能很有用。例如，假如您有一个比舞台尺寸更大的大型全景图像，而且不想减小图像尺寸或增大舞台的大小，则可以创建一个影片剪辑，将其作为较大图像的遮罩。

下面的过程演示如何动态遮罩影片剪辑并使用 `onEnterFrame` 事件处理函数使图像在遮罩后面移动。

使用代码在舞台上平移实例：

1. 创建一个名为 **pan.fla** 的新 Flash 文档。
2. 在属性检查器中，将文档的帧频更改为 **24** fps。

   如果使用较高的帧频（例如 24 fps），动画要平滑得多。
3. 在时间轴中选择第 1 帧，在“动作”面板中添加下面的代码。

```
System.security.allowDomain("http://www.helpexamples.com/");
// 初始化变量
var direction:Number = -1;
var speed:Number = 5;
// 创建要向其中加载图像的剪辑
this.createEmptyMovieClip("img_mc", 10);
// 创建要用作遮罩的剪辑
this.createEmptyMovieClip("mask_mc", 20);
// 使用 Drawing API 绘制 / 创建遮罩
with (mask_mc) {
  beginFill(0xFF0000, 0);
  moveTo(0, 0);
  lineTo(300, 0);
  lineTo(300, 100);
  lineTo(0, 100);
  lineTo(0, 0);
  endFill();
}

var mcl_obj:Object = new Object();
```

```
mcl_obj.onLoadInit = function(target_mc:MovieClip) {
  // 将目标影片剪辑的遮罩设置为 mask_mc
  target_mc.setMask(mask_mc);
  target_mc.onEnterFrame = function() {
    target_mc._x += speed * direction;
    // 如果 target_mc 位于边缘处，则翻转动画方向
    if ((target_mc._x <= -(target_mc._width-mask_mc._width)) ||
  (target_mc._x >= 0)) {
      direction *= -1;
    }
  };
};

var my_mcl:MovieClipLoader = new MovieClipLoader();
my_mcl.addListener(mcl_obj);
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

在此代码示例中，第一部分代码定义了两个变量: direction 和 speed。direction 变量控制被遮罩的图像是从左向右 (1) 滚动还是从右向左 (-1) 滚动。 speed 变量控制每次调用 onEnterFrame 事件处理函数时移动的像素的数量。数量越大，动画移动得越快，不过动画就会显得有点欠平滑。

代码的下一部分创建两个空的影片剪辑: img_mc 和 mask_mc。使用 Drawing API 在 mark_mc 影片剪辑内绘制 300x100 像素的矩形。接下来，将创建一个新对象 (mcl_obj)，可将其用作最后的代码块中创建的 MovieClipLoader 实例的侦听器。此对象定义了一个用于 onLoadInit 事件的事件侦听器，并将遮罩动态加载的图像和设置滚动动画。在图像到达遮罩的左边缘或右边缘后，动画将翻转。

最后的代码块定义了一个 MovieClipLoader 实例，指定了前面创建的侦听器对象，并开始将 JPEG 图像加载到 img_mc 影片剪辑中。

4. 选择 “控制” > “测试影片” 来测试该文档。

图像将加载，然后在平移动画 （端到端动画）中来回动起来。图像在运行时将被遮罩。若要查看原始图像，可以联机进行查看 (http://www.helpexamples.com/flash/images/image1.jpg)。

关于位图缓存、滚动和性能

Flash Player 8 引入了位图缓存，这有助于增强应用程序中不会更改的影片剪辑的性能。在将 `MovieClip.cacheAsBitmap` 或 `Button.cacheAsBitmap` 属性设置为 `true` 时，Flash Player 将缓存影片剪辑或按钮实例的内部位图表示形式。这可以提高包含复杂矢量内容的影片剪辑的性能。具有已缓存位图的影片剪辑的所有矢量数据都会被绘制到位图而不是主舞台上。

注意	然后，将位图复制到主舞台上，作为对齐到最接近像素边界的未拉伸、未旋转的像素。像素与父对象进行一对一映射。如果位图的边界发生更改，则将重新创建位图而不会拉伸它。

有关缓存按钮或影片剪辑实例的详细信息，请参见第 11 章“使用影片剪辑”中的以下部分:

- 第 329 页的 “关于使用 ActionScript 缓存和滚动影片剪辑”
- 第 333 页的 “缓存影片剪辑”
- 第 335 页的 “设置影片剪辑的背景”

最好将 `cacheAsBitmap` 属性与主要包含静态内容且不频繁缩放和旋转的影片剪辑一起使用。对于这样的影片剪辑，使用 `cacheAsBitmap` 属性可在转换影片剪辑时（当 x 和 y 位置更改时）提高性能。有关何时使用此功能的详细信息，请参见第 331 页的“何时启用缓存”。

提供了一个范例源文件，该文件演示如何将位图缓存应用于实例。在硬盘上的 Samples 文件夹中找到名为 cacheBitmap.fla 的文件。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\CacheBitmap。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/CacheBitmap。

提供了一个范例源文件，该源文件演示如何将位图缓存应用于滚动文本。在硬盘上的 Samples 文件夹中找到范例源文件 flashtype.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FlashType。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FlashType。

关于 Tween 类和 TransitionManager 类

安装 Flash Basic 8 或 Flash Professional 8 后，还安装了两个功能强大的类：Tween 类和 TransitionManager 类。本节介绍如何对影片剪辑和 Macromedia V2 组件（包括在 Flash MX 2004 和 Flash 8 中）使用这些类，以便于轻松地向 SWF 文件中添加动画。

如果使用 Flash Professional 8（仅 ActionScript 2.0）创建幻灯片演示文稿或表单应用程序，可以选择在幻灯片间添加不同转变的行为，这类似于创建 PowerPoint 演示文稿时的情况。使用 Tween 和 TransitionManager 类将此功能添加到屏幕应用程序，可以生成根据所选行为让屏幕动起来的 ActionScript 代码。

在 Flash Basic 8 或 Flash Professional 8 中，除了将 Tween 和 TransitionManager 类用于基于屏幕的文档之外，还可以在别处使用。例如，可以将这些类用于 Macromedia Component Architecture 第 2 版的组件集，或用于影片剪辑。如果您希望改变 ComboBox 组件的动画方式，则可以使用 TransitionManager 类在菜单打开时添加一些缓动。“缓动”是动画运行期间的渐进加速和渐进减速效果，有助于使动画显得更逼真。您还可以使用 Tween 和 TransitionManager 类来创建自己的动画菜单系统，而不使用时间轴上的补间动画或通过编写自定义代码来创建。

注意	Tween 类和 TransitionManager 类只在 ActionScript 2.0 中提供，但 Flash Basic 8 和 Flash Professional 8 中均可以使用这些类。

有关 Tween 类的每个方法和属性的信息，请参见《组件语言参考》中的第 51 章“Tween 类”。有关 TransitionManager 类的每个方法和属性的信息，请参见《组件语言参考》中的第 48 章“TransitionManager 类”。有关使用包的信息，请参见第 447 页的“使用滤镜包”。

提供了使用这些类添加脚本动画的范例源文件。在硬盘上的 Samples 文件夹中找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

有关 Tween 和 TransitionManager 类的更多信息，请参见以下主题：

- 第 433 页的“在 Flash Professional 8 中向文件添加补间和转变（仅限 Flash Professional 8）”
- 第 434 页的“使用 TransitionManager 和 Tween 类添加动画”
- 第 437 页的“关于缓动类和方法”
- 第 438 页的“关于 Tween 类”
- 第 439 页的“使用 Tween 类”
- 第 444 页的“将 TransitionManager 与 Tween 类结合使用”

在 Flash Professional 8 中向文件添加补间和转变（仅限 Flash Professional 8）

注意

本节介绍如何向 Flash Professional 幻灯片演示文稿中添加补间和转变，以向 Flash Professional 用户演示其效果。不过，如果您使用代码，则可以向 Flash Basic 8（或 Flash Professional 8）应用程序中添加转变和补间。下面各部分包含了可以演示如何实现的示例。

Tween 和 TransitionManager 类旨在让您使用简单的 ActionScript 将动画添加到 SWF 文件的某些部分中。 Flash 创作环境中包含一些行为，您可以使用这些预置的类为基于屏幕的应用程序添加转变效果。要创建一个幻灯片演示文稿或表单应用程序，您可以选择在幻灯片之间加入不同的转变的行为。

在开始将这些转变用于 Flash 中的影片剪辑之前，您应了解它们在使用基于屏幕的应用程序时所起的作用。

查看在幻灯片演示文稿中创建转变的 ActionScript：

1. 选择 “文件” > “新建”，在 Flash Professional 8 中创建一个新的幻灯片演示文稿。
2. 从 “常规”选项卡中选择 “Flash 幻灯片演示文稿”，然后单击 “确定”。
3. 选择 “窗口” > “行为”以打开 “行为”面板。
4. 单击 “添加行为” (+)。
5. 从弹出菜单中选择 “屏幕” > “转变”打开 “转变”对话框。
6. 选择 “缩放”转变。
7. 在 “持续时间”文本框中键入 **1**。
8. 从 “放松”弹出菜单中选择 “回弹”。
9. 单击 “确定”应用设置并关闭该对话框。

此操作完成后就会有大约 15 行的 ActionScript 直接添加到幻灯片上。下面的代码片断显示了相关的转变代码：

```
mx.transitions.TransitionManager.start(eventObj.target,
  {type:mx.transitions.Zoom, direction:0, duration:1,
  easing:mx.transitions.easing.Bounce.easeOut, param1:empty,
  param2:empty});
```

此代码调用 TransitionManager 类，然后用指定的 `mx.transitions.easing.Bounce.easeOut` 缓动方法应用缩放转变。在本例中，转变应用于选定的幻灯片。若要将这种效果应用于影片剪辑，可以修改在 Flash 动画中使用的 ActionScript。修改代码以使用影片剪辑元件非常易于实现：将 `eventObj.target` 的第一个参数更改为期望的影片剪辑的实例名称即可。

Flash 提供了十种过渡形式，可以通过使用缓动方法和若干可选的参数进行自定义。“缓动”是动画运行期间的渐进加速和渐进减速效果，有助于使动画显得更逼真。例如，一个球在刚开始运动阶段是以加速形式运动的，在接近停止到完全停止阶段是以减速形式运动的。关于加速和减速有许多公式，相应地也就有多种缓动方法。

下表介绍了 Flash Basic 8（使用代码）和 Flash Professional 8（使用代码或行为）中包括的过渡：

转变	说明
光圈	使形状的遮罩以动画形式放大，以显示出屏幕或影片剪辑。
划入 / 划出	使形状的遮罩以动画形式向水平方向运动，显示出屏幕或影片剪辑。
像素溶解	使用消失或出现的矩形遮罩屏幕或影片剪辑。
遮帘	使用消失或出现的矩形显示下一个屏幕或影片剪辑。
淡入 / 淡出	淡入或淡出屏幕或影片剪辑。
飞翔	屏幕或影片剪辑从不同的方向滑入。
缩放	放大或缩小屏幕或影片剪辑。
挤压	水平或垂直缩放当前屏幕或影片剪辑。
旋转	旋转当前屏幕或影片剪辑。
照片	屏幕或影片剪辑像拍摄照片时那样闪烁。

每种转变都有稍微不同的自定义，可以应用到您的动画中。“转变”对话框允许您在将效果应用于幻灯片或表单之前预览范例动画。

提示

若要预览每个转变如何与缓动类中的不同方法一起使用，可以双击 boot drive\Program Files\Macromedia\Flash 8\ 语言 \First Run\Behaviors\ 文件夹或 Macintosh HD:Applications:Macromedia Flash 8:First Run:Behaviors: 中的 Transition.swf，以在独立播放器中打开 SWF 文件。

使用 TransitionManager 和 Tween 类添加动画

使用 ActionScript，可以在 Flash Basic 8 和 Flash Professional 8 中使用 TransitionManager 和 Tween 类将动画加入到影片剪辑、组件和帧上。如果不使用 TransitionManager 或 Tween 类，那您就需要编写自定义代码来使您的影片剪辑具有动画效果，或修改它们的透明度 (Alpha) 和坐标 （位置）。如果您想在动画中加入缓动方法，则 ActionScript （和数学计算）就会很快变得复杂起来。然而，如果您想改变特定动画中的缓动方法，并且使用了这些预置的类，那么您可以选择不同的类，而不是想办法通过各种数学公式来创建一个平滑的动画。

下面的过程为一个影片剪辑添加动画效果，以便其使用 TransitionManager 类在舞台上放大。

使用 TransitionManager 类为影片剪辑添加动画效果：

1. 选择 “文件” > “新建”，然后选择 “Flash 文档”。
2. 单击 “确定”创建一个新的 FLA 文件。
3. 将该 FLA 文件保存为 **zoom.fla**。
4. 选择 “文件” > “导入” > “导入到舞台”，然后在硬盘上选择要导入到 FLA 文件中的图像。

   该图像是以位图图像的格式导入到文件中，因此需要手动将该图像转换为影片剪辑元件。
5. 单击 “打开”以导入图像。
6. 在舞台上选择导入的图像，然后选择 “修改” > “转换为元件”。
7. 将该元件命名为 **img1**，确保将行为设置为 “影片剪辑”。

   默认状态下，元件的注册点在元件的左上角。
8. 单击 “确定”将位图图像转换为影片剪辑。
9. 在该图像还处于选中状态时，打开属性检查器（“窗口” > “属性” > “属性”），为影片剪辑分配实例名称 **img1_mc**。
10. 在主时间轴中选择第 1 帧，在 “动作”面板中添加下面的 ActionScript：

```
mx.transitions.TransitionManager.start(img1_mc,
  {type:mx.transitions.Zoom, direction:0, duration:1,
  easing:mx.transitions.easing.Bounce.easeOut, param1:empty,
  param2:empty});
```

> **注意** 有关使用包的信息，请参见第 447 页的 “使用滤镜包”。

11. 选择 “控制” > “测试影片”来测试动画。

    图像逐渐放大，并且在恢复到其原始大小之前伴有轻微的回弹效果。如果动画运动过快，则可以将动画的持续时间（在上面的代码片断中）从一秒增加到两秒或三秒（例如，`duration:3`）。

    您可能会注意到图像最初定位在左上角，然后向右下角放大。这与之前在 “转变”对话框中见到的预览不一样。

    使用 Tween 和 TransitionManager 类创建复杂的动画非常容易，不需要在时间轴上创建补间动画或补间形状。最为重要的是，不需要编写复杂的数学算法来创建缓动方法。如果想让图像从中心开始放大，而不是从一个角上的位置开始，那么在将图像从位图转换成元件时修改元件的注册点就可以了。

从图像中心放大图像：

1. 完成前一过程中的步骤。
2. 打开 zoom.fla 文件，选择 “文件” > “另存为”，保存该文档的一个新副本。

   将该文件保存为 **zoom2.fla**。
3. 将该位图元件的副本从 “库” 面板拖到舞台上，位于当前影片剪辑元件的旁边。
4. 在该位图图像在舞台上仍然处于选定状态时，按 F8 将该元件转换为影片剪辑。

   将该元件命名为 **img2**。
5. 在 “转换为元件” 对话框中，单击 3 x 3 网格的中心，以将注册点设置在位图的中心，并单击 “确定”。

6. 在舞台上选择新的影片剪辑，使用属性检查器为将其实例名称指定为 **img2_mc**。
7. 在主时间轴中选择第 1 帧，将下面的 ActionScript 添加到现有代码中：

```
mx.transitions.TransitionManager.start(img2_mc,
  {type:mx.transitions.Zoom, direction:mx.transitions.Transition.IN,
  duration:1, easing:mx.transitions.easing.Bounce.easeOut});
```

8. 选择 “控制” > “测试影片” 来测试动画。

   现在，第二个影片剪辑将从元件的中心放大，而不是从角部放大。

提示

一些转变对注册点的位置比较敏感。更改注册点的位置，可能会使动画在 SWF 文件中表现出意想不到的效果。例如，如果使用 “缩放” 转变时注册点位于左上角 （默认），则转变从该位置开始。

有关 Tween 类的每个方法和属性的信息，请参见《组件语言参考》中的第 51 章 “Tween 类”。有关 TransitionManager 类的每个方法和属性的信息，请参见《组件语言参考》中的第 48 章 “TransitionManager 类”。

提供了使用这些类来添加脚本动画的范例源文件。可在硬盘上的 Samples 文件夹中找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

关于缓动类和方法

第 433 页的“在 Flash Professional 8 中向文件添加补间和转变（仅限 Flash Professional 8）”中介绍了如何使用 Bounce 缓动类向影片剪辑添加回弹效果。除了 Bounce 之外，Flash 8 还提供了其它五种缓动类，下表中对这些类进行了说明：

转变	说明
返回	将动画扩展到转变范围的一端或两端之外一次，以模拟溢出效果。
回弹	在转变范围的一端或两端内添加回弹效果。回弹次数与持续时间相关 -- 持续时间越长，回弹次数越多。
弹性	添加在一端或两端转变范围之外的弹性效果。弹性量不受持续时间影响。
常规	在一端或两端添加较慢的动作。此特性可以帮助您添加加速效果，减速效果，或者同时添加这两种效果。
强制	在一端或两端添加较慢的动作。此效果类似于常规缓动，但它更明显。
无	添加从开始到结尾的无任何减速或加速效果的匀速运动。此转变也称为线性转变。

以上六种缓动类每个又包含三种缓动方法，如下表所示：

方法	说明
`easeIn`	在转变的开始提供缓动效果。
`easeOut`	在转变的结尾提供缓动效果。
`easeInOut`	在转变的开始和结尾提供缓动效果。

若要在 Flash 或 ActionScript 编辑器中打开这些类，请在 Windows（假定是默认安装）中浏览到 硬盘 \Program Files\Macromedia\Flash 8\ 语言 \First Run\Classes\mx\transitions\easing\ 文件夹，或者浏览到 Macintosh HD:Applications:Macromedia Flash 8:First Run:Classes:mx:transitions:easing。

第 434 页的 “使用 TransitionManager 和 Tween 类添加动画”中关于缩放图像的过程使用了 mx.transitions.easing.Bounce.easeOut 缓动类和方法。在硬盘上的文件夹中，ActionScript 引用了 Bounce.as 类中的 `easeOut()` 方法。此 ActionScript 文件位于 easing 文件夹中。

有关 Tween 类的每个方法和属性的信息，请参见《组件语言参考》中的第 51 章“Tween 类”。有关 TransitionManager 类的每个方法和属性的信息，请参见《组件语言参考》中的第 48 章“TransitionManager 类”。

提示

若要预览每个转变如何使用缓动类中的不同方法，可以双击 启动驱动器 \Program Files\Macromedia\Flash 8\ 语言 \First Run\Behaviors\ 或 Macintosh HD:Applications:Macromedia Flash 8:First Run:Behaviors: 中的 Transition.swf，以在独立播放器中打开 SWF 文件。

提供了使用这些类来添加脚本动画的范例源文件。可在硬盘上的 Samples 文件夹中找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

关于 Tween 类

Tween 类允许您轻松地在舞台上移动、缩放和淡入淡出影片剪辑。 mx.transitions.Tween 类的构造函数有以下参数名称和类型：

```
function Tween(obj, prop, func, begin, finish, duration, useSeconds) {
    // 代码……
}
```

`obj` Tween 实例的目标影片剪辑对象。

`prop` `obj` 中值要补间的属性的字符串名称。

`func` 为补间对象的属性值计算缓动效果的缓动方法。

`begin` 一个指示 `prop`（要补间的目标对象属性）的开始值的数字。

`finish` 一个指示 `prop`（要补间的目标对象属性）的结束值的数字。

`duration` 一个数字，指示补间动画的时间长度。如果省略，duration 会默认设置为 `infinity`。

`useSeconds` 与您在 `duration` 参数中指定的值相关的一个布尔值，如果该值为 `true`，则指示使用秒，如果为 `false`，则使用帧。

例如，假设您想让一个影片剪辑通过舞台。您可以向主时间轴上添加关键帧，并在这些帧之间插入补间动画或补间形状；可以在 onEnterFrame 事件处理函数中编写一些代码实现；或者可以使用 setInterval() 函数每隔一段时间就调用一个函数而加以实现。如果您使用 Tween 类，则可以通过另外一种方法来修改影片剪辑的 _x 和 _y 属性。您还可以添加前面介绍的缓动方法。若要利用 Tween 类，可以使用下面的 ActionScript：

```
new mx.transitions.Tween(ball_mc, "_x",
  mx.transitions.easing.Elastic.easeOut, 0, 300, 3, true);
```

此 ActionScript 代码片断创建了 Tween 类的一个新实例，它使 ball_mc 影片剪辑在舞台上沿 x 轴（从左到右）运动。该影片剪辑在 3 秒钟的时间内从 0 像素移动到 300 像素，并且 ActionScript 应用了一种弹性缓动方法。这意味着在以流畅的效果向回移动之前，球沿着 x 轴运动了 300 像素。

提供了使用这些类添加脚本动画的范例源文件。在硬盘上的 Samples 文件夹中可以找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

使用 Tween 类

如果在您的 Flash 文档中的不止一处地方要使用 Tween 类，则可以选用 import 语句。它允许您导入 Tween 类和缓动方法，而不用每次使用它们时都给出完全限定类名，如下面的过程所示。

导入和使用 Tween 类：

1. 创建一个新文档，并将其命名为 **easeTween.fla**。
2. 在舞台上创建一个影片剪辑。
3. 选择影片剪辑实例，并在属性检查器的 “实例名称” 文本框中键入 **ball_mc**。
4. 在时间轴中选择第 1 帧，在 “动作” 面板中，添加下面的代码：

   ```
   import mx.transitions.Tween;
   import mx.transitions.easing.*;
   new Tween(ball_mc, "_x", Elastic.easeOut, Stage.width, 0, 3, true);
   ```

   此代码示例使用两个 import 语句。第一个语句只导入 mx.transitions.Tween 类，第二个 import 语句使用通配符 (*) 快捷方式通过使用单行代码导入六个缓动类。第二个语句可导入整个类包。

参阅	有关使用包的信息，请参见第 447 页的 “使用滤镜包”。

5. 选择 “控制” > “测试影片”，以查看动画。

Flash 文档将包 定义为 “包含一个或多个类文件并位于指定的类路径目录中的目录”。在本例中，包位于 C:\Program Files\Macromedia\Flash 8\ 语言 \First Run\Classes\mx\transitions\easing 文件夹中 (Windows)，或位于 HD:Applications:Macromedia Flash 8:First Run:Classes:mx:transitions:easing 中 (Macintosh)。导入整个包比分别导入六个类好得多，您想必也同意这一点。 ActionScript 可以直接引用 Tween 类，而不必引用 mx.transitions.Tween 类。同样，对于缓动类而言就不必使用完全限定类名，如 mx.transitions.easing.Elastic.easeOut，您可以在 ActionScript 代码内键入 **Elastic.easeOut**。有关更多信息，请参见第 447 页的 “使用滤镜包”。

通过使用类似的代码，您可以设置 _alpha 属性 （而非 _x 属性）来淡入淡出实例，如接下来的过程所示。

使用 Tween 类淡化实例：

1. 创建一个新文档，并将其命名为 **fadeTween.fla**。
2. 在舞台上创建一个影片剪辑。
3. 选择影片剪辑实例，并在属性检查器的 “实例名称” 文本框中键入 **ball_mc**。
4. 在时间轴中选择第 1 帧，在 “动作” 面板中添加下面的代码：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;
new Tween(ball_mc, "_alpha", Strong.easeIn, 100, 0, 3, true);
```

   现在，ball_mc 不在舞台上移动了，而是在三秒钟内从 100% 可见淡化为完全透明。若要使元件更快地淡出，请将 duration 参数从 3 更改为 1 或 2。

5. 选择 “控制” > “测试影片”，以查看动画。

   如果更改文档的帧频，动画将播放得更平滑。有关动画和帧频的信息，请参见第 423 页的 “关于动画和帧频”。

除了使用秒之外，您还可以通过几个帧来淡化元件。若要在 Tween 类中将持续时间设置为帧数而不是秒数，可以将最后一个参数 useSeconds 从 true 更改为 false。将该参数设置为 true 时，就告诉 Flash 指定的持续时间以秒为单位。若将该参数设置为 false，则持续时间是您希望用于补间的帧的数量。接下来的过程演示如何将补间设置为帧数而不是秒数。

将持续时间设置为帧数而不是秒数：

1. 创建一个新文档，并将其命名为 **framesTween.fla**。
2. 在舞台上创建一个影片剪辑。
3. 选择影片剪辑实例，并在属性检查器的 “实例名称” 文本框中键入 **ball_mc**。

4. 在时间轴中选择第 1 帧，在 “动作”面板中，添加下面的代码：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;
new Tween(ball_mc, "_alpha", Strong.easeIn, 100, 0, 24, false);
```

此代码使用 `Strong.easeIn` 缓动方法淡出 `ball_mc` 实例。不是在三秒内淡化实例，而是通过 24 个帧淡化实例。

5. 选择 “控制” > “测试影片”，以查看动画。

稍等，实例将通过 24 个帧淡出。

6. 返回到创作环境，打开属性检查器。
7. 将文档的帧频更改为 **24** fps。

如果增加 FLA 文件的帧频，您将看到实例以更快的速度淡出。有关动画和帧频的信息，请参见第 423 页的 “关于动画和帧频”。

使用帧数而不是秒数来计算持续时间提供了更大的灵活性，但请记住，持续时间与当前 Flash 文档的帧频相关。如果 Flash 文档使用的是 12 帧 / 秒 (fps) 的帧频，那么上面的代码片断就可以在两秒内淡化实例 （24 帧 /12 fps = 2 秒）。但是，如果帧频是 24 fps，同样的代码就可以在 1 秒内淡化实例 （24 帧 /24 fps = 1 秒）。如果您使用帧数来计算持续时间，则在更改文档的帧频时可以明显地改变动画速度，而不需要修改 ActionScript。

Tween 类还有几种更有用的功能。例如，可以编写一个在动画完成后触发的事件处理函数，如接下来的过程所示。

在动画完成时触发代码：

1. 创建一个新文档，并将其命名为 **triggerTween.fla**。
2. 在舞台上创建一个影片剪辑。
3. 选择影片剪辑实例，并在属性检查器的 “实例名称”文本框中键入 **ball_mc**。
4. 在时间轴中选择第 1 帧，在 “动作”面板中，添加下面的代码：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;
var tween_handler:Object = new Tween(ball_mc, "_alpha", Strong.easeIn,
  100, 0, 3, true);
tween_handler.onMotionFinished = function() {
  trace("onMotionFinished triggered");
};
```

如果在 FLA 文件中测试此 ActionScript 代码，当 `ball_mc` 在舞台上完成淡化后会在 “输出”面板中显示 “onMotionFinished triggered”的消息。

5. 选择 “控制” > “测试影片”，以查看动画。

稍等片刻，实例将淡出。当补间完成时，您将在 “输出”面板中看到消息。

有关功能的更多信息，请参见第 6 章 “类”。

关于使用 continueTo() 方法继续动画

第 439 页的 “使用 Tween 类”演示如何在您的应用程序中使用 Tween 类。但是，如果您要在初始动画完成后移动球，至少有两种方法可以实现。一种解决方案是使用 onMotionFinished 事件处理函数重新为该球添加动画效果。不过，Tween 类提供了一个更简单的解决方案，即 continueTo() 方法。continueTo() 方法指示已经补间的动画继续从其当前值进行到一个新值，如下面的 ActionScript 所示：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;
var ball_tween:Object = new Tween(ball_mc, "_x", Regular.easeIn, 0, 300, 3,
  true);
ball_tween.onMotionFinished = function() {
    ball_tween.continueTo(0, 3);
};
```

初始补间完成后，ball_mc 影片剪辑将补间回其原始位置（0 像素处）。下面的代码片断（进行了简化）显示了 continueTo() 方法的函数原型：

```
function continueTo(finish:Number, duration:Number):Void {
    /* 为节省空间而省略。 */
}
```

只向 continueTo() 方法传递两个参数，而不是用于 Tween 构造函数方法的七个参数，如下面的代码片断所示：

```
function Tween (obj, prop, func, begin, finish, duration, useSeconds) {
    /* 为节省空间而省略。 */
}
```

continueTo() 方法不需要的五个参数（obj、prop、func、begin 和 useSeconds）将使用您前面在调用 Tween 类时定义的参数。调用 continueTo() 方法时，将假定 obj、prop、func（缓动类型）和 useSeconds 参数与前面调用 Tween 类时所用的参数相同。continueTo() 方法使用调用 Tween 类时的 finish 值作为 begin 参数的值，而不另外指定，如下面的 ActionScript 所示：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;
var ball_tween:Object = new Tween(ball_mc, "_x", Regular.easeIn, 0, 300, 3,
  true);
ball_tween.onMotionFinished = function() {
     ball_tween.continueTo(0, 3);
};
```

此代码可以使 ball_mc 实例在三秒钟的时间内沿 x 轴从 0 像素移动到 300 像素。动画结束后，将触发 onMotionFinished 事件处理函数并调用 continueTo() 方法。continueTo() 方法告诉其目标对象 (ball_mc) 继续沿 x 轴移动，从当前位置开始在三秒钟内使用相同的缓动方法移回 0 像素。您可以使用调用 Tween 构造函数方法时指定的值作为在 continueTo() 方法中未定义的任何参数。如果您没有为 continueTo() 方法指定持续时间，它就会使用在调用 Tween 构造函数时指定的持续时间。

创建连续运行的动画

您可以使动画沿 x 轴来回移动永不停止。Tween 类将这种动画恰当地命名为 yoyo() 方法。yoyo() 方法将等待 onMotionFinished 事件处理函数以开始执行，随后交换 begin 和 finish 参数。动画将再次开始，如下面的过程所示。

创建无止境运动的动画：

1. 创建一个名为 **yoyo.fla** 的新 Flash 文档。
2. 打开 “动作”面板并在时间轴的第 1 帧上输入以下 ActionScript：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;

this.createEmptyMovieClip("box_mc", this.getNextHighestDepth());
with (box_mc) {
  beginFill(0xFF0000, 60);
  moveTo(0, 0);
  lineTo(20, 0);
  lineTo(20, Stage.height);
  lineTo(0, Stage.height);
  lineTo(0, 0);
  endFill();
}
```

   第一部分代码首先导入 Tween 类和 easing 包中的每个类。下一部分代码创建一个实例名称为 box_mc 的新影片剪辑，并在舞台上绘制一个宽度为 20 像素、高度与舞台相等的矩形。

3. 在上一步中创建的代码后添加以下 ActionScript：

```
var box_tween:Tween = new Tween(box_mc, "_x", Regular.easeInOut, 0,
  Stage.width, 3, true);
box_tween.onMotionFinished = function() {
     box_tween.yoyo();
};
```

   此代码将创建一个新补间，以使 box_mc 影片剪辑在 3 秒内沿 x 轴通过舞台。

4. 选择 “控制” > “测试影片” 来测试动画。

   矩形框从左到右来回移动。如果动画不平滑，可能需要将文档的帧频从 12 fps 增加到 24 fps。

   随着矩形框接近舞台的右侧边缘，它可能会运动到舞台的边界之外。虽然这看起来并不是多大的事，但是您可能不希望该矩形从舞台的边缘消失，然后在一秒之后再次出现，并从另一个方向移动。

   若要进行调整，您需要将该矩形从 0 像素移到舞台宽度减去 box_mc 影片剪辑的宽度的位置。

5. 若要阻止矩形消失，请在步骤 3 中修改对应的代码行，以与下面的代码相符：

```
var box_tween:Tween = new Tween(box_mc, "_x", Regular.easeInOut, 0,
  (Stage.width - box_mc._width), 3, true);
```

6. 再次测试动画 （“控制” > “测试影片”）。

   现在，该框在从舞台边缘消失之前将停止缓动。

将 TransitionManager 与 Tween 类结合使用

将 TransitionManager 与 Tween 类结合使用后，可以生成有趣的效果。可以使用 TransitionManager 类沿 x 轴移动影片剪辑，同时使用 Tween 类调整同一剪辑的 _alpha 属性。每个类都可以使用不同的缓动方法，这意味着 SWF 文件中的对象有多种可能的动画。您可以利用 Tween 类的 continueTo() 和 yoyo() 方法或 onMotionFinished 事件处理函数来创建独特的效果。

可以将 TransitionManager 与 Tween 类结合使用，以为动态加载的影片剪辑实现动画效果，在从远程服务器上完全加载之后淡入到舞台上，如下面的过程所示。

将 TransitionManager 与 Tween 类一起使用：

1. 创建一个新的 Flash 文档，并将该文件保存为 **combination.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import mx.transitions.*;
import mx.transitions.easing.*;

var mcl_obj:Object = new Object();
mcl_obj.onLoadInit = function(target_mc:MovieClip) {
    new Tween(target_mc, "_alpha", Strong.easeIn, 0, 100, 2, true);
    TransitionManager.start(target_mc, {type:Fly,
  direction:Transition.IN, duration:3, easing:Elastic.easeInOut,
  startPoint:6});
};
```

```
var my_mcl:MovieClipLoader = new MovieClipLoader();
my_mcl.addListener(mcl_obj);
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  this.createEmptyMovieClip("img_mc", this.getNextHighestDepth()));
```

此代码分为三个主要部分。

第一部分代码导入转变包以及 `transitions.easing` 包中的类。在此示例中，可以导入整个转变包，以便不需要输入 Tween 类、TransitionManager 类或选定的转变（在本例中为 Fly）的完全限定类名。此过程可以减少键入的代码量，避免了潜在的键入错误。

ActionScript 的第二部分为 MovieClipLoader 类实例创建了一个侦听器对象，此实例是您在代码的第三部分创建的。当目标影片剪辑加载到 MovieClipLoader 实例中时，`onLoadInit` 事件就会触发并执行同时调用 Tween 类和 TransitionManager 类的代码块。因为您修改了 Tween 类中的 `_alpha` 属性，此事件处理函数将淡入目标影片剪辑，使目标影片剪辑沿 x 轴飞行。

ActionScript 代码的第三部分创建了一个 MovieClipLoader 实例，并应用前面创建的侦听器对象（以便目标影片剪辑加载器实例可以侦听 `onLoadInit` 事件）。然后，将目标 JPEG 图像加载到通过调用 `createEmptyMovieClip()` 方法动态创建的影片剪辑中。

3. 保存文档，然后选择“控制”>“测试影片”在测试环境中观看动画。

   当外部 JPEG 图像从服务器下载完毕后，该图像就会逐渐淡入，并从右向左通过舞台。

有关使用 Tween 类的信息，请参见第 439 页的“使用 Tween 类”。

有关 Tween 类的每个方法和属性的信息，请参见《组件语言参考》中的第 51 章“Tween 类”。有关 TransitionManager 类的每个方法和属性的信息，请参见《组件语言参考》中的第 48 章“TransitionManager 类”。

提供了使用这些类添加脚本动画的范例源文件。在硬盘上的 Samples 文件夹中可以找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

使用滤镜效果

滤镜是可以对 Flash 播放器在运行时呈现的对象应用的视觉效果，例如影片剪辑实例。滤镜包括投影、模糊、发光、斜角、渐变发光和渐变斜角。还可以使用调整颜色滤镜，通过该滤镜可以编辑影片剪辑的亮度、对比度、饱和度和色相。可以使用 Flash Professional 8 中的 Flash 用户界面或者在 Flash Basic 8 或 Flash Professional 8 中使用 ActionScript 来应用滤镜。

通过使用属性检查器中的 “滤镜”选项卡或使用 ActionScript 可以将上述每个滤镜效果应用于影片剪辑、按钮或文本字段。如果您使用 ActionScript 将滤镜应用于实例，那么还可以使用置换图滤镜（请参见第 472 页的 “使用置换图滤镜”）或盘绕滤镜（请参见第 471 页的 “使用卷积滤镜”）。这些滤镜将应用于矢量定义，因此没有用于在 SWF 文件中存储位图图像的开销。还可以编写 ActionScript，以便对应用于文本字段、影片剪辑或按钮的现有滤镜进行修改。

下面的过程演示如何使用 `onEnterFrame` 事件处理函数为影片剪辑添加发光滤镜的动画效果。

为应用于影片剪辑实例的滤镜实现动画效果：

1. 创建一个新的 Flash 文档，并将其保存为 **animFilter.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("box_mc", 10);
box_mc.lineStyle(20, 0x000000);
box_mc.beginFill(0x000000);
box_mc.moveTo(0, 0);
box_mc.lineTo(160, 0);
box_mc.lineTo(160, 120);
box_mc.lineTo(0, 120);
box_mc.lineTo(0, 0);
box_mc.endFill();
box_mc._x = 100;
box_mc._y = 100;

box_mc.filters = [new flash.filters.GlowFilter()];
var dir:Number = 1;
box_mc.blur = 10;
box_mc.onEnterFrame = function() {
  box_mc.blur += dir;
  if ((box_mc.blur >= 30) || (box_mc.blur <= 10)) {
    dir *= -1;
  }
  var filter_array:Array = box_mc.filters;
  filter_array[0].blurX = box_mc.blur;
  filter_array[0].blurY = box_mc.blur;
  box_mc.filters = filter_array;
};
```

此代码完成了两个不同的功能。第一部分创建并定位影片剪辑实例，并在舞台上绘制黑色的圆角矩形。第二个代码块对舞台上的矩形应用发光滤镜，并定义负责添加滤镜动画效果的 onEnterFrame 事件处理函数。onEnterFrame 事件处理函数为 10 到 30 像素之间的模糊区域中的发光滤镜添加动画效果，并且在动画大于等于 30 或小于等于 10 之后，动画将反转方向。

3. 保存对 Flash 文档所做的更改，然后选择“控制”>“测试影片”对 SWF 文件进行测试。

有关在应用程序中使用滤镜的更多信息，请参见以下主题：

- 第 447 页的 “使用滤镜包”
- 第 449 页的 “使用滤镜、缓存和 MovieClip 类”
- 第 450 页的 “关于点击检测、旋转、倾斜和缩放滤镜”
- 第 451 页的 “将滤镜应用于对象实例和 BitmapData 实例”
- 第 451 页的 “关于错误处理、性能和滤镜”

有关使用 ActionScript 应用滤镜的示例，可在硬盘上的 Samples 文件夹中找到范例源文件 Filters.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Filters。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Filters。

使用滤镜包

包是位于指定的类路径目录下并且包含一个或多个类文件的目录。例如，flash.filters 包是硬盘上的一个目录，对于 Flash 8 中的每种滤镜类型（例如 BevelFilter、BlurFilter、DropShadowFilter 等等），它都包含几个对应的类文件。当以这种方式组织类文件时，必须以特定方式访问类。要么导入 类，要么使用完全限定名称 来引用类。

注意 若要使用 import 语句，必须在 FLA 文件的 “发布设置” 对话框的 “Flash” 选项卡上指定 ActionScript 2.0 和 Flash Player 6 或更高版本。

使用 import 语句可以访问类，而无需指定类的完全限定名称。例如，若要在脚本中使用 BlurFilter，必须通过其完全限定名称 (flash.filters.BlurFilter) 来引用它，或将其导入；如果将其导入，则可在代码中通过类名称 (BlurFilter) 来引用它。下面的 ActionScript 代码演示了使用 import 语句和使用完全限定名称之间的差别。

如果不导入 BlurFilter 类，则代码需要使用完全限定的类名称 （包名称加上类名称），才能使用滤镜：

```
// 不导入
var myBlur:flash.filters.BlurFilter = new flash.filters.BlurFilter(10, 10,
  3);
```

使用用 import 语句编写的相同代码，您可以使用类名称来访问 BlurFilter，而无需不断使用完全限定名称来引用它。这样可以减少键入量，并可以减少出现键入错误的机会：

```
// 同时导入
import flash.filters.BlurFilter;
var myBlur:BlurFilter = new BlurFilter(10, 10, 3);
```

若要导入一个包内的多个类（例如 BlurFilter、DropShadowFilter 和 GlowFilter），您可以使用下面的两种方法之一来导入每个类。导入多个类的第一种方法是用单独的 import 语句导入每个类，如下面的代码片断所示：

```
import flash.filters.BlurFilter;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
```

如果对包中的每个类使用单独的导入语句，则编写代码既费时间，又容易出现键入错误。使用通配符 导入可以避免逐个导入单独的类文件，这样可以导入包中某一级别内的所有类。下面的 ActionScript 演示了一个使用通配符导入的示例：

```
import flash.filters.*; // 导入 flash.filters 包中的每个类
```

import 语句只应用到调用它的当前脚本（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。在这一帧上，可以使用类名称而不是类的完全限定名称来引用包中的类。若要在其它帧脚本中使用类名称，请使用类的完全限定名称引用该包中的类，或者在这个其它帧中添加一个 import 语句，以导入该包中的类。

使用导入语句时，切记只为您所指定的级别导入类。例如，如果您导入 mx.transitions 包中的所有类，则只会导入 /transitions/ 目录中的类，而不导入子目录中的类（例如 mx.transitions.easing 包中的类）。

提示

如果您导入一个类，但没有在脚本中使用该类，则该类不会作为 SWF 文件的一部分导出。这意味着您导入大型包时可以不必担心 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。

有关使用 ActionScript 应用滤镜的示例，可在硬盘上的 Samples 文件夹中找到范例源文件 Filters.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Filters。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Filters。

使用滤镜、缓存和 MovieClip 类

如果影片剪辑具有关联的滤镜，则加载 SWF 文件时它将被标记为将自身缓存为透明位图。只要影片剪辑至少应用了一个滤镜， Flash 播放器便会将 cacheAsBitmap 属性强制设置为 true，以在运行时将该影片剪辑作为位图缓存。此缓存的位图用作滤镜效果的源图像。每个影片剪辑通常具有两个位图：一个位图是原始的未过滤的源影片剪辑，另一个位图是过滤后的最终图像。如果您在运行时未更改影片剪辑的外观，则最终图像不需要进行更新，这有助于提高性能。

通过调用 MovieClip.filters 属性可以访问应用于实例的滤镜。调用此属性将返回一个数组，其中包含当前与该影片剪辑实例关联的每个滤镜对象。滤镜本身具有该滤镜特有的一组属性，如下所示：

```
trace(my_mc.filters[0].angle); // 45.0
trace(my_mc.filters[0].distance); // 4
```

可以像处理普通数组对象那样访问和修改滤镜。使用该属性设置并获取滤镜将返回滤镜对象的副本，而不是引用。

若要修改现有滤镜，可以使用与以下过程中的代码相似的代码。

在滤镜应用于影片剪辑实例时修改滤镜属性：

1. 创建一个新的 Flash 文档，然后将该文件保存为 **modifyFilter.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("my_mc", 10);
// 绘制方形
with (my_mc) {
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 100);
  lineTo(0, 100);
  lineTo(0, 0);
  endFill();
}
my_mc._x = 100;
my_mc._y = 100;

// 使用默认的 DropShadowFilter 值
my_mc.filters = [new flash.filters.DropShadowFilter()];
trace(my_mc.filters[0].distance); // 4
var filter_array:Array = my_mc.filters;
filter_array[0].distance = 10;
my_mc.filters = filter_array;
trace(my_mc.filters[0].distance); // 10
```

此代码的第一部分使用 Drawing API 创建一个红色方形，并将该形状定位在舞台上。代码的第二部分对方形应用一个投影滤镜。然后，代码创建一个临时数组，以存放要应用到舞台上的红色方形的当前滤镜。第一个滤镜的 `distance` 属性设置为 10 像素，并且将修改过的滤镜重新应用于 `my_mc` 影片剪辑实例。

3. 选择 “控制” > “测试影片” 来测试该文档。

注意	目前，不支持任何滤镜根据其父级的旋转或其它种类的旋转来执行旋转。模糊滤镜可始终实现水平或垂直的完美模糊效果，而不受对父对象树中的任何项目旋转或倾斜的影响。

提示	过滤的内容与 `cacheAsBitmap` 属性设置为 `true` 的内容对大小具有相同的限制。如果作者对 SWF 文件放大过多，则当任一方向的位图表示形式大于 2880 像素时，滤镜将不再可见。当您发布具有滤镜的 SWF 文件时，最好禁用缩放菜单选项。

有关使用 ActionScript 应用滤镜的示例，可在硬盘上的 Samples 文件夹中找到范例源文件 Filters.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Filters。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Filters。

关于点击检测、旋转、倾斜和缩放滤镜

在影片剪辑实例的边框矩形之外的任何过滤区域（例如投影）都不被视为用于进行点击检测（确定实例是否与其它实例重叠或交叉）表面的一部分。由于点击检测是基于矢量的，因此不能对位图结果执行点击检测。例如，如果您对按钮实例应用斜角滤镜，则在该实例的斜角部分，点击检测不可用。

滤镜不支持缩放、旋转和倾斜；如果实例本身进行了缩放（`_xscale` 和 `_yscale` 不是 100%），则滤镜效果将不随该实例缩放。这意味着，实例的原始形状将旋转、缩放或倾斜；而滤镜不随实例一起旋转、缩放或倾斜。

可以使用滤镜给实例添加动画，以形成理想的效果，或者嵌套实例并使用 BitmapData 类使滤镜动起来，以获得此效果。

将滤镜应用于对象实例和 BitmapData 实例

滤镜的具体使用取决于要应用滤镜的对象实例。当您将滤镜应用于对象或 BitmapData 实例时，请使用下面的原则：

- 若要在运行时对影片剪辑、文本字段和按钮应用滤镜，请使用 `filters` 属性。设置对象的 `filters` 属性不会修改对象，清除 `filters` 属性即可撤消设置操作。
- 若要对 BitmapData 实例应用滤镜，请使用 `BitmapData.applyFilter()` 方法。在 BitmapData 对象上调用 `applyFilter` 将修改 BitmapData 对象，并且，此操作是无法撤消的。

提示 （仅限 Flash Professional 8）在创作期间，还可以使用属性检查器中的 “滤镜”选项卡对图像和视频应用滤镜效果。

关于错误处理、性能和滤镜

如果在一个应用程序中使用太多滤镜，则可能使用大量内存，引起 Flash Player 性能降低。由于附加了滤镜的影片剪辑有两个 32 位位图，因此如果使用过多位图，这些位图会导致应用程序使用大量内存。您可能会看到计算机的操作系统生成内存不足的错误。在现在的计算机中，内存不足错误应该很少出现，除非在一个应用程序中过多地使用滤镜效果（例如，在舞台中存在数千个位图）。

但是，如果您确实遇到内存不足错误，则将出现以下情况：

- 滤镜数组被忽略。
- 使用常规矢量渲染器绘制影片剪辑。
- 不为影片剪辑缓存任何位图。

在出现内存不足错误后，影片剪辑绝不会尝试使用滤镜数组或位图缓存。影响播放器性能的另一个因素是您对所应用的每个滤镜的 `quality` 参数使用的值。值越高，则呈现所需的 CPU 时间和内存越多，而将 `quality` 参数设置为较低的值则需要较少的计算机资源。因此，应避免使用过多的滤镜，并且尽可能使用较低的 `quality` 设置。

小心 如果 100 x 100 像素的对象放大一倍，它将使用四倍的内存，因为现在的尺寸为 200 x 200 像素。如果再放大两倍，则该形状将绘制为 800 x 800 像素的对象，而它使用的内存为最初的 100 x 100 像素对象所使用内存的 64 倍。无论何时在 SWF 文件中使用滤镜时，最好都从 SWF 文件的上下文菜单中禁用缩放菜单选项。

如果使用无效的参数类型，也可能遇到错误。有些滤镜参数还有一个特定的有效范围。如果设置了有效范围之外的值，则该值将更改为该范围之内的有效值。例如，对于标准操作，`quality` 应该是在 1 到 3 之间的值，只能设置为 0 到 15。高于 15 的任何值都将设置为 15。

而且，有些构造函数还对作为输入参数所需的数组长度具有限制。如果使用无效数组（大小不正确）创建盘绕滤镜或颜色矩阵滤镜，则构造函数将失败，无法成功创建滤镜。如果随后将该滤镜对象用作影片剪辑滤镜数组的一项，则它将被忽略。

提示	使用模糊滤镜时，如果用于 blurX 和 blurY 的值是 2 的整数次幂（例如 2、4、8、16 和 32），则可以加快计算速度，并且可以使性能提高 20% 到 30%。

通过 ActionScript 使用滤镜

flash.filters 包中包含用于 Flash Player 8 中新增的位图滤镜效果的类。滤镜允许您使用 ActionScript 来为文本、影片剪辑和按钮实例应用丰富的视觉效果（如模糊、斜角、发光和投影）。还可以使用 Flash 创作工具将滤镜效果应用于文本、图像和视频等对象。 Flash 有九个滤镜效果，但使用 Flash Professional 8 中的用户界面只能访问七个滤镜效果。ConvolutionFilter 和 DisplacementMapFilter 滤镜只能通过使用 ActionScript 代码来使用。

注意	在 Flash Basic 8 和 Flash Professional 8 中均可以通过 ActionScript 使用所有滤镜。

下面的过程加载了一个半透明的 PNG 图像，并对该图像的不透明部分应用了 GlowFilter 效果。

对半透明的图像应用滤镜：

1. 创建一个新的 Flash 文档，并将其保存为 **transparentImg.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.GlowFilter;
System.security.allowDomain("http://www.helpexamples.com");
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
  var glow:GlowFilter = new GlowFilter();
  target_mc.filters = [glow];
};
this.createEmptyMovieClip("img_mc", 10);
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/logo.png",
  img_mc);
```

此代码使用影片剪辑加载器实例来加载半透明的 PNG 图像。图像加载完成后，图像将移动到舞台中央，并应用发光滤镜。

3. 选择“控制”>“测试影片”来测试该文档。

发光滤镜效果只应用于 PNG 图像的不透明区域。

下面各部分介绍了如何使用滤镜：

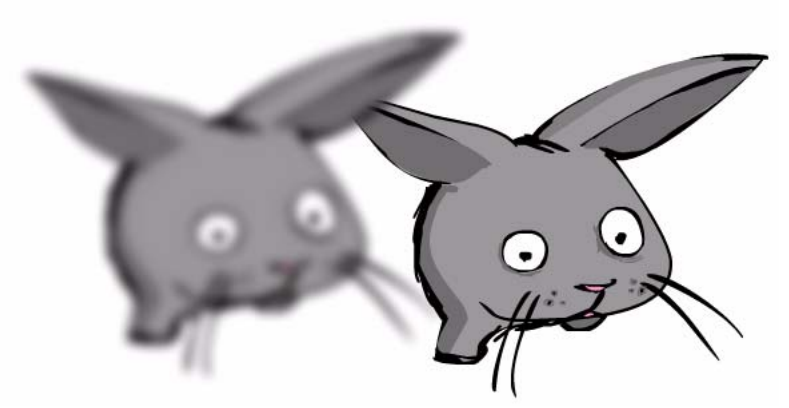
- 第 454 页的“使用模糊滤镜”
- 第 455 页的“使用投影滤镜”
- 第 459 页的“使用发光滤镜”
- 第 460 页的“创建渐变发光”
- 第 462 页的“使用斜角滤镜”
- 第 467 页的“应用渐变斜角滤镜”
- 第 469 页的“使用颜色矩阵滤镜”
- 第 471 页的“使用卷积滤镜”
- 第 472 页的“使用置换图滤镜”

有关使用 ActionScript 应用滤镜的示例，可在硬盘上的 Samples 文件夹中找到范例源文件 Filters.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Filters。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Filters。

使用模糊滤镜

BlurFilter 类允许您将模糊视觉效果应用于 Flash 中的各种对象。模糊效果可以柔化图像的细节。您可以生成一些模糊效果，范围包括：创建一个柔化的、未聚焦的外观到高斯模糊（就像通过半透明玻璃查看图像一样的朦胧的外观）。模糊滤镜基于 box-pass 模糊滤镜。`quality` 参数定义模糊重复的次数（三次大约传递一个 Gaussian 模糊滤镜）。



提示	仅当您在舞台中进行缩放时，模糊滤镜才会缩放。

有关此滤镜的更多信息，请参见《ActionScript 2.0 语言参考》中的 BlurFilter (flash.filters.BlurFilter)。

下面的过程根据鼠标在舞台上的当前位置来使动态加载的图像变得模糊。指针离舞台中央越远，图像就越模糊。

基于鼠标指针的位置使图像模糊：

1. 创建一个新的 Flash 文档，并将其保存为 **dynamicblur.fla**。
2. 将下面的代码添加到时间轴中的第 1 帧：

```
import flash.filters.BlurFilter;
System.security.allowDomain("http://www.helpexamples.com");
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  // 使 target_mc 影片剪辑位于舞台中央。
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
};
this.createEmptyMovieClip("img_mc", 10);
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
```

```
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
var blur:BlurFilter = new BlurFilter(10, 10, 2);

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  /* Moving the pointer to the center of the Stage sets the blurX and
  blurY properties to 0%. */
  blur.blurX = Math.abs(_xmouse - (Stage.width / 2)) / Stage.width * 2 *
  255;
  blur.blurY = Math.abs(_ymouse - (Stage.height / 2)) / Stage.height * 2
  * 255;
  img_mc.filters = [blur];
};
Mouse.addListener(mouseListener);
```

本代码的第一部分加载图像并确定动态加载的图像在舞台上的位置。第二部分定义一个在鼠标移动时就会调用的侦听器。根据鼠标在舞台上的当前位置，可以计算水平和垂直的模糊量。将指针移动得距离舞台中央越远，则应用于实例的模糊效果就越强。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

沿 x 轴移动鼠标指针，以修改水平的模糊量。当指针移动得距离舞台中央越远，该实例就变得越模糊。沿 y 轴移动指针，将使垂直模糊增加或减少，具体取决于到舞台的垂直中心的距离。

提示	当您使用模糊滤镜时，如果用于 blurX 和 blurY 的值是 2 的整数次幂 （例如 2、 4、 8、 16 和 32），则可以加快计算速度，并可将性能提高 20% 到 30%。

小心	将模糊值设置为低于 1.03125 的值时将禁用模糊效果。

使用投影滤镜

使用 DropShadowFilter 类，可以在 Flash 中为各种对象添加投影。阴影算法基于模糊滤镜使用的同一个框型滤镜（请参见第 454 页的 “使用模糊滤镜”）。投影样式有多个选项，包括内缘或外缘阴影和挖空模式。

有关投影滤镜的更多信息，请参见 《ActionScript 2.0 语言参考》中的 DropShadowFilter (flash.filters.DropShadowFilter)。

下面的过程使用 Drawing API 在舞台上绘制方形。沿舞台水平移动鼠标指针时，此代码将修改投影出现的位置到方形的距离，垂直移动鼠标时，则会修改投影模糊的量。

使用投影滤镜：

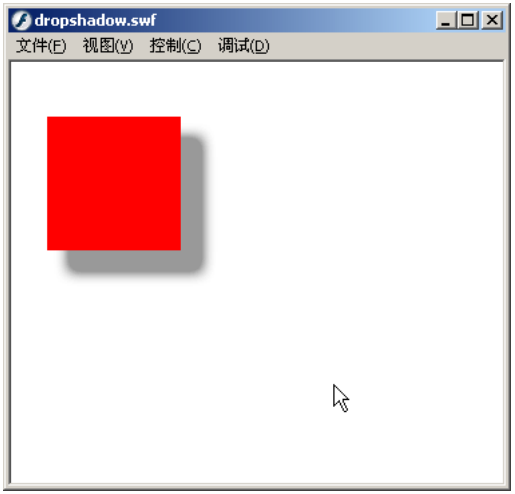
1. 创建一个新的 Flash 文档，并将其保存为 **dropshadow.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
// import the filter classes
import flash.filters.DropShadowFilter;
// 创建一个名为 shapeClip 的影片剪辑
this.createEmptyMovieClip("shapeClip", 1);
// 使用 Drawing API 绘制形状
with (shapeClip) {
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 100);
  lineTo(0, 100);
  lineTo(0, 0);
  endFill();
}
// 定位形状
shapeClip._x = 100;
shapeClip._y = 100;
// 单击方形，增加阴影强度
shapeClip.onPress = function():Void {
  dropShadow.strength++;
  shapeClip.filters = [dropShadow];
};
// 创建一个滤镜
var dropShadow:DropShadowFilter = new DropShadowFilter(4, 45, 0x000000,
  0.4, 10, 10, 2, 3);

var mouseListener:Object = new Object();
// 创建并应用一个侦听器，用于在鼠标移动时控制滤镜
mouseListener.onMouseMove = function():Void {
  dropShadow.distance = (_xmouse / Stage.width) * 50 - 20;
  dropShadow.blurX = (_ymouse / Stage.height) * 10;
  dropShadow.blurY = dropShadow.blurX;
  shapeClip.filters = [dropShadow];
};
Mouse.addListener(mouseListener);
```

代码的第一部分创建一个新的影片剪辑并使用 Drawing API 绘制红色的方形。代码的第二部分定义鼠标侦听器，当鼠标移动时将随时调用该侦听器。鼠标侦听器根据鼠标指针当前的 x 和 y 位置来计算投影的距离和模糊的级别，并重复应用投影滤镜。如果您单击红色方形，则投影的强度将增加。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

沿 x 轴移动鼠标指针，以更改投影距离的值，并沿 y 轴移动鼠标指针，以更改应用于影片剪辑实例的模糊量。



还可以创建投影并将其应用于动态加载的图像。下面的过程演示如何加载外部图像并应用跟随鼠标指针的投影。指针距离图像的左上角越远，则应用于图像的水平和垂直模糊越多。

创建跟随鼠标指针的投影：

1. 创建一个新的 Flash 文档，并将其保存为 **dropshadowmouse.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.DropShadowFilter;
System.security.allowDomain("http://www.helpexamples.com");
var dropShadow:DropShadowFilter = new DropShadowFilter(4, 45, 0x000000,
  0.8, 10, 10, 2, 2);
// 加载图像并在舞台上定位该图像。
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip):Void {
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
};
this.createEmptyMovieClip("img_mc", 10);
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);

// 当鼠标移动时，重新计算投影的位置。
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
```

```
    var p1:Number = img_mc._y - _ymouse;
    var p2:Number = img_mc._x - _xmouse;
    var degrees:Number = Math.atan2(p1, p2) / (Math.PI / 180);
    dropShadow.distance = Math.sqrt(Math.pow(p1, 2) + Math.pow(p2, 2)) *
    0.5;
    dropShadow.blurX = dropShadow.distance;
    dropShadow.blurY = dropShadow.blurX;
    dropShadow.angle = degrees - 180;
    img_mc.filters = [dropShadow];
  };
  Mouse.addListener(mouseListener);
```

此代码的第一部分定义投影实例，加载外部图像，并将该图像重新定位在舞台中央。代码的第二部分定义鼠标侦听器，当用户沿舞台移动鼠标指针时，将随即调用该侦听器。无论何时移动鼠标，事件处理函数都将重新计算鼠标指针与图像左上角之间的距离和角度。根据此计算，投影滤镜将重新应用于影片剪辑。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

   运行 SWF 文件时，投影将跟随鼠标指针。鼠标指针距离舞台中图像的左上角越近，应用于图像的模糊效果越少。鼠标指针距离图像的左上角越远，投影效果便越明显。

还可以将投影应用于动态加载的半透明 PNG 图像中。在下面的过程中，投影滤镜将只应用于 PNG 的实体区域，而不应用于透明区域。

将投影应用于半透明图像：

1. 创建一个新的 Flash 文档，并将其保存为 **dropshadowTransparent.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.DropShadowFilter;
System.security.allowDomain("http://www.helpexamples.com");
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip):Void {
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
  var dropShadow:DropShadowFilter = new DropShadowFilter(4, 45, 0x000000,
  0.5, 10, 10, 2, 3);
  target_mc.filters = [dropShadow];
};
mclListener.onLoadError = function(target_mc:MovieClip):Void {
  trace("unable to load image.");
};
this.createEmptyMovieClip("logo_mc", 10);
var my_mcl:MovieClipLoader = new MovieClipLoader();
my_mcl.addListener(mclListener);
my_mcl.loadClip("http://www.helpexamples.com/flash/images/logo.png",
  logo_mc);
```

此 ActionScript 代码使用 MovieClipLoader 类来加载图像，并在图像从远程服务器中加载完成时应用投影滤镜。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

   Flash 使用透明背景加载 PNG 图像。当您应用投影滤镜时，只有图像的不透明部分应用滤镜。

使用发光滤镜

使用 GlowFilter 类，您可以在 Flash 中给各种对象添加发光效果。发光算法基于模糊滤镜使用的同一个框型滤镜（请参见第 454 页的 “使用模糊滤镜”）。有多种方式可用于设置发光样式，包括内缘发光或外缘发光以及挖空模式。在投影的 `distance` 属性和 `angle` 属性设置为 `0` 时，发光滤镜与投影滤镜极为相似。

有关发光滤镜的更多信息，请参见 《ActionScript 2.0 语言参考》中的 GlowFilter (flash.filters.GlowFilter)。

下面的过程演示如何将发光滤镜应用于舞台上动态创建的影片剪辑。沿舞台移动鼠标指针将使影片剪辑的模糊更改，而单击动态创建的形状将使滤镜的强度增加。

使用发光滤镜：

1. 创建一个新的 Flash 文档，并将其保存为 **glowfilter.fla**。
2. 将下面的 ActionScript 代码添加到时间轴中的第 1 帧：

```
import flash.filters.GlowFilter;

this.createEmptyMovieClip("shapeClip", 10);
with (shapeClip) {
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 100);
  lineTo(0, 100);
  lineTo(0, 0);
  endFill();
}
shapeClip._x = 100;
shapeClip._y = 100;
shapeClip.onPress = function():Void {
  glow.strength++;
  shapeClip.filters = [glow];
};
var glow:GlowFilter = new GlowFilter(0xCC0000, 0.5, 10, 10, 2, 3);
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  glow.blurX = (_xmouse / Stage.width) * 255;
  glow.blurY = (_ymouse / Stage.width) * 255;
  shapeClip.filters = [glow];
};
Mouse.addListener(mouseListener);
```

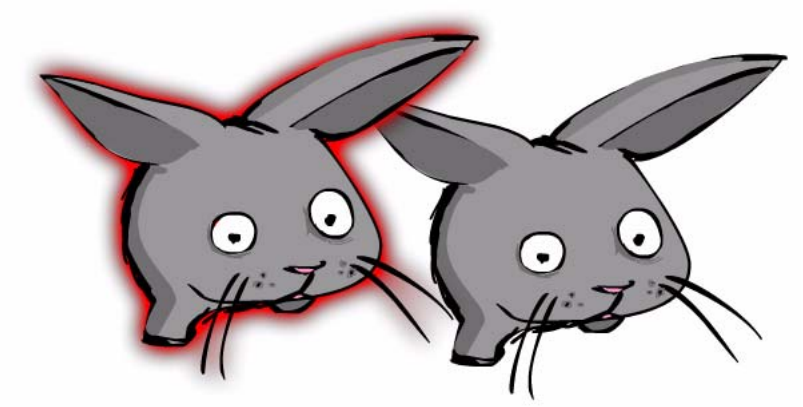
此代码使用 Drawing API 在舞台上绘制方形，并对该形状应用发光滤镜。无论何时沿 x 轴或 y 轴移动鼠标指针时，都将计算发光滤镜的模糊并将其应用于形状。

3. 选择 “控制” > “测试影片” 来测试该文档。

   水平和垂直模糊量是根据鼠标指针当前的 _xmouse 和 _ymouse 位置计算所得。当您将鼠标指针向舞台的左上角移动时，水平和垂直模糊量将减少。反之，当鼠标指针移向舞台的右下角时，水平和垂直模糊量将增加。

创建渐变发光

使用 GradientGlowFilter 类，您可以在 Flash 中为各种对象创建渐变发光效果。渐变发光是一种非常逼真的发光效果，您可以指定颜色渐变。可以在对象的内缘或外缘的周围或者对象的顶部应用渐变发光。



有关此滤镜的更多信息，请参见 《ActionScript 2.0 语言参考》中的 GradientBevelFilter (flash.filters.GradientBevelFilter)。

下面的过程使用 Drawing API 在舞台中绘制一个方形，然后对该形状应用渐变发光滤镜。单击舞台上的方形可以增加滤镜的强度，而水平或垂直移动鼠标指针则将修改沿 x 轴或 y 轴的模糊量。

应用渐变发光滤镜：

1. 创建一个新的 Flash 文档，并将其保存为 **gradientglow.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.GradientGlowFilter;
// 创建一个新的 shapeClip 实例
var shapeClip:MovieClip = this.createEmptyMovieClip("shapeClip", 10);
// 使用 Drawing API 创建形状
with (shapeClip) {
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 100);
  lineTo(0, 100);
  lineTo(0, 0);
  endFill();
}

// 定位该形状
shapeClip._x = 100;
shapeClip._y = 100;
// 定义渐变发光
var gradientGlow:GradientGlowFilter = new GradientGlowFilter(0, 45,
  [0x000000, 0xFF0000], [0, 1], [0, 255], 10, 10, 2, 3, "outer");

// 定义鼠标侦听器，侦听两个事件
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function():Void {
  gradientGlow.strength++;
  shapeClip.filters = [gradientGlow];
};
mouseListener.onMouseMove = function():Void {
  gradientGlow.blurX = (_xmouse / Stage.width) * 255;
  gradientGlow.blurY = (_ymouse / Stage.height) * 255;
  shapeClip.filters = [gradientGlow];
};
Mouse.addListener(mouseListener);
```

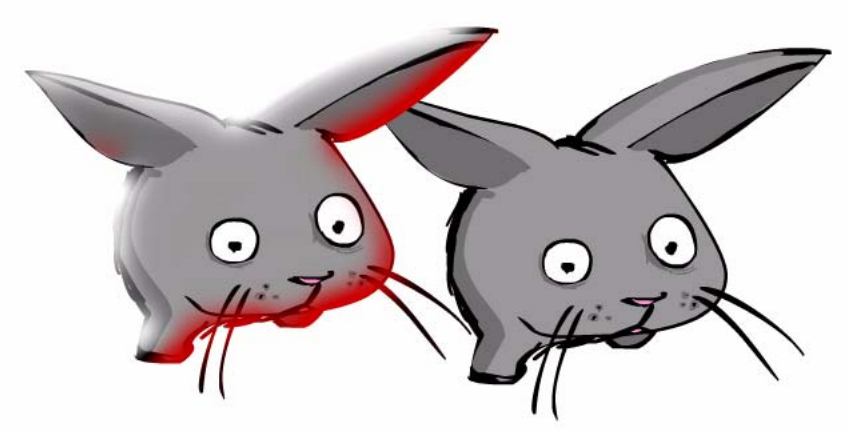
前面的代码分为三个部分。第一部分代码使用 Drawing API 创建一个方形，并在舞台上定位该形状。第二部分代码定义一个新的渐变发光滤镜实例，该实例将创建从红色到黑色的发光。第三部分代码定义一个鼠标侦听器，它侦听两个鼠标事件处理函数。第一个事件处理函数是 onMouseDown，它可以引起渐变发光的强度增加。第二个事件处理函数是 onMouseMove，当鼠标指针在 SWF 文件中移动时，将调用该函数。鼠标指针距离 Flash 文档的左上角越远，应用的发光效果就越强。

3. 选择“控制”>“测试影片”来测试该文档。

当您沿着舞台移动鼠标指针时，渐变发光滤镜的模糊效果将会增加或降低强度。单击鼠标左键将增加发光强度。

使用斜角滤镜

BevelFilter 类允许您在 Flash 中给各种不同对象添加斜角效果。斜角效果使对象具有三维外观。您可以利用不同的加亮颜色和阴影颜色、斜角上的模糊量、斜角的角度、斜角的位置和挖空效果来自定义斜角的外观。



有关此滤镜的更多信息，请参见《ActionScript 2.0 语言参考》中的 BevelFilter (flash.filters.BevelFilter)。

下面的过程使用 Drawing API 创建一个方形，并向该形状添加一个斜角。

使用斜角滤镜：

1. 创建一个新的 Flash 文档，并将其保存为 **bevel.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.BevelFilter;
// 定义一个斜角滤镜
var bevel:BevelFilter = new BevelFilter(4, 45, 0xFFFFFF, 1, 0xCC0000, 1,
  10, 10, 2, 3);
// 创建一个新的 shapeClip 实例
var shapeClip:MovieClip = this.createEmptyMovieClip("shapeClip", 1);
// 使用 Drawing API 创建一个形状
with (shapeClip) {
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 100);
  lineTo(0, 100);
  lineTo(0, 0);
  endFill();
}
// 在舞台上定位该形状
```

```
shapeClip._x = 100;
shapeClip._y = 100;
// 单击鼠标以增加强度
shapeClip.onPress = function():Void {
  bevel.strength += 2;
  shapeClip.filters = [bevel];
};

// 定义侦听器，以在指针移动时修改滤镜
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  bevel.distance = (_xmouse / Stage.width) * 10;
  bevel.blurX = (_ymouse / Stage.height) * 10;
  bevel.blurY = bevel.blurX;
  shapeClip.filters = [bevel];
};
Mouse.addListener(mouseListener);
```

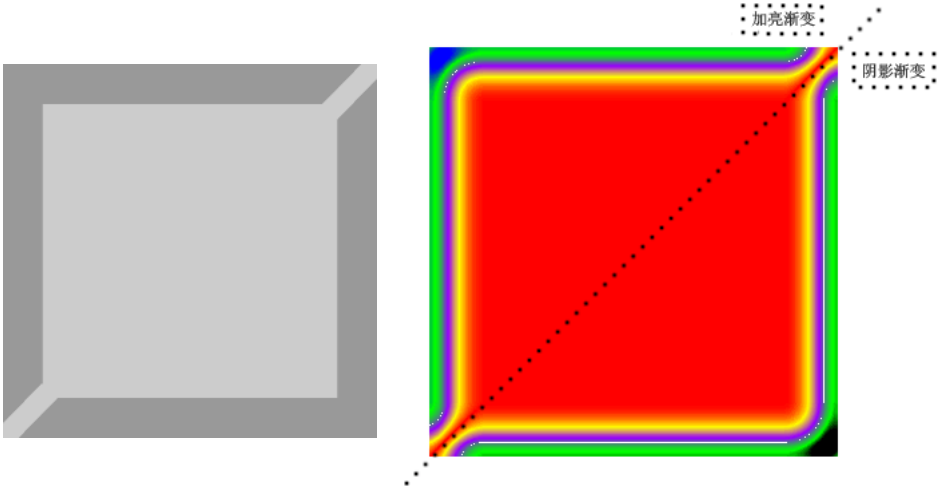
第一部分代码定义一个 BevelFilter 实例，并且使用 Drawing API 在舞台上绘制一个方形。当您在舞台上单击方形时，则斜角的当前强度值将递增，并使斜角具有更高、更清晰的外观。第二部分代码定义一个鼠标侦听器，它将根据鼠标指针的当前位置修改斜角的距离和模糊。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

当您沿 x 轴移动鼠标指针时，斜角的偏移距离将增加或减小。当您沿 y 轴移动鼠标指针时，鼠标指针的当前坐标将修改水平和垂直模糊量。

关于渐变斜角滤镜

渐变斜角滤镜应用于作为矩形的对象，渐变颜色分布于矩形的三个部分：两个斜角边缘（一个加亮和一个阴影）以及一个称为基本填充的区域。下面的关系图描绘了这个矩形，其斜角类型设置为内侧。在矩形的左侧，深灰色区域是斜角边缘，而浅灰色区域是基本填充。在矩形的右侧，应用了一个彩虹渐变斜角，每个边缘具有一个四色的斜角。



渐变斜角滤镜的不同属性控制着应用滤镜的方式。渐变斜角的颜色在 colors 数组中设置。矩形每个部分中颜色的实际分布由 ratios 数组来决定。distance 属性确定偏移距离，或者应用的斜角边缘距离对象的像素数。`blurX` 和 `blurY` 属性控制斜角中颜色的清晰度；较高的值将使斜角更宽、更缓和，而较低的值则使斜角更窄、更尖锐。 angle 属性是投射在对象上的理论光源，所以将在对象的边缘上产生加亮和阴影效果。The `strength` 属性控制颜色的散布: 较低的 strength 值将使颜色变淡，如示例中所示；较高的 strength 值可使数组中的外侧数更强，从而强制数组中的中间颜色不特别突出。最后，`knockout` 和 `type` 属性确定斜角滤镜如何以及在何处应用于整个对象：滤镜是否挖空对象及其放置位置。

应用于渐变斜角滤镜的一个较为复杂的概念是颜色分布。要理解渐变斜角中颜色的分布方式，请首先考虑要在渐变斜角中使用的颜色。由于简单斜角具有可以理解的加亮颜色和阴影颜色的概念，可以应用相同的概念来理解渐变斜角滤镜：您具有加亮的渐变和阴影渐变。加亮出现在左上角，而阴影出现在右下角。加亮和阴影部分分别有四种颜色。但是，还必须添加另外一种颜色（基本填充颜色），此颜色出现在加亮和阴影相交的边缘处。该 colors 数组中存在九种颜色，您在前面的关系图中可以看到。

colors 数组中的颜色数确定 alphas 和 ratios 数组中的元素数。 colors 数组中的第一项与 alphas 数组和 ratios 数组中的第一项对应，依此类推。由于具有九种颜色，因此在 alphas 数组和 ratios 数组中分别有九个值。 Alpha 值用于设置颜色的 Alpha 透明度值。

ratios 数组中的比例值可在 0 到 255 像素之间。中间值为 128； 128 是基本填充值。对于大多数应用，为获得您所需的斜角效果，应该如下所示使用九种颜色的示例来分配比例值：

- 前四种颜色的范围在 0 到 127 之间，且按顺序增加，所以每个值都大于或等于前面的值。这是第一个斜角边缘，即加亮部分。
- 第五种颜色（中间颜色）是基本填充，设置为 128。像素值 128 设置基本填充，如果类型设置为外侧，则它出现在形状外面 （在斜角边缘周围）；如果类型设置为内侧，则它出现在形状内部，将有效覆盖对象自己的填充。
- 后面的四种颜色的范围在 129 到 255 之间，且按顺序增加，所以每个值都大于或等于前面的值。这是第二个斜角边缘，例如阴影。

如果您认为渐变由各种颜色的条纹组成，彼此混合，每个比例值设置关联颜色的像素数，从而设置渐变中颜色条纹的宽度。如果您希望每个边缘的颜色均匀分布，则请执行以下操作：

- 使用奇数个颜色，其中的中间颜色是基本填充。
- 将您的颜色均匀分布在 0 到 127 以及 129 到 255 之间的值。
- 调整该值，以更改渐变中每个条纹颜色的宽度。

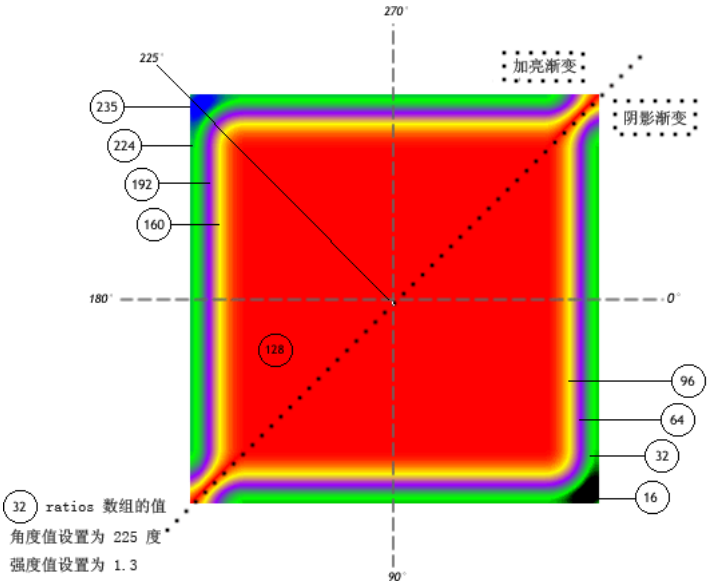
提示	角度值分别确定作为加亮和阴影的边缘。

角度值确定渐变颜色应用于对象时使用的角度；也就是说，加亮和阴影出现在对象中的位置。颜色应用的顺序与数组中的顺序相同。

下面的代码采用一个粉红色的方形 （使用 Drawing API 绘制）并应用彩虹渐变滤镜。颜色（按数组中出现的顺序）为：蓝色、绿色、紫色和黄色 （加亮）；红色 （基本填充）；黄色、紫色、绿色、黑色 （阴影）。为了确定比例值，我们分配了 0 到 127 之间的四种加亮颜色值，它们大致均匀分布，并分配了 129 到 255 之间的阴影颜色。外部边缘的颜色为蓝色 (16) 和黑色 (235)。

```
var colors:Array = [0x0000FF, 0x00FF00, 0x9900FF, 0xFFFF00, 0xFF0000,
  0xFFFF00, 0x9900FF, 0x00FF00,0x000000];
var alphas:Array = [1, 1, 1, 1, 1, 1, 1, 1, 1];
var ratios:Array = [16, 32, 64, 96, 128, 160, 192, 224, 235];
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(8, 225,
  colors, alphas, ratios, 16, 16, 1.3, 2, "inner", false);
```

下图显示了上面的代码所创建的渐变斜角滤镜，这是应用于红色矩形影片剪辑的九色彩虹斜角：



虚线显示确定角度的方式。该图显示了如何在滤镜中实现 225° 的角度，还显示了每种颜色的所有比例值。将角度设置为 225° 表示数组中的第一种颜色从 225° 开始，它位于左上角（加亮）。点划线显示应用加亮渐变和阴影渐变的位置。

原始的影片剪辑颜色是粉红色，但为红色设置值 128 意味着 128 像素值是基本填充，将覆盖原始影片剪辑填充。但是，当您设置 filters 属性时，原始对象不会更改；只需清除 filters 属性，便可还原为原始的影片剪辑填充。

所有滤镜的属性都会互相影响，所以如果调整一个属性以更改要应用的效果，则可能还需要调整其它属性。

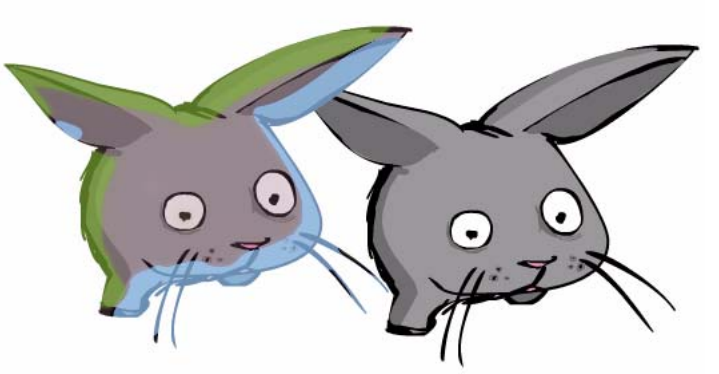
创建前面的图形的完整 ActionScript 代码如下：

```
import flash.filters.GradientBevelFilter;

// 绘制填充的方形形状
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
    square_mc.beginFill(0xFF99CC);
    square_mc.moveTo(40, 40);
    square_mc.lineTo(200, 40);
    square_mc.lineTo(200, 200);
    square_mc.lineTo(40, 200);
    square_mc.lineTo(40, 40);
    square_mc.endFill();
```

```
/* GradientBevelFilter(distance:Number, angle:Number, colors:Array,
  alphas:Array, ratios:Array, blurX:Number, blurY:Number, strength:Number,
  quality:Number, type:String, knockout:Boolean) */

// 创建 colors、alphas 和 ratios 数组
var colors:Array = [0x0000FF, 0x00FF00, 0x9900FF, 0xFFFF00, 0xFF0000,
  0xFFFF00, 0x9900FF, 0x00FF00,0x000000];//blue, green, purple, yellow, red,
  yellow, purple, green, black
var alphas:Array = [1, 1, 1, 1, 1, 1, 1, 1, 1];
var ratios:Array = [16, 32, 64, 96, 128, 160, 192, 224, 235];

// 创建滤镜对象
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(8, 225,
  colors, alphas, ratios, 16, 16, 1.3, 2, "inner", false);

// 将滤镜应用于方形影片剪辑
square_mc.filters = [gradientBevel];
```

应用渐变斜角滤镜

使用 GradientBevelFilter 类，您可以在 Flash 中对对象应用渐变斜角效果。渐变斜角是位于对象外部、内部或顶部的使用渐变色增强的有斜面的边缘。有斜面的边缘可使对象具有三维外观，并且可以得到如下图所示的彩色结果。



有关此滤镜的更多信息，请参见《ActionScript 2.0 语言参考》中的 GradientBevelFilter (flash.filters.GradientBevelFilter)。

下面的过程使用 Drawing API 在舞台上绘制方形，并对该形状应用渐变斜角滤镜。

使用渐变斜角滤镜：

1. 创建一个新的 Flash 文档，并将它保存为 **gradientbevel.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.GradientBevelFilter;
var shapeClip:MovieClip = this.createEmptyMovieClip("shape_mc", 1);
with (shapeClip) {
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(200, 0);
  lineTo(200, 200);
  lineTo(0, 200);
  lineTo(0, 0);
  endFill();
}
shapeClip._x = (Stage.width - shapeClip._width) / 2;
shapeClip._y = (Stage.height - shapeClip._height) / 2;
var colors:Array = new Array(0xFFFFFF, 0xCCCCCC, 0x000000);
var alphas:Array = new Array(1, 0, 1);
var ratios:Array = new Array(0, 128, 255);
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(10, 45,
  colors, alphas, ratios, 4, 4, 5, 3);
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  gradientBevel.strength++;
  shapeClip.filters = [gradientBevel];
};
mouseListener.onMouseMove = function() {
  gradientBevel.blurX = (_xmouse / Stage.width) * 255;
  gradientBevel.blurY = (_ymouse / Stage.height) * 255;
  shapeClip.filters = [gradientBevel];
};
Mouse.addListener(mouseListener);
```

   此代码使用 Drawing API 在舞台上绘制方形，该形状位于舞台中央。当您沿舞台移动鼠标指针时，沿 x 轴和 y 轴的模糊量将增加或减少。当您向舞台的左侧移动指针时，水平模糊量将减少。当您向舞台的右侧移动指针时，模糊量将增加。同样，指针在舞台上的位置越高，则沿 y 轴的模糊量越小。

3. 选择 “控制” > “测试影片”，以测试文档并查看结果。

使用颜色矩阵滤镜

ColorMatrixFilter 类使您可以将 4 x 5 矩阵转换应用于输入图像上的每个像素的 ARGB 颜色和 Alpha 值，以产生具有一组新的 ARGB 颜色和 Alpha 值的结果。此滤镜允许色相（明显的颜色或阴影）旋转、饱和度（特定色相的浓度）更改、Alpha 的亮度（亮度或颜色浓度），以及其它各种效果。而且，可以为这些滤镜实现动画效果，以便在您的应用程序中形成特定的效果。

参考	可以将颜色矩阵滤镜应用于位图和影片剪辑实例。

有关颜色矩阵滤镜的更多信息，请参见 《ActionScript 2.0 语言参考》中的 ColorMatrixFilter (flash.filters.ColorMatrixFilter)。

可以使用颜色矩阵滤镜来修改实例的亮度，如下面的示例所示。

增加影片剪辑的亮度：

1. 创建一个新的 Flash 文档，并将它保存为 **brightness.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.ColorMatrixFilter;
System.security.allowDomain("http://www.helpexamples.com/");
var mcl_obj:Object = new Object();
mcl_obj.onLoadInit = function(target_mc:MovieClip):Void {
  var myElements_array:Array = [1, 0, 0, 0, 100,
       0, 1, 0, 0, 100,
       0, 0, 1, 0, 100,
       0, 0, 0, 1, 0];
  var myColorMatrix_filter:ColorMatrixFilter = new
  ColorMatrixFilter(myElements_array);
  target_mc.filters = [myColorMatrix_filter];
}
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mcl_obj);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image2.jpg",
  img_mc);
```

   此代码通过使用 MovieClipLoader 实例动态加载 JPEG 图像。图像加载完成并放置在舞台上之后，通过使用颜色矩阵滤镜将实例的亮度设置为 100%。

3. 选择 “控制” > “测试影片” 来测试该文档。

还可以通过结合使用 Tween 类和 ColorMatrixFilter 类来创建动画的亮度效果，如下面的过程所示。

使用 Tween 类为实例的亮度级别添加动画效果：

1. 创建一个新的 Flash 文档，并将它保存为 **brightnesstween.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.ColorMatrixFilter;
import mx.transitions.Tween;
import mx.transitions.easing.*;
System.security.allowDomain("http://www.helpexamples.com");
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip):Void {
  // 使影片剪辑实例在舞台上居中
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
  target_mc.watch("brightness", brightnessWatcher, target_mc);
  // 为 target_mc 影片剪辑添加动画效果，使其亮度在 -100 到 +100 之间变化
  var t:Object = new Tween(target_mc, "brightness", Elastic.easeOut, 100,
  -100, 3, true);
  t.onMotionFinished = function() {
    this.yoyo();
  };
};
this.createEmptyMovieClip("img_mc", 10);
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);

function brightnessWatcher(prop:String, oldVal:Number, newVal:Number,
  target_mc:MovieClip):Number {
  var brightness_array:Array = [1, 0, 0, 0, newVal,
       0, 1, 0, 0, newVal,
       0, 0, 1, 0, newVal,
       0, 0, 0, 1, 0];
  target_mc.filters = [new ColorMatrixFilter(brightness_array)];
  return newVal;
};
```

第一部分代码使用 MovieClipLoader 类将 JPEG 图像加载到舞台上。图像加载完成后，将该图像重新定位在舞台的中央。然后使用 Tween 类为图像亮度级别添加动画效果。若要为亮度添加动画效果，可以使用 `Object.watch()` 方法，该方法可以注册指定的 ActionScript 对象属性更改时将要启动的事件处理函数。无论何时某些 ActionScript 尝试设置 `target_mc` 实例的自定义亮度属性时，都可以调用 `brightnessWatcher` 函数。自定义 `brightnessWatcher` 函数创建一个新数组，该数组使用颜色矩阵滤镜将目标图像的亮度设置为指定值。

3. 选择 “控制” > “测试影片” 来测试该文档。

图像加载并放置在舞台上之后，图像的亮度将在 -100 和 100 之间变化。在完成亮度补间后，将使用 `Tween.yoyo()` 方法使动画反向，从而为补间实现不间断的动画。

使用卷积滤镜

ConvolutionFilter 类应用矩阵盘绕滤镜效果。盘绕将指定的源图像的像素与相邻像素合并以生成图像。使用盘绕滤镜可以实现各种图像操作，其中包括模糊、边缘检测、锐化、浮雕以及斜角效果。

典型	可以将此滤镜应用于位图和影片剪辑实例。

矩阵盘绕基于一个 n x m 矩阵，该矩阵说明输入图像中的给定像素值如何与其相邻的像素值合并以生成最终的像素值。每个结果像素通过将矩阵应用到相应的源像素及其相邻像素来确定。

只有通过 ActionScript 才能使用此滤镜。有关此滤镜的更多信息，请参见《ActionScript 2.0 语言参考》中的 ConvolutionFilter (flash.filters.ConvolutionFilter)。

下面的过程将盘绕滤镜应用于动态加载的 JPEG 图像。

使用盘绕滤镜修改图像颜色：

1. 创建一个新的 Flash 文档，并将它保存为 **convolution.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.ConvolutionFilter;
import flash.display.BitmapData;

this.createEmptyMovieClip("shape_mc", 1);
shape_mc.createEmptyMovieClip("holder_mc", 1);
var imageLoader:MovieClipLoader = new MovieClipLoader();
imageLoader.loadClip("http://www.helpexamples.com/flash/images/
  image1.jpg", shape_mc.holder_mc);
var matrixArr:Array = [1, 4, 6, 4, 1, 4, 16, 24, 16, 4, 16, 6, 24, 36, 24,
  6, 4, 16, 24, 16, 4, 1, 4, 6, 4, 1];
var convolution:ConvolutionFilter = new ConvolutionFilter(5, 5,
  matrixArr);
shape_mc.filters = [convolution];

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  convolution.divisor = (_xmouse / Stage.width) * 271;
  convolution.bias = (_ymouse / Stage.height) * 100;
  shape_mc.filters = [convolution];
};
Mouse.addListener(mouseListener);
```

前面的代码分为三个部分。第一部分导入两个类：ConvolutionFilter 和 BitmapData。第二部分创建一个嵌套的影片剪辑并使用影片剪辑加载器对象将图像加载到嵌套的影片剪辑中。将创建一个盘绕滤镜对象并将其应用于 shape_mc 影片剪辑。最后一部分代码定义一个鼠标侦听器对象，它根据鼠标指针的当前位置修改盘绕滤镜的 divisor 和 bias 属性，并将盘绕滤镜重新应用于 shape_mc 影片剪辑。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

沿舞台的 x 轴移动鼠标指针将修改滤镜的除数，而沿舞台的 y 轴移动鼠标指针则将修改滤镜的偏差。

使用置换图滤镜

DisplacementMapFilter 类使用来自指定 BitmapData 对象的像素值 （称为置换图图像）来对舞台上的实例 （例如影片剪辑实例或位图数据实例）执行置换。您可以使用此滤镜在指定的实例上获得扭曲或斑点效果。

只有通过 ActionScript 才能使用此滤镜。有关此滤镜的更多信息，请参见 《ActionScript 2.0 语言参考》中的 DisplacementMapFilter (flash.filters.DisplacementMapFilter)。

下面的过程加载一个 JPEG 图像，并对其应用置换图滤镜，从而使图像扭曲。当用户移动鼠标时，将重新生成置换图。

使用置换图滤镜使图像扭曲：

1. 创建一个新的 Flash 文档，并将它保存为 **displacement.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.display.BitmapData;

var perlinBmp:BitmapData;
var displacementMap:DisplacementMapFilter;
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip):Void  {
     target_mc._x = (Stage.width - target_mc._width) / 2;
     target_mc._y = (Stage.height - target_mc._height) / 2;
     perlinBmp = new BitmapData(target_mc._width, target_mc._height);
     perlinBmp.perlinNoise(target_mc._width, target_mc._height, 10,
  Math.round(Math.random() * 100000), false, true, 1, false);
     displacementMap = new DisplacementMapFilter(perlinBmp, new Point(0,
  0), 1, 1, 100, 100, "color");
     shapeClip.filters = [displacementMap];
};
var shapeClip:MovieClip = this.createEmptyMovieClip("shapeClip", 1);
shapeClip.createEmptyMovieClip("holderClip", 1);
var imageLoader:MovieClipLoader = new MovieClipLoader();
imageLoader.addListener(mclListener);
imageLoader.loadClip("http://www.helpexamples.com/flash/images/
  image1.jpg", shapeClip.holderClip);

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
```

```
    perlinBmp.perlinNoise(shapeClip._width, shapeClip._height, 10,
    Math.round(Math.random() * 100000), false, true, 1, false);
    shapeClip.filters = [displacementMap];
};
Mouse.addListener(mouseListener);
```

此代码加载一个 JPEG 图像，并将其置于舞台上。图像加载完成后，此代码将创建一个 BitmapData 实例，并使用 `perlinNoise()` 方法以随机放置的像素对其进行填充。BitmapData 实例传递给置换图滤镜，后者应用于图像并使图像外观扭曲。

3. 选择 “控制” > “测试影片” 来测试该文档。

在舞台周围移动鼠标指针，以通过调用 `perlinNoise()` 方法重新创建置换图，这将更改 JPEG 图像的外观。

使用代码处理滤镜效果

Flash Basic 8 和 Flash Professional 8 允许您向影片剪辑、文本字段和舞台上的按钮中动态添加各种滤镜，而不必在 Flash Professional 8 创作环境中添加滤镜（使用属性检查器中的 “滤镜” 选项卡）。在播放期间添加和操作滤镜时，可以添加逼真的阴影、模糊和发光效果，以对鼠标移动或用户事件做出反应。

有关如何使用代码操作滤镜的示例，请参见以下主题：

- 第 474 页的 “调整滤镜属性”
- 第 475 页的 “使用 ActionScript 为滤镜添加动画效果”
- 第 476 页的 “使用 clone() 方法”

调整滤镜属性

应用于对象的滤镜数组可以通过标准 ActionScript 调用使用 `MovieClip.filters` 属性来访问。此过程将返回一个数组，其中包含当前与 MovieClip 关联的每个滤镜对象。每个滤镜都有该滤镜特定的一组属性。滤镜可以像数组对象那样进行访问和修改，但使用 `filters` 属性获取和设置滤镜将返回滤镜对象的副本，而不是引用。

设置 `filters` 属性将复制传入的 filters 数组，而不将其存储为引用。获取 filters 属性时，它将返回该数组的新副本。使用这种方法的一个负面影响是，下面的代码将无法正常工作：

```
// 无法正常工作
my_mc.filters[0].blurX = 20;
```

由于前面的代码片断返回 filters 数组的副本，因此代码将修改副本而不是原始数组。为了修改 blurX 属性，需要使用下面的 ActionScript 代码：

```
// 工作
var filterArray:Array = my_mc.filters;
filterArray[0].blurX = 20;
my_mc.filters = filterArray;
```

下面的过程根据鼠标指针在舞台上的当前位置使图像模糊。只要鼠标指针水平或垂直移动，模糊滤镜的 `blurX` 和 `blurY` 属性就会相应地被修改。

调整影片剪辑的滤镜属性：

1. 创建一个新的 Flash 文档，并将它保存为 **adjustfilter.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.BlurFilter;

this.createEmptyMovieClip("holder_mc", 10);
holder_mc.createEmptyMovieClip("img_mc", 20);
holder_mc.img_mc.loadMovie("http://www.helpexamples.com/flash/images/
  image2.jpg");
holder_mc.filters = [new BlurFilter(10, 10, 2)];
holder_mc._x = 75;
holder_mc._y = 75;

holder_mc.onMouseMove = function() {
  var tempFilter:BlurFilter = holder_mc.filters[0];
  tempFilter.blurX = Math.floor((_xmouse / Stage.width) * 255);
  tempFilter.blurY = Math.floor((_ymouse / Stage.height) * 255);
  holder_mc.filters = [tempFilter];
};
```

前面的代码分为三部分。第一部分导入 flash.filters.BlurFilter 类，这样引用 BlurFilter 类时便无需使用完全限定类名。第二部分代码创建一些影片剪辑，并将图像加载到一个嵌套剪辑中。第三部分代码响应鼠标在舞台上的移动，并相应调整模糊效果。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

   沿 x 轴移动鼠标指针将修改模糊滤镜的 `blurX` 属性。沿 y 轴移动鼠标指针将修改模糊滤镜的 `blurY` 属性。鼠标指针距离舞台左上角越近，应用于影片剪辑的模糊越少。

使用 ActionScript 为滤镜添加动画效果

可以使用 ActionScript （例如 Tween 类）在运行时为滤镜添加动画效果，这样可对 Flash 应用程序应用有趣的动画效果。

在下面的示例中，您将看到如何结合使用 BlurFilter 和 Tween 类来创建有动画的模糊效果，在运行时将在 0 和 10 之间修改模糊滤镜的值。

使用 Tween 类为模糊添加动画效果：

1. 创建一个新的 Flash 文档，并将它保存为 **animatedfilter.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.filters.BlurFilter;
import mx.transitions.Tween;
import mx.transitions.easing.*;

this.createEmptyMovieClip("holder_mc", 10);
holder_mc.createEmptyMovieClip("img_mc", 20);

var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
  var myTween:Tween = new Tween(target_mc, "blur", Strong.easeInOut, 0,
  20, 3, true);
  myTween.onMotionChanged = function() {
    target_mc._parent.filters = [new BlurFilter(target_mc.blur,
  target_mc.blur, 1)];
  };
  myTween.onMotionFinished = function() {
    myTween.yoyo();
  }
};
var my_mcl:MovieClipLoader = new MovieClipLoader();
my_mcl.addListener(mclListener);
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  holder_mc.img_mc);
```

前面的代码分为三部分。第一部分导入所需的类和包。第二部分创建用于加载图像的嵌套影片剪辑，并对该影片剪辑应用滤镜。最后一部分代码创建一个新的 MovieClipLoader 实例，并为该影片剪辑加载器创建一个侦听器。该侦听器对象定义一个事件处理函数 onLoadInit，一旦图像成功加载并且可在舞台上使用，便启动该函数。首先，将图像重新定位在舞台中央，然后会创建一个新的 Tween 对象，它将为影片剪辑添加动画效果，并应用 0 和 10 的模糊滤镜。

3. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

使用 clone() 方法

每个滤镜类中的 clone() 方法都会返回一个新的滤镜实例，它与原始滤镜实例具有相同的属性。当您使用滤镜时，可能要制作滤镜的副本，为此需要使用 clone() 方法复制滤镜。如果不使用克隆方法来复制滤镜，Flash 将只创建对原始滤镜的引用。如果 Flash 创建对原始滤镜的引用，则对复制的滤镜所做的任何更改也将同时修改原始滤镜对象。

下面的过程创建 DropShadowFilter (greenDropShadow) 的一个新实例，调用 clone() 方法来复制绿色投影滤镜，并将新滤镜保存为 redDropShadow。克隆的滤镜设置一个新的投影颜色，新旧两个滤镜都将应用于舞台上的 flower_mc 影片剪辑实例中。

使用克隆方法：

1. 创建一个新的 Flash 文档并将它命名为 **clone.fla**。
2. 在舞台上创建一个影片剪辑。
3. 选择影片剪辑实例，然后在属性检查器的 “实例名称” 文本框中键入 **flower_mc**。
4. 在时间轴中选择第 1 帧，在 “动作” 面板中，添加下面的代码：

```
import flash.filters.DropShadowFilter;
var greenDropShadow:DropShadowFilter = new DropShadowFilter();
greenDropShadow.color = 0x00FF00; // green
var redDropShadow:DropShadowFilter = greenDropShadow.clone();
redDropShadow.color = 0xFF0000; // red
flower_mc.filters = [greenDropShadow, redDropShadow];
```

前面的代码创建投影滤镜的一个新实例，并将其命名为 greenDropShadow。将使用 DropShadowFilter.clone() 方法复制绿色投影对象，并创建一个名为 redDropShadow 的新滤镜对象。绿色和红色的投影滤镜都将应用于舞台上的 flower_mc 影片剪辑实例中。如果没有调用 clone() 方法，则两个投影滤镜都将显示为红色。出现此现象的原因在于，由于红色投影包含对绿色投影的引用，所以设置 redDropShadow.color 属性将同时更改红色投影和绿色投影对象。

5. 选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

该滤镜将被复制并克隆，并且两个滤镜都将应用于 `flower_mc` 实例。

有关 `clone()` 方法的更多信息，请参见 《ActionScript 2.0 语言参考》中的 clone（DropShadowFilter.clone 方法）。有关信息，还可以参见任何滤镜类的 `clone()` 方法。

使用 BitmapData 类创建位图

BitmapData 类允许您在运行时创建任意大小的透明或不透明位图图像并采用各种方式操作这些图像。当您使用 ActionScript 直接操作 BitmapData 实例时，可以创建非常复杂的图像，而不会带来不断从 Flash Player 中的矢量数据中重绘内容的开销。BitmapData 类的方法支持多种无法通过通用 Flash 工作区的 “滤镜” 选项卡获得的效果。

BitmapData 对象包含像素数据的数组。此数据既可以表示完全不透明的位图，也可以表示包含 Alpha 通道数据的透明位图。以上任一类型的 BitmapData 对象都作为 32 位整数的缓冲区进行存储。每个 32 位整数确定位图中单个像素的属性。每个 32 位整数都是四个 8 位通道 值（从 0 到 255）的组合，这些值描述像素的 Alpha 透明度值和红、绿、蓝 (ARGB) 值。

有关使用包的信息，请参见第 447 页的 “使用滤镜包”。

提供了使用 BitmapData 类操作图像的范例源文件。在硬盘上的 Samples 文件夹中可以找到名为 BitmapData.fla 的文件。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\BitmapData。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/BitmapData。

下面的过程将一个 JPEG 图像动态加载到舞台上，并使用 BitmapData 类创建杂点效果，类似于电视中的静电干扰效果。每隔 100 毫秒 （一秒的 1/10），都将以随机模式重新绘制该杂点效果。沿 x 轴和 y 轴移动鼠标指针将影响每个间隔内绘制的静电干扰量。

使用 BitmapData 类创建杂点效果：

1. 创建一个新的 Flash 文档，并将它保存为 **noise.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import flash.display.BitmapData;
this.createTextField("status_txt", 90, 0, 0, 100, 20);
status_txt.selectable = false;
status_txt.background = 0xFFFFFF;
status_txt.autoSize = "left";
function onMouseMove() {
  status_txt._x = _xmouse;
  status_txt._y = _ymouse-20;
  updateAfterEvent();
}
```

```
this.createEmptyMovieClip("img_mc", 10);
img_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
var noiseBmp:BitmapData = new BitmapData(Stage.width, Stage.height,
  true);
this.attachBitmap(noiseBmp, 20);
setInterval(updateNoise, 100);
var grayScale:Boolean = true;
function updateNoise():Void {
  var low:Number = 30 * _xmouse / Stage.width;
  var high:Number = 200 * _ymouse / Stage.height;
  status_txt.text = "low:" + Math.round(low) + ", high:" +
  Math.round(high);
  noiseBmp.noise(Math.round(Math.random() * 100000), low, high, 8, true);
}
```

此代码使用实例名称 `status_txt` 创建一个文本字段，该字段跟随鼠标指针并为 `noise()` 方法显示 `high` 和 `low` 参数的当前值。`setInterval()` 函数可更改杂点效果，杂点效果通过连续调用 `updateNoise()` 函数每隔 100 毫秒（一秒的 1/10）更新一次。`noise()` 方法的 `high` 和 `low` 参数通过计算指针在舞台上的当前位置来确定。

3. 选择“控制”>“测试影片”来测试该文档。

沿 x 轴移动鼠标指针将影响 `low` 参数；而沿 y 轴移动鼠标指针将影响 `high` 参数。

BitmapData 类还允许您通过结合使用 `perlinNoise()` 方法效果和置换图滤镜来使动态加载的图像扭曲。下面的过程将显示这一效果。

将置换图滤镜应用于图像：

1. 创建一个新的 Flash 文档，并将它保存为 **displacement.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
// 导入类。
import flash.filters.DisplacementMapFilter;
import flash.display.BitmapData;
import flash.geom.Point;
// 创建剪辑和嵌套剪辑。
var shapeClip:MovieClip = this.createEmptyMovieClip("shapeClip", 1);
shapeClip.createEmptyMovieClip("holderClip", 1);
// 加载 JPEG。
var imageLoader:MovieClipLoader = new MovieClipLoader();
imageLoader.loadClip("http://www.helpexamples.com/flash/images/
  image4.jpg", shapeClip.holderClip);
// 创建 BitmapData 实例。
var perlinBmp:BitmapData = new BitmapData(Stage.width, Stage.height);
perlinBmp.perlinNoise(Stage.width, Stage.height, 10,
  Math.round(Math.random() * 100000), false, true, 1, false);
// 创建并应用置换图滤镜。
var displacementMap:DisplacementMapFilter = new
  DisplacementMapFilter(perlinBmp, new Point(0, 0), 1, 1, 100, 100,
  "color", 1);
```

```
shapeClip.filters = [displacementMap];
// 创建并应用侦听器。
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  perlinBmp.perlinNoise(Stage.width, Stage.height, 10,
   Math.round(Math.random() * 100000), false, true, 1, false);
  shapeClip.filters = [displacementMap];
}
Mouse.addListener(mouseListener);
```

此代码示例由五个逻辑部分组成。第一部分导入示例必需的类。第二个代码块创建一个嵌套的影片剪辑并从远程服务器加载一个 JPEG 图像。第三个代码块创建一个名为 `perlinBmp` 的新 BitmapData 实例，该实例与舞台尺寸大小相同。`perlinBmp` 实例包含 Perlin 杂点效果的结果，稍后会将该实例用作置换图滤镜的参数。第四个代码块将创建置换图滤镜效果并将其应用于前面创建的动态加载的图像。第五个也是最后一个代码块为鼠标创建一个侦听器，只要用户移动鼠标指针，就将重新生成置换图滤镜使用的 Perlin 杂点。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

关于混合模式

通过使用 Flash 工作区 (Flash Professional 8) 或 ActionScript（Flash Basic 8 和 Flash Professional 8）可以对影片剪辑对象应用混合模式。在运行时，多个图形将合并为一个形状。因此，不能对不同的图形元件应用不同的混合模式。

有关使用 ActionScript 应用混合模式的更多信息，请参见第 480 页的 “应用混合模式”。

混合模式涉及将一个图像（基图像）的颜色与另一个图像（混合图像）的颜色进行组合以生成第三个图像。图像中的每个像素值都会被使用其它图像的对应像素值进行处理，以便在结果的同一位置生成一个像素值。

`MovieClip.blendMode` 属性支持以下混合模式：

add　通常用于在两个图像之间创建动画的亮度分解效果。

alpha　通常用于在背景中应用前景的透明度。

darken　通常用于重叠类型。

difference　通常用于创建更多的变化颜色。

erase　通常用于使用前景 Alpha 剪掉（擦除）背景的部分。

hardlight　通常用于创建阴影效果。

invert　用于反转背景。

layer　用于强制为特定影片剪辑的预构成创建临时缓冲区。

lighten　通常用于重叠类型。

multiply　通常用于创建阴影和深度效果。

normal　用于指定混合图像的像素值覆盖基本图像的像素值。

overlay　通常用于创建阴影效果。

screen　通常用于创建加亮和镜头眩光。

subtract　通常用于在两个图像之间创建动画的变暗分解效果。

应用混合模式

下面的过程加载动态图像，并允许您通过从舞台上的组合框中选择混合模式来对该图像应用不同的混合模式。

对图像应用不同的混合模式：

1. 创建一个新的 Flash 文档，并将它保存为 **blendmodes.fla**。
2. 将一个 ComboBox 组件实例拖动到舞台上，并为其指定实例名称 **blendMode_cb**。
3. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var blendMode_dp:Array = new Array();
blendMode_dp.push({data:"add", label:"add"});
blendMode_dp.push({data:"alpha", label:"alpha"});
blendMode_dp.push({data:"darken", label:"darken"});
blendMode_dp.push({data:"difference", label:"difference"});
blendMode_dp.push({data:"erase", label:"erase"});
blendMode_dp.push({data:"hardlight", label:"hardlight"});
blendMode_dp.push({data:"invert", label:"invert"});
blendMode_dp.push({data:"layer", label:"layer"});
blendMode_dp.push({data:"lighten", label:"lighten"});
blendMode_dp.push({data:"multiply", label:"multiply"});
blendMode_dp.push({data:"normal", label:"normal"});
blendMode_dp.push({data:"overlay", label:"overlay"});
blendMode_dp.push({data:"screen", label:"screen"});
blendMode_dp.push({data:"subtract", label:"subtract"});
blendMode_cb.dataProvider = blendMode_dp;

var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  var blendModeClip:MovieClip =
  target_mc.createEmptyMovieClip("blendModeType_mc", 20);
  with (blendModeClip) {
    beginFill(0x999999);
    moveTo(0, 0);
    lineTo(target_mc._width / 2, 0);
    lineTo(target_mc._width / 2, target_mc._height);
    lineTo(0, target_mc._height);
    lineTo(0, 0);
    endFill();
```

```
    }
    target_mc._x = (Stage.width - target_mc._width) / 2;
    target_mc._y = (Stage.height - target_mc._height) / 2;
    blendModeClip.blendMode = blendMode_cb.value;
};
this.createEmptyMovieClip("img_mc", 10);
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
    img_mc);

function cbListener(eventObj:Object):Void {
    img_mc.blendModeType_mc.blendMode = eventObj.target.value;
}
blendMode_cb.addEventListener("change", cbListener);
```

此 ActionScript 代码以每种类型的混合模式来填充组合框，因此用户可在动态加载的图像中查看每种效果。创建了一个侦听器对象，用于和 MovieClipLoader 实例一起使用。该侦听器对象定义一个事件侦听器 `onLoadInit`，当图像完全下载并被 Flash 初始化时，将调用该侦听器。该事件侦听器将创建一个名为 `blendModeType_mc` 的新影片剪辑，并使用 Drawing API 在图像的左半部分上绘制一个矩形形状。然后，当前为 ComboBox 实例选定的混合模式将应用于 `blendModeType_mc` 影片剪辑。

其余代码设置了 MovieClipLoader 实例，该实例负责将指定的图像加载到舞台上的影片剪辑中。最后，为 `blendMode_cb` ComboBox 实例定义一个侦听器，当从 ComboBox 实例中选择一个新项时，该实例便将应用选定的混合模式。

4. 选择 “控制” > “测试影片” 来测试该文档。

关于操作顺序

下面的列表是为影片剪辑实例附加或执行滤镜数组、混合模式、颜色转换和遮罩层的顺序：

1. 影片剪辑的位图将从矢量内容中更新 （`cacheAsBitmap` 属性设置为 `true`）。
2. 如果使用 `setMask()` 方法，并且遮罩有位图缓存，则 Flash 将在两个图像之间执行 Alpha 混合。
3. 然后将应用滤镜 （模糊、投影、发光等等）。
4. 如果使用 ColorTransform 类，则将执行颜色转换操作，并缓存为位图结果。
5. 如果应用混合模式，则将执行混合 （使用矢量渲染器）。
6. 如果应用外部遮罩层，则这些层将执行遮罩 （使用矢量渲染器）。

使用 ActionScript 绘画

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 SWF 文件中绘制响应事件的形状。下面是 MovieClip 类绘画方法：

- `beginFill()`
- `beginGradientFill()`
- `clear()`
- `curveTo()`
- `endFill()`
- `lineTo()`
- `lineStyle()`
- `moveTo()`

您可以在任何一个影片剪辑中使用绘制方法。不过，如果对在创作模式下创建的影片剪辑使用绘制方法，则该绘制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制方法绘制的内容上面。

您可以将使用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。

有关使用 ActionScript 绘画的更多信息，请参见以下主题：

- 第 483 页的 “使用绘制方法绘制直线、曲线和形状”
- 第 485 页的 “绘制特定形状”
- 第 488 页的 “使用复杂的渐变填充”
- 第 489 页的 “使用线条样式”
- 第 495 页的 “使用 Drawing API 方法和实现脚本动画”

在硬盘上的 Samples 文件夹中可以找到范例源文件 drawingapi.fla，该文件演示了如何在 Flash 应用程序中使用 Drawing API。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DrawingAPI。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DrawingAPI。

使用绘制方法绘制直线、曲线和形状

可以使用 Flash Drawing API 在运行时在舞台上动态创建形状。可以使用这些形状来动态遮罩内容，对其应用滤镜，或者在舞台上为其实现动画效果。还可以使用 Drawing API 创建各种绘画工具，从而允许用户使用鼠标或键盘在 SWF 文件中绘制形状。

绘制直线：

1. 创建一个新的 Flash 文档，并将它保存为 **line.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("line_mc", 10);
line_mc.lineStyle(1, 0x000000, 100);
line_mc.moveTo(0, 0);
line_mc.lineTo(200, 100);
line_mc._x = 100;
line_mc._y = 100;
```

   此代码在舞台上的 0,0 与 200,100 之间绘制一条直线。该直线的 `_x` 和 `_y` 坐标随后将进行修改，以将直线重新定位在舞台上的 100,100。

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该 SWF 文件进行测试。

若要绘制更复杂的形状，请继续调用 `MovieClip.lineTo()` 方法来绘制矩形、方形或椭圆，如以下过程所示。

绘制曲线：

1. 创建一个新的 Flash 文档，并将它保存为 **curve.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("circle_mc", 1);
with (circle_mc) {
  lineStyle(4, 0x000000, 100);
  beginFill(0xFF0000);
  moveTo(200, 300);
  curveTo(300, 300, 300, 200);
  curveTo(300, 100, 200, 100);
  curveTo(100, 100, 100, 200);
  curveTo(100, 300, 200, 300);
  endFill();
}
```

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

   此代码使用 Drawing API 在舞台上绘制一个圆形。该圆形只调用 `MovieClip.curveTo()` 方法四次，因此看上去可能有些扭曲。有关使用 Drawing API 创建圆形的另一个示例，请参见第 485 页的 “绘制特定形状”下的创建圆形的过程，其中的代码对 `MovieClip.curveTo()` 方法进行了八次调用，以绘制更逼真的圆形。

绘制三角形：

1. 创建一个新的 Flash 文档，并将它保存为 **triangle.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("triangle_mc", 1);
```

   此代码使用 `MovieClip.createEmptyMovieClip()` 方法在舞台上创建一个空的影片剪辑。新的影片剪辑是现有影片剪辑（在此例中为主时间轴）的子级。

3. 将下面的 ActionScript 添加到时间轴中的第 1 帧上，位于在上一步骤中添加的代码之后：

```
with (triangle_mc) {
  lineStyle(5, 0xFF00FF, 100);
  moveTo(200, 200);
  lineTo(300, 300);
  lineTo(100, 300);
  lineTo(200, 200);
}
```

   在此代码中，空的影片剪辑 (`triangle_mc`) 将调用绘画方法。此代码使用 5 像素紫色线绘制三角形，并且不进行填充。

4. 保存 Flash 文档，然后选择“控制”>“测试影片”对该 Flash 文档进行测试。

有关这些方法的详细信息，请参见《ActionScript 2.0 语言参考》中 `MovieClip` 中的相应条目。

在硬盘上的 Samples 文件夹中可以找到范例源文件 drawingapi.fla，该文件演示了如何在 Flash 应用程序中使用 Drawing API。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DrawingAPI。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DrawingAPI。

绘制特定形状

本节介绍如何创建一些更灵活的方法，以用来绘制更高级的形状，例如圆角矩形和圆形。

创建矩形：

1. 创建一个新的 Flash 文档，并将它保存为 **rect.fla**。
2. 将下面的 ActionScript 代码添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("rectangle_mc", 10);
rectangle_mc._x = 100;
rectangle_mc._y = 100;
drawRectangle(rectangle_mc, 240, 180, 0x99FF00, 100);
function drawRectangle(target_mc:MovieClip, boxWidth:Number,
  boxHeight:Number, fillColor:Number, fillAlpha:Number):Void {
  with (target_mc) {
    beginFill(fillColor, fillAlpha);
    moveTo(0, 0);
    lineTo(boxWidth, 0);
    lineTo(boxWidth, boxHeight);
    lineTo(0, boxHeight);
    lineTo(0, 0);
    endFill();
  }
}
```

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该 Flash 文档进行测试。

   Flash 在舞台上绘制一个简单的绿色矩形，并将其定位在 100,100。若要更改矩形的尺寸、其填充颜色或透明度，可以在对 `drawRectangle()` 方法的调用中更改这些值，而无需修改 `MovieClip.beginFill()` 方法的内容。

还可以使用 Drawing API 创建带有圆角的矩形，如下面的过程所示。

创建圆角矩形：

1. 创建一个新的 Flash 文档，并将它保存为 **roundrect.fla**。
2. 将下面的 ActionScript 代码添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("rectangle_mc", 10);
rectangle_mc._x = 100;
rectangle_mc._y = 100;
drawRoundedRectangle(rectangle_mc, 240, 180, 20, 0x99FF00, 100);
function drawRoundedRectangle(target_mc:MovieClip, boxWidth:Number,
  boxHeight:Number, cornerRadius:Number, fillColor:Number,
  fillAlpha:Number):Void {
  with (target_mc) {
    beginFill(fillColor, fillAlpha);
    moveTo(cornerRadius, 0);
    lineTo(boxWidth - cornerRadius, 0);
    curveTo(boxWidth, 0, boxWidth, cornerRadius);
```

```
        lineTo(boxWidth, cornerRadius);
        lineTo(boxWidth, boxHeight - cornerRadius);
        curveTo(boxWidth, boxHeight, boxWidth - cornerRadius, boxHeight);
        lineTo(boxWidth - cornerRadius, boxHeight);
        lineTo(cornerRadius, boxHeight);
        curveTo(0, boxHeight, 0, boxHeight - cornerRadius);
        lineTo(0, boxHeight - cornerRadius);
        lineTo(0, cornerRadius);
        curveTo(0, 0, cornerRadius, 0);
        lineTo(cornerRadius, 0);
        endFill();
    }
}
```

3. 保存 Flash 文档，然后选择“控制”>“测试影片”对该文档进行测试。

   舞台上将出现一个绿色矩形，其宽度和高度分别为 240 和 180 像素，并且带有 20 像素的圆角。通过使用 `MovieClip.createEmptyMovieClip()` 创建新影片剪辑并调用自定义的 `drawRoundedRectangle()` 函数可以创建多个圆角矩形实例。

使用 Drawing API 可以创建完美的圆形，如下面的过程所示。

创建圆形：

1. 创建一个新的 Flash 文档，并将它保存为 **circle2.fla**。
2. 将下面的 ActionScript 代码添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("circle_mc", 10);
circle_mc._x = 100;
circle_mc._y = 100;
drawCircle(circle_mc, 100, 0x99FF00, 100);

function drawCircle(target_mc:MovieClip, radius:Number, fillColor:Number,
  fillAlpha:Number):Void {
  var x:Number = radius;
  var y:Number = radius;
  with (target_mc) {
    beginFill(fillColor, fillAlpha);
    moveTo(x + radius, y);
    curveTo(radius + x, Math.tan(Math.PI / 8) * radius + y,
  Math.sin(Math.PI / 4) * radius + x, Math.sin(Math.PI / 4) * radius + y);
    curveTo(Math.tan(Math.PI / 8) * radius + x, radius + y, x, radius +
  y);
    curveTo(-Math.tan(Math.PI / 8) * radius + x, radius+ y, -
  Math.sin(Math.PI / 4) * radius + x, Math.sin(Math.PI / 4) * radius + y);
    curveTo(-radius + x, Math.tan(Math.PI / 8) * radius + y, -radius + x,
  y);
    curveTo(-radius + x, -Math.tan(Math.PI / 8) * radius + y, -
  Math.sin(Math.PI / 4) * radius + x, -Math.sin(Math.PI / 4) * radius +
  y);
```

```
    curveTo(-Math.tan(Math.PI / 8) * radius + x, -radius + y, x, -radius
  + y);
    curveTo(Math.tan(Math.PI / 8) * radius + x, -radius + y,
  Math.sin(Math.PI / 4) * radius + x, -Math.sin(Math.PI / 4) * radius +
  y);
    curveTo(radius + x, -Math.tan(Math.PI / 8) * radius + y, radius + x,
  y);
    endFill();
  }
}
```

3. 保存 Flash 文档，然后选择 “控制” > “测试影片”对该 SWF 文件进行测试。

此代码可创建比上一个圆形示例更复杂、更逼真的圆形。此示例不止调用了四次 curveTo() 方法，而是调用了八次 curveTo() 方法，使形状具有更圆滑的外观。

可以使用 Drawing API 来创建三角形，如下面的过程所示。

创建奇特的三角形：

1. 创建一个新的 Flash 文档，并将它保存为 **fancytriangle.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("triangle_mc", 10);
triangle_mc._x = 100;
triangle_mc._y = 100;
drawTriangle(triangle_mc, 100, 0x99FF00, 100);

function drawTriangle(target_mc:MovieClip, sideLength:Number,
  fillColor:Number, fillAlpha:Number):Void {
  var tHeight:Number = sideLength * Math.sqrt(3) / 2;
  with (target_mc) {
    beginFill(fillColor, fillAlpha);
    moveTo(sideLength / 2, 0);
    lineTo(sideLength, tHeight);
    lineTo(0, tHeight);
    lineTo(sideLength / 2, 0);
    endFill();
  }
}
```

Drawing API 在舞台上绘制一个等边三角形，并使用指定的填充颜色和 Alpha 量（透明度）来填充它。

3. 保存 Flash 文档，然后选择 “控制” > “测试影片”对该 Flash 文档进行测试。

在硬盘上的 Samples 文件夹中可以找到范例源文件 drawingapi.fla，该文件演示了如何在 Flash 应用程序中使用 Drawing API。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DrawingAPI。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DrawingAPI。

使用复杂的渐变填充

Flash Drawing API 支持渐变填充和实体填充。下面的过程在舞台上创建一个新的影片剪辑，并使用 Drawing API 创建一个方形，然后使用放射状红色和蓝色渐变来填充该方形。

创建复杂渐变：

1. 创建一个新的 Flash 文档，并将它保存为 **radialgradient.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("gradient_mc", 10);
var fillType:String = "radial";
var colors:Array = [0xFF0000, 0x0000FF];
var alphas:Array = [100, 100];
var ratios:Array = [0, 0xFF];
var matrix:Object = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};
var spreadMethod:String = "reflect";
var interpolationMethod:String = "linearRGB";
var focalPointRatio:Number = 0.9;
with (gradient_mc) {
  beginGradientFill(fillType, colors, alphas, ratios, matrix,
  spreadMethod, interpolationMethod, focalPointRatio);
  moveTo(100, 100);
  lineTo(100, 300);
  lineTo(300, 300);
  lineTo(300, 100);
  lineTo(100, 100);
  endFill();
}
```

   前面的 ActionScript 代码使用 Drawing API 在舞台上创建方形，然后调用 beginGradientFill() 方法以红色和蓝色圆形渐变来填充该方形。

3. 保存 Flash 文档，然后选择 “控制” > “测试影片”对该 Flash 文件进行测试。

使用线条样式

Flash Drawing API 允许您指定 Flash 在以后调用 MovieClip.lineTo() 和 MovieClip.curveTo() 时使用的线条样式，这些线条样式设置将持续保持，直到您使用不同的参数调用 MovieClip.lineStyle() 为止，如下所示：

```
lineStyle(thickness:Number, rgb:Number, alpha:Number, pixelHinting:Boolean,
  noScale:String, capsStyle:String, jointStyle:String, miterLimit:Number)
```

可以在绘制路径的中间调用 MovieClip.lineStyle() 来为路径中的不同线条段指定不同的样式。

有关使用 ActionScript 设置线条样式的更多信息，请参见以下各部分：

- 第 489 页的 “设置笔触和端点样式”
- 第 490 页的 “设置线条样式的参数”

设置笔触和端点样式

Flash 8 对线条绘制进行了一些改进。 Flash Player 8 中添加的新线条参数包括 pixelHinting、noScale、capsStyle、jointStyle 和 miterLimit。

下面的过程演示了 Flash Player 8 中三种新的端点样式之间的差别：none、round 和 square。

使用 ActionScript 设置端点样式：

1. 创建一个新的 Flash 文档，并将它保存为 **capstyle.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
// 设置网格影片剪辑。
this.createEmptyMovieClip("grid_mc", 50);
grid_mc.lineStyle(0, 0x999999, 100);
grid_mc.moveTo(50, 0);
grid_mc.lineTo(50, Stage.height);
grid_mc.moveTo(250, 0);
grid_mc.lineTo(250, Stage.height);
// 第 1 行 (capsStyle:round}
this.createEmptyMovieClip("line1_mc", 10);
with (line1_mc) {
  createTextField("label_txt", 1, 5, 10, 100, 20);
  label_txt.text = "round";
  lineStyle(20, 0x99FF00, 100, true, "none", "round", "miter", 0.8);
  moveTo(0, 0);
  lineTo(200, 0);
  _x = 50;
  _y = 50;
}
// 第 2 行 (capsStyle:square}
this.createEmptyMovieClip("line2_mc", 20);
```

```
with (line2_mc) {
  createTextField("label_txt", 1, 5, 10, 100, 20);
  label_txt.text = "square";
  lineStyle(20, 0x99FF00, 100, true, "none", "square", "miter", 0.8);
  moveTo(0, 0);
  lineTo(200, 0);
  _x = 50;
  _y = 150;
}
// 第 3 行 (capsStyle:none}
this.createEmptyMovieClip("line3_mc", 30);
with (line3_mc) {
  createTextField("label_txt", 1, 5, 10, 100, 20);
  label_txt.text = "none";
  lineStyle(20, 0x99FF00, 100, true, "none", "none", "miter", 0.8);
  moveTo(0, 0);
  lineTo(200, 0);
  _x = 50;
  _y = 250;
}
```

前面的代码动态创建四个影片剪辑，并使用 Drawing API 在舞台上创建一系列线条。第一个影片剪辑包括两条竖线，一条位于 x 轴上的 50 像素处，另一条位于 250 像素处。接下来的三个影片剪辑各自在舞台上绘制一条绿色线条，并将 capsStyle 设置为 round、square 或 none。

3. 选择“控制”>“测试影片”来测试该文档。

运行时，舞台上将出现不同的端点样式。

设置线条样式的参数

可以设置线条样式的参数，以更改笔触的外观。可以使用参数来更改线条样式的粗细、颜色、Alpha、比例因子和其它属性。

设置线条粗细

通过 `MovieClip.lineStyle()` 方法的 `thickness` 参数可以用点数指定所绘制线条的粗细。可以绘制粗细在 0 到 255 点宽之间的任意线条，但将粗细设置为 0 将创建所谓的极细 线条，其中无论如何缩放 SWF 文件或调整其大小，笔触始终为 1 像素。

下面的过程演示标准的 1 像素粗细线条与极细线条之间的差别。

创建极细笔触：

1. 创建一个新的 Flash 文档，并将它保存为 **hairline.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
this.createEmptyMovieClip("drawing_mc", 10);
// 创建一条红色的极细线条
drawing_mc.lineStyle(0, 0xFF0000, 100);
drawing_mc.moveTo(0, 0);
drawing_mc.lineTo(200, 0);
drawing_mc.lineTo(200, 100);
// 创建一条蓝色的粗细为 1 像素的线条
drawing_mc.lineStyle(1, 0x0000FF, 100);
drawing_mc.lineTo(0, 100);
drawing_mc.lineTo(0, 0);
drawing_mc._x = 100;
drawing_mc._y = 100;
```

   前面的代码使用 Drawing API 在舞台上绘制两条线。第一条线为红色，粗细为 0，表示极细，第二条线为蓝色，粗细为 1 像素。

3. 保存 Flash 文档，然后选择 “控制” > “测试影片”对该 SWF 文件进行测试。

   最初，红色和蓝色线条看上去完全相同。如果在 SWF 文件中右击，并从上下文菜单中选择 “放大”，则红色线条始终显示为 1 像素宽的线条；但是，每当您在 SWF 文件中放大时，蓝色线条都将变粗。

设置线条颜色 (rgb)

通过 `lineStyle()` 方法中的第二个参数 `rgb` 可以将当前线条段的颜色作为数字来进行控制。默认情况下，Flash 将绘制黑色线条 (`#000000`)，但您可以使用 `0xRRGGBB` 语法通过设置新的十六进制颜色值来指定不同颜色。在此语法中，`RR` 是红色值 （在 `00` 和 `FF` 之间），`GG` 是绿色值 （`00` 到 `FF`），`BB` 是蓝色值 （`00` 到 `FF`）。

例如，可将红色线条表示为 `0xFF0000`，绿色线条表示为 `0x00FF00`，蓝色线条表示为 `0x0000FF`，紫色线条表示为 `0xFF00FF` （红色和蓝色），白色线条表示为 `#FFFFFF`，灰色线条表示为 `#999999`，等等。

设置线条 Alpha

通过 `lineStyle()` 方法中的第三个参数 `alpha` 可以控制线条的透明度 (alpha) 级别。透明度是在 0 到 100 之间的一个值，其中 0 表示完全透明的线条，而 100 表示完全不透明（可见）的线条。

设置线条像素提示 (pixelHinting)

用于 strokes 参数的像素提示 `pixelHinting` 表示线条和曲线锚记在完全像素中设置。对于任何笔触粗细，笔触都在完全像素中，这表示您从来看不到模糊的垂直或水平线条。将 `pixelHinting` 参数设置为布尔值 （`true` 或 `false`）。

设置线条缩放 (noScale)

通过使用 String 值设置 noScale 参数，可以为线条指定缩放模式。可以以水平模式或垂直模式使用不可缩放的笔触，缩放线条 （普通），或不使用缩放。

提示	在用户放大时为用户界面元素启用缩放将很有用，但是只垂直或水平缩放影片剪辑时，则不能这样做。

可以使用四种不同的模式之一指定何时应进行缩放，何时不应进行缩放。下面是 noScale 属性的可能值：

normal　始终缩放粗细 （默认）。

vertical　如果对象垂直缩放，则不缩放粗细。

horizontal　如果对象水平缩放，则不缩放粗细。

none　从不缩放粗细。

设置线条端点 (capsStyle) 和联接点 (jointStyle)

可以为 capsStyle 参数设置三种类型的端点样式：

- round （默认）
- square
- none

下面的过程演示了这三种端点样式之间的差别。当您对 SWF 文件进行测试时，舞台上将出现每种端点样式的可视表示形式。

设置不同的端点样式：

1. 创建一个新的 Flash 文档，并将它保存为 **capsstyle2.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var lineLength:Number = 100;
// round
this.createEmptyMovieClip("round_mc", 10);
round_mc.lineStyle(20, 0xFF0000, 100, true, "none", "round");
round_mc.moveTo(0, 0);
round_mc.lineTo(lineLength, 0);
round_mc.lineStyle(0, 0x000000);
round_mc.moveTo(0, 0);
round_mc.lineTo(lineLength, 0);
round_mc._x = 50;
round_mc._y = 50;
var lbl:TextField = round_mc.createTextField("label_txt", 10, 0, 10,
  lineLength, 20);
lbl.text = "round";
var lineLength:Number = 100;
// square
```

```
this.createEmptyMovieClip("square_mc", 20);
square_mc.lineStyle(20, 0xFF0000, 100, true, "none", "square");
square_mc.moveTo(0, 0);
square_mc.lineTo(lineLength, 0);
square_mc.lineStyle(0, 0x000000);
square_mc.moveTo(0, 0);
square_mc.lineTo(lineLength, 0);
square_mc._x = 200;
square_mc._y = 50;
var lbl:TextField = square_mc.createTextField("label_txt", 10, 0, 10,
  lineLength, 20);
lbl.text = "square";
// none
this.createEmptyMovieClip("none_mc", 30);
none_mc.lineStyle(20, 0xFF0000, 100, true, "none", "none");
none_mc.moveTo(0, 0);
none_mc.lineTo(lineLength, 0);
none_mc.lineStyle(0, 0x000000);
none_mc.moveTo(0, 0);
none_mc.lineTo(lineLength, 0);
none_mc._x = 350;
none_mc._y = 50;
var lbl:TextField = none_mc.createTextField("label_txt", 10, 0, 10,
  lineLength, 20);
lbl.text = "none";
```

前面的代码使用 Drawing API 绘制三条线，每条具有不同的 capsStyle 值。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

可以为 jointStyle 参数设置三种不同类型的联接点样式：

- round（默认）
- miter
- bevel

下面的示例演示这三种联接点样式之间的差别。

设置不同的联接点样式：

1. 创建一个新的 Flash 文档，并将它保存为 **jointstyles.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var lineLength:Number = 100;
// miter
this.createEmptyMovieClip("miter_mc", 10);
miter_mc.lineStyle(25, 0xFF0000, 100, true, "none", "none", "miter", 25);
miter_mc.moveTo(0, lineLength);
miter_mc.lineTo(lineLength / 2, 0);
miter_mc.lineTo(lineLength, lineLength);
miter_mc.lineTo(0, lineLength);
miter_mc._x = 50;
```

```
miter_mc._y = 50;
var lbl:TextField = miter_mc.createTextField("label_txt", 10, 0,
  lineLength + 20, lineLength, 20);
lbl.autoSize = "center";
lbl.text = "miter";
// round
this.createEmptyMovieClip("round_mc", 20);
round_mc.lineStyle(25, 0xFF0000, 100, true, "none", "none", "round");
round_mc.moveTo(0, lineLength);
round_mc.lineTo(lineLength / 2, 0);
round_mc.lineTo(lineLength, lineLength);
round_mc.lineTo(0, lineLength);
round_mc._x = 200;
round_mc._y = 50;
var lbl:TextField = round_mc.createTextField("label_txt", 10, 0,
  lineLength + 20, lineLength, 20);
lbl.autoSize = "center";
lbl.text = "round";
// bevel
this.createEmptyMovieClip("bevel_mc", 30);
bevel_mc.lineStyle(25, 0xFF0000, 100, true, "none", "none", "bevel");
bevel_mc.moveTo(0, lineLength);
bevel_mc.lineTo(lineLength / 2, 0);
bevel_mc.lineTo(lineLength, lineLength);
bevel_mc.lineTo(0, lineLength);
bevel_mc._x = 350;
bevel_mc._y = 50;
var lbl:TextField = bevel_mc.createTextField("label_txt", 10, 0,
  lineLength + 20, lineLength, 20);
lbl.autoSize = "center";
lbl.text = "bevel";
```

Flash 使用 Drawing API 在舞台上创建三个三角形。每个三角形具有不同的联接点样式值。

3. 保存 Flash 文档，然后选择 “控制” > “测试影片” 对该文档进行测试。

设置线条尖角 (miterLimit)

`miterLimit` 属性是一个数值，它指示切断尖角联接点（请参见第 492 页的 “设置线条端点 (capsStyle) 和联接点 (jointStyle)”）时的限制。`miterLimit` 值通常是笔触值的倍数。例如，如果值为 2.5，则 `miterLimit` 在笔触大小的 2.5 倍处切断。有效值为 0 到 255（如果 `miterLimit` 的值为 `undefined`，则默认值为 3）。仅当 `jointStyle` 设置为 `miter` 时，才使用 `miterLimit` 属性。

使用 Drawing API 方法和实现脚本动画

可以将 Drawing API 与 Tween 和 TransitionManager 类结合使用以创建一些出色的动画结果，并且只需编写少量 ActionScript。

下面的过程加载 JPEG 图像并动态遮罩图像，以便在加载图像后通过补间图像遮罩慢慢揭开图像。

为动态遮罩添加动画效果：

1. 创建一个新的 Flash 文档，并将它保存为 **dynmask.fla**。
2. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
import mx.transitions.Tween;
import mx.transitions.easing.*;
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc._visible = false;
  // 在舞台上将图像居中。
  target_mc._x = (Stage.width - target_mc._width) / 2;
  target_mc._y = (Stage.height - target_mc._height) / 2;
  var maskClip:MovieClip = target_mc.createEmptyMovieClip("mask_mc", 20);
  with (maskClip) {
    // 绘制与加载的图像大小相同的遮罩。
    beginFill(0xFF00FF, 100);
    moveTo(0, 0);
    lineTo(target_mc._width, 0);
    lineTo(target_mc._width, target_mc._height);
    lineTo(0, target_mc._height);
    lineTo(0, 0);
    endFill();
  }
  target_mc.setMask(maskClip);
  target_mc._visible = true;
  var mask_tween:Object = new Tween(maskClip, "_yscale", Strong.easeOut,
  0, 100, 2, true);
};
this.createEmptyMovieClip("img_mc", 10);
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

此代码示例导入 Tween 类和 easing 包中的每个类。然后，它创建作为 MovieClipLoader 实例的侦听器对象的对象，该实例将在后面部分的代码中创建。侦听器对象定义一个事件侦听器 `onLoadInit`，它使动态加载的 JPEG 图像在舞台上居中。代码重新定位图像后，将在 `target_mc` 影片剪辑（其中包含动态加载的 JPEG 图像）中创建新的影片剪辑实例。Drawing API 代码将在这一新的影片剪辑中绘制一个尺寸与 JPEG 图像相同的矩形。新的影片剪辑通过调用 `MovieClip.setMask()` 方法来遮罩 JPEG 图像。绘制并设置遮罩后，遮罩将使用 Tween 类添加动画效果，从而使图像慢慢显露出来。

3. 保存 Flash 文档，然后选择“控制”>“测试影片”对该 SWF 文件进行测试。

注意 若要为前面示例中的 `_alpha` 而不是 `_yscale` 添加动画效果，请直接对 `target_mc` 进行补间，而不要遮罩影片剪辑。

在硬盘上的 Samples 文件夹中可以找到范例源文件 drawingapi.fla，该文件演示了如何在 Flash 应用程序中使用 Drawing API。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\DrawingAPI。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/DrawingAPI。

了解缩放和切片辅助线

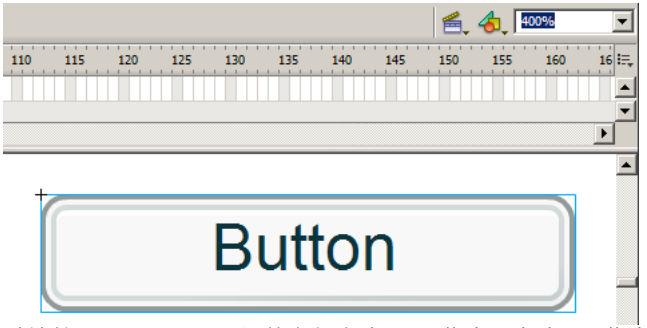
可以使用 9 切片缩放（缩放 9）为影片剪辑指定组件样式缩放比例。与通常应用于图形和设计元素的缩放类型不同，指定 9 切片缩放允许您创建能恰当缩放以用作用户界面组件的影片剪辑元件。

了解 9 切片缩放的工作原理

说明 9 切片缩放工作原理的最简单方式就是查看一个有关 9 切片缩放在 Flash 中的工作方式的示例。

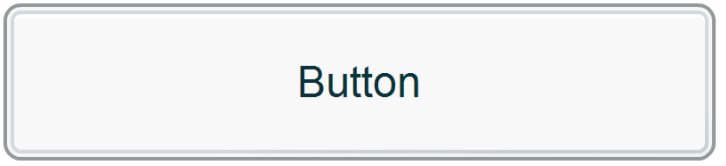
了解 Flash 中的缩放：

1. 创建一个新的 Flash 文档，并将它保存为 **dynmask.fla**。
2. 将 Button 组件的副本从 “组件” 面板 （“窗口” > “组件”）拖动到舞台上。
3. 使用 “缩放” 工具将舞台的缩放级别增加到 400%。



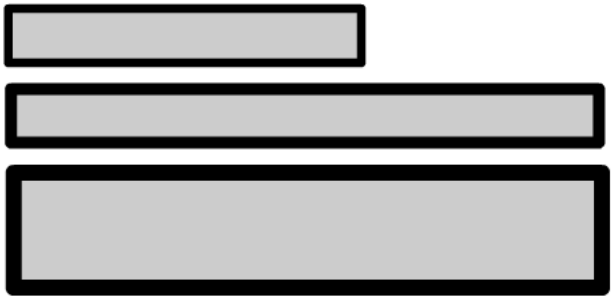
默认情况下，Button 组件实例宽为 100 像素，高为 22 像素。

4. 使用属性检查器将 Button 组件实例的大小调整为宽度为 200 像素，高度为 44 像素。



您可以看到，即使组件调整了大小，但按钮的边框和文本标签都不会扭曲。按钮的标签保持居中，并且保持字体大小。Macromedia Component Architecture 第 2 版的组件不使用 9 切片缩放，但组件仍处理第 2 版组件体系结构中的缩放，以使轮廓不会改变大小（如下图所示）。

假设按钮实例切片为 9 个单独的部分，或 3 x 3 网格，类似于电话上的小键盘或键盘。当您水平调整按钮实例的大小时，只有居中的三个竖直段（小键盘上的数字 2、5 和 8）拉伸，以使内容不会扭曲。如果垂直调整按钮实例的大小，则只有居中的三个水平段（小键盘上的数字 4、5 和 6）将调整大小。缩放网格的四个角根本不会缩放，这样组件可以变大，而不会产生被拉伸的感觉 （请参见下面的图像）。



提示 笔触在 9 切片缩放转换后从边缘处创建，因此不会变形或失去细节。

在 Flash 环境的 “转换为元件”对话框或 “元件属性”对话框中，可以为 9 切片缩放启用切片辅助线。仅当为 Flash Player 8 发布并且行为设置为影片剪辑时，为 9 切片缩放启用辅助线复选框才可用。对于 Flash 的早期版本或者当您创建按钮或图形元件时，9 切片缩放辅助线不可用。在 ActionScript 中通过设置影片剪辑实例的 `scale9Grid` 属性可以启用 9 切片缩放。

无论使用用户界面还是使用 ActionScript 创建切片辅助线，都可以通过跟踪影片剪辑的 `scale9Grid` 属性来跟踪 *x* 坐标、 y 坐标、宽度和高度。

```
trace(my_mc.scale9Grid); // (x=20, y=20, w=120, h=120)
```

此代码片断输出 `scale9Grid` 属性所使用的矩形对象的值。该矩形的 x 和 y 坐标均为 20 像素，宽度为 120 像素，高度为 120 像素。

在 ActionScript 中使用 9 切片缩放

在下面的示例中，将使用绘画工具绘制 300 x 300 像素的方形，并使用 9 切片缩放来调整该方形的大小。该方形分为九个较小的方形，每个约为 100 像素宽，100 像素高。调整方形的大小时，除角之外的每个段都将展开，以与指定的宽度和高度匹配。

通过 ActionScript 使用 9 切片缩放：

1. 创建一个新的 Flash 文档，并将它保存为 **ninescale.fla**。
2. 将一个 Button 组件拖动到当前文档的库中。
3. 选择 “矩形”工具并在舞台上绘制一个红色方形 （300 x 300 像素），其笔触为 15 像素的黑色笔触。
4. 选择 “椭圆”工具并在舞台上绘制一个紫色圆形 （50 x 50 像素），其笔触为 2 像素的黑色笔触。
5. 选择紫色圆形并将其拖动到前面创建的红色方形的右上角。
6. 选择“椭圆”工具并绘制一个新的圆形（约为 200 x 200 像素），将其放置在舞台边缘。
7. 选择舞台上的新圆形并进行拖动，以使圆形的中点位于方形的左下角。
8. 在圆形实例外单击鼠标，以取消选择该圆形。
9. 再次双击该圆形以将其选中，并按下退格键删除该形状，并删除方形的圆形部分。
10. 使用鼠标选择整个红色的方形以及内部的紫色圆形。
11. 按 F8 将该形状转换为影片剪辑元件。
12. 为舞台上的影片剪辑指定实例名称 **my_mc**。
13. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
import mx.controls.Button;
import flash.geom.Rectangle;

var grid:Rectangle = new Rectangle(100, 100, 100, 100);

var small_button:Button = this.createClassObject(Button, "small_button",
  10, {label:"Small"});
small_button.move(10, 10);
small_button.addEventListener("click", smallHandler);
function smallHandler(eventObj:Object):Void {
     my_mc._width = 100;
     my_mc._height = 100;
}

var large_button:Button = this.createClassObject(Button, "large_button",
  20, {label:"Large"});
large_button.move(120, 10);
large_button.addEventListener("click", largeHandler);
function largeHandler(eventObj:Object):Void {
```

```
    my_mc._width = 450;
    my_mc._height = 300;
}

var toggle_button:Button = this.createClassObject(Button,
  "toggle_button", 30, {label:"scale9Grid=OFF", toggle:true,
  selected:false});
toggle_button.move(420, 10);
toggle_button.setSize(120, 22);
toggle_button.addEventListener("click", toggleListener);
function toggleListener(eventObj:Object):Void {
  if (eventObj.target.selected) {
    eventObj.target.label = "scale9Grid=ON";
    my_mc.scale9Grid = grid;
  } else {
    eventObj.target.label = "scale9Grid=OFF";
    my_mc.scale9Grid = undefined;
  }
}
```

前面的代码分为五个部分。第一部分代码导入两个类：mx.controls.Button（Button 组件类）和 flash.geom.Rectangle。第二部分代码创建一个新的 Rectangle 类实例，并将 x 和 y 坐标指定为 100 像素，将宽度和高度也指定为 100 像素。此矩形实例用于为后面创建的影片剪辑形状设置 9 切片缩放网格。

下一步，您将创建一个新的 Button 组件实例，并为其给定实例名称 `small_button`。只要单击此按钮，前面创建的影片剪辑便重新调整为 100 像素宽和 100 像素高。第四部分代码动态创建一个名为 `large_button` 的新 Button 实例，单击该实例时，将把目标影片剪辑调整为 450 像素宽和 300 像素高。最后一部分代码创建一个新的 Button 实例，用户可以将其切换为打开或关闭。当该按钮处于打开状态时，将应用 9 切片网格。如果按钮处于关闭状态，则禁用 9 切片网格。

14. 保存 Flash 文档，然后选择“控制”>“测试影片”对该 SWF 文件进行测试。

此代码示例在舞台上添加并定位三个 Button 组件实例，并为每个按钮创建事件侦听器。如果单击禁用了 9 切片网格的 Large 按钮，则可以看到图像扭曲，呈现拉伸效果。通过单击切换按钮启用 9 切片网格，然后再次单击 Large 按钮。因为已启用了 9 切片网格，左上角的圆形应不再显示为扭曲。

第 14 章

用 ActionScript 创建交互操作

14

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 ActionScript 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 ActionScript 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。

有关使用 ActionScript 创建交互的更多信息，请参见以下主题：

关于事件和交互

只要用户单击鼠标或按下某个按键，该动作就会生成一个事件。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动作而生成的。您可以编写 ActionScript 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。ActionScript 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 9 章 “处理事件”。

控制 SWF 文件回放

下面的 ActionScript 函数用于控制时间轴上的播放头并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()` 方法来导航特定影片剪辑对象的时间轴。请参见第 502 页的 “跳到某一帧或场景”。
- `play()` 和 `stop()` 动作用于播放和停止 SWF 文件。请参见第 503 页的 “播放和停止影片剪辑”。
- `getURL()` 动作用于跳到不同的 URL。请参见第 504 页的 “跳到不同的 URL”。

有关更多信息，请参见以下主题：

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 `MovieClip.gotoAndPlay()` 和 `MovieClip.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。

下面的示例在按钮对象的 `onRelease` 事件处理函数内使用全局 `gotoAndPlay()` 函数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` 的影片剪辑实例的时间轴发送到第 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);
};
```

在最后一个示例中，全局 `gotoAndStop()` 函数用于将播放头移动到场景 2 第 1 帧。如果没有指定场景，则播放头将转到当前场景的指定帧。只能在根时间轴上使用 scene 参数，不能在影片剪辑或文档中其它对象的时间轴内使用该参数。

```
nextScene_mc.onRelease = function() {
  gotoAndStop("Scene 2", 1);
}
```

播放和停止影片剪辑

除非在启动 SWF 文件之后另有指示，否则将逐帧播放时间轴。您可以通过使用 play() 和 stop() 全局函数或者等效的 MovieClip 方法开始或停止播放 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() 或 gotoAndPlay() 来明确指示要重新开始播放。

可以使用 play() 和 stop() 函数或 MovieClip 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。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 onPress() 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：

```
// 附加到 myMovie_mc 影片剪辑实例
myMovie_mc.onPress() {
  stop();
};
```

相同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，将停止那个包含 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，可以让一个按钮链接到新 Web 站点，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 锚标签 (<a>) 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：

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

您还可以使用 GET 或 POST 方法将变量随 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、预计先接收 name 然后是 age 的两个表单变量的 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.firstName = "Francois";
variables_mc.age = 32;
```

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

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

getURL() 的功能由使用的浏览器决定。使所有浏览器具有相同功能的最可靠方法就是在 HTML 代码中调用 JavaScript 函数，使用 JavaScript window.open() 方法打开窗口。在您的 HTML 模板中添加以下 HTML 和 JavaScript：

```
<script language="JavaScript">
<--
  function openNewWindow(myURL) {
    window.open(myURL, "targetWindow");
  }
// -->
</script>
```

可以使用以下 ActionScript 在 SWF 文件中调用 openNewWindow：

```
var myURL:String = "http://foo.com";
getURL("javascript:openNewWindow('" + String(myURL) + "');");
```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 getURL 函数。

创建交互性和视觉效果

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

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 ActionScript 中隐藏标准指针，并跟踪自定义指针的移动。若要隐藏标准指针，可以使用内置 Mouse 类的 `hide()` 方法（请参见 《ActionScript 2.0 语言参考》中的 hide （Mouse.hide 方法）。

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. 在属性检查器中的 “实例名称” 文本框中，键入 **cursor_mc**。
4. 在时间轴中选择第 1 帧，然后在 “动作” 面板中添加下面的代码：

```
Mouse.hide();
cursor_mc.onMouseMove = function() {
  this._x = _xmouse;
  this._y = _ymouse;
  updateAfterEvent();
};
```

当影片剪辑刚出现在舞台上时，`Mouse.hide()` 方法会隐藏标准指针；`onMouseMove` 函数会将自定义指针放在标准指针所在的位置，并且只要用户移动鼠标就会调用 `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《ActionScript 2.0 语言参考》中的 updateAfterEvent 函数。）

5. 选择 “控制” > “测试影片” 来测试您的自定义指针。

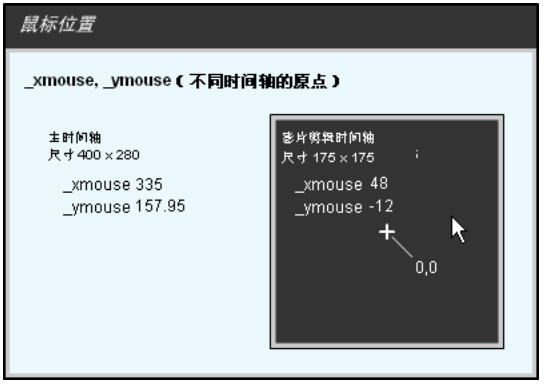
当使用自定义鼠标指针时，按钮仍然起作用。将自定义指针放在时间轴的顶层是一个很好的方法，这样当您在 SWF 文件中移动鼠标时，自定义指针就可以显示在其它层中的按钮和对象的前面。此外，自定义指针的 “尖端” 是您用作自定义指针的影片剪辑的注册点。因此，如果您希望影片剪辑的某一部分作为鼠标尖端，则应将剪辑的注册点坐标设置为该点的坐标。

有关 Mouse 类的方法的更多信息，请参见 《ActionScript 2.0 语言参考》中的 Mouse。

获取指针位置

可以使用 _xmouse 和 _ymouse 属性找到 SWF 文件中指针的位置。例如，可以在地图应用程序中使用这些属性，获取 _xmouse 和 _ymouse 属性的值，并使用该值计算特定位置的经度和纬度。

每个时间轴都有一个 _xmouse 和 _ymouse 属性，可返回指针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。对于影片剪辑，注册点取决于创建影片剪辑时设置的注册点或影片剪辑在舞台上的位置。



主时间轴内的 *_xmouse* 和 *_ymouse* 属性以及影片剪辑时间轴

下面的步骤演示了几种获取主时间轴或影片剪辑中的指针位置的方法。

获取当前指针位置：

1. 创建两个动态文本字段，并将其命名为 **box1_txt** 和 **box2_txt**。
2. 为文本框分别添加以下标签：分别为 X 位置和 Y 位置。
3. 选择 “窗口” > “动作”打开 “动作”面板 （如果尚未打开）。
4. 将以下代码添加到 “脚本”面板：

```
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  // 返回鼠标的 X 和 Y 位置
  box1_txt.text = _xmouse;
  box2_txt.text = _ymouse;
};
Mouse.addListener(mouseListener);
```

5. 选择 “控制” > “测试影片”测试该 Flash 影片。当您在舞台上移动指针时，box1_txt 和 box2_txt 字段就会显示指针的位置。

有关 _xmouse 和 _ymouse 属性的更多信息，请参见 《ActionScript 2.0 语言参考》中的 _xmouse（MovieClip._xmouse 属性）和 _ymouse （MovieClip._ymouse 属性）。

捕获按键

在 Flash Player 中，您可以使用全局 on() 处理函数截获按键的内置行为，如下面的示例所示：

```
/* 当按下左箭头或右箭头键时，处理函数所附加到的影片剪辑的透明度会更改。 */
on (keyPress "<Left>") {
  this._alpha -= 10;
}
on (keyPress "<Right>") {
  this._alpha += 10;
}
```

当您使用 “控制” > “测试影片”测试应用程序时，请确保选择了 “控制” > “禁用快捷键”，否则具有内置行为的某些键不会被覆盖。请参见《ActionScript 2.0 语言参考》中 on 处理函数的 keyPress 参数。

您可以使用内置的 Key 类的方法来检测用户最后按下的键。 Key 类不需要构造函数；要使用其方法，只需调用该类上的方法即可，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美国信息交换标准码 (ASCII) 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：

- 请参见附录 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 “ActionScript 2.0 类” > “影片” > “Key” > “常数”。）
- 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。事件侦听器 `Key.onKeyDown` 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。

以下示例演示如何捕获按键动作，以便根据相应的方向键（上、下、左或右）在舞台上向上、向下、向左或向右移动影片剪辑。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，该影片剪辑能够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_mc`。
2. 在时间轴上选择第 1 帧；如果看不到“动作”面板，则选择“窗口”>“动作”将其打开。
3. 要设置每次按键时汽车沿屏幕移动的距离，请定义 `distance` 变量并将它的值设置为 10：

   ```
   var distance:Number = 10;
   ```
4. 将下面的 ActionScript 代码添加到 “动作”面板中现有代码下面：

   ```
   this.createTextField("display_txt", 999, 0, 0, 100, 20);
   ```
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：

   ```
   var keyListener:Object = new Object();
   keyListener.onKeyDown = function() {
   };
   Key.addListener(keyListener);
   ```

6. 要检查是否已按下向左箭头键并且随之移动汽车影片剪辑，请将以下代码添加到 onEnterFrame 事件处理函数体中。

   您的代码应如下面的示例所示（新加代码以粗体显示）：

```
var distance:Number = 10;
this.createTextField("display_txt", 999, 0, 0, 100, 20);
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.LEFT)) {
    car_mc._x = Math.max(car_mc._x - distance, 0);
    display_txt.text = "Left";
  }
};
Key.addListener(keyListener);
```

   如果按下左箭头键，则汽车的 _x 属性被设置为当前 _x 值减去距离所得的值与 0 值之间的较大值。因此，_x 属性的值绝不会小于 0。此外，Left 一词也应显示在 SWF 文件中。

7. 使用类似的代码检查是否按下了右箭头键、上箭头键或下箭头键。

   您的完整代码应如下面的示例所示（新加代码以粗体显示）：

```
var distance:Number = 10;
this.createTextField("display_txt", 999, 0, 0, 100, 20);
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.LEFT)) {
    car_mc._x = Math.max(car_mc._x - distance, 0);
    display_txt.text = "Left";
  } else if (Key.isDown(Key.RIGHT)) {
    car_mc._x = Math.min(car_mc._x + distance, Stage.width -
  car_mc._width);
    display_txt.text = "Right";
  } else if (Key.isDown(Key.UP)) {
    car_mc._y = Math.max(car_mc._y - distance, 0);
    display_txt.text = "Up";
  } else if (Key.isDown(Key.DOWN)) {
    car_mc._y = Math.min(car_mc._y + distance, Stage.height -
  car_mc._height);
    display_txt.text = "Down";
  }
};
Key.addListener(keyListener);
```

8. 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《ActionScript 2.0 语言参考》中的 Key。

设置颜色值

可以使用内置 ColorTransform 类的方法 (flash.geom.ColorTransform) 调整影片剪辑的颜色。ColorTransform 类的 rgb 属性可为影片剪辑指定十六进制的红、绿、蓝 (RGB) 值。下面的示例根据用户按键使用 rgb 更改对象颜色。

设置影片剪辑的颜色值：

1. 创建一个新的 Flash 文档，并将其保存为 **setrgb.fla**。
2. 选择 “矩形”工具并在舞台上绘制一个大方型。
3. 将该形状转换为影片剪辑元件并在“属性”检查器中将该元件实例名称命名为 **car_mc**。
4. 创建一个名为 colorChip 的按钮元件，在舞台上放置该按钮的四个实例，并将这些实例分别命名为 red_btn、green_btn、blue_btn 和 black_btn。
5. 在主时间轴上选择第 1 帧，然后选择 “窗口” > “动作”。
6. 将下面的代码添加到主时间轴中的第 1 帧：

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
var trans:Transform = new Transform(car_mc);
trans.colorTransform = colorTrans;
```

7. 要使蓝色按钮将 car_mc 影片剪辑的颜色更改为蓝色，请将以下代码添加到“动作”面板：

```
blue_btn.onRelease = function() {
  colorTrans.rgb = 0x333399; // ¿¹…´
  trans.colorTransform = colorTrans;
};
```

   前面的代码片断在按下该按钮时更改颜色转换对象的 rgb 属性并将颜色转换效果重新应用于 car_mc 影片剪辑。

8. 对于其它按钮 （red_btn、green_btn 和 black_btn）重复执行第 7 步，以便将影片剪辑的颜色更改为相应的颜色。

   现在您的代码应如下面的示例所示 （新代码以粗体显示）：

```
import flash.geom.ColorTransform;
import flash.geom.Transform;

var colorTrans:ColorTransform = new ColorTransform();
var trans:Transform = new Transform(car_mc);
trans.colorTransform = colorTrans;

blue_btn.onRelease = function() {
  colorTrans.rgb = 0x333399; // 蓝色
  trans.colorTransform = colorTrans;
};
```

```
red_btn.onRelease = function() {
  colorTrans.rgb = 0xFF0000; // 红色
  trans.colorTransform = colorTrans;
};
green_btn.onRelease = function() {
  colorTrans.rgb = 0x006600; // 绿色
  trans.colorTransform = colorTrans;
};
black_btn.onRelease = function() {
  colorTrans.rgb = 0x000000; // 黑色
  trans.colorTransform = colorTrans;
};
```

9. 选择 “控制” > “测试影片”，更改影片剪辑的颜色。

有关 ColorTransform 类的方法的更多信息，请参见 《ActionScript 2.0 语言参考》中的 ColorTransform (flash.geom.ColorTransform)。

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。

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。

以下过程演示如何创建声音控制。

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 ActionScript 导出”和 “在第一帧导出”，然后指定声音标识符 `a_thousand_ways`。
4. 在舞台上添加一个按钮，然后将它命名为 `play_btn`。
5. 在舞台上添加一个按钮，然后将它命名为 `stop_btn`。

6. 在主时间轴上选择第 1 帧，然后选择 “窗口” > “动作”。

   将以下代码添加到 “动作”面板中：

```
var song_sound:Sound = new Sound();
song_sound.attachSound("a_thousand_ways");
play_btn.onRelease = function() {
  song_sound.start();
};
stop_btn.onRelease = function() {
  song_sound.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_sound)，并向该对象附加链接标识符为 a_thousand_ways 的声音。与 playButton 和 stopButton 对象关联的 onRelease 事件处理函数通过使用 Sound.start() 和 Sound.stop() 方法启动和停止该声音，并且还播放和停止附加的声音。

7. 选择 “控制” > “测试影片”来试听声音。

创建滑动音量控件：

1. 使用 “矩形”工具，在舞台上绘制一个小矩形，大约 30 像素高 10 像素宽。
2. 选择 “选择”工具并双击舞台上的形状。
3. 按 F8 键打开 “转换为元件”对话框。
4. 选择 “按钮”类型，输入 **volume** 作为元件名称并单击 “确定”。
5. 在舞台上选中该按钮元件，在 “属性”检查器中输入实例名称 **handle_btn** 。
6. 选择该按钮，然后选择 “修改” > “转换为元件”。

   选择影片剪辑行为时要小心。这将创建一个在第一帧中带有按钮的影片剪辑。

7. 选择该影片剪辑，然后在属性检查器中输入 **volume_mc** 作为实例名称。
8. 在主时间轴上选择第 1 帧，然后选择 “窗口” > “动作”。
9. 在 “动作”面板中输入下面的代码：

```
this.createTextField("volume_txt", 10, 30, 30, 200, 20);
volume_mc.top = volume_mc._y;
volume_mc.bottom = volume_mc._y;
volume_mc.left = volume_mc._x;
volume_mc.right = volume_mc._x + 100;
volume_mc._x += 100;

volume_mc.handle_btn.onPress = function() {
  startDrag(this._parent, false, this._parent.left, this._parent.top,
  this._parent.right, this._parent.bottom);
};
volume_mc.handle_btn.onRelease = function() {
  stopDrag();
  var level:Number = Math.ceil(this._parent._x - this._parent.left);
```

```
    this._parent._parent.song_sound.setVolume(level);
    this._parent._parent.volume_txt.text = level;
};
volume_mc.handle_btn.onReleaseOutside = slider_mc.handle_btn.onRelease;
```

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

10. 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. 使用 “矩形”工具在舞台上绘制一个小矩形，大约 30 像素高 10 像素宽。
2. 选择 “选择”工具并双击舞台上的形状。
3. 按 F8 键启动 “转换为元件”对话框。
4. 选择 “按钮”类型，输入元件名称 **balance** 并单击 “确定”。
5. 在舞台上选中该按钮元件，在 “属性”检查器中输入实例名称 **handle_btn** 。
6. 选择该按钮，然后选择 “修改” > “转换为元件”。

   选择影片剪辑行为时要小心。这将创建一个在第一帧中带有按钮的影片剪辑。
7. 选择该影片剪辑，然后在属性检查器中输入 **balance_mc** 作为实例名称。
8. 在 “动作”面板中输入下面的代码：

```
balance_mc.top = balance_mc._y;
balance_mc.bottom = balance_mc._y;
balance_mc.left = balance_mc._x;
balance_mc.right = balance_mc._x + 100;
balance_mc._x += 50;
balance_mc.handle_btn.onPress = function() {
  startDrag(this._parent, false, this._parent.left, this._parent.top,
  this._parent.right, this._parent.bottom);
};
balance_mc.handle_btn.onRelease = function() {
  stopDrag();
  var level:Number = Math.ceil((this._parent._x - this._parent.left - 50)
  * 2);
  this._parent._parent.song_sound.setPan(level);
};
balance_mc.handle_btn.onReleaseOutside = balance_mc.handle_btn.onRelease;
```

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

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 《ActionScript 2.0 语言参考》中的 Sound。

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）。

您可能想了解是否发生了冲突，以测试用户是否已到达舞台上的某个特定静态区域，或确定一个影片剪辑何时到达另一个影片剪辑。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，则 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。如果二者相交，则 hitTest() 返回 true。如果两个边框没有相交，则 hitTest() 返回 false。

您也可以使用 hitTest() 来测试两个影片剪辑之间的冲突。

下面的示例演示如何检测舞台上鼠标和影片剪辑之间的冲突。

检测影片剪辑和鼠标指针之间的冲突：

1. 在时间轴的图层 1 上选择第一个帧。
2. 选择 “窗口” > “动作”，打开 “动作”面板 （如果尚未打开）。
3. 在 “动作”面板中添加以下代码：

```
this.createEmptyMovieClip("box_mc", 10);
with (box_mc) {
  beginFill(0xFF0000, 100);
  moveTo(100, 100);
  lineTo(200, 100);
  lineTo(200, 200);
  lineTo(100, 200);
  lineTo(100, 100);
  endFill();
}

this.createTextField("status_txt", 999, 0, 0, 100, 22);

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function():Void {
  status_txt.text = _level0.hitTest(_xmouse, _ymouse, true);
}
Mouse.addListener(mouseListener);
```

4. 选择 “控制” > “测试影片”，然后将指针滑过影片剪辑以测试冲突。

   只要指针停留在不透明像素上，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_mc 和 area_mc。
2. 在时间轴上选择第 1 帧。
3. 如果看不到 “动作”面板，则选择 “窗口” > “动作”将其 打开。
4. 在 “动作”面板中输入下面的代码：

```
this.createTextField("status_txt", 999, 10, 10, 100, 22);
area_mc.onEnterFrame = function() {
  status_txt.text = this.hitTest(car_mc);
};

car_mc.onPress = function() {
  this.startDrag(false);
  updateAfterEvent();
};
car_mc.onRelease = function() {
  this.stopDrag();
};
```

5. 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《ActionScript 2.0 语言参考》中的 hitTest （MovieClip.hitTest 方法）。

创建简单的线条绘制工具

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图工具，并且可以在 SWF 文件中绘制响应事件的形状。绘图方法包括 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 (_level0.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() 绘图方法将当前绘图位置设置为指定的 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。

创建简单线条绘制工具：

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。
3. 选择 “窗口” > “动作”，打开 “动作”面板 （如果尚未打开）。
4. 在 “动作”面板中输入以下代码：

```
this.createEmptyMovieClip("canvas_mc", 999);
var isDrawing:Boolean = false;
//
clear_btn.onRelease = function() {
  canvas_mc.clear();
};
//
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  canvas_mc.lineStyle(5, 0xFF0000, 100);
  canvas_mc.moveTo(_xmouse, _ymouse);
  isDrawing = true;
};
mouseListener.onMouseMove = function() {
  if (isDrawing) {
    canvas_mc.lineTo(_xmouse, _ymouse);
    updateAfterEvent();
  }
};
mouseListener.onMouseUp = function() {
  isDrawing = false;
};
Mouse.addListener(mouseListener);
```

5. 选择 “控制” > “测试影片”对该文档进行测试。
6. 拖动指针在舞台上绘制线条。
7. 单击该按钮可以擦除所绘制的内容。

使用 ActionScript 创建运行时数据绑定

如果使用组件创建应用程序，通常需要在这些组件间添加绑定，以便您能够与数据交互或让组件彼此交互。组件间的交互对于创建用户能够与其交互的可用表单或界面非常有必要。可以使用组件检查器中的“绑定”选项卡在舞台上的组件间添加绑定。可以使用组件检查器中的“绑定”选项卡在舞台上的组件间绑定数据。

有关使用“绑定”选项卡的更多信息，请参见《使用 Flash》中的第 333 页的“在“绑定”选项卡中处理绑定（仅限于 Flash Professional）”。还可以在以下在线文章中找到其它信息：构建每日提示应用程序（第 2 部分），Macromedia Flash MX Professional 2004 中的数据绑定和使用 Macromedia Flash MX Professional 构建 Google 搜索应用程序。

可以使用 ActionScript 创建组件间绑定，而不使用“绑定”选项卡。添加代码通常比依靠创作环境更快，而且更为高效。当使用代码向应用程序添加组件时，必须使用 ActionScript 创建绑定。可以选择使用 `createClassObject()` 方法在舞台上动态添加组件；但是不能使用“绑定”选项卡创建绑定，因为组件直到运行时才存在。使用 ActionScript 添加数据绑定通常被称为运行时数据绑定。

有关更多信息，请参见以下主题：

- 使用 ActionScript 在 UI 组件间创建绑定
- 第 521 页的“使用组件、绑定和自定义格式程序”
- 第 524 页的“在舞台上添加或绑定组件”

使用 ActionScript 在 UI 组件间创建绑定

在运行时绑定两个组件之间的数据并不难。在 Flash Basic 8 或 Flash Professional 8 中均可执行此操作。要使文档正常工作，请务必在该文档中包含 DataBindingClasses 组件，因为该组件包含要使用的类。

使用 ActionScript 在两个 TextInput 组件间创建绑定：

1. 创建一个名为 **panel_as.fla** 的新 Flash 文档。
2. 将 TextInput 组件的两个副本拖动到舞台上。
3. 为组件指定下面的实例名称：**in_ti** 和 **out_ti**。
4. 选择“窗口”>“公用库”>“类”，打开名为 Classes.fla 的新公用库。
5. 将 DataBindingClasses 组件的副本拖动到“库”面板中，或将该组件拖动到舞台上然后删除。

完成后关闭公用库。从舞台删除 DataBindingClasses 组件后，Flash 就会在库中保存一个副本。

提示	如果忘了从舞台删除 DataBindingClasses 组件，运行时就会看到该组件的图标。

注意	在前一示例中，使用组件检查器创建绑定时，Flash 自动向 FLA 文件中添加了 DataBindingClasses 组件。使用 ActionScript 创建数据绑定时，必须将类手动复制到库中，如下一步所示。

6. 插入一个新的图层，并将其命名为 **actions**。
7. 在动作图层的第 1 帧中添加以下 ActionScript：

```
var src:mx.data.binding.EndPoint = new mx.data.binding.EndPoint();
src.component = in_ti;
src.property = "text";
src.event = "focusOut";
var dest:mx.data.binding.EndPoint = new mx.data.binding.EndPoint();
dest.component = out_ti;
dest.property = "text";
new mx.data.binding.Binding(src, dest);
```

如果您更喜欢稍短的版本，可以导入绑定类，并改用下面的代码：

```
import mx.data.binding.*;
var src:EndPoint = new EndPoint();
src.component = in_ti;
src.property = "text";
src.event = "focusOut";
var dest:EndPoint = new EndPoint();
dest.component = out_ti;
dest.property = "text";
new Binding(src, dest);
```

此 ActionScript 创建了两个数据绑定端点，要绑定的每个组件对应其中一个端点。您创建的第一个端点定义了它从哪个组件进行绑定 (`in_ti`)，监视哪个属性 (`text`)，以及哪个事件将触发绑定 (`focusOut`)。您创建的第二个端点只列出了组件和属性（分别是 `out_ti` 和 `text`）。最后，在为 Binding 类调用构造函数 (`new Binding(src, dest)`) 时，您创建了两个端点间的绑定。

不必在 ActionScript 中使用完全限定的类名称（如 `mx.data.binding.EndPoint`），如在第一个代码片段中看到的那样。如果在代码开头使用 `import` 语句，您可以避免使用完全限定名称。当使用通配符 (`*`) 导入 `mx.data.binding` 包中的所有类时（该包同时包括 EndPoint 和 Binding 类），可以缩短代码，并能直接引用 EndPoint 类和 Binding 类。有关 import 语句的更多信息，请参见《ActionScript 2.0 语言参考》中的 `import` 条目。

8. 选择“控制”>“测试影片”，在测试环境中测试此代码。在 in_ti 文本输入字段中输入一些文本。

   在 in_ti 实例失去焦点（单击“舞台”，按“选项卡”，或单击第二个字段）后，Flash 会将您输入 in_ti 的文本复制到 out_ti 文本字段。

9. 选择“文件”>“保存”以保存所做的更改。

如果希望修改上一练习中 out_ti 文本输入字段中的文本，您的代码就会变得复杂得多。如果使用组件检查器设置绑定，则默认情况下将创建一个双向连接。这意味着，如果更改了舞台上其中一个文本字段，则另一个文本字段也会更改。当使用 ActionScript 创建绑定时，应用程序的工作方式恰好相反。默认状态下运行时数据绑定是单向的，除非另有指定，如下面的示例所示。

若要使用 ActionScript 创建一个双向绑定，您需要对前面过程中的代码片断做出一些小小的改动。此示例中使用的是第二种方式，即第 7 步中缩短的 ActionScript 代码片断。

创建双向绑定：

1. 打开上例中的 **panel_as.fla**。
2. 对 ActionScript 稍加修改（见**粗体**代码），使之与下面的 ActionScript 匹配：

```
import mx.data.binding.*;
var src:EndPoint = new EndPoint();
src.component = in_ti;
src.property = "text";
src.event = "focusOut";
var dest:EndPoint = new EndPoint();
dest.component = out_ti;
dest.property = "text";
dest.event = "focusOut";
new Binding(src, dest, null, true);
```

   您对 ActionScript 所做的两处更改起到以下作用：

   - 为目标 EndPoint 实例定义事件属性。
   - 为 Binding 构造函数定义两个附加参数。

   第一个参数用于高级格式设置选项；可以将该值设为 null 或 undefined。第二个参数定义绑定是双向 (true) 还是单向 (false)。

   您可能觉得奇怪：focusOut 事件从何而来？这正是 ActionScript 变得复杂之处。您可以仔细看一看 TextInput 类，并使用列出的一些方法（如 change() 或 enter()），但在里面找不到 focusOut 事件。TextInput 类继承自 UIObject 和 UIComponent 类。如果查看一下 UIComponent 类（它向组件添加焦点支持），会看到四个附加的事件：focusIn、focusOut、keyDown 和 keyUp。可以将这些事件用于 TextInput 组件。

3. （可选）如果想使用上例来更新 out_ti 文本输入字段中的值，可以将事件从 focusOut 改为 change。

4. 选择 “控制” > “测试影片” 来测试该文档。

   Flash 更改 in_ti 文本输入字段中的第二个值，并为 out_ti 更新该值。您已成功创建了一个双向连接。

您可以在 Macromedia Component Architecture 第二版的大多数用户界面组件中使用 Binding 类，而不仅限于 TextInput 组件中。下面的示例演示了如何使用 ActionScript 在运行时绑定 CheckBox 实例和 Label 组件。

将绑定类用于 CheckBox 组件：

1. 创建一个新的 Flash 文档。
2. 选择 “文件” > “另存为”，并将新文件命名为 **checkbox_as.fla**。
3. 选择 “窗口” > “公用库” > “类”。
4. 将 DataBindingClasses 类的副本拖入文档的库中。
5. 将 CheckBox 组件的副本拖至舞台上，并为其指定实例名称 **my_ch**。
6. 将 Label 组件的副本拖至舞台，并为其指定实例名称 **my_lbl**。
7. 创建一个新图层并命名为 **actions**。
8. 在动作图层的第 1 帧中添加以下 ActionScript：

```
var srcEndPoint:Object = {component:my_ch, property:"selected",
  event:"click"};
var destEndPoint:Object = {component:my_lbl, property:"text"};
new mx.data.binding.Binding(srcEndPoint, destEndPoint);
```

   使用对象来定义端点，而不是创建 EndPoint 类的新实例，如本节上面的练习所示。这一步中的代码片段创建了两个对象，充当绑定所用的端点。在为 Binding 类调用构造函数时，就创建了绑定。若要进一步减少代码量（和可读性），请定义对象内联，如下面的代码片断所示：

```
new mx.data.binding.Binding({component:my_ch, property:"selected",
  event:"click"}, {component:my_lbl, property:"text"});
```

   此 ActionScript 降低了代码的可读性，但是也减少了需要键入的代码数量。如果要共享 FLA（或 ActionScript）文件，那么您可能想使用第一个 ActionScript 片断，因为它更易于阅读。

使用组件、绑定和自定义格式程序

自定义格式程序有助于以特定的方式设置复杂数据的格式。您还可以使用自定义格式设置来帮助在一个组件（如 DataGrid）内显示图像、HTML 格式文本或其它组件。下面的示例说明自定义格式程序的用途。

在文档中使用自定义格式程序：

1. 创建一个新的 FLA 文件，并将 DataBindingClasses 类添加到库（“窗口”>“公用库”>“类”）。
2. 将 DateChooser 组件的副本拖至舞台上，并为其指定实例名称 **my_dc**。
3. 将 Label 组件的副本拖至到舞台，请为其指定实例名称 **my_lbl**。
4. 插入一个新的图层，并命名为 **actions**。
5. 在动作图层的第 1 帧中添加以下 ActionScript 代码：

```
import mx.data.binding.*;
var src:EndPoint = new EndPoint();
src.component = my_dc;
src.property = "selectedDate";
src.event = "change";
var dest:EndPoint = new EndPoint();
dest.component = my_lbl;
dest.property = "text";
new Binding(src, dest);
```

   此代码在 DateChooser 的 `selectedDate` 属性和舞台上 Label 组件的 `text` 属性之间创建绑定。每次在日历上单击新日期时，选定的日期就会出现在 Label 组件中。

6. 将该 Flash 文档在硬盘上的恰当位置保存为 **customformat.fla**。

   （下一练习中还要使用它。）

7. 选择“控制”>“测试影片”来测试该文档。

   尝试更改 Calendar 组件中的日期，就会看到当前选定的日期出现在 Label 组件中。Label 组件不够宽，无法显示整个日期，所以 Flash 截短了文字。

8. 关闭测试 SWF 文件并返回到创作环境。

   要么在舞台上重新调整 Label 组件大小，要么选择 Label 组件，在属性检查器的“参数”选项卡中将 `autoSize` 属性设置为 `left`。

9. 选择 “控制” > “测试影片” 再次测试该文档。

现在文本字段会显示完整的日期，尽管不大美观而且缺少格式设置。根据您的时区和选定的日期，日期显示可能类似于下面的样子：`Thu Nov 4 00:00:00 GMT-0800 2004`

虽然绑定正常并显示 `selectedDate` 属性，但这些日期在用户友好方面表现不甚理想。没有人希望看到时区偏移，您甚至可能也不希望显示时、分和秒。您需要一种设置日期格式的方式，以使日期更具可读性，不再那么呆板。自定义格式程序对于设置文本格式特别有用。

使用 CustomFormatter 类设置数据格式

CustomFormatter 类定义两个方法 `format()` 和 `unformat()`，能将数据值在特定数据类型和字符串之间进行转换。默认情况下，这些方法不会执行任何操作；您必须在 `mx.data.binding.CustomFormatter` 的子类中实现它们。 CustomFormatter 类让您能够将数据类型转换为字符串并再转换过来。在这种情况下，您会希望将 DateChooser 组件的 `selectedDate` 属性转换为一种格式漂亮的字符串 （在值复制到 Label 组件中时）。

下面的示例说明如何创建自定义格式程序，以将日期显示为 `NOV 4, 2004`，而不是显示为默认日期字符串。

前提

您需要完成第 521 页的 “使用组件、绑定和自定义格式程序”中的练习，然后才能开始此练习。

使用 CustomFormatter 类设置数据的格式：

1. 选择 “文件” > “新建”，然后选择 “ActionScript 文件”，创建一个新的 AS 文件。
2. 选择 “文件” > “另存为”，将新文件保存为 **DateFormat.as**。
3. 在 “脚本” 窗口中输入下面的代码：

```
class DateFormat extends mx.data.binding.CustomFormatter {
  function format(rawValue:Date):String {
    var returnValue:String;
    var monthName_array:Array =
["JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DE
C"];
    returnValue = monthName_array[rawValue.getMonth()]+"
"+rawValue.getDate()+", "+rawValue.getFullYear();
    return returnValue;
  }
}
```

第一部分代码定义名为 DateFormat 的新类，该类扩展了 `mx.data.binding` 包中的 CustomFormatter 类。请记住， Flash 将编译 DataBindingClasses 组件文件中的绑定类，因此您不能在 Flash 安装目录的 Classes 文件夹中直接查看或找到这些类。

您唯一使用的方法是 format() 方法，它将日期实例转换成一种自定义字符串格式。下一步是创建月份名称数组，以便让最终结果看上去更接近 NOV 4, 2004，而不是默认的日期格式。请记住，在 Flash 中数组都从零开始，如果 rawValue.getMonth() 的值返回的是 1，它代表的是 2 月，不是 1 月（因为 1 月是第 0 月）。其余代码通过连接值并返回 returnValue 字符串而构建了自定义格式的字符串。

使用已编译剪辑中的类会产生一个问题，从上面的代码片断中可看到此问题。因为扩展了位于 DataBindingClasses 类中的一个类，但该类不是随时可供 Flash 使用，所以在前面的类中检查语法时会遇到下面的错误：

```
**Error** <path to DateFormat class>\DateFormat.as: Line 1: The class
  'mx.data.binding.CustomFormatter' could not be loaded.
     class DateFormat extends mx.data.binding.CustomFormatter {

Total ActionScript Errors: 1  Reported Errors: 1
```

您的代码可能没问题。此问题会在 Flash 无法定位该类时发生，因为这一点，语法检查失败。

4. 保存 DateFormat.as 文件。
5. 打开 “使用组件、绑定和自定义格式程序”中的练习中的 customformat.fla。确保将 DateFormat.as 保存或复制到与此文件相同的目录中。
6. 在 customformat.fla 中，修改动作图层的第 1 帧中的 ActionScript 代码，以与下面的代码匹配：

```
import mx.data.binding.*;
var src:EndPoint = new EndPoint();
src.component = my_dc;
src.property = "selectedDate";
src.event = "change";
var dest:EndPoint = new EndPoint();
dest.component = my_lbl;
dest.property = "text";
new Binding(src, dest, {cls:mx.data.formatters.Custom,
  settings:{classname:"DateFormat", classname_class:DateFormat}});
```

这一次您定义了一个 customFormatter 对象，它将告诉 Flash 您正在使用新创建的 DateFormat 类设置绑定时的端点格式。

7. 保存对文档的更改并选择 “控制” > “测试影片”来测试您的代码。

在舞台上添加或绑定组件

将绑定类用于 ActionScript 的最大优点之一是，您能够在 Flash 在运行时添加到舞台上的组件之间创建绑定。假定您创建了自己的在运行时向舞台添加适当文本字段的自定义类，接下来验证必要的数据并添加必要的绑定。只要您的库中有组件，您就可以动态添加这些组件并通过添加几行代码创建绑定。

使用 ActionScript 先在舞台上添加组件，然后绑定组件：

1. 创建一个新的 Flash 文档。
2. 将一个 ComboBox 和 Label 组件拖入文档的库中。
3. 插入一个新的图层，并命名为 **actions**。
4. 在动作图层的第 1 帧中添加以下代码。

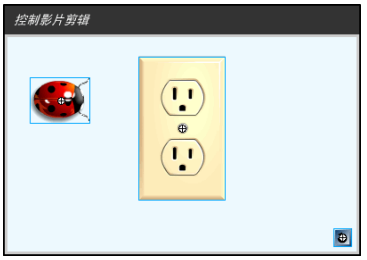
```
import mx.data.binding.*;
this.createClassObject(mx.controls.ComboBox, "my_cb", 1, {_x:10, _y:10});
this.createClassObject(mx.controls.Label, "my_lbl", 2, {_x:10, _y:40});
my_cb.addItem("JAN", 0);
my_cb.addItem("FEB", 1);
my_cb.addItem("MAR", 2);
my_cb.addItem("APR", 3);
my_cb.addItem("MAY", 4);
my_cb.addItem("JUN", 5);
var src:EndPoint = new EndPoint();
src.component = my_cb;
src.property = "value";
src.event = "change";
var dest:EndPoint = new EndPoint();
dest.component = my_lbl;
dest.property = "text";
new Binding(src, dest);
```

第一行 ActionScript 从 `mx.data.binding` 包中导入类，这样就不需要在代码中使用完全限定路径了。下两行 ActionScript 将文档库中的组件附加到舞台。接下来您在舞台上定位这些组件。

最后，您将数据添加到 ComboBox 实例，并在舞台上的 `my_cb` ComboBox 和 `my_lbl` Label 组件间创建绑定。

分析示例脚本

在范例 SWF 文件 zapper.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。每个对象都是一个影片剪辑实例。



下面的脚本附加到主时间轴第 1 帧：

```
var initx:Number = bug_mc._x;
var inity:Number = bug_mc._y;
var zapped:Boolean = false;

reset_btn.onRelease = function() {
  zapped = false;
  bug_mc._x = initx;
  bug_mc._y = inity;
  bug_mc._alpha = 100;
  bug_mc._rotation = 0;
};

bug_mc.onPress = function() {
  this.startDrag();
};
bug_mc.onRelease = function() {
  this.stopDrag();
};
bug_mc.onEnterFrame = function() {
  if (this.hitTest(this._parent.zapper_mc)) {
    this.stopDrag();
    zapped = true;
    bug_mc._alpha = 75;
    bug_mc._rotation = 20;
    this._parent.zapper_mc.play();
  }
  if (zapped) {
    bug_mc._y += 25;
  }
};
```

瓢虫的实例名称为 bug_mc，插座的实例名称为 zapper_mc。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。

有几个事件处理函数，它们具有不同的事件：onRelease()、onPress() 和 onEnterFrame()。这些事件处理函数是在加载 SWF 文件后在第 1 帧定义的。每次播放头进入帧时，onEnterFrame() 事件处理函数中的操作都会执行。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。

initx 和 inity 这两个变量被指定用于存储 bug_mc 影片剪辑实例的初始 x 和 y 位置。为 reset_btn 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 reset_btn 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。

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 (this._parent.zapper_mc)。该计算有两种可能的输出结果，即 true 或 false：

- 如果 hitTest() 方法返回 true，则 Flash 调用 stopDrag() 方法，将 zapper_mc 变量设置为 true，更改 alpha 和旋转属性，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。

在 bug_mc 实例上按下鼠标按钮时，会执行 onPress() 语句中的动作。在 bug_mc 实例上松开鼠标按钮时，会执行 onRelease() 语句中的动作。

startDrag() 动作可拖动瓢虫。因为脚本被附加到 bug_mc 实例，所以关键字 this 表明 bug 实例是可以拖动的实例：

```
bug_mc.onPress = function() {
  this.startDrag();
};
```

stopDrag() 动作可停止拖动动作：

```
bug_mc.onRelease = function() {
  this.stopDrag();
};
```

第 15 章

15 使用图像、声音和视频

如果在 Macromedia Flash Basic 8 或 Macromedia Flash Professional 8 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（包括其它 SWF 文件）。出于某些原因，您可能想将媒体保存在 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为多个独立的 SWF 文件，随后在运行时根据需要加载这些独立的文件。此过程可减少初次下载时间，还可更加轻松地维护和更新演示文稿。

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图像。通过在运行时加载每个图像的文件，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，例如动态加载的 Flash Video (FLV) 和 MP3 回放，只有在运行时才能通过 ActionScript 使用。

本部分介绍如何在 Flash 应用程序中使用图像文件、声音文件和 FLV 视频。有关更多信息，请参见以下主题：

关于加载和使用外部媒体

您可以在运行时向 Flash 应用程序中加载多种类型的媒体文件: SWF、MP3、JPEG、GIF、PNG 和 FLV 文件。但是，不是所有版本的 Flash Player 都支持每一种媒体。有关 Macromedia Flash Player 8 中所支持图像文件类型的更多信息，请参见第 529 页的“加载外部 SWF 和图像文件”。有关 Flash Player 中 FLV 视频支持的信息，请参见第 538 页的“关于使用 FLV 视频”

Macromedia 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或 `file://` 协议从本地磁盘加载。

若要加载外部 SWF 和图像文件，您可以使用 `loadMovie()` 或 `loadMovieNum()` 函数，`MovieClip.loadMovie()` 方法或 `MovieClipLoader.loadClip()` 方法。通常类方法提供的功能和灵活性要比全局函数提供的多，适用于较复杂的应用程序。当加载 SWF 或图像文件时，应指定影片剪辑或 SWF 文件级别作为该媒体的目标。有关加载 SWF 和图像文件的更多信息，请参见第 529 页的 “加载外部 SWF 和图像文件”。

若要回放外部 MP3 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 文件之前应进行渐进下载还是完全下载。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 536 页的“读取 MP3 文件中的 ID3 标签”。

Flash Video 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播放 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 541 页的“动态播放外部 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 或图像文件的下载进度。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 544 页的 “预加载 FLV 文件”。

您可以在硬盘上找到图库应用的范例。这些文件提供了一些示例，可帮助您了解在向 SWF 文件加载图像文件时，如何使用 ActionScript 动态地控制影片剪辑。可以在硬盘上的 Samples 文件夹中找到 gallery_tree.fla 和 gallery_tween.fla 范例源文件。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

加载外部 SWF 和图像文件

若要加载 SWF 或图像文件，请使用 `loadMovie()` 或 `loadMovieNum()` 全局函数、MovieClip 类的 `loadMovie()` 方法或 MovieClipLoader 类的 `loadClip()` 方法。有关 `loadClip()` 方法的更多信息，请参见 《ActionScript 2.0 语言参考》中的 `MovieClipLoader.loadClip()`。

对于图像文件，Flash Player 8 支持 JPEG（渐进式或非渐进式）图像文件类型、GIF 图像（透明和不透明，不过只加载动画 GIF 的第一帧）和 PNG 文件（透明和不透明）。

若要在 Flash Player 中将 SWF 或图像文件加载到某个级别，请使用 `loadMovieNum()` 函数。若要将 SWF 或图像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 或图像文件加载到影片剪辑目标中时，SWF 文件或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，但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 ActionScript 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 SWF 或图像文件。

当使用全局 `loadMovie()` 或 `loadMovieNum()` 函数时，指定目标级别或剪辑作为参数。下面的示例将 Flash 应用程序 contents.swf 加载到名为 `image_mc` 的影片剪辑实例中：

```
loadMovie("contents.swf", image_mc);
```

您可以使用 `MovieClip.loadMovie()` 实现相同的结果：

```
image_mc.loadMovie("contents.swf");
```

下面的示例将 JPEG 图像 image1.jpg 加载到影片剪辑实例 `image_mc` 中：

```
image_mc.loadMovie("http://www.helpexamples.com/flash/images/image1.jpg");
```

有关加载外部 SWF 和图像文件的更多信息，请参见第 532 页的 “关于加载 SWF 文件和根时间轴”。

若要将 SWF 和 JPEG 文件预加载到影片剪辑实例中，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。若要使用 MovieClipLoader 对象预加载 SWF 和 JPEG 文件，必须完成以下操作：

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。创建新的影片剪辑、将内容加载到其中，然后按下面的代码所示创建 MovieClipLoader 对象：

```
this.createEmptyMovieClip("img_mc", 999);
var my_mcl:MovieClipLoader = new MovieClipLoader();
```

创建侦听器对象和创建事件处理程序 侦听器对象可以是任何 ActionScript 对象，例如通用 Object 对象、影片剪辑或自定义组件。

下面的示例创建一个名为 loadListener 的通用侦听器对象，然后为自身定义 onLoadError、onLoadStart、onLoadProgress 和 onLoadComplete 函数：

```
// 创建侦听器对象:
var mclListener:Object = new Object();
mclListener.onLoadError = function(target_mc:MovieClip, errorCode:String,
  status:Number) {
  trace("Error loading image: " + errorCode + " [" + status + "]");
};
mclListener.onLoadStart = function(target_mc:MovieClip):Void {
  trace("onLoadStart: " + target_mc);
};
mclListener.onLoadProgress = function(target_mc:MovieClip,
  numBytesLoaded:Number, numBytesTotal:Number):Void {
  var numPercentLoaded:Number = numBytesLoaded / numBytesTotal * 100;
  trace("onLoadProgress: " + target_mc + " is " + numPercentLoaded + "%
  loaded");
};
mclListener.onLoadComplete = function(target_mc:MovieClip,
  status:Number):Void {
  trace("onLoadComplete: " + target_mc);
};
```

注意

Flash Player 8 允许在 onLoadComlete 和 onLoadError 事件侦听器中检查 MovieClipLoader 下载的 HTTP 状态。这种能力允许您检查文件不能下载的原因：是服务器错误，还是没有找到文件，等等。

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，如下面的代码所示：

```
my_mcl.addListener(mclListener);
```

开始将文件（图像或 SWF）加载到目标剪辑中 若要开始下载图像或 SWF 文件，可以使用 MovieClipLoader.loadClip() 方法，如下面的代码所示：

```
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

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。

下面的示例使用 ProgressBar 组件的 setProgress() 方法显示 SWF 文件的下载进度。（请参见《组件语言参考》中的 “ProgressBar.setProgress()”。）

使用 ProgressBar 组件显示下载进度：

1. 创建一个新的 Flash 文档，并将它保存为 **progress.fla**。
2. 打开 “组件” 面板 （“窗口” > “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（“窗口” > “属性” > “属性”），将 ProgressBar 组件命名为 **my_pb**。
5. 在时间轴中选择第 1 帧，然后打开 “动作” 面板 （“窗口” > “动作”）。
6. 将以下代码添加到 “动作” 面板中：

```
var my_pb:mx.controls.ProgressBar;
my_pb.mode = "manual";

this.createEmptyMovieClip("img_mc", 999);

var my_mcl:MovieClipLoader = new MovieClipLoader();
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip):Void {
  my_pb.label = "loading: " + target_mc._name;
};
mclListener.onLoadProgress = function(target_mc:MovieClip,
  numBytesLoaded:Number, numBytesTotal:Number):Void {
  var pctLoaded:Number = Math.ceil(100 * (numBytesLoaded /
  numBytesTotal));
  my_pb.setProgress(numBytesLoaded, numBytesTotal);
};
my_mcl.addListener(mclListener);
my_mcl.loadClip("http://www.helpexamples.com/flash/images/image1.jpg",
  img_mc);
```

7. 选择 “控制” > “测试影片”，对文档进行测试。

   将图像加载到影片剪辑 `img_mc` 中。
8. 选择 “文件” > “发布设置” > “格式”，确保选择 SWF 和 HTML 选项。
9. 单击 “发布”，然后在硬盘上查找 HTML 和 SWF 文件。

   它们与在步骤 1 中保存的 progress.fla 位于同一文件夹中。
10. 双击 HTML 文档，在浏览器中打开，并查看进度条动画。

> **注意** 在测试环境中加载文件时，如果要查看进度条的进度，请确保从 Internet 加载未缓存的文件，而不要加载本地文件。本地文件的加载速度太快，无法查看进度。或者，上传您的 SWF 文件，然后测试服务器上的文档。

相关信息，请参见第 532 页的 “关于加载 SWF 文件和根时间轴”。有关 MovieClipLoader 类的更多信息，请参见《ActionScript 2.0 语言参考》中的 `MovieClipLoader`。有关创建进度条动画的信息，请参见第 558 页的 “为加载 SWF 和图像文件创建进度动画”。

您可以在硬盘上找到图库应用的范例。这些文件提供了一些示例，可帮助您了解在向 SWF 文件加载图像文件时，如何使用 ActionScript 动态地控制影片剪辑。您可以在硬盘上的 Samples 文件夹中找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

关于加载 SWF 文件和根时间轴

ActionScript 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 `_lockroot`（`MovieClip._lockroot` 属性）强制将影片剪辑对 `_root` 的引用解析到它自己的时间轴，而不是解析到包含该影片剪辑的 SWF 文件的时间轴。有关更多信息，请参见第 317 页的“指定加载的 SWF 文件的根时间轴”。有关使用 `_root` 和 `_lockroot` 的更多信息，请参见第 651 页的第 19 章“ActionScript 2.0 的最佳做法和编码约定”。

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 System.security.allowDomain 的更多信息，请参见《ActionScript 2.0 语言参考》中的 `allowDomain`（`security.allowDomain` 方法）和第 618 页的“关于域、跨域安全性和 SWF 文件”。

关于加载和使用外部 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，如下例所示：

```
var song1_sound:Sound = new Sound();
```

使用这个新对象调用 `loadSound()` 来加载事件声音或声音流。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` 参数来指定声音是声音流还是事件声音。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 `my_sound` 的 Sound 对象，然后加载一个名为 song1.mp3 的 MP3 文件。在时间轴的第 1 帧中输入下面的 ActionScript：

```
var my_sound:Sound = new Sound();
my_sound.loadSound("http://www.helpexamples.com/flash/sound/song1.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指定时间（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。

有关更多信息，请参见以下主题：

- 第 534 页的 “加载 MP3 文件”
- 第 535 页的 “预加载 MP3 文件”
- 第 536 页的 “读取 MP3 文件中的 ID3 标签”

您可以在硬盘上的 Samples 文件夹中找到加载 MP3 文件的范例源文件 jukebox.fla。此范例演示如何使用数据类型、一般编码原则和几个组件创建一个自动唱片点唱机：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\Jukebox。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/Jukebox。

加载 MP3 文件

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 (song2.mp3) 加载到名为 `game_sound` 的 Sound 对象中，然后在完全下载时播放该声音：

加载 MP3 文件：

1. 创建一个名为 **loadMP3.fla** 的新 FLA 文件。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
var game_sound:Sound = new Sound();
game_sound.onLoad = function(success:Boolean):Void {
  if (success) {
    trace("Sound Loaded");
    game_sound.start();
  }
};
game_sound.loadSound("http://www.helpexamples.com/flash/sound/song2.mp3"
  false);‘
```

3. 选择 “控制” > “测试影片”，对声音进行测试。

对于在运行时加载的声音文件， Flash Player 只支持 MP3 声音文件类型。

有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `Sound.loadSound()`、`Sound.start()` 和 `Sound.onLoad`。有关预加载 MP3 文件的信息，请参见第 535 页的 “预加载 MP3 文件”。有关在加载声音文件时创建进度条动画的信息，请参见第 560 页的 “使用 ActionScript 为加载 MP3 文件创建进度条”。

您可以在硬盘上的 Samples 文件夹中找到加载 MP3 文件的范例源文件 jukebox.fla。此范例演示如何使用数据类型、一般编码原则和几个组件创建一个自动唱片点唱机。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\Jukebox。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/Jukebox。

预加载 MP3 文件

预加载 MP3 文件时，可以使用 `setInterval()` 函数创建轮询机制，该机制以预先确定的间隔检查为 Sound 或 NetStream 对象加载的字节数。若要跟踪 MP3 文件的下载进度，请使用 `Sound.getBytesLoaded()` 和 `Sound.getBytesTotal()` 方法。

下面的示例使用 `setInterval()` 以预先确定的间隔检查为 Sound 对象加载的字节数。

预加载 MP3 文件：

1. 创建一个名为 **preloadMP3.fla** 的新 FLA 文件。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
// 创建一个新的 Sound 对象来播放声音。
var songTrack:Sound = new Sound();
// 创建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。
function checkProgress (soundObj:Object):Void {
  var numBytesLoaded:Number = soundObj.getBytesLoaded();
  var numBytesTotal:Number = soundObj.getBytesTotal();
  var numPercentLoaded:Number = Math.floor(numBytesLoaded / numBytesTotal
  * 100);
  if (!isNaN(numPercentLoaded)) {
    trace(numPercentLoaded + "% loaded.");
  }
};
// 当文件完成加载之后，清除间隔轮询。
songTrack.onLoad = function ():Void {
  trace("load complete");
  clearInterval(poll);
};
// 加载 MP3 流文件并开始调用 checkProgress(),
songTrack.loadSound("http://www.helpexamples.com/flash/sound/song1.mp3",
  true);
var poll:Number = setInterval(checkProgress, 100, songTrack);
```

3. 选择 “控制” > “测试影片”，对声音进行测试。

   “输出”面板将显示加载进度。

您可以使用轮询技术预加载外部 FLV 文件。若要获取 FLV 文件的总字节数和当前加载的字节数，请使用 `NetStream.bytesLoaded` 和 `NetStream.bytesTotal` 属性（有关更多信息，请参见 bytesLoaded（NetStream.bytesLoaded 属性）和 bytesTotal（NetStream.bytesTotal 属性））。

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `MovieClip.getBytesLoaded()`、`MovieClip.getBytesTotal()`、`setInterval()`、`Sound.getBytesLoaded()` 和 `Sound.getBytesTotal()`。

有关创建进度条动画的信息，请参见第 560 页的 “使用 ActionScript 为加载 MP3 文件创建进度条”。

您可以在硬盘上的 Samples 文件夹中找到加载 MP3 文件的范例源文件 jukebox.fla。此范例演示如何使用数据类型、一般编码原则和几个组件创建一个自动唱片点唱机。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\Jukebox。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/Jukebox。

读取 MP3 文件中的 ID3 标签

ID3 标签是添加到 MP3 文件中的数据字段。ID3 标签包含有关文件的信息，如歌曲名称、专辑和歌手。

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。

下面的示例将名为 song1.mp3 的 MP3 文件加载到 `song_sound` Sound 对象中。当该文件的 ID3 标签可用时，`display_txt` 文本字段将显示歌手姓名和歌曲名称。

读取 MP3 文件中的 ID3 标签：

1. 创建一个名为 **id3.fla** 的新 FLA 文件。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
this.createTextField("display_txt", this.getNextHighestDepth(), 0, 0,
  100, 100);
display_txt.autoSize = "left";
display_txt.multiline = true;
var song_sound:Sound = new Sound();
song_sound.onLoad = function() {
  song_sound.start();
};
song_sound.onID3 = function():Void  {
  display_txt.text += "Artist:\t" + song_sound.id3.artist + "\n";
  display_txt.text += "Song:\t" + song_sound.id3.songname + "\n";
};
song_sound.loadSound("http://www.helpexamples.com/flash/sound/
  song1.mp3");
```

3. 选择 “控制” > “测试影片”，对声音进行测试。

ID3 标签便会出现在舞台上，并播放声音。

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），所以这些标签直到整个 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。因此，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，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《ActionScript 2.0 语言参考》中的 `id3 (Sound.id3 属性)`。

您可以在硬盘上的 Samples 文件夹中找到加载 MP3 文件的范例源文件 jukebox.fla。此范例演示如何使用数据类型、一般编码原则和几个组件创建一个自动唱片点唱机：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\Jukebox。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/Jukebox。

为库中的资源分配链接

您可以为库中的资源（例如影片剪辑和字体元件）分配链接标识符。在 Flash Basic 8 和 Flash Professional 8 中，您可以为库中的声音和图像资源设置链接标识符。这种方式支持结合共享库和新的 BitmapData 类使用图像文件和声音文件。

下面的示例为库中的一个位图图像添加了一个设置为 `myImage` 的链接。然后再将该图像添加到舞台，并使其可拖放。

对位图文件使用链接：

1. 创建一个名为 **linkBitmap.fla** 的新 FLA 文件。
2. 将一个位图图像导入到库中。
3. 右击 (Windows) 或按住 Control 键单击 (Macintosh) 库中的图像，然后从上下文菜单中选择 “链接”。
4. 选择 “为 ActionScript 导出”和 “在第一帧导出”，并在 “标识符”文本框中键入 **myImage**。
5. 单击 “确定”设置链接标识符。

6. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
import flash.display.BitmapData;
// 创建 imageBmp 并从库中附加位图。
var imageBmp:BitmapData = BitmapData.loadBitmap("myImage");
// 创建影片剪辑，并附加 imageBmp
this.createEmptyMovieClip("imageClip", 10);
imageClip.attachBitmap(imageBmp, 2);
// 使剪辑可以拖放
imageClip.onPress = function() {
     this.startDrag();
};
imageClip.onRelease = function() {
     this.stopDrag();
}
```

7. 选择 “控制” > “测试影片”，对该文档进行测试。

   库中的位图即出现在舞台上，而且该图像是可以拖放的。

关于使用 FLV 视频

FLV 文件格式包含用 Flash Player 编码的、用于交付的音频和视频数据。例如，如果您有 QuickTime 或 Windows Media 视频文件，便可使用编码器（如 Flash 8 视频编码器 或 Sorensen Squeeze）将该文件转换为 FLV 文件。

Flash Player 7 支持用 Sorenson Spark 视频编解码器编码的 FLV 文件。Flash Player 8 支持用 Flash Professional 8 中的 Sorenson Spark 或 On2 VP6 编码器编码的 FLV 文件。On2 VP6 视频编解码器支持 Alpha 通道。不同的 Flash Player 版本支持 FLV 的方式也不同。有关更多信息，请参见下表：

编解码器	SWF 文件版本（发布版本）	Flash Player 版本 （回放所需要的版本）
Sorenson Spark	6	6、7 或 8
	7	7、8
On2 VP6	6	8*
	7	8
	8	8

* 如果 SWF 文件加载 FLV 文件，您可以使用 On2 VP6 视频，但只要用户在使用 Flash Player 8 查看 SWF 文件，就必须为 Flash Player 8 重新发布 SWF 文件。只有 Flash Player 8 既支持发布又支持回放 On2 VP6 视频。

有关视频基础知识的信息（如流、渐进下载、尺寸、编码、导入和带宽考虑），请参见《使用 Flash》中的第 11 章 “使用视频”。

本部分讨论如何在不用组件的情况下使用 FLV 视频。您也可以使用 FLVPlayback 组件播放 FLV 文件，或使用 VideoPlayback 类创建可动态加载 FLV 文件的自定义视频播放器（请参见 www.macromedia.com/devnet/flash/ 或 www.macromedia.com/support/documentation/）。有关如何通过 FLVPlayback 组件和 Media 组件使用 FLV 视频的信息，请参见以下部分：

- 第 467 页的 “FLVPlayback 组件 （仅限 Flash Professional）”
- 第 769 页的 “媒体组件 （仅限 Flash Professional）”

您可以在 Flash Player 中使用 ActionScript 动态地播放外部 FLV 文件，而不把视频直接导入到 Flash 创作环境中。可以从 HTTP 地址或从本地文件系统播放 FLV 文件。若要播放 FLV 文件，可以使用 NetConnection 和 NetStream 类以及 Video 类的 `attachVideo()` 方法。有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `NetConnection`、`NetStream` 和 `attachVideo (Video.attachVideo 方法)`。

您可以将视频导入 Flash 创作工具，然后将其导出为 FLV 文件，这样就可创建 FLV 文件。如果有 Flash Professional 8，则可使用 “FLV 导出”插件从支持的视频编辑应用程序中导出 FLV 文件。

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：

- 无需降低回放速度就可以在 Flash 文档中使用较长的视频剪辑。可以使用缓存内存 的方式来播放外部 FLV 文件，这意味着可以将大型文件分成若干个小片段存储，对其进行动态访问，这种方式比嵌入的视频文件所需的内存更少。
- 外部 FLV 文件可以和它所在的 Flash 文档具有不同的帧频。例如，可以将 Flash 文档帧频设置为 30 帧 / 秒 (fps)，并将视频帧频设置为 21 fps。与嵌入的视频相比，此项设置可使您更好地控制视频，确保视频顺畅地回放。此项设置还允许您在不改变现有 Flash 内容的前提下以不同的帧频播放 FLV 文件。
- 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。

提示 若要从 Web 服务器加载 FLV 文件，则可能需要向您的 Web 服务器注册文件扩展名和 MIME 类型；请查看您的 Web 服务器文档。 FLV 文件的 MIME 类型是 video/x-flv。有关更多信息，请参见第 556 页的 “关于配置 FLV 文件以在服务器上寄宿”。

有关 FLA 视频的更多信息，请参见以下主题：

- 第 540 页的 “创建视频对象”
- 第 541 页的 “动态播放外部 FLV 文件”
- 第 542 页的 “创建视频旗标”
- 第 544 页的 “预加载 FLV 文件”
- 第 545 页的 “使用提示点”
- 第 554 页的 “使用元数据”
- 第 556 页的 “关于配置 FLV 文件以在服务器上寄宿”
- 第 557 页的 “关于在 Macintosh 上将本地 FLV 文件设定为目标”

创建视频对象

在使用 ActionScript 加载和操作视频之前，需要创建一个视频对象，并将其拖到舞台上，并赋予其实例名称。下面的示例描述了如何向应用程序添加视频实例。

创建视频对象：

1. 使文档在 Flash 创作工具中保持打开状态，从 “库”面板（“窗口” > “库”）的弹出菜单中选择 “新建视频”。
2. 在 “视频属性”对话框中，命名视频元件，并选择 “视频”（受 ActionScript 控制）。
3. 单击 “确定”以创建一个视频对象。
4. 将该视频对象从 “库”面板拖到舞台上，以创建视频对象实例。
5. 使视频对象在舞台上保持选中状态，在属性检查器（“窗口” > “属性” > “属性”）中的 “实例名称”文本框中键入 **my_video**。

   现在，舞台上已经有了一个视频实例，您可以为其添加 ActionScript，以不同的方式加载视频或操作该实例。

有关动态加载 FLV 文件的信息，请参见 “动态播放外部 FLV 文件”。有关创建视频旗标的信息，请参见第 542 页的 “创建视频旗标”。

动态播放外部 FLV 文件

您可以在运行时加载 FLV 文件，并在 SWF 文件中播放。可以将这些文件加载到视频对象或诸如 FLVPlayback 之类的组件中。下面的示例演示如何在视频对象中播放名为 clouds.flv 的文件。

在 Flash 文档中回放外部 FLV 文件：

1. 创建一个名为 **playFLV.fla** 的新 Flash 文档。
2. 在 “库”面板 （“窗口” > “库”）中，从 “库”弹出菜单中选择 “新建视频”。
3. 在 “视频属性”对话框中，命名视频元件并选择 “视频”（受 ActionScript 控制）。
4. 单击 “确定”以创建视频对象。
5. 将视频对象从 “库”面板拖到舞台上，以创建视频对象实例。
6. 使视频对象在舞台上保持选中状态，在属性检查器 （“窗口” > “属性” > “属性”）中的 “实例名称”文本框中键入 **my_video**。
7. 在时间轴中选择第 1 帧，然后打开 “动作”面板 （“窗口” > “动作”）。
8. 在 “动作”面板中键入以下代码：

```
this.createTextField("status_txt", 999, 0, 0, 100, 100);
status_txt.autoSize = "left";
status_txt.multiline = true;
// 创建一个 NetConnection 对象
var my_nc:NetConnection = new NetConnection();
// 创建一个本地流连接
my_nc.connect(null);
// 创建一个 NetStream 对象并定义一个 onStatus() 函数
var my_ns:NetStream = new NetStream(my_nc);
my_ns.onStatus = function(infoObject:Object):Void {
  status_txt.text += "status (" + this.time + " seconds)\n";
  status_txt.text += "\t Level: " + infoObject.level + "\n";
  status_txt.text += "\t Code: " + infoObject.code + "\n\n";
};
// 将 NetStream 视频输入信号附加到 Video 对象
my_video.attachVideo(my_ns);
// 设置缓冲时间
my_ns.setBufferTime(5);
// 开始播放 FLV 文件
my_ns.play("http://www.helpexamples.com/flash/video/clouds.flv");
```

9. 选择 “控制” > “测试影片”，对该文档进行测试。

有关预加载 FLV 文件的信息，请参见第 544 页的 “预加载 FLV 文件”。有关将 FLV 视频动态加载到组件中的信息，请参见第 469 页的 “创建具有 FLVPlayback 组件的应用程序”。有关 FLV 文件和服务器，以及 FLV 文件和在 Macintosh 上本地播放 FLV 文件的信息，请参见第 556 页的 “关于配置 FLV 文件以在服务器上寄宿”。

创建视频旗标

旗标及其它 Flash 广告中的视频内容通常用于广告，如显示 Flash 格式的电影预告片或电视广告。下面的示例演示如何在 FLA 文件中创建视频实例和添加 ActionScript，以创建包含视频的旗标广告。

创建视频旗标：

1. 创建一个名为 **vidBanner.fla** 的新 Flash 文档。
2. 选择 “修改” > “文档”。
3. 更改 FLA 文件的大小，在宽度文本框中键入 **468**，并在高度文本框中键入 **60**。
4. 在 “库”面板 （“窗口” > “库”）中，从 “库”选项中选择 “新建视频”。
5. 在 “视频属性”对话框中，命名视频元件，并选择 “视频”（受 ActionScript 控制）。
6. 单击 “确定”以创建一个视频对象。
7. 将该视频对象从 “库”面板拖到舞台上，以创建视频实例。
8. 使视频对象在舞台上保持选中状态，在属性检查器 （“窗口” > “属性” > “属性”）中的 “实例名称”文本框中键入 **my_video**。
9. 保持视频实例的选中状态，并在属性检查器的宽度文本框中键入 **105**，在高度文本框中键入 **60**。
10. 将视频实例拖到舞台上的某一位置，或使用属性检查器设置其 x 和 y 坐标。
11. 在时间轴中选择第 1 帧，然后打开 “动作”面板 （“窗口” > “动作”）。
12. 将以下代码添加到 “动作”面板中：

```
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.setBufferTime(5);
my_ns.play("http://www.helpexamples.com/flash/video/vbanner.flv");
```

13. 选择 “插入” > “时间轴” > “图层”，创建一个新的图层，并将其命名为 **button**。
14. 从 “工具”面板中选择 “矩形”工具。
15. 在 “工具”面板的 “颜色”部分，单击铅笔图标以选择 “笔触”颜色控件。
16. 选择 “没有颜色”，这一操作将禁用矩形的轮廓。
17. 在舞台上沿对角方向拖动指针，以创建一个矩形。

    矩形的大小随意，因为将通过属性检查器调整其大小。

18. 在 “工具”面板中单击 “选择”工具，然后单击舞台上的矩形，选择该矩形。

19. 使矩形保持选中状态，在属性检查器的宽度文本框中键入 **468**，在高度文本框中键入 **60**。然后将 X 和 Y 坐标 （X 和 Y 文本框）更改为 **0**。

20. 使矩形在舞台上保持选中状态，按 F8 将矩形更改为元件。

21. 在 “转换为元件”对话框的 “名称”文本框中键入 **invisible btn** ，选择 “按钮”，然后单击 “确定”。

22. 双击舞台上的新按钮，进入元件编辑模式。

    矩形当前位于所创建按钮的第一个 “弹起”帧上。这是按钮的 “弹起”状态，即按钮在舞台上时用户所看到的状态。不过，您希望按钮在舞台上不可见，因此需要将矩形移动到 “点击”帧，即为按钮的点击区域 （用户可单击激活按钮动作的活动区域）。

23. 在 “弹起”帧上单击关键帧，并按住鼠标键将关键帧拖到 “点击”帧。

    现在便可在整个旗标区域中单击，但不会看到旗标上按钮的外形。

24. 单击第 1 个场景返回到主时间轴。

    在旗标区域的上方出现一个蓝绿色矩形，表示不可见按钮的点击区域。

25. 选择您所创建的按钮，打开属性检查器，在 “实例名称”文本框中键入 **inv_btn**。

26. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

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

27. 再对旗标进行其它的修改，如添加图形或文本。

28. 选择 “控制” > “测试影片”在 Flash Player 中测试此旗标。

在此示例中，您创建了一个旗标，并将其大小调整为 “互动广告局”所指定的标准大小。有关标准广告大小（以及其它许多非常有用的指导）的信息，请参见 “互动广告局”的 “Standards and Guidelines”（标准和指导）页：www.iab.net/standards/adunits.asp。

除了遵守标准化指南外，还要确保符合广告服务、客户或发布广告的 Web 站点的广告规范要求。如果向广告公司提交旗标，请确保文件满足指定文件大小、尺寸、目标 Flash Player 版本和帧频要求。还有，您可能还需要考虑所用媒体的类型、FLA 文件中使用的按钮代码等方面的规则。

预加载 FLV 文件

若要跟踪 FLV 文件的下载进度，请使用 NetStream.bytesLoaded 和 NetStream.bytesTotal 属性。若要获取 FLV 文件的总字节数和当前下载的字节数，请使用 NetStream.bytesLoaded 和 NetStream.bytesTotal 属性。

下面的示例使用 bytesLoaded 和 bytesTotal 属性显示将 video1.flv 加载到名为 my_video 的视频对象实例中的进度。将动态创建一个名为 loaded_txt 的文本字段，以便显示有关加载进度的信息。

预加载 FLV 文件：

1. 创建一个名为 **preloadFLV.fla** 的新 FLA 文件。
2. 在 “库”面板 （“窗口” > “库”）中，从 “库”弹出菜单中选择 “新建视频”。
3. 在 “视频属性”对话框中，命名视频元件并选择 “视频”（受 ActionScript 控制）。
4. 单击 “确定”以创建一个视频对象。
5. 将将视频对象从 “库”面板拖到舞台上，以创建视频对象实例。
6. 使视频对象在舞台上保持选中状态，在属性检查器 （“窗口” > “属性” > “属性”）中的 “实例名称”文本框中键入 **my_video**。
7. 使视频实例保持选中状态，然后在属性检查器的宽度文本框中键入 **320**，在高度文本框中键入 **213**。
8. 在时间轴中选择第 1 帧，然后打开 “动作”面板 （“窗口” > “动作”）。
9. 在 “动作”面板中键入以下代码：

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("http://www.helpexamples.com/flash/video/
  lights_short.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10,
  160, 22);
var loaded_interval:Number = setInterval(checkBytesLoaded, 500,
  stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
  var pctLoaded:Number = Math.round(my_ns.bytesLoaded / my_ns.bytesTotal
  * 100);
  loaded_txt.text = Math.round(my_ns.bytesLoaded / 1000) + " of " +
  Math.round(my_ns.bytesTotal / 1000) + " KB loaded (" + pctLoaded +
  "%)";
  progressBar_mc.bar_mc._xscale = pctLoaded;
  if (pctLoaded >= 100) {
    clearInterval(loaded_interval);
  }
}
```

10. 选择 “控制” > “测试影片”，对代码进行测试。

提示	如果进度条立刻走完，则表明视频被缓存到了硬盘上（可能是早已测试过此示例，也可能在其它过程中早已加载了它）。如果发生这种情况，请将 FLV 文件上载到服务器，然后再加载。

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 流进行缓冲的秒数。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `setBufferTime (NetStream.setBufferTime 方法)`、`getBytesLoaded (MovieClip.getBytesLoaded 方法)`、`getBytesTotal (MovieClip.getBytesTotal 方法)`、`bytesLoaded (NetStream.bytesLoaded 属性)`、`bytesTotal (NetStream.bytesTotal 属性)` 和 `setInterval 函数`。

使用提示点

您可以对 Flash 视频使用几种不同类型的提示点。可以使用 ActionScript 与在创建 FLV 文件时嵌入到 FLV 文件中的提示点进行交互，也可以与用 ActionScript 创建的提示点进行交互。

导航提示点 您可以在编码 FLV 文件时，将导航提示点嵌入到 FLV 流和 FLV 元数据包中。使用导航提示点可以使用户搜索到文件的指定部分。

事件提示点 您可以在编码 FLV 文件时，将事件提示点嵌入到 FLV 流和 FLV 元数据包中。还可以编写代码来处理在 FLV 回放期间于指定点上触发的事件。

ActionScript 提示点 使用 ActionScript 代码创建的外部提示点。您可以编写代码来触发这些与视频回放有关的提示点。这些提示点的精确度要低于嵌入的提示点（最高时相差 1/10 秒），因为视频播放器单独跟踪这些提示点。

由于导航提示点会在指定的提示点位置创建一个关键帧，因此可以使用代码将视频播放器的播放头移动到该位置。您可以在 FLV 文件中设置一些特殊点，让用户搜索这些点。例如，视频可能会具有多个章节或段，在这种情况下您就可以在视频文件中嵌入导航提示点，以此方式来控制视频。

如果您计划创建一个应用程序，希望用户能在其中导航至提示点，则应在编码文件时创建并嵌入提示点，而不应使用 ActionScript 提示点。您应将提示点嵌入到 FLV 文件中，因为这些提示点需要更加精确的处理。有关用提示点编码 FLV 文件的更多信息，请参见《使用 Flash》中的第 245 页的 “嵌入指令点（仅限 Flash Professional）”。

您可以通过编写 ActionScript 来访问提示点参数。提示点参数是用 `cuePoint` 事件 (`event.info.parameters`) 接收的事件对象的一部分。

跟踪 FLV 文件中的提示点

您可以使用 NetStream.onMetaData 跟踪 FLV 文档中嵌入的提示点。需要递归返回的元数据的结构，以查看提示点信息。

下面的代码跟踪 FLV 文件中的提示点：

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.onMetaData = function(metaProp:Object) {
  trace("The metadata:");
  traceMeta(metaProp);
  // traceObject(metaProp, 0);
};
my_video.attachVideo(stream_ns);
stream_ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

function traceMeta(metaProp:Object):Void {
  var p:String;
  for (p in metaProp) {
    switch (p) {
    case "cuePoints" :
      trace("cuePoints: ");
      //cycles through the cue points
      var cuePointArr:Array = metaProp[p];
      for (var j:Number = 0; j < cuePointArr.length; j++) {
        //cycle through the current cue point parameters
        trace("\t cuePoints[" + j + "]:");
        var currentCuePoint:Object = metaProp[p][j];
        var metaPropPJParams:Object = currentCuePoint.parameters;
        trace("\t\t name: " + currentCuePoint.name);
        trace("\t\t time: " + currentCuePoint.time);
        trace("\t\t type: " + currentCuePoint.type);
        if (metaPropPJParams != undefined) {
          trace("\t\t parameters:");
          traceObject(metaPropPJParams, 4);
        }
      }
      break;
    default :
      trace(p + ": " + metaProp[p]);
      break;
    }
  }
}
function traceObject(obj:Object, indent:Number):Void {
  var indentString:String = "";
  for (var j:Number = 0; j < indent; j++) {
    indentString += "\t";
  }
```

```
  for (var i:String in obj) {
    if (typeof(obj[i]) == "object") {
      trace(indentString + " " + i + ": [Object]");
      traceObject(obj[i], indent + 1);
    } else {
      trace(indentString + " " + i + ": " + obj[i]);
    }
  }
}
```

显示以下输出：

```
The metadata:
canSeekToEnd: true
cuePoints:
  cuePoints[0]:
    name: point1
    time: 0.418
    type: navigation
    parameters:
      lights: beginning
  cuePoints[1]:
    name: point2
    time: 7.748
    type: navigation
    parameters:
      lights: middle
  cuePoints[2]:
    name: point3
    time: 16.02
    type: navigation
    parameters:
      lights: end
audiocodecid: 2
audiodelay: 0.038
audiodatarate: 96
videocodecid: 4
framerate: 15
videodatarate: 400
height: 213
width: 320
duration: 16.334
```

有关通过 FLVPlayback 组件使用提示点的信息，请参见 “通过 FLVPlayback 组件使用嵌入的提示点（仅限 Flash Professional）”。

通过 FLVPlayback 组件使用嵌入的提示点（仅限 Flash Professional）

在使用 FLVPlayback 组件时，可以在属性检查器中查看 FLV 文件的提示点。设置了 FLVPlayback 实例的 `contentPath` 属性后，便可以查看视频文件中嵌入的任何提示点了。通过使用“参数”选项卡，可以查找 `cuePoints` 属性，然后单击放大镜图标，查看文件中的提示点列表。

注释	若要在“参数”选项卡上查看提示点，必须在 contentPath 文本框中键入 FLV 文件的名称，而不是使用代码分配 contentPath。

下面的示例演示如何通过 FLVPlayback 组件使用提示点信息。

通过 FLVPlayback 组件使用提示点：

1. 创建一个名为 **cueFlv.fla** 的新 Flash 文档。
2. 打开“组件”面板（“窗口”>“组件”），然后将 FLVPlayback 和 TextArea 组件的实例拖到舞台上。
3. 选择 TextArea 组件，然后在属性检查器（“窗口”>“属性”>“属性”）的“实例名称”文本框中键入 **my_ta**。
4. 使 TextArea 组件保持选中状态，然后在宽度文本框中键入 **200**，在高度文本框中键入 **100**。
5. 在舞台上选择 FLVPlayback 实例，然后在“实例名称”文本框中键入 **my_flvPb**。
6. 在时间轴中选择第 1 帧，然后在“动作”面板中键入下面的代码：

```
var my_flvPb:mx.video.FLVPlayback;
var my_ta:mx.controls.TextArea;
my_flvPb.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject:Object) {
  my_ta.text += "Elapsed time in seconds: " + my_flvPb.playheadTime +
  "\n";
};
my_flvPb.addEventListener("cuePoint",listenerObject);
```

7. 选择“控制”>“测试影片”对 SWF 文件进行测试。

   在播放头经过文档中嵌入的每个提示点时，便会在 TextArea 实例中出现已运行时间。

有关使用 FLVPlayback 组件的更多信息，请参见第 467 页的“FLVPlayback 组件（仅限 Flash Professional）”。

用 ActionScript 创建提示点，以使用组件 （仅限 Flash Professional）

您可以用 ActionScript 创建提示点，然后再结合视频对象实例或一个视频播放器组件（Flash Player 8 的 FLVPlayback 或 Flash Player 7 的 MediaPlayback）使用这些提示点。下面的示例演示如何轻松使用 ActionScript 代码创建提示点并使用脚本访问这些提示点。

注意

如果打算向应用程序添加导航功能，则可以将导航提示点嵌入到文档中。有关更多信息，请参见第 545 页的 “使用提示点”。有关使用嵌入的提示点的示例，请参见第 548 页的 “通过 FLVPlayback 组件使用嵌入的提示点 （仅限 Flash Professional）”。

通过 FLVPlayback 组件创建和使用提示点：

1. 创建一个名为 **cueFlvPb.fla** 的新 Flash 文档。
2. 将 FLVPlayback 组件的实例从 “组件” 面板 （“窗口” > “组件”）拖到舞台上。

   该组件位于 FLVPlayback - Player 8 文件夹中。
3. 选择该组件并打开属性检查器 （“窗口” > “属性” > “属性”）。
4. 在 “实例名称” 文本框中键入 **my_flvPb**。
5. 将 TextArea 组件的实例从 “组件” 面板拖到舞台上。
6. 选择该 TextArea 组件，并在 “实例名称” 文本框中键入 **my_ta**。
7. 使 TextArea 组件保持选中状态，然后在宽度文本框中键入 **200**，在高度文本框中键入 **100**。
8. 在时间轴中选择第 1 帧，然后在 “动作” 面板中键入下面的代码：

```
var my_flvPb:mx.video.FLVPlayback;
my_flvPb.contentPath = "http://www.helpexamples.com/flash/video/
  clouds.flv";

// 创建 cuePoint 对象。
var cuePt:Object = new Object();
cuePt.time = 1;
cuePt.name = "elapsed_time";
cuePt.type = "actionscript";
// 添加 AS 提示点。
my_flvPb.addASCuePoint(cuePt);

// 添加另一个 AS 提示点。
my_flvPb.addASCuePoint(2, "elapsed_time2");

// 在文本字段中显示提示点信息。
```

```
var listenerObject:Object = new Object();
listenerObject.cuePoint = function(eventObject) {
  my_ta.text += "Elapsed time in seconds: " + my_flvPb.playheadTime +
  "\n";
};
my_flvPb.addEventListener("cuePoint",listenerObject);
```

9. 选择 “控制” > “测试影片”，对代码进行测试。

   将在 “输出”面板中输出以下提示点：

```
Elapsed time in seconds: 1.034
Elapsed time in seconds: 2.102
```

有关 `addASCuePoint()` 的信息，请参见第 508 页的 “FLVPlayback.addASCuePoint()”。有关使用提示点和 FLVPlayback 组件的信息，请参见第 474 页的 “使用提示点”和第 467 页的 “FLVPlayback 组件 （仅限 Flash Professional）”。

下面的示例演示在运行时如何添加提示点，然后在 MediaPlayback 组件中播放 FLV 文件时跟踪提示点。

通过 MediaPlayback 组件创建和使用提示点：

1. 创建一个名为 cuePointMP.fla 的新 Flash 文档
2. 将 MediaPlayback 组件的实例从 “组件”面板 （“窗口” > “组件”）拖到舞台上。

   该组件位于 Media - Player 6 - 7 文件夹中。
3. 选择该组件，并打开属性检查器 （“窗口” > “属性” > “属性”）。
4. 在 “实例名称”文本框中键入 **my_mp**。
5. 选择 “参数”选项卡，单击 “启动组件检查器”。
6. 在 “组件”检查器的 URL 文本框中键入 **http://www.helpexamples.com/flash/video/clouds.flv**。
7. 打开 “动作”面板 （“窗口” > “动作”），在 “脚本”窗格中键入以下代码：

```
import mx.controls.MediaPlayback;
var my_mp:MediaPlayback;
my_mp.autoPlay = false;
my_mp.addEventListener("cuePoint", doCuePoint);
my_mp.addCuePoint("one", 1);
my_mp.addCuePoint("two", 2);
my_mp.addCuePoint("three", 3);
my_mp.addCuePoint("four", 4);
function doCuePoint(eventObj:Object):Void {
     trace(eventObj.type + " = {cuePointName:" + eventObj.cuePointName + "
  cuePointTime:" + eventObj.cuePointTime + "}");
}
```

8. 选择 “控制” > “测试影片”，对代码进行测试。

   将在 “输出” 面板中输出以下提示点：

```
cuePoint = {cuePointName:one cuePointTime:1}
cuePoint = {cuePointName:two cuePointTime:2}
cuePoint = {cuePointName:three cuePointTime:3}
cuePoint = {cuePointName:four cuePointTime:4}
```

有关使用 MediaPlayback 组件的更多信息，请参见第 769 页的 “媒体组件 （仅限 Flash Professional）”。有关使用 FLVPlayback 组件的更多信息，请参见第 467 页的 “FLVPlayback 组件 （仅限 Flash Professional）”。

用提示点添加搜索功能 （仅限 Flash Professional）

您可以将导航提示点嵌入到 FLV 文件中，以向应用程序添加搜索功能。FLVPlayback 组件的 `seekToNavCuePoint()` 方法用指定名称定位 FLV 文件中指定时间或指定时间之后的提示点。您可以将名称指定为一个字符串 （例如 `"part1"` 或 `"theParty"`）。

您还可以使用 `seekToNextNavCuePoint()` 方法，此方法根据当前的 `playheadTime` 搜索下一个导航提示点。您可以向该方法传递一个参数 `time`，该参数即为查找下一个导航提示点的起始时间。默认值为当前的 `playheadTime`。

或者，您还可以使用 `seek()` 方法搜索 FLV 文件的指定持续时间。

在下面的示例中，您将添加一个可在 FLVPlayback 组件所播放的 FLV 文件的提示点之间跳转或跳过指定持续时间的按钮，还将添加一个可跳转至指定提示点的按钮。

搜索指定的持续时间：

1. 创建一个名为 **seekduration.fla** 的新 Flash 文档。
2. 从 “组件” 面板 （“窗口” > “组件”）中拖动 FLVPlayback 组件的一个实例。

   该组件位于 FLVPlayback - Player 8 文件夹中。
3. 选择该组件，并打开属性检查器 （“窗口” > “属性” > “属性”）。
4. 在 “实例名称” 文本框中键入 **my_flvPb**。
5. 将 Button 组件的实例从 “组件” 面板拖到舞台上。
6. 选择该 Button 组件，并在 “实例名称” 文本框中键入 **my_button**。
7. 在时间轴中选择第 1 帧，然后在 “动作” 面板中键入下面的代码：

```
import mx.controls.Button;
import mx.video.FLVPlayback;
var seek_button:Button;
var my_flvPb:FLVPlayback;
my_flvPb.autoPlay = false;
my_flvPb.contentPath = "http://www.helpexamples.com/flash/video/
  sheep.flv";
seek_button.label = "Seek";
```

```
seek_button.addEventListener("click", seekFlv);
function seekFlv(eventObj:Object):Void {
  // 搜索 2 秒钟
  my_flvPb.seek(2);
}
```

8. 选择 “控制” > “测试影片”，对代码进行测试。

   在单击按钮时，视频播放头将移动到指定的持续时间：视频中第 2 秒的位置。

用 FLVPlayback 组件添加搜索功能：

1. 创建一个名为 **seek1.fla** 的新 Flash 文档。
2. 从 “组件”面板 （“窗口” > “组件”）中拖动 FLVPlayback 组件的实例。

   该组件位于 FLVPlayback - Player 8 文件夹中。
3. 选择该组件，打开属性检查器 （“窗口” > “属性” > “属性”）。
4. 在 “实例名称”文本框中键入 **my_flvPb**。
5. 将 Button 组件的实例从 “组件”面板拖到舞台上。
6. 选择该 Button 组件，并在 “实例名称”文本框中键入 **my_button**。
7. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
import mx.video.FLVPlayback;
var my_flvPb:FLVPlayback;
my_flvPb.autoPlay = false;
my_flvPb.contentPath = "http://www.helpexamples.com/flash/video/
  cuepoints.flv";
my_button.label = "Next cue point";

function clickMe(){
  my_flvPb.seekToNextNavCuePoint();
}
my_button.addEventListener("click", clickMe);
```

8. 选择 “控制” > “测试影片”，对代码进行测试。

   cuepoints.flv 文件包含三个导航提示点：这三个提示点分别靠近视频文件的开头、中间和结尾处。在单击按钮时，FLVPlayback 实例搜索下一个提示点，直到到达视频文件中的最后一个提示点为止。

您还可以使用 `seekToCuePoint()` 方法搜索 FLV 文件中的指定提示点，如下面的示例所示。

搜索指定的提示点：

1. 创建一个名为 **seek2.fla** 的新 Flash 文档。
2. 从 “组件”面板 （“窗口” > “组件”）中拖动 FLVPlayback 组件的实例。

   该组件位于 FLVPlayback - Player 8 文件夹。
3. 选择该组件，打开属性检查器 （“窗口” > “属性” > “属性”）。

4. 在 “实例名称”文本框中键入 **my_flvPb**。
5. 使 FLVPlayback 实例保持选中状态，然后单击 “参数”选项卡。
6. 在 contentPath 文本框中键入 **http://www.helpexamples.com/flash/video/cuepoints.flv**。

   当您在 contentPath 文本框中键入 URL 时，提示点便会出现在 “参数”选项卡（cuePoint 参数的旁边）中。因此，您可以确定要在代码中查找的提示点的名称。如果单击放大镜图标，则可以在表中查看视频文件的所有提示点和每个提示点的相关信息。
7. 将 Button 组件的实例从 “组件”面板拖到舞台上。
8. 选择该 Button 组件，并在 “实例名称”文本框中键入 **my_button**。
9. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
import mx.video.FLVPlayback;
var my_flvPb:FLVPlayback;
my_flvPb.autoPlay = false;
my_button.label = "Seek to point2";

function clickMe(){
  my_flvPb.seekToNavCuePoint("point2");
}
my_button.addEventListener("click", clickMe);
```

10. 选择 “控制” > “测试影片”，对代码进行测试。

    cuepoints.flv 文件包含三个导航提示点：这三个提示点分别靠近视频文件的开头、中间和结尾处。当您单击该按钮时， FLVPlayback 实例将搜索指定的提示点 (`point2`)。

有关提示点的更多信息，请参见第 474 页的 “使用提示点”。有关 FLVPlayback 组件的更多信息，请参见第 467 页的 “FLVPlayback 组件 （仅限 Flash Professional）”。

使用元数据

您可以使用 onMetaData 方法查看 FLV 文件中的元数据信息。元数据包含 FLV 文件的相关信息，如持续时间、宽度、高度和帧频。添加到 FLV 文件中的元数据信息取决于编码 FLV 文件时所使用的软件或添加元数据信息时所使用的软件。

注意	如果视频文件没有元数据信息，则可以使用工具将元数据信息添加到文件中。

若要使用 NetStream.onMetaData，必须具有包含元数据的 Flash 视频。如果用 Flash 8 视频编码器对 FLV 文件进行编码，则该 FLV 文件中将包含元数据信息（有关用 Flash 8 视频编码器编码的 FLV 文件中元数据的列表，请参见下面的示例）。

注意	Flash Video Exporter 1.2 和更高版本（包括 Flash 8 Video Exporter）会向 FLV 文件中添加元数据。 Sorenson Squeeze 4.1 和更高版本也会向视频文件中添加元数据。

下面的示例使用 NetStream.onMetaData 来跟踪用 Flash 8 视频编码器编码的 FLV 文件的元数据信息。

使用 NetStream.onMetaData 查看元数据信息：

1. 创建一个名为 **flvMetadata.fla** 的新 FLV 文件。
2. 在 “库”面板 （“窗口” > “库”）中，从 “库”弹出菜单中选择 “新建视频”。
3. 在 “视频属性”对话框中，命名视频元件并选择 “视频”（受 ActionScript 控制）。
4. 单击 “确定”以创建一个视频对象。
5. 将该视频对象从 “库”面板拖到舞台上，以创建视频对象实例。
6. 使视频对象在舞台上保持选中状态，在属性检查器 （“窗口” > “属性” > “属性”）中的 “实例名称”文本框中键入 **my_video**。
7. 使视频实例保持选中状态，在宽度文本框中键入 **320**，在高度文本框中键入 **213**。
8. 在时间轴中选择第 1 帧，然后打开 “动作”面板 （“窗口” > “动作”）。

9. 在 “动作”面板中键入以下代码：

```
// 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 nStream:NetStream = new NetStream(netConn);
// 将 NetStream 视频输入信号附加到 Video 对象。
my_video.attachVideo(nStream);
// 设置缓冲时间。
nStream.setBufferTime(30);
// 播放 FLV 文件。
nStream.play("http://www.helpexamples.com/flash/video/lights_short.flv");
// 输出元数据。
nStream.onMetaData = function(myMeta) {
     for (var i in myMeta) {
          trace(i + ":\t" + myMeta[i])
     }
};
```

10. 选择 “控制” > “测试影片”，对代码进行测试。

这样便会在 “输出”面板中看到以下信息：

```
canSeekToEnd:true
audiocodecid:2
audiodelay:0.038
audiodatarate:96
videocodecid:4
framerate:15
videodatarate:400
height:213
width:320
duration:8.04
```

注释 如果您的视频没有音频，则与音频相关的元数据信息 （如 `audiodatarate`）将返回 `undefined`，因为在编码期间没有将音频信息添加到元数据中。

您还可以使用以下格式显示大部分元数据信息。例如，以下代码显示 FLV 文件的持续时间：

```
nStream.onMetaData = function(myMeta) {
  trace("FLV duration: " + myMeta.duration + " sec.");
};
```

此格式不能跟踪 `cuePoint` 元数据信息。有关跟踪提示点的信息，请参见第 546 页的 “跟踪 FLV 文件中的提示点”。

关于配置 FLV 文件以在服务器上寄宿

在处理 FLV 文件时，您可能需要配置服务器以便于处理 FLV 文件格式。多用途 Internet 邮件扩展 (MIME) 是标准的数据规范，允许您通过 Internet 连接发送非 ASCII 文件。Web 浏览器和电子邮件客户端经过配置，可以解释多种 MIME 类型，因此它们可以发送和接收视频、音频、图形和格式化文本。若要从 Web 服务器加载 FLV 文件，则可能需要向您的 Web 服务器注册文件扩展名和 MIME 类型，因此应当检查您的 Web 服务器文档。FLV 文件的 MIME 类型是 `video/x-flv`。下面列出了 FLV 文件类型的完整信息：

Mime 类型：video/x-flv

文件扩展名：.flv

必需的参数：无

可选的参数：无

编码注意事项：FLV 文件是二进制文件；有些应用程序可能需要设置应用程序 / 八位字节流子类型。

安全问题：无

已发布的规范：www.macromedia.com/go/flashfileformat。

Microsoft 更改了在 Microsoft Internet 信息服务 (IIS) 6.0 Web 服务器中处理流媒体的方式，不再采用早期版本中的处理方式。早期版本的 IIS 不需要对 Flash 视频流做任何修改。在 Windows 2003 附带的默认 Web 服务器 IIS 6.0 中，服务器需要借助 MIME 类型来确认 FLV 文件为流媒体。

当采用流式媒体的方式加载外部 FLV 文件的 SWF 文件被置于 Microsoft Windows 2003 服务器上，并在浏览器中查看时，可以正确播放 SWF 文件，但 FLV 视频却不能采用流式媒体的方式加载。这个问题会影响到放置在 Windows 2003 服务器上的所有 FLV 文件，包括用早期版本的 Flash 创作工具 (Macromedia Flash Video Kit for Dreamweaver MX 2004) 制作的那些文件。如果在其它操作系统上对这些文件进行测试，则这些文件可以正常工作。

有关配置 Microsoft Windows 2003 和 Microsoft IIS Server 6.0 以采用流式媒体的方式加载 FLV 视频的信息，请访问 www.macromedia.com/go/tn_19439。

关于在 Macintosh 上将本地 FLV 文件设定为目标

如果您尝试从 Macintosh 计算机的非系统驱动器上用以斜杠 (/) 表示的路径来播放非系统驱动器上的 FLV，将无法播放视频。非系统驱动器 包括 （但不限于） CD-ROM、硬盘分区、可移除的存储媒体以及连接的存储设备。

注释	导致失败发生的原因是操作系统的局限，而非 Flash 或 Flash Player 的问题。

若要从 Macintosh 的非系统驱动器上播放 FLV 文件，需使用基于冒号记号 (:) 的绝对路径来代替基于斜杠记号 (/) 的路径引用该文件。下面的列表显示了这两种记号的不同：

基于斜杠的记号 myDrive/myFolder/myFLV.flv

基于冒号的记号 (Macintosh) myDrive:myFolder:myFLV.flv

您还可以为想让 Macintosh 回放的 CD-ROM 创建一个放映文件。有关 Macintosh CD-ROM 和 FLV 文件的最新信息，请参见 www.macromedia.com/go/3121b301。

关于为媒体文件创建进度动画

ActionScript 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。您可以创建进度条或进度动画，用可视形式显示加载进度或已加载的内容量。

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以检查下载进度。有关更多信息，请参见第 529 页的 “加载外部 SWF 和图像文件”。

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 535 页的 “预加载 MP3 文件”。

有关创建进度条以加载媒体文件的信息，请参见以下主题：

- 第 558 页的 “为加载 SWF 和图像文件创建进度动画”
- 第 560 页的 “使用 ActionScript 为加载 MP3 文件创建进度条”
- 第 562 页的 “用 ActionScript 为加载 FLV 文件创建进度条”

您可以找到一个用脚本动画创建进度条动画的范例源文件。可在您的硬盘上的 Samples 文件夹中找到 tweenProgress.fla 文件：

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

为加载 SWF 和图像文件创建进度动画

当您在应用程序中加载大型 SWF 或图像文件时，可能想创建一个动画来显示加载进度。您可以创建一个进度条，在加载动画时显示进度增长。也可以创建一个随文件加载进度而变化的动画。有关加载 SWF 和图像文件的信息，请参见第 529 页的 “加载外部 SWF 和图像文件”。。

下面的示例演示如何使用 MovieClipLoader 类和 Drawing API 显示图像文件的加载进度。

为加载图像文件或 SWF 文件创建进度条：

1. 创建一个名为 **loadImage.fla** 的新 Flash 文档。
2. 选择 “修改” > “文档”，在宽度文本框中键入 **700**，在高度文本框中键入 **500**，从而更改文档的尺寸。
3. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
// 创建剪辑来承载您的内容
this.createEmptyMovieClip("progressBar_mc", 0);
progressBar_mc.createEmptyMovieClip("bar_mc", 1);
progressBar_mc.createEmptyMovieClip("stroke_mc", 2);
// 使用绘画方法创建一个进度条
with (progressBar_mc.stroke_mc) {
   lineStyle(0, 0x000000);
   moveTo(0, 0);
   lineTo(100, 0);
   lineTo(100, 10);
   lineTo(0, 10);
   lineTo(0, 0);
}
with (progressBar_mc.bar_mc) {
   beginFill(0xFF0000, 100);
   moveTo(0, 0);
   lineTo(100, 0);
   lineTo(100, 10);
   lineTo(0, 10);
   lineTo(0, 0);
   endFill();
   _xscale = 0;
}
progressBar_mc._x = 2;
progressBar_mc._y = 2;
// 加载进度
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
   progressBar_mc.bar_mc._xscale = 0;
};
mclListener.onLoadProgress = function(target_mc:MovieClip,
  bytesLoaded:Number, bytesTotal:Number) {
```

```
    progressBar_mc.bar_mc._xscale = Math.round(bytesLoaded/
   bytesTotal*100);
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
    progressBar_mc.removeMovieClip();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._height = 500;
    target_mc._width = 700;
};
// 创建一个剪辑来承载图像。
this.createEmptyMovieClip("image_mc", 100);
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
/* 将图像加载到剪辑中。
You can change the following URL to a SWF or another image file. */
image_mcl.loadClip("http://www.helpexamples.com/flash/images/gallery1/
   images/pic3.jpg", image_mc);
```

4. 选择“控制”>“测试影片”来查看图像加载并观察进度条。

注意 如果您是第二次测试此代码，图像将为已缓存状态，因此进度条会立即走完。若要执行多次测试，请使用不同的图像并从外部源加载它们。如果使用本地源测试应用程序，可能会出现问题，因为本地内容的加载速度太快。

您可以找到一个用脚本动画创建进度条动画的范例源文件。可以在您的硬盘上的 Samples 文件夹中找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

您还能找到图库应用的范例。这些文件提供了一些示例，可帮助您了解在向 SWF 文件加载图像文件时，如何使用 ActionScript 动态地控制影片剪辑。您可以在硬盘上的 Samples 文件夹中找到范例源文件 gallery_tree.fla 和 gallery_tween.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Galleries。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Galleries。

使用 ActionScript 为加载 MP3 文件创建进度条

下面的示例将几首歌曲加载到 SWF 文件中。一个使用 Drawing API 创建的进度条会显示加载进度。当音乐开始加载和完成加载时，“输出”面板中会显示信息。有关加载 MP3 文件的信息，请参见第 534 页的 “加载 MP3 文件”。。

为加载 MP3 文件创建进度条：

1. 创建一个名为 **loadSound.fla** 的新 Flash 文档。
2. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
var pb_height:Number = 10;
var pb_width:Number = 100;
var pb:MovieClip = this.createEmptyMovieClip("progressBar_mc",
  this.getNextHighestDepth());
pb.createEmptyMovieClip("bar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("vBar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("stroke_mc", pb.getNextHighestDepth());
pb.createTextField("pos_txt", pb.getNextHighestDepth(), 0, pb_height,
  pb_width, 22);

pb._x = 100;
pb._y = 100;

with (pb.bar_mc) {
   beginFill(0x00FF00);
   moveTo(0, 0);
   lineTo(pb_width, 0);
   lineTo(pb_width, pb_height);
   lineTo(0, pb_height);
   lineTo(0, 0);
   endFill();
   _xscale = 0;
}
with (pb.vBar_mc) {
   lineStyle(1, 0x000000);
   moveTo(0, 0);
   lineTo(0, pb_height);
}
with (pb.stroke_mc) {
   lineStyle(3, 0x000000);
   moveTo(0, 0);
   lineTo(pb_width, 0);
   lineTo(pb_width, pb_height);
   lineTo(0, pb_height);
   lineTo(0, 0);
}

var my_interval:Number;
var my_sound:Sound = new Sound();
```

```
my_sound.onLoad = function(success:Boolean) {
   if (success) {
      trace("sound loaded");
   }
};
my_sound.onSoundComplete = function() {
   clearInterval(my_interval);
   trace("Cleared interval");
}
my_sound.loadSound("http://www.helpexamples.com/flash/sound/song2.mp3",
  true);
my_interval = setInterval(updateProgressBar, 100, my_sound);

function updateProgressBar(the_sound:Sound):Void {
   var pos:Number = Math.round(the_sound.position / the_sound.duration *
  100);
   pb.bar_mc._xscale = pos;
   pb.vBar_mc._x = pb.bar_mc._width;
   pb.pos_txt.text = pos + "%";
}
```

3. 选择“控制”>“测试影片”来加载 MP3 文件并观察进度条。

注意	如果您是第二次测试此代码，图像将为已缓存状态，因此进度条会立即走完。若要执行多次测试，请使用不同的图像并从外部源加载它们。如果使用本地源测试应用程序，可能会出现问题，因为本地内容的加载速度太快。

有关使用声音的更多信息，请参见《ActionScript 2.0 语言参考》中的 Sound 类条目 Sound。

您可以找到一个用脚本动画创建进度条动画的范例源文件。可以在您的硬盘上的 Samples 文件夹中找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

您还能在硬盘上的 Samples 文件夹中找到一个加载 MP3 文件的范例源文件 jukebox.fla。此范例演示如何使用数据类型、一般编码规则和几个组件创建一个自动唱片点唱机。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\Components\Jukebox。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/Components/Jukebox。

用 ActionScript 为加载 FLV 文件创建进度条

您可以创建一个进度条来显示 FLV 文件的加载进度。有关在 SWF 文件中加载 FLV 文件的信息，请参见第 544 页的 “预加载 FLV 文件”。有关 FLV 文件和 Flash 的其它信息，请参见第 538 页的 “关于使用 FLV 视频”。

下面的示例使用 Drawing API 来创建进度条。该示例还使用 `bytesLoaded` 和 `bytesTotal` 属性来显示向名为 `my_video` 的视频对象实例中加载 video1.flv 的加载进度。会动态创建一个 `loaded_txt` 文本字段，以显示有关加载进度的信息。

创建显示加载进度的进度条：

1. 创建一个名为 **flvProgress.fla** 的新 FLA 文件。
2. 在 “库”面板 （“窗口” > “库”）中，从 “库”弹出菜单中选择 “新建视频”。
3. 在 “视频属性”对话框中，为视频元件命名并选择 “视频”（受 ActionScript 控制）。
4. 单击 “确定”，创建一个视频对象。
5. 将该视频对象从 “库”面板拖动到舞台上，以创建视频对象实例。
6. 使视频对象在舞台上保持选中状态，在属性检查器 （“窗口” > “属性” > “属性”）中的 “实例名称”文本框中键入 **my_video**。
7. 使视频实例保持选中状态，在宽度文本框中键入 **320**，在高度文本框中键入 **213**。
8. 在时间轴中选择第 1 帧，然后在 “动作”面板中键入下面的代码：

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("http://www.helpexamples.com/flash/video/
  typing_short.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10,
  160, 22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc",
  progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
   beginFill(0xFF0000);
   moveTo(0, 0);
   lineTo(100, 0);
   lineTo(100, 10);
   lineTo(0, 10);
   lineTo(0, 0);
   endFill();
   _xscale = 0;
}
progressBar_mc.createEmptyMovieClip("stroke_mc",
  progressBar_mc.getNextHighestDepth());
```

```
with (progressBar_mc.stroke_mc) {
   lineStyle(0, 0x000000);
   moveTo(0, 0);
   lineTo(100, 0);
   lineTo(100, 10);
   lineTo(0, 10);
   lineTo(0, 0);
}

var loaded_interval:Number = setInterval(checkBytesLoaded, 500,
  stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
   var pctLoaded:Number = Math.round(my_ns.bytesLoaded / my_ns.bytesTotal
  * 100);
   loaded_txt.text = Math.round(my_ns.bytesLoaded / 1000) + " of " +
  Math.round(my_ns.bytesTotal / 1000) + " KB loaded (" + pctLoaded +
  "%)";
   progressBar_mc.bar_mc._xscale = pctLoaded;
   if (pctLoaded>=100) {
      clearInterval(loaded_interval);
   }
}
```

9. 选择“控制”>“测试影片”，对代码进行测试。

   将加载视频，并通过动画栏和变化的文本值指示加载进度。如果这些元素覆盖了您的视频，可在舞台上移动视频对象。通过修改前面代码片断中的 `beginFill` 和 `lineStyle`，可以自定义进度条的颜色。

提示	如果进度条立刻走完，则表明视频被缓存到了硬盘上（可能是早已测试过此示例，也可能在其它过程中早已加载了它）。如果发生这种情况，请将 FLV 文件上载到服务器，然后再加载。

您可以找到一个用脚本动画创建进度条动画的范例源文件。可以在您的硬盘上的 Samples 文件夹中找到 tweenProgress.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Tween ProgressBar。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Tween ProgressBar。

第 16 章

16 使用外部数据

在 Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 中，可以使用 ActionScript 从外部源向 SWF 文件中加载数据。还可以将用户或服务器提供的数据从 SWF 文件发送至应用程序服务器（例如 Macromedia ColdFusion 或 Macromedia Jrun）或其它类型的服务器端脚本（例如 PHP 或 Perl）。Macromedia Flash Player 可以通过 HTTP 或 HTTPS 发送和加载数据，也可以从本地文本文件加载数据。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。Flash Player 8 中的一个新增功能，可以从用户的计算机向服务器上载文件，还可以从服务器向用户计算机下载文件。

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）或同一台计算机或网页上 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相同域（例如 www.macromedia.com）中的数据。（有关更多信息，请参见第 618 页的 “关于域、跨域安全性和 SWF 文件”。）

有关使用外部数据的更多信息，请参见以下主题：

发送和加载变量

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 ActionScript 函数和方法，可以与服务器端脚本互相收发信息，也可从文本文件和 XML 文件接收信息。

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，您可以创建消息板、用户的个人配置文件，以及能够跟踪用户购物行为的购物车。

许多 ActionScript 函数和方法可用于将信息传入 SWF 文件以及从 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()`。

有关更多信息，请参见以下主题：

- 第 567 页的 “检查已加载的数据”
- 第 568 页的 “创建进度条显示数据加载进度”

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 文件中使用加载的数据之前，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，因为待处理的数据在加载之前并不存在于该文件中。在下面的脚本中，在您确认变量 lastSiteVisited 已从文件 myData.txt 中加载之前，您不能使用该变量。在 myData.txt 文件中，具有与以下示例相似的文本：

```
lastSiteVisited=www.macromedia.com
```

但如果使用了下面的代码，您将不能跟踪正在加载的数据：

```
loadVariables("myData.txt", 0);
trace(lastSiteVisited); // undefined
```

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 函数或 loadMovie 函数，则可以向影片剪辑目标加载信息，并使用 onData 处理函数来执行脚本。如果使用 loadVariables 函数加载数据，则在最后一个变量加载后将执行 onData 处理函数。如果使用 loadMovie 函数加载数据，则当每次 SWF 文件的片段进入 Flash Player 时都会执行 onData 处理函数。

例如，下面的 ActionScript 从 myData.txt 文件中向影片剪辑 loadTarget_mc 加载变量。分配给 loadTarget_mc 实例的 onData() 处理函数将使用变量 lastSiteVisited，该变量从文件 myData.txt 中加载。仅当所有的变量（包括 lastSiteVisited）都已加载，才会进行下面的输出操作：

```
this.createEmptyMovieClip("loadTarget_mc", this.getNextHighestDepth());
this.loadTarget_mc.onData = function() {
  trace("Data Loaded");
  trace(this.lastSiteVisited);
};
loadVariables("myData.txt", this.loadTarget_mc);
```

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。此处理函数是 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，这些处理函数将被自动调用。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 583 页的“使用 XML 类”和第 588 页的“使用 XMLSocket 类”。有关使用 LoadVars 发送和加载数据（可在接收后进行处理的数据）的更多信息，请参见第 571 页的 “使用 LoadVars 类”。

创建进度条显示数据加载进度

下面的练习使用 Drawing API 动态创建一个简单的预加载器，并显示 XML 文档的加载进度。

提示 如果远程 XML 文件加载过快，无法看到预加载效果，则请尝试将一个较大的 XML 文件上载到 Internet 并加载该文件。

使用 Drawing API 创建进度条：

1. 创建一个新的 Flash 文档，并将该文档保存为 **drawapi.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var barWidth:Number = 200;
var barHeight:Number = 6;

this.createEmptyMovieClip("pBar_mc", 9999);
var bar:MovieClip = pBar_mc.createEmptyMovieClip("bar_mc", 10);
bar.beginFill(0xFF0000, 100);
bar.moveTo(0, 0);
bar.lineTo(barWidth, 0);
bar.lineTo(barWidth, barHeight);
bar.lineTo(0, barHeight);
bar.lineTo(0, 0);
bar.endFill();
bar._xscale = 0;

var stroke:MovieClip = pBar_mc.createEmptyMovieClip("stroke_mc", 20);
stroke.lineStyle(0, 0x000000);
stroke.moveTo(0, 0);
stroke.lineTo(barWidth, 0);
stroke.lineTo(barWidth, barHeight);
stroke.lineTo(0, barHeight);
stroke.lineTo(0, 0);

pBar_mc.createTextField("label_txt", 30, 0, barHeight, 100, 21);
pBar_mc.label_txt.autoSize = "left";
pBar_mc.label_txt.selectable = false;

pBar_mc._x = (Stage.width - pBar_mc._width) / 2;
pBar_mc._y = (Stage.height - pBar_mc._height) / 2;

var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
  pBar_mc.onEnterFrame = undefined;
  if (success) {
    trace("");
  } else {
    trace("Unable to load XML");
```

```
    }
};
my_xml.load("http://www.helpexamples.com/flash/xml/ds.xml");

pBar_mc.onEnterFrame = function() {
    var pctLoaded:Number = Math.floor(my_xml.getBytesLoaded() /
    my_xml.getBytesTotal() * 100);
    if (!isNaN(pctLoaded)) {
        pBar_mc.bar_mc._xscale = pctLoaded;
        pBar_mc.label_txt.text = pctLoaded + "% loaded";
        if (pctLoaded >= 100) {
            pBar_mc.onEnterFrame = undefined;
        }
    }
};
```

前面的代码分为七个部分。第一部分定义当在舞台上绘制进度条时所用的宽度和高度。在下面的一部分中，该进度条将在舞台上居中。下一部分代码创建两个影片剪辑 pBar_mc 和 bar_mc。bar_mc 影片剪辑嵌套在 pBar_mc 中，会在舞台上绘制一个红色矩形。从远程网站加载外部 XML 文件时，bar_mc 实例将修改其 _xscale 属性。

然后，第二个影片剪辑 stroke_mc 嵌套在 pBar_mc 影片剪辑中。stroke_mc 影片剪辑将在舞台上绘制一个轮廓，此轮廓与在第一部分中定义的 barHeight 和 barWidth 变量指定的尺寸相匹配。第四部分代码在 pBar_mc 影片剪辑中创建一个文本字段，用于显示已加载的 XML 文件的百分比，它与 ProgressBar 组件上的标签类似。然后，影片剪辑 pBar_mc（包括嵌套的 bar_mc、stroke_mc 和 label_txt 实例）将在舞台上居中。

第六部分代码定义了一个新的 XML 对象实例，用于加载外部 XML 文件。定义了一个 onLoad 事件处理函数，将在“输出”面板输出一条消息。onLoad 事件处理函数还将删除 pBar_mc 影片剪辑的 onEnterFrame 事件处理函数（此函数在下一部分中定义）。最后一部分代码为 pBar_mc 影片剪辑定义一个 onEnterFrame 事件处理函数。此事件处理函数将监视已加载的外部 XML 文件的量，并修改 bar_mc 影片剪辑的 _xscale 属性。onEnterFrame 事件处理函数将首先计算文件已完成的下载的百分比。只要已加载的文件百分比是有效数字，就将设置 bar_mc 的 _xscale 属性和 pBar_mc 中的文本字段，以显示已加载文件的百分比。如果文件已完成加载（加载百分比达到 100%），则 onEnterFrame 事件处理函数将被删除，以不再监视下载进度。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

加载外部 XML 文件时，嵌套的 bar_mc 影片剪辑将调整大小，以显示 XML 的下载进度。一旦 XML 文件加载完成后，onEnterFrame 事件处理函数将被删除，以使其不再继续计算下载进度。根据下载完成的速度，您应该能看到进度条慢慢变大，直到 bar_mc 与 stroke_mc 影片剪辑宽度相同为止。如果下载速度过快，进度条可能迅速从 0% 达到 100%，使效果很难看到；在这种情况下，需要尝试下载较大的 XML 文件。

使用 HTTP 连接到服务器端脚本

loadVariables 函数、loadVariablesNum 函数、getURL 函数、loadMovie 函数、loadMovieNum 函数以及 loadVariables（MovieClip.loadVariables 方法）、loadMovie（MovieClip.loadMovie 方法）和 getURL（MovieClip.getURL 方法）方法均可以使用 HTTP 或 HTTPS 协议与服务器端脚本通讯。这些函数和方法将所有变量从时间轴发送到附加函数的位置。当用作 MovieClip 对象的方法时，`loadVariables()`、`getURL()` 和 `loadMovie()` 会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()` 函数将所有信息返回到浏览器窗口，而不是 Flash Player。
- `loadVariables()` 方法将变量加载到 Flash Player 中指定的时间轴或级别。
- `loadMovie()` 方法将 SWF 文件加载到 Flash Player 中指定的级别或影片剪辑中。

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》中的第 1 章“关于多个时间轴和层”。
- *`Variables`* 设置发送变量的 HTTP 方法，可以是 `GET`（将变量追加到 URL 的末尾）或 `POST`（在单独的 HTTP 标头中发送变量）。当此参数被省略时，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。该函数调用可能如下面的示例所示：

```
this.createEmptyMovieClip("highscore_mc", 10);
loadVariables("http://www.helpexamples.com/flash/highscore.php",
  highscore_mc, "GET");
```

此示例会将变量从名为 high_score.cfm 的 ColdFusion 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（CFM 和 CGI 脚本使用的标准格式）。您在 `loadVariables()` 的 *`URL`* 参数中指定的文件必须以此格式写出变量和值对，Flash 才能进行读取。此文件可以指定任意数量的变量；变量和值对必须使用与符号 (&) 分隔，而值中的词必须用加号 (+) 分隔。例如，下面的语句定义了多个变量：

```
highScore1=54000&playerName1=RGoulet&highScore2=53455&playerName2=
  WNewton&highScore3=42885&playerName3=TJones
```

注意	您可能需要对某些字符进行 URL 编码，例如加号 (+) 或与符号 (&) 字符。有关更多信息，请参见 www.macromedia.com/go/tn_14143。

有关更多信息，请参见以下主题：第 571 页的 “使用 LoadVars 类”。另外，请参见《ActionScript 2.0 语言参考》中的 loadVariables 函数、getURL 函数、loadMovie 函数和 LoadVars 条目。

使用 LoadVars 类

如果要发布到 Flash Player 6 或更高版本并且需要的灵活性比 `loadVariables()` 提供的灵活性更高，可以使用 LoadVars 类（而不是在 SWF 文件和服务器之间传输变量）。

LoadVars 类在 Flash Player 6 中引入，为与 Web 服务器交换 CGI 数据的常见任务提供一个更清晰、更面向对象的接口。 LoadVars 类的优点包括以下几项：

- 您无需创建容器影片剪辑来保存数据，或在现有影片剪辑中塞满特定于客户端/服务器通讯的变量。
- 类的接口与 XML 对象类似（XML 对象在 ActionScript 中提供某些一致性）。它使用 `load()`、`send()` 和 `sendAndLoad()` 方法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 文档对象模型 (DOM) 树的属性。
- 类接口比旧的 loadVariables 接口更直观，其方法名为 `load`、`send` 和 `sendAndLoad`。
- 可以使用 `getBytesLoaded` 和 `getBytesTotal` 方法获取有关通讯的其它信息。
- 可以获取与数据下载有关的进度信息（尽管不能在数据完全下载前访问这些数据）。
- 回调接口通过 ActionScript 方法 (`onLoad`) 实现，而不是 loadVariables 需要的 `onClipEvent`（数据）方法（此方法已经废弃，不赞成使用）。
- 具有错误通知。
- 可以添加自定义的 HTTP 请求标头。

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。

下面的步骤显示如何使用 ColdFusion 和 LoadVars 类来发送 SWF 文件中的电子邮件。

注意	对于本示例， Web 服务器上必须安装了 ColdFusion。

用 LoadVars 对象加载数据：

1. 在 Macromedia Dreamweaver 或您喜爱的文本编辑器中创建一个 CFM 文件。在文件中添加下面的文本：

```
<cfif StructKeyExists(Form, "emailTo")>
<cfmail to="#Form.emailTo#" from="#Form.emailFrom#"
  subject="#Form.emailSubject#">#Form.emailBody#</cfmail>
&result=true
<cfelse>
&result=false
</cfif>
```

2. 将该文件保存为 **email.cfm**，并上传到您的网站。
3. 在 Flash 中，创建一个新文档。
4. 在舞台上创建四个输入文本字段，分别为其指定实例名称：**emailFrom_txt**、**emailTo_txt**、**emailSubject_txt** 和 **emailBody_txt**。
5. 在舞台上创建一个动态文本字段，实例名称为 **debug_txt**。
6. 创建一个按钮元件，将一个实例拖动到舞台上，并给定其实例名称 **submit_btn**。
7. 在时间轴中选择第 1 帧，如果“动作”面板尚未打开，则将其打开（“窗口”>“动作”）。
8. 在 “动作”面板中输入下面的代码：

```
this.submit_btn.onRelease = function() {
  var emailResponse:LoadVars = new LoadVars();
  emailResponse.onLoad = function(success:Boolean) {
  if (success) {
       debug_txt.text = this.result;
  } else {
       debug_txt.text = "error downloading content";
  }
  };
  var email:LoadVars = new LoadVars();
  email.emailFrom = emailFrom_txt.text;
  email.emailTo = emailTo_txt.text;
  email.emailSubject = emailSubject_txt.text;
  email.emailBody = emailBody_txt.text;
  email.sendAndLoad("http://www.yoursite.com/email.cfm", emailResponse,
  "POST");
};
```

此 ActionScript 创建一个新的 LoadVars 对象实例，将文本字段中的值复制到该实例中，然后将数据发送到服务器。CFM 文件发送电子邮件并将一个变量（`true` 或 `false`）返回到名为 `result` 的 SWF 文件 （出现在 `debug_txt` 文本字段中）。

记住将 URL www.yoursite.com 更改为您自己的域。

9. 将文档保存为 **sendEmail.fla**，然后通过选择 “文件” > “发布” 来发布此文件。

10. 将 sendEmail.swf 上传到包含 email.cfm（您在步骤 2 中保存和上传的 ColdFusion 文件）的同一个目录。

11. 在浏览器中查看和测试该 SWF 文件。

有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 LoadVars 条目。

Flash Player 8 为 LoadVars 类、XML 类和 MovieClipLoader 类引入了 `onHTTPStatus` 事件处理函数，允许用户从 HTTP 请求中访问状态代码。这使开发人员可以确定特定加载操作失败的原因，而不仅仅确定加载操作已失败。

下面的代码示例演示如何使用 LoadVars 类的 `onHTTPStatus` 事件处理函数检查是否从服务器成功下载了文本文件，并检查从 HTTP 请求所返回的状态代码。

使用 LoadVarso 对象检查 HTTP 状态：

1. 创建一个新的 Flash 文档，并保存为 **loadvars.fla**。

2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
this.createTextField("params_txt", 10, 10, 10, 100, 21);
params_txt.autoSize = "left";

var my_lv:LoadVars = new LoadVars();
my_lv.onHTTPStatus = function(httpStatus:Number) {
  trace("HTTP status is: " + httpStatus);
};
my_lv.onLoad = function(success:Boolean) {
  if (success) {
    trace("text file successfully loaded");
    params_txt.text = my_lv.dayNames;
  } else {
    params_txt.text = "unable to load text file";
  }
};
my_lv.load("http://www.helpexamples.com/flash/404.txt");
/* output:
  Error opening URL "http://www.helpexamples.com/flash/404.txt"
  HTTP status is: 404
*/
```

前面的代码在舞台上创建一个新的文本字段，并启用文本字段自动调整大小功能。然后，将创建一个 LoadVars 对象和两个事件处理函数：onHTTPStatus 和 onLoad。onHTTPStatus 事件处理函数是 Flash Player 8 中新增的，将在 LoadVars.load() 或 LoadVars.sendAndLoad() 操作完成时被调用。传递给 onHTTPStatus 事件处理函数的值（前面代码中的 httpStatus）包含用于当前加载操作的 HTTP 状态代码定义。如果 SWF 文件能够成功加载文本文件，则 httpStatus 值设置为 200（“确定”的 HTTP 状态代码）。如果服务器上不存在该文件，则 httpStatus 值设置为 404（“找不到”的 HTTP 状态代码）。在文件加载完成后，将调用第二个事件处理函数 LoadVars.onLoad()。如果文件成功加载，则 success 参数的值设置为 true，否则 success 参数将设置为 false。最后，使用 LoadVars.load() 方法加载外部文件。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

Flash 向“输出”面板显示一条错误信息，指出由于服务器中不存在该图像，因此无法加载。由于在服务器上找不到文件，onHTTPStatus 事件处理函数将输出状态码 404，并且 onLoad 事件处理函数会将 params_txt 文本字段的文本属性设置为“unable to load text file”。

小心	如果 Web 服务器不向 Flash Player 返回状态代码，则将为 onHTTPStatus 事件处理函数返回数字 0。

关于文件上载和下载

FileReference 类允许您在客户端和服务器之间添加上载和下载文件的功能。您的用户可以在计算机和服务器之间上载或下载文件。将在一个对话框中（例如，Windows 操作系统中的“打开”对话框）提示用户选择要上载的文件或用于下载的位置。

您使用 ActionScript 创建的每个 FileReference 对象都引用用户硬盘上的一个文件。该对象的属性包含有关文件大小、类型、名称、创建日期和修改日期的信息。在 Macintosh 上，还有一个有关文件的创建者类型的属性。

可以通过两种方式创建 FileReference 类的实例。可以使用以下的 new 运算符：

```
import flash.net.FileReference;
var myFileReference:FileReference = new FileReference();
```

或者，可以调用 FileReferenceList.browse() 方法，该方法将在用户的系统中打开一个对话框，提示用户选择要上载的文件，如果用户成功选择了一个或多个文件，则将创建一个 FileReference 对象组成的数组。每个 FileReference 对象表示用户在对话框中选择的一个文件。在调用了 FileReference.browse() 方法或 FileReferenceList.browse() 方法，并且用户已从文件选取器中选择一个文件之前，或者已将 FileReference.download() 方法用于从文件选取器中选择文件之前， FileReference 对象的 FileReference 属性 （例如 name、size 或 modificationDate）中不会包含任何数据。

注意	FileReference.browse() 允许用户选择一个文件。 FileReferenceList.browse() 允许用户选择多个文件。

成功调用 browse() 方法之后，调用 FileReference.upload() 以便一次上载一个文件。

还可以为您的 Flash 应用程序添加下载功能。FileReference.download() 方法将提示最终用户输入硬盘上的一个位置，以保存来自服务器的文件。此方法还将启动从远程 URL 进行的下载。使用 download() 方法时，当调度了 onSelect 事件时将只能访问 FileReference.name 属性。在调度 onComplete 事件之前，其余属性均不可访问。

注意	当最终用户的计算机中出现对话框时，该对话框中显示的默认位置是最近浏览的文件夹 （如果该位置可以确定）或桌面 （如果最近浏览的文件夹无法确定）。 FileReference 和 FileReferenceList API 不允许您设置默认的文件位置。

有关 FileReference API 的功能和安全的信息，请参见第 576 页的“关于 FileReference API 功能和安全”。有关使用 FileReference API 的应用程序示例，请参见第 577 页的 “为应用程序添加文件上载功能” 。可在硬盘上的 Samples 文件夹中找到此示例的范例源文件 FileUpload.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FileUpload。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FileUpload。

有关 FileReference API 的每个方法、属性和事件的信息，请参见 《ActionScript 2.0 语言参考》中的 FileReference (flash.net.FileReference) 和 FileReferenceList (flash.net.FileReferenceList)。

关于 FileReference API 功能和安全

Flash Player 和 FileReference API （请参见第 574 页的 “关于文件上载和下载”）支持最高达 100 MB 的文件上载和下载。FileReference API 不 允许启动文件传输的 Flash 应用程序执行以下操作：

- 访问上载或下载的文件
- 访问计算机中的文件路径

如果服务器要求身份验证，唯一可能成功的操作是使用 Flash Player 浏览器插件执行文件下载。在要求身份验证的服务器上，所有 Flash 播放器上进行的上载或者通过独立或外部 Flash 播放器进行的下载都会失败。使用 FileReference 事件侦听器确定操作是否成功完成，或用于进行错误处理。

文件上载和下载均被限制到 SWF 文件的域中，包括由跨域策略文件指定的任何域。如果启动上载或下载的 SWF 文件不是来服务器所做的同一个域，则需要在服务器上放置一个策略文件。有关跨域策略和安全的更多信息，请参见第 618 页的 “关于域、跨域安全性和 SWF 文件”。

在调用 `FileReference.browse()`、`FileReferenceList.browse()` 时或执行 `FileReference.download()` 时，在下列平台上会暂停 SWF 回放：Mac OS X Flash Player 浏览器插件、 Macintosh 外部 Flash Player 以及 Mac OS X 10.1 和较早版本上的 Macintosh 独立播放器。SWF 文件会继续在所有 Windows 播放器以及 Mac OS X 10.2 和更高版本上的 Macintosh 独立 Flash Player 中运行。

警告	在允许用户向服务器上载文件时，在将文件保存到硬盘之前应始终仔细检查文件类型。例如，您可能不希望允许用户上载可用于删除服务器上的文件夹或文件的服务器端脚本。如果只希望允许用户上载图像文件，则请确保上载文件的服务器端脚本对正在上载的文件进行检查，确保它是有效的图像。

有关使用 FileReference API 的应用程序的示例，请参见第 577 页的 “为应用程序添加文件上载功能”。

为应用程序添加文件上载功能

下面的过程演示如何构建可用于向服务器上载图像文件的应用程序。该应用程序允许用户在硬盘上选择要上载的图像，然后将其发送给服务器。用户上载的图像然后会出现在他们用于上载图像的 SWF 文件中。

下面的示例将构建 Flash 应用程序，用以展示服务器端代码的细节。请记住，图像文件在大小上受到限制：只能上载 200K 或更小的图像。

使用 FileReference API 构建 FLA 应用程序：

1. 创建一个新的 Flash 文档，并将它保存为 **fileref.fla**。
2. 打开 “组件”面板，然后将一个 ScrollPane 组件拖动到舞台上，并为其给定实例名称 **imagePane**。（在后面的步骤中，将使用 ActionScript 调整此滚动窗格实例的大小和位置。）
3. 将一个 Button 组件拖到舞台上，然后为其给定实例名称 **uploadBtn**。
4. 将两个 Label 组件拖动到舞台上，并为其给定实例名称 **imageLbl** 和 **statusLbl**。
5. 将 ComboBox 组件拖到舞台上，然后为其给定实例名称 **imagesCb**。
6. 将 TextArea 组件拖到舞台上，然后为其给定实例名称 **statusArea**。
7. 在舞台上创建一个新的影片剪辑元件，然后打开该元件用于编辑 （双击实例以在元件编辑模式中将其打开）。
8. 在影片剪辑字段中创建一个新的静态文本字段，然后添加以下文本：

   您尝试下载的文件不在服务器上。

   在最终的应用程序中，可能因下面的原因之一出现此警告：

   - 当上载其它图像时，从服务器的队列中删除了该图像。
   - 由于文件大小超过 200K，因此服务器未复制该图像。
   - 文件类型不是有效的 JPEG、GIF 或 PNG 文件。

注意	文本字段的宽度应小于滚动窗格实例的宽度 （400 像素）；否则，用户必须水平滚动才能看到错误信息。

9. 在 “库”中右键单击该元件，并从上下文菜单中选择 “链接”。
10. 选中 “为 ActionScript 导出”和 “在第一帧导出”复选框，并在 “标识符”文本框中键入 **Message**。单击 “确定”。

11. 将下面的 ActionScript 添加到时间轴中的第 1 帧：

注释	代码注释包括有关功能的详细信息。此示例后提供了代码概述。

```
import flash.net.FileReference;

imagePane.setSize(400, 350);
imagePane.move(75, 25);
uploadBtn.move(75, 390);
uploadBtn.label = "Upload Image";
imageLbl.move(75, 430);
imageLbl.text = "Select Image";
statusLbl.move(210, 390);
statusLbl.text = "Status";
imagesCb.move(75, 450);
statusArea.setSize(250, 100);
statusArea.move(210, 410);

/* 侦听器对象侦听 FileReference 事件。 */
var listener:Object = new Object();

/* 当用户选择某一文件时，将调用 onSelect() 方法，并将一个引用传递给 FileReference
   对象。 */
listener.onSelect = function(selectedFile:FileReference):Void {
  /* 更新文本区域，以通知用户 Flash 正在尝试上载图像。 */
  statusArea.text += "Attempting to upload " + selectedFile.name + "\n";
  /* 将文件上载到服务器上的 PHP 脚本。 */
  selectedFile.upload("http://www.helpexamples.com/flash/file_io/
  uploadFile.php");
};

/* 当文件开始上载时，调用 onOpen() 方法，以通知用户该文件将开始上载。 */
listener.onOpen = function(selectedFile:FileReference):Void {
  statusArea.text += "Opening " + selectedFile.name + "\n";
};

/* 当文件上载后，调用 onComplete() 方法。 */
listener.onComplete = function(selectedFile:FileReference):Void {
  /* 通知用户 Flash 已开始下载图像。 */
  statusArea.text += "Downloading " + selectedFile.name + " to player\n";
  /* 将图像添加到组合框组件。 */
  imagesCb.addItem(selectedFile.name);
  /* 将组合框的选定索引设置为最近添加的图像的索引。 */
  imagesCb.selectedIndex = imagesCb.length - 1;
  /* 调用自定义 downloadImage() 函数。 */
  downloadImage();
};

var imageFile:FileReference = new FileReference();
```

```
imageFile.addListener(listener);

imagePane.addEventListener("complete", imageDownloaded);
imagesCb.addEventListener("change", downloadImage);
uploadBtn.addEventListener("click", uploadImage);

/* 如果该图像不下载，则事件对象的总属性将等于 ñ1。在此情况下，将向用户显示一条消息。 */
function imageDownloaded(event:Object):Void {
  if (event.total == -1) {
    imagePane.contentPath = "Message";
  }
}

/* 当用户从组合框中选择图像时，或直接从 listener.onComplete() 方法中直接调用
  downloadImage() 函数时，downloadImage() 函数将设置滚动窗格的 contentPath，以便
  开始向播放器下载图像。 */
function downloadImage(event:Object):Void {
  imagePane.contentPath = "http://www.helpexamples.com/flash/file_io/
  images/" + imagesCb.value;
}

/* 当用户单击按钮时，Flash 将调用 uploadImage() 函数，并打开一个文件浏览器对话框。 */
function uploadImage(event:Object):Void {
  imageFile.browse([{description: "Image Files", extension:
  "*.jpg;*.gif;*.png"}]);
}
```

此 ActionScript 代码首先导入 FileReference 类，然后对舞台上每个组件进行初始化、定位和大小调整。接下来，定义一个侦听器对象，并定义三个事件处理函数：`onSelect`、`onOpen` 和 `onComplete`。然后，侦听器对象将被添加到名为 `imageFile` 的新 FileReference 对象中。然后，向 `imagePane` 滚动窗格实例、`imagesCb` 组合框实例和 `uploadBtn` 按钮实例中添加事件侦听器。在本部分代码后面的代码中，将定义每个事件侦听器函数。

第一个函数 `imageDownloaded()` 将进行检查，以确定下载图像的总字节数是否为 ñ1，如果是这样，它将把滚动窗格实例的 `contentPath` 设置为具有 Message 链接标识符的影片剪辑（您在前面的步骤中创建）。第二个函数 `downloadImage()` 会尝试将最近上载的图像下载到滚动窗格实例中。当图像下载完成时，将触发前面定义的 `imageDownloaded()` 函数，以检查是否成功下载了图像。最后一个函数 `uploadImage()` 将打开一个文件浏览器对话框，它将筛选所有 JPEG、GIF 和 PNG 图像。

12. 保存对文档的更改。

13. 选择“文件”>“发布设置”，然后选择“格式”选项卡，并确保 Flash 和 HTML 都已选中。

14. （可选）在 “发布设置”对话框中，选择 “Flash”选项卡，然后从 “本地回放安全性”弹出菜单中选择 “只访问网络”。

    如果您完成了此步骤，并且在本地浏览器中测试文档，则将不会遇到安全限制。

15. 在 “发布设置”对话框中，单击 “发布”创建 HTML 和 SWF 文件。

    完成时，请继续下面的过程，在此您将为 SWF 文件创建容器。

可在硬盘上 Samples 文件夹中找到此示例的范例源文件 FileUpload.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FileUpload。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FileUpload。

下面的过程要求在 Web 服务器中安装 PHP，并且要求您对名为 images 和 temporary 的子文件夹具有写入权限。您需要先完成前面的过程，或使用前面提及的文件中提供的已完成 SWF 文件。

为图像上载应用程序创建服务器端脚本：

1. 使用 Dreamweaver 或记事本等文本编辑器创建新的 PHP 文档。
2. 将以下 PHP 代码添加到文档中。（该脚本后提供了代码概述。）

```
<?php

$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // ±£Š÷²žŒÒý˜÷-ÞÓ¹‡”- 10 ³^Œfºž
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE) {
  move_uploaded_file($_FILES['Filedata']['tmp_name'], "./temporary/
  ".$_FILES['Filedata']['name']);
  $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
  if ($type == 1 || $type == 2 || $type == 3) {
    rename("./temporary/".$_FILES['Filedata']['name'], "./images/
  ".$_FILES['Filedata']['name']);
  } else {
    unlink("./temporary/".$_FILES['Filedata']['name']);
  }
}
$directory = opendir('./images/');
$files = array();
while ($file = readdir($directory)) {
  array_push($files, array('./images/'.$file, filectime('./images/
  '.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT) {
  $files_to_delete = array_splice($files, 0, count($files) -
  $MAXIMUM_FILE_COUNT);
```

```
  for ($i = 0; $i < count($files_to_delete); $i++) {
    unlink($files_to_delete[$i][0]);
  }
}
print_r($files);
closedir($directory);

function sorter($a, $b) {
  if ($a[1] == $b[1]) {
    return 0;
  } else {
    return ($a[1] < $b[1]) ? -1 : 1;
  }
}
?>
```

此 PHP 代码首先定义两个常量变量：`$MAXIMUM_FILESIZE` 和 `$MAXIMUM_FILE_COUNT`。这些变量描述上载到服务器中的图像的最大大小 (200KB)，以及在图像文件夹中可以保留的最近上载文件的数量 (10)。如果当前上载的图像的文件大小小于或等于 `$MAXIMUM_FILESIZE` 的值，则图像将被移动到临时文件夹中。

然后，将检查上载文件的文件类型，以确保图像为 JPEG、GIF 或 PNG。如果图像是兼容的图像类型，则将该图像从临时文件夹复制到图像文件夹中。如果上载的文件不是允许的图像类型，则将从文件系统中删除它。

然后，将创建一个图像文件夹的目录列表，并使用 while 循环循环访问。 images 文件夹中的每个文件都将添加到数组中，然后进行排序。如果图像文件夹中的当前文件数大于 `$MAXIMUM_FILE_COUNT` 的值，则将删除文件，直到剩下的图像数为 `$MAXIMUM_FILE_COUNT` 为止。这可以防止图像文件夹增大到超出可管理大小，因为文件夹中同时只能有 10 个图像，并且每个图像只能为 200KB 或更小 （或任何时间的图像总共占用的空间大约为 2 MB）。

3. 保存对 PHP 文档的更改。
4. 将 SWF、HTML 和 PHP 文件上载到 Web 服务器。
5. 在 Web 浏览器中查看远程 HTML 文档，并单击 SWF 文件中的 “上载图像”按钮。
6. 在硬盘中查找图像文件并从对话框中选择 “打开”。

   SWF 文件将图像文件上载到远程 PHP 文档，并在滚动窗格（会根据需要添加滚动条）中显示该文件。如果您要查看前面上载的图像，可以在舞台上的组合框实例中选择文件名。如果用户尝试上载不允许的图像类型（只允许 JPEG、GIF 或 PNG 图像）或文件大小过大（超过 200 KB）的图像，Flash 将从库的“消息”影片剪辑中显示错误信息。

可在硬盘上的 Samples 文件夹中找到此示例的范例源文件 FileUpload.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\FileUpload。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/FileUpload。

有关本地文件安全性的更多信息，请参见第 605 页的“关于本地文件安全性和 Flash Player”。

有关编写 PHP 的更多信息，请转到 www.php.net/。

关于 XML

可扩展标记语言 (XML) 正逐渐成为 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，从而构建复杂的应用程序，例如聊天系统或者代理系统。

在 XML 中，和 HTML 一样，使用标签来指定（或者说标记）文本的正文。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 `<b>` 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 把信息的结构与其显示方式分开，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（指示 XML 元素的 1，或者是指示文本节点的 3），并且元素也可以有多个属性。嵌套在节点中的节点称作子节点。节点的这种分层树结构称为 XML DOM，它与 Web 浏览器中元素的结构 JavaScript DOM 很类似。

在下面的示例中，`<portfolio>` 是父级节点；它不具有属性，但包含子节点 `<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>
```

有关更多信息，请参见以下主题：

- 第 583 页的“使用 XML 类”
- 第 588 页的“使用 XMLSocket 类”

有关 XML 的更多信息，请参见 www.w3.org/XML。

硬盘中有几个范例文件，这些文件可在运行时将 XML 加载到 SWF 文件中。一个范例演示了如何通过加载、分析和操作 XML 数据来创建 Web 日志跟踪器。可在硬盘上的 Samples 文件夹中找到范例源文件 xml_blogTracker.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_BlogTracker。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_BlogTracker。

第二个范例演示了如何使用 XML 和嵌套数组来选择不同语言的字符串，以填充文本字段。可在硬盘上的 Samples 文件夹中找到范例源文件 xml_languagePicker.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_LanguagePicker。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_LanguagePicker。

第三个范例演示了如何使用 XML 数据创建动态菜单。该范例调用 ActionScript `XmlMenu()` 构造函数并向其传递两个参数：到 XML 菜单文件的路径以及对当前时间轴的引用。该功能的其余部分均位于自定义类文件 XmlMenu.as 中。

可在硬盘上的 Samples 文件夹中找到范例源文件 xmlmenu.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

使用 XML 类

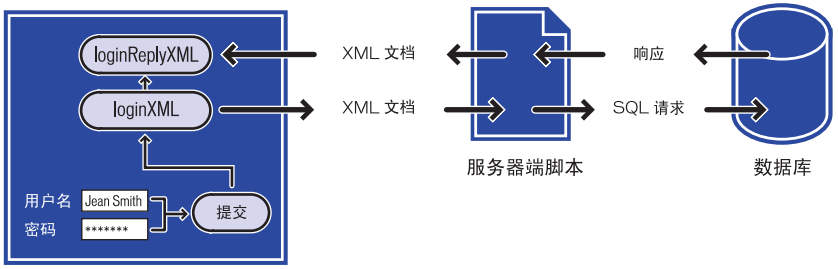
ActionScript XML 类的方法（例如 `appendChild()`、`removeNode()` 和 `insertBefore()`）使您可以在 Flash 中组织要发送到服务器的 XML 数据，并且它们还可以操作和解释下载的 XML 数据。

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 ActionScript XML 对象中。
- `send()` 方法将 XML 对象编码为 XML 文档，并使用 `POST` 方法将其发送到指定的 URL。如果指定了浏览器窗口，该窗口将显示返回的数据。
- `sendAndLoad()` 方法向 URL 发送 XML 对象。返回的任何信息都放在一个 ActionScript XML 对象中。

例如，您可创建一个代理系统，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 格式读写数据。可用 ActionScript 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 ActionScript 将服务器返回的 XML 文档加载到 XML 对象中，以在 SWF 文件中使用该文档。



SWF 文件、服务器端脚本和数据库之间的数据流以及数据转换

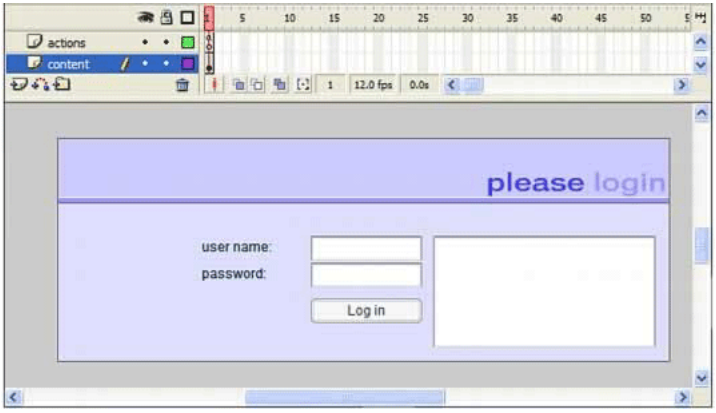
代理系统的密码验证需要两个脚本：一个在第 1 帧中定义的函数，以及一个在文档中创建 XML 对象并发送该对象的脚本。

当用户用变量 `username` 和 `password` 在 SWF 文件中的文本字段中输入信息时，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。当用户单击按钮登录时，`loginXML` 对象先被转换成 XML 字符串，然后被发送到服务器。

下面的 ActionScript 放置在时间轴上，用于将 XML 格式的数据发送到服务器。为了理解此脚本，请阅读注释行（用字符 `//` 表示）：

```
// 忽略 XML 空白
XML.prototype.ignoreWhite = true;
// 构建一个 XML 对象来保存服务器的应答
var loginReplyXML:XML = new XML();
// 此函数在接收到来自服务器的 XML 包时触发。
loginReplyXML.onLoad = function(success:Boolean) {
  if (success) {
    // （可选）为状态 / 调试创建两个文本字段
    // status_txt.text = this.firstChild.attributes.status;
    // debug_txt.text = this.firstChild;
    switch (this.firstChild.attributes.STATUS) {
    case 'OK' :
      _global.session = this.firstChild.attributes.SESSION;
      trace(_global.session);
      gotoAndStop("welcome");
      break;
    case 'FAILURE' :
      gotoAndStop("loginfailure");
      break;
```

```
    default :
      // 这种情况不应该发生
      trace("Unexpected value received for STATUS.");
    }
  } else {
    trace("an error occurred.");
  }
};
// 此函数在单击 login_btn 时触发
login_btn.onRelease = function() {
  var loginXML:XML = new XML();
  // 创建要发送到服务器的 XML 格式的数据
  var loginElement:XMLNode = loginXML.createElement("login");
  loginElement.attributes.username = username_txt.text;
  loginElement.attributes.password = password_txt.text;
  loginXML.appendChild(loginElement);
  // 将 XML 格式的数据发送到服务器
  loginXML.sendAndLoad("http://www.flash-mx.com/mm/main.cfm",
  loginReplyXML);
};
```



您可以使用用户名 `JeanSmith` 和密码 `VerySecret` 来测试此代码。当用户单击登录按钮时，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。如果拒绝了该密码，服务器会通过以下消息进行响应：

```
<LOGINREPLY STATUS="OK" SESSION="4D968511" />
```

此 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，它将用于该会话其余部分客户端和服务器之间的所有通信中。如果拒绝了该密码，服务器会通过以下消息进行响应：

```
<LOGINREPLY STATUS="FAILURE" />
```

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：

```
// 构建一个 XML 对象来保存服务器的应答
var loginReplyXML:XML = new XML();
loginReplyXML.onLoad = function(success:Boolean) {
```

此 ActionScript 的第二个语句定义一个匿名（内联）函数，它在 onLoad 事件触发时调用。

登录按钮（login_btn 实例）用于将用户名和密码以 XML 的形式发送到服务器，并将 XML 响应加载到 SWF 文件中。您可以使用 sendAndLoad() 方法来实现，如下面的示例所示：

```
loginXML.sendAndLoad("http://www.flash-mx.com.com/mm/main.cfm",
  loginReplyXML);
```

首先，使用用户在 SWF 文件中输入的值创建 XML 格式的数据，然后使用 sendAndLoad 方法发送该 XML 对象。与 loadVariables() 函数中的数据类似，loginreply XML 元素异步到达（也就是说，它不等待结果即返回），并加载到 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 loginReplyXML 函数，该函数在 onLoad 处理函数触发时被调用，因此它可处理 loginreply 元素。

注意	此函数必须总是位于包含登录按钮的 ActionScript 的帧上。

如果登录成功，SWF 文件将移动到 welcome 帧标签。如果登录不成功，播放头将移动到 loginfailure 帧标签。这通过使用条件和 case 语句来处理。有关 case 和 break 语句的更多信息，请参见《ActionScript 2.0 语言参考》中的 case 语句和 break 语句。有关条件的更多信息，请参见《ActionScript 2.0 语言参考》中的 if 语句和 else 语句。

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《ActionScript 2.0 语言参考》中的 XML 条目。有关本地文件安全性的更多信息，请参见第 605 页的“关于本地文件安全性和 Flash Player”。

可在硬盘上的 Samples 文件夹中找到范例源文件 login.fla。此范例演示如何使用 ActionScript 2.0 向网站中添加简单登录功能。该范例使用 ActionScript 和组件创建小型表单，可在其中输入用户名和密码，然后单击按钮以进入站点。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\Login。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/Login。

Flash Player 8 为 XML 类、LoadVars 类和 MovieClipLoader 类引入 onHTTPStatus 事件处理函数，允许用户从 HTTP 请求中访问状态代码。这样开发人员可以确定特定加载操作失败的原因，而不只是确定加载操作已经失败。

下面的示例演示如何使用 XML 类的 onHTTPStatus 事件处理函数检查是否从服务器中成功下载了 XML 文件，并检查从 HTTP 请求返回的状态代码。

使用 XML 类检查 HTTP 状态代码：

1. 创建一个新的 Flash 文档，并将该文档保存为 **xmlhttp.fla**。
2. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onHTTPStatus = function(httpStatus:Number) {
  trace("HTTP status is: " + httpStatus);
};
my_xml.onLoad = function(success:Boolean) {
  if (success) {
    trace("XML successfully loaded");
    // 0 (No error; parse was completed successfully.)
    trace("XML status is:" + my_xml.status);
  } else {
    trace("unable to load XML");
  }
};
my_xml.load("http://www.helpexamples.com/crossdomain.xml");
```

   前面的代码使用变量名 my_xml 定义新的 XML 对象，还定义两个事件处理函数（onHTTPStatus 和 onLoad），并加载外部 XML 文件。onLoad 事件处理函数将进行检查，以确定是否成功加载了 XML 文件，如果是这样，则向“输出”面板发送一条消息，并输出 XML 对象的状态属性。务必记住，onHTTPStatus 事件侦听器将返回从 Web 服务器返回的状态代码，而 XML.status 属性包含一个数值，指示是否能成功分析 XML 对象。

3. 选择“控制”>“测试影片”对该 Flash 文档进行测试。

提示 XML.onHTTPStatus 事件处理函数是 Flash Player 8 中的新增函数。

警告 不要将 HTTP httpStatus 代码与 XML 类的 status 属性相混淆。onHTTPStatus 事件处理函数从 HTTP 请求返回服务器的状态代码，而 status 属性自动设置并返回一个数值，指示 XML 文档是否已成功分析为 XML 对象。

小心 如果 Web 服务器不将 status 代码返回给 Flash Player，将为 onHTTPStatus 事件处理函数返回数字 0。

硬盘中有几个范例文件可在运行时将 XML 加载到 SWF 文件中。一个范例演示如何通过加载、分析和操作 XML 数据来创建 Web 日志跟踪器。可在硬盘上的 Samples 文件夹中找到范例源文件 xml_blogTracker.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_BlogTracker。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_BlogTracker。

第二个范例演示如何使用 XML 和嵌套数组来选择不同语言的字符串，以填充文本字段。可在硬盘上的 Samples 文件夹中找到范例源文件 xml_languagePicker.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_LanguagePicker。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_LanguagePicker。

第三个示例演示如何使用 XML 数据创建动态菜单。该示例调用 ActionScript `XmlMenu()` 构造函数并为其传递两个参数：XML 菜单文件的路径和对当前时间轴的引用。该功能的其余部分均位于自定义类文件 XmlMenu.as 中。

可在硬盘上的 Samples 文件夹中找到范例源文件 xmlmenu.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\XML_Menu。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/XML_Menu。

使用 XMLSocket 类

ActionScript 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。套接字连接使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反应时间问题，它通常用于实时的应用程序，比如聊天。数据以单个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。

警告	XMLSocket 类不能自动穿过防火墙，因为 XMLSocket 没有 HTTP 隧道功能（这与实时消息传递协议 (RTMP) 不同）。如果您需要使用 HTTP 隧道，应考虑改用 Flash Remoting 或 Flash Communication Server（支持 RTMP）。

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。

调用 connect() 方法时，Flash Player 打开与服务器的 TCP/IP 连接，并使该连接保持打开状态，直到发生以下任一事件：

- 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 XMLSocket 对象
var theSocket:XMLSocket = new XMLSocket();
// 使用 connect() 方法连接到站点上大于 1024 的空闲端口。
// 输入 localhost 或 127.0.0.1 进行本地测试。
// 对现场服务器，输入您的域 www.yourdomain.com
theSocket.connect("localhost", 12345);
// 显示关于连接的文本
theSocket.onConnect = function(myStatus) {
  if (myStatus) {
    conn_txt.text = "connection successful";
  } else {
    conn_txt.text = "no connection made";
  }
};
// 要发送的数据
function sendData() {
  var myXML:XML = new XML();
  var mySend = myXML.createElement("thenode");
  mySend.attributes.myData = "someData";
  myXML.appendChild(mySend);
  theSocket.send(myXML);
}
// 按钮发送数据
sendButton.onRelease = function() {
  sendData();
};
// 输出从套接字连接返回的数据
theSocket.onData = function(msg:String):Void {
  trace(msg);
};
```

有关更多信息，请参见《ActionScript 2.0 语言参考》中的 XMLSocket 条目。

有关本地文件安全性的更多信息，请参见第 605 页的“关于本地文件安全性和 Flash Player”。

向 Flash Player 发送消息以及从 Flash Player 接收消息

要从 SWF 文件向其主机环境（例如 Web 浏览器、Macromedia Director 影片或独立 Flash Player）发送消息，可以使用 `fscommand()` 函数。通过此函数可以使用主机的功能来扩展 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。

若要从 Web 浏览器脚本撰写语言（例如 JavaScript、VBScript 和 Microsoft Jscript）控制 Flash Player 中的 SWF 文件，可使用 Flash Player 的方法，即从宿主环境向 SWF 文件发送消息的函数。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。

有关更多信息，请参见以下主题：

- 第 590 页的 “使用 fscommand() 函数”
- 第 593 页的 “关于使用 JavaScript 控制 Flash 应用程序”
- 第 593 页的 “关于 Flash Player 方法”

使用 fscommand() 函数

注意

外部 API 是对 Flash 8 中的 `fscommand()` 的替代，用于与 HTML 页或容器应用程序进行互操作。在此情况下，外部 API 可比 `fscommand()` 提供更可靠的功能。有关更多信息，请参见第 594 页的 “关于外部 API”。

使用 `fscommand()` 函数，可以将消息发送到承载 Flash Player 的那个程序，例如 Web 浏览器。

注意

使用 `fscommand()` 调用 JavaScript 在 Macintosh 的 Safari 或 Internet Explorer 浏览器上不起作用。

`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。例如，下面的事件处理函数将独立播放器设置为在按钮释放时将 SWF 文件缩放至整个显示器屏幕大小：

```
my_btn.onRelease = function() {
  fscommand("fullscreen", true);
};
```

下表显示了可为 fscommand() 的 *command* 和 *arguments* 参数指定的值，能借此控制 SWF 文件在独立播放器（包括放映文件）中的回放和外观。

注释 放映文件 是以可作为独立应用程序运行的格式保存的 SWF 文件，也就是将 Flash Player 和内容嵌入可执行文件中。

命令	参量	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	指向应用程序的路径	在投影仪内执行应用程序。

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 *moviename*_DoFSCommand，其中 *moviename* 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 SWF 文件指定的名称为 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. 创建一个新的 FLA 文件，并保存为 **myMovie.fla**。
2. 将 Button 组件的两个实例拖动到舞台上，分别为它们指定实例名称 **window_btn** 和 **alert_btn** 以及标签 **Open Window** 和 **Alert**。
3. 在时间轴上插入一个新图层，并重命名为 **Actions**。
4. 选择 “动作”图层的 “第 1 帧”，在 “动作”面板中添加下面的 ActionScript：

```
window_btn.onRelease = function() {
  fscommand("popup", "http://www.macromedia.com/");
};
alert_btn.onRelease = function() {
  fscommand("alert", "You clicked the button.");
};
```

5. 选择“文件”>“发布设置”，并确保在“HTML”选项卡上的“模板”菜单中选择带有 FSCommand 的 Flash。
6. 选择“文件”>“发布”来生成 SWF 和 HTML 文件。
7. 在 HTML 或文本编辑器中，打开在步骤 6 中生成的 HTML 文件并检查代码。当您使用“发布设置”对话框的“HTML”选项卡上“带有 FSCommand 的 Flash”模板来发布 SWF 文件时，将会在 HTML 文件中插入一些附加代码。该 SWF 文件的 NAME 和 ID 属性将是文件名。例如，对于文件 myMovie.fla，这些属性将设置为 myMovie。
8. 在 HTML 文件中，在文档中带有 // 请在此处添加代码。字样的位置添加下面的 JavaScript 代码：

```
if (command == "alert") {
  alert(args);
} else if (command == "popup") {
  window.open(args, "mmwin", "width=500,height=300");
}
```

有关发布的更多信息，请参见《使用 Flash》中的第 17 章“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>
```

9. 保存和关闭 HTML 文件。

当您用这种方法在 Flash 外部编辑 HTML 文件时，请记住必须在“文件”>“发布设置”中取消选择 HTML 复选框，否则重新发布时您的 HTML 代码将被 Flash 覆盖。

10. 在 Web 浏览器中，打开并查看 HTML 文件。单击“打开窗口”按钮；将打开一个到 Macromedia 网站的窗口。单击“Alert”按钮；即会打开一个警告窗口。

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，请参见 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 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。

关于使用 JavaScript 控制 Flash 应用程序

Flash Player 6 (6.0.40.0) 和更高版本支持在 Netscape 6.2 或更高版本中使用特定于 Flash 应用程序的 JavaScript 方法和 FSCommand。早期版本不支持在 Netscape 6.2 或更高版本中使用这些 JavaScript 方法和 FSCommand。有关更多信息，请参见 Macromedia 支持中心的文章 “使用 Flash 撰写脚本”，网址为 www.macromedia.com/support/flash/publishexport/scriptingwithflash/。

对于 Netscape 6.2 和更高版本，无需将 swliveconnect 属性设置为 true。但是，将 swLiveConnect 设置为 true 并不会给 SWF 文件带来负面影响。有关更多信息，请参见《使用 Flash》中第 405 页的 “参数和属性”中的 swLiveConnect 属性。

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 ActionScript）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。

有两种技术可允许浏览器和 Flash Player 之间进行通信: LiveConnect（Windows 95/98/2000/NT/XP 或 Power Macintosh 上的 Netscape Navigator 3.0 或更高版本）和 ActiveX（Windows 95/98/2000/NT/XP 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 在 Flash 支持中心进行搜索，网址为 www.macromedia.com/go/flash_support_cn。

关于外部 API

ExternalInterface 类也被称为 外部 API，是一个新的子系统，通过它可以在 ActionScript 和 Flash Player 容器与具有 JavaScript 的 HTML 页或嵌入 Flash Player 的桌面应用程序之间轻松进行通信。

注意

此功能替换较旧的 `fscommand()` 函数，以与 HTML 页或容器应用程序进行互操作。在此情况下，外部 API 可比 `fscommand()` 提供更可靠的功能。有关更多信息，请参见第 594 页的 “关于外部 API”。

ExternalInterface 类只在以下环境下可用：

- 在所有受支持的 Internet Explorer for Windows （5.0 和更高版本）版本中。
- 在嵌入式自定义 ActiveX 容器中，例如嵌入 Flash Player ActiveX 控件的桌面应用程序。
- 在支持 NPRuntime 接口的任何刘浏览器中 （当前包括以下浏览器：
  - Firefox 1.0 及更高版本
  - Mozilla 1.7.5 及更高版本
  - Netscape 8.0 及更高版本
  - Safari 1.3 及更高版本。

在所有其它情况下，`ExternalInterface.available` 属性均返回 false。

从 ActionScript 中，可以在 HTML 页上调用 JavaScript 函数。外部 API 与 `fscommand()` 相比，可提供以下改进的功能：

- 可以使用任何 JavaScript 函数，而不仅仅是可与 `fscommand` 函数一起使用的函数。
- 可以传递任意数量的、具有任意名称的参数；不限于传递一个命令和参数。
- 可以传递各种数据类型（例如 Boolean、Number 和 String）；不再仅限于 String 参数。
- 现在可以接收调用值，并将该值直接返回给 ActionScrip （作为调用的返回值）。

可以从 HTML 页上的 JavaScript 中调用 ActionScript 函数。有关更多信息，请参见 `ExternalInterface (flash.external.ExternalInterface)`。有关本地文件安全性的更多信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。

下面部分中包含了使用外部 API 的示例：

- 第 595 页的 “创建与外部 API 的交互”
- 第 598 页的 “使用外部 API 控制 Flash Video”

创建与外部 API 的交互

可在浏览器和嵌入网页的 SWF 文件之间创建交互。以下过程将文本发送到包含 SWF 文件的 HTML 页，而 HTML 在运行时会将消息发送回 SWF 文件。

创建 Flash 应用程序：

1. 创建一个新的 Flash 文档，并将它保存为 **extint.fla**。
2. 将两个 TextInput 组件拖动到舞台上，并为其给定实例名称 **in_ti** 和 **out_ti**。
3. 将一个 Label 组件拖动到舞台上，为其分配实例名称 **out_lbl**，将其定位在 `out_ti` TextInput 实例之上，并在属性检查器的“参数”选项卡中将文本属性设置为 **Sending to JS:**。
4. 将一个 Button 组件拖动到舞台上，将其定位在 `out_lbl` 标签旁边，并为其给定实例名称 **send_button**。
5. 将一个 Label 组件拖动到舞台上，并为其分配实例名称 **in_lbl**，将其定位在 `in_ti` TextInput 实例之上，并在“参数”选项卡中将其文本属性设置为从 **JS 接收 :**。
6. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
import flash.external.ExternalInterface;

ExternalInterface.addCallback("asFunc", this, asFunc);
function asFunc(str:String):Void {
  in_ti.text = "JS > Hello " + str;
}

send_button.addEventListener("click", clickListener);
function clickListener(eventObj:Object):Void {
  trace("click > " + out_ti.text);
  ExternalInterface.call("jsFunc", out_ti.text);
}
```

   前面的代码分为三个部分。第一个部分导入 ExternalInterface 类，因此无需使用完全限定的类名称。第二部分代码定义一个回调函数 `asFunc()`，在后面的示例中将在 HTML 文档中从 JavaScript 调用该函数。此函数设置舞台上 TextInput 组件内的文本。第三部分代码定义一个函数，并将其指定为用户单击舞台上的 Button 组件实例时的事件侦听器。无论何时单击按钮时，SWF 文件都将在 HTML 页中调用 `jsFunc()` JavaScript 函数，并传递 `out_ti` 文本输入实例的 text 属性。

7. 选择“文件”>“发布设置”，然后选择“格式”选项卡并确保 Flash 和 HTML 都已选中。
8. 单击“发布”创建 HTML 和 SWF 文件。

   完成后，转到下一过程为 SWF 文件创建容器。

在对前面的 Flash 文档进行测试之前，需要修改生成的 HTML 代码并添加一些其它的 HTML 和 JavaScript。下面的过程为 SWF 文件修改 HTML 容器，以便两个文件在浏览器中运行时可以进行交互。

为 SWF 文件创建 HTML 容器：

1. 完成前面的过程。
2. 打开当您发布应用程序时 Flash 创建的 extint.html 文件。

   它与 Flash 文档位于相同文件夹中。
3. 在开始和结束 `head` 标签之间添加以下 JavaScript 代码：

```
<script language="JavaScript">
<!--
  function thisMovie(movieName) {
    var isIE = navigator.appName.indexOf("Microsoft") != -1;
    return (isIE) ? window[movieName] : document[movieName];
  }

  function makeCall(str) {
    thisMovie("extint").asFunc(str);
  }

  function jsFunc(str) {
    document.inForm.inField.value = "AS > Hello " + str;
  }
// -->
</script>
```

   此 JavaScript 代码定义三种方法。第一个方法根据用户的浏览器是 Microsoft Internet Explorer (IE) 还是 Mozilla 浏览器返回对嵌入式 SWF 文件的引用。第二个函数 `makeCall()` 调用 `asFunc()` 方法，您在前面示例中的 Flash 文档中定义了该方法。`thisMovie()` 函数调用中的 `"extint"` 参数引用嵌入式 SWF 文件的对象 ID 和嵌入名称。如果将 Flash 文档保存为不同的名称，则需要更改此字符串以便与对象和嵌入标签中的值匹配。第三个函数 `jsFunc()` 设置在 HTML 文档中 `inField` 文本字段的值。当用户单击 `send_button` 按钮组件时，将从 Flash 文档中调用此函数。

4. 在结束 </body> 标签之前添加以下 HTML 代码：

```
<form name="outForm" method="POST"
  action="javascript:makeCall(document.outForm.outField.value);">
  Sending to AS:<br />
  <input type="text" name="outField" value="" /><br />
  <input type="submit" value="Send" />
</form>

<form name="inForm" method="POST" action="">
  Receiving from AS:<br />
  <input type="text" name="inField">
</form>
```

此 HTML 代码创建两个 HTML 表单，它们与上一练习中在 Flash 环境内创建的表单类似。第一个表单将 outField 文本字段的值提交给上一步骤中所定义的 makeCall() JavaScript 函数。第二个表单用于显示当用户单击 send_button 实例时从 SWF 文件发送的值。

5. 保存该 HTML 文档，并将 HTML 和 SWF 文件都上载到 Web 服务器。

6. 在 Web 浏览器中查看 HTML 文件，在 out_ti TextInput 实例中输入一个字符串，然后单击 “Send” 按钮。

   Flash 将调用 jsFunc() JavaScript 函数并传递 out_ti 文本字段的内容，它将在 HTML 表单 inForm inField 输入文本字段中显示此内容。

7. 在 outField HTML 文本字段中键入一个值，然后单击 “Send” 按钮。

   Flash 将调用 SWF 文件的 asFunc() 函数，后者将在 in_ti TextInput 实例中显示字符串。

可在硬盘上的 Samples 文件夹中找到范例源文件 ExtInt.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\ExternalAPI\simple example。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/ExternalAPI/simple example。

有关使用外部 API 的更复杂的示例，请参见第 598 页的 “使用外部 API 控制 Flash Video”。有关本地文件安全性的更多信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。

提示 请避免使用其它访问插件对象的方法，例如 document.getElementById("pluginName") 或 document.all.pluginName，因为这些其它方法不能在所有浏览器中一致地工作。

使用外部 API 控制 Flash Video

下面的过程演示如何使用 HTML 页中的控件控制 Flash Video (FLV) 文件，并在 HTML 文本字段中显示有关视频的信息。此过程使用外部 API 来实现此功能。

使用外部 API 构建 Flash 应用程序：

1. 创建一个新的 Flash 文档，并将该文档保存为 **video.fla**。
2. 从 “库”面板的弹出菜单中选择 “新建视频”，以向库中添加新的视频元件。
3. 将视频元件拖动到舞台上，并为其给定实例名称 **selected_video**。
4. 选择 selected_video 实例，然后选择属性检查器，将实例的大小调整为 **320** 像素宽，**240** 像素高。
5. 将视频位置的 x 和 y 坐标都设置为 **0**。
6. 选择舞台，并使用属性检查器将其尺寸调整为 **320** x **240** 像素。

   现在，舞台即与视频实例的尺寸匹配。
7. 将下面的 ActionScript 添加到主时间轴中的第 1 帧：

```
import flash.external.ExternalInterface;

/* 注册 playVideo() 和 pauseResume() 以便可以
从容器 HTML 页中的 JavaScript 中调用它们。 */
ExternalInterface.addCallback("playVideo", null, playVideo);
ExternalInterface.addCallback("pauseResume", null, pauseResume);

/* 该视频需要使用 NetConnection 和 NetStream 对象。 */
var server_nc:NetConnection = new NetConnection();
server_nc.connect(null);
var video_ns:NetStream = new NetStream(server_nc);

/* 将 NetStream 对象附加到舞台上的视频对象，
以便在视频对象中显示 NetStream 数据。 */
selected_video.attachVideo(video_ns);

/* 当 NetStream 对象的状态更新时 (例如视频开始播放)，
将自动调用 onStatus() 方法。
当出现这种情况时，请通过 ExternalInterface
调用 JavaScript updateStatus() 函数，以将代码属性值发送至 HTML 页。 */
video_ns.onStatus = function(obj:Object):Void {
  ExternalInterface.call("updateStatus", "   " + obj.code);
};

function playVideo(url:String):Void {
  video_ns.play(url);
}

function pauseResume():Void {
  video_ns.pause();
}
```

第一部分 ActionScript 代码定义两个 ExternalInterface 回调函数：playVideo() 和 pauseResume()。在下面的过程中，将从 JavaScript 中调用这些函数。第二部分代码创建一个新的 NetConnection 和 NetStream 对象，可将其用于视频实例，以动态回放 FLV 文件。

下一过程中的代码为 video_ns NetStream 对象定义了一个 onStatus 事件处理函数。只要 NetStream 对象一更改状态，Flash 便使用 ExternalInterface.call() 方法触发自定义 JavaScript 函数 updateStatus()。最后两个函数 playVideo() 和 pauseResume() 控制视频实例在舞台上的回放。在以下过程中编写的 JavaScript 中，这两个函数都会被调用。

8. 保存 Flash 文档。
9. 选择 “文件” > “发布设置”，然后选择 “格式”选项卡，并确保 HTML 和 Flash 都已选中。
10. 单击 “发布”，将 SWF 和 HTML 文件发布到硬盘。

    完成后，转到下一过程并为 SWF 文件创建容器。

可在硬盘上的 Samples 文件夹中找到范例源文件 external.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\ExternalAPI。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/ExternalAPI。

在下面的过程中，您将修改在上一过程中由 Flash 生成的 HTML 代码。本过程创建在 SWF 文件中播放 FLV 文件所需的 JavaScript 和 HTML。

为 SWF 文件创建容器：

1. 完成前面的过程。
2. 打开您在前一过程的最后一个步骤中发布的 video.html 文档。
3. 修改现有代码，以使其与下面的代码匹配：

注意	查看以下示例中的代码注释。此代码示例后提供了代码概述。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
  www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" /
  >
<title>ExternalInterface</title>

<script language="JavaScript">
  // 使用一个变量来引用嵌入的 SWF 文件。
```

```
    var flashVideoPlayer;

    /* 当 HTML 页加载时（通过 <body> 标签的 onLoad 事件），它将调用 initialize() 函
    数。 */
    function initialize() {
      /* 检查浏览器是否为 IE。如果是这样，则 flashVideoPlayer 为
  window.videoPlayer。否则，它为 document.videoPlayer。
  videoPlayer 是分配给 <object> 和 <embed> 标签的 ID。 */
      var isIE = navigator.appName.indexOf("Microsoft") != -1;
      flashVideoPlayer = (isIE) ? window['videoPlayer'] :
    document['videoPlayer'];
    }

    /* 当用户单击表单中的播放按钮时，将更新 videoStatus 文本区域，并调用 SWF 文件中的
    playVideo() 函数，同时向其传递 FLV 文件的 URL。 */
    function callFlashPlayVideo() {
      var comboBox = document.forms['videoForm'].videos;
      var video = comboBox.options[comboBox.selectedIndex].value;
      updateStatus("____" + video + "____");
      flashVideoPlayer.playVideo("http://www.helpexamples.com/flash/video/
    " + video);
    }

    // 调用 SWF 文件中的 pauseResume() 函数。
    function callFlashPlayPauseVideo() {
      flashVideoPlayer.pauseResume();
    }

    /* 将在 SWF 文件中 NetStream 对象的 onStatus() 访法中调用 updateStatus() 函
    数。 */
    function updateStatus(message) {
      document.forms['videoForm'].videoStatus.value += message + "\n";
    }
  </script>
  </head>
  <body bgcolor="#ffffff" onLoad="initialize();">

  <object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
    swflash.cab#version=8,0,0,0" width="320" height="240" id="videoPlayer"
    align="middle">
  <param name="allowScriptAccess" value="sameDomain" />
  <param name="movie" value="video.swf" />
  <param name="quality" value="high" />
  <param name="bgcolor" value="#ffffff" />
  <embed src="video.swf" quality="high" bgcolor="#ffffff" width="320"
    height="240" name="videoPlayer" align="middle"
    allowScriptAccess="sameDomain" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
  </object>
```

```
<form name="videoForm">
  Select a video:<br />
  <select name="videos">
    <option value="lights_long.flv">lights_long.flv</option>
    <option value="clouds.flv">clouds.flv</option>
    <option value="typing_long.flv">typing_long.flv</option>
    <option value="water.flv">water.flv</option>
  </select>
  <input type="button" name="selectVideo" value="play"
  onClick="callFlashPlayVideo();" />
  <br /><br />

  Playback <input type="button" name="playPause" value="play/pause"
  onClick="callFlashPlayPauseVideo();" />

  <br /><br />
  Video status messages <br />
  <textarea name="videoStatus" cols="50" rows="10"></textarea>
</form>

</body>
</html>
```

此 HTML 代码定义四个 JavaScript 函数：`initialize()`、`callFlashPlayVideo()`、`callFlashPlayPauseVideo()` 和 `updateStatus()`。`initialize()` 函数将在 `onLoad` 事件的 `body` 标签内调用。当用户在 HTML 文档中单击播放按钮或播放 / 暂停按钮时，将调用 `callFlashPlayVideo()` 和 `callFlashPlayPauseVideo()` 函数，并触发 SWF 文件中的 `playVideo()` 和 `pauseResume()` 函数。

只要触发 `video_ns` NetStream 对象的 `onStatus` 事件处理函数，SWF 文件就将调用最后一个函数 `updateStatus()`。此 HTML 代码还定义一个表单，其中包含用户可从中进行选择的视频组合框。只要用户选择一个视频并单击播放按钮，便会调用 `callFlashPlayVideo()` JavaScript 函数，该函数然后将调用 SWF 文件中的 `playVideo()` 函数。此函数将传递加载到视频实例的 SWF 文件的 URL。随着视频的播放以及 NetStream 对象状态的更改，舞台上 HTML 文本区域的内容将更新。

4. 保存对 HTML 文档的更改，然后将 HTML 和 SWF 文件上载到网站中。
5. 从网站中打开远程 video.html 文档，从组合框中选择一个视频，然后单击播放按钮。Flash 播放选定的 FLV 文件，并更新 HTML 文档中 videoStatus 文本区域中的内容。

可在硬盘上的 Samples 文件夹中找到范例源文件 external.fla。

- 在 Windows 中，浏览到 boot drive\Program Files\Macromedia\Flash 8\Samples and Tutorials\Samples\ActionScript\ExternalAPI。
- 在 Macintosh 上，浏览到 Macintosh HD/Applications/Macromedia Flash 8/Samples and Tutorials/Samples/ActionScript/ExternalAPI。

有关外部 API 的更多信息，请参见 《ActionScript 2.0 语言参考》中的 `ExternalInterface (flash.external.ExternalInterface)`。

有关本地文件安全性的更多信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。

注意

请避免使用其它访问插件对象的方法，例如 document.getElementById("pluginName") 或 document.all.pluginName，因为这些其它方法不能在所有浏览器中一致地工作。

第 17 章

17 了解安全性

在 Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 中，可以使用 ActionScript 将外部源数据加载到 SWF 文件中或将数据发送到服务器。将数据加载到 SWF 文件时，需要了解并考虑 Flash 8 安全模型。打开硬盘上的 SWF 文件时，可能需要进行一些特殊配置，以便在本地测试文件。

有关本地文件安全性的信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。有关 Flash Player 7 和 Flash Player 8 安全模型之间的不同，请参见第 604 页的 “关于与早期 Flash Player 安全模型的兼容性”。有关如何从服务器加载和分析数据的信息，请阅读第 565 页的第 16 章 “使用外部数据”。有关安全性的更多信息，请参见 www.macromedia.com/devnet/security 和 www.macromedia.com/software/flashplayer/security/。

有关 Flash 8 中安全性的更多信息，请参见以下主题：

关于与早期 Flash Player 安全模型的兼容性

由于 Flash Player 7 中的安全功能有所改变，因此在 Flash Player 6 或更低版本中按预期运行的内容在 Flash Player 更高版本中可能不会按照预期运行。例如，在 Flash Player 6 中，驻留在 www.macromedia.com 中的 SWF 文件可以读取 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 文件加载类似域中的数据。

在 Flash Player 7 和更高版本中，如果第 6 版（或更低版本）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 文件的域进行读取操作的策略文件，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”，则该 SWF 文件可以访问请求的数据；如果用户单击 “拒绝”，则该 SWF 文件不能访问请求的数据。

如果不想让该对话框出现，应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 625 页的 “允许跨域数据加载”。

如果没有安全策略文件，则 Flash Player 7 和更高版本将不允许跨域访问。

Flash Player 8 改变了其处理 `System.security.allowDomain` 的方式。使用任何参数调用 `System.security.allowDomain` 的 Flash 8 SWF 文件，或其它任何使用通配符 (*) 值的 SWF 文件，允许只访问本身。现在可以对通配符 (*) 值进行支持，例如：`System.security.allowDomain("*")` and `System.security.allowInsecureDomain("*")`。如果第 7 版或更低版本的 SWF 文件使用通配符 (*) 之外的参数调用 `System.security.allowDomain` 或 `System.security.allowInsecureDomain`，这会影响到进行调用的 SWF 文件域中的所有第 7 版或更早版本的 SWF 文件，和 Flash Player 7 一样。然而，这种调用不会影响进行调用的 SWF 文件域中的任何 Flash Player 8 （或更高版本） SWF 文件。这有助于很好地保证 Flash Player 的兼容性。

有关更多信息，请参见第 618 页的 “关于域、跨域安全性和 SWF 文件”、 allowDomain （security.allowDomain 方法）和 allowInsecureDomain （security.allowInsecureDomain 方法）。

如果不对计算机进行特定配置，则 Flash Player 8 将不允许本地 SWF 文件与 Internet 通信。假定您有在这些限制生效以前发布的旧内容。如果该内容试图与网络或 / 和本地文件系统进行通信，则 Flash Player 8 将停止该操作，您必须显式提供使应用程序正常工作的权限。有关更多信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。

关于本地文件安全性和 Flash Player

Flash Player 8 已增强了安全模型，在该安全模型中，默认设置为不允许本地计算机上的 Flash 应用程序和 SWF 文件与 Internet 和本地文件系统通信。本地 SWF 文件 是指在用户计算机上本地安装的 SWF 文件，它不接受 Web 站点提供的服务，也不包括放映 (EXE) 文件。

注释	本部分讨论的限制不影响 Internet 上的 SWF 文件无效。

创建 FLA 文件时，可以指示是否允许 SWF 文件与网络或本地文件系统通信。在 Flash Player 早期版本中，本地 SWF 文件可以与其它 SWF 文件交互并且可以从任何远程或本地位置加载数据。在 Flash Player 8 中，SWF 文件不能连接本地文件系统和 Internet。这是一种安全变化，使 SWF 文件不能读取您硬盘上的文件然后通过 Internet 发送这些文件的内容。

此安全限制影响所有本地部署的内容，无论是旧内容 （在 Flash 的早期版本中创建的 FLA 文件）还是在 Flash 8 中创建的内容。假定您使用 Flash MX 2004 或更早版本部署了一个本地运行并且也访问 Internet 的 Flash 应用程序。在 Flash Player 8 中，该应用程序现在将提示用户提供与 Internet 通信的权限。

测试硬盘上的文件时，可通过一系列的步骤来确定文件是本地受信任文档还是潜在不受信任文档。如果文件是在 Flash 创作环境中创建的 （例如，选择 “控制” > “测试影片”），由于是在测试环境中，因此该文件是受信任的。

在 Flash Player 7 和更低版本中，本地 SWF 文件具有从本地文件系统和网络（如 Internet）读取内容的权限。在 Flash Player 8 中，本地 SWF 文件可以具有以下权限级别:

只访问本地文件 （默认） 本地 SWF 文件可从本地文件系统和统一命名约定 (UNC) 网络路径读取内容，但不能与 Internet 通信。有关进行本地文件访问的 SWF 文件的更多信息，请参见第 611 页的 “只访问本地文件 （默认）”。

只访问网络 本地 SWF 文件可以访问网络 （如 Internet），但不能访问安装该文件的本地文件系统。有关只访问网络的 SWF 文件的更多信息，请参见第 612 页的 “只访问网络”。

访问本地文件系统和网络 本地 SWF 文件可以从安装它的本地文件系统中读取内容，对服务器进行读写操作，并可以对网络或本地文件系统上的其它 SWF 文件进行跨脚本操作。这些文件都是受信任文件，其行为与在 Flash Player 7 中相同。有关进行本地和网络访问的 SWF 文件的更多信息，请参见第 612 页的 “访问文件系统和网络”。

有关 Flash 8 中适用于创作工具的本地文件安全性的更多信息，请参见以下各部分：

- 第 606 页的 “理解本地安全性沙箱”
- 第 607 页的 “关于 Flash Player 安全设置”
- 第 609 页的 “关于本地文件安全性和放映文件”
- 第 609 页的 “关于旧 SWF 文件疑难解答”
- 第 610 页的 “修复部署在本地计算机上的旧内容”
- 第 610 页的 “发布文件以便在本地部署”

有关面向用户的本地文件安全性的信息，请参见第 607 页的 “关于 Flash Player 安全设置”。有关安全性的更多信息，请参见 www.macromedia.com/devnet/security/ 和 www.macromedia.com/software/flashplayer/security/。

理解本地安全性沙箱

Flash Player 有几种不同的安全性沙箱。每个安全性沙箱确定 SWF 文件如何与本地文件系统、网络或如何同时与本地文件系统和网络交互。限制文件如何与本地文件系统或网络交互可以帮助确保计算机和文件的安全。理解安全性沙箱有助于在计算机上顺利开发和测试 Flash 应用程序，而不会遇到意外错误。

Local-with-file-system

出于安全性的考虑，默认时（除非进行其它设置）Flash Player 8 会将所有本地 SWF 文件（包括所有旧的本地 SWF 文件）放入 Local-with-file-system 沙箱。对于有些旧（Flash Player 8 之前版本）SWF 文件，强制限制其访问会影响到一些操作（没有外部网络访问），但这样可以为用户保护提供最安全的默认设置。

从该沙箱中，SWF 文件可以读取本地文件系统或 UNC 网络路径上文件的内容（使用 `XML.load()` 方法），但可能不能以任何方式与网络通信。这样可向用户保证本地数据不会泄漏到网络或以其它方式不适当地共享。

Local-with-networking

将本地 SWF 文件分配到 Local-with-networking 沙箱时，这些文件会失去其本地文件系统访问权限。但允许这些 SWF 文件访问网络。然而，仍然不允许 Local-with-networking SWF 文件读取任何来自网络的数据，除非获得该操作的权限。因此，Local-with-networking SWF 文件没有本地访问权限，但能够通过网络传输数据并能从授予特定站点访问权限的站点读取网络数据。

Local-trusted

分配到 Local-trusted 沙箱的 SWF 文件可以与其它任何 SWF 文件交互，并能从任何位置（远程或本地）加载数据。

关于 Flash Player 安全设置

Macromedia 将 Flash Player 设计为可提供安全设置，并且在大多数情况下这些安全设置不要求您显式允许或拒绝访问。您可能偶尔会遇到使用 Flash Player 7 或更低版本的旧安全规则创建的旧 Flash 内容。在这些情况下，Flash Player 可以使用旧的安全规则以允许这些内容按照开发人员的意图那样工作；或者，您可以选择实施更新更严格的规则。后一个选择可确保您只查看或播放满足最新安全标准的内容，但有时也会使旧的 Flash 内容不能正常工作。

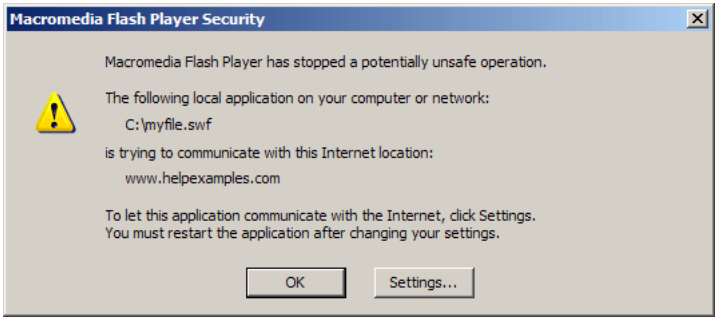
查看 SWF 文件的所有用户（包括非 Flash 开发人员）都可以通过“Flash Player”的“设置管理器”中的“全局安全设置”面板设置全局权限（如下图所示）。

在新版本播放器中运行旧内容并且 Flash Player 要求您决定是否实施新规则时，您可能会看到以下弹出对话框之一。这些对话框会在允许旧 Flash 内容与 Internet 上的其它位置通信之前要求您许可：

- 可能会出现一个对话框，警告您使用的 Flash 内容正在试图使用旧的安全规则访问其所在域外部的站点信息，并且将在两个站点之间共享该信息。Flash Player 询问您是允许还是拒绝此访问。

  除了响应对话框外，您还可以使用“全局安全设置”面板指定 Flash Player 是否应在允许访问之前通过对话框要求您许可；始终拒绝访问但不先询问；或始终允许访问其它站点或域但不要求您许可。

- （仅 Flash Player 8）可能会出现一个对话框，警告您 SWF 文件正在试图与 Internet 通信。默认情况下， Flash Player 8 不允许本地 Flash 内容与 Internet 通信。



  单击 “设置”访问 “全局安全设置”面板，在此可以指定计算机上的某些 Flash 应用程序可以与 Internet 通信。

若要更改安全设置或了解有关选项的更多内容，请使用 “全局安全设置”面板。使用此面板可重置 Macromedia Flash Player 中的隐私设置：

- 如果选择始终拒绝 然后确认选择，则任何试图使用您的摄像头和麦克风的 Web 站点将被拒绝访问。不会再次询问 Web 站点是否可以使用摄像头或麦克风。此动作同时适用于已访问和未访问的 Web 站点。
- 如果选择始终询问 然后确认选择，则任何试图使用摄像头和麦克风的 Web 站点必须要求您的许可。此动作同时适用于已访问和未访问的 Web 站点。

如果以前在 “隐私设置”面板中选择了 “记住”（见下图）以便永久允许或拒绝一个或多个 Web 站点的访问，则对于所有这些 Web 站点，选择 “始终询问”或 “始终拒绝”与取消选择 “记住”效果相同。也就是说，您在此做出的选择将覆盖您以前在 “隐私设置”面板中做出的任何选择，如下图所示。

选择了 “始终询问”或 “始终拒绝”（或不执行此操作）之后，您可以为已访问的各个 Web 站点指定隐私设置。例如，您可能在此选择 “始终拒绝”，然后使用 “Web 站点隐私设置”面板为您了解并信任的各个 Web 站点选择 “始终允许”。

对于本地部署的内容和本地数据，用户有另一个选项：可以使用 “全局安全设置”面板指定哪些 SWF 文件可以访问 Internet。有关在 “全局安全设置”面板中指定设置的更多信息，请参见第 613 页的 “使用 “设置管理器”指定受信任文件”。有关 “全局安全设置”面板的更多信息，请参见 www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04a.html。

注意	用户在 “全局安全设置”面板中所做的选择将覆盖在安全弹出对话框中所做的任何选择。

关于本地文件安全性和放映文件

由于最终用户必须运行可执行文件才能使用 SWF 文件，因此本地文件安全性限制对放映文件和放映文件中包含的 SWF 文件或在运行时加载到放映文件的 SWF 文件无效。 Flash Player 8 中的安全性和放映文件没有任何变化；它与 Flash Player 早期版本具有相同的访问级别和安全级别。

请记住用户在执行放映文件时总是非常小心。放映文件是一个可执行的 EXE 或 Macintosh 应用程序，用户在计算机上执行这种文件时应小心。如果使用放映文件分发应用程序，有些用户可能不安装该应用程序。

此外，放映文件在其内部嵌入 Flash Player 的特定版本，该版本可能比可从 Macromedia Web 站点下载的 Flash Player 的最新版本低。嵌入放映文件中的 Flash Player 可能是旧版本 （如果放映文件是用早期版本的 Flash 创建），或是在 Flash 创作工具当前版本之后发布的 Flash Player 版本。因此，应尽可能使用 SWF 文件分发应用程序。

关于旧 SWF 文件疑难解答

在对有些旧 FLA 和 SWF 文件 （使用 Flash MX 2004 和更低版本创建）进行本地测试或部署 （在硬盘上）时，由于 Flash 8 中的安全变化，这些文件可能不工作。 SWF 文件试图访问其所在域外部的 Web 站点时会发生这种情况，此时，需要实现跨域策略文件。

您可能有在 Flash MX 2004 或更低版本中创建的 FLA 或 SWF 文件并且这些文件分发到不使用 Flash 8 创作工具但已升级到 Flash Player 8 的用户。在 Flash Player 8 中播放在本地测试或部署的旧内容 （用户硬盘上的旧 SWF 文件）时，如果这些内容由于试图与 Internet 通信而损坏，则您必须依靠用户显式信任内容以确保正确播放（通过单击对话框中的按钮）。

有关如何修复旧内容以便在本地计算机上播放的信息，请参见第 610 页的 “修复部署在本地计算机上的旧内容”。

修复部署在本地计算机上的旧内容

如果发布部署在本地计算机上并与 Internet 通信的 Flash Player 7 或更低版本的 SWF 文件，用户必须显式允许 Internet 通信。用户可以通过将其本地计算机上 SWF 文件的位置添加到设置管理器中的 “受信任”沙箱来防止内容损坏。

若要修复 SWF 文件以便在本地播放，请使用以下任何选项：

重新部署 运行本地内容更新程序本地内容更新程序可重新配置您的 SWF 文件，使其与 Flash Player 8 安全模型兼容。您可以重新配置本地 SWF 文件，使其仅访问网络或仅访问本地文件系统。有关更多信息，或者要下载本地内容更新程序，请参见 www.macromedia.com/support/flashplayer/downloads.html。

重新发布和重新部署 使用 Flash Basic 8 或 Flash Professional 8 重新发布文件。创作工具要求您在“发布设置”对话框中指定本地 SWF 文件是否可以访问网络还是可以访问本地文件系统（但不能同时访问二者）。如果指定本地 SWF 文件可以访问网络，您还必须在其要访问的 SWF、HTML、数据和 / 或服务器文件中为该 SWF 文件（和所有本地 SWF 文件）赋予权限。有关更多信息，请参见第 610 页的 “发布文件以便在本地部署”。

部署新内容 使用 #Security/FlashPlayerTrust 文件夹中的配置 (.cfg) 文件。您可以使用此文件设置网络和本地访问权限。有关更多信息，请参见第 615 页的 “创建面向 Flash 开发的配置文件”。

注意	这些选项都要求您重新发布或重新部署 SWF 文件。

发布文件以便在本地部署

您可能会将 Flash 8 FLA 或 SWF 文件发送到用户以进行测试或审批，因此需要应用程序访问 Internet。如果您的文档在本地系统播放但又要访问 Internet 上的文件（例如，加载 XML 或发送变量），则您的用户可能需要一个配置文件以确保该内容正常工作，或者您可能需要对 FLA 文件进行设置使发布的 SWF 文件能访问网络。或者，还可以在 FlashPlayerTrust 目录内设置配置文件。有关设置配置文件的更多信息，请参见第 615 页的 “创建面向 Flash 开发的配置文件”。

使用 Flash Basic 8 或 Flash Professional 8 创建要在本地部署的内容，该内容使用 Flash Player 8 本地文件安全性。在 Flash 8 发布设置中，必须指定本地内容是可以访问网络还是可以访问本地文件系统，但不能同时访问二者。

可以在“发布设置”对话框中设置 FLA 文件的权限级别。在本地硬盘上播放 FLA 文件时，这些权限级别会对 FLA 文件的本地播放产生影响。

注意	如果指定本地文件可以访问网络，您还必须在该本地 SWF 文件访问的 SWF、HTML、数据和服务器文件中为其授予权限。

网络 SWF 文件 从网络上（如联机服务器）下载的 SWF 文件位于与其特有的 Web 站点源域相应的独立沙箱 中。指定具有网络访问权限的本地 SWF 文件位于 Local-with-networking 沙箱中。默认情况下，这些文件只能从提供下载该文件的相同站点读取数据。精确的域匹配应用于这些文件。如果具有相应权限，网络 SWF 文件就可以访问其它域的数据。有关网络 SWF 文件的更多信息，请参见第 612 页的 “只访问网络”。

本地 SWF 文件 对本地文件系统或 UNC 网络路径进行操作的 SWF 文件位于 Flash Player 8 中的三个沙箱之一。默认情况下，本地 SWF 文件位于 Local-with-file-system 沙箱。（使用配置文件）注册为 “受信任”的本地 SWF 文件位于 Local-trusted 沙箱。有关这三个沙箱的信息，请参见第 611 页的 “只访问本地文件 （默认）”。

有关安全性沙箱的更多信息，请参见第 606 页的 “理解本地安全性沙箱”。

前两个权限级别在 Flash 创作环境中设置，第三个使用 “全局安全设置”面板或 FlashAuthor.cfg 文件设置。下面的示例演示在发布文件以便在本地硬盘上进行测试时可用的选项。

发布已指定权限级别的文档：

1. 打开要指定权限级别的 FLA 文件。
2. 选择 “文件” > “发布设置” > “Flash”。
3. 找到 “本地回放安全性”对话框，从弹出菜单中选择以下选项之一：
   - 只访问本地文件 （请参见 “只访问本地文件 （默认）”）
   - 只访问网络 （请参见 “只访问网络”）
4. 单击 “确定”继续创作 FLA 文件，或单击 “发布”创建 SWF 文件。

有关可为应用程序设置的权限级别的更多信息，请参见第 611 页的 “只访问本地文件 （默认）”、第 612 页的 “只访问网络”和第 612 页的 “访问文件系统和网络”。

只访问本地文件 （默认）

若要设置此权限级别，请选择 “发布设置” > “Flash”，然后从 “本地回放安全性”弹出菜单中选择 “只访问本地文件”。此权限级别允许本地 SWF 文件仅访问运行该 SWF 文件的本地文件系统。 SWF 文件可以不受任何限制地从本地磁盘上的已知文件中读取内容。然而，以下限制应用于访问网络的应用程序：

- SWF 文件不能以任何方式访问网络。 SWF 文件不能对网络 SWF 文件执行跨脚本操作，也不能由网络 SWF 文件执行跨脚本操作。
- SWF 文件不能与具有只访问网络权限的本地 SWF 文件通信，并且 SWF 文件不能与 HTML 页面通信。然而，有时是允许进行通信的，例如，当 HTML 是受信任的而 `allowScriptAccess` 设置为`始终`，或者当 `allowScriptAccess` 未设置而 SWF 文件是 Flash Player 7 或更低版本时。

只访问网络

若要设置此权限级别，请选择 “发布设置” > “Flash”，然后从 “本地回放安全性” 弹出菜单中选择 “只访问网络”。如果某个服务器包含一个具有 `<allow-access-from-domain="*">` 的跨域策略文件，则具有网络访问权限的本地 SWF 文件就可以读取此服务器的内容。具有网络访问权限的本地 SWF 文件可以对其它 SWF 文件进行跨脚本操作，前提是正在被访问的其它 SWF 文件包含 `System.security.allowDomain("*")`。如果具有网络访问权限的本地 SWF 文件包含 `allowDomain("*")`，则该本地 SWF 文件可被网络 SWF 文件执行跨脚本操作。 SWF 文件始终不能读取本地文件的内容。在有些情况下， SWF 文件的类型会影响访问。有关信息，请参见 《ActionScript 2.0 语言参考》中的 allowDomain (`security.allowDomain` 方法)。

通配符 (*) 值指示允许访问所有域，包括本地主机。在使用通配符参数之前，请确定是否要提供这么大范围的访问权限。

如果没有这些权限，则具有网络访问权限的本地 SWF 文件就只能与其它具有网络访问权限的本地 SWF 文件通信，并且可以向服务器发送数据 （例如，使用 `XML.send()`） 。在这些情况下，如果 HTML 文件是受信任的，则允许进行访问。

访问文件系统和网络

此级别是最高的权限级别。具有这些权限的本地 SWF 文件是受信任的本地 SWF 文件。受信任的本地 SWF 文件可以读取其它本地 SWF 文件的内容，与任何服务器交互，并为其它没有显式禁止文件权限（例如，通过 `allowScriptAccess="none"`）的 SWF 文件或 HTML 文件编写 ActionScript。此权限级别可由用户或 Flash 开发人员通过以下方式授予：

- 使用 “设置管理器” 中的 “全局安全设置” 面板。
- 使用全局配置文件。

配置文件可随 SWF 文件安装，可由 Flash 开发人员创建，或由管理员（为所有用户或当前用户）或任何 Flash 开发人员 （为当前用户）添加。

有关配置文件和 “全局安全设置” 面板的更多信息，请参见第 607 页的 “关于 Flash Player 安全设置”、第 613 页的 “使用 “设置管理器” 指定受信任文件” 和第 615 页的 “创建面向 Flash 开发的配置文件”。

在存在 Flash 8 本地文件安全性限制的情况下对内容进行本地测试

作为 Flash 开发人员，您经常会在本地测试 Flash 应用程序，因此当本地 Flash 应用程序试图与 Internet 通信时，您可能会看到对话框提示。在 Flash Player 中测试 SWF 文件时，如果该 SWF 文件不具有网络访问权限，您可能会看到此对话框。有关发布具有指定权限级别的 SWF 文件的更多信息，请参见第 610 页的 “发布文件以便在本地部署”。发布具有这些选项之一的 SWF 文件表示您可以与网络或 本地文件系统通信。

在测试文档时，有时您可能需要与本地文件系统和网络通信。由于新的安全模型可能会在您创作 Flash 应用程序时中断您的工作流程，因此您可以使用 Flash Player “设置管理器”的“全局安全设置”面板指定您计算机上始终可与 Internet 和本地文件系统通信的 Flash 应用程序。或者，可以修改配置文件以指定硬盘上的受信任目录。

有关更多信息，请参见以下部分：

- 第 613 页的 “使用 “设置管理器”指定受信任文件”
- 第 615 页的 “创建面向 Flash 开发的配置文件”

使用 “设置管理器”指定受信任文件

可以指定计算机上的哪些 Flash 内容可以始终使用旧安全规则，只要将这些内容的位置添加到 Flash Player “设置管理器”中的 “全局安全设置”面板即可。将计算机上的位置添加到“安全”面板之后，该位置中的内容就成为受信任的内容。即使在 “安全”面板中选择了“始终拒绝”， Flash Player 也不会要求您许可而是始终允许使用旧安全规则。“这些位置”列表中的 “始终信任文件”将覆盖 “设置”面板中的选项。也就是说，如果选择始终拒绝授予本地和 Web 内容使用旧安全规则的权限，则受信任列表中的本地文件将始终允许使用这些旧规则。

该面板底部的 “始终信任文件”列表特别适用于已下载到计算机的 Flash 内容，而不适用于在访问 Web 站点时使用的内容。

下面的示例演示如何指定本地 SWF 文件可与 Internet 通信。在浏览器中对文件进行本地测试时，（“文件” > “发布预览” > “HTML”），可能会出现一个安全对话框。单击 “设置”，将出现 “设置管理器全局安全设置”面板

指定本地 SWF 文件可与 Internet 和本地文件系统通信：

1. 在 “全局安全设置”面板中，单击弹出菜单并选择 “添加位置”。

   “添加位置”框即打开。

   如果通过单击对话框中的 “设置”按钮使 “设置管理器”出现，“添加位置”框包含类似 C:\directoryname\filename.swf 或 /Users/directoryname/filename.swf 的路径；该路径指示试图与 Internet 通信且被 Flash Player 安全设置阻止的文件。如果此路径中包含您希望允许与 Internet 通信的内容，请复制此路径并粘贴到 “信任此位置”框。或者，单击其中一个 “浏览”按钮，自行找到该内容。

   您可以添加单个文件或整个目录。如果添加整个目录，则该目录中的所有文件和子目录都为受信任内容。有些 Flash 内容由多个相关文件组成，可能需要信任所有相关文件所在的整个目录。总之，应避免信任顶级目录。

2. 单击 “确认”。

该位置就添加到 “安全设置”面板上了。即使在 “安全”面板顶部选中 “始终拒绝”或 “始终询问”选项时，列出的位置始终允许使用旧安全规则。

添加受信任的位置后，必须通过刷新浏览器或重启播放器重新启动本地 Flash 内容。

如果单击 “始终允许”，则仅将该设置应用于始终允许旧内容 （Flash Player 7 和更低版本）。该设置并不 “始终允许” Flash Player 8 内容。建议您在计算机上指定可同时与 Internet 和本地文件系统通信的 Flash 应用程序和目录。

创建面向 Flash 开发的配置文件

Flash 8 创作工具在您的硬盘上设置了一个标志，将您标识为开发人员，使您可以使用 “全局安全设置”面板面向开发人员的特定版本，而不是面向用户的 “全局安全设置”面板。此标志位于硬盘上的 FlashAuthor.cfg 文件中，该文件在安装 Flash Basic 8 和 Flash Professional 8 创作工具时自动安装。

FlashAuthor.cfg 文件所在目录如下：

Windows　boot disk\Documents and Settings\<UserName>\Application Data\Macromedia\Flash Player\#Security

Macintosh　/Users/<UserName>/Library/Preferences/Macromedia/Flash Player/#Security/

默认情况下，该文件设置为 `LocalSecurityPrompt=Author`，表示在没有安装创作工具时您在计算机上看到的警告将您视为 Flash 开发人员而非用户。

您可以以最终用户的身份测试本地应用程序，并且可以看到最终用户看到的警告对话框。为此，请在文本编辑器中打开 FlashAuthor.cfg，并将 FlashAuthor.cfg 中的 LocalSecurityPrompt 更改为如下所示：

```
LocalSecurityPrompt=User
```

对于设计或开发过程中的其他开发人员或在其本地硬盘上测试 Flash 应用程序但没有安装 Flash 8 创作工具的用户，您可能需要为他们提供将 `LocalSecurityPrompt` 设置为 `Author` 的 FlashAuthor.cfg 文件。通过本地部署内容有助于您模拟最终用户体验。

注释	如果删除了 FlashAuthor.cfg 文件，则启动 Flash 8 创作工具时将重新创建该文件。

在硬盘上的 #Security 目录中，您可以创建一个 FlashPlayerTrust 目录，其中可以存储唯一的配置文件。您可以在这些文件内指定硬盘上要信任的目录或应用程序。此目录不需要管理访问权限，因此用户不具有管理权限也可以设置 SWF 文件的权限并测试应用程序。

如果不指定一个目录，则您的内容可能无法达到预期效果。FlashPlayerTrust 目录中的配置文件包含目录路径。该文件可包含一些目录的列表，您可以向该文件添加新的路径。 Flash Player 要求配置文件中每行包含一个路径。将任何以 # 符号开头的行（之前没有前导空格）视为注释。

创建配置文件以信任某个目录：

1. 定位到硬盘上的 #Security 文件夹。
2. 在 #Security 文件夹内部创建一个名为 **FlashPlayerTrust** 的文件夹。
3. 使用文本编辑器在 FlashPlayerTrust 目录中创建一个新文件，并将其另存为 **myTrustFiles.cfg**。

   您可以对该配置文件使用任何唯一的名称。
4. 定位到测试 Flash 应用程序的目录。
5. 在文件中键入或粘贴每个目录路径 （硬盘上的任何目录路径），一行一个目录路径。您可以在不同行上粘贴多个目录路径。完成时，文件将看起来如下所示：

   ```
   C:\Documents and Settings\<yourname>\My Documents\files\
   C:\Documents and Settings\<yourname>\My Documents\testapps\
   ```
6. 保存对 myTrustFiles.cfg 的更改。
7. 测试从您添加到该文件的目录中访问本地和网络文件的文档。

   现在，此目录中保存的 Flash 应用程序就可以访问本地文件和网络了。

每个配置文件中可以保存大量目录路径，FlashPlayerTrust 目录中也可以保存大量 *.cfg 文件。

如果您要创建安装在最终用户硬盘上的应用程序，您可能需要在 FlashPlayerTrust 中创建一个配置文件，为该应用程序指定受信任目录。您可以在 FlashPlayerTrust 目录中创建配置文件，用于指定受信任应用程序的位置。有关此目录和创建配置文件的信息，请参见上面的过程。

注意	安装程序由具有管理员权限的用户在计算机上运行。

应开发一种唯一命名方案，避免与可能在此目录中安装文件的其它应用程序冲突。例如，您可能希望在文件名称中使用唯一公司名称和软件名称以避免冲突。

提示	如果不希望使用配置文件，则可以将您的 Flash 应用程序发布到单独的测试服务器，而不是向客户端或其他开发人员提供可在其本地硬盘上运行的 SWF 文件。

有关配置文件的更多信息，请参见 www.macromedia.com/go/flashauthorcfg。还可以创建一个唯一的配置文件，信任一个或多个目录。有关安全的详细信息，请参见 www.macromedia.com/devnet/security/ 和 www.macromedia.com/software/flashplayer/security/。

关于 sandboxType 属性

Flash Player 8 的 `System.security.sandboxType` 属性返回调用 SWF 文件在其中操作的安全性沙箱的类型。

`sandboxType` 属性具有以下 4 个值之一：

remote 该 SWF 文件承载在 Internet 上，操作时符合基于域的沙箱规则。

localTrusted 该 SWF 文件是已受用户信任（使用“全局安全设置管理器”或 FlashPlayerTrust 配置文件）的本地文件。该 SWF 文件可从本地数据源读取内容，并能与网络（如 Internet）通信。

localWithFile 该 SWF 文件是未受用户信任的本地文件，不使用网络名称发布。该 SWF 文件可从本地数据源读取内容，但不能与网络（如 Internet）通信。

localWithNetwork 该 SWF 文件是未受用户信任的本地文件，通过在“发布设置”对话框（Flash 选项卡）中选择的“只访问网络”发布。该 SWF 文件可与网络通信但不能从本地数据源读取内容。

尽管属性值仅在面向 Flash Player 8 发布的文件中返回，您仍可以检查任何 SWF 文件的 `sandboxType` 属性。也就是说面向 Flash Player 7 或更低版本发布时，您并不知道在运行时是否支持 `sandboxType` 属性。当 Flash Player 版本（由 `System.capabilities.version` 属性指示）低于 8 时，如果在运行时不支持该属性，则该值为 `undefined`。如果该值为 `undefined`，则您可以根据 SWF 文件的 URL 是否为本地文件来确定沙箱类型。如果 SWF 文件是本地文件，则 Flash Player 会将 SWF 分类为 `localTrusted`（Flash Player 8 之前的所有本地内容的处理方式）；否则 Flash Player 会将 SWF 文件分类为 `remote`。

关于 Local-with-file-system 限制

Local-with-file-system 文件尚未使用 FlashPlayerTrust 目录中的配置文件、“设置管理器”中的“全局安全设置”面板进行注册，或者尚未在 Flash 创作环境中的“发布设置”对话框中授予网络权限。

有关安全性沙箱的信息，请参见第 606 页的“理解本地安全性沙箱”。

这些文件包括在 Flash Player 8 中播放的旧内容。如果是在 Flash 8 中进行开发，或者您的内容属于以下类别之一，则您（或您的用户）应将该文件注册为受信任。有关将文件注册为受信任的信息，请参见第 613 页的“使用“设置管理器”指定受信任文件”。有关使用配置文件为本地文件播放授予权限的信息，请参见第 615 页的“创建面向 Flash 开发的配置文件”。

Local-with-file-system SWF 文件有以下限制：

- 不能访问网络，包括：
  - 从网络加载其它 SWF 文件 （除非 使用非 Internet UNC 路径）
  - 发送 HTTP 请求
  - 使用 XMLSocket、 Flash Remoting 或 NetConnection 进行连接
  - 调用 `getURL()` 除非 使用 `getURL("file:...")` 或 `getURL("mailto:...")`
- 可以与其它 Local-with-file-system 文件交互，但包括对以下内容的限制：
  - 跨脚本操作 （如 ActionScript 访问其它 SWF 文件中的对象）。
  - 调用 `System.security.allowDomain`
  - 将 `LocalConnection` 用作发送方或侦听器，并且不考虑 `LocalConnection.allowDomain` 处理函数。

  注意：Local-with-file-system SWF 文件可以与其它 Local-with-file-system、非网络 SWF 文件交互。但是不能与 local-with-network SWF 文件交互。

  Local-with-file-system SWF 文件对本地文件系统中的已知文件具有读访问权限。例如，只要您是从本地文件系统而不是从 Internet 加载，您就可以在 Local-with-file-system SWF 文件使用 `XML.load()`。
- Local-with-file-system SWF 文件不能与 HTML 页面通信，其中包括以下内容：
  - 传入脚本 （如 ExternalInterface API、 ActiveX、 LiveConnect 和 XPConnect）
  - 传出脚本 （如自定义 `fscommand` 调用和 `getURL("javascript:...")`）

  注意：HTML 页面为受信任时例外。

关于域、跨域安全性和 SWF 文件

默认情况下，Flash Player 7 和更高版本防止从一个域提供的 SWF 文件读取从另一个域提供的 SWF 文件的数据、对象或变量。另外，通过不安全的 （非 HTTPS）协议加载的内容不能读取通过安全的 (HTTPS) 协议加载的内容，即使两者都在完全相同的域中。例如，未经显式许可，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 文件也不能加载另一个域的数据 （例如使用 `loadVars()`）。

相同的数字 IP 地址是兼容的。但是，即使域名解析为相同的 IP 地址，该域名也与 IP 地址不兼容。

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 地址）

未经正确配置，Flash Player 8 不允许本地 SWF 文件与 Internet 通信。有关设置配置文件以在本地进行内容测试的信息，请参见第 615 页的 “创建面向 Flash 开发的配置文件”。

有关安全性的更多信息，请参见 www.macromedia.com/devnet/security/ 和 www.macromedia.com/software/flashplayer/security/。

有关更多信息，请参见以下主题：

- 第 619 页的 “设置和本地数据的域名规则”
- 第 620 页的 “SWF 文件之间的跨域和子域访问”
- 第 625 页的 “允许跨域数据加载”

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，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，在 here.xyz.com 承载的文件的设置和数据存储在 here.xyz.com 中，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。通过 `System.exactSettings` 可指定要使用的规则。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，默认值为 `false`，表示使用超域匹配规则。对于以 Flash Player 7 或 8 为目标播放器发布的文件，默认值为 `true`，表示使用完全域匹配规则。如果您使用设置或永久本地数据并想要在 Flash Player 7 或 8 中发布 Flash Player 6 SWF 文件，您可能需要在移植文件中将该值设置为 `false`。有关更多信息，请参见 《ActionScript 2.0 语言参考》中的 `exactSettings (System.exactSettings 属性)`。

SWF 文件之间的跨域和子域访问

在开发互相在线通信的一系列 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 SWF 文件，或者在一个超域的不同子域中承载 SWF 文件。

在以 Flash Player 5 或更早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，允许 someOtherSite.com 上的文件访问 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，正被访问的文件（即使该文件以早于 Flash Player 7 的 Flash Player 版本为目标播放器发布）必须显式允许跨域或子域访问；本部分将讨论这一主题。其次，在某一使用安全协议 (HTTPS) 的站点上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；在下一部分中将讨论这一主题（请参见第 629 页的“SWF 文件之间的 HTTP 到 HTTPS 协议访问”）。

通常会在应用程序中调用 System.security.allowDomain。然而，如果 LocalConnection 接收方是 HTTPS SWF 文件，而发送方不是，则调用 allowInsecureDomain。

以下问题仅影响以 Flash Player 7 为目标播放器发布的 SWF 文件。如果接收方是 HTTPS 而发送方是本地 SWF 文件，则调用 allowDomain()（即使应调用 allowInsecureDomain()）。但是，在 Flash Player 8 中，如果 HTTPS LocalConnection 接收方是 Flash Player 8，而发送方是本地文件，则调用 allowInsecureDomain()。

文件在 Flash Player 8 中的运行方式已经与其在 Flash Player 7 中的运行方式不同。只有正被访问的 SWF 文件就是调用 System.security.allowDomain 的文件时，调用 System.security.allowDomain 才允许进行跨脚本操作。也就是说，调用 System.security.allowDomain 的 SWF 文件现在仅允许访问它本身。在早期版本中，如果正被访问的 SWF 文件是与调用 System.security.allowDomain 的文件位于相同域的任何 SWF 文件，调用 System.security.allowDomain 就允许进行跨脚本操作这将打开调用 SWF 文件的整个域。

System.security.allowDomain("*") 和 System.security.allowInsecureDomain("*") 已增加对通配符 (*) 值的支持。通配符 (*) 值允许在访问文件是任何文件并且可从任何位置（如全局权限）加载时进行跨脚本操作。通配符权限非常有用，但必须符合 Flash Player 8 中的新本地文件安全性规则。尤其由于本地文件并不是来自一个域，因此必须使用通配符值。但是，由于任何域都有访问您的文件的权限，因此应慎用通配符值。有关更多信息，请参见 allowInsecureDomain（security.allowInsecureDomain 方法）。

从不同于调用某子级 SWF 文件的域中加载该子级 SWF 文件时，可能会遇到以下情况。您可能希望允许该文件编写父级 SWF 文件脚本，但又不知道子级 SWF 文件源自的最终域。例如，当您使用负载平衡重定向或第三方服务器时就可能发生这种情况。在这种情况下，您可以使用 `MovieClip._url` 属性作为此方法的参数。例如，如果您将 SWF 文件加载到 `my_mc` 中，则可以调用 `System.security.allowDomain(my_mc._url)`。若要这样做，必须等到 `my_mc` 中的 SWF 文件开始加载，因为 `_url` 属性还没有得到其最终正确值。若要确定子级 SWF 文件开始加载的时间，请使用 `MovieClipLoader.onLoadStart`。

相反的情况也会发生；即，您可能会创建一个子级 SWF 文件，希望允许其父级文件编写其脚本，但是又不知道其父级 SWF 文件要达到的域（也就是说，该 SWF 文件可能由多个域加载）。在这种情况下，可从该子级 SWF 中调用 `System.security.allowDomain(_parent._url)`。因为父级 SWF 文件是在加载子级文件前加载的，因此不必等到父级 SWF 文件开始加载。

注意 如果被访问的 Internet SWF 文件是从 HTTPS URL 加载的，则 Internet SWF 文件必须调用 `System.security.allowInsecureDomain("*")`。

下表总结了不同 Flash Player 版本中的域匹配规则：

以 Flash Player 为目标播放器发布的文件	SWF 文件之间的跨域访问（需要 allowDomain()）	SWF 文件之间的子域访问
第 5 版或更早版本	无限制	无限制
6	超域匹配：如果超域不匹配，则需要 `allowDomain()`。	无限制
7 及更高版本	完全域匹配 显式许可使用 HTTPS 承载的文件访问使用 HTTP 或 FTP 承载的文件	完全域匹配 显式许可使用 HTTPS 承载的文件访问使用 HTTP 或 FTP 承载的文件

注意 如果正在执行 HTTP 到 HTTPS 访问，即使具有完全域匹配，在 Flash Player 7 和更高版本中也需要 `System.security.allowInsecureDomain`。

控制 Flash Player 行为的版本是 SWF 文件的版本（SWF 文件的指定 Flash Player 版本），而不是 Flash Player 版本本身。例如，当 Flash Player 8 播放面向第 7 版的 SWF 文件时，Flash Player 将应用与第 7 版一致的行为。这样可确保播放器升级不会更改已部署 SWF 文件中的 `System.security.allowDomain()` 的行为。

因为 Flash Player 7 和更高版本实现完全域匹配规则，而不是超域匹配规则，所以，如果您要从以 Flash Player 7 或 8 为目标播放器发布的文件读取现有脚本，则可能需要修改这些脚本。（修改后的文件仍可以 Flash Player 6 为目标播放器进行发布）。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，则必须更改参数以指定完全域。下面的示例演示了可能需要对 Flash Player 6 代码做出的更改：

```
// 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");
}
// 允许以 Flash Player 7 或更高版本为目标播放器发布
// SWF 文件进行访问的相应命令
System.security.allowDomain("www.someSite.com", "store.someSite.com");
my_lc.allowDomain = function(sendingDomain) {
  return(sendingDomain=="www.someSite.com" ||
    sendingDomain=="store.someSite.com");
}
```

如果当前没有使用，则可能还需要向您的文件中添加这类语句。例如，如果您的 SWF 文件承载在 www.someSite.com 中并且要允许由 store.someSite.com 上以 Flash Player 7 或更高版本为目标播放器发布的 SWF 文件访问该文件，则必须将语句添加到 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），如下例所示：

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

此外还要考虑到，如果在 Flash Player 7 中运行的 Flash Player 6 应用程序试图访问其确切域以外的数据，就会强制实施 Flash Player 7 和更高版本的域匹配规则，并会提示用户允许或拒绝访问。

总之，如果您以 Flash Player 7 或更高版本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 `allowDomain` 语句：

- 您实现的跨 SWF 文件脚本（请参见第623页的“允许跨域 SWF 文件间的数据访问”）。
- 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 文件是 HTTPS，请参见第 629 页的 “SWF 文件之间的 HTTP 到 HTTPS 协议访问”。）
- 这些 SWF 文件不在同一域中（例如，一个文件位于 www.domain.com 中，而另一个文件位于 store.domain.com 中）。

您必须进行以下更改：

- 如果被调用的 SWF 文件是以 Flash Player 7 或更高版本为目标播放器发布的，则在被调用的 SWF 文件中包含 `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，如本部分前面的代码示例所示。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 7 中并添加 `System.security.allowDomain` 语句，并且使用完全域名匹配，如本部分前面的代码示例所示。（在 Flash Player 5 或更早版本中不支持 LocalConnection 对象）。

有关本地安全性沙箱的信息，请参见第 605 页的 “关于本地文件安全性和 Flash Player”。

允许跨域 SWF 文件间的数据访问

为了使两个 SWF 文件能够访问彼此的数据 （变量和对象），这两个文件必须源自同一个域。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使两个文件共享数据。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。

`System.security.allowDomain()` 允许位于指定域的 SWF 文件和 HTML 文件访问包含 `allowDomain()` 调用的 SWF 文件中的对象和变量。

如果两个 SWF 文件是同一个域提供，例如， http://mysite.com/movieA.swf 和 http://mysite.com/movieB.swf，则 movieA.swf 可以检查和修改 movieB.swf 中的变量、对象、属性、方法等，而且 movieB 也可以对 movieA 执行同样的操作。这被称为跨影片脚本编写，或跨脚本编写。

如果两个 SWF 文件是不同的域提供，例如 http://mysite.com/movieA.swf 和 http://othersite.com/movieB.swf，则在默认情况下， Flash Player 不允许 movieA.swf 编写 movieB.swf 的脚本，也不允许 movieB 编写 movieA 的脚本。通过调用 `System.security.allowDomain("mysite.com")`， movieB.swf 可以授予 movieA.swf 编写 movieB.swf 的脚本的权限。通过调用 `System.security.allowDomain()`，一个 SWF 文件可以赋予其它域中的 SWF 文件编写其脚本的权限。这称为跨域脚本编写。

有关 `System.security.allowDomain()`、跨脚本编写和跨域脚本编写的详细信息，请参见《ActionScript 2.0 语言参考》中的 allowDomain （security.allowDomain 方法）。

例如，假设 main.swf 是由 www.macromedia.com 提供。随后该 SWF 文件将来自 data.macromedia.com 的另一个 SWF 文件 (data.swf) 加载到使用 `createEmptyMovieClip()` 动态创建的影片剪辑实例中。

```
// 在 macromedia.swf 中
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc.loadMovie("http://data.macromedia.com/data.swf");
```

假设 data.swf 在其主时间轴上定义了一个名为 `getData()` 的方法。默认情况下，data.swf 文件完成加载后，main.swf 就不能调用在该文件中定义的 `getData()` 方法，因为这两个 SWF 文件不在相同的域中。例如，在 data.swf 完成加载之后，在 main.swf 中进行下面的方法调用将失败：

```
// 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` 处理函数和 `System.security.allowDomain()` 方法向由 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
this._lockroot = true;
System.security.allowDomain("www.macromedia.com");
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String):Boolean {
  return (sendingDomain == "www.macromedia.com");
};
function getData():Void {
  var timestamp:Date = new Date();
  output_txt.text += "data.swf:"+ timestamp.toString() + "\n\n";
}
output_txt.text = "**INIT**:\n\n";
```

现在，macromedia.swf 文件可以调用所加载的 SWF 文件中的 `getData` 函数。请注意，`allowDomain` 允许所允许域中的任何 SWF 文件编写允许该访问的域中的任何其它 SWF 文件的脚本，除非正在被访问的 SWF 文件位于使用安全协议 (HTTPS) 的站点上。

有关域名匹配的更多信息，请参见第 620 页的 “SWF 文件之间的跨域和子域访问”。

允许数据访问的服务器端策略文件

Flash 文档可通过使用以下某种数据加载调用方法从外部源加载数据：`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、资源或 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，请参见以下主题：

- 第 625 页的 “允许跨域数据加载”
- 第 627 页的 “关于自定义策略文件位置”
- 第 628 页的 “关于 XMLSocket 策略文件”

允许跨域数据加载

当 Flash 文档试图访问另一个域中的数据时，Flash Player 将自动试图从该域加载策略文件。如果试图访问数据的 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。

策略文件必须命名为 crossdomain.xml，并且可以驻留在服务器的根目录和其它目录之中，该服务器提供具有其它 ActionScript 的数据（请参见第 627 页的 “关于自定义策略文件位置”）。只有在通过 HTTP、HTTPS 或 FTP 进行通信的服务器上，策略文件才起作用。策略文件特定于它所驻留的服务器的端口和协议。

例如，位于 https://www.macromedia.com:8080/crossdomain.xml 的策略文件只适用于在端口 8080 通过 HTTPS 对 www.macromedia.com 进行的数据加载调用。

此规则的例外情况是，使用 XMLSocket 对象连接到另一个域中的套接字服务器。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，该属性指定一个确切的 IP 地址、一个确切的域或一个通配符域（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 SWF 文件授予访问权限。Flash Player 不执行 DNS 解析。

下面的示例显示一个策略文件，该策略文件允许从 foo.com 上的 Flash 文档访问来自 foo.com、www.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>
```

您也可以允许访问来自任何域的文档，如下面的示例所示：

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

每个 <allow-access-from> 标签还具有可选的 secure 属性。secure 属性默认为 true。如果您的策略文件在 HTTPS 服务器上，并且要允许 HTTP 服务器上的 SWF 文件从 HTTPS 服务器加载数据，则可以将此属性设置为 false。

将 secure 属性设置为 false 可能会危及 HTTPS 提供的安全性。

如果您正在下载来自 HTTPS 服务器的 SWF 文件，而加载它的 SWF 文件在 HTTP 服务器上，则您需要为 <allow-access-from> 标签添加 secure="false" 属性，如下面的代码所示：

```
<allow-access-from domain="www.foo.com" secure="false" />
```

不包含任何 <allow-access-from> 标签的策略文件相当于服务器上没有策略。

关于自定义策略文件位置

Flash Player 7 (7.0.19.0) 支持名为 System.security.loadPolicyFile 的方法。此方法使您可以在服务器上指定一个能够找到跨域策略文件的自定义位置，因此不必将策略文件放在根目录中。 Flash Player 7 (7.0.14.0) 只在服务器的根目录位置搜索策略文件，但将此文件放在根目录中可能会给站点管理员带来不便。有关 loadPolicyFile 方法和 XMLSocket 连接的更多信息，请参见第 628 页的 “关于 XMLSocket 策略文件”和 《ActionScript 2.0 语言参考》中的 loadPolicyFile (security.loadPolicyFile 方法)。

使用 loadPolicyFile 方法时，站点管理员可以将策略文件放在任何目录中，只要需要使用该策略文件的 SWF 文件调用 loadPolicyFile 来通知 Flash Player 该策略文件的位置。但是，未放在根目录中的策略文件的范围很有限。策略文件只允许访问服务器层次结构中与其同层或低于该层的位置。

loadPolicyFile 方法只在 Flash Player 7 (7.0.19.0) 或更高版本中可用。使用 loadPolicyFile 方法的 SWF 文件的作者必须执行以下操作之一：

- 需要 Flash Player 7 (7.0.19.0) 或更高版本。
- 安排数据源站点，以便将策略文件放在站点默认位置（根目录）及非默认位置中。Flash Player 的早期版本使用默认位置。

否则，作者必须创建 SWF 文件，这样跨域加载操作失败才可能实现。

小心

如果您的 SWF 文件依赖于 loadPolicyFile，使用 Flash Player 6 或更低版本或 Flash Player 7 (7.0.19.0) 或更高版本则不会出现问题。但是，使用 Flash Player 7 (7.0.14.0) 的访问者不支持 loadPolicyFile。

如果要在服务器上的自定义位置使用策略文件，您必须首先 调用 System.security.loadPolicyFile，然后根据策略文件发出请求，如下所示：

```
System.security.loadPolicyFile("http://www.foo.com/folder1/folder2/
  crossdomain.xml");
var my_xml:XML = new XML();
my_xml.load("http://www.foo.com/folder1/folder2/myData.xml");
```

您可以使用 loadPolicyFile 加载多个具有重叠范围的策略文件。对于所有的请求， Flash Player 试图查询其范围包括所请求位置的所有文件。如果一个策略文件未能授予跨域访问权限，则不会阻止另一个文件授予访问数据的权限。如果所有的访问尝试均失败，则 Flash Player 会查找 crossdomain.xml 文件的默认位置 （在根目录中） 。如果未能在默认位置找到任何策略文件，则请求失败。

关于 XMLSocket 策略文件

对于 XMLSocket 连接试图而言，Flash Player 7 (7.0.14.0) 会在试图连接的子域内的 HTTP 服务器（端口 80）上查找 crossdomain.xml。Flash Player 7 (7.0.14.0) 以及所有早期版本将 XMLSocket 连接限制在端口 1024 和更高的端口。但是，在 Flash Player 7 (7.0.19.0) 和更高版本中，ActionScript 可以使用 `System.security.loadPolicyFile` 将策略文件的非默认位置通知 Flash Player。XMLSocket 策略文件的任何自定义位置都必须仍在 XML 套接字服务器上。

在下面的示例中，Flash Player 从指定的 URL 检索策略文件：

```
System.security.loadPolicyFile("http://www.foo.com/folder/policy.xml");
```

由该位置的策略文件授予的所有权限均适用于服务器层次结构中与该位置同层或低于该层的所有内容。因此，如果试图加载下面的数据，您将发现只能加载某些位置的数据：

```
myLoadVars.load("http://foo.com/sub/dir/vars.txt"); // 允许
myLoadVars.load("http://foo.com/sub/dir/deep/vars2.txt"); // 允许
myLoadVars.load("http://foo.com/elsewhere/vars3.txt"); // 不允许
```

要解决此问题，可以使用 `loadPolicyFile` 向一个 SWF 文件加载多个策略文件。Flash Player 总是等所有策略文件下载完毕后，才会拒绝需要策略文件的请求。如果 SWF 中没有其它授权的策略，则 Flash Player 将查询 crossdomain.xml 的默认位置。

特殊语法允许直接从 XMLSocket 服务器检索策略文件：

```
System.security.loadPolicyFile("xmlsocket://foo.com:414");
```

在本例中，Flash Player 试图从指定的主机和端口检索策略文件。如果策略文件不在默认（根）目录中，可以使用任意端口；否则只能使用端口 1024 和更高的端口 （与早期版本的播放器相同）。建立与指定端口的连接后，Flash Player 将发送 `<policy-file-request />`，以空字节结束。

可以配置 XML 套接字服务器，使其通过以下方式提供策略文件：

- 在同一端口上提供策略文件和常规套接字连接。在传送策略文件之前，服务器应等待 `<policy-file-request />`。
- 通过常规连接在不同端口上提供策略文件，在这种情况下，可以在专用策略文件端口上建立了连接之后立刻发送策略文件。

服务器必须在关闭连接前发送一个空字节以终止策略文件。如果服务器不关闭连接，则 Flash Player 将在接收到结束空字节时关闭连接。

XML 套接字服务器提供的策略文件具有与其它策略文件相同的语法，只是它还必须指定授予访问权限的端口。允许的端口在 <allow-access-from> 标签中的 to-ports 属性中指定。如果策略文件的端口低于 1024，则它可以给任何端口授予访问权限；如果策略文件来自端口 1024 或更高端口，则它只能给高于 1024 的其它端口授予访问权限。允许使用单端口号、端口范围和通配符。下面的代码是一个 XMLSocket 策略文件的示例：

```
<cross-domain-policy>
<allow-access-from domain="*" to-ports="507" />
<allow-access-from domain="*.foo.com" to-ports="507,516" />
<allow-access-from domain="*.bar.com" to-ports="516-523" />
<allow-access-from domain="www.foo.com" to-ports="507,516-523" />
<allow-access-from domain="www.bar.com" to-ports="*" />
</cross-domain-policy>
```

因为连接到低于 1024 的端口的能力是 Flash Player 7 (7.0.19.0) 和更高版本中的新功能，因此即使 SWF 文件连接到其子域时，也总是需要使用通过 loadPolicyFile 加载的策略文件来进行授权。

SWF 文件之间的 HTTP 到 HTTPS 协议访问

您必须使用 allowDomain 处理函数或方法允许某个域中的 SWF 文件由另一个域中的 SWF 文件访问。但是，如果正被访问的 SWF 文件承载在使用安全协议 (HTTPS) 的站点上，则 allowDomain 处理函数或方法将不允许承载在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.allowInsecure Domain() 或 System.security.allowInsecureDomain() 语句。有关更多信息，请参见第 629 页的“允许 SWF 文件之间的 HTTP 到 HTTPS 协议访问”。

允许 SWF 文件之间的 HTTP 到 HTTPS 协议访问

除了完全域匹配规则外，您必须显式允许使用不安全协议的站点上承载的文件可以访问使用安全协议 (HTTPS) 的站点上承载的文件。根据被调用的文件是以 Flash Player 6、7 还是 8 为目标播放器发布的，您必须实现一个 allowDomain 语句（请参见第 620 页的“SWF 文件之间的跨域和子域访问”），或者使用 LocalConnection.allowInsecure Domain 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 文件进行访问，则添加到 data.swf 中的下面的代码将允许此访问：

```
// 在 data.swf 中
System.security.allowInsecureDomain("www.someSite.com");
my_lc.allowInsecureDomain = function(sendingDomain:String):Boolean {
```

```
  return (sendingDomain == "www.someSite.com");
};
```

警告 实现 `allowInsecureDomain()` 语句将会危及 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。

下面的代码显示您可能要进行更改的示例：

```
// 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");
}
```

如果您的文件当前没有使用这些语句，可能还必须向这些文件中添加这类语句。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 或更高版本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 文件脚本编写（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，如本部分前面的代码示例所示。

- 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` 语句，并且使用完全域名匹配，如本部分前面的代码示例所示。
- 如果被调用文件是以 Flash Player 6 为目标播放器发布的，并且您要将该被调用文件移植到 Flash Player 7 中，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，如本部分前面的代码示例所示。
- 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，如本部分前面的代码示例所示；也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，如本部分前面的代码示例所示。

第 18 章

18 调试应用程序

Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 提供了多种在 SWF 文件中测试 ActionScript 的工具。当在 Flash 调试播放器中运行 SWF 文件时，调试器能够帮助您找到其中包含的错误（请参见第633页的“调试脚本”）。Flash 还提供以下这些调试工具：

- “输出”面板，可显示错误消息（包括某些运行时错误）以及变量和对象列表（请参见第 645 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 649 页的“使用 trace 语句”）
- `throw` 和 `try..catch..finally` 语句，使您可以测试和响应脚本中的运行时错误。

本部分描述如何使用调试器调试脚本和 Flash 应用程序，以及如何使用“输出”面板。有关更多信息，请参见以下主题：

调试脚本

在 Flash Player 中运行 SWF 文件时，Flash 8 中的调试器可帮助您发现其中的错误。您必须在 Flash Player 的被称作 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，当您安装 Flash 并浏览含有 Flash 内容的 Web 站点时，或者使用“测试影片”选项时，实际使用的正是 Flash 调试播放器。您还可以在 Windows 或 Macintosh 的以下目录中运行安装程序：Flash install directory\Players\Debug\ 目录或从同一目录下启动独立的 Flash 调试播放器。

在您使用 “控制” > “测试影片”命令测试执行键盘控制 （Tab 键切换或使用 `Key.addListener()` 创建的快捷键等）的 SWF 文件时，请选择 “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

在英文系统上使用非英文应用程序时，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，“测试影片”命令都将失败。例如，在英文系统上使用的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 ActionScript 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。调试器让您在 ActionScript 中设置断点，断点会在运行时暂停 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。将影片剪辑添加到文件中或从文件中删除时，显示列表会立即反映出这些更改。通过移动水平拆分器，可以重新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”。

  该命令将导出带有调试信息的 SWF 文件 （SWD 文件）并启用对 SWF 文件的调试。它打开调试器，然后在测试模式下打开 SWF 文件。

附加

如有必要，可以调整 “调试器” 面板各个区域的大小。当指针在各个区域的交界处形状发生变化时，便可以拖动更改 “显示” 列表、“监视点” 列表和代码视图的大小。

有关更多信息，请参见以下主题：

- “从远程位置调试 SWF 文件”
- 第 638 页的 “显示和修改变量”
- 第 639 页的 “使用 “监视点” 列表”
- 第 640 页的 “显示影片剪辑属性和更改可编辑属性”
- 第 641 页的 “设置和删除断点”
- 第 643 页的 “关于跟踪代码行”

从远程位置调试 SWF 文件

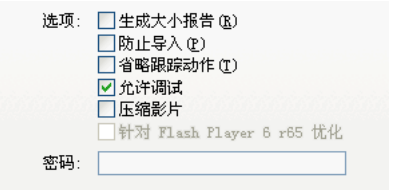
通过使用 Flash Player 的独立版本、 ActiveX 版本或者插件版本，可以调试远程 SWF 文件。若要查找 Flash Player 的这些版本，请在 Windows 或 Macintosh 的以下目录中查找：Flash install directory\Players\Debug\。

在导出 SWF 文件时，可以在文件中启用调试，并创建调试密码。如果您没有启用调试，则不会激活调试器。

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，可以使用调试密码发布文件。在 JavaScript 或 HTML 中时，用户可以在 ActionScript 中查看客户端的变量。要安全地存储变量，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。

启用 SWF 文件的远程调试：

1. 选择 “文件” > “发布设置”。
2. 在 “发布设置”对话框中的 “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” > “导出影片”
   - “文件” > “发布”

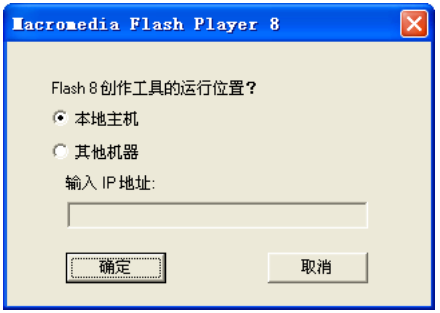
   Flash 创建以 .swd 为扩展名的调试文件，并把它与 SWF 文件一起保存在同一个目录中。 SWD 文件用于调试 ActionScript，并包含允许您使用断点和跟踪代码的信息。
5. 把 SWD 文件放到服务器上与 SWF 文件相同的目录中。

   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信息，这样您就不能跟踪代码。
6. 在 Flash 中，选择 “窗口” > “调试器”。
7. 在调试器中，从弹出菜单 （位于面板的右上角）选择 “启用远程调试”。

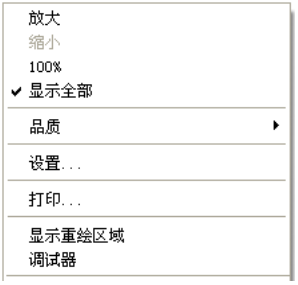
从远程位置激活调试器：

1. 打开 Flash 创作应用程序。
2. 在浏览器中或在独立播放器的调试版本中，从远程位置打开已发布的 SWF 文件。

   “远程调试”对话框就会出现。



> **注意** 如果没有出现 “远程调试”对话框， Flash 就不能查找 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，就会出现一个密码提示。

   如果设置了调试密码，请输入密码。

   在调试器中会出现 SWF 文件的显示列表。如果没有播放 SWF 文件，调试器可能已暂停，此时单击 “继续”可以启动调试器。

显示和修改变量

调试器中的“变量”选项卡显示在显示列表中选择的 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。

调试器中的“本地变量”选项卡会显示 ActionScript 特定行（即在断点处或在用户定义函数内任何位置停止 SWF 文件时的当前行）中所有可用的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。

   注释 如有必要，可以调整“调试器”面板各个区域的大小。当指针在每个区域交界处发生形状变化时，便可以拖动以更改“显示”列表、“监视点”列表和代码视图的大小。

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 639 页的“使用“监视点”列表”），则该变量将从“变量”选项卡中删除，但在“监视点”选项卡中仍然可以查看它。

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`）。

注释 若要在测试模式下将表达式的值写入“输出”面板上，请使用 `trace` 语句。请参见第 649 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，使之显示在 “监视点”列表中。“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，您还可以在 “监视点”列表中输入新的变量值。“监视点”列表只能显示使用绝对目标路径（如 `_global` 或 `_root`）可以访问的变量和属性。

如果您将某个本地变量添加到 “监视点”列表中，只有当 Flash Player 停止在该变量所在范围的 ActionScript 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。

提示 如果有必要，可以调整 “调试器”面板各个区域的大小。当指针在各个区域的交界处形状发生变化时，便可以拖动更改 “显示”列表、“监视点”列表和代码视图的大小。

“监视点”列表只能显示变量，不能显示属性或函数。

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” 。在名称列中双击，并在该字段中输入变量名的目标路径。

从 “监视点”列表中删除变量：

- 在 “监视点”选项卡或 “变量”选项卡上右键单击 (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意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。

提示

如有必要，可以调整“调试器”面板各个区域的大小。当指针在各个区域的交界处形状发生变化时，便可以拖动更改“显示”列表、“监视点”列表和代码视图的大小。

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]`）。

注意	若要在测试模式下将表达式的值写入“输出”面板上，请使用 `trace` 语句。请参见第 649 页的“使用 trace 语句”。

设置和删除断点

断点可以让您在 ActionScript 的特定行终止正在 Flash 调试播放器中运行的 Flash 应用程序。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检查这些语句。

您可以在“动作”面板、“脚本”窗口或者调试器中设置断点。在“动作”面板中设置的断点会保存在 FLA 文件中。在调试器和“脚本”窗口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。

小心	如果在“动作”面板或“脚本”窗口中设置断点并按下“自动套用格式”按钮，则可能发现有些断点已经不在正确的位置上了。在您设置代码格式时，ActionScript 可能会被移动到另一行，因为有时空行会被删除。单击“自动套用格式”后可能有必要检查和修改断点，或是在设置断点之前对您的脚本使用自动套用格式。

若要在调试会话期间在“动作”面板或“脚本”窗口中设置或删除断点，请执行以下操作之一：

- 在左页边空白处单击。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“设置断点”、“删除断点”或者“删除此文件中的断点”。

注意	在“脚本”窗口中，您也可以选择“删除所有 AS 文件中的断点”。

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

注意	在某些 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 Ctrl 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：

- 在左页边空白处单击。红点指示断点。
- 单击代码视图上方的 “切换断点”或 “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 “设置断点”、“删除断点”或者 “删除此文件中的断点”。
- 按下 Ctrl+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。（请参见第 643 页的 “关于跟踪代码行”。）

如果调试器中的 ActionScript 文件与 “脚本”窗口中打开的 ActionScript 文件具有相同的路径，则可以在 “脚本”窗口中设置断点，并在调试器中显示它们。同样，如果在 “脚本”窗口中打开 ActionScript 文件，则可以在调试会话期间在调试器中设置断点，并使断点出现在 ActionScript 文件中。

注意	不要在注释行或者空行上设置断点；如果将断点设置在注释行或者空行上，这些断点将被忽略。

关于断点 XML 文件

在 “脚本”窗口中处理外部脚本文件中的断点时，AsBreakpoints.xml 文件可帮助您存储断点信息。AsBreakpoints.xml 文件写入以下位置处的 Local Setting 目录中：

Windows：

Hard Disk\Documents and Settings\User\Local Settings\Application Data\Macromedia\Flash 8\language\Configuration\Debugger\

Macintosh：

Macintosh HD/Users/User/Library/Application Support/Macromedia Flash 8/Configuration/Debugger/

以下是 AsBreakpoints.xml 文件的一个示例：

```
<?xml version="1.0"?>
<flash_breakpoints version="1.0">
  <file name="c:\tmp\myscript.as">
    <breakpoint line="10"></breakpoint>
    <breakpoint line="8"></breakpoint>
    <breakpoint line="6"></breakpoint>
  </file>
  <file name="c:\tmp\myotherscript.as">
    <breakpoint line="11"></breakpoint>
    <breakpoint line="7"></breakpoint>
    <breakpoint line="4"></breakpoint>
  </file>
</flash_breakpoints>
```

XML 文件由下列标签组成：

flash_breakpoints 此节点具有一个名为 `version` 的属性，该属性表示 XML 文件的版本。Flash 8 的版本为 1.0。

file 是 `flash_breakpoints` 的子节点。此节点有一个名为 `name` 属性，该属性表示包含断点文件的名称。

breakpoint 是 `file` 的子节点。此节点有一个名为 `line` 的属性，该属性表示断点所在的行号。

Flash 在启动时会读取 AsBreakpoints.xml 文件，而在关闭时会再次生成该文件。AsBreakpoints.xml 用于跟踪 Flash 开发会话间的断点。当您在 Flash 中进行开发，设置和删除断点时，将由专门的内部数据结构来维护断点。

关于跟踪代码行

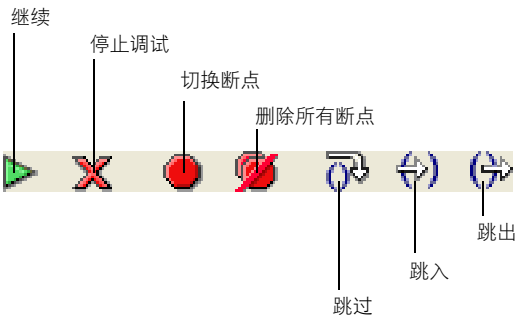
当开始调试会话时，Flash Player 会暂停，以便切换断点。如果在“动作”面板中设置断点，可以单击“继续”按钮播放 SWF 文件，直到到达断点。如果没有在“动作”面板中设置断点，可以使用调试器中的跳转菜单来选择 SWF 文件中的任意脚本。一旦选择了一个脚本，就可以将断点添加上去。

添加断点之后，必须单击“继续”按钮来启动 SWF 文件。到达断点时，调试器就会停止。例如，在下面的代码中，假定在 `myFunction()` 行上的一个按钮内设置了断点：

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

当单击按钮时，到达断点并且 Flash Player 会暂停。现在，无论 `myFunction()` 函数被定义在文档中的什么位置，您都可以将调试器移到该函数的第一行。还可以继续跟踪或者退出函数。

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。

在下面的示例中，如果在第 7 行放置一个断点，并单击“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;
```

注解	此代码片断中的数字表示行号。这些行号并非代码的一部分。

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；它将黄色箭头移到调用该函数的代码行后面一行。在上面的示例中，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中的下一行。在上面的例子中，如果在第 7 行停止，然后单击“跳过”，则会直接进入第 8 行，而不会跟踪 `myFunction()`，尽管 `myFunction()` 代码仍然会执行。

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除 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 646 页的 “列出 SWF 文件的对象”和第 647 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 649 页的 “使用 trace 语句”。）

若要显示或隐藏 “输出” 面板，请执行以下操作之一：

- 选择 “窗口” > “输出”。
- 按下 F2 键。

若要处理 “输出” 面板上的内容，请单击右上角的弹出菜单来查看选项。

下表列出了 “输出” 面板弹出菜单中的可用选项：

菜单项	功能
自动换行	确定长行是否自动换行，以便用户不必使用水平滚动条就可查看整行字符。如果选中，则换行；否则，不换行。
复制	将 “输出” 面板的全部内容复制到计算机的剪贴板上。若要复制所选的输出部分，请选择需要复制的区域，然后选择 “复制”。
清除	清除当前 “输出” 面板中的所有输出。
查找	打开一个对话框，可将其用于查找 “输出” 面板内容中的关键字或短语。
再次查找	尝试在 “输出” 面板内容中查找关键字或短语的下一个实例。
保存到文件	将 “输出” 面板中的当前内容保存到外部文本文件中。

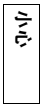
菜单项	功能
打印	显示 “打印”对话框，这使您能够将 “输出”面板的当前内容打印到已安装的打印机或程序 （如 Flash Paper 或 Acrobat）中。
过滤器级别	允许您选择两种可能的输出级别：“无”或 “太多”。选择 “无”则不会将所有输出发送到浏览器。
最大化面板	在停靠时最大化 “输出”面板。
关闭面板	关闭 “输出”面板并清除面板内容。

有关 “输出”面板的更多信息，请参见以下主题：

- 第 646 页的 “列出 SWF 文件的对象”
- 第 647 页的 “列出 SWF 文件的变量”
- 第 648 页的 “关于显示文本字段属性以进行调试”
- 第 649 页的 “使用 trace 语句”
- 第 649 页的 “为测试更新 Flash Player”

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。此选项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。



选择 “对象列表”命令将清除当前出现在 “输出”面板中的所有信息。如果您不想丢失 “输出”面板中的信息，请从 “输出”面板 “选项”弹出菜单中选择 “保存到文件”，或在选择 “对象列表”命令之前将信息复制并粘贴到另一个位置。

“对象列表”命令不会列出所有的 ActionScript 数据对象。在此上下文中，对象被看成舞台上的形状或者元件。

显示 SWF 文件中的对象列表：

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” > “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，如下面的示例所示：

```
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。此列表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。全局变量将出现在 “全局变量”部分中的 “变量列表”输出的顶部，每个变量都有一个 `_global` 前缀。

此外，“变量列表”命令还显示 getter/setter 属性（这是用 `Object.addProperty()` 方法创建并启动 `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。为使这些属性更易于与其它变量区别，getter/setter 属性的值以 `[getter/setter]` 字符串为前缀。为 getter/setter 属性显示的值是通过计算该属性的 `get` 函数确定的。

小心

选择 “变量列表”命令将清除出现在 “输出”面板中的所有信息。如果您不想丢失 “输出”面板中的信息，请从 “输出”面板 “选项”弹出菜单中选择 “保存到文件”，或在选择 “变量列表”命令之前将信息复制并粘贴到另一个位置。

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，如下面的示例所示：

```
Global Variables:
  Variable _global.mycolor = "lime_green"
Level #0:
Variable _level0.$version = "WIN 7,0,19,0"
Variable _level0.myArray = [object #1, class 'Array'] [
    0:"socks",
    1:"gophers",
    2:"mr.claw"
  ]
Movie Clip: Target="_level0.my_mc"
```

关于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，您可以在测试模式中使用 “调试” > “变量列表”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，则不会显示属性。
- 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，则将它们按照此列表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（例如，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`。

“调试菜单”（测试模式期间）中的 “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"
```

有关“变量列表”或“对象列表”命令的更多信息，请参见第 645 页的“使用“输出”面板”。

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 SWF 文件或场景时，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，表达式的值将出现在 “输出” 面板中，如下面的代码片断和 “输出” 面板的图像所示：

若要在脚本中使用 trace 语句，请执行以下操作：

1. 在时间轴中选择第 1 帧，在 “动作” 面板中，添加下面的代码：

```
this.createEmptyMovieClip("img_mc", 10);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace(target_mc+" loaded in "+getTimer()+" ms");
};
mclListener.onLoadError = function(target_mc:MovieClip, errorCode:String,
  httpStatus:Number) {
  trace(">> error downloading image into "+target_mc);
  trace(">>\t errorCode="+errorCode+", httpStatus="+httpStatus);
};
var img_mcl:MovieClipLoader = new MovieClipLoader();
img_mcl.addListener(mclListener);
img_mcl.loadClip("http://www.helpexamples.com/flash/images/404.jpg",
  img_mc);
```

2. 选择 “控制” > “测试影片” 以测试 SWF 文件。

   “输出” 面板将显示以下消息：

为测试更新 Flash Player

您可以从 Macromedia 支持中心 (http://www.macromedia.com/go/flash_support_cn) 下载最新的 Flash Player 版本，并用它来测试 SWF 文件。

第 19 章

ActionScript 2.0 的最佳做法和编码约定

19

Macromedia Flash 的设计人员和开发人员必须以一种直观的方法编写代码和构建应用程序，以便他们自己以及共同从事同一项目的其他人能够从中受益。这对具有大量资源的 FLA 文件或长代码文件尤其重要。在遵循最佳做法和代码约定的情况下，设计和开发小组的每个成员都能了解文件结构和 ActionScript 代码，可以更加有效地工作。本文档旨在帮助您将 Flash 开发和编码过程正规化。

因为由多个设计人员或开发人员共同开发一个 Flash 项目的情况很常见，所以当每个人都按照同一套标准准则来使用 Flash、组织 FLA 文件及编写 ActionScript 2.0 代码时，就能使整个项目小组受益。本章各部分概述了编写 ActionScript 的最佳做法，《使用 Flash》文档的一些部分介绍了使用 Flash 创作工具时的最佳做法。

下列准则可以促使那些学习如何使用 Flash 及编写 ActionScript 代码的人员保持一致性。不管是设计人员还是开发人员，也不论是独自工作还是作为项目小组成员工作，都应始终采用这些最佳做法。

- 在处理 Flash 或 ActionScript 文档时

  采用一致有效的做法有助于加快您的工作流程。使用已确定的编码约定能加快开发速度，在希望进一步编辑文档时也能更容易了解和记住该文档是如何构建而成的。另外，在较大项目的框架内，代码往往更易于移植和重复使用。

- 在共享 FLA 或 AS 文件时

  编辑文档的其他人员可快速地查找和了解 ActionScript，对代码进行一致的改动，查找和编辑资源。

- 在处理应用程序时

  多个作者可以共同处理一个应用程序，并且冲突更少、效率更高。如果遵循最佳做法和编码约定，则可使项目或站点管理员管理和构建复杂项目或应用程序时的冲突或冗余更少。

- 在学习或教授 Flash 和 ActionScript 时

  学习如何通过最佳做法和遵循编码约定来构建应用程序可减少再学习特殊方法的需要。如果学习 Flash 的学生使用一致而且更佳的方法来构建代码，则他们可以更快且更顺利地学习这门语言。

一致的技巧和下列准则对学习 Flash 或在团队环境中有效工作的人员会有帮助。使用一致的方法有助于在独自工作时想起文档是如何构建而成的，特别是在您最近一直没有处理过 FLA 文件的情况下。

这些仅是学习并遵循最佳做法的理由中的少数几个而已。当您阅读了这些最佳做法并形成自己的良好习惯时，您一定会发现许多其它理由。使用 Flash 时，请考虑将以下各个主题作为准则；您可以选择遵循其中的某些或所有建议。您也可以修改建议以适合您的工作方式。本章中的许多准则有助于形成一种用一致的方式来使用 Flash 和编写 ActionScript 代码的习惯。

本章介绍以下有关编码约定和最佳做法的主题：

命名约定

通常，80% 的开发时间要花在进行调试、排查问题和执行常规维护，对于大项目而言更是如此。即使是处理小项目时，也要花大量时间进行分析和修复代码。代码的可读性对于您自己和项目小组成员而言都很重要。如果遵循命名约定，您就会提高可读性，它可以加快工作流程且能让您找到并修复代码中的任何错误。所有编程人员都遵循一种标准化的编写代码方式；这样做从多个方面改善了项目。

将命名约定用于变量名称可提供下列重要功能：

- 这些约定使代码易于阅读，以便您可以立即识别出变量的数据类型。这样做可以帮助学生（学习代码的人）和不熟悉您的代码的开发人员。
- 必要时便于搜索和替换。
- 有助于减少保留字和语言构造之间的冲突。
- 有助于区分不同作用范围的变量（局部变量、类属性、参数等等）。

以下各部分包含编写 ActionScript 代码的命名准则，如命名文件、变量、常数、组件，等等。第 679 页的“设置 ActionScript 语法的格式”讨论一些特定于 ActionScript 的，以及一些对其它编程语言通用的格式设置约定。第 663 页的“ActionScript 编码约定”讨论特定于 ActionScript 编码和使用 Flash 8 开发的编码约定。

注意

Flash Player 7 和 8 并没有非常严格地遵循 ECMAScript (ECMA-262) 第 3 版语言规范。如想了解语言的工作原理，参见此规范会很有用。（请参见 www.ecma-international.org/publications/standards/Ecma-262.htm。）

本部分包含以下主题：

常规命名准则

本节回顾编写 ActionScript 代码的命名准则。对于编写逻辑代码而言，命名约定非常重要。主要目的是提高 ActionScript 2.0 代码的可读性。请记住，所有变量名称必须是唯一的。在 Flash Player 7 和更高版本中，名称是区分大小写的。不应该使用具有不同大小写的相同名称，因为这样可能扰乱程序员对代码的阅读，也可能给未强制区分大小写的早期版本的 Flash 带来问题。在 Flash 中命名变量、文件和类等项时要牢记以下这些准则：

- 限制使用缩写词。

  使用一致的缩写词。缩写词必须清楚地仅表示一种含义。例如，缩写词 “sec” 可能表示 “section （部分）” 和 “second （秒）”。
- 连接单词以创建名称。

  在连接单词时请使用混合大小写以便区分每个单词，增强可读性。例如，选择使用 `myPelican` 而不要使用 `mypelican`。
- 按描述过程或描述项的方式命名文件，如 `addUser`。
- 不要为方法或变量使用非描述性的名称。

  例如，如果检索访问者的用户名数据，则可以使用 `getUserName()` 方法，而不要使用意义不明确的 `getData()` 方法。此示例表示要发生什么，而不是实现的方法。
- 尽可能保持所有名称最短。

  切记，一定要保证名称具有描述性。

下面几节提供了关于在代码中命名变量、类、包和常数等项的更详细的信息。

避免使用保留字或语言构造

命名实例和变量时，应避免使用保留字，保留字可引起代码错误。保留字包括 ActionScript 语言中的关键字。

此外，不要用 ActionScript 2.0 语言中的任何字（称为语言构造）作为实例或变量名称。ActionScript 构造包括类名称，组件类名称，方法和属性名称和接口名称。

警告

永远不要使用不同的大小写来避免与保留字发生冲突。例如，将 TextField 类的一个实例命名为 `textfield`（它不会与 TextField 发生冲突，因为 Flash 是区分大小写的）就是一种欠佳的编码做法。

下表列出了 ActionScript 2.0 中的保留关键字，这些保留关键字在用作变量名称时会在脚本中引起错误：

add	and	break	case
catch	class	continue	default
delete	do	dynamic	else
eq	extends	false	finally
for	function	ge	get
gt	if	ifFrameLoaded	implements
import	in	instanceof	interface
intrinsic	le	it	ne
new	not	null	on
onClipEvent	or	private	public
return	set	static	super
switch	tellTarget	this	throw
try	typeof	var	void
while	with		

根据 ECMAScript (ECMA-262) 第 4 版语言规范草案，在 Flash 中将下列字保留起来以供将来使用。请避免使用这些字，因为在未来版本的 Flash 中可能会使用它们。

as	abstract	Boolean	bytes
char	const	debugger	double
enum	export	final	float
goto	is	long	namespace
native	package	protected	short
synchronized	throws	transient	use
volatile			

命名变量

变量名称仅能包含字母、数字和美元符号 ($)。变量名称不能以数字开头。变量必须是唯一的，且在 Flash Player 7 和更高版本中要求区分大小写。例如，避免使用下列变量名称：

```
my/warthog = true;   // 包括一个斜杠
my warthogs = true;  // 包括一个空格
my.warthogs = true; // 包括一个点
5warthogs = 55;     // 以一个数字开头
```

请尽可能对变量使用严格数据类型指定，因为这样做会在以下几个方面帮助您：

- 添加代码完成功能，该功能可加快编码速度。
- 由于在 “输出”面板中生成错误，因此在编译 SWF 文件时不会出现 “无提示故障” (silent failure) 的情况。这些错误可帮助您查找和修复应用程序中的问题。

若要对变量添加数据类型，则必须使用 `var` 关键字定义变量。在下面的示例中，创建 LoadVars 对象时应使用严格数据类型指定：

```
var paramsLv:LoadVars = new LoadVars();
```

严格数据类型指定提供代码完成功能，并确保 `paramsLv` 的值包含 LoadVars 对象。严格数据类型指定还可确保 LoadVars 对象不会用于存储数字数据或字符串数据。因为严格类型指定依赖于 `var` 关键字，所以无法向对象或数组中的全局变量或属性添加严格数据类型指定。有关对变量进行严格类型指定的更多信息，请参见第 284 页的 “关于指定数据类型和严格数据类型指定”。

提示 严格数据类型指定不会减慢 SWF 文件的速度。在编译时 （创建 SWF 文件时），而不是在运行时进行类型检查。

当命名代码中的变量时请遵循以下准则：

- 所有变量名称必须是唯一的。
- 不要使用大小写不同的相同变量名称。

  例如，不要在应用程序中将 `firstname` 和 `firstName` 用于两个不同的变量。尽管在 Flash Player 7 和更高版本中名称是区分大小写的，但使用具有不同大小写的相同变量名称可能扰乱程序员对代码的阅读，也可能给未强制区分大小写的早期版本的 Flash 带来问题。
- 不要将 ActionScript 1.0 或 2.0 中的部分字用作变量名称。

  特别是不能将关键字用作实例名称，因为它们可能会引起代码错误。不要依赖区分大小写来避免冲突并使代码正常运行。
- 不要使用作为公共编程构造组成部分的变量。

  如果知道某些字是其它编程语言中的语言构造，则即使 Flash 不包括或不支持这些语言构造，也不要使用这些语言构造。例如，不要将下列关键字用作变量：

  ```
  textfield = "myTextField";
  switch = true;
  new = "funk";
  ```
- 始终在代码中添加数据类型注释。

  向变量添加类型注释也称为 “对变量使用严格数据类型” 或 “对变量执行严格的类型检查”，它非常重要，可以：

  - 在编译时生成错误，因此应用程序不会在无提示的情况下失败。
  - 触发代码完成。
  - 帮助用户了解您的代码。

  有关添加类型注释的信息，请参见第284页的 “关于指定数据类型和严格数据类型指定”。
- 不要过多使用 Object 类型。

  数据类型注释应力求精确，以提高性能。只有在没有适当的备选数据类型时，才使用 Object 类型。
- 在保持意义明确的同时尽可能地保持变量简短。

  虽然要确保变量名称具有一定描述性，但也不能太过，不能使用过于复杂和过长的名称。
- 为了在循环中起到优化作用，请只使用单字符变量名称。

  另外，您还可以在循环中将单字符变量作为临时变量（例如 `i`、`j`、`k`、`m` 和 `n`）。仅对短循环索引，或当性能优化和速度很关键时才使用这些单字符变量名称。下面的示例演示了这种用法：

  ```
  var fontArr:Array = TextField.getFontList();
  fontArr.sort();
  var i:Number;
  for (i = 0; i<fontArr.length; i++) {
    trace(fontArr[i]);
  }
  ```

- 变量以小写字母开头。

  首字母大写的名称应保留给类、接口，等等名称。
- 对于相连接的词使用混合大小写。

  例如，使用 myFont，而不要使用 myfont。
- 不要使用首字母缩写词和缩写词。

  此规则只有一种例外情况，即首字母缩写词或缩写词表示这一术语的标准使用方式，例如 HTML 或 CFM。对于常用的首字母缩写词，应使用混合大小写，如 newHtmlParser 而不用 newHTMLParser 以提高可读性。
- 在创建一组相关的变量名称时，应使用互补对。

  例如，您可以使用互补对来指示游戏中的最低得分和最高得分，如下所示：

```
var minScoreNum:Number = 10; // 最低得分
var maxScoreNum:Number = 500; // 最高得分
```

命名常数

若要引用某些定值的属性，则可使用常数。这有助于查找代码中的字面错误，如果使用了文本则可能找不到这些错误。使用常数还可以在单独的位置上更改值。

变量应该是小写字母或混合大小写字母；但是，您应该遵循以下这些准则来命名静态常数（不更改的变量）：

- 常数应为大写字母。
- 分隔单词应包含下划线。

在下面的 ActionScript 代码片断中，您可以看到这些准则的具体应用：

```
var BASE_URL:String = "http://www.macromedia.com"; // 常数
var MAX_WIDTH:Number = 10;  // 常数
```

不要直接对数字常数进行编码，除非常数为 1、0 或 -1，这些常数可能在 for 循环中用作计数器值。

命名布尔变量

布尔变量应以字 “is”开头（因为布尔值本质上只有 “is”或 “is not”两个值）。因此，可以使用下面的变量判断一个婴儿是不是女孩（是一个布尔值）：

```
isGirl
```

或者对于判断用户已登录（还是没有登录）的变量，可以使用下面的变量名称：

```
isLoggedIn
```

命名函数和方法

在代码中命名函数和方法时，应遵循以下准则。有关编写函数和方法的信息，请参见第 5 章“函数和方法”。

- 使用具有描述性的名称
- 在相连接的字中使用混合大小写。

  `singLoud()` 就是一个很好的示例。
- 函数和方法的名称以小写字母开头
- 在函数名称中描述要返回的值。

  例如，如果要返回歌曲名称，则可将函数命名为 `getCurrentSong()`。
- 为相关的类似函数建立命名标准。

  ActionScript 2.0 不允许重载。在面向对象编程的上下文中，重载 指根据传递的数据类型而使函数具有不同行为的能力。
- 将方法命名为动词。

  您可能连接名称，但名称中应当包含动词。大多数方法都用动词来表示，因为方法对对象执行操作。

方法名称的示例有：

```
sing();
boogie();
singLoud();
danceFast();
```

命名类和对象

当创建新类文件，为类和 ActionScript 文件命名时应遵循下列准则。有关适当的格式设置，请参见下面的类名称示例：

```
class Widget;
class PlasticWidget;
class StreamingVideo;
```

类中可能有公共和私有成员变量。类可以包含不希望用户直接设置或访问的变量。将这些变量设为私有变量，并仅允许用户使用 getter/setter 方法访问值。

下列准则应用于命名类：

- 类名称以大写字母开头。
- 当类名称为组合词或相连接的词时，用混合大小写编写类名称。

  对于组合词或相连接的词，应以大写字母开头。`NewMember` 便是一个很好的示例。

- 类名称通常为名词或限定名词。

  限定词描述名词或短语。例如，不要使用“member”，而改用 NewMember 或 OldMember 来限定名词。
- 名称的含义清楚比名称简短更为重要。
- 不要使用首字母缩写词和缩写词。

  此规则只有一种例外情况，即首字母缩写词或缩写词表示这一术语的标准使用方式，例如 HTML 或 CFM。对于常用的首字母缩写词，应使用混合大小写，如 NewHtmlParser 而不是 NewHTMLParser，以提高可读性。
- 使用对类内容具有描述性意义的简短名称。

  为了避免产生含糊或误解，应使用通用名称。
- 有时类名称会是一个组合词。

  限定词可以描述名词或短语。例如，不使用“member”，改用 NewMember 或 OldMember 来限定名词。
- 在类名称中不要使用复数形式的字 （如 Witches 或 BaldPirates）。

  大多数情况下，将字保留为限定名词效果更佳。限定词描述名词或短语。例如，不使用“cat”或 “buckaneer”，改用 BlackCat 或 OldBuckaneer 来限定名词。
- 不要在类的属性中使用类名称，因为这会过于冗长。

  例如，不要使用 Cat.catWhiskers.改为 Cat.whiskers 会好得多。
- 不要使用可以解释为动词的名词。

  例如， Running 或 Gardening。使用这些名词有可能导致与方法、状态或其它应用程序活动发生混淆。
- 在一个应用程序中为每个类使用唯一的类名称。
- 不要给类命名，因为可能会与 Flash 中内置类的名称发生冲突。
- 尝试表明类在层次中的关系。

  这可帮助在应用程序中显示类的关系。例如，您可能有 Widget 接口，而且 Widget 的实现可能为 PlasticWidget、 SteelWidget 和 SmallWidget。

有关接口的信息，请参见第 8 章 “接口”。

命名包

使用“反向域”命名约定命名包是很常见的。反向域名称的示例有用 com.macromedia 命名 macromedia.com，用 org.yourdomain 命名 yourdomain.org。

命名包时应遵循下列准则：

- 包名称的前缀全部使用小写字母。

  例如，com、mx 或 org。
- 将相关的类（具有相关功能的类）放入相同的包中。
- 包名称以一致的前缀开始。

  例如，可能要使用 com.macromedia.projectName 来保持一致性。另一个示例是《学习 Flash 中的 ActionScript 2.0》全书中使用的 com.macromedia.docs.learnAS2.Users。
- 使用意义清晰并且具有自我说明性质的包名称。

  对包的责任进行说明非常重要。例如，您可能拥有一个名为 Pentagons 的包，该包负责使用 Flash 绘图 API 在文档示例中绘制不同种类的五边形；则其名称可以是 com.macromedia.docs.as2.Pentagons。
- 为组合包名称或相连接的包名称使用混合大小写。

  packageName 便是组合的相连接的包名称的示例。请记住，前缀全部使用小写字母（如 com、org 等）。
- 不要使用下划线或美元符号字符

命名接口

接口名称以大写字母“I”开头可帮助您将接口和类区分开。下面的接口名称 IEmployeeRecords 使用大写的首字母和带有混合大小写的相连接的字，如下所示：

```
interface IEmployeeRecords{}
```

下列约定也适用：

- 接口名称的首字母要大写。

  此约定与类名称约定相同。
- 接口名称通常为形容词。

  Printable 便是一个非常好的示例。

有关接口的更多信息，请参见第 8 章 “接口”。

命名自定义组件

组件名称的第一个字母为大写，并且任何相连接的字都是以混合大小写来书写。例如，下面的默认用户接口组件设置均使用相连接的单词和混合大小写：

- CheckBox
- ComboBox
- DataGrid
- DateChooser
- DateField
- MenuBar
- NumericStepper
- ProgressBar
- RadioButton
- ScrollPane
- TextArea
- TextInput

不使用相连接的字的组件以大写字母开头。

如果开发自定义组件，应使用命名约定以防止与 Macromedia 组件的命名不兼容。必须将组件的名称与 Flash 提供的默认设置的名称区分开来。如果采用自己的一致的命名约定，则可帮助您防止命名冲突。

请记住，本部分中的命名约定均为指导性质。最重要的是使用一种适合您的命名方案，并且应以一致的方式使用该命名方案。

在代码中使用注释

本部分介绍如何在代码中使用注释。注释会记录您在代码中所做的决策，回答如何做 和为何做 这两个问题。例如，可以在注释中说明一个变通办法。另一个开发人员可以方便地找到相关的需要更新和修复的代码。最后，此问题可能会在 Flash 或 Flash Player 将来的版本中得到解决，因此，就不再需要该变通办法了。

有关在 ActionScript 代码中编写注释的更多信息，请参见以下部分：

- 第 662 页的 “编写好的注释”
- 第 663 页的 “向类添加注释”

编写好的注释

在 ActionScript 2.0 代码中以一致的方式使用注释允许您描述复杂的代码区域或无法以其它方式解释清楚的重要的交互。注释必须清楚地解释代码的意图，而不是仅仅翻译代码。如果代码中有些内容含义不明显，则应对其添加注释。

如果对代码使用“自动套用格式”工具，您会注意到尾随注释（请参见第 82 页的“尾随注释”）会移至下一行。可以在设置代码格式之后添加这些注释，或在使用“自动套用格式”工具之后必须修改这些注释的新位置。

有关在类中使用注释的信息，请参见第 663 页的“向类添加注释”。

向代码中添加注释时使用以下准则：

- 多行注释使用块注释 (/* 和 */)，而简短的注释则使用单行注释 (//)。

  如有必要，还可以在 ActionScript 代码的同一行使用尾随注释。
- 确保不要使用注释对 ActionScript 代码进行简单翻译。

  ActionScript 代码中意义明确的元素无须注释。
- 对代码中含义不明确的元素添加注释。

  特别是在主题前后的段落中未对主题进行描述时应添加注释。
- 不要使用混乱的注释。

  一行混乱的注释通常包含等号 (=) 或星号 (*)。请改为使用空白来分隔注释与 ActionScript 代码。

注意	如果使用“自动套用格式”工具来设置 ActionScript 格式，将会删除空白。请记住将空白添加回去或使用单行注释 (//) 来维持间距；设置代码格式后，这些行很容易删除。

- 在部署项目前，将多余的注释从代码中删除。

  如果发现 ActionScript 代码中有许多注释，请考虑是否需要改写代码中的某些部分。如果感觉必须包括许多关于 ActionScript 代码如何工作的注释，则通常表示编写的代码很差。

注意	使用注释在以教授用户为目的的 ActionScript 代码中最为重要。例如，如果正在以教授 Flash 为目的而创建应用程序范例，或正在编写关于 ActionScript 代码的教程，则可向代码添加注释。

向类添加注释

典型的类或接口文件中的两类注释分别为文档注释 和实现注释。

注意	文档注释和实现注释在 ActionScript 语言中并没有正式描述。但是，在编写类文件和接口文件时，开发人员通常都使用这两种注释。

可以使用文档注释来描述代码的规范，但不能描述实现。可以使用实现注释来注释掉代码，或对代码特定部分的实现加以注释。文档注释是以 /** 和 */ 分隔的，而实现注释是以 /* 和 */ 分隔的。

可以使用文档注释描述接口、类、方法和构造函数。每个类、接口或成员都包含一个文档注释，并在其后紧跟着声明。如果您有其它不适合放在文档注释中的文档信息，请使用实现注释 （采用块注释或单行注释的形式）。

类应以一个标准的注释开始，该注释使用以下格式：

```
/**
  User 类
  1.2 版
  3/21/2004
  版权所有 Macromedia, Inc.
 */
```

在文档注释后，声明类。实现注释应紧随声明之后。

注意	不要包含与正在阅读的类没有直接关联的注释。例如，不要包含描述相应包的注释。

在类体内使用块、单行和尾随注释对 ActionScript 代码进行注释。有关在类文件中使用注释的更多信息，请参见第 663 页的 “向类添加注释”。

ActionScript 编码约定

编程工作最重要的方面之一是一致性，不管是变量命名方案 （在第 652 页的 “命名约定”中介绍），设置代码格式 （在第 679 页的 “设置 ActionScript 语法的格式”中介绍），还是 ActionScript 2.0 代码的编码标准和放置（本节中介绍）都需要保持一致性。如果代码组织有序并遵守标准，那么您可以极大地简化代码调试和维护。

有关编码约定的更多信息，请参见以下主题：

- 第 664 页的 “将 ActionScript 代码保存在一个位置”
- 第 664 页的 “将代码附加到对象”
- 第 665 页的 “处理范围”
- 第 668 页的 “构建类文件”
- 第 675 页的 “关于使用函数”

将 ActionScript 代码保存在一个位置

应可能将 ActionScript 2.0 代码放在一个位置，如一个或多个外部 ActionScript 文件或时间轴的第 1 帧上（将代码放在时间轴上时，该代码称为帧脚本）。

如果将 ActionScript 代码放在帧脚本中，请将 ActionScript 代码放在时间轴的第 1 帧或第 2 帧名为 Actions 的图层上，这也是时间轴上的第一图层或第二图层。有时您可能会创建两个图层（一种可接受的做法），以便 ActionScript 将函数分离开来。有些 Flash 应用程序并不总是将所有代码放在同一位置（尤其是在使用屏幕或行为的时候）。

尽管存在上述极少例外，您通常还是可以将所有代码放在同一位置。以下是将 ActionScript 放在一个位置的优点：

- 容易在可能很复杂的源文件中找到代码。
- 容易调试代码。

调试 FLA 文件的最困难的工作之一就是找到所有代码。找到所有代码后，必须弄清楚这些代码如何与其它代码片断以及 FLA 文件进行交互。如果您将所有代码放在一个帧中，因为代码是集中的，调试起来就容易得多，而且此类问题出现的次数也有所减少。有关将代码附加到对象（和分散代码）的信息，请参见第 664 页的“将代码附加到对象”。有关行为和分散的代码的信息，请参见《使用 Flash》中的第 3 章“使用行为的最佳做法”。

将代码附加到对象

一定要避免将 ActionScript 代码附加到 FLA 文件中的对象（如按钮或影片剪辑实例）上，即使是在简单的或原型应用程序中也不应这样。将代码附加到对象意味着您选择一个影片剪辑、组件或按钮实例，打开 ActionScript 编辑器（“动作”面板或“脚本”窗口），然后使用 `on()` 或 `onClipEvent()` 处理函数添加 ActionScript 代码。

出于以下原因，强烈建议不要采取此做法：

- 附加到对象的 ActionScript 代码很难定位，而且 FLA 文件很难编辑。
- 附加到对象的 ActionScript 代码很难调试。
- 在时间轴上或类中编写的 ActionScript 代码更完美、更容易进行构建。
- 附加到对象的 ActionScript 代码导致编码风格欠佳。
- 附加到对象的 ActionScript 代码使学员和读者不得不学习其它语法，以及通常拙劣而有限的不同的编码风格。
- 通常，用户必须在以后重新学习如何在时间轴上编写函数等。

有些 Flash 用户可能会说，将代码附加到对象后学习 ActionScript 变得容易了。还有一些用户会说使用这种方法添加简易的代码以及编写或教授 ActionScript 比较容易。但是，这两种编码风格间的对照（放置在对象上的代码和帧脚本）可能会给学习 ActionScript 的开发人员造成混乱，应予以避免。而且，学习如何编写附加到对象的代码的用户通常必须在以后重新学习如何以帧脚本的形式放置等效代码。这说明了在学习过程中通过编写帧脚本来始终贯彻一致性的好处。

以下为将 ActionScript 代码附加到称为 `myBtn` 的按钮。应避免使用这种方法：

```
on (release) {
  // 完成一些代码。
}
```

不过，以下为将等效 ActionScript 代码放置在时间轴上：

```
// 优良的代码
myBtn.onRelease = function() {
  // 完成一些代码。
};
```

有关 ActionScript 语法的更多信息，请参见第 679 页的“设置 ActionScript 语法的格式”。

注意

使用行为和屏幕有时需要将代码附加到对象，所以在使用这些功能时要采取不同的做法。有关更多信息，请参见《使用 Flash》中的第 3 章“使用行为的最佳做法”。

处理范围

范围是变量已知并可用在 SWF 文件中的区域（如在时间轴上），大到跨越应用程序的全局范围，小至函数内部的局部范围。在编写代码时，通常可以通过多种方式引用范围。正确使用范围意味着您能够创建可移植且可重复使用的 ActionScript 代码，并在构建新模块时不会存在中断应用程序的风险。

了解全局范围和根范围之间的区别非常重要。根范围对于加载的每个 SWF 文件来说都是唯一的。全局范围适用于 SWF 文件内的所有时间轴和范围。可以使用相对寻址而不是引用根时间轴，因为相对寻址使得您的代码可以重复使用和移植。有关在应用程序中处理范围的更多信息，请参见以下部分：

第 297 页的“关于变量和作用域”

第 72 页的“关于作用域和目标设定”

第 213 页的“理解类和作用域”

避免使用绝对目标 (_root)

您可以使用多种方法来将实例确定为目标从而避免使用 _root，这些方法将在本部分的后面进行讨论。尽量避免在 ActionScript 2.0 中使用 _root，因为加载到其它 SWF 文件中的 SWF 文件可能无法正常工作。_root 标识符以正在加载的基本 SWF 文件为目标，而不是以使用相对寻址（而非 _root）的 SWF 文件为目标。此问题限制了加载到其它文件中的 SWF 文件的代码可移植性，在组件和影片剪辑中尤其如此。您可以使用 _lockroot 帮助解决这些问题，但如非必要（例如，加载 SWF 文件但没有访问 FLA 文件的权限时）尽量不要使用 _lockroot。有关使用 _lockroot 的更多信息，请参见第 666 页的 “使用 _lockroot”。

使用 this、this._parent 或 _parent 关键字（而不是 _root），具体取决于 ActionScript 2.0 代码所在的位置。下面的示例显示相对寻址：

```
myClip.onRelease = function() {
  trace(this._parent.myButton._x);
};
```

必须给所有变量确定范围，但作为函数参数的变量和局部变量除外。只要可能，就应使用相对寻址（如 this 属性）方法参照变量的当前路径确定变量的范围。有关使用 this 属性的更多信息，请参见《ActionScript 2.0 语言参考》中的 this 属性。

使用 _lockroot

可以使用 _lockroot 将内容确定为目标，并做为一种解决范围确定问题的方式，范围确定问题有时与 _root 的使用不当有关。尽管这可以解决应用程序中的许多问题，但应将 _lockroot 视为由使用 _root 而造成问题的变通办法。如果在将内容加载到 SWF 文件或组件实例时遇到问题，可尝试对加载内容的影片剪辑应用 _lockroot。例如，如果您使用名为 myClip 的影片剪辑加载内容，而该影片剪辑在完成加载后停止工作，请尝试使用下面的代码（位于时间轴上）：

```
this._lockroot = true;
```

使用 this 关键字

如果可能，请将 this 关键字用作前缀而不是省略该关键字，即使代码没有此关键字也可以正常运行。使用 this 关键字了解方法或属性在何时属于特定的类。例如，对于时间轴上的函数，可以使用下面的格式编写 ActionScript 2.0 代码：

```
circleClip.onPress = function() {
  this.startDrag();
};
circleClip.onRelease = function() {
  this.stopDrag();
};
```

对于类，可以使用下面的格式编写代码：

```
class User {
  private var username:String;
  private var password:String;
  function User(username:String, password:String) {
    this.username = username;
    this.password = password;
  }
  public function get username():String {
    return this.username;
  }
  public function set username(username:String):Void {
    this.username = username;
  }
}
```

如果在这些情况下始终添加 this 关键字， ActionScript 2.0 代码将更容易阅读和理解。

关于类中的范围

将代码移植到 ActionScript 2.0 类中时，您可能必须更改使用 this 关键字的方式。例如，如果具有使用回调函数的类方法（如 LoadVars 类的 onLoad 方法），将很难知道 this 关键字引用的是类还是 LoadVars 对象。在此情况下，可能需要创建一个指向当前类的指针，如下面的示例所示：

```
class Product {
  private var m_products_xml:XML;
  // 构造函数
  // targetXmlstr 包含 XML 文件的路径
  function Product(targetXmlStr:String) {
    /* 创建对当前类的局部引用。
       即使在 XML 的 onLoad 事件处理函数内，您
       也可以引用当前类，而不仅仅是引用 XML 包。 */
    var thisObj:Product = this;
    // 创建一个局部变量，该变量用来加载 XML 文件。
    var prodXml:XML = new XML();
    prodXml.ignoreWhite = true;
```

```
    prodXml.onLoad = function(success:Boolean) {
      if (success) {
        /* 如果 XML 加载和分析成功，
           请将该类的 m_products_xml 变量设置为已分析
           的 XML 文档并调用 init 函数。 */
        thisObj.m_products_xml = this;
        thisObj.init();
      } else {
        /* 加载 XML 文件时出错。 */
        trace("error loading XML");
      }
    };
    // 开始加载 XML 文档
    prodXml.load(targetXmlStr);
  }
  public function init():Void {
    // 显示 XML 包
    trace(this.m_products_xml);
  }
}
```

由于您尝试引用 `onLoad` 处理函数内的私有成员变量，`this` 关键字实际上引用的是 `prodXml` 实例而不是您期望的 `Product` 类。因此，您必须创建一个指向本地类文件的指针，才能从 `onLoad` 处理函数直接引用该类。

有关类的更多信息，请参见第 213 页的 “理解类和作用域”有关范围的更多信息，请参见第 665 页的 “处理范围”。

构建类文件

在导入正在编译的 SWF 文件的独立 ActionScript 2.0 文件中创建类。

编译应用程序时，在导入到 SWF 文件中的独立 ActionScript 2.0 文件中创建类。要创建类文件，编写的代码可以具有一定的方法和顺序。以下部分将对此方法进行讨论。

以下构建类文件的约定说明可以如何对一个类的各部分进行排序以提高代码效率和改进代码可读性。

要构建类文件，请使用以下元素：

1. 添加文档注释，其中包含对代码的基本描述以及作者信息和版本信息。
2. 添加导入语句（如果适用）。
3. 编写类声明或接口声明，如下面的示例：

   ```
   UserClass{...}
   ```

4. 包括任何必要的类或接口实现注释。

   在此注释中，添加与整个类或接口有关的信息。

5. 添加所有静态变量。

   先编写公共类变量，接着编写私有类变量。

6. 添加实例变量。

   先编写公共成员变量，接着编写私有成员变量。

7. 添加构造函数语句，如下面示例中的语句：

   ```
   public function UserClass(username:String, password:String) {...}
   ```

8. 编写您的方法。

   要按功能对方法进行分组，而不要按可访问性或作用范围分组。使用这种方式的组织方法有助于提高代码的可读性和清晰度。

9. 在类文件中编写 getter/setter 方法。

创建类的准则

创建类文件时，请记住下面的准则：

- 每行仅放入一项声明。
- 不要在一行放置多项声明。

  例如，按以下示例所示设置声明的格式：

  ```
  var prodSkuNum:Number;// 产品 SKU（识别）号
  var prodQuantityNum:Number; // 产品数量
  ```

  本示例显示的格式比将两个声明放在同一行上要好。将这些声明放置在代码块的开头。

- 在声明时初始化局部变量。

  如果初始值设定项为编译时常数，则类属性只应在声明中初始化。

- 使用变量前应先声明。

  其中包括循环。

- 避免使用隐藏更高级声明的局部声明。

  例如，不要将一个变量声明两次，如下面的示例所示：

  ```
  var counterNum:Number = 0;
  function myMethod() {
    for (var counterNum:Number = 0; counterNum<=4; counterNum++) {
      // 语句;
    }
  }
  ```

  此代码在内部块中声明了相同的变量，这种做法应该避免。

- 不要在语句中为多个变量赋一个值。

  应遵循此约定，否则会使代码非常难于阅读，如下面的 ActionScript 代码所示：

  ```
  playBtn.onRelease = playBtn.onRollOut = playsound;
  ```

  或者

  ```
  class User {
    private var m_username:String, m_password:String;
  }
  ```

- 只有在出于某一原因需要公开时才将方法或属性设置为公共。否则，应将方法和属性设置为私有。
- 不要在类文件中过多使用 getter/setter 函数。

  虽然 Getter/setter 函数在许多方面（请参见第 187 页的“关于 getter 和 setter 方法”）非常出色，但过多使用则可能说明您应当改进应用程序的结构或组织。
- 将大多数成员变量设置为私有变量，除非有好的理由将其设置为公共变量。

  将成员变量设置为私有变量并且仅允许通过一组 getter/setter 函数访问这些变量，从设计的角度而言这样要好得多。

在类文件中使用 this 前缀

在类中使用 `this` 关键字作为方法和成员变量的前缀。尽管 this 关键字并不是必需的，但是当一个属性或方法带有前缀时，很容易判断它属于一个类；如果没有，则很难辨别该属性或方法是否属于该超类。

即使在一个类中，您也可以对静态变量和静态方法使用类名称前缀。这有助于限定所做的引用。限定引用使代码具有很好的可读性。根据所使用的编码环境，您的前缀可能还会触发代码完成和提示。下面的代码演示用类名称作为静态属性的前缀：

```
class Widget {
  public static var widgetCount:Number = 0;
  public function Widget() {
    Widget.widgetCount++;
  }
}
```

注意

您不一定非得添加这些前缀，有些开发人员觉得这样做没有必要。Macromedia 建议添加 `this` 关键字作为前缀，因为它可以提高可读性，并通过提供上下文来帮助您编写整洁的代码。

关于初始化

对于变量的初始值，指定一个默认值或者允许使用 undefined 值，如下面的类示例所示。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数。即，该表达式不能引用任何在运行时设置或定义的变量。编译时常数包括字符串、数字、布尔值、 null 和 undefined，以及以下顶级类的构造函数：Array、 Boolean、 Number、 Object 和 String。此类将 m_username 和 m_password 的初始值设置为空字符串：

```
class User {
  private var m_username:String = "";
  private var m_password:String = "";
  function User(username:String, password:String) {
    this.m_username = username;
    this.m_password = password;
  }
}
```

不再需要变量时，请删除变量或使变量为 null。将变量设置为 null 也可提高性能。此过程通常称为垃圾回收。删除变量有助于在运行时优化内存利用，因为从 SWF 文件中删除了不需要的资源。删除变量比将变量设置为 null 更好。有关性能的更多信息，请参见第 678 页的 “优化代码”。

注意	Flash Player 8 对 Flash Player 中的垃圾回收功能做了进一步的改进。

有关命名变量的信息，请参见第 655 页的 “命名变量”。有关删除对象的更多信息，请参见《ActionScript 2.0 语言参考》中的 delete 语句。

使用 ActionScript 2.0 初始化代码的最容易的方法之一就是使用类。您可以在类的构造函数内封装一个实例的所有初始化，或将初始化提取到一个单独的方法中，在创建变量后再显式调用此方法，如下面的代码所示：

```
class Product {
  function Product() {
    var prodXml:XML = new XML();
    prodXml.ignoreWhite = true;
    prodXml.onLoad = function(success:Boolean) {
      if (success) {
        trace("loaded");
      } else {
        trace("error loading XML");
      }
    };
    prodXml.load("products.xml");
  }
}
```

下面的代码可能是应用程序中的第一个函数调用，而且是您为初始化而进行的唯一一次函数调用。正在加载 XML 的 FLA 文件的第 1 帧可能会使用类似如下 ActionScript 的代码：

```
if (init == undefined) {
  var prodXml:XML = new XML();
  prodXml.ignoreWhite = true;
  prodXml.onLoad = function(success:Boolean) {
    if (success) {
      trace("loaded");
    } else {
      trace("error loading XML");
    }
  };
  prodXml.load("products.xml");
  init = true;
}
```

使用 trace 语句

创作 FLA 文件时，在文档中使用 `trace` 语句有助于调试代码。例如，通过使用 `trace` 语句和 `for` 循环，您可以在“输出”面板中查看变量的值，如字符串、数组和对象，如下面的示例所示：

```
var dayArr:Array = ["sun", "mon", "tue", "wed", "thu", "fri", "sat"];
var numOfDays:Number = dayArr.length;
for (var i = 0; i<numOfDays; i++) {
  trace(i+": "+dayArr[i]);
}
```

此示例在“输出”面板中显示下面的信息：

```
0: sun
1: mon
2: tue
3: wed
4: thu
5: fri
6: sat
```

使用 `trace` 语句是一种有效的调试 ActionScript 2.0 的方法。

您可以在发布 SWF 文件时删除 `trace` 语句，这可使播放性能得到一定的提高。发布 SWF 文件前，请打开“发布设置”，并在“Flash”选项卡上选择“省略跟踪动作”。有关使用 trace 的更多信息，请参见《ActionScript 2.0 语言参考》中的 trace 函数。

调试器工具对于调试 ActionScript 代码也非常有用。有关更多信息，请参见第 18 章“调试应用程序”。

关于 super 前缀

如果您引用父类中的某个方法，可为该方法加上 super 前缀，这样其他开发人员就知道从何处调用该方法。下面的 ActionScript 2.0 代码片断演示使用 super 前缀确定适当范围的用法:

在下面的示例中，您创建了两个类。在 Socks 类中使用 super 关键字，调用其父类 (Clothes) 中的函数。虽然 Socks 和 Clothes 这两个类都有一个名为 getColor() 的函数，但使用 super 可使您明确引用基类的方法和属性。创建名为 Clothes.as 的新 AS 文件，并输入以下代码:

```
class Clothes {
   private var color:String;
   function Clothes(paramColor) {
      this.color = paramColor;
      trace("[Clothes] I am the constructor");
   }
   function getColor():String {
      trace("[Clothes] I am getColor");
      return this.color;
   }
   function setColor(paramColor:String):Void {
      this.color = paramColor;
      trace("[Clothes] I am setColor");
   }
}
```

创建一个名为 Socks 的新类，扩展 Clothes 类，如下面的示例所示:

```
class Socks extends Clothes {
   private var color:String;
   function Socks(paramColor:String) {
      this.color = paramColor;
      trace("[Socks] I am the constructor");
   }
   function getColor():String {
      trace("[Socks] I am getColor");
      return super.getColor();
   }
   function setColor(paramColor:String):Void {
      this.color = paramColor;
      trace("[Socks] I am setColor");
   }
}
```

然后创建一个新 AS 文件或 FLA 文件，并在文档中输入以下 ActionScript:

```
import Socks;
var mySock:Socks = new Socks("maroon");
trace(" -> "+mySock.getColor());
mySock.setColor("Orange");
trace(" -> "+mySock.getColor());
```

下面的结果显示在“输出”面板中：

```
[Clothes] I am the constructor
[Socks] I am the constructor
[Socks] I am getColor
[Clothes] I am getColor
-> maroon
[Socks] I am setColor
[Socks] I am getColor
[Clothes] I am getColor
-> Orange
```

如果忘记在 Socks 类的 `getColor()` 方法前放置 `super` 关键字，则 `getColor()` 可能会反复调用自己，进而可能由于无限递归问题而导致脚本失败。如果没有使用 super 关键字，“输出”面板会显示以下错误：

```
[Socks] I am getColor
[Socks] I am getColor
...
[Socks] I am getColor
256 levels of recursion were exceeded in one action list.
This is probably an infinite loop.
Further execution of actions has been disabled in this SWF file.
```

避免使用 with 语句

对于学习 ActionScript 2.0 的人来说，更容易混淆的一个概念是使用 `with` 语句。请看使用 `with` 语句的如下代码：

```
this.attachMovie("circleClip", "circle1Clip", 1);
with (circle1Clip) {
  _x = 20;
  _y = Math.round(Math.random()*20);
  _alpha = 15;
  createTextField("label1Txt", 100, 0, 20, 100, 22);
  label1Txt.text = "Circle 1";
  someVariable = true;
}
```

在此代码中，您从库中附加一个影片剪辑实例，并使用 `with` 语句修改其属性。如果不指定变量的范围，则无法总是了解在何处设置属性，从而代码就会混淆不清。在上面的代码中，您可能希望在 `circle1Clip` 影片剪辑中设置 `someVariable`，但实际上它是在 SWF 文件的时间轴中设置的。

如果明确指定了变量范围而不是依赖于 with 语句，则比较容易跟踪代码中发生的情况。下面的示例显示稍长但效果较好的 ActionScript 示例，它指定了变量范围：

```
this.attachMovie("circleClip", "circle1Clip", 1);
circle1Clip._x = 20;
circle1Clip._y = Math.round(Math.random()*20);
circle1Clip._alpha = 15;
circle1Clip.createTextField("labelTxt", 100, 0, 20, 100, 22);
circle1Clip.labelTxt.text = "Circle 1";
circle1Clip.someVariable = true;
```

此规则的一个例外是，在用绘图 API 绘制形状时，由于绘图 API 的功能，您可能会对同样的方法 （如 lineTo 或 curveTo）进行多次类似的调用。例如，绘制一个简单的矩形时，需要对 lineTo 方法进行四次单独的调用，如下面的代码所示：

```
this.createEmptyMovieClip("rectangleClip", 1);
with (rectangleClip) {
  lineStyle(2, 0x000000, 100);
  beginFill(0xFF0000, 100);
  moveTo(0, 0);
  lineTo(300, 0);
  lineTo(300, 200);
  lineTo(0, 200);
  lineTo(0, 0);
  endFill();
}
```

如果用全限定实例名称编写每个 lineTo 或 curveTo 方法，代码很快就会变得混乱并且难以阅读和调试。

关于使用函数

尽可能重复使用代码块。一种重复使用代码的方法是多次调用一个函数，而不是每次都创建不同的代码。函数可以是一般代码片断；因此，可以在一个 SWF 文件中使用相同的代码块达到稍有差别的多个目的。重复使用代码使您可以创建高效的应用程序，并可以将必须编写的 ActionScript 2.0 代码数量减小到最低限度，从而缩短开发时间。您可以在时间轴上、类文件中创建函数，或编写驻留在基于代码的组件上的 ActionScript，并以不同的方式重复使用这些代码。

如果使用 ActionScript 2.0，则应避免在时间轴上编写函数。使用 ActionScript 2.0 时，请尽可能将函数放在类文件中，如下面的示例所示：

```
class Circle {
public function area(radius:Number):Number {
  return (Math.PI*Math.pow(radius, 2));
}
public function perimeter(radius:Number):Number {
  return (2 * Math.PI * radius);
}
```

```
public function diameter(radius:Number):Number {
  return (radius * 2);
}
}
```

在创建函数时使用下面的语法：

```
function myCircle(radius:Number):Number {
  //...
}
```

避免使用下面的难以阅读的语法：

```
myCircle = function(radius:Number):Number {
  //...
}
```

下面的示例将函数放在类文件中。选择使用 ActionScript 2.0 时，这是最佳的做法，因为它可以最大限度地提高代码可重用性。若要重复使用其它应用程序中的函数，则可以导入现有的类（而不是从头重新编写代码），或者将函数直接复制到新的应用程序中。

```
class mx.site.Utils {
  static function randomRange(min:Number, max:Number):Number {
    if (min>max) {
      var temp:Number = min;
      min = max;
      max = temp;
    }
    return (Math.floor(Math.random()*(max-min+1))+min);
  }
  static function arrayMin(numArr:Array):Number {
    if (numArr.length == 0) {
      return Number.NaN;
    }
    numArr.sort(Array.NUMERIC | Array.DESCENDING);
    var min:Number = Number(numArr.pop());
    return min;
  }
  static function arrayMax(numArr:Array):Number {
    if (numArr.length == 0) {
      return undefined;
    }
    numArr.sort(Array.NUMERIC);
    var max:Number = Number(numArr.pop());
    return max;
  }
}
```

您可能会通过将下面的 ActionScript 添加到 FLA 文件来使用这些函数：

```
import mx.site.Utils;
var randomMonth:Number = Utils.randomRange(0, 11);
var min:Number = Utils.arrayMin([3, 3, 5, 34, 2, 1, 1, -3]);
var max:Number = Utils.arrayMax([3, 3, 5, 34, 2, 1, 1, -3]);
trace("month: "+randomMonth);
trace("min: "+min);
trace("max: "+max);
```

关于停止代码重复

`onEnterFrame` 事件处理函数十分有用，这是因为 Flash 可以用它来以 SWF 文件的帧频重复代码。但是，请尽可能限制 Flash 文件使用重复代码的数量，以免影响性能。例如，如果一段代码在播放头进入帧时重复，则该代码会占用大量处理器资源。这种行为可能会导致播放该 SWF 文件的计算机出现性能问题。如果在 SWF 文件中将 `onEnterFrame` 事件处理函数用于任何种类的动画或重复，则应在 `onEnterFrame` 处理函数使用完毕时将其删除。在下面的 ActionScript 2.0 代码中，通过删除 `onEnterFrame` 事件处理函数来停止重复：

```
circleClip.onEnterFrame = function() {
  circleClip._alpha -= 5;
  if (circleClip._alpha<=0) {
    circleClip.unloadMovie();
    delete this.onEnterFrame;
    trace("deleted onEnterFrame");
  }
};
```

同样，请限制使用 `setInterval`，记住在使用完毕时清除间隔，以减少 SWF 文件对处理器的要求。

ActionScript 和 Flash Player 优化

如果您编译一个包含 ActionScript 2.0 的 SWF 文件，且该文件的发布设置为 Flash Player 6 和 ActionScript 1.0，则只要代码不使用 ActionScript 2.0 类，代码就会起作用。代码不区分大小写，只是与 Flash Player 有关。因此，如果编译 “发布设置” 为 Flash Player 7 或 8 和 ActionScript 1.0 的 SWF 文件，则 Flash 将强制区分大小写。

“发布设置” 为 ActionScript 2.0 时，将在编译时为 Flash Player 7 和 8 强制添加数据类型注释 （严格数据类型）。

在发布应用程序时， ActionScript 2.0 编译为 ActionScript 1.0 字节代码，因此在使用 ActionScript 2.0 时目标播放器可以为 Flash Player 6、 7 或 8。

有关优化应用程序的更多信息，请参见 “优化代码”。

优化代码

优化代码时，请遵循下面的准则：

- 避免从一个循环中多次调用一个函数。

  在循环中包含小函数的内容，可使效果更佳。
- 尽可能使用本机函数。

  本机函数要比用户定义的函数运行速度更快。
- 不要过多使用 Object 类型。

  数据类型注释应力求精确，因为这样可以提高性能。只有在没有适当的备选数据类型时，才使用 Object 类型。
- 避免使用 `eval()` 函数或数据访问运算符。

  通常，较为可取且更有效的做法是只设置一次局部引用。
- 在开始循环前将 `Array.length` 赋予变量。

  在开始循环前将 `Array.length` 赋予变量，将其作为条件使用，而不是使用 `myArr.length` 本身。例如，应使用

  ```
  var fontArr:Array = TextField.getFontList();
  var arrayLen:Number = fontArr.length;
  for (var i:Number = 0; i < arrayLen; i++) {
    trace(fontArr[i]);
  }
  ```

  来代替：

  ```
  var fontArr:Array = TextField.getFontList();
  for (var i:Number = 0; i < fontArr.length; i++) {
    trace(fontArr[i]);
  }
  ```
- 注重优化循环及所有重复动作。

  Flash Player 花费许多时间来处理循环 （如使用 `setInterval()` 函数的循环）。
- 声明变量时，添加 `var` 关键字。
- 在局部变量够用时，不要使用类变量或全局变量。

设置 ActionScript 语法的格式

以标准化的方法设置 ActionScript 2.0 代码的格式对于编写可维护的代码是很必要的，并且能够使其他开发人员较容易理解和修改代码。例如，对于没有缩进或注释、以及没有一致的命名约定和格式的 FLA 文件，将极难理解其逻辑含义。通过对各代码块进行缩进（如循环和 `if` 语句），可以使代码易于阅读和调试。

有关设置代码格式的更多信息，请参见以下各部分：

一般格式设置准则

使用空格、换行符和 TAB 缩进向代码中添加空白，可以提高代码的可读性。空白可增强可读性，因为它有助于显示代码的层次结构。对于学生及处理复杂项目的经验老到的用户而言，通过使代码更加易读而使 ActionScript 2.0 更容易理解非常重要。调试 ActionScript 代码时可读性很重要，因为如果代码格式设置正确且间距适当，则更容易发现错误。

可以通过多种方式设置 ActionScript 2.0 代码段的格式，或编写代码段。您会发现在 ActionScript 编辑器（“动作”面板或 “脚本”窗口）中，供开发人员选择的跨多行设置语法格式的方法也有差别，例如，放置大括号 ({}) 或小括号 [()]) 的位置。

Macromedia 建议遵循以下格式设置要点来帮助增强 ActionScript 代码的可读性。

- 在 ActionScript 的段落（模块）间放置一空行。

  ActionScript 代码段落是逻辑相关的代码组。在段落之间添加空行有助于用户阅读 ActionScript 代码并了解其逻辑。

- 在代码中使用一致的缩进，有助于显示代码结构的层次。

  请在整个 ActionScript 代码中使用相同的缩进样式，并确保适当地对齐大括号 ({})。对齐大括号可提高代码的可读性。如果 ActionScript 语法正确，按 “Enter” (Windows) 或 “Return” (Macintosh) 时，Flash 会自动地以正确格式缩进代码。如果语法正确，则还可以在 ActionScript 编辑器（“动作”面板或 “脚本”窗口）中单击 “自动套用格式”按钮以缩进 ActionScript 代码。

- 使用换行符，使复杂语句更易阅读。

  可以多种方式设置某些语句的格式，如条件语句。有时将语句设置为跨多行而不是跨单行的格式，会使代码更容易阅读。

- 在后跟小括号 [()] 的关键字后加一个空格。

  下面的 ActionScript 代码显示了一个这样的示例：

```
do {
  // 一些代码
} while (condition);
```

- 不要在方法名称与小括号间加空格。

  下面的 ActionScript 代码显示了一个这样的示例：

```
function checkLogin():Boolean {
  // 语句;
}
checkLogin();
```

  或者

```
printSize("size is " + foo + "\n");
```

- 参数列表中的逗号后应跟有一个空格。

  在逗号后使用空格更容易区分方法调用和关键字，如以下示例所示：

```
function addItems(item1:Number, item2:Number):Number {
  return (item1 + item2);
}
var sum:Number = addItems(1, 3);
```

- 用空格分隔所有运算符和操作数。

  使用空格可更容易地区别方法调用和关键字，如以下示例所示：

```
// 好
var sum:Number = 7 + 3;
// 不好
var sum:Number=7+3;
```

  此准则的一个例外是点 (.) 运算符。

- 不要在一元运算符及其操作数间包含空格。

  例如，递增 (++) 和递减 (--) 运算符，如以下示例所示：

```
while (d++ = s++)
-2, -1, 0
```

- 不要在左括号后和右括号前包含空格。

  下面的 ActionScript 代码显示了一个这样的示例：

```
// 不好
( "size is " + foo + "\n" );
// 好
("size is " + foo + "\n");
```

- 将每条语句放置在一个单独的代码行中，以增加 ActionScript 代码的可读性。

  下面的 ActionScript 代码显示了一个这样的示例：

```
theNum++;        // 正确
theOtherNum++;   // 正确
aNum++; anOtherNum++;  // 错误
```

- 不要嵌入赋值语句。

  嵌入语句有时可以用来提高运行时 SWF 文件的性能，但代码的阅读和调试会变得非常困难。下面的 ActionScript 代码显示了一个这样的示例（但是要记住不要在实际代码中使用单字符命名）：

```
var myNum:Number = (a = b + c) + d;
```

- 将变量指定为单独的语句。

  下面的 ActionScript 代码显示了一个这样的示例（但是要记住不要在实际代码中使用单字符命名）：

```
var a:Number = b + c;
var myNum:Number = a + d;
```

- 在运算符之前换行。
- 在逗号之后换行。
- 将代码的下一行与上一行代码的表达式起始处对齐。

注意 选择 “编辑” > “首选参数” (Windows) 或 “Flash” > “首选参数” (Macintosh)，然后选择 ActionScript 选项卡，便可控制自动缩进和缩进设置。

编写条件语句

编写条件语句时请遵循下列准则：

- 在 if、else..if 和 if..else 语句中的单独行放置条件。
- 在 if 语句中使用大括号 ({})。
- 按照下面的示例所示设置大括号的格式：

```
// if 语句
if (condition) {
  // 语句
}

// if..else 语句
if (condition) {
  // 语句
} else {
  // 语句
}

// else..if 语句
if (condition) {
  // 语句
} else if (condition) {
  // 语句
} else {
  // 语句
}
```

编写复杂条件时，使用小括号 [()] 对条件进行组合是恰当的形式。如果不使用小括号，您（或使用 ActionScript 2.0 代码的其他人）就可能会遇到运算符优先级错误。

例如，下面的代码未在条件前后使用小括号：

```
if (fruit == apple && veggie == leek) {}
```

下面的代码通过在条件前后添加小括号而具有了恰当的形式：

```
if ((fruit == apple) && (veggie == leek)) {}
```

您可以用两种方式编写返回布尔值的条件语句。第二个示例更可取：

```
if (cartArr.length>0) {
  return true;
} else {
  return false;
}
```

将此示例与前一个示例进行比较：

```
// 更好
return (cartArr.length > 0);
```

第二个代码片断更短，需要计算的表达式更少。它更容易阅读和理解。

下面的示例检查变量 y 是否大于零 (0)，并返回结果 x/y 或值零 (0)。

```
return ((y > 0) ? x/y : 0);
```

下面的示例显示编写此代码的另一种方法。此示例更为可取：

```
if (y>0) {
  return x/y;
} else {
  return 0;
}
```

第一个示例中的缩短了的 if 语句语法称为条件运算符 (?:)。它允许您将简单的 if..else 语句转换为单个代码行。在这种情况下，缩短了的语法降低了可读性。

如果必须使用条件运算符，请在小括号内放入前导条件（在问号 [?] 前）以提高代码的可读性。在上面的代码片断中可以看到这样的示例。

编写复合语句

复合语句在大括号 ({}) 内包含一系列语句。这些大括号中的语句从复合语句的位置缩进。下面的 ActionScript 代码显示了一个这样的示例：

```
if (a == b) {
  // 此代码是缩进的。
  trace("a == b");
}
```

在构成控制结构 （if..else 或 for）的每个语句前后添加大括号，即使该控制结构只包含一个语句。下面的示例显示写法欠佳的代码：

```
// 不好
if (numUsers == 0)
  trace("no users found.");
```

尽管此代码有效，但其编写欠佳，因为语句前后缺少大括号。这种情况下，如果在 trace 语句后面添加另一个语句，则无论 numUsers 变量是否等于 0，代码都会执行：

```
// 不好
var numUsers:Number = 5;
if (numUsers == 0)
  trace("no users found.");
  trace("I will execute");
```

如果不考虑 numUsers 变量而执行代码，则可能导致意外结果。出于此原因，应该添加大括号，如下面的示例所示：

```
var numUsers:Number = 0;
if (numUsers == 0) {
  trace("no users found");
}
```

编写条件语句时，不要在代码中添加多余的 ==true，如下所示：

```
if (something == true) {
  // 语句
}
```

如果要与 false 比较，可以使用 if (something==false) 或 if(!something)。

编写 for 语句

可以使用下面的格式编写 for 语句：

```
for (init; condition; update) {
  // 语句
}
```

下面的结构演示 for 语句：

```
var i:Number;
for (var i = 0; i<4; i++) {
  myClip.duplicateMovieClip("newClip" + i + "Clip", i + 10, {_x:i*100,
  _y:0});
}
```

请记住在 for 语句中每个表达式后都加一个空格。

编写 while 和 do..while 语句

可以使用下面的格式编写 while 语句：

```
while (condition) {
  // 语句
}
```

可以使用下面的格式编写 do-while 语句：

```
do {
  // 语句
} while (condition);
```

编写 return 语句

不要对任何包含值的 return 语句使用小括号 [()]。唯一需要对 return 语句使用小括号的情况就是它们使值更明确时，如下面的 ActionScript 代码的第三行中所示：

```
return;
return myCar.paintColor;
// 用来使返回值更明确的小括号
return ((paintColor)? paintColor: defaultColor);
```

编写 switch 语句

- 所有 switch 语句都包含一个默认 case。

  default case 是 switch 语句中最后一个 case。default case 包括一个 break 语句，用于在添加其它 case 时阻止落空错误。

- 如果 case 没有 break 语句， case 将会落空 （请参见下面代码示例中的 case A）。

  语句应在 break 语句的位置处添加注释，您可以在下面示例中 case A 后面看到。在本例中，如果条件符合 case A， case A 和 case B 都会执行。

可以使用下面的格式编写 switch 语句：

```
switch (condition) {
case A :
  // 语句
  // 落空
case B :
  // 语句
  break;
case Z :
  // 语句
  break;
default :
  // 语句
  break;
}
```

编写 try..catch 和 try..catch..finally 语句

可以使用以下格式编写 try..catch 和 try..catch..finally 语句：

```
var myErr:Error;
// try..catch
try {
  // 语句
} catch (myErr) {
  // 语句
}

// try..catch..finally
try {
  // 语句
} catch (myErr) {
  // 语句
} finally {
  // 语句
}
```

关于使用侦听器语法

在 Flash 8 中可以通过多种方法为事件编写侦听器。下面的代码示例显示了一些常用技巧。第一个示例显示一个格式正确的侦听器语法，它使用 Loader 组件将内容加载到 SWF 文件中。progress 事件在加载内容时开始，而 complete 事件指示加载完成的时间。

```
var boxLdr:mx.controls.Loader;
var ldrListener:Object = new Object();
ldrListener.progress = function(evt:Object) {
  trace("loader loading:" + Math.round(evt.target.percentLoaded) + "%");
};
ldrListener.complete = function(evt:Object) {
  trace("loader complete:" + evt.target._name);
};
boxLdr.addEventListener("progress", ldrListener);
boxLdr.addEventListener("complete", ldrListener);
boxLdr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

也可以将本节中第一个示例稍微更改一下，即使用 handleEvent 方法，但这种技巧稍稍麻烦一些。Macromedia 不建议采用此技巧，因为必须使用一系列 if..else 语句或一个 switch 语句来检测捕获了哪个事件。

```
var boxLdr:mx.controls.Loader;
var ldrListener:Object = new Object();

ldrListener.handleEvent = function(evt:Object) {
  switch (evt.type) {
  case "progress" :
    trace("loader loading:" + Math.round(evt.target.percentLoaded) + "%");
    break;
  case "complete" :
    trace("loader complete:" + evt.target._name);
    break;
  }
};
boxLdr.addEventListener("progress", ldrListener);
boxLdr.addEventListener("complete", ldrListener);
boxLdr.load("http://www.helpexamples.com/flash/images/image1.jpg");
```

附录 A

错误消息

A

如果您发布到 ActionScript 2.0（默认设置），Macromedia Flash Basic 8 和 Macromedia Flash Professional 8 将提供编译时错误报告。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	类本身不能扩展。
1113	接口本身不能扩展。
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	接口方法的实现与其定义不匹配。
1123	此构造仅可用于 ActionScript 1.0。
1124	此构造仅在 ActionScript 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	在类的外面使用的属性。
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 ActionScript 2.0 类脚本中定义。
1140	ActionScript 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。 （因为现有类的全名是与之冲突的某个类的名称的一部分， ActionScript 2.0 编译器不能对这样的类进行编译，所以出现错误。例如，如果类 `mx.com` 是一个已编译的类，则编译类 `mx.com.util` 会生成错误 1141。）
1142	无法加载类或接口 “‹Class or Interface Name›”。

错误编号	消息文本
1143	接口只能在外部 ActionScript 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。
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 Flash 影片导出为 Flash 7 格式才能使用此动作。
1151	必须将您的 Flash 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的类或接口。
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 ActionScript 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 ActionScript 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。 （类名称与导入类的名称大小写不同时，发生此错误。例如，如果 util.as 文件中有语句 `import mx.Com`，则编译类 `mx.com.util` 会生成此错误 1166。）
1167	必须输入类名。
1168	输入的类名中有语法错误。
1169	输入的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。

错误编号	消息文本
1172	必须输入接口名。
1173	必须输入类名或接口名。
1174	输入的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：‹ 用户定义类的列表 ›。 此运行时错误指示您的自定义类彼此错误引用。
1180	当前的目标 Flash Player 不支持调试。
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›”，因为其叶名已经解析为正在定义的类 “‹C.B›”。 （例如，如果 util.as 文件中有语句 `import mx.util`，则编译类 `util` 会生成此错误 1189。）
1190	不能导入类 “‹A.B›”，因为其叶名已经解析为前面已导入的类 “‹C.B›”。 （例如，如果 AS 文件中也有语句 `import mx.util`，则编译 `import jv.util` 会生成错误 1190。）
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。

错误编号	消息文本
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。
1202	找不到 ActionScript 2.0 类型检查所需的 “toplevel.as”文件。请确保 ActionScript 首选参数的全局类路径中列出了目录 “$(LocalData)/Classes”。
1203	‹spanStart› 和 ‹spanEnd›› 之间的分支超过 32K 范围。
1204	包 “‹PackageName›”中没有名为 “‹packageName›”的类或包。
1205	当前的目标 Flash Player 不支持 FSCommand2 动作。
1206	成员函数 “‹functionName›”超过 32K。
1207	第 ‹lineNumber› 行附近的匿名函数超过 32K 范围。
1208	第 ‹lineNumber› 行附近的代码超过 32K 范围。
1210	包名称 “‹PackageName›”不能同时用作方法名称。
1211	包名称 “‹PackageName›”不能同时用作属性名称。
1212	无法为类 “‹ClassName›”创建 ASO 文件。请确保完全限定的类名称尽量短，以使 ASO 文件名 “‹ClassName.aso›”的长度小于 255 个字符。
1213	ActionScript 中不允许使用此类引号。请将它更改为标准 （直）双引号。

附录 B

不赞成使用的 Flash 4 运算符

B

下表列出了仅用于 Flash 4 的运算符，在 ActionScript 2.0 中不赞成使用。不要使用这些运算符，除非您要在 Flash Player 4 和更低版本中进行发布。

运算符	说明	结合律
not	逻辑 “非”	从右到左
and	逻辑 “与”	从左到右
or	逻辑 “或” (Flash 4)	从左到右
add	字符串连接 （原为 &）	从左到右
instanceof	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
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

附录 C

键盘键和键控代码值

C

以下表列出了标准键盘上的所有键，及其相应的键控代码值和 ASCII 键控代码值，这些值用于在 ActionScript 中标识这些键：

您可以使用键常量来截获按键的内置行为。有关 on() 处理函数的更多信息，请参见《ActionScript 2.0 语言参考》中的 on 处理函数。若要用 SWF 文件和按键捕获键控代码值和 ASCII 键控代码值，可以使用以下 ActionScript 代码：

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("DOWN -> Code:" + Key.getCode() + "\tACSII:" + Key.getAscii() +
  "\tKey:" + chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

有关 Key 类的更多信息，请参见《ActionScript 2.0 语言参考》中的 Key。若要在创作环境中测试 SWF 文件（“控制” > “测试影片”）时捕获键，请确保选择了 “控制” > “禁用快捷键”。

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，及其相应的键控代码值，这些值用于在 ActionScript 中标识这些键：

字母或数字键	键控代码	ASCII 键控代码
A	65	65
B	66	66
C	67	67
D	68	68
E	69	69
F	70	70
G	71	71
H	72	72
I	73	73
J	74	74
K	75	75
L	76	76
M	77	77
N	78	78
O	79	79
P	80	80
Q	81	81
R	82	82
S	83	83
T	84	84
U	85	85
V	86	86
W	87	87
X	88	88
Y	89	89
Z	90	90
0	48	48

字母或数字键	键控代码	ASCII 键控代码
1	49	49
2	50	50
3	51	51
4	52	52
5	53	53
6	54	54
7	55	55
8	56	56
9	57	57
a	65	97
b	66	98
c	67	99
d	68	100
e	69	101
f	70	102
g	71	103
h	72	104
i	73	105
j	74	106
k	75	107
l	76	108
m	77	109
n	78	110
o	79	111
p	80	112
q	81	113
r	82	114
s	83	115
t	84	116
u	85	117

字母或数字键	键控代码	ASCII 键控代码
v	86	118
w	87	119
x	88	120
y	89	121
z	90	122

数字键盘上的键

下表列出了数字键盘上的键，及其相应的键控代码值，这些值用于在 ActionScript 中标识这些键：

数字键盘键	键控代码	ASCII 键控代码
数字键盘 0	96	48
数字键盘 1	97	49
数字键盘 2	98	50
数字键盘 3	99	51
数字键盘 4	100	52
数字键盘 5	101	53
数字键盘 6	102	54
数字键盘 7	103	55
数字键盘 8	104	56
数字键盘 9	105	57
乘号	106	42
加号	107	43
Enter	13	13
减号	109	45
小数点	110	46
除号	111	47

功能键

下表列出了标准键盘上的功能键，及其相应的键控代码值，这些值用于在 ActionScript 中标识这些键：

功能键	键控代码	ASCII 键控代码
F1	112	0
F2	113	0
F3	114	0
F4	115	0
F5	116	0
F6	117	0
F7	118	0
F8	119	0
F9	120	0
F10	此键是系统保留的，不能在 ActionScript 中使用。	此键是系统保留的，不能在 ActionScript 中使用。
F11	122	0
F12	123	0
F13	124	0
F14	125	0
F15	126	0

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，及其相应的键控代码值，这些值用于在 ActionScript 中标识这些键：

Key	键控代码	ASCII 键控代码
Backspace	8	8
Tab	9	9
Enter	13	13
Shift	16	0
Control	17	0
Caps Lock	20	0
Esc	27	27
空格键	32	32
Page Up	33	0
Page Down	34	0
End	35	0
Home	36	0
左箭头	37	0
向上箭头	38	0
右箭头	39	0
向下箭头	40	0
Insert	45	0
Delete	46	127
Num Lock	144	0
ScrLk	145	0
Pause/Break	19	0
; :	186	59
= +	187	61

Key	键控代码	ASCII 键控代码
- _	189	45
/ ?	191	47
` ~	192	96
[{	219	91
\ |	220	92
] }	221	93
" '	222	39
,	188	44
.	190	46
/	191	47

有关其它键控代码和 ASCII 值，请使用本附录开始处的 ActionScript 并按所需的键以输出它的键控代码。

附录 D

为早期的 Flash Player 版本编写脚本

D

ActionScript 已根据 Macromedia Flash 创作工具和 Flash Player 的每个版本作出了相当大的更改。为 Macromedia Flash Player 8 创建内容时，可以使用 ActionScript 的全部功能。虽然仍可以使用 Flash 8 为早期版本的 Flash Player 创建内容，但不是 ActionScript 的所有元素都能被使用。

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。

注意

您可以在 Macromedia 网站上查看有关 Flash Player 版本使用率的调查，网址为 www.macromedia.com/software/player_census/flashplayer/。

关于将早期版本的 Flash Player 作为目标播放器

编写脚本时，可以使用《ActionScript 2.0 语言参考》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显示为黄色。

如果要为 Flash Player 6、Flash Player 7 或 Flash Player 8 创建内容，应使用 ActionScript 2.0，它具有 ActionScript 1.0 所没有的几个重要功能，例如，改进的编译器错误报告功能和更强的面向对象编程功能。

若要在发布文档时指定要使用的播放器和 ActionScript 版本，请选择“文件”>“发布设置”，然后在“Flash”选项卡上进行选择。如果要以 Flash Player 4 作为目标播放器，请参见下一部分。

使用 Flash 8 为 Flash Player 4 创建内容

若要使用 Flash 8 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 的 ActionScript 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 编写应用程序时，必须使用已否决的字符串运算符。这些运算符位于 “ActionScript”工具箱的 “否决的” > “运算符”类别中。

发布适用于 Flash Player 4 的文档时，可以使用以下 Flash 8 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 8 将它们导出为级数近似值，使用这种方法创建的结果数字精确度较低。另外，由于 SWF 文件中包含级数近似值，这些语言元素在 FlashPlayer 4 SWF 文件中需要的空间要比在 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

有关其它信息，请参见第 703 页的 “关于将早期版本的 Flash Player 作为目标播放器”。

使用 Flash 8 打开 Flash 4 文件

Flash 4 的 ActionScript 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 ActionScript 表达式，以使它们与新语法兼容。 Flash 进行以下数据类型转换和运算符转换：

- 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 的 ActionScript 中，这些运算符始终是数字运算符。在 Flash 5 和更高版本中，它们的行为因操作数的数据类型而异。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。（常数已经是明显的数字，因此无需包含在 Number() 中。）有关这些运算符的更多信息，请参见第 121 页的 “关于运算符的优先级和结合律”和第 693 页的 “不赞成使用的 Flash 4 运算符”中的运算符表。

- 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 文件中的所有 & 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右小括号，例如 Get Timer、Set Variable、Stop 和 Play。为了保持语法一致，getTimer 函数和所有动作现在都需要有小括号 [()]。这些小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，它将返回值 undefined，而不是 0。在 Flash 4 的更高版本的 ActionScript 中，语句 undefined == 0 为 false（而在 Flash 4 中，undefined == 1 为 false）。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 ActionScript 中使用了任何 Flash 5 或更高版本的关键字作为变量名，在 Flash 8 中进行编译时该语法将返回一个错误。若要解决此问题，请在所有出现的位置重命名这些变量。有关信息，请参见第 86 页的 “关于保留字”和第 293 页的 “关于命名变量”。

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，并且变量前有一个冒号，如下例所示：

```
myMovieClip/childMovieClip:myVariable
```

若要编写目标路径，使其与用 Flash Player 5 和更高版本支持的点语法编写的目标路径相同，请使用下面的语法：

```
myMovieClip.childMovieClip.myVariable
```

斜杠语法通常与 `tellTarget` 动作配合使用，但我们也不再建议使用该动作。现在，`with` 是首选动作，因为它与点语法的兼容性更好。有关更多信息，请参见《ActionScript 2.0 语言参考》中的 `tellTarget` 函数和 `with` 语句。

附录 E

使用 ActionScript 1.0 进行面向对象的编程

E

本附录中的信息选自 Macromedia Flash MX 文档，提供有关使用 ActionScript 1.0 对象模型编写脚本的信息。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 的面向对象的脚本，则必须使用 ActionScript 1.0。
- 如果您已使用 ActionScript 1.0 编写面向对象的脚本，而尚未准备好转换到 ActionScript 2.0，则可以在编写脚本时使用本附录来查找或查看需要的信息。

如果您从未使用 ActionScript 编写面向对象的脚本，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，因为已经否决了使用 ActionScript 1.0 编写面向对象的脚本。而应参见第 163 页的第 6 章 “类”以获得有关使用 ActionScript 2.0 的信息。

本章包含以下各部分：

警告

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此方法；如果您要以 Flash Player 5 作为目标播放器，请不要使用此方法。

关于 ActionScript 1.0

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

ActionScript 是一种面向对象的编程语言。面向对象的编程使用对象 或数据结构将控制对象行为或外观的属性和方法组合在一起。使用对象可以组织并重复使用代码。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。

ActionScript 提供了几个内置类，包括 MovieClip 类、Sound 类和其它一些类。您也可以创建自定义类，为您的应用程序定义各种对象类别。

ActionScript 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 Button 类的实例。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，您需要创建一个称为构造函数 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有关某个自行车车手的信息，您可以创建一个构造函数 `Biker()`，它具有 `time` 和 `distance` 属性，以及能够告诉您自行车车手行进速度的 `getSpeed()` 方法：

```
function Biker(t, d) {
  this.time = t;
  this.distance = d;
  this.getSpeed = function() {return this.time / this.distance;};
}
```

在此示例中，创建的函数需要两项信息或参数 才能完成工作：`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 对象的实例 `emma` 和 `hamish`，并使用以前的 ActionScript 中的 `getSpeed()` 方法跟踪 `emma` 实例的速度：

```
emma = new Biker(30, 5);
hamish = new Biker(40, 5);
trace(emma.getSpeed()); // 输出 6
```

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：String、Number、Boolean、Object、MovieClip 或 undefined。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 = 609;
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 Number。

在 ActionScript 1.0 中创建自定义对象

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

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该函数。例如，下面的代码就是一个构造函数，用于创建具有 radius 属性的圆：

```
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，然后将该对象分配给变量 myCircle：

```
myCircle = new Circle(5);
```

注意

对象的作用域与其被分配到的变量的作用域相同。

在 ActionScript 1.0 中将方法分配给自定义对象

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

您可以在对象的构造函数中定义对象的方法。但是，不建议采用这种方式，因为这种方式在每次使用构造函数时都会定义方法。下面的示例创建 `getArea()` 和 `getDiameter()` 方法：跟踪构造的半径设置为 55 的实例 `myCircle` 的面积和直径：

```
function Circle(radius) {
  this.radius = radius;
  this.getArea = function(){
    return Math.PI * this.radius * this.radius;
  };
  this.getDiameter = function() {
    return 2 * this.radius;
  };
}
var myCircle = new Circle(55);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
```

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的默认属性值。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，因为方法存在于一个位置，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 713 页的 “在 ActionScript 1.0 中创建继承”。）

下面的过程将演示如何将 `getArea()` 方法分配给自定义 Circle 对象。

将方法分配给自定义对象：

1. 定义构造函数 `Circle()`：

   ```
   function Circle(radius) {
     this.radius = radius;
   }
   ```

2. 定义 Circle 对象的 `getArea()` 方法。`getArea()` 方法计算圆的面积。在下面的示例中，可以使用函数文本来定义 `getArea()` 方法，并将 `getArea` 属性分配给该圆的原型对象：

   ```
   Circle.prototype.getArea = function () {
       return Math.PI * this.radius * this.radius;
   };
   ```

3. 下面的示例创建 Circle 对象的一个实例：

```
var myCircle = new Circle(4);
```

4. 使用下面的代码调用新的 myCircle 对象的 getArea() 方法：

```
var myCircleArea = myCircle.getArea();
trace(myCircleArea); // 输出 50.265……
```

ActionScript 将在 myCircle 对象中搜索 getArea() 方法。由于该对象没有 getArea() 方法，因此会在它的原型对象 Circle.prototype 中搜索 getArea()。ActionScript 找到该方法，调用该方法并跟踪 myCircleArea。

在 ActionScript 1.0 中定义事件处理函数方法

提示

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

可以为影片剪辑创建一个 ActionScript 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() 事件处理函数方法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() 方法不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. 创建一个影片剪辑元件，并通过以下方式将链接标识符设置为 theID：在 “库”面板中选择该元件，然后从 “库”弹出菜单中选择 “链接”。
2. 在 “动作”面板（“窗口” > “动作”）中，使用 function 语句定义一个新类，如下面的示例所示：

```
// 定义类
function myClipClass() {}
```

这个新类被分配给由时间轴添加到应用程序中（或者是用 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。

3. 输入代码，如下面的示例：

```
// 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. 输入如下例所示的代码，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. 选择要与新类相关联的元件，然后从 “库”面板弹出菜单中选择 “链接”。
7. 在 “链接属性”对话框中，选择 “为 ActionScript 导出”。
8. 在 “标识符”文本框中输入一个链接标识符。

   对于要与新类关联的所有元件，它们的链接标识符必须都相同。在 myClipClass 示例中，标识符是 theID。
9. 在 “动作”面板中输入如下面的示例所示的代码：

```
// 注册类
Object.registerClass("theID", myClipClass);
this.attachMovie("theID","myName",1);
```

   此步骤向 myClipClass 类注册了链接标识符为 theID 的元件。myClipClass 的所有实例都提供具有如步骤 4 中定义的行为的事件处理函数方法。同时这些实例的行为也与 MovieClip 类的所有实例相同，因为您在步骤 3 中指示新类继承 MovieClip 类。

   下面的示例显示了完整的代码：

```
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);
this.attachMovie("myclipID","clipName",3);
```

在 ActionScript 1.0 中创建继承

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

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，并添加其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`）。每个子类还具有其自己的属性和方法，这些属性和方法扩展了超类（例如，MountainBike 子类具有 `gears` 属性）。在 ActionScript 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的默认属性值。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 710 页的 “在 ActionScript 1.0 中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， ActionScript 会查看该对象，确定是否存在这样一个元素。如果不存在，ActionScript 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，ActionScript 就会查看 `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;
  this.pos = 0;
}
```

下面的代码将 `roll()` 方法添加到 Bike 类中：

```
Bike.prototype.roll = function() {return this.pos += 20;};
```

然后，可以使用下面的代码跟踪该自行车的位置：

```
var myBike = new Bike(55, "blue");
trace(myBike.roll());   // 输出 20。
trace(myBike.roll());   // 输出 40。
```

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，如下面的示例所示：

```
MountainBike.prototype = new Bike();
```

现在，可以调用 MountainBike 的 roll() 方法，如下面的示例所示：

```
var myKona = new MountainBike(20, "teal");
trace(myKona.roll());   // 输出 20
```

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() 给影片剪辑分配一个 MovieClip 类之外的类。

在 ActionScript 1.0 中向对象中添加 getter/setter 属性

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

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。您可以在一个位置（即原型对象中）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 函数的引用是最初引用的对象，而不是该原型对象。

测试模式中的 “调试” > “列出变量” 命令支持使用 Object.addProperty() 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 645 页的 “使用 “输出” 面板”。

在 ActionScript 1.0 中使用 Function 对象属性

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

使用 Function 对象的 `call()` 和 `apply()` 方法可以指定要将函数应用于的对象，以及要传递给该函数的参数值。ActionScript 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 ActionScript 1.0 中使用 Function.call() 方法调用函数

注意

许多 Flash 用户可以从使用 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

`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`，如下面的示例所示：

```
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，如下面的示例所示：

```
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)
```

使用 Function.call() 方法调用函数：

- 使用以下语法：

  `myFunction``.call(`*`thisObject`*`, `*`parameter1`*`, ..., `*`parameterN`*`)`

  该方法采用以下参数：

  - 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 ActionScript 1.0 中使用 Function.apply() 指定要将函数应用于的对象

注意

许多 Flash 用户可以从 ActionScript 2.0 中获益良多，特别是对于复杂应用程序更是如此。有关使用 ActionScript 2.0 的信息，请参见第 163 页的第 6 章 “类”。

`Function.apply()` 方法指定将在 ActionScript 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《ActionScript 2.0 语言参考》中的 `apply (Function.apply 方法)`。

使用 Function.apply() 指定要将函数应用于哪个对象：

- 使用以下语法：

  myFunction`.apply(`*thisObject*`, `*argumentsObject*`)`

  该方法采用以下参数：

  - 参数 *thisObject* 指定要将 *myFunction* 应用于的对象。
  - 参数 *argumentsObject* 定义一个数组，该数组的元素将作为参数传递给 *myFunction*。

附录 F

术语

F

和任何脚本撰写语言一样，ActionScript 使用自己的术语。Macromedia Flash 也使用专有术语。下面的列表对重要的 ActionScript 术语以及与使用 ActionScript 进行编程有关的 Flash 术语 （Flash 创作环境中的专用术语）进行介绍。

ActionScript 编辑器是 “动作”面板和 “脚本”窗口中的代码编辑器。 ActionScript 编辑器包含一些功能，如自动格式设置、显示隐藏字符和对脚本内容进行颜色编码。（另请参见: “脚本”窗口、“动作”面板）。

“动作”面板是 Flash 创作环境中的面板，可在其中编写 ActionScript 代码。

匿名函数是引用自身的未命名函数；创建匿名函数时就将引用该函数。有关信息和示例，请参见第 147 页的 “编写匿名函数和回调函数”。

锯齿是指带锯齿的文本，而带锯齿的文本不使用颜色变化使其锯齿边缘显得更平滑，这一点与消除锯齿文本不同 （请参见下面的定义）。

消除锯齿是指消除锯齿字符，可以使文本平滑，从而使显示在屏幕上的字符的边缘表现出较小的锯齿状。 Flash 中的 “消除锯齿”选项可以沿像素边界对齐文本轮廓，使文本更清楚，并且对于清晰呈现较小字体尤为有效。

数组是一些对象，其属性由表示对象在结构中位置的数字进行标识。实质上，数组是一系列项。

创作环境是 Flash 工作区，包括所有用户界面元素。可以使用创作环境创建 FLA 文件或脚本文件 （在 “脚本”窗口中）。

位图图形 （或光栅 图形）通常是照片级真实图像，或具有大量详细信息的图形。图像中的每个像素 （或位）都包含一个数据，这些位共同组成图像本身。位图可保存为 JPEG、 BMP 或 GIF 文件格式。与位图不同的另一个图像类型是 矢量。

布尔值为 `true` 或 `false` 值。

缓存是指应用程序中重复使用的信息，或是存储在计算机上以便重复使用的信息。例如，如果从 Internet 下载一个图像，通常对其进行缓存，这样您无需下载该图像的数据就可以重复查看。

回调函数是与某事件关联的匿名函数。函数将在特定事件发生后调用回调函数，如加载某些内容完毕之后 (onLoad()) 或在完成动画之后 (onMotionFinished())。有关更多信息和示例，请参见第 147 页的 “编写匿名函数和回调函数”。

字符是用来组成字符串的字母、数字和标点。有时也称为字型。

类是您可以创建的用来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”面板中编写的脚本中）使用 class 关键字。

类路径是指 Flash 在其中搜索类或接口定义的文件夹列表。创建类文件时，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。类路径存在于全局（应用程序）层和文档层中。

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。在比较值时常数很有用。

构造函数 是用于定义（初始化）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，以下代码定义了一个 Circle 类并实现一个构造函数：

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

在根据特定类创建（实例化）对象时，也会使用术语构造函数。以下语句是对内置 Array 类和自定义 Circle 类的构造函数的调用：

```
var my_array:Array = new Array();
var my_circle:Circle = new Circle(9);
```

数据类型描述变量或 ActionScript 元素可以包含的信息种类。内置的 ActionScript 数据类型包括 String、Number、Boolean、Object、MovieClip、Function、null 和 undefined。有关更多信息，请参见第 276 页的 “关于数据类型”。

设备字体是 Flash 中的特殊字体，未嵌入到 Flash SWF 文件中。相反，Flash Player 会使用本地计算机上与设备字体最相近的任何字体。由于未嵌入字体轮廓，因此 SWF 文件大小比使用嵌入的字体轮廓时更小。然而，由于未嵌入设备字体，因此使用这些字体创建的文本在未安装与设备字体相对应的字体的计算机系统上不能显示出预期效果。 Flash 包含三种设备字体：_sans（类似于 Helvetica 和 Arial 字体）、_serif（类似于 Times Roman 字体）和 _typewriter （类似于 Courier 字体）。

点语法是指在 ActionScript 中使用点 (.) 运算符 （点语法）访问属于舞台上的对象或实例的属性或方法。您还可以使用点运算符来确定实例 （例如影片剪辑）、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。

事件在播放 SWF 文件时发生。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鼠标单击时间或数据加载完成时间的特殊事件。 ActionScript 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 处理函数和 onClipEvent 处理函数，您可以将它们直接分配给按钮和影片剪辑。）在 “动作”面板中，每个具有事件处理函数方法或事件侦听器的 ActionScript 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。有关事件管理的更多信息，请参见第 255 页的 “处理事件”。

表达式是代表值的 ActionScript 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。

Flash Player **容器**是指保存 Flash 应用程序的系统，如浏览器或桌面应用程序。可以添加 ActionScript 和 JavaScript 来促进 Flash Player 容器和 SWF 文件之间的通信。

FlashType 是指 Flash 8 中的高级字体呈现技术。例如，可读性锯齿文本使用 FlashType 呈现技术，而动画锯齿文本则不使用。有关信息，请参见第 364 页的 “关于字体呈现和消除锯齿文本”。

帧脚本 是向时间轴上的某一帧中添加的代码块。

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 141 页的 “关于函数和方法”。

函数文本是可以用表达式（而不是语句）声明的未命名函数。在您需要临时使用一个函数，或者在您可以在代码中使用表达式代替函数时，函数文本非常有用。

IDE 是指“集成开发环境”(Integrated Development Environment)，IDE 是一种应用程序，开发人员可在此交互环境中编码、测试和调试应用程序。 Flash 创作工具有时也称为 IDE。

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 (_) 或美元符号 ($)。其后的字符必须是字母、数字、下划线或美元符号。例如，`firstName` 是一个变量的名称。

实例是包含某个特定类的所有属性和方法的对象。例如，所有数组都是 Array 类的实例，因此您可以将 Array 类的任何方法或属性用于任何数组实例。

实例名称是让您识别创建的实例或者舞台上的影片剪辑和按钮实例的唯一名称。例如，在以下代码中，“names”和 “studentName”是两个对象（一个数组对象和一个字符串对象）的实例名称：

```
var names:Array = new Array();
var studentName:String = new String();
```

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：

```
this.scorePlayer1_mc.score = 0;
this.scorePlayer2_mc.score = 0;
```

可以在创建实例时使用严格数据类型，以便在键入代码时显示代码提示。

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 不是合法的变量名称。有关关键字的列表，请参见第 86 页的 “关于关键字”和第 86 页的 “关于保留字”。

文本表示具有特定类型的值，如数字或字符串。文本并不存储在变量中。文本是直接出现在代码中的值，是 Flash 文档中的常数 （不变）值。另请参见函数文本 和字符串。

方法是与类关联的函数。例如，sortOn() 是与 Array 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，在下面的代码中，clear() 成为您前面定义的 controller 对象的方法：

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

下面的示例演示了创建类的方法：

```
// ActionScript 1.0 示例
A = new Object();
A.prototype.myMethod = function() {
  trace("myMethod");
}

// ActionScript 2.0 示例
class B {
  function myMethod() {
    trace("myMethod");
  }
}
```

命名函数是一种函数，通常在 ActionScript 代码中创建，用于执行各种动作。有关信息和示例，请参见第 146 页的 “编写命名函数”。

对象代码是附加到实例的 ActionScript。若要添加对象代码，请在舞台上选择一个实例，然后在 “动作”面板中键入代码。不应将代码附加到舞台上的对象。有关最佳做法的信息，请参见第 651 页的 “ActionScript 2.0 的最佳做法和编码约定”。

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 ActionScript 语言中预先定义的。例如，内置的 Date 类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。运算符操作的值称为操作数。

参数（也称参量）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 `firstName` 和 `hobby` 参数中接收到的两个值：

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

包是位于指定的类路径目录下并且包含一个或多个类文件的目录（请参见第 165 页的 “关于包”）。

固定脚本允许固定各种对象中的多个脚本并在 “动作”面板中同时使用这些脚本。此功能最好结合脚本导航器使用。

渐进式 JPEG 图像在从服务器下载的过程中逐渐构建并显示。通常，JPEG 图像在从服务器下载时是逐行显示的。

属性是定义对象的特性。例如，`length` 是所有数组的一个属性，它指定数组中的元素个数。

标点符号是帮助构成 ActionScript 代码的特殊字符。在 Flash 中有几种语言标点符号。最常用的标点符号种类有分号 (;)、冒号 (:)、小括号 [()] 和大括号 ({})。这些标点符号中的每一种在 Flash 语言中都有特殊含义，并可帮助定义数据类型、终止语句或构造 ActionScript。

脚本助手是“动作”面板中的新助手模式。通过“脚本助手”，无需具备丰富的 ActionScript 知识即可更加方便地创建脚本。脚本助手通过从“动作”面板的“动作”工具箱中选择项帮助您生成脚本，并提供一个由文本字段、单选按钮和复选框构成的界面，可以提示输入正确的变量及其它脚本语言构造。此功能类似于 Flash 创作工具早期版本中的标准模式。

“脚本”窗格是“动作”面板或“脚本”窗口中的窗格，是用于键入 ActionScript 代码的区域。

“脚本”窗口是一个代码编辑环境，可在此创建和修改外部脚本，如 Flash JavaScript 文件或 ActionScript 文件。例如，选择 “文件” > “新建”，然后选择 “ActionScript 文件”，使用 “脚本”窗口编写类文件。

语句是执行或指定动作的语言元素。例如，`return` 语句返回一个结果，作为执行它的函数的值。`if` 语句对一个条件求值，以确定应采取的下一个动作。`switch` 语句创建 ActionScript 语句的分支结构。

字符串是字符和数据类型的序列。有关更多信息，请参见第 404 页的 “关于字符串和 String 类”。

字符串是直引号字符包含的字符的序列。字符本身是数据值，而不是对数据的引用。字符串并不是 String 对象。有关更多信息，请参见第 404 页的 “关于字符串和 String 类”。

Surface 是其位图缓存标志打开的影片剪辑。有关位图缓存的信息，请参见第 333 页的“缓存影片剪辑”。

语法是指编程所用语言的语法和拼写方式。编译器无法识别不正确的语法，因此，当您尝试在测试环境中测试文档时，会在“输出”面板中看到错误或警告。因此，语法是帮助您构成正确 ActionScript 的规则和准则的集合。

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命名影片剪辑实例。（主时间轴名称始终为 `_root`。）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或属性的值。例如，下面的语句是名为 `stereoControl` 的对象的 `volume` 属性的目标路径：

```
stereoControl.volume
```

文本是可在文本字段或用户界面组件中显示的一个或多个字符串系列。

文本字段是舞台上的可见元素，要通过它向用户显示文本，可使用文本工具或使用 ActionScript 代码创建。Flash 允许将文本字段设置为可编辑（只读）、允许 HTML 格式设置、启用多行支持、密码遮罩或将 CSS 样式表应用于 HTML 格式文本。

文本格式设置可应用于文本字段，或文本字段中的某些字符。可应用于文本的一些文本格式设置选项示例有：对齐、缩进、粗体、颜色、字体大小、边距宽度、斜体和字母间距。

顶级函数是不属于类的函数（有时称为预定义 或内置函数），这意味着无需构造函数即可调用这些函数。内置在 ActionScript 语言顶级的函数的示例包括 `trace()` 和 `setInterval()`。

用户定义的函数是指用户自己创建的在应用程序中使用的函数，与执行预定义函数的内置类中的函数相反。您可以自己命名函数，并在函数块内添加语句。

变量是包含任何数据类型值的标识符。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:Number = 5;
var name:String = "Lolo";
var c_color:Color = new Color(mcinstanceName);
```

有关变量的更多信息，请参见第 288 页的 “关于变量”。

矢量图形使用称作矢量的直线和曲线描述图像，还包括颜色和位置属性。每个矢量均使用数学计算（而不是位）描述形状，这样在对其进行缩放时不会使质量下降。另一个图形类型是位图，由点或像素表示。

索 引

符号

数字

英文

A

B

C

D

F

G

H

J

K

L

T

W

X

Y

Z