ACTIONSCRIPT® 3.0

开发人员指南

法律声明

有关法律声明，请参阅 http://help.adobe.com/zh_CN/legalnotices/index.html。

上次更新 2015/10/14

目录

第 1 章 : 使用日期和时间

管理日历日期和时间 1

控制时间间隔 3

日期和时间示例：简单模拟时钟 5

第 2 章 : 使用字符串

字符串基础知识 9

创建字符串 10

length 属性 11

处理字符串中的字符 11

比较字符串 12

获取其他对象的字符串表示形式 12

连接字符串 12

在字符串中查找子字符串和模式 13

转换字符串的大小写 16

字符串示例：ASCII 图表 17

第 3 章 : 使用数组

数组基础知识 21

索引数组 22

关联数组 31

多维数组 33

克隆数组 35

扩展 Array 类 35

数组示例：播放列表 39

第 4 章 : 处理错误

错误处理基础知识 43

错误类型 44

ActionScript 3.0 中的错误处理 45

使用 Flash 运行时的调试版 47

在应用程序中处理同步错误 47

创建自定义错误类 51

响应错误事件和状态 52

比较错误类 55

处理错误示例：CustomErrors 应用程序 57

第 5 章 : 使用正则表达式

正则表达式基础知识 63

正则表达式语法 64

对字符串使用正则表达式的方法 75
正则表达式示例：Wiki 解析程序 76

第 6 章 : 使用 XML
XML 基础知识 80
用于处理 XML 的 E4X 方法 82
XML 对象 84
XMLList 对象 86
初始化 XML 变量 87
组合和变换 XML 对象 88
遍历 XML 结构 89
使用 XML 命名空间 93
XML 类型转换 94
读取外部 XML 文档 96
在 ActionScript 中使用 XML 的示例：从 Internet 加载 RSS 数据 96

第 7 章 : 使用本机 JSON 功能
JSON API 概述 100
定义自定义 JSON 行为 101

第 8 章 : 处理事件
事件处理基础知识 106
ActionScript 3.0 事件处理与早期版本事件处理的不同之处 107
事件流 109
事件对象 110
事件侦听器 113
事件处理示例：闹钟 118

第 9 章 : 使用应用程序域

第 10 章 : 显示编程
显示编程的基础知识 126
核心显示类 128
显示列表方法的优点 130
使用显示对象 132
处理显示对象 144
对象动画 160
舞台方向 161
动态加载显示内容 164
显示对象示例：SpriteArranger 169

第 11 章 : 使用几何结构
几何结构基础知识 175
使用 Point 对象 176

使用 Rectangle 对象 177
使用 Matrix 对象 180
几何形状示例：对显示对象应用矩阵转换 181

第 12 章 : 使用绘图 API
绘制 API 的基础 185
Graphics 类 186
绘制线条和曲线 186
使用内置方法绘制形状 188
创建渐变线条和填充 189
将 Math 类与绘制方法配合使用 193
使用绘图 API 进行动画处理 193
绘制 API 示例：算法可视化生成器 194
绘图 API 高级用法 196

第 13 章 : 使用位图
位图使用基本知识 203
Bitmap 和 BitmapData 类 204
处理像素 206
复制位图数据 208
压缩位图数据 209
使用杂点功能制作纹理 209
滚动位图 211
利用 mipmap 处理 212
位图示例：带动画效果的旋转的月亮 213
位图图像的异步解码 221

第 14 章 : 过滤显示对象
过滤显示对象的基础知识 224
创建和应用滤镜 224
可用的显示滤镜 230
筛选显示对象示例：Filter Workbench 244

第 15 章 : 使用 Pixel Bender 着色器
Pixel Bender 着色器基础知识 251
加载或嵌入着色器 253
访问着色器元数据 254
指定着色器输入和参数值 254
使用着色器 259

第 16 章 : 使用影片剪辑
影片剪辑基础知识 270
使用 MovieClip 对象 270
控制影片剪辑播放 271

使用 ActionScript 创建 MovieClip 对象 273
加载外部 SWF 文件 275
影片剪辑示例：RuntimeAssetsExplorer 277

第 17 章 : 使用补间动画
补间动画基础知识 281
在 Flash 中复制补间动画脚本 281
合并补间动画脚本 282
描述动画 283
添加滤镜 285
将补间动画与其显示对象关联 287

第 18 章 : 使用反向运动
反向运动的基础知识 288
IK 骨架动画处理概述 289
获取有关 IK 骨架的信息 290
实例化 IKMover 并限制其移动 291
移动 IK 骨架 291
使用弹簧 292
使用 IK 事件 292

第 19 章 : 在三维 (3D) 环境中工作
3D 显示对象的基础知识 294
了解 Flash Player 和 AIR 运行时中的 3D 显示对象 295
创建和移动 3D 显示对象 296
将 3D 对象投影到 2D 视图上 298
示例：透视投影 299
执行复杂的 3D 转换 301
通过三角形获得 3D 效果 304

第 20 章 : 文本使用基础知识

第 21 章 : 使用 TextField 类
显示文本 313
选择和操作文本 316
捕获文本输入 317
限制文本输入 318
设置文本格式 319
高级文本呈现 322
使用静态文本 324
TextField 示例：报纸风格的文本格式设置 325

第 22 章 : 使用 Flash 文本引擎

创建和显示文本 334
处理 FTE 中的事件 338
设置文本格式 342
使用字体 345
控制文本 347
Flash 文本引擎示例：新闻版面布局 352

第 23 章 : 使用 Text Layout Framework

Text Layout Framework 概述 360
使用 Text Layout Framework 361
使用 TLF 构建文本结构 365
使用 TLF 设置文本格式 369
使用 TLF 导入和导出文本 370
使用 TLF 管理文本容器 370
使用 TLF 启用文本选择、编辑和撤消 371
使用 TLF 处理事件 372
在文本内定位图像 372

第 24 章 : 处理声音

声音处理基础知识 373
了解声音体系结构 374
加载外部声音文件 375
处理嵌入的声音 377
处理声音流文件 378
处理动态生成的音频 379
播放声音 381
加载和播放声音时的安全注意事项 384
控制音量和声相 384
处理声音元数据 386
访问原始声音数据 386
捕获声音输入 389
声音示例：Podcast Player 394

第 25 章 : 使用视频

视频基础知识 401
了解视频格式 402
了解 Video 类 405
加载视频文件 405
控制视频播放 406
在全屏模式下播放视频 408
流式传输视频文件 411
了解提示点 411

编写元数据和提示点的回调方法 412
使用提示点和元数据 416
监控 NetStream 活动 425
视频文件的高级主题 428
视频示例：视频自动唱片点唱机 429
使用 StageVideo 类来实现硬件加速呈现 433

第 26 章 : 使用摄像头

了解 Camera 类 441
在屏幕上显示摄像头内容 441
设计摄像头应用程序 442
连接至用户的摄像头 442
验证是否已安装摄像头 443
检测摄像头的访问权限 444
最优化摄像头视频品质 445
监控摄像头状态 446

第 27 章 : 使用数字权限管理

了解受保护的内容工作流程 448
NetStream 类中与 DRM 相关的成员和事件 454
使用 DRMStatusEvent 类 456
使用 DRMAuthenticateEvent 类 457
使用 DRMErrorEvent 类 460
使用 DRMManager 类 461
使用 DRMContentData 类 462
更新 Flash Player 以支持 Adobe Access 462
带外许可证 463
域支持 465
使用域支持播放加密的内容 465
许可证预览 465
提交内容 466
Open Source Media Framework 466

第 28 章 : 在 AIR 中添加 PDF 内容

检测 PDF 功能 469
加载 PDF 内容 470
编写 PDF 内容的脚本 470
对 AIR 中的 PDF 内容的已知限制 472

第 29 章 : 用户交互的基础知识

捕获用户输入 473
管理焦点 473
了解输入类型 475

第 30 章 : 键盘输入

捕获键盘输入 477
使用 IME 类 479
虚拟键盘 483

第 31 章 : 鼠标输入

捕获鼠标输入 490
鼠标输入示例：WordSearch 492

第 32 章 : 触摸、多点触控和手势输入

触摸输入的基础知识 496
触摸支持发现 498
Touch 事件处理 499
触摸和拖动 502
Gesture 事件处理 503
疑难解答 506

第 33 章 : 复制和粘贴

复制粘贴基础知识 509
读取和写入系统剪贴板 509
AIR 中的 HTML 复制和粘贴 510
剪贴板数据格式 512

第 34 章 : 加速计输入

检查加速计支持 517
检测加速计更改 517

第 35 章 : AIR 中的拖放

AIR 中拖放的基础知识 519
支持拖出手势 521
支持拖入手势 523
HTML 中的拖放 525
将数据拖出 HTML 元素 528
将数据拖入 HTML 元素 529
示例：覆盖默认的 HTML 拖入行为 529
在非应用程序 HTML 沙箱中处理文件放置 531
放置文件释放 532

第 36 章 : 使用菜单

菜单基础知识 540
创建本机菜单 (AIR) 546
关于 HTML 中的上下文菜单 (AIR) 548
显示弹出本机菜单 (AIR) 549

处理菜单事件 549
本机菜单示例：窗口和应用程序菜单 (AIR) 551

第 37 章 : AIR 中的任务栏图标
关于任务栏图标 554
停靠栏图标 554
系统任务栏图标 555
Window 任务栏图标和按钮 557

第 38 章 : 使用文件系统
使用 FileReference 类 559
使用 AIR 文件系统 API 571

第 39 章 : 存储本地数据
共享对象 602
加密的本地存储区 610

第 40 章 : 在 AIR 中使用本地 SQL 数据库
关于本地 SQL 数据库 613
创建和修改数据库 616
操作 SQL 数据库数据 622
使用同步和异步数据库操作 645
对 SQL 数据库使用加密 649
使用 SQL 数据库的策略 664

第 41 章 : 使用字节数组
读取并写入 ByteArray 666
ByteArray 示例：读取 .zip 文件 671

第 42 章 : 网络和通信基础知识
网络接口 677
网络连接更改 678
域名系统 (DNS) 记录 681

第 43 章 : 套接字
TCP 套接字 683
UDP 套接字 (AIR) 693
IPv6 地址 694

第 44 章 : HTTP 通信
加载外部数据 696
Web 服务请求 703
在其他应用程序中打开 URL 710

第 45 章 : 与其他 Flash Player 和 AIR 实例通信
关于 LocalConnection 类 712
在两个应用程序之间发送消息 714
连接到不同域中的内容和 AIR 应用程序 715

第 46 章 : 与 AIR 中的本机进程通信
本机进程通信概述 717
启动和关闭本机进程 718
与本机进程通信 718
本机进程通信的安全性注意事项 720

第 47 章 : 使用外部 API
使用外部 API 的基础知识 721
外部 API 要求和优点 723
使用 ExternalInterface 类 724
外部 API 示例：在 ActionScript 和 Web 浏览器中的 JavaScript 之间进行通信 727

第 48 章 : AIR 中的 XML 签名验证
XML 签名验证的基础知识 733
关于 XML 签名 736
实现 IURIDereferencer 接口 738

第 49 章 : 客户端系统环境
客户端系统环境基础知识 745
使用 System 类 746
使用 Capabilities 类 747
功能示例：检测系统功能 747

第 50 章 : AIR 应用程序的调用和终止
应用程序调用 751
捕获命令行参数 752
用户登录时调用 AIR 应用程序 755
从浏览器调用 AIR 应用程序 756
应用程序终止 757

第 51 章 : 处理 AIR 运行时和操作系统信息
管理文件关联 759
获取运行时版本和修补级别 759
检测 AIR 功能 760
跟踪用户当前状态 760

第 52 章 : 使用 AIR 本机窗口
AIR 中的本机窗口的基础知识 761
创建窗口 767
管理窗口 775

侦听窗口事件 782
显示全屏窗口 783

第 53 章 : AIR 中的显示屏幕
AIR 中的显示屏幕的基础知识 785
枚举屏幕 786

第 54 章 : 打印
打印基础知识 789
打印页面 789
Flash 运行时任务和系统打印 790
设置大小、缩放和方向 792
高级打印技术 794
打印示例：多页面打印 796
打印示例：缩放、裁剪和响应 798
打印示例：页面设置和打印选项 799

第 55 章 : Geolocation
检测 geolocation 更改 802

第 56 章 : 应用程序国际化
应用程序国际化基础知识 805
flash.globalization 包概述 806
确定区域设置 807
设置数字格式 808
设置货币值格式 810
设置日期和时间格式 812
排序和比较字符串 813
大小写转换 814
示例：国际化股票报价应用程序 815

第 57 章 : 本地化应用程序
选择区域设置 819
本地化 Flex 内容 819
本地化 Flash 内容 820
本地化 AIR 应用程序 820
对日期、时间和货币进行本地化 820

第 58 章 : 关于 HTML 环境
HTML 环境概述 823
AIR 和 WebKit 825

第 59 章 : 在 AIR 中进行 HTML 和 JavaScript 编程
关于 HTMLLoader 类 839
避免与安全相关的 JavaScript 错误 841

通过 JavaScript 访问 AIR API 类 845
关于 AIR 中的 URL 846
使 ActionScript 对象可用于 JavaScript 847
从 ActionScript 访问 HTML DOM 和 JavaScript 对象 848
在 HTML 中嵌入 SWF 内容 849
在 HTML 页中使用 ActionScript 库 852
转换 Date 和 RegExp 对象 853
从 ActionScript 操作 HTML 样式表 854
跨脚本访问不同安全沙箱中的内容 855

第 60 章 : 为 AIR HTML 容器编写脚本
HTMLLoader 对象的显示属性 859
滚动 HTML 内容 861
访问 HTML 历史记录列表 862
设置在加载 HTML 内容时使用的用户代理 863
设置用于 HTML 内容的字符编码 863
为 HTML 内容定义类似于浏览器的用户界面 863
创建 HTMLLoader 类的子类 872

第 61 章 : 处理 AIR 中与 HTML 相关的事件
HTMLLoader 事件 874
使用 ActionScript 处理 DOM 事件 874
响应未捕获的 JavaScript 异常 875
使用 JavaScript 处理运行时事件 877

第 62 章 : 在移动应用程序中显示 HTML 内容
StageWebView 对象 880
内容 880
导航事件 882
历史记录 883
焦点 884
位图捕获 886

第 63 章 : 将 worker 用于并发
了解 worker 与并发 888
创建和管理 worker 889
在 worker 之间进行通信 891

第 64 章 : 安全性
Flash Platform 安全概述 894
安全沙箱 895
权限控制 899
限制网络 API 905
全屏模式安全性 907

全屏交互模式安全性 908
加载内容 908
跨脚本访问 911
作为数据访问加载的媒体 913
加载数据 915
从导入到安全域的 SWF 文件加载嵌入内容 918
使用旧内容 918
设置 LocalConnection 权限 919
控制外出 URL 访问 919
共享对象 920
摄像头、麦克风、剪贴板、鼠标和键盘访问 921
AIR 安全性 922

第 65 章 : 如何使用 ActionScript 示例

示例类型 939
在 Flash Professional 中运行 ActionScript 3.0 示例 940
在 Flash Builder 中运行 ActionScript 3.0 示例 942
在移动设备上运行 ActionScript 3.0 示例 943

第 66 章 : 本地数据库中的 SQL 支持

支持的 SQL 语法 946
数据类型支持 963

第 67 章 : SQL 错误详细消息、 ID 和参数

第 68 章 : Adobe 图形汇编语言 (AGAL)

AGAL 字节码格式 970

第 1 章：使用日期和时间

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

时间可能并不代表一切，但它在软件应用程序中通常是一个关键要素。 ActionScript 3.0 提供了多种强大的手段来管理日历日期、时间和时间间隔。以下两个主类提供了大部分的计时功能：Date 类和 flash.utils 包中的新 Timer 类。

日期和时间是在 ActionScript 程序中使用的一种常见信息类型。例如，您可能需要了解当前星期值，或测量用户在特定屏幕上花费多少时间，并且还可能会执行很多其他操作。在 ActionScript 中，可以使用 Date 类来表示某一时刻，其中包含日期和时间信息。 Date 实例中包含各个日期和时间单位的值，其中包括年、月、日、星期、小时、分钟、秒、毫秒以及时区。对于更高级的用法， ActionScript 还包括 Timer 类，您可以使用该类在一定延迟后执行动作，或按重复间隔执行动作。

更多帮助主题

Date

flash.utils.Timer

管理日历日期和时间

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 的所有日历日期和时间管理函数都集中在顶级 Date 类中。Date 类包含一些方法和属性，这些方法和属性能够使您按照通用协调时间 (UTC) 或特定于时区的本地时间来处理日期和时间。UTC 是一种标准时间定义，它实质上与格林尼治标准时间 (GMT) 相同。

创建 Date 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Date 类是所有核心类中构造函数方法形式最为多变的类之一。您可以用以下四种方式来调用 Date 类。

第一，如果未给定参数，则 Date() 构造函数将按照您所在时区的本地时间返回包含当前日期和时间的 Date 对象。这里提供了一个示例：

```
var now:Date = new Date();
```

第二，如果仅给定了一个数字参数，则 Date() 构造函数将其视为自 1970 年 1 月 1 日以来经过的毫秒数，并且返回对应的 Date 对象。请注意，您传入的毫秒值将被视为自 1970 年 1 月 1 日 （UTC 时间）以来经过的毫秒数。但是，该 Date 对象会按照您所在的本地时区来显示值，除非您使用特定于 UTC 的方法来检索和显示这些值。如果仅使用一个毫秒参数来创建新的 Date 对象，则应确保考虑到您的当地时间和 UTC 之间的时区差异。以下语句创建一个设置为 1970 年 1 月 1 日午夜（UTC 时间）的 Date 对象：

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// gets a Date one day after the start date of 1/1/1970
var startTime:Date = new Date(millisecondsPerDay);
```

第三，您可以将多个数值参数传递给 Date() 构造函数。该构造函数将这些参数分别视为年、月、日、小时、分钟、秒和毫秒，并将返回一个对应的 Date 对象。假定这些输入参数采用的是本地时间而不是 UTC。以下语句获取一个设置为 2000 年 1 月 1 日开始的午夜 （本地时间）的 Date 对象：

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

第四，您可以将单个字符串参数传递给 Date() 构造函数。该构造函数将尝试把字符串分析为日期或时间部分，然后返回对应的 Date 对象。如果使用此方法，最好将 Date() 构造函数包含在 try..catch 块中，以捕获所有分析错误。 Date() 构造函数可接受多种不同的字符串格式 （用于 Adobe Flash Platform 的 ActionScript 3.0 参考中列出了这些格式）。以下语句使用字符串值初始化一个新的 Date 对象：

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

如果 Date() 构造函数无法成功分析该字符串参数，它不会引发异常。但是，所得到的 Date 对象将包含一个无效的日期值。

获取时间单位值

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 Date 类的属性或方法从 Date 对象中提取各种时间单位的值。下面的每个属性为您提供了 Date 对象中的一个时间单位的值：

- fullYear 属性
- month 属性，以数字格式表示，分别以 0 到 11 表示一月到十二月
- date 属性，表示月中某一天的日历数字，范围为 1 到 31
- day 属性，以数字格式表示一周中的某一天，其中 0 表示星期日
- hours 属性，范围为 0 到 23
- minutes 属性
- seconds 属性
- milliseconds 属性

实际上， Date 类为您提供了获取这些值的多种方式。例如，您可以用四种不同方式获取 Date 对象的月份值：

- month 属性
- getMonth() 方法
- monthUTC 属性
- getMonthUTC() 方法

所有四种方式实质上具有同等的效率，因此您可以任意使用一种最适合应用程序的方法。

刚才列出的属性表示总日期值的各个部分。例如， milliseconds 属性永远不会大于 999，因为当它达到 1000 时，秒钟值就会增加 1 并且 milliseconds 属性会重置为 0。

如果要获取 Date 对象自 1970 年 1 月 1 日 (UTC) 起所经过毫秒数的值，可以使用 getTime() 方法。通过使用与其相对应的 setTime() 方法，可以使用自 1970 年 1 月 1 日 (UTC) 起所经过的毫秒数更改现有 Date 对象的值。

执行日期和时间运算

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以使用 Date 类对日期和时间执行加法和减法运算。日期值在内部以毫秒的形式保存，因此您应将其他值转换成毫秒，然后再将它们与 Date 对象进行加减。

如果应用程序将执行大量的日期和时间运算，您可能会发现创建常量来保存常见时间单位值 （以毫秒的形式）非常有用，如下所示：

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

现在，可以方便地使用标准时间单位来执行日期运算。下列代码使用 getTime() 和 setTime() 方法将日期值设置为当前时间一个小时后的时间：

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

设置日期值的另一种方式是仅使用一个毫秒参数创建新的 Date 对象。例如，下列代码将一个日期加上 30 天以计算另一个日期：

```
// sets the invoice date to today's date
var invoiceDate:Date = new Date();

// adds 30 days to get the due date
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

接着，将 millisecondsPerDay 常量乘以 30 以表示 30 天的时间，将结果与 invoiceDate 值相加，用于设置 dueDate 值。

在时区之间进行转换

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在需要将日期从一种时区转换成另一种时区时，使用日期和时间运算十分方便。也可以使用 getTimezoneOffset() 方法，该方法返回的值表示 Date 对象的时区与 UTC 之间相差的分钟数。此方法之所以返回以分钟为单位的值，是因为并不是所有时区之间都正好相差一个小时，有些时区与邻近的时区仅相差半个小时。

以下示例使用时区偏移量将日期从本地时间转换成 UTC。该示例首先以毫秒为单位计算时区值，然后按照该量调整 Date 值：

```
// creates a Date in local time
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// converts the Date to UTC by adding or subtracting the time zone offset
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

控制时间间隔

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Adobe Flash CS4 Professional 开发应用程序时，您可以访问时间轴，这会使您稳定且逐帧地完成该应用程序。但在纯 ActionScript 项目中，您必须依靠其他计时机制。

循环与计时器之比较

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在某些编程语言中，必须使用循环语句 （如 for 或 do..while）来设计自己的计时方案 while.

通常，循环语句会以本地计算机所允许的速度尽可能快地执行，这表明应用程序在某些计算机上的运行速度较快而在其他计算机上则较慢。如果应用程序需要一致的计时间隔，则您需要将其与实际的日历或时钟时间联系在一起。许多应用程序 （如游戏、动画和实时控制器）需要在不同计算机上均能保持一致的、规则的时间驱动计时机制。

ActionScript 3.0 的 Timer 类提供了一个功能强大的解决方案。使用 ActionScript 3.0 事件模型， Timer 类在每次达到指定的时间间隔时都会调度计时器事件。

Timer 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中处理计时函数的首选方式是使用 Timer 类 (flash.utils.Timer)，可以使用它在每次达到间隔时调度事件。

要启动计时器，请先创建 Timer 类的实例，并告诉它每隔多长时间生成一次计时器事件以及在停止前生成多少次事件。

例如，下列代码创建一个每秒调度一个事件且持续 60 秒的 Timer 实例：

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Timer 对象在每次达到指定的间隔时都会调度 TimerEvent 对象。 TimerEvent 对象的事件类型是 timer （由常量 TimerEvent.TIMER 定义）。 TimerEvent 对象包含的属性与标准 Event 对象包含的属性相同。

如果将 Timer 实例设置为固定的间隔数，则在达到最后一次间隔时，它还会调度 timerComplete 事件 （由常量 TimerEvent.TIMER_COMPLETE 定义）。

以下是一个用来展示 Timer 类实际操作的小示例应用程序：

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // creates a new five-second Timer
            var minuteTimer:Timer = new Timer(1000, 5);

            // designates listeners for the interval and completion events
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE, onTimerComplete);

            // starts the timer ticking
            minuteTimer.start();
        }

        public function onTick(event:TimerEvent):void
        {
            // displays the tick count so far
            // The target of this event is the Timer instance itself.
            trace("tick " + event.target.currentCount);
        }

        public function onTimerComplete(event:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

创建 ShortTimer 类时，它会创建一个用于每秒计时一次并持续五秒的 Timer 实例。然后，它将两个侦听器添加到计时器：一个用于侦听每次计时，另一个用于侦听 timerComplete 事件。

接着，它启动计数器计时，并且从此时起以一秒钟的间隔执行 onTick() 方法。

onTick() 方法只显示当前的时间计数。五秒钟后，执行 onTimerComplete() 方法，告诉您时间已到。

运行该示例时，您应会看到下列行以每秒一行的速度显示在控制台或跟踪窗口中：

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

flash.utils 包中的计时函数

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 包含许多与 ActionScript 2.0 提供的计时函数类似的计时函数。这些函数是作为 flash.utils 包中的包级别函数提供的，它们的功能与 ActionScript 2.0 中完全相同。

函数	说明
clearInterval(id:uint):void	取消指定的 setInterval() 调用。
clearTimeout(id:uint):void	取消指定的 setTimeout() 调用。
getTimer():int	返回自 Adobe® Flash® Player 或 Adobe® AIR™ 初始化以来经过的毫秒数。
setInterval(closure:Function, delay:Number, ... arguments):uint	以指定的间隔 （以毫秒为单位）运行函数。
setTimeout(closure:Function, delay:Number, ... arguments):uint	在指定的延迟 （毫秒）后运行指定的函数。

这些函数仍保留在 ActionScript 3.0 以实现向后兼容。 Adobe 不建议您在新的 ActionScript 3.0 应用程序中使用这些函数。通常，在应用程序中使用 Timer 类会更容易且更有效。

日期和时间示例：简单模拟时钟

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

简单的模拟时钟示例说明了这两个日期和时间概念：

- 获取当前日期和时间并提取小时、分钟和秒的值
- 使用 Timer 设置应用程序的运行速度

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/SimpleClock 文件夹中找到 SimpleClock 应用程序文件。该应用程序包含以下文件：

文件	说明
SimpleClockApp.mxml 或 SimpleClockApp.fla	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/simpleclock/SimpleClock.as	主应用程序文件。
com/example/programmingas3/simpleclock/AnalogClockFace.as	根据时间绘制一个圆形的钟面以及时针、分针和秒针。

定义 SimpleClock 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此时钟示例很简单，但是将即使很简单的应用程序也组织得十分有条理是一种很好的做法，以便您将来能够很轻松地扩展这些应用程序。为此， SimpleClock 应用程序使用 SimpleClock 类处理启动和时间保持任务，然后使用另一个名称为 AnalogClockFace 的类来实际显示该时间。

以下代码用于定义和初始化 SimpleClock 类 （请注意，在 Flash 版本中， SimpleClock 扩展了 Sprite 类）：

```
public class SimpleClock extends UIComponent
{
    /**
     * The time display component.
     */
    private var face:AnalogClockFace;

    /**
     * The Timer that acts like a heartbeat for the application.
     */
    private var ticker:Timer;
```

该类具有两个重要的属性：

- face 属性，它是 AnalogClockFace 类的实例
- ticker 属性，它是 Timer 类的实例

SimpleClock 类使用默认构造函数。 initClock() 方法处理实际的设置工作，它创建钟面并启动 Timer 实例的计时。

创建钟面

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

SimpleClock 代码中后面的几行代码创建用于显示时间的钟面：

```
    /**
     * Sets up a SimpleClock instance.
     */
    public function initClock(faceSize:Number = 200)
    {
        // creates the clock face and adds it to the display list
        face = new AnalogClockFace(Math.max(20, faceSize));
        face.init();
        addChild(face);

        // draws the initial clock display
        face.draw();
```

可以将钟面的大小传递给 initClock() 方法。如果未传递 faceSize 值，则使用 200 个像素的默认大小。

接着，应用程序将钟面初始化，然后使用从 DisplayObjectContainer 类继承的 addChild() 方法将该钟面添加到显示列表。然后，它调用 AnalogClockFace.draw() 方法显示一次钟面，同时显示当前时间。

启动计时器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

创建钟面后， initClock() 方法会设置一个计时器：

```
        // creates a Timer that fires an event once per second
        ticker = new Timer(1000);

        // designates the onTick() method to handle Timer events
        ticker.addEventListener(TimerEvent.TIMER, onTick);

        // starts the clock ticking
        ticker.start();
```

首先，该方法初始化一个每秒 （每隔 1000 毫秒）调度一次事件的 Timer 实例。由于没有向 Timer() 构造函数传递第二个 repeatCount 参数，因此 Timer 将无限期地重复计时。

SimpleClock.onTick() 方法将在每秒收到 timer 事件时执行一次。

```
    public function onTick(event:TimerEvent):void
    {
        // updates the clock display
        face.draw();
    }
```

AnalogClockFace.draw() 方法仅绘制钟面和指针。

显示当前时间

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

AnalogClockFace 类中大多数代码都与设置钟面的显示元素有关。 AnalogClockFace 在进行初始化时，会绘制一个圆形轮廓，将数字文本标签放在每个小时刻度处，然后创建三个 Shape 对象，分别表示时钟的时针、分针和秒针。

在 SimpleClock 应用程序运行后，它会每秒调用一次 AnalogClockFace.draw() 方法，如下所示：

```
    /**
     * Called by the parent container when the display is being drawn.
     */
    public override function draw():void
    {
        // stores the current date and time in an instance variable
        currentTime = new Date();
        showTime(currentTime);
    }
```

此方法将当前时间保存在变量中，因此在绘制时钟指针的过程中无法改变时间。然后，调用 showTime() 方法以显示指针，如下所示：

```
/**
 * Displays the given Date/Time in that good old analog clock style.
 */
public function showTime(time:Date):void
{
    // gets the time values
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // multiplies by 6 to get degrees
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // Multiply by 30 to get basic degrees, then
    // add up to 29.5 degrees (59 * 0.5)
    // to account for the minutes.
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

首先，此方法提取当前时间的小时、分钟和秒的值。然后使用这些值来计算每个指针的角度。由于秒针会在 60 秒内旋转一圈，因此它每秒都会旋转 6 度 (360/60)。分针每分钟都旋转同样的度数。

时针每分钟都在更新，因此时针能够随着分针的跳动显示出某些时间变化过程。时针每小时旋转 30 度 (360/12)，但也会每分钟旋转半度 （30 度除以 60 分钟）。

第 2 章：使用字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

String 类包含使您能够使用文本字符串的方法。在使用许多对象时，字符串都十分重要。此处介绍的这些方法对使用在对象中（如 TextField、 StaticText、 XML、 ContextMenu 和 FileReference 对象）使用的字符串很有帮助。

字符串是字符的序列。 ActionScript 3.0 支持 ASCII 字符和 Unicode 字符。

更多帮助主题

String

RegExp

parseFloat()

parseInt()

字符串基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在编程语言中，字符串是指一个文本值，即串在一起而组成单个值的一系列字母、数字或其他字符。例如，以下一行代码创建一个数据类型为 String 的变量，并为该变量赋予一个文本字符串值：

```
var albumName:String = "Three for the money";
```

正如此示例所示，在 ActionScript 中，可使用双引号或单引号将本文引起来以表示字符串值。以下是另外几个字符串示例：

```
"Hello"
"555-7649"
"http://www.adobe.com/"
```

每当在 ActionScript 中使用一段文本时，您都会用到字符串值。 ActionScript String 类是一种可用来使用文本值的数据类型。 String 实例通常用于很多其他 ActionScript 类中的属性、方法参数等。

重要概念和术语

以下参考列表包含您会遇到的与字符串有关的重要术语：

ASCII 在计算机程序中用于表示文本字符和符号的系统。 ASCII 系统支持 26 个字母英文字母表，以及有限的一组其他字符。

字符 文本数据的最小单位 （单个字母或符号）。

连接 通过将一个字符串值添加到其他字符串值的末端，将多个字符串值连接到一起，从而创建新的字符串值。

空字符串 不包含文本、空白或其他字符的字符串，可以写为 ""。空字符串值不同于具有 null 值的 String 变量；值为 null 的 String 变量是指没有赋予 String 实例的变量，而空字符串则包含一个实例，其值不包含任何字符。

String 文本值 （字符序列）。

字符串文本 （或 " 文本字符串 "） 在代码中明确编写的字符串值，书写方式为由双引号或单一引号括起来的文本值。

子字符串 作为其他字符串一部分的字符串。

Unicode 在计算机程序中用于表示文本字符和符号的标准系统。 Unicode 系统允许使用任何编写系统中的任何字符。

创建字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中， String 类用于表示字符串 （文本）数据。 ActionScript 字符串支持 ASCII 字符和 Unicode 字符。创建字符串的最简单方式是使用字符串文本。要声明字符串文本，请使用双直引号 (") 或单直引号 (') 字符。例如，以下两个字符串是等效的：

```
var str1:String = "hello";
var str2:String = 'hello';
```

您还可以使用 new 运算符来声明字符串，如下所示：

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 == ""
```

下面的两个字符串是等效的：

```
var str1:String = "hello";
var str2:String = new String("hello");
```

若要在使用单引号 (') 分隔符定义的字符串文本内使用单引号 (')，请使用反斜杠转义符 (\)。类似地，要在使用双引号 (") 分隔符定义的字符串文本内使用双引号 (")，请使用反斜杠转义符 (\)。下面的两个字符串是等效的：

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

您可以根据字符串文本中存在的任何单引号或双引号来选择使用单引号或双引号，如下所示：

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

请务必记住 ActionScript 可区分单直引号 (') 和左右单引号 (' 或 ')。对于双引号也同样如此。请使用直引号来分割字符串文本。在将文本从其他来源粘贴到 ActionScript 中时，请确保使用正确的字符。

如下表所示，可以使用反斜杠转义符 (\) 在字符串文本中定义其他字符：

转义序列	字符
\b	Backspace
\f	换页符
\n	换行符
\r	回车符
\t	Tab
\u*nnnn*	Unicode 字符，字符代码由十六进制数字 nnnn 指定；例如， \u263a 为笑脸字符。
\\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
\\	单个反斜杠字符

length 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

每个字符串都有 length 属性，其值等于字符串中的字符数：

```
var str:String = "Adobe";
trace(str.length);            // output: 5
```

空字符串和 null 字符串的长度均为 0，如下例所示：

```
var str1:String = new String();
trace(str1.length);           // output: 0

str2:String = '';
trace(str2.length);           // output: 0
```

处理字符串中的字符

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

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，在下面的字符串中，字符 y 的位置为 0，而字符 w 的位置为 5：

```
"yellow"
```

您可以使用 charAt() 方法和 charCodeAt() 方法检查字符串各个位置上的字符，如此示例所示：

```
var str:String = "hello world!";
for (var i:int = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

在运行此代码时，会产生如下输出：

```
h - 104
e - 101
l - 108
l - 108
o - 111
- 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

您还可以通过字符代码，使用 fromCharCode() 方法定义字符串，如下例所示：

```
var myStr:String = String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
        // Sets myStr to "hello world!"
```

比较字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用以下运算符比较字符串：<、<=、!=、==、=> 和 >。可以将这些运算符用于条件语句（如 if 和 while），如下例所示：

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

在将这些运算符用于字符串时， ActionScript 会使用字符串中每个字符的字符代码值从左到右比较各个字符，如下所示：

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

使用 == 和 != 运算符可比较两个字符串，也可以将字符串与其他类型的对象进行比较，如下例所示：

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b); // true
trace(str1 == str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

获取其他对象的字符串表示形式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以获取任何类型对象的字符串表示形式。所有对象都提供了 toString() 方法来实现此目的：

```
var n:Number = 99.47;
var str:String = n.toString();
    // str == "99.47"
```

在使用 + 连接运算符连接 String 对象和不属于字符串的对象时，无需使用 toString() 方法。有关连接的详细信息，请参阅下一节。

对于给定对象， String() 全局函数返回的值与调用该对象的 toString() 方法返回的值相同。

连接字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

字符串连接的含义是：将两个字符串按顺序合并为一个字符串。例如，可以使用 + 运算符来连接两个字符串：

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2; // str3 == "greenish"
```

还可以使用 += 运算符来得到相同的结果，如下例所示：

```
var str:String = "green";
str += "ish"; // str == "greenish"
```

此外， String 类还包括 concat() 方法，可按如下方式对其进行使用：

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = str1.concat(" ", str2, " ", str3);
// str4 == "Bonjour from Paris"
```

如果使用 + 运算符 （或 += 运算符）对 String 对象和非 字符串的对象进行运算，则 ActionScript 会自动将非字符串对象转换为 String 对象以计算该表达式，如下例所示：

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area; // str == "Area = 28.274333882308138"
```

但是，可以使用括号进行分组，为 + 运算符提供运算的上下文，如下例所示：

```
trace("Total: $" + 4.55 + 1.45); // output: Total: $4.551.45
trace("Total: $" + (4.55 + 1.45)); // output: Total: $6
```

在字符串中查找子字符串和模式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

子字符串是字符串内的字符序列。例如，字符串 "abc" 具有如下子字符串：""、 "a"、 "ab"、 "abc"、 "b"、 "bc"、 "c"。可使用 ActionScript 的方法来查找字符串的子字符串。

模式是在 ActionScript 中通过字符串或正则表达式定义的。例如，下面的正则表达式定义了一个特定模式，即字母 A、B 和 C 的后面跟着一个数字字符 （正斜杠是正则表达式的分隔符）：

```
/ABC\d/
```

ActionScript 提供了在字符串中查找模式的方法，以及使用替换子字符串替换找到的匹配项的方法。随后的章节将介绍这些方法。

正则表达式可定义十分复杂的模式。有关详细信息，请参阅第 63 页的 “ 使用正则表达式 ”。

通过字符位置查找子字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

substr() 和 substring() 方法非常类似。两个方法都返回字符串的一个子字符串。并且两个方法都具有两个参数。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结尾 处字符的位置 （该字符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：

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // output: Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

slice() 方法与 substring() 方法的工作方式类似。当指定两个非负整数作为参数时，其运行方式将完全一样。但是， slice() 方法可以使用负整数作为参数，此时字符位置将从字符串末尾开始向前算起，如下例所示：

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // output: Pari
trace(str.slice(-3,-1)); // output: !!
trace(str.slice(-3,26)); // output: !!!
trace(str.slice(-3,str.length)); // output: !!!
trace(str.slice(-8,-3)); // output: Texas
```

可以结合使用非负整数和负整数作为 slice() 方法的参数。

查找匹配子字符串的字符位置

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

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：

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // output: 10
```

请注意， indexOf() 方法区分大小写。

可以指定第二个参数以指出在字符串中开始进行搜索的起始索引位置，如下所示：

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // output: 21
```

lastIndexOf() 方法在字符串中查找子字符串的最后一个匹配项：

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // output: 30
```

如果为 lastIndexOf() 方法提供了第二个参数，搜索将从字符串中的该索引位置反向 （从右到左）进行：

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // output: 21
```

创建由分隔符分隔的子字符串数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可使用 split() 方法创建子字符串数组，该数组根据分隔符进行划分。例如，可以将逗号分隔或制表符分隔的字符串分为多个字符串。

以下示例说明如何使用 “ 与 ” 字符 (&) 作为分隔符，将数组分割为多个子字符串：

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2); // params == ["first=joe","last=cheng"]
```

split() 方法的第二个参数是可选参数，该参数定义所返回数组的最大大小。

此外，还可以使用正则表达式作为分隔符：

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/); // a == ["Give","me","5."]
```

有关详细信息，请参阅第 63 页的 “ 使用正则表达式 ” 和用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

在字符串中查找模式并替换子字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

String 类提供了使用字符串中的模式的以下方法：

- 使用 match() 和 search() 方法可查找与模式相匹配的子字符串。
- 使用 replace() 方法可查找与模式相匹配的子字符串并使用指定子字符串替换它们。

随后的章节将介绍这些方法。

您可以使用字符串或正则表达式定义在这些方法中使用的模式。有关正则表达式的详细信息，请参阅第 63 页的 " 使用正则表达式 "。

查找匹配的子字符串

search() 方法返回与给定模式相匹配的第一个子字符串的索引位置，如下例所示：

```
var str:String = "The more the merrier.";
// (This search is case-sensitive.)
trace(str.search("the")); // output: 9
```

您还可以使用正则表达式定义要匹配的模式，如下例所示：

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

trace() 方法的输出为 0，因为字符串中第一个字符的索引位置为 0。在正则表达式中设置了 i 标志，因此搜索时不区分大小写。

search() 方法仅查找一个匹配项并返回其起始索引位置，即便在正则表达式中设置了 g （全局）标志。

下例展示一个更复杂的正则表达式，该表达式匹配被双引号引起来的字符串：

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern)); // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern)); // output: -1
// (Indicates no match, since there is no closing double quotation mark.)
```

match() 方法的工作方式与此类似。它搜索一个匹配的子字符串。但是，如果在正则表达式模式中使用了全局标志 （如下例所示）， match() 将返回一个包含匹配子字符串的数组：

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

对 results 数组进行如下设置：

```
["bob@example.com","omar@example.org"]
```

有关正则表达式的详细信息，请参阅第 63 页的 " 使用正则表达式 "。

替换匹配的子字符串

您可以使用 replace() 方法在字符串中搜索指定模式并使用指定的替换字符串替换匹配项，如下例所示：

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch")); //sche sells seaschells by the seaschore.
```

请注意，在本例中，因为在正则表达式中设置了 i (ignoreCase) 标志，所以匹配的字符串是不区分大小写的；而且因为设置了 g (global) 标志，所以会替换多个匹配项。有关详细信息，请参阅第 63 页的 " 使用正则表达式 "。

可以在替换字符串中包括以下 $ 替换代码。下表所示的替换文本将插入并替换 $ 替换代码：

$ 代码	替换文本
$$	$
$&	匹配的子字符串。
$`	字符串中位于匹配的子字符串前面的部分。此代码使用左单直引号字符 (`) 而不是单直引号 (') 或左单弯引号 (')。
$'	字符串中位于匹配的子字符串后的部分。此代码使用单直引号 (')。
$n	第 n 个捕获的括号组匹配项，其中 n 是 1-9 之间的数字，而且 $n 后面没有十进制数字。
$nn	第 nn 个捕获的括号组匹配项，其中 nn 是一个十进制的两位数 (01-99)。如果未定义第 nn 个捕获内容，则替换文本为空字符串。

例如，下面说明了如何使用 $2 和 $1 替换代码，它们分别表示匹配的第一个和第二个捕获组：

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

也可以使用函数作为 replace() 方法的第二个参数。匹配的文本将被函数的返回值替换。

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));

function usdToEuro(matchedSubstring:String,  capturedMatch1:String,  index:int,
str:String):String
{
    var usd:String = capturedMatch1;
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.853690;
    var euro:Number = parseFloat(usd) * exchangeRate;
    const euroSymbol:String = String.fromCharCode(8364);
    return euro.toFixed(2) + " " + euroSymbol;
}
```

在使用函数作为 replace() 方法的第二个形参时，将向该函数传递如下实参：

- 字符串的匹配部分。
- 任何捕获的括号组匹配项。按这种方式传递的参数数目因括号匹配项的数目而异。您可以通过检查函数代码中的 arguments.length - 3 来确定括号匹配项的数目。
- 字符串中匹配开始的索引位置。
- 完整的字符串。

转换字符串的大小写

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如下例所示， toLowerCase() 方法和 toUpperCase() 方法分别将字符串中的英文字母字符转换为小写和大写：

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase()); // dr. bob roberts, #9.
trace(str.toUpperCase()); // DR. BOB ROBERTS, #9.
```

执行完这些方法后，源字符串仍保持不变。要转换源字符串，请使用下列代码：

```
str = str.toUpperCase();
```

这些方法使用扩展字符，而并不仅限于 a–z 和 A–Z：

```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

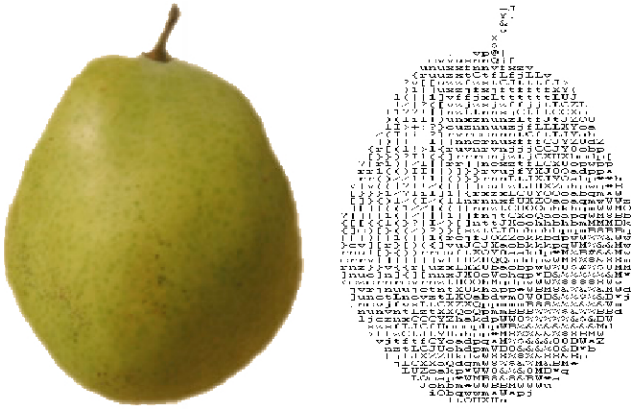
字符串示例：ASCII 图表

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此 ASCII 字符图示例说明了可以在 ActionScript 3.0 中使用 String 类实现的大量功能，其中包括：

- 使用 String 类的 split() 方法可从由某个字符分隔的字符串中提取值 （制表符分隔的文本文件中的图像信息）。
- 使用多种字符串操作技术 （包括 split()、连接，以及使用 substring() 和 substr() 提取字符串的一部分）可将图像标题中每个单词的第一个字母变为大写形式。
- 使用 getCharAt() 方法可从字符串中获取单个字符 （以确定对应于某个灰度位图值的 ASCII 字符）。
- 使用字符串连接可以按一次一个字符的方式建立图像的 ASCII 字符图表示形式。

ASCII 字符图 这一术语指的是图像的文本表示形式，即使用等宽字体字符 （如 Courier New 字符）的网格来绘制图像。下图便是该应用程序所生成 ASCII 字符图的一个例子：



图形的 ASCII 字符图版本显示在右侧。

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在文件夹 Samples/AsciiArt 下找到 ASCIIArt 应用程序文件。该应用程序包含以下文件：

文件	说明
AsciiArtApp.mxml 或 AsciiArtApp.fla	Flash (FLA) 或 Flex (MXML) 中的主应用程序文件
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	此类提供了应用程序主要功能，包括了从文本文件中提取图像元数据、加载图像和管理图像到文本的转换过程等功能。
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	此类提供了用于将图像数据转换为字符串版本的 parseBitmapData() 方法。
com/example/programmingas3/asciiArt/Image.as	此类表示所加载的位图图像。

文件	说明
com/example/programmingas3/asciiArt/ImageInfo.as	此类表示 ASCII 字符图像的元数据 （如标题、图像文件 URL 等）。
image/	此文件夹包含应用程序使用的图像。
txt/ImageData.txt	此制表符分隔的文本文件中包含与应用程序要加载的图像有关的信息。

提取由制表符分隔的值

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此示例使用了将应用程序数据与应用程序本身分开存储的通行做法；通过这种方式，在数据发生更改时 （例如，添加了另一幅图像或更改了图像标题），无需重新创建该 SWF 文件。在本例中，图像元数据 （包括图像标题、实际图像文件的 URL 以及用来操作图像的某些值）存储在一个文本文件中 （项目中的 txt/ImageData.txt 文件）。该文本文件的内容如下所示：

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

文件使用特定的制表符分隔格式。第一行为标题行。其余行包含要加载的每个位图的如下数据：

- 位图的文件名。
- 位图的显示名称。
- 位图的白色阀值和黑色阀值。这些值是十六进制值，高于这些值或低于这些值的像素将分别被认为是全白或全黑的。

应用程序启动后， AsciiArtBuilder 类将加载并分析文本文件的内容，以便创建它要显示的图像的 “ 堆栈 ”，执行这些操作时使用 AsciiArtBuilder 类的 parseImageInfo() 方法中的如下代码：

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // Create a new image info record and add it to the array of image info.
        var imageInfo:ImageInfo = new ImageInfo();

        // Split the current line into values (separated by tab (\t)
        // characters) and extract the individual properties:
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

文本文件的完整内容包含在单个 String 实例中，即 _imageInfoLoader.data 属性。使用 split() 方法并以换行符 (\n) 为参数，将该 String 实例分割到一个 Array (lines) 中，数组元素为文本文件的各个行。然后，代码使用循环来使用各行 （第一行除外，因为它只包含标题而不包含实际内容）。在循环内部，再次使用 split() 方法将每行的内容分为一组值（名为 imageProperties 的 Array 对象）。在本例中，用于 split() 方法的参数为制表符 (\t)，因为每行中的值均由制表符进行分隔。

使用 String 的方法标准化图像标题

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此应用程序的设计目标之一便是使用一种标准格式显示所有图像标题，即标题中每一个单词的第一个字母均为大写形式 （在英文标题中通常不大写的少数单词除外）。应用程序并不假定文本文件已经包含格式正确的标题，它在从文本文件中提取标题时对标题格式进行设置。

在前面的代码清单中，使用了如下代码行来提取每个图像元数据值：

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

在该代码中，来自文本文件的图像标题在存储到 ImageInfo 对象之前先传递给 normalizeTitle() 方法进行处理：

```
private function normalizeTitle(title:String):String
{
    var words:Array = title.split(" ");
    var len:uint = words.length;
    for (var i:uint; i < len; i++)
    {
        words[i] = capitalizeFirstLetter(words[i]);
    }

    return words.join(" ");
}
```

此方法使用 split() 方法将标题分割为各个单独的单词 （由空格字符加以分隔），然后将每个单词传递给 capitalizeFirstLetter() 方法进行处理，接着使用 Array 类的 join() 方法将单词重新合并为一个字符串。

如同其名称所表达的含义， capitalizeFirstLetter() 方法实际执行将每个单词的第一个字母变为大写形式的工作：

```
/**
 * Capitalizes the first letter of a single word, unless it's one of
 * a set of words that are normally not capitalized in English.
 */
private function capitalizeFirstLetter(word:String):String
{
    switch (word)
    {
        case "and":
        case "the":
        case "in":
        case "an":
        case "or":
        case "at":
        case "of":
        case "a":
            // Don't do anything to these words.
            break;
        default:
            // For any other word, capitalize the first character.
            var firstLetter:String = word.substr(0, 1);
            firstLetter = firstLetter.toUpperCase();
            var otherLetters:String = word.substring(1);
            word = firstLetter + otherLetters;
    }
    return word;
}
```

在英文中，如果标题中某个单词为以下单词之一，则不会 将其首字符变为大写形式：“and”、“the”、“in”、“an”、“or”、“at”、“of” 或 “a”。（这是相关规则的简化版本。）为了实现此逻辑，代码首先使用 switch 语句来检查单词是否为不应将其首字符大写的单词之一。如果是，代码直接跳出 switch 语句。另一方面，如果单词应大写，则在几个步骤中完成此操作，如下所示：

1 使用 substr(0, 1) 提取出单词的第一个字母，该命令从位于索引位置 0 的字符 （即字符串的第一个字母，由第一个参数 0 指定）开始提取子字符串。该子字符串的长度为一个字符 （由第二个参数 1 指定）。
2 使用 toUpperCase() 方法将该字符变为大写形式。
3 使用 substring(1) 提取原始单词的其余字符，该命令提取从索引位置 1 （第二个字母）开始直至字符串结尾 （通过将 substring() 方法的第二个参数保留为空进行指定）的子字符串。
4 使用字符串连接 firstLetter + otherLetters 将刚才变为大写形式的第一个字母与其余字母合并在一起，创建出最终的单词。

生成 ASCII 字符图文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

BitmapToAsciiConverter 类提供了将位图图像转换为其 ASCII 文本表示形式的功能。此过程由 parseBitmapData() 方法执行，下面展示了该方法的部分工作过程：

```
var result:String = "";

// Loop through the rows of pixels top to bottom:
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // Within each row, loop through pixels left to right:
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // Convert the gray value in the 0-255 range to a value
        // in the 0-64 range (since that's the number of "shades of
        // gray" in the set of available characters):
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

此代码首先定义一个名为 result 的 String 实例，用于构建位图图像的 ASCII 字符图版本。然后，它遍历源位图图像的每个像素。通过使用若干颜色处理技术 （为了简便起见，此处省略了对这些技术的介绍），它将每个像素的红色、绿色和蓝色值转换为单个灰度值 （一个介于 0-255 之间的数字）。接着，代码将该值除以 4 （如下所示）以将其转换为介于 0-63 之间的一个值，此值存储在变量 index 中。（之所以使用 0-63 的范围，是因为此应用程序使用的可用 ASCII 字符的 “ 调色板 ” 包含 64 个值。）该字符调色板在 BitmapToAsciiConverter 类中定义为一个 String 实例：

```
// The characters are in order from darkest to lightest, so that their
// position (index) in the string corresponds to a relative color value
// (0 = black).
private static const palette:String = "@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

因为变量 index 定义调色板中的哪个 ASCII 字符对应于位图图像中的当前像素，所以可使用 charAt() 方法从 palette 字符串中检索该字符。然后，使用连接赋值运算符 (+=) 将其追加到 result 字符串实例。此外，在每行像素的末尾，会将一个换行符连接到 result 字符串的末尾，强制该行换行以创建新的一行字符 “ 像素 ”。

第 3 章：使用数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用数组可以在单数据结构中存储多个值。可以使用简单的索引数组 （使用固定有序整数索引存储值），也可以使用复杂的关联数组 （使用任意键存储值）。数组也可以是多维的，即包含本身是数组的元素。最后，您可以对其元素均是同一数据类型的实例的数组使用 Vector。

更多帮助主题

Array

Vector

数组基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

编程时经常需要使用一组项目而不是单个对象。例如在一个音乐播放器应用程序中，您可能需要创建一个待播放歌曲列表。您不希望必须为该列表中的每首歌曲都单独创建一个变量。更好的做法是将所有 Song 对象放在一个包中，从而能够将其作为一个组进行使用。

数组是一种编程元素，它用作一组项目的容器，如一组歌曲。通常，数组中的所有项目都是相同类的实例，但这在 ActionScript 中并不是必需的。数组中的各个项目称为数组的 " 元素 "。可以将数组视为变量的 " 文件柜 "。变量可以作为元素添加到数组中，就像将文件夹放到文件柜中一样。您可以将数组作为单个变量使用 （就像将整个文件柜搬到另一个地方一样）。您可以将变量作为一个组使用 （就像逐一浏览文件夹以搜索一条信息一样）。也可以分别访问这些变量 （就像打开文件柜并选择一个文件夹一样）。

例如，假设您要创建一个音乐播放器应用程序，用户可以在其中选择多首歌曲，并将这些歌曲添加到播放列表中。在您的 ActionScript 代码中有一个名为 addSongsToPlaylist() 的方法，该方法接受单个数组作为参数。无论要将多少首歌曲 （几首、很多首甚至只有一首）添加到列表中，您都只调用一次 addSongsToPlaylist() 方法，并向其传递包含 Song 对象的数组。在 addSongsToPlaylist() 方法中，可以使用循环来逐个访问数组元素 （歌曲），并将歌曲实际添加到播放列表中。

最常见的 ActionScript 数组类型为 " 索引数组 "。在索引数组中，每个项目都存储在编号位置 （称为 " 索引 "）。可以使用该编号来访问项目，如同地址一样。索引数组可以很好地满足大多数编程需要。 Array 类是用于表示索引数组的常见类。

索引数组常常用于存储具有相同类型的多个项目 （作为同一类的实例的对象）。 Array 类没有任何办法限制它所包含的项目的类型。Vector 类是一种索引数组类型，其中单个数组中的所有项目都具有同一类型。使用 Vector 实例而不是 Array 实例还可以提供性能改进和其他优势。从 Flash Player 10 和 Adobe AIR 1.5 开始提供 Vector 类。

索引数组的一个特殊用法是 " 多维数组 "。多维数组是一种索引数组，其中的元素也是索引数组 （这些数组又包含其他元素）。

另一种数组类型是 " 关联数组 "，该数组使用字符串 " 键 " （而不是数字索引）来标识各个元素。最后， ActionScript 3.0 还包括表示 " 字典 " 的 Dictionary 类。字典是允许您将任何类型的对象用作键来区分元素的数组。

重要概念和术语

以下参考列表包含在对处理例程的数组和矢量进行编程时将遇到的重要术语：

Array 用作容器以将多个对象组合在一起的对象。

数组访问 ([]) 运算符 一对中括号，其中含有唯一标识数组元素的索引或键。此语法用在数组变量名称之后，以指定数组的单个元素而不是整个数组。

关联数组 使用字符串键来标识各个元素的数组。

基本类型 允许 Vector 实例存储的对象的数据类型。

字典 其项目由一对对象 （称为键和值）组成的数组。它使用键来标识单个元素，而不是使用数字索引。

元素 数组中的单个项目。

索引 用于标识索引数组中的单个元素的数字 “ 地址 ”。

索引数组 标准类型的数组，将每个元素存储在编号位置中，并使用数字 （索引）来标识各个元素。

键 用于标识关联数组或字典中的单个元素的字符串或对象。

多维数组 包含的项目是数组 （而不是单个值）的数组。

T 本文档中使用的标准约定，用于表示 Vector 实例的基本类型（与具体的基本类型无关）。T 约定用于表示类名称，如 Type 参数说明中所示。（“T” 表示 “ 类型 ”，如同在 “ 数据类型 ” 中一样。）。

类型参数 与 Vector 类名称一起使用以指定 Vector 的基本类型 （它存储的对象的数据类型）的语法。该语法包括一个点 (.)，然后是由尖括号 (<>) 括起来的数据类型名称。放在一起后类似于：Vector.<T>。在本文档中，在类型参数中指定的类通常表示为 T。

Vector 一种数组类型，其所有元素都是同一数据类型的实例。

索引数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

索引数组存储一系列经过组织的单个或多个值，其中的每个值都可以通过使用一个无符号整数值进行访问。第一个索引始终是数字 0，且添加到数组中的每个后续元素的索引以 1 为增量递增。在 ActionScript 3.0 中，有两个类可用作索引数组：Array 类和 Vector 类。

索引数组使用无符号 32 位整数作为索引号。索引数组的最大大小为 2^{32} - 1，即 4,294,967,295。如果尝试创建超过该最大大小的数组，则会导致运行时错误。

若要访问索引数组中的单个元素，请使用数组访问 ([]) 运算符指定要访问的元素的索引位置。例如，下面的代码表示名为 songTitles 的索引数组中的第一个元素 （位于索引 0 处的元素）：

```
songTitles[0]
```

数组变量名称后跟索引 （在中括号中）的组合用作一个标识符。（换句话说，可以按照变量名称的任何使用方式来使用这个组合）。可以在赋值语句左侧使用名称和索引来向索引数组元素赋值。

```
songTitles[1] = "Symphony No. 5 in D minor";
```

同样，可以在赋值语句右侧使用名称和索引来检索索引数组元素的值。

```
var nextSong:String = songTitles[2];
```

您还可以在中括号中使用变量而不是提供显式值。（变量必须包含一个非负整数值，如 uint、正 int 或正整数 Number 实例）。此方法通常用于 “ 循环访问 ” 索引数组中的元素，并对某些元素或所有元素执行操作。下面的代码清单演示了这种方法。这段代码使用循环来访问名为 oddNumbers 的 Array 对象中的每个值。它使用 trace() 语句以 “oddNumber[index] = value” 的形式输出每个值：

```
var oddNumbers:Array = [1, 3, 5, 7, 9, 11];
var len:uint = oddNumbers.length;
for (var i:uint = 0; i < len; i++)
{
    trace("oddNumbers[" + i.toString() + "] = " + oddNumbers[i].toString());
}
```

Array 类

索引数组的第一个类型为 Array 类。Array 实例的值可以为任意数据类型。同一 Array 对象中包含的对象可以具有不同的数据类型。例如，一个 Array 实例可以在索引 0 处有 String 值，在索引 1 处有 Number 实例，而在索引 2 处有 XML 对象。

Vector 类

ActionScript 3.0 中可用的另一种索引数组类型为 Vector 类。 Vector 实例是 " 指定类型的数组 "，这表示 Vector 实例中的所有元素始终具有同一数据类型。

注：从 Flash Player 10 和 Adobe AIR 1.5 开始提供 Vector 类。

在声明 Vector 变量或实例化 Vector 对象时，要明确指定 Vector 可以包含的对象的数据类型。指定的数据类型称为 Vector 的 " 基本类型 "。在运行时和编译时 （在严格模式下），会检查任何设置 Vector 元素的值或从 Vector 检索值的代码。如果要添加或检索的对象的数据类型与 Vector 的基本类型不匹配，则会发生错误。

除数据类型限制之外， Vector 类还具有一些其他限制，从而有别于 Array 类：

- Vector 是一种密集数组。即使某个 Array 对象在位置 1 到 6 没有值，该对象的索引 0 和 7 处也可以有值。但是， Vector 的每个索引位置都必须有值 （或为 null）。
- Vector 还可以是固定长度。这表示 Vector 包含的元素数不能更改。
- 对 Vector 的元素的访问需要接受范围检查。绝对不能从大于最后一个元素索引 (length - 1) 的索引中读取值。绝对不能对超过当前最后一个索引一个以上位置的索引设置值 （也就是说，只能在现有索引或索引 [length] 处设置值）。

由于 Vector 具有这些限制，因此 Vector 相对于所有元素均为单个类的实例的 Array 实例有三个主要优点：

- 性能：使用 Vector 实例时的数组元素访问和迭代的速度比使用 Array 实例时的速度要快很多。
- 类型安全性：在严格模式下，编译器可以识别数据类型错误。这类错误的例子包括将数据类型错误的值分配给 Vector 或从 Vector 中读取值时使用错误的数据类型。在运行时，当向 Vector 对象添加数据或从 Vector 对象读取数据时也会检查数据类型。但请注意，当使用 push() 方法或 unshift() 方法向 Vector 添加值时，在编译时不会检查参数的数据类型。不过在使用这些方法时，仍会在运行时检查值。
- 可靠性：与 Array 相比，运行时范围检查 （或固定长度检查）大大提高了可靠性。

除了有一些限制和优点以外， Vector 类与 Array 类非常相似。 Vector 对象的属性和方法与 Array 的属性和方法类似 （大多数情况下完全相同）。对于大多数需要使用所有元素都具有相同数据类型的 Array 的情况， Vector 实例更为可取。

创建数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用多种方法来创建 Array 实例或 Vector 实例。但是，创建每种数组类型的方法多少有些不同。

创建 Array 实例

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以通过调用 Array() 构造函数或使用 Array 文本语法来创建 Array 对象。

Array() 构造函数有三种使用方式。第一种，如果调用不带参数的构造函数，会得到空数组。可以使用 Array 类的 length 属性来验证数组是否不包含元素。例如，下面的代码调用不带参数的 Array() 构造函数：

```
var names:Array = new Array();
trace(names.length); // output: 0
```

第二种，如果将一个数字用作 Array() 构造函数的唯一参数，则会创建长度等于此数值的数组，并且每个元素的值都设置为 undefined。参数必须为介于值 0 和 4,294,967,295 之间的无符号整数。例如，下面的代码调用带有一个数字参数的 Array() 构造函数：

```
var names:Array = new Array(3);
trace(names.length); // output: 3
trace(names[0]); // output: undefined
trace(names[1]); // output: undefined
trace(names[2]); // output: undefined
```

第三种，如果调用构造函数并传递一个元素列表作为参数，将创建具有与每个参数对应的元素的数组。下面的代码将三个参数传递给 Array() 构造函数：

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // output: 3
trace(names[0]); // output: John
trace(names[1]); // output: Jane
trace(names[2]); // output: David
```

也可以使用 Array 文本创建数组。可以将 Array 文本直接分配给数组变量，如下面的示例所示：

```
var names:Array = ["John", "Jane", "David"];
```

创建 Vector 实例

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

可以通过调用 Vector.<T>() 构造函数创建 Vector 实例。还可以通过调用 Vector.<T>() 全局函数创建 Vector。该函数将指定对象转换为 Vector 实例。在 Flash Professional CS5 和更高版本、Flash Builder 4 和更高版本以及 Flex 4 和更高版本中，还可以使用 Vector 文本语法创建 Vector 实例。

只要声明 Vector 变量 （Vector 方法参数或方法返回类型也一样），就需要指定 Vector 变量的基本类型。在通过调用 Vector.<T>() 构造函数创建 Vector 实例。换句话说，只要在 ActionScript 中使用术语 Vector，就需要指定基本类型。

可以使用 type 参数语法指定 Vector 的基本类型。在代码中，类型参数紧跟单词 Vector。它包括一个点 (.)，然后是由尖括号 (<>) 括起来的基类名称，如此示例中所示：

```
var v:Vector.<String>;
v = new Vector.<String>();
```

在此示例的第一行内容中，变量 v 声明为 Vector.<String> 实例。换句话说，它表示只能包含 String 实例的索引数组。第二行调用 Vector() 构造函数创建一个具有相同 Vector 类型的实例（也就是其中所有元素都是 String 对象的 Vector）。该示例将该对象分配给 v。

使用 **Vector.<T>()** 构造函数

如果使用不带任何参数的 Vector.<T>() 构造函数，则该函数会创建一个空的 Vector 实例。可以通过检查 Vector 的 length 属性来测试它是否为空。例如，下面的代码调用不带任何参数的 Vector.<T>() 构造函数：

```
var names:Vector.<String> = new Vector.<String>();
trace(names.length); // output: 0
```

如果您预先知道 Vector 最初需要多少元素，则可以预定义 Vector 中的元素数。若要使用特定数量的元素创建 Vector，请将元素数作为第一个参数 （length 参数）进行传递。因为 Vector 元素不能为空，所以会使用具有基本类型的实例填充这些元素。如果基本类型是允许使用 null 值的引用类型，则所有元素都包含 null。否则，所有元素都包含该类的默认值。例如， uint 变量不能为 null。因此，在下面的代码清单中，使用七个元素创建名为 ages 的 Vector，其中每个元素都包含值 0：

```
var ages:Vector.<uint> = new Vector.<uint>(7);
trace(ages); // output: 0,0,0,0,0,0,0
```

最后，还可以使用 Vector.<T>() 构造函数创建固定长度 Vector，方法是将 true 作为第二个参数 （fixed 参数）进行传递。在这种情况下，将使用指定的元素数创建 Vector，且元素数不可更改。但是请注意，仍然可以更改固定长度 Vector 的元素值。

使用 **Vector** 文本语法构造函数

在 Flash Professional CS5 和更高版本、Flash Builder 4 和更高版本以及 Flex 4 和更高版本中，可以向 Vector.<T>() 构造函数传递值列表来指定 Vector 的初始值：

```
// var v:Vector.<T> = new <T>[E0, ..., En-1 ,];
// For example:
var v:Vector.<int> = new <int>[0,1,2,];
```

下列信息适用于此语法：

- 尾部的逗号是可选的。
- 数组中不支持空项目；类似 var v:Vector.<int> = new <int>[0,,2,] 的语句会引发编译器错误。
- 无法为 Vector 实例指定默认长度。该长度与初始化列表中的元素数相同。
- 无法指定 Vector 实例是否具有固定长度。应使用 fixed 属性。
- 如果作为值传递的项目与指定类型相匹配，则可能会发生数据丢失或出现错误。例如：

  ```
  var v:Vector.<int> = new <int>[4.2]; // compiler error when running in strict mode
  trace(v[0]); //returns 4 when not running in strict mode
  ```

使用 **Vector.<T>()** 全局函数

除了 Vector.<T>() 和 Vector 文本语法构造函数之外，还可以使用 Vector.<T>() 全局函数创建 Vector 对象。Vector.<T>() 全局函数是一个转换函数。当调用 Vector.<T>() 全局函数时，需指定该方法返回的 Vector 的基本类型。可将单个索引数组（Array 或 Vector 实例）作为参数进行传递。该方法随后返回具有指定基本类型的 Vector，其中包含源数组参数中的值。下面的代码清单演示了用于调用 Vector.<T>() 全局函数的语法：

```
var friends:Vector.<String> = Vector.<String>(["Bob", "Larry", "Sarah"]);
```

Vector.<T>() 全局函数在两个级别上执行数据类型转换。首先，当 Array 实例传递给该函数时，会返回 Vector 实例。其次，无论源数组是 Array 还是 Vector 实例，该函数都会试图将源数组的元素转换为基本类型的值。该转换使用标准 ActionScript 数据类型转换规则。例如，下面的代码清单将源 Array 中的 String 值转换为结果 Vector 中的整数。第一个值 （“1.5”）的小数部分被截断，非数字的第三个值 （“Waffles”）在结果中转换为 0：

```
var numbers:Vector.<int> = Vector.<int>(["1.5", "17", "Waffles"]);
trace(numbers); // output: 1,17,0
```

如果无法转换任何源元素，则会发生错误。

当代码调用 Vector.<T>() 全局函数时，如果源数组中的某个元素是指定基本类型的子类的实例，则该元素会添加到结果 Vector 中 （不发生错误）。只有使用 Vector.<T>() 全局函数才能将基本类型为 T 的 Vector 转换为基本类型为 T 的超类的 Vector。

插入数组元素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用于将元素添加到索引数组的最基本方法是使用数组访问 ([]) 运算符。若要设置索引数组元素的值，请在赋值语句的左侧使用 Array 或 Vector 对象名称和索引编号：

```
songTitles[5] = "Happy Birthday";
```

如果 Array 或 Vector 在该索引处还没有元素，则会创建该索引并将值存储在那里。如果该索引处存在值，则新值会替换现有值。

Array 对象允许您在任何索引位置创建元素。但是对于 Vector 对象，您只能向现有索引或下一个可用索引赋值。下一个可用索引对应于 Vector 对象的 length 属性。向 Vector 对象添加新元素的最安全方式是使用类似于下面的清单的代码：

```
myVector[myVector.length] = valueToAdd;
```

可以使用 Array 和 Vector 类的三种方法（push()、unshift() 和 splice()）将元素插入索引数组。push() 方法用于在数组末尾添加一个或多个元素。换言之，使用 push() 方法在数组中插入的最后一个元素将具有最大索引号。 unshift() 方法用于在数组开头插入一个或多个元素，并且始终在索引号 0 处插入。 splice() 方法用于在数组中的指定索引处插入任意数目的项目。

下面的示例对所有三种方法进行了说明。它创建一个名为 planets 的数组，以便按照行星距离太阳由近到远的顺序存储各个行星的名称。首先，调用 push() 方法以添加初始项目 Mars。接着，调用 unshift() 方法在数组开头插入项 Mercury。最后，调用 splice() 方法在 Mercury 之后和 Mars 之前插入项 Venus 和 Earth。传递给 splice() 的第一个参数是整数 1，它用于指示从索引 1 处开始插入。传递给 splice() 的第二个参数是整数 0，它指示不应删除任何项。传递给 splice() 的第三和第四个参数 Venus 和 Earth 为要插入的项。

```
var planets:Array = new Array();
planets.push("Mars"); // array contents: Mars
planets.unshift("Mercury"); // array contents: Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets); // array contents: Mercury,Venus,Earth,Mars
```

push() 和 unshift() 方法均返回一个无符号整数，它们表示修改后的数组长度。在用于插入元素时，splice() 方法返回空数组，这看上去也许有点奇怪，但考虑到 splice() 方法的多用途性，您便会觉得这样更有意义。通过使用 splice() 方法，不仅可以将元素插入到数组中，而且还可以从数组中删除元素。用于删除元素时， splice() 方法将返回包含被删除元素的数组。

注：如果某个 Vector 对象的 fixed 属性为 true，则不能更改该 Vector 中的元素数。如果尝试使用此处介绍的方法向固定长度 Vector 添加新元素，则会发生错误。

检索值和删除数组元素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用于检索索引数组中的元素值的最简单方法是使用数组访问 ([]) 运算符。若要检索索引数组元素的值，请在赋值语句的右侧使用 Array 或 Vector 对象名称和索引编号：

```
var myFavoriteSong:String = songTitles[3];
```

可以尝试使用不存在元素的位置的索引来检索 Array 或 Vector 中的值。在这种情况下， Array 对象返回未定义的值，而 Vector 会引发 RangeError 异常。

可以使用 Array 和 Vector 类的三种方法（pop()、shift() 和 splice()）删除元素。pop() 方法用于从数组末尾删除一个元素。换言之，它将删除位于最大索引号处的元素。 shift() 方法用于从数组开头删除一个元素，也就是说，它始终删除索引号 0 处的元素。 splice() 方法既可用来插入元素，也可以删除任意数目的元素，其操作的起始位置位于由发送到此方法的第一个参数指定的索引号处。

下面的示例使用所有三种方法从 Array 实例中删除元素。该示例创建一个名为 oceans 的 Array，用于存储较大水域的名称。Array 中的某些名称为湖泊的名称而非海洋的名称，因此需要将其删除。

首先，使用 splice() 方法删除项 Aral 和 Superior，并插入项 Atlantic 和 Indian。传递给 splice() 的第一个参数是整数 2，它指示应从列表中的第三个项（即索引 2 处）开始执行操作。第二个参数 2 指示应删除两个项。其余两个参数 Atlantic 和 Indian 是要在索引 2 处插入的值。

然后，使用 pop() 方法删除数组中的最后一个元素 Huron。最后，使用 shift() 方法删除数组中的第一个项 Victoria。

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian", "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // replaces Aral and Superior
oceans.pop(); // removes Huron
oceans.shift(); // removes Victoria
trace(oceans);// output: Pacific,Arctic,Atlantic,Indian
```

pop() 和 shift() 方法均返回已删除的项。对于 Array 实例，由于数组可以包含任意数据类型的值，因而返回值的数据类型为 Object。对于 Vector 实例，返回值的数据类型是 Vector 的基本类型。splice() 方法返回包含所删除的值的 Array 或 Vector。可以更改 oceans Array 示例，以使 splice() 调用将返回的 Array 分配给新的 Array 变量，如下面的示例所示：

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // output: Aral,Superior
```

您可能会遇到对 Array 对象元素使用 delete 运算符的代码。 delete 运算符会将 Array 元素的值设置为 undefined，但它不会从 Array 中删除元素。例如，下面的代码对 oceans Array 中的第三个元素使用 delete 运算符，但此 Array 的长度仍然为 5：

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);// output: Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]); // output: undefined
trace(oceans.length); // output: 5
```

可以使用数组的 length 属性截断 Array 或 Vector。如果您将某个索引数组的 length 属性设置为小于该数组的当前长度的长度，则该数组会被截断，从而删除存储在比 length 的新值减 1 大的索引编号处的所有元素。例如，如果 oceans 数组进行了排序，以使所有有效项都位于数组开头，则可以使用 length 属性删除位于数组后部的项，如下面的代码所示：

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // output: Arctic,Pacific
```

注：如果某个 Vector 对象的 fixed 属性为 true，则不能更改该 Vector 中的元素数。如果尝试使用此处介绍的方法删除固定长度 Vector 中的元素或截断固定长度 Vector，则会发生错误。

对数组排序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用三种方法 （reverse()、 sort() 和 sortOn()）通过排序或反向排序来更改索引数组的顺序。所有这些方法都用来修改现有数组。下表概述了这些方法及其针对 Array 和 Vector 对象的行为：

方法	Array 行为	Vector 行为
reverse()	更改元素的顺序，使最后一个元素变为第一个元素，倒数第二个元素变为第二个元素，依此类推。	与 Array 行为相同
sort()	用于按照各种预定义的方式对 Array 的元素进行排序 （如字母顺序或数字顺序）。还可以指定自定义排序算法。	根据您指定的自定义排序算法对元素进行排序
sortOn()	用于对具有一个或多个公共属性的对象进行排序，排序时指定这样的属性作为排序键	在 Vector 类中不可用

reverse() 方法

reverse() 方法不带参数，也不返回值，但可以将数组从当前顺序切换为相反顺序。以下示例颠倒了 oceans 数组中列出的海洋顺序：

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // output: Pacific,Indian,Atlantic,Arctic
```

使用 **sort()** 方法的基本排序 （仅适用于 **Array** 类）

对于 Array 实例， sort() 方法按照 “ 默认排序顺序 ” 重新排列数组中的元素。默认排序顺序具有以下特征：

- 排序区分大小写，也就是说大写字符优先于小写字符。例如，字母 D 优先于字母 b。
- 排序按照升序进行，也就是说低位字符代码 （例如 A）优先于高位字符代码 （例如 B）。
- 排序将相同的值互邻放置，并且不区分顺序。
- 排序基于字符串，也就是说，在比较元素之前，先将其转换为字符串 （例如， 10 优先于 3，因为相对于字符串 "3" 而言，字符串 "1" 具有低位字符代码）。

您也许需要不区分大小写或者按照降序对 Array 进行排序，或者您的数组中包含数字，从而需要按照数字顺序而非字母顺序进行排序。Array 类的 sort() 方法具有 options 参数，可通过该参数改变默认排序顺序的各个特征。options 是由 Array 类中的一组静态常量定义的，如以下列表所示：

- Array.CASEINSENSITIVE：此选项可使排序不区分大小写。例如，小写字母 b 优先于大写字母 D。
- Array.DESCENDING：用于颠倒默认的升序排序。例如，字母 B 优先于字母 A。
- Array.UNIQUESORT：如果发现两个相同的值，此选项将导致排序中止。
- Array.NUMERIC：这会导致排序按照数字顺序进行，比方说 3 优先于 10。

以下示例重点说明了这些选项中的某些选项。它创建一个名为 poets 的 Array，并使用几种不同的选项对其进行排序。

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // default sort
trace(poets); // output: Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // output: Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // output: cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // use two options
trace(poets); // output: Dante,cummings,Blake,Angelou
```

使用 **sort()** 方法的自定义排序 （适用于 **Array** 和 **Vector** 类）

除了可用于 Array 对象的基本排序之外，您还可以定义自定义排序规则。此方法是可用于 Vector 类的唯一一种形式的 sort() 方法。若要定义自定义排序，请编写自定义排序函数，并将该函数作为参数传递给 sort() 方法。

例如，如果有一个名称列表，其中每个列表元素都包含一个人的全名，但现在要按照姓氏对列表排序，则必须使用自定义排序函数分析每个元素，并在排序函数中使用姓氏。下面的代码说明如何使用作为参数传递给 Array.sort() 方法的自定义函数来完成上述工作：

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
trace(names); // output: John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // output: Jane Doe,Mike Jones,John Q. Smith
```

自定义排序函数 orderLastName() 使用正则表达式从每个元素中提取姓，以用于比较操作。针对 names 数组调用 sort() 方法时，函数标识符 orderLastName 用作唯一的参数。排序函数接受两个参数 a 和 b，因为它每次对两个数组元素进行操作。排序函数的返回值指示应如何对元素排序：

- 返回值 -1 指示第一个参数 a 优先于第二个参数 b。

- 返回值 1 指示第二个参数 b 优先于第一个参数 a。
- 返回值为 0 指示元素具有相同的排序优先级。

sortOn() 方法 （仅适用于 **Array** 类）

sortOn() 方法是为具有包含对象的元素的 Array 对象设计的。这些对象应至少具有一个可用作排序键的公共属性。如果将 sortOn() 方法用于任何其他类型的数组，则会产生意外结果。

注：Vector 类不包含 sortOn() 方法。此方法仅用于 Array 对象。

下面的示例修改 poets Array，以使每个元素均为对象而非字符串。每个对象既包含诗人的姓氏又包含诗人的出生年份。

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

可以使用 sortOn() 方法，按照 born 属性对 Array 进行排序。 sortOn() 方法定义两个参数 fieldName 和 options。必须将 fieldName 参数指定为字符串。在以下示例中，使用两个参数 "born" 和 Array.NUMERIC 来调用 sortOn()。Array.NUMERIC 参数用于确保按照数字顺序进行排序，而不是按照字母顺序。即使所有数字具有相同的数位，这也是一种很好的做法，因为当后来在数组中添加较少数位或较多数位的数字时，它会确保排序如期继续进行。

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

在不修改原始数组的情况下进行排序 （仅适用于 **Array** 类）

通常， sort() 和 sortOn() 方法用于修改 Array。如果要对 Array 排序而又不修改现有数组，请将 Array.RETURNINDEXEDARRAY 常量作为 options 参数的一部分进行传递。此选项将指示方法返回反映排序的新 Array，同时保留原始 Array 原封不动。方法返回的 Array 为由反映新排序顺序的索引号组成的简单 Array，不包含原始 Array 的任何元素。例如，若要根据出生年份对 poets Array 排序而不修改该 Array，请在为 options 形参传递的实参中包括 Array.RETURNINDEXEDARRAY 常量。

下面的示例将返回的索引信息存储在名为 indices 的 Array 中，然后使用 indices 数组和未修改的 poets 数组按出生年份的顺序输出诗人：

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
/* output:
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

查询数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Array 和 Vector 类的四种方法 concat()、 join()、 slice() 和 toString() 均可用于查询数组的信息，而不修改数组。 concat() 和 slice() 方法返回新数组；而 join() 和 toString() 方法返回字符串。 concat() 方法将新数组和元素列表作为参数，并将其与现有数组结合起来创建新数组。 slice() 方法有两个名为 startIndex 和 endIndex 的参数，并返回一个新数组，新数组中包含从现有数组 " 分离 " 的元素副本。分离从 startIndex 处的元素开始，到 endIndex 处的前一个元素结束。值得强调的是， endIndex 处的元素不包括在返回值中。

以下示例通过 concat() 和 slice() 方法，使用其他数组的元素创建新数组：

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // output: alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // output: alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // output: alpha,beta,gamma
```

可以使用 join() 和 toString() 方法查询数组，并将其内容作为字符串返回。如果 join() 方法没有使用参数，则这两个方法的行为相同，它们都返回一个字符串，其中包含数组中所有元素的逗号分隔列表。与 toString() 方法不同， join() 方法接受名为 delimiter 的参数；可以使用此参数，选择要用作返回字符串中各个元素之间分隔符的符号。

下面的示例创建名为 rivers 的 Array，并调用 join() 和 toString() 以便采用字符串形式返回该 Array 中的值。 toString() 方法用于返回以逗号分隔的值 (riverCSV) ；而 join() 方法用于返回以 + 字符分隔的值。

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // output: Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // output: Nile+Amazon+Yangtze+Mississippi
```

对于 join() 方法，应注意的一个问题是，无论为主数组元素指定的分隔符是什么，为嵌套 Array 或 Vector 实例返回的值始终以逗号作为分隔符，如下面的示例所示：

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // output: a+b,c,d+e
```

关联数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

关联数组有时也称为 " 哈希 " 或 " 映射 "，它使用 " 键 " 而非数字索引来组织存储的值。关联数组中的每个键都是用于访问一个存储值的唯一字符串。关联数组为 Object 类的实例，也就是说每个键都与一个属性名称对应。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。

ActionScript 3.0 中还引入了名为 " 字典 " 的高级关联数组类型。字典是 flash.utils 包中的 Dictionary 类的实例，使用的键可以为任意数据类型。换言之，字典的键不局限于 String 类型的值。

具有字符串键的关联数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中有两种创建关联数组的方式。第一种方式是使用 Object 实例。使用 Object 实例，您可以通过对象文本来初始化数组。 Object 类的实例 （也称为 " 通用对象 "）在功能上等同于关联数组。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。

下面的示例创建一个名为 monitorInfo 的关联数组，并使用对象文本初始化具有两个键和值对的数组：

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

如果在声明数组时不需要初始化，可以使用 Object 构造函数创建数组，如下所示：

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

使用对象文本或 Object 类构造函数创建数组后，可以使用数组访问 ([]) 运算符或点运算符 (.)。以下示例将两个新值添加到 monitorArray 中：

```
monitorInfo["aspect ratio"] = "16:10"; // bad form, do not use spaces
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
// output: 16:10 16.7 million
```

请注意，名为 aspect ratio 的键包含空格字符。可以使用数组访问 ([]) 运算符，但是如果尝试使用点运算符，则会生成错误。不建议在键名称中使用空格。

创建关联数组的第二种方式是使用 Array 构造函数（或任何动态类的构造函数），然后使用数组访问 ([]) 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。以下示例使用 Array 构造函数创建一个名为 monitorInfo 的关联数组，并添加一个名为 type 的键和一个名为 resolution 的键以及它们的值：

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

使用 Array 构造函数创建关联数组没有什么优势。即使使用 Array 构造函数或 Array 数据类型，也不能将 Array 类的 Array.length 属性或任何方法用于关联数组。最好将 Array 构造函数用于创建索引数组。

具有对象键的关联数组 （字典）

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 Dictionary 类创建使用对象而非字符串作为键的关联数组。这样的数组有时候也称作字典、哈希或映射。例如，考虑这样一个应用程序，它可根据 Sprite 对象与特定容器的关联确定 Sprite 对象的位置。可以使用 Dictionary 对象，将每个 Sprite 对象映射到一个容器。

以下代码创建三个用作 Dictionary 对象的键的 Sprite 对象实例。它为每个键分配了值 GroupA 或 GroupB。值可以是任意数据类型，但在此示例中， GroupA 和 GroupB 均为 Object 类的实例。然后，可以使用数组访问 ([]) 运算符访问与每个键关联的值，如下面的代码所示：

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// objects to use as keys
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// objects to use as values
var groupA:Object = new Object();
var groupB:Object = new Object();

// Create new key-value pairs in dictionary.
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

使用对象键循环访问

您可以使用 for..in 循环或 for each..in 循环来遍历 Dictionary 对象的内容。for..in 循环用于基于键进行遍历；而 for each..in 循环用于基于与每个键关联的值进行遍历。

使用 for..in 循环可以直接访问 Dictionary 对象的对象键。还可以使用数组访问 ([]) 运算符访问 Dictionary 对象的值。下面的代码使用前面的 groupMap 字典示例来说明如何使用 for..in 循环来遍历 Dictionary 对象：

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* output:
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

使用 for each..in 循环可以直接访问 Dictionary 对象的值。下面的代码也使用 groupMap 字典来说明如何使用 for each..in 循环来遍历 Dictionary 对象：

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* output:
[object Object]
[object Object]
[object Object]
*/
```

对象键和内存管理

Adobe® Flash® Player 和 Adobe® AIR™ 使用垃圾回收系统来恢复不再使用的内存。当对象不具有指向它的引用时，即可对其进行垃圾回收，并会在下次执行垃圾回收系统时恢复内存。例如，下面的代码创建一个新对象，并将对此对象的引用分配给变量 myObject：

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

只要有对此对象的引用，垃圾回收系统就不会恢复此对象占用的内存。如果更改 myObject 的值以使其指向其他对象或将其设置为值 null，并且没有对原始对象的其他引用，则可以对原始对象占用的内存进行垃圾回收。

如果将 myObject 用作 Dictionary 对象中的键，则会创建对原始对象的另一个引用。例如，下面的代码创建两个对象引用（myObject 变量和 myMap 对象中的键）：

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

若要使 myObject 引用的对象能够进行垃圾回收，您必须删除对它的所有引用。在此情况下，必须更改 myObject 的值并从 myMap 中删除 myObject 键，如以下代码所示：

```
myObject = null;
delete myMap[myObject];
```

或者，可以使用 Dictionary 构造函数的 useWeakReference 参数，以使所有字典键均成为“弱引用”。垃圾回收系统忽略弱引用，也就是说只具有弱引用的对象可以进行垃圾回收。例如，在下面的代码中，您不需要从 myMap 中删除 myObject 键就可以使该对象能够进行垃圾回收：

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // Make object eligible for garbage collection.
```

多维数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

多维数组将其他数组作为其元素。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。每个元素包含一个与 tasks 数组类似的索引数组，而该索引数组存储任务列表。在多维数组中，可以使用任意组合的索引数组和关联数组。以下部分中的示例使用了两个索引数组或由索引数组组成的关联数组。练习的时候，您也许需要采用其他组合。

两个索引数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用两个索引数组时，可以将结果呈示为表或电子表格。第一个数组的元素表示表的行，第二个数组的元素表示表的列。

例如，以下多维数组使用两个索引数组跟踪一周中每一天的任务列表。第一个数组 masterTaskList 是使用 Array 类构造函数创建的。此数组中的各个元素分别表示一周中的各天，其中索引 0 表示星期一，索引 6 表示星期日。可将这些元素当成是表的行。可通过为 masterTaskList 数组中创建的七个元素中的每个元素分配数组文本来创建每一天的任务列表。这些数组文本表示表的列。

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

可以使用数组访问 ([]) 运算符访问任意任务列表中的各个项。第一组括号表示一周的某一天，第二组括号表示这一天的任务列表。例如，若要检索星期三的列表中的第二项任务，请首先使用表示星期三的索引 2，然后使用表示列表中的第二项任务的索引 1。

```
trace(masterTaskList[2][1]); // output: dentist
```

若要检索星期日的列表中的第一项，请使用表示星期日的索引 6 和表示列表中的第一项任务的索引 0。

```
trace(masterTaskList[6][0]); // output: mow lawn
```

具有索引数组的关联数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要使单个数组的访问更加方便，可以使用关联数组表示一周的各天并使用索引数组表示任务列表。通过使用关联数组可以在引用一周中特定的一天时使用点语法，但要访问关联数组的每个元素还需额外进行运行时处理。以下示例使用关联数组作为任务列表的基础，并使用键和值对来表示一周中的每一天：

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

点语法通过避免使用多组括号改善了代码的可读性。

```
trace(masterTaskList.Wednesday[1]); // output: dentist
trace(masterTaskList.Sunday[0]);// output: mow lawn
```

可以使用 for..in 循环来遍历任务，但访问与每个键关联的值时必须使用数组访问 ([]) 运算符，而不是点语法。由于 masterTaskList 为关联数组，因而不一定会按照您所期望的顺序检索元素，如以下示例所示：

```
for (var day:String in masterTaskList)
{
    trace(day + ": " + masterTaskList[day])
}
/* output:
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

克隆数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Array 类不具有复制数组的内置方法。通过调用不带参数的 concat() 或 slice() 方法，可以创建数组的 " 浅副本 "。在浅副本中，如果原始数组具有对象元素，则仅复制指向对象的引用而非对象本身。与原始数组一样，副本也指向相同的对象。对对象所做的任何更改都会在两个数组中反映出来。

在 " 深副本 " 中，还将复制原始数组中的所有对象，从而使新数组和原始数组指向不同的对象。深度复制需要多行代码，通常需要创建函数。可以将此类函数作为全局实用程序函数或 Array 子类的方法来进行创建。

以下示例定义一个名为 clone() 的函数以执行深度复制。其算法采用了一般的 Java 编程技巧。此函数创建深副本的方法是：将数组序列化为 ByteArray 类的实例，然后将此数组读回到新数组中。此函数接受对象，因此既可以将此函数用于索引数组，又可以将其用于关联数组，如以下代码所示：

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

扩展 Array 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Array 类是少数不是最终类的核心类之一，也就是说您可以创建自己的 Array 子类。本部分提供了创建 Array 子类的示例，并讨论了在创建子类过程中会出现的一些问题。

如前所述， ActionScript 中的数组是不能指定数据类型的，但您可以创建只接受具有指定数据类型的元素的 Array 子类。以下部分中的示例定义名为 TypedArray 的 Array 子类，该子类中的元素限定为具有第一个参数中指定的数据类型的值。这里的 TypedArray 类仅用于说明如何扩展 Array 类，并不一定适用于生产，这有以下若干原因。第一，类型检查是在运行时而非编译时进行的。第二，当 TypedArray 方法遇到不匹配时，将忽略不匹配并且不会引发异常；当然，修改此方法使其引发异常也是很简单的。第三，此类无法防止使用数组访问运算符将任一类型的值插入到数组中。第四，其编码风格倾向于简洁而非性能优化。

注：可以使用此处介绍的方法创建指定类型的数组。但是，更好的方法是使用 Vector 对象。 Vector 实例是真正的指定类型的数组，在性能和其他一些方面比 Array 类或任何子类都好。本文所进行的讨论旨在演示如何创建 Array 子类。

声明子类

可以使用 extends 关键字来指示类为 Array 的子类。与 Array 类一样，Array 的子类应使用 dynamic 属性。否则，子类将无法正常发挥作用。

以下代码显示 TypedArray 类的定义，该类包含一个保存数据类型的常量、一个构造函数方法和四个能够将元素添加到数组的方法。此示例省略了各方法的代码，但在以后的部分中将列出这些代码并详加说明：

```
public dynamic class TypedArray extends Array
{
    private const dataType:Class;

    public function TypedArray(...args) {}

    AS3 override function concat(...args):Array {}

    AS3 override function push(...args):uint {}

    AS3 override function splice(...args) {}

    AS3 override function unshift(...args):uint {}
}
```

由于本示例中假定将编译器选项 -as3 设置为 true，而将编译器选项 -es 设置为 false，因而这四个被覆盖的方法均使用 AS3 命名空间而非 public 属性。这些是 Adobe Flash Builder 和 Adobe Flash Professional 的默认设置。

如果您是倾向于使用原型继承的高级开发人员，您可能会在以下两个方面对 TypedArray 类进行较小的改动，以使其在编译器选项 -es 设置为 true 的情况下进行编译。一方面，删除出现的所有 override 属性，并使用 public 属性替换 AS3 命名空间。另一方面，使用 Array.prototype 替换出现的所有四处 super。

TypedArray 构造函数

由于此子类构造函数必须接受一列任意长度的参数，从而造成了一种有趣的挑战。该挑战就是如何将参数传递给超类构造函数以创建数组。如果将一列参数作为数组进行传递，则超类构造函数会将其视为 Array 类型的一个参数，并且生成的数组长度始终只有 1 个元素。传递参数列表的传统处理方式是使用 Function.apply() 方法，此方法将参数数组作为第二个参数，但在执行函数时将其转换为一列参数。遗憾的是， Function.apply() 方法不能和构造函数一起使用。

剩下的唯一方法是在 TypedArray 构造函数中重新创建 Array 构造函数的逻辑。以下代码说明在 Array 类构造函数中使用的算法，您可以在 Array 子类构造函数中重复使用此算法：

```
public dynamic class Array
{
    public function Array(...args)
    {
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen;
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")");
            }
            length = ulen;
        }
        else
        {
            length = n;
            for (var i:int=0; i < n; i++)
            {
                this[i] = args[i]
            }
        }
    }
}
```

TypedArray 构造函数与 Array 构造函数中的大部分代码都相同，只在四个地方对代码进行了改动。其一，参数列表中新增了一个必需的 Class 类型参数，使用此参数可以指定数组的数据类型。其二，将传递给构造函数的数据类型分配给 dataType 变量。其三，在 else 语句中，在 for 循环之后为 length 属性赋值，以使 length 只包括相应类型的参数。其四，for 循环的主体使用 push() 方法的被覆盖版本，以便仅将正确数据类型的参数添加到数组中。以下示例显示 TypedArray 构造函数：

```
public dynamic class TypedArray extends Array
{
    private var dataType:Class;
    public function TypedArray(typeParam:Class, ...args)
    {
        dataType = typeParam;
        var n:uint = args.length
        if (n == 1 && (args[0] is Number))
        {
            var dlen:Number = args[0];
            var ulen:uint = dlen
            if (ulen != dlen)
            {
                throw new RangeError("Array index is not a 32-bit unsigned integer ("+dlen+")")
            }
            length = ulen;
        }
        else
        {
            for (var i:int=0; i < n; i++)
            {
                // type check done in push()
                this.push(args[i])
            }
            length = this.length;
        }
    }
}
```

TypedArray 覆盖的方法

TypedArray 类覆盖前述四种能够将元素添加到数组的方法。在每种情况下，被覆盖方法均添加类型检查，这种检查可以防止添加不正确数据类型的元素。然后，每种方法均调用其自身的超类版本。

push() 方法使用 for..in 循环来遍历参数列表，并对每个参数执行类型检查。可以使用 splice() 方法，从 args 数组中删除所有不是正确类型的参数。在 for..in 循环结束后， args 数组仅包含 dataType 类型的值。然后对更新后的 args 数组调用 push() 的超类版本，如以下代码所示：

```
AS3 override function push(...args):uint
{
    for (var i:* in args)
    {
        if (!(args[i] is dataType))
        {
            args.splice(i,1);
        }
    }
    return (super.push.apply(this, args));
}
```

concat() 方法创建一个名为 passArgs 的临时 TypedArray 来存储通过类型检查的参数。这样，便可以重复使用 push() 方法中的类型检查代码。for..in 循环用于遍历 args 数组，并为每个参数调用 push()。由于将 passArgs 指定为类型 TypedArray，因而将执行 push() 的 TypedArray 版本。然后， concat() 方法将调用其自身的超类版本，如以下代码所示：

```
AS3 override function concat(...args):Array
{
    var passArgs:TypedArray = new TypedArray(dataType);
    for (var i:* in args)
    {
        // type check done in push()
        passArgs.push(args[i]);
    }
    return (super.concat.apply(this, passArgs));
}
```

splice() 方法使用任意一列参数，但前两个参数始终引用索引号和要删除的元素个数。这就是为什么被覆盖的 splice() 方法仅对索引位置 2 或其以后的 args 数组元素执行类型检查。该代码中一个有趣的地方是，for 循环中的 splice() 调用看上去似乎是递归调用，但实际上并不是递归调用，这是由于 args 的类型是 Array 而非 TypedArray，也就是说， args.splice() 调用是对此方法超类版本的调用。在 for..in 循环结束后，args 数组中将只包含索引位置 2 或其以后位置中具有正确类型的值，并且 splice() 会调用其自身的超类版本，如下面的代码所示：

```
AS3 override function splice(...args):*
{
    if (args.length > 2)
    {
        for (var i:int=2; i< args.length; i++)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
    }
    return (super.splice.apply(this, args));
}
```

用于将元素添加到数组开头的 unshift() 方法也可以接受任意一列参数。被覆盖的 unshift() 方法使用的算法与 push() 方法使用的算法非常类似，如下面的示例代码所示：

```
    AS3 override function unshift(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.unshift.apply(this, args));
    }
}
```

数组示例：播放列表

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此 PlayList 示例在管理歌曲列表的音乐播放列表应用程序环境中展示了使用数组的多种技巧。这些方法包括：

- 创建索引数组
- 向索引数组中添加项
- 使用不同的排序选项按照不同的属性对对象数组排序
- 将数组转换为以字符分隔的字符串

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/PlayList 文件夹中找到 PlayList 应用程序文件。该应用程序包含以下文件：

文件	说明
PlayList.mxml 或 PlayList.fla	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/playlist/PlayList.as	表示歌曲列表的类。该类使用 Array 存储列表，并管理列表项的排序。
com/example/programmingas3/playlist/Song.as	表示一首歌曲信息的值对象。由 PlayList 类管理的项为 Song 实例。
com/example/programmingas3/playlist/SortProperty.as	伪枚举，其可用值代表 Song 类的属性，可以根据这些属性对 Song 对象列表排序。

PlayList 类概述

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

PlayList 类管理一组 Song 对象。它具有一些公共方法，可将歌曲添加到播放列表 （addSong() 方法）以及对列表中的歌曲排序 （sortList() 方法）。此外，它还包括一个只读存取器属性 songList，可提供对播放列表中实际歌曲集的访问。在内部，PlayList 类使用私有 Array 变量跟踪其歌曲：

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

除了 PlayList 类用来跟踪其歌曲列表的 _songs Array 变量以外，还有另外两个私有变量，分别用来跟踪是否需要对列表排序 (_needToSort) 以及在给定时间对列表进行排序所依据的属性 (_currentSort)。

跟所有对象一样，声明 Array 实例只做了创建 Array 工作的一半。在访问 Array 实例的属性或方法之前，必须在 PlayList 类的构造函数中完成对该实例的实例化。

```
    public function PlayList()
    {
        this._songs = new Array();
        // Set the initial sorting.
        this.sortList(SortProperty.TITLE);
    }
```

该构造函数的第一行用于实例化 _songs 变量，以使其处于待用状态。此外，还会调用 sortList() 方法来设置初始排序所依据的属性。

向列表中添加歌曲

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用户在应用程序中输入新歌曲后，数据条目表单上的代码将调用 PlayList 类的 addSong() 方法。

```
    /**
     * Adds a song to the playlist.
     */
    public function addSong(song:Song):void
    {
        this._songs.push(song);
        this._needToSort = true;
    }
```

在 addSong() 内，将调用 _songs 数组的 push() 方法，以便将传递给 addSong() 的 Song 对象添加为该数组中的新元素。不管之前应用的是哪种排序方法，使用 push() 方法时，都会将新元素添加到数组末尾。也就是说，调用 push() 方法后，歌曲列表的排序很可能不正确，因此将 _needToSort 变量设置为 true。理论上说，可以立即调用 sortList() 方法，而无需跟踪是否要在给定时间对列表进行排序。实际上，不需要立即对歌曲列表排序，除非要对其进行检索。通过延迟排序操作，应用程序不会执行不必要的排序，例如，在将几首歌曲添加到列表中且不需要立即对其进行检索时。

对歌曲列表排序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

由于由 PlayList 管理的 Song 实例为复杂对象，因而此应用程序的用户可能要根据不同的属性 （例如，歌曲名称或发行年份）来对播放列表排序。在 PlayList 应用程序中，对歌曲列表排序的任务由以下三部分组成：确定列表排序所依据的属性，指出按照该属性排序时应使用的排序操作，以及执行实际排序操作。

排序属性

Song 对象跟踪若干属性，包括歌曲名称、歌手、发行年份、文件名和用户选择的歌曲所属流派。在这些属性当中，只有前三个可用于排序。为了方便开发人员，此示例中包括 SortProperty 类，此类与枚举作用相同，其值表示可用于排序的属性。

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

SortProperty 类包含 TITLE、ARTIST 和 YEAR 三个常量，其中的每个常量都存储一个字符串，该字符串中包含可用于排序的相关 Song 类属性的实际名称。在代码的其余部分，只要指示排序属性，都是使用枚举成员完成的。例如，在 PlayList 构造函数中，最初通过调用 sortList() 方法对列表排序，如下所示：

```
// Set the initial sorting.
this.sortList(SortProperty.TITLE);
```

由于将排序属性指定为 SortProperty.TITLE，因而将根据歌曲名称对歌曲排序。

依据属性排序和指定排序选项

对歌曲列表排序的实际工作是由 PlayList 类在 sortList() 方法中进行的，如下所示：

```
/**
 * Sorts the list of songs according to the specified property.
 */
public function sortList(sortProperty:SortProperty):void
{
    ...
    var sortOptions:uint;
    switch (sortProperty)
    {
        case SortProperty.TITLE:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // Perform the actual sorting of the data.
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // Save the current sort property.
    this._currentSort = sortProperty;

    // Record that the list is sorted.
    this._needToSort = false;
}
```

如果根据歌名或歌手排序，则应按照字母顺序排序；若要根据年份排序，则按照数字顺序排序最为合理。 switch 语句用于根据 sortProperty 参数中指定的值定义合适的排序选项，此选项将存储在 sortOptions 变量中。这里，再次使用命名的枚举成员而非硬编码值来区分不同属性。

确定排序属性和排序选项之后，将通过调用 sortOn() 方法并将上述两个值作为参数传递来完成对 _songs 数组的排序。对歌曲列表进行排序之后，记录当前排序属性。

将数组元素组合为以字符分隔的字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在本示例中，数组除了用于在 PlayList 类中维护歌曲列表以外，还用于在 Song 类中帮助管理指定歌曲所属的流派的列表。请考虑 Song 类定义中的以下代码片断：

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String, genres:Array)
{
    ...
    // Genres are passed in as an array
    // but stored as a semicolon-separated string.
    this._genres = genres.join(";");
}
```

创建新 Song 实例时，会将用于指定歌曲所属流派的 genres 参数定义为 Array 实例。这有助于将多个流派组合为一个可以传递给构造函数的变量。但在内部，Song 类在 _genres 私有变量中以分号分隔的 String 实例形式来保存流派。通过调用 join() 方法（使用文本字符串值 ";" 作为指定分隔符）， Array 参数将转换为以分号分隔的字符串。

通过使用相同的标记， genres 存取器可将流派作为 Array 进行设置或检索：

```
    public function get genres():Array
    {
        // Genres are stored as a semicolon-separated String,
        // so they need to be transformed into an Array to pass them back out.
        return this._genres.split(";");
    }
    public function set genres(value:Array):void
    {
        // Genres are passed in as an array,
        // but stored as a semicolon-separated string.
        this._genres = value.join(";");
    }
```

genresset 存取器的行为与构造函数完全相同；它接受 Array 并调用 join() 方法以将其转换为以分号分隔的 String。 get 存取器执行相反的操作：调用 _genres 变量的 split() 方法，使用指定分隔符 （采用与前面相同的文本字符串值 ";"）将 String 拆分为值数组。

第 4 章：处理错误

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

“ 处理 ” 错误意味着您在自己的应用程序中构建逻辑来响应或修复错误。当编译应用程序时，或编译的应用程序正在运行时，可能会生成错误。应用程序在处理错误时，当遇到错误时会出现某种响应，而不是没有响应 （当任何导致错误的进程在无提示的情况下失败时）。正确使用错误处理有助于防止应用程序和应用程序的使用者执行其他意外行为。

不过，错误处理涵盖的内容很广，它包括对编译期间或应用程序运行时引发的许多种错误予以响应。此讨论主要介绍在 ActionScript 3.0 环境中，如何处理运行时错误 （在应用程序运行时引发的错误）、可能生成的不同类型的错误以及错误处理系统的优点。

错误处理基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

运行时错误是指阻止 ActionScript 内容按预期运行的 ActionScript 代码错误。要确保用户可以顺利运行您的 ActionScript 代码，请在应用程序中编写可以处理错误的代码，该代码可以修复、解决错误，或者至少可以让用户了解已发生了此问题。此过程称为 “ 错误处理 ”。

错误处理涵盖的内容很广，它包括对编译期间或应用程序运行时引发的许多种错误予以响应。在编译时出现的错误通常比较容易识别 — 您只需修正这些错误即可完成创建 SWF 文件的过程。

运行时错误可能更难于检测，因为必须实际运行错误代码才会发生这些错误。如果程序片断包含几个代码分支 （如 if..then..else 语句），利用实际用户可能使用的所有可能的输入值测试每种可能的情况，以确认代码没有错误。

运行时错误可以分为以下两类：“ 程序错误 ” 是指 ActionScript 代码中的错误，如为方法参数指定了错误的数据类型；“ 逻辑错误 ” 是指程序的逻辑 （数据检查和值处理）错误，如在银行业应用程序中使用错误的公式来计算利率。同样，通过事先仔细地测试应用程序，通常可以检测到并纠正这两种类型的错误。

理想情况下，您希望在将应用程序发布到最终用户之前找出并消除其中的所有错误。但是，并非所有错误都是可以预见或避免的。例如，假设您的 ActionScript 应用程序是从您无法控制的特定网站加载信息。如果该网站在某一时刻不可用，则依赖于该外部数据的应用程序部分将无法正确运行。错误处理的最重要方面包括应对这些未知情况并妥善进行处理。用户需要继续使用您的应用程序，或者至少获得一条友好的错误消息来解释应用程序为什么无法运行。

ActionScript 使用以下两种方式来表示运行时错误：

- 错误类：很多错误都具有一个关联的错误类。当发生错误时， Flash 运行时 （如 Flash Player 或 Adobe AIR）将创建与该错误关联的特定错误类的实例。代码可以使用该错误对象中包含的信息对错误进行相应的响应。
- 错误事件：有时，当 Flash 运行时正常触发事件时，也会发生错误。在这类情况下，触发的是错误事件。每个错误事件都有一个与之关联的类， Flash 运行时会将该类的实例传递给订阅了该错误事件的方法。

要确定特定方法是否会触发错误或错误事件，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中该方法的条目。

重要概念和术语

以下参考列表包含与错误处理例程编程相关的重要术语：

异步 不提供即时结果的程序命令 （如方法调用），而是以事件形式提供结果 （或错误）。

捕捉 当异常 （一种运行时错误）发生时，代码识别了该异常，则表示该代码 “ 捕捉 ” 了异常。捕获异常后， Flash 运行时会停止通知其他 ActionScript 代码发生了异常。

调试器版本 Flash 运行时的特殊版本 （例如 Flash Player 调试器版本或 AIR Debug Launcher (ADL)），其中包含通知用户发生运行时错误的代码。在 Flash Player 或 Adobe AIR 标准版（大多数用户使用的版本）中，将忽略 ActionScript 代码未处理的错误。在 Adobe Flash CS4 Professional 和 Adobe Flash Builder 提供的调试器版本中，当出现未处理的错误时，会出现警告消息。

异常 在应用程序运行时发生的错误， Flash 运行时无法自行解决此问题。

重新引发 当代码捕捉到异常时， Flash 运行时不再通知其他对象发生了异常。如果让其他对象收到发生异常的消息很重要，则代码必须重新引发异常才能再次启动通知进程。

同步 可提供即时结果 （或立即引发错误）的程序命令 （例如方法调用），这意味着此响应可以在同一代码块使用。

引发 通知 Flash 运行时 （并因此通知其他对象和 ActionScript 代码）发生了错误的行为称为 " 引发 " 错误。

错误类型

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

开发和运行应用程序时，您会遇到不同类型的错误和错误术语。下面列出了主要的错误类型和术语：

- 编译时错误，这类错误在代码编译期间由 ActionScript 编译器引发。当代码中的语法问题导致应用程序无法生成时即会发生编译时错误。
- 运行时错误，这类错误在应用程序编译之后运行时发生。运行时错误指在 Flash 运行时 （如 Adobe Flash Player 或 Adobe AIR）中播放 SWF 文件时产生的错误。大多数情况下，您会在出现运行时错误时处理这些错误，将错误报告给用户，并采取相应的步骤让应用程序继续运行。如果遇到的错误是致命错误，例如无法连接到远程网站或无法加载所需的数据，您可以使用错误处理让应用程序顺利地完成运行。
- 同步错误是在调用函数时发生的运行时错误 — 例如，当您尝试使用特定方法但传递到该方法的参数无效时， Flash 运行时会引发异常。多数错误都是在语句执行时同步发生，并且控制流会立即传递给最适用的 catch 语句。

  例如，以下代码将会引发一个运行时错误，原因是在程序试图上载文件之前没有调用 browse() 方法：

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
      fileRef.upload(new URLRequest("http://www.yourdomain.com/fileupload.cfm"));
  }
  catch (error:IllegalOperationError)
  {
      trace(error);
      // Error #2037: Functions called in incorrect sequence, or earlier
      // call was unsuccessful.
  }
  ```

  在本例中，将同步引发一个运行时错误，原因是 Flash Player 断定在试图上载文件之前没有调用 browse() 方法。

  有关同步错误处理的详细信息，请参阅第 47 页的 " 在应用程序中处理同步错误 "。

- 异步错误是发生在正常程序流之外的运行时错误。这些错误可生成事件，而事件侦听器可对其进行捕获。在异步操作中，函数发起操作但并不等待操作完成。您可以创建错误事件侦听器，以等待应用程序或用户尝试某操作。如果操作失败，则使用事件侦听器捕捉错误并响应错误事件。然后，该事件侦听器调用一个事件处理函数，以便通过一种有益的方式来响应错误事件。例如，事件处理函数可以启动一个对话框，以提示用户解决该错误。

  下面以前面提到的文件上载同步错误为例。如果在文件上载开始之前成功调用了 browse() 方法，则 Flash Player 会调度若干个事件。例如，上载开始时，将调度 open 事件。文件上载操作成功完成时，将调度 complete 事件。由于事件处理是异步进行的 （即，不在特定、已知、预先指定的时间发生），因此使用 addEventListener() 方法可以侦听这些特定的事件，如以下代码所示：

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

有关异步错误处理的详细信息，请参阅第 52 页的 “ 响应错误事件和状态 ”。

- 未捕获的异常，这类错误在引发后并没有相应的逻辑 （如 catch 语句）来响应它。应用程序引发错误后，如果在当前级别或更高级别找不到适当的 catch 语句或事件处理函数来处理错误，则认为该错误是未捕获的异常。

  如果出现未捕获的错误，运行时会调度 uncaughtError 事件。此事件也称为 “ 全局错误处理程序 ”。 SWF 的 UncaughtErrorEvents 对象调度此事件，可通过 LoaderInfo.uncaughtErrorEvents 属性获得该对象。如果没有为 uncaughtError 事件注册任何侦听器，只要未捕获的错误不停止 SWF，运行时就会忽略未捕获的错误并尝试继续运行。

  除调度 uncaughtError 事件之外，Flash 运行时的调试器版本将通过终止当前脚本来响应未捕获的错误。然后，在 trace 语句输出中显示未捕获的错误，或将错误消息写入日志文件。如果异常对象为 Error 类的实例或其子类之一，则输出中也显示堆栈跟踪信息。有关使用 Flash 运行时调试版的详细信息，请参阅第 47 页的 “ 使用 Flash 运行时的调试版 ”。

  注：处理 uncaughtError 事件时，如果错误事件由 uncaughtError 事件处理函数引发，该事件处理函数将被多次调用。这将导致异常的无限循环。建议您避免这样的情况出现。

ActionScript 3.0 中的错误处理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

由于许多应用程序在没有构建错误处理逻辑的情况下也可以运行，因此开发人员往往会拖延在其应用程序中构建错误处理逻辑。但如果没有错误处理逻辑，应用程序很容易终止运行，或因某一操作无法按预期执行而让用户感到烦恼。 ActionScript 2.0 具有一个 Error 类，可用来在自定义函数中构建逻辑，以便引发具有特定消息的异常。由于错误处理对于构建用户友好的应用程序至关重要，因此 ActionScript 3.0 提供了一个扩展的体系结构用来捕获错误。

注：虽然用于 Adobe Flash Platform 的 ActionScript 3.0 参考中记录了许多方法引发的异常，但不可能包括每种方法可能引发的所有异常。对于某个方法，即使方法描述中确实列出了它可能引发的某些异常，但仍可能存在方法描述中未明确阐述的语法错误或其他问题而引发的异常。

ActionScript 3.0 错误处理的构成元素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 提供了许多用来进行错误处理的工具，其中包括：

- 错误类。 ActionScript 3.0 中包括大量 Error 类，扩展了可产生错误对象的情形范围。每个错误类都可以帮助应用程序处理和响应特定的错误条件，无论这些错误条件是与系统错误相关 （如 MemoryError 条件）、与代码编写错误相关 （如 ArgumentError 条件）、与网络和通信错误相关 （如 URIError 条件），还是与其他情形相关。有关每个类的详细信息，请参阅第 55 页的 " 比较错误类 "。
- 更少的无提示失败。在 Flash Player 以前的版本中，只有明确使用了 throw 语句，才会产生并报告错误。对于 Flash Player 9 和最新的 Flash 运行时，本机 ActionScript 方法和属性会引发运行时错误。这些错误允许您更有效处理发生的异常，然后逐个响应每个异常。
- 在调试期间显示清楚的错误消息。使用 Flash 运行时调试器版本时，有问题的代码或情形会生成可靠的错误消息，这有助于您轻松识别特定代码块失败的原因。这些消息有助于更高效地修复错误。有关详细信息，请参阅第 47 页的 " 使用 Flash 运行时的调试版 "。
- 精确的错误指示可以向用户显示清楚的错误消息。在 Flash Player 的先前版本中，如果 upload() 调用不成功，则 FileReference.upload() 方法会返回布尔值 false，表示发生了五个可能错误中的一种。如果在 ActionScript 3.0 中调用 upload() 方法时出错，四个特有的错误有助于向最终用户显示更准确的错误消息。
- 经过优化完善的错误处理。一些明显的错误是由许多常见原因引发的。例如，在 ActionScript 2.0 中，在填充 FileReference 对象之前， name 属性的值为 null （因此，在您可以使用或显示 name 属性前，请确保已将此值设置为 null 之外的值）。在 ActionScript 3.0 中，如果在填充 name 属性之前试图访问该属性， Flash Player 或 AIR 将引发 IllegalOperationError，通知您尚未设定该值，这时您可以使用 try..catch..finally 块来处理该错误。有关详细信息，请参阅第 47 页的 " 使用 try..catch..finally 语句 "。
- 没有明显的性能缺点。与以前的 ActionScript 版本相比，使用 try..catch..finally 块仅占用很少的额外资源或根本不占用任何额外资源。
- 允许针对特定异步错误事件构建侦听器的 ErrorEvent 类。有关详细信息，请参阅第 52 页的 " 响应错误事件和状态 "。

错误处理策略

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

只要应用程序未遇到会导致问题的情况，则即使未在程序代码中构建错误处理逻辑，应用程序仍然可以成功运行。但是，如果您没有主动处理错误，并且应用程序确实遇到了问题，用户在应用程序失败时将无从知道原因所在。

在应用程序中进行错误处理有几种不同的方式。下面概括介绍三种主要的错误处理方式：

- 使用 try..catch..finally 语句。这些语句可以在发生同步错误时捕捉这些错误。可以将语句嵌套在一个层次中，以便在不同的代码执行级别捕获异常。有关详细信息，请参阅第 47 页的 " 使用 try..catch..finally 语句 "。
- 创建您自己的自定义错误对象。您可以使用 Error 类创建您自己的自定义错误对象，以便跟踪应用程序中内置错误类型未涵盖的特定操作。然后，您可以对自定义错误对象使用 try..catch..finally 语句。有关详细信息，请参阅第 51 页的 " 创建自定义错误类 "。
- 编写用以响应错误事件的事件侦听器和处理函数。使用此策略，您可以创建全局错误处理程序以处理类似事件，而无需在 try..catch..finally 代码块中复制许多代码。您还可以使用此方法来捕获异步错误。有关详细信息，请参阅第 52 页的 " 响应错误事件和状态 "。

使用 Flash 运行时的调试版

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe 为开发人员提供了一个特殊版本的 Flash 运行时来帮助进行调试。安装 Adobe Flash Professional 或 Adobe Flash Builder 后，您可以获得一个 Flash Player 调试版。在安装其中任一工具时，您还可以获得用于调试 Adobe AIR 应用程序的实用工具 （该工具称为 ADL），该工具也包含在 Adobe AIR SDK 中。

Flash Player 和 Adobe AIR 的调试版与发行版在错误指示方面有明显的不同。调试版显示错误类型 （如一般错误、 IOError 或 EOFError）、错误编号和可读错误消息。发行版则仅显示错误类型和错误编号。例如，请看以下代码：

```
try
{
    tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
    tf.text = error.toString();
}
```

如果 readBoolean() 方法在 Flash Player 调试器版本中引发 EOFError，将在 tf 文本字段中显示下列消息：“EOFError: 错误 : #2030: 已到文件尾 ”。)

同样的代码在 Flash Player 或 Adobe AIR 的发行版中则会显示以下文字：“EOFError: 错误 #2030。 ”

注：调试器播放器将广播名为 “allComplete” 的事件；避免使用名称 “allComplete” 创建自定义事件。否则，您在调试时会遇到不可预测的行为。

为了使资源和大小在发行版中达到最小，因此未提供错误消息字符串。您可以在文档 （用于 Adobe Flash Platform 的 ActionScript 3.0 参考的附录）中查找与错误消息关联的错误编号。或者，也可以使用 Flash Player 和 AIR 调试版重现错误以查看完整的错误消息。

在应用程序中处理同步错误

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

最常见的错误处理是同步错误处理逻辑，您可以在处理逻辑中将适当的语句插入代码，以便在应用程序运行时捕获同步错误。这种错误处理可以让应用程序在功能失败时注意到发生运行时错误并从错误中恢复。同步错误捕获逻辑包括 try..catch..finally 语句，从字面意义上看，这种方式先尝试 (try) 某个操作，然后捕获 (catch) 来自 Flash 运行时的任何错误响应，最后 (finally) 执行另外的操作来处理失败的操作。

使用 try..catch..finally 语句

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

处理同步运行时错误时，可以使用 try..catch..finally 语句来捕获错误。当发生运行时错误时， Flash 运行时将引发异常，这意味着它将暂停正常的操作而创建一个 Error 类型的特殊对象。 Error 对象随后会被引发到第一个可用的 catch 块。

try 语句将有可能产生错误的语句括在一起。 catch 语句应始终与 try 语句一起使用。如果在 try 语句块的其中一个语句中检测到错误，则将运行附加到该 try 语句的 catch 语句。

finally 语句中包含无论 try 块中是否出错均会运行的语句。如果没有错误， finally 块中的语句将在 try 语句块执行完毕之后执行。如果有错误，则首先执行相应的 catch 语句，然后执行 finally 块中的语句。

以下代码说明了使用 try..catch..finally 语句的语法：

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

每个 catch 语句识别它要处理的特定类型的异常。 catch 语句指定的错误类只能是 Error 类的子类。将按顺序检查每个 catch 语句。只运行与所引发的错误类型匹配的第一个 catch 语句。换句话说，如果您首先检查更高级别的 Error 类，然后检查 Error 类的子类，则只有更高级别的 Error 类匹配。以下代码说明了这一点：

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
```

上面这段代码的输出如下：

```
<Error> I am an ArgumentError
```

为正确捕获 ArgumentError，请确保首先列出最具体的错误类型，然后再列出较为一般的错误类型，如以下代码所示：

```
try
{
    throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
    trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
    trace("<Error> " + error.message);
}
```

ActionScript API 中有几种方法和属性，如果在执行时它们遇到错误，便会引发运行时错误。例如， Sound 类中的 close() 方法，它如果无法关闭音频流，便会引发 IOError 错误，如以下代码所示：

```
var mySound:Sound = new Sound();
try
{
    mySound.close();
}
catch (error:IOError)
{
    // Error #2029: This URLStream object does not have an open stream.
}
```

随着对用于 Adobe Flash Platform 的 ActionScript 3.0 参考的逐渐熟悉，您会发现哪些方法会引发异常。详见每个方法的说明。

throw 语句

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果 Flash 运行时在应用程序运行时遇到错误，便会引发异常。此外，您也可以自己使用 throw 语句明确引发异常。如果是明确引发错误，Adobe 建议您引发 Error 类或其子类的实例。以下代码所展示的 throw 语句引发一个 Error 类实例 MyErr，并且最后调用一个函数 myFunction() 在引发错误之后进行响应：

```
var MyError:Error = new Error("Encountered an error with the numUsers value", 99);
var numUsers:uint = 0;
try
{
    if (numUsers == 0)
    {
        trace("numUsers equals 0");
    }
}
catch (error:uint)
{
    throw MyError; // Catch unsigned integer errors.
}
catch (error:int)
{
    throw MyError; // Catch integer errors.
}
catch (error:Number)
{
    throw MyError; // Catch number errors.
}
catch (error:*)
{
    throw MyError; // Catch any other error.
}
finally
{
    myFunction(); // Perform any necessary cleanup here.
}
```

请注意， catch 语句进行了排序，以便先列出最具体的数据类型。如果首先列出的是 Number 数据类型的 catch 语句，则 uint 数据类型和 int 数据类型的 catch 语句均不会运行。

注：在 Java 编程语言中，每个可以引发异常的函数都必须先声明这一点，并在附加到函数声明的 throw 子句中列出该函数可以引发的异常类。 ActionScript 不要求您声明由函数引发的异常。

显示简单错误消息

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

新的异常和错误事件模型的一个最大优点就是：它可以让您向用户告知操作失败的时间和原因。您的工作是编写用来显示消息的代码和在响应中提供选项。

以下代码使用一个简单的 try..catch 语句在一个文本字段中显示错误：

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
                <costCenter>1-234</costCenter>
            </EmpCode>;

        public function SimpleError()
        {
            try
            {
                if (employee.costCenter.length() != 1)
                {
                    throw new Error("Error, employee must have exactly one cost center assigned.");
                }
            }
            catch (error:Error)
            {
                var errorMessage:TextField = new TextField();
                errorMessage.autoSize = TextFieldAutoSize.LEFT;
                errorMessage.textColor = 0xFF0000;
                errorMessage.text = error.message;
                addChild(errorMessage);
            }
        }
    }
}
```

与 ActionScript 的先前版本相比， ActionScript 3.0 使用了更加丰富的错误类和内置编译器错误，因此可以提供有关失败原因的更多信息。通过此信息，您可以构建更加稳定且具有更佳错误处理能力的应用程序。

重新引发错误

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

构建应用程序时，有时候，如果无法正确处理错误，则需要重新引发该错误。例如，以下代码使用一个嵌套的 try..catch 块，它在嵌套的 catch 块无法处理错误时重新引发一个自定义的 ApplicationError：

```
try
{
    try
    {
        trace("<< try >>");
        throw new ApplicationError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

上面这个代码片断的输出如下：

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

嵌套的 try 块引发一个自定义的 ApplicationError 错误，该错误由后续 catch 块捕获。此嵌套的 catch 块会尝试处理错误，如果不成功，则将 ApplicationError 对象引发到包含此 catch 块的 try..catch 块中来进行处理。

创建自定义错误类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以通过扩展其中一种标准的错误类，在 ActionScript 中创建您自己的专用错误类。有多种原因需要创建您自己的错误类：

- 识别应用程序特有的错误或错误组。

  例如，除由 Flash 运行时捕获的那些错误外，您还可以采取其他操作来处理由自己的代码引发的错误。您可以创建 Error 类的一个子类，以便在 try..catch 块中跟踪新的错误数据类型。

- 为应用程序生成的错误提供特有的错误显示能力。

  例如，可以创建一个以某种方式设置错误消息格式的新的 toString() 方法。还可以定义一个 lookupErrorString() 方法，该方法获取错误代码并根据用户的语言首选参数查找适当的消息。

专用的错误类必须扩展 ActionScript 的核心错误类。以下是一个扩展了 Error 类的专用 AppError 类示例：

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

以下是在项目中使用 AppError 的一个示例：

```
try
{
    throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
    trace(error.errorID + ": " + error.message)
}
```

注：如果要在子类中覆盖 Error.toString() 方法，请为其提供一个 ... （其余）参数。 ActionScript 3.0 所依据的 ECMAScript 语言规范通过这种方式定义 Error.toString() 方法， ActionScript 3.0 通过同样的方式定义该方法以实现向后兼容。因此，当您覆盖 Error.toString() 方法时，请与参数完全匹配。在运行时，您不希望将任何参数传递给 toString() 方法，因为这些参数都会被忽略。

响应错误事件和状态

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中，对错误处理最为明显的一项改进就是支持对应用程序运行时出现的异步错误予以响应。（有关异步错误的定义，请参阅第 44 页的 “ 错误类型 ”。）

可以创建事件侦听器和事件处理函数来响应错误事件。对于许多类来说，它们调度错误事件的方式与调度其他事件的方式相同。例如，一般情况下， XMLSocket 类实例调度三种类型的事件：Event.CLOSE、 Event.CONNECT 和 DataEvent.DATA。但是，当发生问题时， XMLSocket 类还可以调度 IOErrorEvent.IOError 或 SecurityErrorEvent.SECURITY_ERROR。有关事件侦听器和事件处理函数的详细信息，请参阅第 106 页的 “ 处理事件 ”。

错误事件分为两类：

- 扩展 ErrorEvent 类的错误事件

  flash.events.ErrorEvent 类包含用于管理与网络和通信操作 （在运行应用程序中）有关的错误的属性和方法。 AsyncErrorEvent、IOErrorEvent 和 SecurityErrorEvent 类扩展了 ErrorEvent 类。如果您使用的是 Flash 运行时调试器版本，则会出现一个对话框，向您通知播放器在运行时遇到的没有侦听器函数的任何错误事件。

- 基于状态的错误事件

  基于状态的错误事件与网络和通信类的 netStatus 和 status 属性有关。如果 Flash 运行时在读写数据时遇到问题，netStatus.info.level 或 status.level 属性 （取决于使用的类对象）的值将被设置为值 "error"。可以通过检查事件处理函数中的 level 属性是否包含值 "error" 来响应此错误。

使用错误事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ErrorEvent 类及其子类包含用于处理 Flash 运行时尝试读写数据时调度的错误的错误类型。

下面的示例同时使用了 try..catch 语句和错误事件处理函数来显示在尝试读取本地文件时检测到的任何错误。您可以在由 “ 在此处添加您的错误处理代码 ” 注释所指示的位置处添加更复杂的处理代码，以便为用户提供处理选项或者自动处理错误：

```
package
{
    import flash.display.Sprite;
    import flash.errors.IOError;
    import flash.events.IOErrorEvent;
    import flash.events.TextEvent;
    import flash.media.Sound;
    import flash.media.SoundChannel;
    import flash.net.URLRequest;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class LinkEventExample extends Sprite
    {
        private var myMP3:Sound;
        public function LinkEventExample()
        {
            myMP3 = new Sound();
            var list:TextField = new TextField();
            list.autoSize = TextFieldAutoSize.LEFT;
            list.multiline = true;
            list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
            list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
            addEventListener(TextEvent.LINK, linkHandler);
            addChild(list);
        }

        private function playMP3(mp3:String):void
        {
            try
            {
                myMP3.load(new URLRequest(mp3));
                myMP3.play();
            }
            catch (err:Error)
            {
                trace(err.message);
                // your error-handling code here
            }
            myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
        }

        private function linkHandler(linkEvent:TextEvent):void
        {
            playMP3(linkEvent.text);
            // your error-handling code here
        }

        private function errorHandler(errorEvent:IOErrorEvent):void
        {
            trace(errorEvent.text);
            // your error-handling code here
        }
    }
}
```

使用状态更改事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

对于支持 level 属性的类， Flash 运行时会在应用程序运行时动态更改 netStatus.info.level 或 status.level 属性的值。具有 netStatus.info.level 属性的类有 NetConnection、 NetStream 和 SharedObject。具有 status.level 属性的类有 HTTPStatusEvent、Camera、Microphone 和 LocalConnection。可以编写一个处理函数来响应 level 值的更改并跟踪通信错误。

以下示例使用 netStatusHandler() 函数测试 level 属性的值。如果 level 属性指示遇到错误，该代码将跟踪消息 “Video stream failed” （视频流失败）。

```
package
{
    import flash.display.Sprite;
    import flash.events.NetStatusEvent;
    import flash.events.SecurityErrorEvent;
    import flash.media.Video;
    import flash.net.NetConnection;
    import flash.net.NetStream;

    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, securityErrorHandler);
            connection.connect(null);
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if (event.info.level == "error")
            {
                trace("Video stream failed")
            }
            else
```

```
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

比较错误类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 提供了一些预定义的 Error 类。但您也可以在自己的代码中使用相同的 Error 类。在 ActionScript 3.0 中，有两种主要类型的错误类：ActionScript 核心错误类和 flash.error 包错误类。 flash.error 包中包含其他有助于 ActionScript 3.0 应用程序进行开发和调试的类。

核心错误类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

核心错误类包括 Error、 ArgumentError、 EvalError、 RangeError、 ReferenceError、 SecurityError、 SyntaxError、 TypeError、 URIError 和 VerifyError 类。其中的每个类均位于顶级命名空间中。

类名称	说明	备注
Error	Error 类用于引发异常，并且是 ECMAScript 中定义的其他异常类的基类，这些异常类包括 EvalError、RangeError、 ReferenceError、 SyntaxError、TypeError 和 URIError。	Error 类用作所有运行时错误的基类，并建议将其用作任何自定义错误类的基类。
ArgumentError	ArgumentError 类表示在函数调用期间提供的参数与为该函数定义的参数不一致时发生的错误。	以下是一些参数错误示例： • 向方法提供的参数过少或过多。 • 参数应是某个枚举值，但实际上不是。
EvalError	如果为 Function 类的构造函数传递了任何参数，或者用户代码调用 eval() 函数，则会引发 EvalError 异常。	在 ActionScript 3.0 中，已取消对 eval() 函数的支持，并在尝试使用该函数时出错。 Flash Player 的先前版本使用 eval() 函数来按照名称访问变量、属性、对象或影片剪辑。
RangeError	如果数值在可接受范围之外，将引发 RangeError 异常。	例如，如果延迟为负或无限，Timer 类就会引发 RangeError。试图在无效深度处添加显示对象也会引发 RangeError。

类名称	说明	备注
ReferenceError	如果试图对密封 （非动态）对象引用未定义的属性，则会引发 ReferenceError 异常。试图访问 undefined 的属性时， ActionScript 3.0 之前的 ActionScript 编译器版本并不会引发错误。然而， ActionScript 3.0 在此情况下会引发 ReferenceError 异常。	由于访问未定义变量而引发异常表明存在潜在的错误，有助于提高软件质量。但是，如果您不习惯初始化变量，则需要改变一些代码编写习惯来适应这种新的 ActionScript 行为。
SecurityError	如果发生安全违规且访问被拒绝时，则会引发 SecurityError 异常。	以下是一些安全错误示例： • 通过安全沙箱边界进行未经授权的属性访问或方法调用。 • 尝试访问安全沙箱不允许的 URL。 • 尝试与某个端口进行套接字连接，但是必需的套接字策略文件不存在。 • 已尝试使用用户的摄像机或麦克风，但用户拒绝了对该设备的使用。
SyntaxError	如果 ActionScript 代码发生分析错误，则会引发 SyntaxError 异常。	在以下情况下会引发 SyntaxError： • 当 RegExp 类解析无效的正则表达式时， ActionScript 会引发 SyntaxError 异常。 • 当 XMLDocument 类解析无效的 XML 时， ActionScript 会引发 SyntaxError 异常。
TypeError	如果操作数的实际类型与所需类型不同，则会引发 TypeError 异常。	在以下情况下会引发 TypeError： • 无法将函数的实际参数或方法强制为正式参数类型。 • 值已赋给变量，但无法强制为变量的类型。 • is 或 instanceof 运算符右侧的内容不是有效类型。 • 非法使用了 super 关键字。 • 属性查找生成了多个绑定，因此造成该查找不明确。 • 对不兼容对象调用了某个方法。例如，如果将 RegExp 类中的某个方法 “ 转接 ” 到通用对象上，然后调用该方法，会引发 TypeError 异常。
URIError	如果采用与某个全局 URI 处理函数的定义相矛盾的方式使用该函数，则会引发 URIError 异常。	在以下情况下会引发 URIError： 为需要有效 URI 的 Flash Player API 函数 （如 Socket.connect()）指定了无效的 URI。
VerifyError	如果遇到格式不正确或损坏的 SWF 文件，则会引发 VerifyError 异常。	当 SWF 文件加载另一个 SWF 文件时，父 SWF 文件可以捕获由加载的 SWF 文件生成的 VerifyError。

flash.error 包 Error 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

flash.error 包中包含的 Error 类被视为 Flash 运行时 API 的一部分。与前面描述的 Error 类不同， flash.error 包与特定于 Flash 运行时 （例如 Flash Player 和 Adobe AIR）的错误事件进行通信。

类名称	说明	备注
EOFError	如果尝试读取的内容超出可用数据的末尾，则会引发 EOFError 异常。	例如，当调用 IDataInput 接口中的一个读取方法，而数据不足以满足读取请求时，将引发 EOFError。
IllegalOperationError	如果方法未实现或者实现中未涵盖当前用法，则会引发 IllegalOperationError 异常。	以下是非法操作错误异常的示例： • 基类 （如 DisplayObjectContainer）提供的功能比舞台可以支持的功能多。例如，如果尝试在舞台上获取或设置遮罩层 （使用 stage.mask）， Flash 运行时会引发 IllegalOperationError，并显示消息 “Stage 类不实现此属性或方法 ”。 • 子类继承了不需要且不想支持的方法。 • 在没有辅助功能支持的情况下编译 Flash Player 之后，又调用了某些辅助功能方法。 • 从 Flash Player 的运行时版本调用仅创作功能。 • 试图为放在时间轴上的对象设置名称。
IOError	如果发生某种类型的 I/O 异常，则会引发 IOError 异常。	例如，当试图对尚未连接或已断开连接的套接字进行读 / 写操作时，将引发此错误。
MemoryError	如果内存分配请求失败，则会引发 MemoryError 异常。	默认情况下， ActionScript Virtual Machine 2 不会强制限制 ActionScript 程序所分配的内存大小。在桌面系统中，内存分配故障并不常见。当系统无法分配操作所需的内存时，将会看到这样的错误。因此，在桌面系统中，除非分配请求极大，否则此异常很罕见；例如，分配三十亿个字节的请求是不可能的，因为 32 位 Microsoft® Windows® 程序仅可以访问 2 GB 的地址空间。
ScriptTimeoutError	如果达到了 15 秒的脚本超时间隔，则会引发 ScriptTimeoutError 异常。通过捕获 ScriptTimeoutError 异常，可以更加妥善地处理脚本超时。如果没有异常处理函数，则未捕获的异常处理函数将显示一个带有错误消息的对话框。	为防止恶意开发者捕获这种异常并导致无限循环，仅能够捕获特定脚本运行过程中引发的第一个 ScriptTimeoutError 异常。随后的 ScriptTimeoutError 异常无法由您的代码捕获，并且立即转至未捕获的异常处理函数。
StackOverflowError	如果脚本可用堆栈已经用尽，则会引发 StackOverflowError 异常。	StackOverflowError 异常可能指示发生了无限递归。

处理错误示例：CustomErrors 应用程序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

CustomErrors 应用程序展示了构建应用程序时使用自定义错误的一些技巧。这些方法包括：

- 验证 XML 包
- 编写自定义错误
- 引发自定义错误
- 引发错误时通知用户

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/CustomError 文件夹中找到 CustomErrors 应用程序的文件。该应用程序包含以下文件：

文件	说明
CustomErrors.mxml 或 CustomErrors.fla	Flash (FLA) 或 Flex (MXML) 中的主应用程序文件
com/example/programmingas3/errors/ApplicationError.as	一个类，用作 FatalError 类和 WarningError 类的基错误类。
com/example/programmingas3/errors/FatalError.as	用于定义由应用程序引发的 FatalError 错误的类。该类扩展了自定义的 ApplicationError 类。
com/example/programmingas3/errors/Validator.as	一个类，定义了用于验证用户提供的员工 XML 包的单个方法。
com/example/programmingas3/errors/WarningError.as	用于定义由应用程序引发的 WarningError 错误的类。该类扩展了自定义的 ApplicationError 类。

CustomErrors 应用程序概述

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当应用程序加载时，将为 Flex 应用程序调用 initApp() 方法，或者为 Flash Professional 应用程序执行时间轴 （非函数）代码。此代码定义将由 Validator 类验证的示例 XML 包。以下代码运行：

```
employeeXML =
    <employee id="12345">
        <firstName>John</firstName>
        <lastName>Doe</lastName>
        <costCenter>12345</costCenter>
        <costCenter>67890</costCenter>
    </employee>;
}
```

稍后，将在舞台上的 TextArea 组件实例中显示该 XML 包。此步骤允许您在尝试重新验证 XML 包之前对其进行修改。

用户单击 Validate 按钮时，将调用 validateData() 方法。该方法使用 Validator 类中的 validateEmployeeXML() 方法来验证员工 XML 包。以下显示的是 validateData() 方法的代码：

```
function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

首先，使用 TextArea 组件实例 xmlText 的内容创建一个临时的 XML 对象。接下来，将调用自定义 Validator 类 (com.example.programmingas3/errors/Validator.as) 中的 validateEmployeeXML() 方法，并将临时 XML 对象作为参数传递。如果这个 XML 包是有效的， Label 组件实例 status 便会显示一条成功消息，然后应用程序退出。如果 validateEmployeeXML() 方法引发了一个自定义错误 （即发生 FatalError、 WarningError 或一般的 Error），则会执行相应的 catch 语句并调用 showFatalError()、 showWarningError() 或 showGenericError() 方法。这几种方法均会在一个名为 statusText 的文本区域中显示相应的消息，以通知用户发生了特定错误。每个方法还会用具体的消息更新 Label 组件实例 status。

如以下代码所示，如果尝试验证员工 XML 包时发生致命错误，则会在一个 statusText 文本区域中显示错误消息，并且禁用 xmlText TextArea 组件实例和 validateBtn Button 组件实例：

```
function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n";
    var title:String = error.getTitle();
    statusText.text = message + " " + title + "\n\nThis application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
    hideButtons();
}
```

如果发生的是警告错误而不是致命错误，则会在 statusText TextArea 实例中显示错误消息，但不会禁用 xmlText TextField 和 Button 组件实例。showWarningError() 方法会在 statusText 文本区域中显示自定义的错误消息。该消息还要求用户决定是希望继续验证 XML 还是取消脚本。以下节选的内容显示的是 showWarningError() 方法代码：

```
function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this application?";
    showButtons();
    var title:String = error.getTitle();
    statusText.text = message;
}
```

当用户单击 “ 是 ” 或 “ 否 ” 按钮时，将调用 closeHandler() 方法。以下节选的内容显示的是 closeHandler() 方法代码：

```
function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case yesButton:
            showFatalError(new FatalError(9999));
            break;
        case noButton:
            statusText.text = "";
            hideButtons();
            break;
    }
}
```

如果用户通过单击 “ 是 ” 选择取消脚本，会引发 FatalError，导致应用程序终止。

构建自定义验证程序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

自定义的 Validator 类仅包含一个方法 validateEmployeeXML()。validateEmployeeXML() 方法采用一个参数，即 employee，此参数是您要验证的 XML 包。 validateEmployeeXML() 方法的代码如下所示：

```
public static function validateEmployeeXML(employee:XML):void
{
    // checks for the integrity of items in the XML
    if (employee.costCenter.length() < 1)
    {
        throw new FatalError(9000);
    }
    if (employee.costCenter.length() > 1)
    {
        throw new WarningError(9001);
    }
    if (employee.ssn.length() != 1)
    {
        throw new FatalError(9002);
    }
}
```

员工必须属于一个 （且只能属于一个）成本中心才能通过验证。如果员工不属于任何成本中心，该方法将引发 FatalError，该异常将向上冒泡到应用程序主文件中的 validateData() 方法。如果员工属于多个成本中心，则引发 WarningError。该 XML 验证程序最后检查用户是否只定义了一个社会安全号码 （XML 包中的 ssn 节点）。如果具有不止一个 ssn 节点，则引发 FatalError 错误。

您可以向 validateEmployeeXML() 方法添加其他检查，例如，确保 ssn 节点包含有效编号，或者员工至少定义了一个电话号码和电子邮件地址，并且两个值均有效。您还可以对该 XML 进行修改，使每个员工具有唯一的 ID 并指定其管理者的 ID。

定义 ApplicationError 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ApplicationError 类用作 FatalError 类和 WarningError 类的基类。 ApplicationError 类扩展了 Error 类，并且定义了自己的自定义方法和属性，其中包括定义一个错误 ID、严重程度以及包含自定义错误代码和消息的 XML 对象。该类还定义了两个静态常量，它们用于定义每种错误类型的严重程度。

ApplicationError 类的构造函数方法如下所示：

```
public function ApplicationError()
{
    messages =
        <errors>
            <error code="9000">
                <![CDATA[Employee must be assigned to a cost center.]]>
            </error>
            <error code="9001">
                <![CDATA[Employee must be assigned to only one cost center.]]>
            </error>
            <error code="9002">
                <![CDATA[Employee must have one and only one SSN.]]>
            </error>
            <error code="9999">
                <![CDATA[The application has been stopped.]]>
            </error>
        </errors>;
}
```

XML 对象中的每个错误节点都包含一个唯一的数值代码和一条错误消息。使用 E4X 可以很容易地通过错误代码查找到相应的错误消息，如以下 getMessageText() 方法所示：

```
public function getMessageText(id:int):String
{
    var message:XMLList = messages.error.(@code == id);
    return message[0].text();
}
```

getMessageText() 方法仅使用一个整数参数 id 并返回一个字符串。该 id 参数就是要查找的错误的错误代码。例如，传递等于 9001 的 id 值将得到错误消息 “Employee must be assigned to only one cost center” （只能为员工指定一个成本中心）。如果多个错误具有同一错误代码， ActionScript 将只为找到的第一个结果返回错误消息 （所返回 XMLList 对象中的 message[0]）。

该类中的下一个方法 getTitle() 不使用任何参数，并返回一个字符串值，其中包含此特定错误的错误 ID。该值用于帮助您轻松识别在验证 XML 包期间发生的具体错误。以下显示的是 getTitle() 方法的代码：

```
public function getTitle():String
{
    return "Error #" + id;
}
```

ApplicationError 类中的最后一个方法是 toString()。该方法覆盖 Error 类中定义的函数，以便您可以自定义错误消息的显示。该方法将返回一个字符串，用于识别所发生错误的具体编号和消息。

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

定义 FatalError 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

FatalError 类扩展了自定义的 ApplicationError 类并定义三个方法：FatalError 构造函数、 getTitle() 和 toString()。第一个方法 （即 FatalError 构造函数）接受一个整数参数 errorID，并使用 ApplicationError 类中定义的静态常量设置错误的严重性，另外还调用 ApplicationError 类中的 getMessageText() 方法获取特定错误的错误消息。 FatalError 构造函数如下所示：

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

FatalError 类中的下一个方法 getTitle() 覆盖先前在 ApplicationError 类中定义的 getTitle() 方法，并在标题后面追加文字 “-- FATAL” 以通知用户发生了致命错误。 getTitle() 方法如下所示：

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

该类中的最后一个方法 toString() 覆盖 ApplicationError 类中定义的 toString() 方法。 toString() 方法如下所示：

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

定义 WarningError 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

WarningError 类扩展了 ApplicationError 类，它与 FatalError 类几乎完全相同，只是字符串稍微有些不同，另外它还将错误严重程度设置为 ApplicationError.WARNING 而不是 ApplicationError.FATAL，如以下代码所示：

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

第 5 章：使用正则表达式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

正则表达式描述用于查找和处理字符串中的匹配文本的模式。正则表达式类似于字符串，但是可以包含特殊代码以描述模式和重复。例如，下面的正则表达式与以字符 A 开头并且后跟一个或多个连续数字的字符串匹配：

```
/A\d+/
```

以下主题描述构建正则表达式所用的基本语法。但是实际上，正则表达式可能非常复杂且具有许多细微差别。您可以从网上或者书店中找到有关正则表达式的详细资料。切记，不同的编程环境实现正则表达式的方式也不同。 ActionScript 3.0 按照 ECMAScript 第 3 版语言规范 (ECMA-262) 中的定义实现正则表达式。

更多帮助主题

RegExp

正则表达式基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

正则表达式描述字符模式。通常，正则表达式用于验证文本值是否符合特定模式 （例如，验证用户输入的电话号码位数是否正确），或者替换与特定模式匹配的部分文本值。

正则表达式可能非常简单。例如，假设您要确认特定字符串与 “ABC” 是否匹配，或者要使用某些其他文本替换字符串中出现的每个 “ABC”。在这种情况下，您可以使用以下正则表达式，它定义了依次包含字母 A、 B 和 C 的模式：

```
/ABC/
```

请注意，正则表达式文本是使用正斜杠 (/) 字符界定的。

正则表达式模式也可能非常复杂，有时候表面上看起来晦涩难懂，例如，以下与有效电子邮件地址匹配的表达式：

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

通常，您使用正则表达式在字符串中搜索模式以及替换字符。在这些情况下，您将创建一个正则表达式对象，并将其作为几个 String 类方法之一的参数。下列 String 类方法将正则表达式作为参数：match()、 replace()、 search() 和 split()。有关这些方法的详细信息，请参阅第 15 页的 “ 在字符串中查找模式并替换子字符串 ”。

RegExp 类包含以下方法：test() 和 exec()。有关详细信息，请参阅第 75 页的 “ 对字符串使用正则表达式的方法 ”。

重要概念和术语

以下参考列表中包含与此功能相关的重要术语：

转义符 此字符指示应将后面的字符视为元字符，而不是字面字符。在正则表达式语法中，反斜杠字符 (\) 就是转义字符，因此反斜杠后跟另一个字符是一个特殊代码，而不仅仅是字符本身。

标志 指定有关应如何使用正则表达式模式的一些选项 （如是否区分大写和小写字符）的字符。

元字符 在正则表达式模式中具有特殊含义的字符，与从字面意义上在模式中表示该字符相对。

限定符 一个或多个字符，用于指示应重复模式的某一部分多少次。例如，使用数量表示符来指定美国邮政编码应包含 5 个或 9 个数字。

正则表达式 用于定义字符模式的程序语句，该字符模式可用来确认其他字符串是否与模式匹配或可用来替换部分字符串。

正则表达式语法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

本节介绍了 ActionScript 正则表达式语法的全部元素。正如您所看到的一样，正则表达式可能非常复杂且具有许多细微差别。您可以从网上或者书店中找到有关正则表达式的详细资料。切记，不同的编程环境实现正则表达式的方式也不同。ActionScript 3.0 按照 ECMAScript 第 3 版语言规范 (ECMA-262) 中的定义实现正则表达式。

通常，您要使用的正则表达式是与比较复杂的模式匹配，而不是与简单的字符串匹配。例如，下面的正则表达式定义了由字母 A、 B 和 C 依次排列且后跟数字的模式：

```
/ABC\d/
```

\d 代码表示 " 任意数字 "。反斜杠 (\) 字符称为转义字符，它与后面的字符 （在本例中为字母 d）配合使用，在正则表达式中具有特殊含义。

下面的正则表达式定义了由字母 ABC 后跟任意数目的数字组成的模式 （注意星号）：

```
/ABC\d*/
```

星号字符 (*) 是 " 元字符 "。元字符是在正则表达式中具有特殊含义的字符。星号是一种称为 " 数量表示符 " 的特定类型的元字符，用于定义某个字符或一组字符重复的次数。有关详细信息，请参阅第 68 页的 " 数量表示符 "。

除了它的模式外，正则表达式还可以包含标志，用于指定正则表达式的匹配方式。例如，下面的正则表达式使用 i 标志指定正则表达式在匹配字符串中忽略大小写：

```
/ABC\d*/i
```

有关详细信息，请参阅第 72 页的 " 标志和属性 "。

您可以通过以下 String 类方法使用正则表达式：match()、 replace() 和 search()。有关这些方法的详细信息，请参阅第 15 页的 " 在字符串中查找模式并替换子字符串 "。

创建正则表达式实例

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有两种方法可以创建正则表达式实例。一种方法是使用正斜杠字符 (/) 来界定正则表达式，另一种是使用 new 构造函数。例如，以下两个正则表达式是等效的：

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

正斜杠界定正则表达式的方式与用引号界定字符串文本的方式相同。正斜杠内的正则表达式部分定义 " 模式 "。正则表达式还可以在后一个界定斜杠后包含 " 标志 "。这些标志也看作是正则表达式的一部分，但是它们独立于模式。

使用 new 构造函数时，使用两个字符串来定义正则表达式。第一个字符串定义模式，第二个字符串定义标志，如下例所示：

```
var pattern2:RegExp = new RegExp("bob", "i");
```

如果在使用正斜杠界定符定义的正则表达式中包含正斜杠，则必须在正斜杠前面加上反斜杠 (\) 转义字符。例如，下面的正则表达式与模式 1/2 匹配：

```
var pattern:RegExp = /1\/2/;
```

若要在使用 new 构造函数定义的正则表达式中包含引号，您必须在引号前面加上反斜杠 (\) 转义字符 （就像定义任何 String 文本一样）。例如，下面的正则表达式与模式 eat at "joe's" 匹配：

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

请勿在使用正斜杠界定符定义的正则表达式中对引号使用反斜杠转义字符。同样地，不要在使用 new 构造函数定义的正则表达式中对正斜杠使用转义字符。下列正则表达式是等效的，它们都定义了模式 1/2 "joe's"：

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

此外，在使用 new 构造函数定义的正则表达式中，若要使用以反斜杠 (\) 字符开头的元序列 （例如，\d，用于匹配任何数字），请键入两个反斜杠字符：

```
var pattern:RegExp = new RegExp("\\d+", ""); // matches one or more digits
```

在本实例中您必须键入两个反斜杠字符，因为 RegExp() 构造函数方法的第一个参数是字符串，而在字符串文本中必须键入两个反斜杠字符才能将其识别为单个反斜杠字符。

后面几节将介绍定义正则表达式模式的语法。

有关标志的详细信息，请参阅第 72 页的 “ 标志和属性 ”。

字符、元字符和元序列

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

最简单的正则表达式是与字符序列匹配的表达式，如以下示例中所示：

```
var pattern:RegExp = /hello/;
```

但是，下列字符 （称为元字符）在正则表达式中具有特殊含义：

```
^ $ \ . * + ? ( ) [ ] { } |
```

例如，下面的正则表达式所匹配的是字母 A 后跟字母 B 的零个或多个实例 （星号元字符指示重复）再跟字母 C：

```
/AB*C/
```

在正则表达式模式中包含元字符时若要使其不具有特殊含义，您必须使用反斜杠 (\) 转义字符。例如，下面的正则表达式与顺序依次为字母 A、字母 B、星号和字母 C 的模式匹配：

```
var pattern:RegExp = /AB\*C/;
```

“ 元序列 ” 与元字符类似，在正则表达式中具有特殊含义。元序列由多个字符组成。以下几节提供了有关使用元字符和元序列的详细信息。

关于元字符

下表总结了可以在正则表达式中使用的元字符：

元字符	说明
^ （尖号）	匹配字符串的开头。设置 m (multiline) 标志后，尖号还匹配行的开头 （请参阅第 72 页的 “ 标志和属性 ”）。请注意，尖号用在字符类的开头时指示符号反转而非字符串的开头。有关详细信息，请参阅第 67 页的 “ 字符类 ”。
$ （美元符号）	匹配字符串的结尾。设置 m (multiline) 标志后，$ 还匹配换行 (\n) 字符前面的位置。有关详细信息，请参阅第 72 页的 “ 标志和属性 ”。
\ (反斜杠)	对特殊字符的特殊元字符含义进行转义。 此外，如果要在正则表达式文本中使用正斜杠字符，也要使用反斜杠字符，例如，/1\/2/ 匹配字符 1 后跟正斜杠字符和字符 2。
. （点）	匹配任意单个字符。 只有设置 s (dotall) 标志时，点才匹配换行字符 (\n)。有关详细信息，请参阅第 72 页的 “ 标志和属性 ”。

元字符	说明
* （星号）	匹配前面重复零次或多次的项目。 有关详细信息，请参阅第 68 页的 “ 数量表示符 ”。
+ （加号）	匹配前面重复一次或多次的项目。 有关详细信息，请参阅第 68 页的 “ 数量表示符 ”。
? （问号）	匹配前面重复零次或一次的项目。 有关详细信息，请参阅第 68 页的 “ 数量表示符 ”。
(和)	在正则表达式中定义组。以下情况下使用组： • 限制逻辑 “ 或 ” 字符 | 的范围：/(a|b|c)d/ • 定义数量表示符的范围：/(walla.){1,2}/ • 用在逆向引用中。例如，下面的正则表达式中的 \1 匹配模式的第一个括号组中的匹配内容： • /(\w*) is repeated: \1/ 有关详细信息，请参阅第 70 页的 “ 组 ”。
[和]	定义字符类，字符类定义单个字符可能的匹配： /[aeiou]/ 匹配所指定字符中的任意一个。 在字符类中，使用连字符 (-) 指定字符的范围： /[A-Z0-9]/ 匹配从 A 到 Z 的大写字母或 0 到 9 的数字。 在字符类中，插入反斜杠对] 和 - 字符： /[+\-]\d+/ 匹配一个或多个数字前面的 + 或 -。 在字符类中，以下字符 （通常为元字符）被看作一般字符 （非元字符），不需要反斜杠： /[$]/£ 匹配 $ 或 £。 有关详细信息，请参阅第 67 页的 “ 字符类 ”。
| （竖线）	用于逻辑 “ 或 ” 操作，匹配左侧或右侧的部分： /abc|xyz/ 匹配 abc 或 xyz。

关于元序列

元序列是在正则表达式模式中具有特殊含义的字符序列。下表说明了这些元序列：

元序列	说明
{*n*} {*n*,} 和 {*n*,*n*}	指定前一项目的数值数量或数量范围： /A{27}/ 匹配重复 27 次的字符 A。 /A{3,}/ 匹配重复 3 次或更多次的字符 A。 /A{3,5}/ 匹配重复 3 到 5 次的字符 A。 有关详细信息，请参阅第 68 页的 “ 数量表示符 ”。
\b	匹配单词字符和非单词字符之间的位置。如果字符串中的第一个或最后一个字符是单词字符，则也匹配字符串的开头或结尾。
\B	匹配两个单词字符之间的位置。也匹配两个非单词字符之间的位置。

元序列	说明
\d	匹配十进制数字。
\D	匹配除数字以外的任何字符。
\f	匹配换页符。
\n	匹配换行符。
\r	匹配回车符。
\s	匹配任何空白字符 （空格、制表符、换行符或回车符）。
\S	匹配除空白字符以外的任何字符。
\t	匹配制表符。
\u*nnnn*	匹配字符代码由十六进制数字 nnnn 指定的 Unicode 字符。例如， \u263a 是一个笑脸字符。
\v	匹配垂直换页符。
\w	匹配单词字符 （A-Z、 a-z、 0-9 或 _）。请注意， \w 不匹配非英文字符，如 é、 ñ 或 ç。
\W	匹配除单词字符以外的任何字符。
\\x*nn*	匹配具有指定 ASCII 值 （由十六进制数字 nn 定义）的字符。

字符类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用字符类指定字符列表以匹配正则表达式中的一个位置。使用中括号 （[和]）定义字符类。例如，下面的正则表达式定义了匹配 bag、 beg、 big、 bog 或 bug 的字符类：

```
/b[aeiou]g/
```

字符类中的转义序列

通常在正则表达式中具有特殊含义的大多数元字符和元序列在字符类中 “ 不具有 ” 那些特殊含义。例如，在正则表达式中星号用于表示重复，但是出现在字符类中时则不具有此含义。下列字符类匹配星号本身以及列出的任何其他字符：

```
/[abc*123]/
```

但是，下表中列出的三个字符功能与元字符相同，在字符类中具有特殊含义：

元字符	在字符类中的含义
]	定义字符类的结尾。
-	定义字符范围 （请参阅后面的 “ 字符类中字符的范围 ” 一节）。
\	定义元序列并撤消元字符的特殊含义。

对于要识别为字面字符 （无特殊元字符含义）的任何字符，必须在该字符前面加反斜杠转义字符。例如，下面的正则表达式包含匹配四个符号 （$、 \、] 或 -）中任意一个符号的字符类。

```
/[$\\\]\-]/
```

除能够保持特殊含义的元字符外，下列元序列在字符类中也具有元序列功能：

元序列	在字符类中的含义
\n	匹配换行符。
\r	匹配回车符。
\t	匹配制表符。
\u*nnnn*	匹配具有指定 Unicode 代码点值 （由十六进制数字 nnnn 定义）的字符。
\\x*nn*	匹配具有指定 ASCII 值 （由十六进制数字 nn 定义）的字符。

其他正则表达式元序列和元字符在字符类中看作普通字符。

字符类中字符的范围

使用连字符指定字符的范围，例如 A-Z、 a-z 或 0-9。这些字符必须在字符类中构成有效的范围。例如，下面的字符类匹配 a-z 范围内的任何一个字符或任何数字：

```
/[a-z0-9]/
```

您还可以使用 \\xnn ASCII 字符代码通过 ASCII 值指定范围。例如，下面的字符类匹配扩展 ASCII 字符集中的任意字符（如 é 和 ê）：

```
\\x
```

反转的字符类

如果在字符类的开头使用尖号 (^) 字符，则将反转该集合的意义，即未列出的任何字符都认为匹配。下面的字符类匹配除小写字母 (a-z) 或数字以外的任何字符：

```
/[^a-z0-9]/
```

必须在字符类的开头键入尖号 (^) 字符以指示反转。否则，您只是将尖号字符添加到字符类的字符中。例如，下面的字符类匹配许多符号字符中的任意一个，其中包括尖号：

```
/[!.,#+*%$&^]/
```

数量表示符

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用数量表示符指定字符或序列在模式中的重复次数，如下所示：

数量表示符元字符	说明
* （星号）	匹配前面重复零次或多次的项目。
+ （加号）	匹配前面重复一次或多次的项目。
? （问号）	匹配前面重复零次或一次的项目。
{*n*} {*n*,} 和 {*n*,*n*}	指定前一项目的数值数量或数量范围： /A{27}/ 匹配重复 27 次的字符 A。 /A{3,}/ 匹配重复 3 次或更多次的字符 A。 /A{3,5}/ 匹配重复 3 到 5 次的字符 A。

您可以将数量表示符应用到单个字符、字符类或组：

- /a+/ 匹配重复一次或多次的字符 a。

- /\d+/ 匹配一个或多个数字。
- /[abc]+/ 匹配重复的一个或多个字符，这些字符可能是 a、 b 或 c 中的某个。
- /(very,)*/ 匹配重复零次或多次的后跟逗号和空格的单词 very。

您可以在应用数量表示符的括号组内使用数量表示符。例如，下面的数量表示符匹配诸如 word 和 word-word-word 的字符串：

```
/\w+(-\w+)*/
```

默认情况下，正则表达式执行所谓的 " 无限匹配 "。正则表达式中的任何子模式 （如 .*）都会尝试在字符串中匹配尽可能多的字符，然后再执行正则表达式的下一部分。例如，使用以下正则表达式和字符串：

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

正则表达式匹配整个字符串：

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

但是，假如您只想匹配一个 ... 组。则可以通过以下操作实现：

```
<p>Paragraph 1</p>
```

在所有数量表示符后添加问号 (?) 以将其更改为所谓的 " 惰性数量表示符 "。例如，下面的正则表达式使用惰性数量表示符 *? 匹配 后跟数量最少 （惰性）的字符，再跟 的模式：

```
/<p>.*?<\/p>/
```

有关数量表示符，请牢记以下几点：

- 数量表示符 {0} 和 {0,0} 不会从匹配中排除项目。
- 不要结合使用多个数量表示符，例如 /abc+*/ 中。
- 在除非设置 s (dotall) 标志，否则不会跨越多行，即使后跟 * 数量表示符。例如，请看以下代码：

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*<\/p>/;
trace(str.match(re)); // null;

re = /<p>.*<\/p>/s;
trace(str.match(re));
    // output: <p>Test
    //                  Multiline</p>
```

有关详细信息，请参阅第 72 页的 " 标志和属性 "。

逻辑 " 或 "

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在正则表达式中使用 | （竖线）字符可使正则表达式引擎考虑其他匹配。例如，下面的正则表达式匹配单词 cat、dog、pig 和 rat 中的任意一个：

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

您可以使用括号定义组以限制逻辑 " 或 " 字符 | 的范围。下面的正则表达式匹配 cat 后跟 nap 或 nip：

```
var pattern:RegExp = /cat(nap|nip)/;
```

有关详细信息，请参阅第 70 页的 " 组 "。

下面两个正则表达式是等效的，一个使用 | 逻辑 " 或 " 字符，另一个使用字符类 （由 [和] 定义）：

```
/1|3|5|7|9/
/[13579]/
```

有关详细信息，请参阅第 67 页的 “ 字符类 ”。

组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以使用括号在正则表达式中指定组，如下所示：

```
/class-(\d*)/
```

组是模式的子部分。您可以使用组实现以下操作：

- 将数量表示符应用到多个字符。
- 界定要应用逻辑 “ 或 ” （通过使用 | 字符）的子模式。
- 捕获正则表达式中的子字符串匹配（例如，在正则表达式中使用 \1 以匹配先前匹配的组，或类似地在 String 类的 replace() 方法中使用 $1）。

下面几节将介绍有关这些组用法的详细信息。

使用带数量表示符的组

如果不使用组，数量表示符将应用到它前面的字符或字符类，如下所示：

```
var pattern:RegExp = /ab*/ ;
// matches the character a followed by
// zero or more occurrences of the character b

pattern = /a\d+/;
// matches the character a followed by
// one or more digits

pattern = /a[123]{1,3}/;
// matches the character a followed by
// one to three occurrences of either 1, 2, or 3
```

然而，您可以使用组将数量表示符应用到多个字符或字符类：

```
var pattern:RegExp = /(ab)*/;
// matches zero or more occurrences of the character a
// followed by the character b, such as ababab

pattern = /(a\d)+/;
// matches one or more occurrences of the character a followed by
// a digit, such as a1a5a8a3

pattern = /(spam ){1,3}/;
// matches 1 to 3 occurrences of the word spam followed by a space
```

有关数量表示符的详细信息，请参阅第 68 页的 “ 数量表示符 ”。

使用带逻辑 “ 或 ” 字符 (|) 的组

可以使用组来定义一组要应用逻辑 “ 或 ” 字符 (|) 的字符，如下所示：

```
var pattern:RegExp = /cat|dog/;
// matches cat or dog

pattern = /ca(t|d)og/;
// matches catog or cadog
```

使用组捕获子字符串匹配

如果您在模式中定义标准括号组，则之后可以在正则表达式中引用它。这称为“逆向引用”，并且此类型的组称为“捕获组”。例如，在下面的正则表达式中，序列 \1 匹配在捕获括号组中匹配的任意子字符串：

```
var pattern:RegExp = /(\d+)-by-\1/;
// matches the following: 48-by-48
```

您可以通过键入 \1、 \2、 ...、 \99 在正则表达式中最多指定 99 个此类逆向引用。

类似地，在 String 类的 replace() 方法中，可以使用 $1-$99 在替换字符串中插入捕获的组子字符串匹配：

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
   // output: Bob, hello.
```

此外，如果使用捕获组， RegExp 类的 exec() 方法和 String 类的 match() 方法将返回与捕获组匹配的子字符串：

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
   // bob@example.com,bob,example,com
```

使用非捕获组和向前查找组

非捕获组是只用于分组的组，它不会被“收集”，也不会匹配有限的逆向引用。可以使用 (?: 和) 来定义非捕获组，如下所示：

```
var pattern = /(?:com|org|net);
```

例如，注意在捕获组和非捕获组中加入 (com|org) 的区别 （exec() 方法在完全匹配后列出捕获组）：

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
// bob@example.com,bob,example,com

//noncapturing:
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
   // bob@example.com,bob,example
```

一类特殊的非捕获组是“向前查找组”，它包括两种类型：“正向前查找组”和“负向前查找组”。

使用 (?= 和) 定义正向前查找组，它指定组中的子模式位置必须匹配。但是，匹配正向前查找组的字符串部分可能匹配正则表达式中的剩余模式。例如，由于 (?=e) 在下面的代码中是正向前查找组，它匹配的字符 e 可以被正则表达式的后续部分匹配，在本例中为捕获组 \w*)：

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

使用 (?! 和) 定义负向前查找组，它指定该组中的子模式位置不得匹配。例如：

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

使用命名组

命名组是正则表达式中给定命名标识符的一类组。使用 (?P<name> 和) 可定义命名组。例如，下面的正则表达式包含标识符命名为 digits 的命名组：

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

如果使用 exec() 方法，将添加一个匹配的命名组作为 result 数组属性：

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

这里还有一个例子，它使用两个命名组，标识符分别为 name 和 dom：

```
var emailPattern:RegExp =
    /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

注：命名组不属于 ECMAScript 语言规范。它们是 ActionScript 3.0 中的新增功能。

标志和属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

下表列出了可以为正则表达式设置的五种标志。每种标志都可以作为正则表达式对象属性进行访问。

标志	属性	说明
g	global	匹配多个匹配。
i	ignoreCase	不区分大小写的匹配。应用于 A-Z 和 a-z 字符，但不能应用于扩展字符，如 É 和 é。
m	multiline	设置此标志后， $ 和 ^ 可以分别匹配行的开头和结尾。
s	dotall	设置此标志后， . （点）可以匹配换行符 (\n)。
x	extended	允许扩展的正则表达式。您可以在正则表达式中键入空格，它将作为模式的一部分被忽略。这可使您更加清晰可读地键入正则表达式代码。

请注意这些属性都是只读属性。您在设置正则表达式变量时，可以设置标志 （g、 i、 m、 s 和 x），如下所示：

```
var re:RegExp = /abc/gimsx;
```

但是，您无法直接设置命名属性。例如，下列代码将导致错误：

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

默认情况下，除非您在正则表达式声明中指定这些标志，否则不会设置，并且相应的属性也会设置为 false。

另外，还有其他两种正则表达式属性：

- lastIndex 属性指定字符串中的索引位置以用于下次调用正则表达式的 exec() 或 test() 方法。
- source 属性指定定义正则表达式的模式部分的字符串。

g (global) 标志

如果不包含 g (global) 标志，正则表达式匹配将不超过一个。例如，如果正则表达式中不包含 g 标志， String.match() 方法只返回一个匹配的子字符串：

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

如果设置 g 标志， Sting.match() 方法将返回多个匹配，如下所示：

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// The same pattern, but this time the g flag IS set.
trace(str.match(pattern)); // output: she,shells,shore
```

i (ignoreCase) 标志

默认情况下，正则表达式匹配区分大小写。如果设置 i (ignoreCase) 标志，将忽略区分大小写。例如，正则表达式中的小写字母 s 不会匹配大写字母 S （字符串中的第一个字符）：

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

但是如果设置 i 标志，正则表达式将匹配大写字母 S：

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

i 标志仅忽略 A-Z 和 a-z 字符的大小写，而不忽略扩展字符的大小写，如 É 和 é。

m (multiline) 标志

如果没有设置 m (multiline) 标志，^ 将匹配字符串的开头，而 $ 匹配字符串的结尾。如果设置 m 标志，这些字符将分别匹配行的开头和结尾。请考虑使用下列包含换行符的字符串：

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

即使在正则表达式中设置 g (global) 标志， match() 方法也会只匹配一个子字符串，因为对于 ^ （字符串开头）只有一个匹配。输出结果为：

```
Test
```

下面是设置了 m 标志的同一段代码：

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

这次输出结果同时包含两行开头的单词：

```
Test,Multiline
```

请注意，只有 \n 字符表示行的结束。下列字符不表示行的结束：

- 回车 (\r) 字符
- Unicode 行分隔符 (\u2028) 字符
- Unicode 段分隔符 (\u2029) 字符

s (dotall) 标志

如果没有设置 s （dotall 或 “dot all”）标志，则正则表达式中的点 (.) 将不匹配换行符 (\n)。因此，下面的示例没有匹配：

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

但是，如果设置了 s 标志，点就匹配换行符：

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

在本例中，匹配内容是 <p> 标签内的整个子字符串，其中包括换行符：

```
<p>Test
Multiline</p>
```

x (extended) 标志

正则表达式有时很难阅读，特别是当其包含很多元字符和元序列时。例如：

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

在正则表达式中使用 x (extended) 标志时，则会忽略在模式中键入的所有空格。例如，下面的正则表达式同前面的示例相同：

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

如果设置了 x 标志，而且希望匹配空格字符，则应在空格前加上反斜杠。例如，以下两个正则表达式是等效的：

```
/foo bar/
/foo \ bar/x
```

lastIndex 属性

lastIndex 属性在字符串中指定开始进行下一次搜索的索引位置。对于将 g 标志设置为 true 的正则表达式，此属性会影响对该表达式调用的 exec() 和 test() 方法。例如，请看以下代码：

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
    trace(pattern.lastIndex);
    result = pattern.exec(str);
}
```

默认情况下， lastIndex 属性设置为 0 （从字符串的开头开始搜索）。每次匹配完成后，都会设为该匹配后的索引位置。因此，前面代码的输出如下所示：

```
0
5
11
18
25
36
44
```

如果将 global 标志设置为 false，则 exec() 和 test() 方法不会使用或设置 lastIndex 属性。

String 类的 match()、 replace() 和 search() 方法都从字符串的开头进行搜索，而不考虑调用该方法时所使用的正则表达式中的 lastIndex 属性设置。（但是， match() 方法不会将 lastIndex 设为 0。）

可以设置 lastIndex 属性来调整字符串中的起始位置以对正则表达式进行匹配。

source 属性

source 属性指定用于定义正则表达式的模式部分的字符串。例如：

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

对字符串使用正则表达式的方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

RegExp 类包含两个方法：exec() 和 test()。

除 RegExp 类的 exec() 和 test() 方法外， String 类还包含以下方法，使您可以在字符串中匹配正则表达式：match()、replace()、 search() 和 splice()。

test() 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

RegExp 类的 test() 方法只检查提供的字符串是否包含正则表达式的匹配内容，如下面的示例所示：

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

exec() 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

RegExp 类的 exec() 方法检查提供的字符串是否有正则表达式的匹配，并返回具有如下内容的数组：

- 匹配的子字符串
- 同正则表达式中的任意括号组匹配的子字符串

该数组还包含 index 属性，此属性指明子字符串匹配起始的索引位置。

例如，请看以下代码：

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; //U.S phone number
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7-415-555-1212
```

在正则表达式设置了 g (global) 标志时，多次使用 exec() 方法可以匹配多个子字符串：

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
//output:
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

使用 RegExp 参数的 String 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

下列 String 类方法将正则表达式作为参数：match()、replace()、search() 和 split()。有关这些方法的详细信息，请参阅第 15 页的 " 在字符串中查找模式并替换子字符串 "。

正则表达式示例：Wiki 解析程序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

这个简单的 Wiki 文本转换示例说明了正则表达式的一些用途：

- 将与源 Wiki 模式匹配的文本行转换为相应的 HTML 输出字符串。
- 使用正则表达式将 URL 模式转换为 HTML <a> 超链接标签。
- 使用正则表达式将美元符号字符串 （如 "$9.95"）转换为欧元符号字符串 （如 "8.24 €"）。

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。 WikiEditor 应用程序文件位于文件夹 Samples/WikiEditor 中。该应用程序包含以下文件：

文件	说明
WikiEditor.mxml 或 WikiEditor.fla	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/regExpExamples/WikiParser.as	该类包含使用正则表达式将 Wiki 输入文本模式转换为等效 HTML 输出的方法。
com/example/programmingas3/regExpExamples/URLParser.as	该类包含使用正则表达式将 URL 字符串转换为 HTML <a> 超链接标签的方法。
com/example/programmingas3/regExpExamples/CurrencyConverter.as	该类包含使用正则表达式将美元符号字符串转换为欧元符号字符串的方法。

定义 WikiParser 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

WikiParser 类包含将 Wiki 输入文本转换为等效 HTML 输出的方法。它虽然不是功能非常强大的 Wiki 转换应用程序，但是说明了正则表达式在模式匹配和字符串转换方面的一些很好的用法。

构造函数和 setWikiData() 方法一起，可简单地初始化 Wiki 输入文本范例字符串，如下所示：

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

当用户单击范例应用程序中的 "Test" 按钮时，应用程序将调用 WikiParser 对象的 parseWikiString() 方法。此方法可调用许多其他方法，这些方法再组合输出的 HTML 字符串。

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

所调用的每个方法（parseBold()、parseItalic()、linesToParagraphs() 和 parseBullets()）都会使用字符串的 replace() 方法替换由正则表达式定义的匹配模式，以便将 Wiki 输入文本转换为 HTML 格式的文本。

转换粗体和斜体模式。

parseBold() 方法会查找 Wiki 粗体文本模式（如 '''foo'''）并将其转换为等效的 HTML 格式（如 foo），如下所示：

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

请注意，正则表达式的 (.?*) 部分匹配两个定义 ''' 模式之间任意数目的字符 (*)。? 数量表示符使匹配不再是无限制的，因此对于字符串 '''aaa''' bbb '''ccc'''，第一个匹配的字符串是 '''aaa''' 而不是以 ''' 模式开头和结尾的整个字符串。

正则表达式中的括号定义捕获组， replace() 方法通过在替换字符串中使用 $1 代码引用此组。正则表达式中的 g (global) 标志确保 replace() 方法替换字符串中的所有匹配（不仅仅是第一个）。

parseItalic() 方法的原理与 parseBold() 方法类似，只是前者检查作为斜体文本分隔符的两个省略号 (''）（而不是三个）：

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

转换项目符号模式

如下面的示例所示， parseBullet() 方法会查找 Wiki 项目符号行模式（如 * foo）并将其转换为等效的 HTML 格式（如 foo）：

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

正则表达式开头的 ^ 符号匹配行的开头。正则表达式中的 m (multiline) 标志使正则表达式用 ^ 符号来匹配行的开头，而不是简单地匹配字符串的开头。

* 模式匹配星号字符（反斜杠用于表示星号本身，而不是 * 数量表示符）。

正则表达式中的括号定义捕获组， replace() 方法通过在替换字符串中使用 $1 代码引用此组。正则表达式中的 g (global) 标志确保 replace() 方法替换字符串中的所有匹配（不仅仅是第一个）。

转换段落 **Wiki** 模式

linesToParagraphs() 方法将每行中输入的 Wiki 字符串转换为 HTML <p> 段落标签。该方法中的这些行从输入的 Wiki 字符串中去除空行：

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

正则表达式中的 ^ 和 $ 符号分别匹配行的开头和结尾。正则表达式中的 m (multiline) 标志使正则表达式用 ^ 符号来匹配行的开头，而不是简单地匹配字符串的开头。

replace() 方法使用空字符串 ("") 替换所有匹配子字符串 （空行）。正则表达式中的 g (global) 标志确保 replace() 方法替换字符串中的所有匹配 （不仅仅是第一个）。

将 URL 转换为 HTML <a> 标签

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当用户单击范例应用程序中的 “Test” 按钮时，如果用户选中了 urlToATag 复选框，应用程序将调用 URLParser.urlToATag() 静态方法以将 URL 字符串从输入的 Wiki 字符串转换为 HTML <a> 标签。

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart, "ig");
var result:String = input.replace(urlPattern, "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

RegExp() 构造函数用于将各组成部分组合成正则表达式 (urlPattern)。这些组成部分是定义正则表达式模式部分的各个字符串。

由 protocol 字符串定义的正则表达式模式的第一部分定义了 URL 协议: http:// 或 ftp://。括号定义了由 ? 符号指示的非捕获组。这意味着括号只是用来定义用于 | 逻辑 “ 或 ” 模式的组，该组不会匹配 replace() 方法的替换字符串中的逆向引用代码 （$1、 $2、 $3）。

正则表达式的其他组成部分都会使用捕获组 （由模式中的括号指示），然后用在 replace() 方法的替换字符串中的逆向引用代码 （$1、 $2、 $3）中。

由 urlPart 字符串定义的模式部分至少匹配下列字符中的一个: a-z、0-9、_ 或 -。+ 数量表示符指示至少一个字符匹配。\。指示一个必需的点 (.) 字符。其余部分匹配至少包含以下字符中的一个的字符串: a-z、 0-9、 _ 或 -。

由 optionalUrlPart 字符串定义的模式部分匹配零个或多个以下字符: 点 (.) 字符后跟任意数目的字母数字字符 （包括 _ 和 -）。* 数量指示符表明零个或多个字符匹配。

调用 replace() 方法可应用正则表达式，并且使用逆向引用来组合替换 HTML 字符串。

然后 urlToATag() 方法会调用 emailToATag() 方法，后者使用类似的技术用 HTML <a> 超链接字符串替换电子邮件模式。在本范例文件中用于匹配 HTTP、 FTP 和电子邮件 URL 的正则表达式是相当简单的，只是起到示范作用，还有更复杂的正则表达式更准确地匹配这类 URL。

将美元符号字符串转换为欧元符号字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当用户单击范例应用程序中的 “Test” 按钮时，如果用户选中了 dollarToEuro 复选框，应用程序将调用 CurrencyConverter.usdToEuro() 静态方法以将美元符号字符串 （如 "$9.95"）转换为欧元符号字符串 （如 "8.24 €"），如下所示:

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

第一行定义的简单模式匹配美元符号字符串。请注意， $ 字符前面加反斜杠 (\) 转义字符。

replace() 方法将正则表达式用作模式匹配参数，并调用 usdStrToEuroStr() 函数以确定替换字符串 （欧元值）。

如果将函数名称用作 replace() 方法的第二个参数时，则会将以下内容作为参数传递给被调用的函数:

- 字符串的匹配部分。

- 任何捕获的括号组匹配。按这种方式传递的参数数目因捕获的括号组匹配的数目而异。您可以通过检查函数代码中的 arguments.length - 3 来确定捕获的括号组匹配的数目。
- 字符串中匹配开始的索引位置。
- 完整的字符串。

usdStrToEuroStr() 方法将美元符号字符串模式转换为欧元符号字符串，如下所示：

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

请注意，args[1] 表示由 usdPrice 正则表达式匹配的捕获括号组。这是美元符号字符串的数字部分，也就是没有 $ 符号的美元数目。该方法应用汇率转换并返回生成的字符串 （带有尾随符号 €，而不是前导符号 $）。

第 6 章：使用 XML

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 包含一组基于 ECMAScript for XML (E4X) 规范 （ECMA-357 第 2 版）的类。这些类包含用于使用 XML 数据的强大且易用的功能。与以前的编程技术相比，使用 E4X 可以更快地用 XML 数据开发代码。此外，开发的代码也更容易阅读。

更多帮助主题

XML 类

ECMA - 357 规范

XML 基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML 是一种表示结构化信息的标准方法，以使计算机能够方便地使用此类信息，并且人们可以非常方便地编写和理解这些信息。 XML 是 eXtensible Markup Language （可扩展标记语言）的缩写。 www.w3.org/XML/ 上提供了 XML 标准。

XML 提供了一种简便的标准方法对数据进行分类，以使其更易于读取、访问以及处理。 XML 使用类似于 HTML 的树结构和标签结构。以下是一个简单的 XML 数据示例：

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

XML 数据也可能会比较复杂，其中包含嵌套在其他标签中的标签以及属性和其他结构组件。以下是一个比较复杂的 XML 数据示例：

```
<album>
    <title>Questions, unanswered</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <tracks>
        <song tracknumber="1" length="4:05">
            <title>What do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:31</lastplayed>
        </song>
        <song tracknumber="2" length="3:45">
            <title>Who do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:35</lastplayed>
        </song>
        <song tracknumber="3" length="5:14">
            <title>When do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:39</lastplayed>
        </song>
        <song tracknumber="4" length="4:19">
            <title>Do you know?</title>
            <artist>Steve and the flubberblubs</artist>
            <lastplayed>2006-10-17-08:44</lastplayed>
        </song>
    </tracks>
</album>
```

请注意，此 XML 文档中包含其他完整 XML 结构 （如 song 标签及其子标签）。此文档还说明了其他 XML 结构，如属性（song 标签中的 tracknumber 和 length）以及包含其他标签而不是包含数据的标签 （如 tracks 标签）。

XML 快速入门

如果您没有或几乎没有 XML 方面的经验，您可以阅读下面对 XML 数据的最常见特性的简要说明。 XML 数据是以纯文本格式编写的，并使用特定语法将信息组织为结构化格式。通常，将一组 XML 数据称为 “XML 文档 ”。在 XML 格式中，通过分层结构将数据组织到元素 （可以是单个数据项，也可以是其他元素的容器）中。每个 XML 文档将一个元素作为顶级项目或主项目；此根元素内可能会包含一条信息，但更可能会包含其他元素，而这些元素又包含其他元素，依此类推。例如，以下 XML 文档包含有关音乐唱片的信息：

```
<song tracknumber="1" length="4:05">
    <title>What do you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <mood>Happy</mood>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

每个元素都是用一组标签来区分的，即元素名称括在尖括号 （小于号和大于号）中。开始标签 （指示元素的开头）包含元素名称：

```
<title>
```

结束标签 （标记元素的结尾）在元素名称前面包含一个正斜杠：

```
</title>
```

如果元素不包含任何内容，则会将其编写为一个空元素 （有时称为自结束元素）。在 XML 中，以下元素：

```
<lastplayed/>
```

与下面的元素完全相同：

```
<lastplayed></lastplayed>
```

除了在开始和结束标签之间包含的元素内容外，元素还可以包含在元素开始标签中定义的其他值 （称为属性）。例如，以下 XML 元素定义一个名为 length 且值为 "4:19" 的属性：

```
<song length="4:19"></song>
```

每个 XML 元素都包含内容，这可以是单个值、一个或多个 XML 元素或没有任何内容 （对于空元素）。

了解有关 **XML** 的详细信息

要了解有关使用 XML 的详细信息，请参阅额外的一些书籍和资源以了解有关 XML 的详细信息，其中包括以下 Web 站点：

- W3Schools XML 教程：http://w3schools.com/xml/
- XMLpitstop 教程、讨论列表等等：http://xmlpitstop.com/

用于使用 **XML** 的 **ActionScript** 类

ActionScript 3.0 包含一些用于使用 XML 结构化信息的类。下面列出了两个主类：

- XML：表示单个 XML 元素，它可以是包含多个子元素的 XML 文档，也可以是文档中的单值元素。
- XMLList：表示一组 XML 元素。当具有多个 " 同级 " （在 XML 文档层次中位于相同级别，并且包含在相同父级中）的 XML 元素时，将使用 XMLList 对象。例如，XMLList 实例是使用以下一组 XML 元素（可能包含在 XML 文档中）的最简便方法：

  ```
  <artist type="composer">Fred Wilson</artist>
  <artist type="conductor">James Schmidt</artist>
  <artist type="soloist">Susan Harriet Thurndon</artist>
  ```

对于涉及 XML 命名空间的更高级用法， ActionScript 还包含 Namespace 和 QName 类。有关详细信息，请参阅第 93 页的 " 使用 XML 命名空间 "。

除了用于使用 XML 的内置类外， ActionScript 3.0 还包含一些运算符，它们提供了用于访问和使用 XML 数据的特定功能。这种使用这些类和运算符来使用 XML 的方法称为 ECMAScript for XML (E4X)，它是由 ECMA-357 第 2 版规范定义的。

重要概念和术语

以下参考列表包含进行 XML 处理例程编程时会遇到的重要术语：

元素 XML 文档中的单个项目，它被标识为开始标签和结束标签之间包含的内容 （包括标签）。 XML 元素可以包含文本数据或其他元素，也可以为空。

空元素 不包含任何子元素的 XML 元素。通常，将空元素编写为自结束标签 （如 <element/>）。

文档 单个 XML 结构。 XML 文档可以包含任意数量的元素 （或者仅包含单个空元素）；但是， XML 文档必须具有一个顶级元素，该元素包含文档中的所有其他元素。

节点 XML 元素的另一个名称。

属性 与元素关联的命名值，它以 attributename="value" 格式写入到元素的开始标签中，而不是编写为嵌套在元素内的单独子元素。

用于处理 XML 的 E4X 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ECMAScript for XML 规范定义了一组用于使用 XML 数据的类和功能。这些类和功能统称为 E4X。 ActionScript 3.0 包含以下 E4X 类：XML、 XMLList、 QName 和 Namespace。

E4X 类的方法、属性和运算符旨在实现以下目标：

- 简单 — 在可能的情况下，使用 E4X 可以更容易地编写和理解用于使用 XML 数据的代码。
- 一致 — E4X 背后的方法和推理在内部是一致的，并与 ActionScript 的其他部分保持一致。
- 熟悉 — 使用众所周知的运算符来处理 XML 数据，如点 (.) 运算符。

注：ActionScript 2.0 中有一个不同的 XML 类。在 ActionScript 3.0 中，已将该类重命名为 XMLDocument，以使该名称不会与作为 E4X 的一部分的 ActionScript 3.0 XML 类冲突。在 ActionScript 3.0 中， flash.xml 包中包含了 XMLDocument、 XMLNode、 XMLParser 和 XMLTag 几个旧类，主要是用于旧支持。新的 E4X 类是核心类；无需导入包即可使用这些类。有关旧 ActionScript 2.0 XML 类的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 flash.xml 包。

下面是使用 E4X 处理数据的一个示例：

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

通常，应用程序都会从外部源 （如 Web 服务或 RSS 供给）加载 XML 数据。然而，为清楚起见，此处提供的代码示例将 XML 数据作为文本进行分配。

如下面的代码所示， E4X 包含了一些直观运算符 （如点 (.) 和属性标识符 (@) 运算符），用于访问 XML 中的属性：

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

使用 appendChild() 方法可为 XML 分配新的子节点，如以下代码片断所示：

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

使用 @ 和 . 运算符不仅可以读取数据，还可以分配数据，如下所示：

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

使用 for 循环可以循环访问 XML 的节点，如下所示：

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

XML 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML 对象可能表示 XML 元素、属性、注释、处理指令或文本元素。

XML 对象分为包含简单内容和包含复杂内容两类。有子节点的 XML 对象归入包含复杂内容的一类。如果 XML 对象是属性、注释、处理指令或文本节点之中的任何一个，我们就说它包含简单内容。

例如，下面的 XML 对象包含复杂内容，包括一条注释和一条处理指令：

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

如下面的示例所示，现在可以使用 comments() 和 processingInstructions() 方法创建新的 XML 对象（一个是注释，一个是处理指令）：

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

XML 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML 类有五个静态属性：

- ignoreComments 和 ignoreProcessingInstructions 属性确定分析 XML 对象时是否忽略注释或处理指令。
- ignoreWhitespace 属性确定在只由空白字符分隔的元素标签和内嵌表达式中是否忽略空白字符。
- prettyIndent 和 prettyPrinting 属性用于设置由 XML 类的 toString() 和 toXMLString() 方法返回的文本的格式。

有关这些属性的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

XML 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下是 XML 对象的分层结构的使用方法：

- appendChild()
- child()
- childIndex()
- children()
- descendants()
- elements()
- insertChildAfter()
- insertChildBefore()
- parent()
- prependChild()

以下方法可与 XML 对象属性一起使用：

- attribute()
- attributes()

以下方法可与 XML 对象属性一起使用：

- hasOwnProperty()
- propertyIsEnumerable()
- replace()
- setChildren()

以下方法用于与限定名和命名空间一起使用：

- addNamespace()
- inScopeNamespaces()
- localName()
- name()
- namespace()
- namespaceDeclarations()
- removeNamespace()
- setLocalName()
- setName()
- setNamespace()

以下方法用于使用和确定某些类型的 XML 内容：

- comments()
- hasComplexContent()
- hasSimpleContent()

- nodeKind()
- processingInstructions()
- text()

以下方法用于转换为字符串和设置 XML 对象的格式：

- defaultSettings()
- setSettings()
- settings()
- normalize()
- toString()
- toXMLString()

另外还有几个方法：

- contains()
- copy()
- valueOf()
- length()

有关这些方法的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

XMLList 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XMLList 实例表示 XML 对象的任意集合。它可以包含完整的 XML 文档、 XML 片断或 XML 查询结果。

以下是 XMLList 对象的分层结构的使用方法：

- child()
- children()
- descendants()
- elements()
- parent()

以下方法可以与 XMLList 对象属性一起使用：

- attribute()
- attributes()

以下方法可以与 XMLList 属性一起使用：

- hasOwnProperty()
- propertyIsEnumerable()

以下方法用于使用和确定某些类型的 XML 内容：

- comments()
- hasComplexContent()

- hasSimpleContent()
- processingInstructions()
- text()

以下方法用于转换为字符串和设置 XMLList 对象的格式：

- normalize()
- toString()
- toXMLString()

另外还有几个方法：

- contains()
- copy()
- length()
- valueOf()

有关这些方法的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

对于只包含一个 XML 元素的 XMLList 对象，可以使用 XML 类的所有属性和方法，因为包含一个 XML 元素的 XMLList 被视为等同于 XML 对象。例如，在下面的代码中，因为 doc.div 是包含一个元素的 XMLList 对象，所以可以使用 XML 类的 appendChild() 方法：

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

有关 XML 属性和方法的列表，请参阅 第 84 页的 “XML 对象 ”。

初始化 XML 变量

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可将 XML 文本赋予 XML 对象，如下所示：

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

如下面的代码片断所示，还可以使用 new 构造函数从包含 XML 数据的字符串创建 XML 对象的实例：

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

如果字符串中的 XML 数据格式有误 （例如缺少结束标签），则会出现运行时错误。

还可以将数据按引用 （从其他变量）传递到 XML 对象，如下面的示例所示：

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

要从 URL 加载 XML 数据，请使用 URLLoader 类，如下面的示例所示：

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

要从套接字连接读取 XML 数据，请使用 XMLSocket 类。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 XMLSocket 类。

组合和变换 XML 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 prependChild() 方法或 appendChild() 方法可在 XML 对象属性列表的开头或结尾添加属性，如下面的示例所示：

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

使用 insertChildBefore() 方法或 insertChildAfter() 方法在指定属性之前或之后添加属性，如下所示：

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

如下面的示例所示，还可以使用大括号运算符 （{ 和 }）在构造 XML 对象时按引用 （从其他变量）传递数据：

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;

    x = x.appendChild(newnode)
}
```

可以使用 = 运算符将属性指定给 XML 对象，如下所示：

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

这将对 XML 对象 x 进行如下设置：

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

可以使用 + 和 += 运算符连接 XMLList 对象：

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

这将对 XMLList 对象 xList 进行如下设置：

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

遍历 XML 结构

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML 的一个强大功能是它能够通过文本字符的线性字符串提供复杂的嵌套数据。将数据加载到 XML 对象时， ActionScript 会分析数据并将其分层结构加载到内存 （如果 XML 数据格式有误，它会发送运行时错误）。

利用 XML 和 XMLList 对象的运算符和方法可以轻松遍历 XML 数据的结构。

使用点 (.) 运算符和后代存取器 (..) 运算符可以访问 XML 对象的子属性。请考虑下面的 XML 对象：

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

对象 myXML.book 是一个 XMLList 对象，它包含名为 book 的 myXML 对象的子属性。它们是两个 XML 对象，与 myXML 对象的两个 book 属性相匹配。

对象 myXML..lastName 是一个 XMLList 对象，它包含名字为 lastName 的所有后代属性。它们是两个 XML 对象，与 myXML 对象的两个 lastName 相匹配。

myXML.book.editor.lastName 对象是一个 XMLList 对象，它包含 myXML 对象的名为 book 的子对象的名为 editor 的子对象的名为 lastName 的所有子对象：在本例中， XMLList 对象只包含一个 XML 对象 （值为 "Case" 的 lastName 属性）。

访问父节点和子节点

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

parent() 方法返回 XML 对象的父项。

可以使用子级列表的序数索引值访问特定的子对象。例如，假设 XML 对象 myXML 有两个名为 book 的子属性。每个名为 book 的子属性都有一个与之关联的索引编号：

```
myXML.book[0]
myXML.book[1]
```

若要访问特定的孙项，可为子项和孙项名称同时指定索引编号：

```
myXML.book[0].title[0]
```

不过，如果 x.book[0] 只有一个名为 title 的子项，则可以省略索引引用，如下所示：

```
myXML.book[0].title
```

同样，如果对象 x 只有一个 book 子对象，并且该子对象只有一个 title 对象，则可以同时省略两个索引引用，如下所示：

```
myXML.book.title
```

可以使用 child() 方法导航到名称基于变量或表达式的子项，如下面的示例所示：

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

访问属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 @ 符号 （属性标识符运算符）可以访问 XML 或 XMLList 对象的属性，如下面的代码所示：

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

可将 * 通配符和 @ 符号一起使用来访问 XML 或 XMLList 对象的所有属性，如下面的代码所示：

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

可以使用 attribute() 或 attributes() 方法访问 XML 或 XMLList 对象的特定属性或所有属性，如下面的代码所示：

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

请注意，还可以使用以下语法访问属性，如下面的示例所示：

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

其中每一个都等效于 employee.@id。但是，语法 employee.@id 是首选方式。

按属性或元素值过滤

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用括号运算符 — (和) — 过滤具有特定元素名称或属性值的元素。请考虑下面的 XML 对象：

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

以下表达式都是有效的：

- x.employee.(lastName == "McGee") — 这是第二个 employee 节点。
- x.employee.(lastName == "McGee").firstName — 这是第二个 employee 节点的 firstName 属性。
- x.employee.(lastName == "McGee").@id — 这是第二个 employee 节点的 id 属性的值。
- x.employee.(@id == 347) — 第一个 employee 节点。
- x.employee.(@id == 347).lastName — 这是第一个 employee 节点的 lastName 属性。
- x.employee.(@id > 300) — 这是具有两个 employee 属性的 XMLList。
- x.employee.(position.toString().search("analyst") > -1) — 这是具有两个 position 属性的 XMLList。

如果您尝试过滤不存在的属性或元素，会引发异常。例如，以下代码的最后一行产生一个错误，因为第二个 p 元素没有 id 属性：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

同样，以下代码的最后一行也会产生一个错误，因为第二个 p 元素没有 b 属性：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

为避免出现这些错误，可以使用 attribute() 和 elements() 方法来识别具有匹配属性或元素的属性，如以下代码所示：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

还可以使用 hasOwnProperty() 方法，如下面的代码所示：

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

使用 for..in 和 for each..in 语句

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 包含 for..in 语句和 for each..in 语句，用于遍历 XMLList 对象。例如，我们来看下面的 XML 对象 myXML 和 XMLList 对象 myXML.item。 XMLList 对象 myXML.item 由 XML 对象的两个 item 节点组成。

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

for..in 语句用于遍历 XMLList 中的一组属性名称：

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

for each..in 语句用于遍历 XMLList 中的属性：

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

使用 XML 命名空间

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML 对象 （或文档）中的命名空间用于标识对象所包含的数据的类型。例如，在将 XML 数据发送和提交给使用 SOAP 消息传递协议的 Web 服务时，您要在 XML 的开始标签中声明命名空间：

```
var message:XML =
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

命名空间有一个前缀 soap 和一个定义该命名空间的 URI http://schemas.xmlsoap.org/soap/envelope/。

ActionScript 3.0 包含用于使用 XML 命名空间的命名空间类。对于上一示例中的 XML 对象，可按如下方式使用命名空间类：

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

XML 类包含下列可与命名空间一起使用的方法：addNamespace()、 inScopeNamespaces()、 localName()、 name()、 namespace()、 namespaceDeclarations()、 removeNamespace()、 setLocalName()、 setName() 和 setNamespace()。

使用 default xml namespace 指令可为 XML 对象指定默认命名空间。例如，在以下代码中，x1 和 x2 具有相同的默认命名空间：

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

XML 类型转换

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以将 XML 对象和 XMLList 对象转换为字符串值。同样，也可以将字符串转换为 XML 对象和 XMLList 对象。还要记住，所有 XML 属性值、名称和文本值都是字符串。以下几节将讨论所有这些形式的 XML 类型转换。

将 XML 和 XMLList 对象转换为字符串

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML 和 XMLList 类都包含 toString() 方法和 toXMLString() 方法。toXMLString() 方法返回一个字符串，其中包含该 XML 对象的所有标签、属性、命名空间声明和内容。对于包含复杂内容 （子元素）的 XML 对象， toString() 方法的作用与 toXMLString() 方法完全相同。对于包含简单内容的 XML 对象 （只包含一个文本元素的对象）， toString() 方法只返回该元素的文本内容，如下面的示例所示：

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

如果使用 trace() 方法但不指定 toString() 或 toXMLString()，则默认情况下将使用 toString() 方法转换数据，如以下代码所示：

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

使用 trace() 方法调试代码时，通常都希望使用 toXMLString() 方法，以便 trace() 方法输出更完整的数据。

将字符串转换为 XML 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 new XML() 构造函数从字符串创建 XML 对象，如下所示：

```
var x:XML = new XML("<a>test</a>");
```

如果试图将表示无效 XML 或格式有误的 XML 的字符串转换为 XML，则会引发运行时错误，如下所示：

```
var x:XML = new XML("<a>test"); // throws an error
```

从字符串转换属性值、名称和文本值

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

所有 XML 属性值、名称和文本值都是 String 数据类型，可能需要将它们转换为其他数据类型。例如，以下代码使用 Number() 函数将文本值转换为数字：

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.95;
```

如果这些代码不使用 Number() 函数，它们将把 + 运算符解释为字符串连接运算符，最后一行中的 trace() 方法将输出以下结果：

```
01.003.95
```

读取外部 XML 文档

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 URLLoader 类从 URL 加载 XML 数据。若要在您的应用程序中使用以下代码，请将示例中的 XML_URL 值替换为有效的 URL：

```
import flash.events.Event;
import flash.net.URLLoader;

var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener(Event.COMPLETE, xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

还可以使用 XMLSocket 类设置与服务器的异步 XML 套接字连接。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

在 ActionScript 中使用 XML 的示例：从 Internet 加载 RSS 数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

RSSViewer 范例应用程序说明了在 ActionScript 中使用 XML 的一些功能，其中包括：

- 使用 XML 方法以 RSS 供给的形式遍历 XML 数据。
- 使用 XML 方法以 HTML 的形式组合 XML 数据，以便在文本字段中使用。

RSS 格式被广泛用于通过 XML 收集新闻。简单的 RSS 数据文件可能如下所示：

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
    <title>Alaska - Weather</title>
    <link>http://www.nws.noaa.gov/alerts/ak.html</link>
    <description>Alaska - Watches, Warnings and Advisories</description>

    <item>
        <title>
            Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
        </title>
        <link>
            http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
</item>
    <item>
        <title>
            Short Term Forecast - Haines Borough (Alaska)
        </title>
            <link>
            http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
        </link>
        <description>
            Short Term Forecast Issued At: 2005-04-11T19:00:00
            Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
            Homepage: http://pajk.arh.noaa.gov
        </description>
    </item>
</channel>
</rss>
```

SimpleRSS 应用程序从 Internet 上读取 RSS 数据，分析标题、链接和描述的数据，并返回这些数据。 SimpleRSSUI 类提供了相应的用户界面，并会调用 SimpleRSS 类，从而执行所有 XML 处理。

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。 RSSViewer 应用程序文件位于 Samples/RSSViewer 文件夹中。该应用程序包含以下文件：

文件	说明
RSSViewer.mxml 或 RSSViewer.fla	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/rssViewer/RSSParser.as	一个类，包含的方法可以使用 E4X 遍历 RSS (XML) 数据并生成相应的 HTML 表示形式。
RSSData/ak.rss	一个 RSS 范例文件。该应用程序被设置为从 Web 上由 Adobe 托管的 Flex RSS 供给处读取 RSS 数据。不过，您也可以轻松更改该应用程序，使之从此文档读取 RSS 数据，此文档所用的架构与 Flex RSS 供给的架构略有不同。

读取和分析 XML 数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

RSSParser 类包含一个 xmlLoaded() 方法，该方法可将输入 RSS 数据（存储在 rssXML 变量中）转换为包含 HTML 格式的输出 (rssOutput) 的字符串。

如果源 RSS 数据包含默认的命名空间，代码会在此方法的开头附近设置默认的 XML 命名空间：

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

下面的几行代码循环访问源 XML 数据的内容，以检查名为 item 的各个后代属性：

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                            itemDescription,
                            itemLink);
}
```

前三行代码只是设置字符串变量，以表示 XML 数据的 item 属性的标题、描述和链接属性。下一行随后调用 buildItemHTML() 方法，以 XMLList 对象形式获取 HTML 数据，并使用三个新的字符串变量作为参数。

组合 XMLList 数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

HTML 数据 （XMLList 对象）具有如下形式：

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

方法的第一行清除默认的 XML 命名空间：

```
default xml namespace = new Namespace();
```

default xml namespace 指令具有函数块级作用域。这意味着此声明的作用域是 buildItemHTML() 方法。

下面的几行代码基于传递给该函数的字符串参数组合 XMLList：

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
        // <font color="#008000"></font></a>
        // 0x008000 = green
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

此 XMLList 对象表示适用于 ActionScript HTML 文本字段的字符串数据。

xmlLoaded() 方法使用 buildItemHTML() 方法的返回值并将其转换为字符串：

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

提取 RSS 源的标题并发送自定义事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

xmlLoaded() 方法根据源 RSS XML 数据中的信息设置 rssTitle 字符串变量：

```
rssTitle = rssXML.channel.title.toString();
```

最后， xmlLoaded() 方法生成一个事件，通知应用程序数据已经过分析并且可用：

```
            dataWritten = new Event("dataWritten", true);
```

第 7 章：使用本机 JSON 功能

ActionScript 3.0 提供了一个本机 API，用于采用 JavaScript Object Notation (JSON) 格式对 ActionScript 对象进行编码和解码。 JSON 类和支持成员函数遵循 ECMA-262 第 5 版规范，并稍有改动。

社区成员 Todd Anderson 提供了本机 JSON API 与第三方 as3corelib JSON 类的比较。请参阅 在 Flash Player 11 中使用本机 JSON。

更多帮助主题

JSON

JSON API 概述

ActionScript JSON API 由 JSON 类和几个本机类上的 toJSON() 成员函数组成。对于要求任何类提供自定义 JSON 编码的应用程序， ActionScript 框架将提供覆盖默认编码的方法。

JSON 类内部处理任何不提供 toJSON() 成员的 ActionScript 类的导入和导出。对于此类情况， JSON 将遍历所遇到的每一个对象的公共属性。如果一个对象包含其他对象， JSON 将以递归形式访问嵌套对象，并执行相同的遍历。如果任何对象提供 toJSON() 方法， JSON 将使用该自定义方法，而不是其内部算法。

JSON 接口包含一个编码方法 stringify() 和一个解码方法 parse()。其中每种方法都提供一个参数，您可以将自己的逻辑插入 JSON 编码和解码工作流程。对于 stringify()，此参数名为 replacer ；对于 parse()，此参数为 reviver。这些参数使用以下签名通过两个参数来定义函数：

```
function(k, v):*
```

toJSON() 方法

toJSON() 方法的签名为

```
public function toJSON(k:String):*
```

JSON.stringify() 在遍历对象期间，对遇到的每个公共属性调用 toJSON() （如果存在）。该属性由一个密钥 - 值对组成。当 stringify() 调用 toJSON() 时，它将传入当前检查的属性的密钥 (k)。典型的 toJSON() 实现计算每个属性名称，并采用所需编码返回其值。

toJSON() 方法可以返回任何类型的值 （用 * 表示），而不仅仅是字符串。此变量返回类型允许 toJSON() 返回一个对象 （如果适用）。例如，如果您的自定义类的某个属性包含来自其他第三方库中的对象，当 toJSON() 遇到您的属性时，您可以返回该对象。随后， JSON 以递归形式访问第三方对象。编码处理流的行为如下：

- 如果 toJSON() 返回一个计算结果不是字符串的对象，则 stringify() 将以递归形式访问该对象。
- 如果 toJSON() 返回一个字符串，则 stringify() 会将此值包装到另一个字符串中，返回包装后的字符串，然后移到下一个值。

在许多情况下，返回对象操作会优先返回您的应用程序所创建的 JSON 字符串。返回对象操作将充分利用内置 JSON 编码算法，并且还允许 JSON 以递归形式访问嵌套对象。

toJSON() 方法不是在 Object 类中定义，也不是在大多数其他本地类中定义。如果缺少此方法，则 JSON 必须对对象的公共属性执行标准遍历。您也可以根据需要使用 toJSON() 来公开对象的私有属性。

不过，有些本地类可能会出现一些问题，因为 ActionScript 库不能有效地解决所有使用案例。对于这些类， ActionScript 提供一个简单实现，客户端可以重新实现以满足其需求。提供普通 toJSON() 成员的类包括：

- ByteArray

- Date
- Dictionary
- XML

您可以使 ByteArray 类子类化以覆盖其 toJSON() 方法，或者您也可以重新定义其原型。最后声明的 Date 和 XML 类要求使用类原型来重新定义 toJSON()。 Dictionary 类声明为动态类，您可以使用该类随时覆盖 toJSON()。

定义自定义 JSON 行为

如果要对本地类实施您自己的 JSON 编码和解码，则可以选择以下几个选项：

- 定义或覆盖非最终本地类的自定义子类上的 toJSON()
- 定义或重新定义类原型上的 toJSON()
- 定义动态类上的 toJSON 属性
- 使用 JSON.stringify() replacer 和 JSON.parser() reviver 参数

更多帮助主题

ECMA-262，第 5 版

在内置类的原型上定义 toJSON()

ActionScript 中的本机 JSON 实现镜像 ECMA-262 第 5 版中定义的 ECMAScript JSON 机制。由于 ECMAScript 不支持类，因此 ActionScript 基于原型的调度定义 JSON 行为。原型是 ActionScript 3.0 类的前体，允许模拟继承以及成员添加和重新定义。

ActionScript 允许您在任何类的原型上定义或重新定义 toJSON()。这种权限也适用于标记为 final 的类。在类原型上定义 toJSON() 时，您的定义将成为应用程序范围内该类的所有实例的最新定义。例如，下面介绍如何在 MovieClip 原型上定义 toJSON() 方法：

```
MovieClip.prototype.toJSON = function(k):* {
    trace("prototype.toJSON() called.");
    return "toJSON";
}
```

当您的应用程序随后在任何 MovieClip 实例上调用 stringify() 时， stringify() 将返回 toJSON() 方法的输出：

```
var mc:MovieClip = new MovieClip();
var js:String = JSON.stringify(mc); //"prototype toJSON() called."
trace("js: " + js); //"js: toJSON"
```

您还可以在定义该方法的本地类中覆盖 toJSON()。例如，以下代码覆盖 Date.toJSON()：

```
Date.prototype.toJSON = function (k):* {
    return "any date format you like via toJSON: "+
        "this.time:"+this.time + " this.hours:"+this.hours;
}
var dt:Date = new Date();
trace(JSON.stringify(dt));
// "any date format you like via toJSON: this.time:1317244361947 this.hours:14"
```

定义或覆盖类级别的 toJSON()

并不总是需要应用程序使用原型重新定义 toJSON()。如果父类不是最终类，您也可以将 toJSON() 定义为子类的成员。例如，您可以扩展 ByteArray 类并定义公用 toJSON() 函数：

```
package {

    import flash.utils.ByteArray;
    public class MyByteArray extends ByteArray
    {
        public function MyByteArray() {
        }

        public function toJSON(s:String):*
        {
            return "MyByteArray";
        }

    }
}


var ba:ByteArray = new ByteArray();
trace(JSON.stringify(ba)); //"ByteArray"
var mba:MyByteArray = new MyByteArray(); //"MyByteArray"
trace(JSON.stringify(mba)); //"MyByteArray"
```

如果类是动态类，您可以将 toJSON 属性添加到该类的对象中，并为其分配一个函数，如下所示：

```
var d:Dictionary = new Dictionary();
trace(JSON.stringify((d))); // "Dictionary"
d.toJSON = function(){return {c : "toJSON override."};} // overrides existing function
trace(JSON.stringify((d))); // {"c":"toJSON override."}
```

您可以在任何 ActionScript 类上覆盖、定义或重新定义 toJSON()。但是，大多数内置 ActionScript 类不会定义 toJSON()。Object 类不会以默认的原型定义 toJSON，也不会将其声明为类成员。只有其他少数本地类将该方法定义为原型函数。因此，在大多数类中，您不能以传统方式覆盖 toJSON()。

不定义 toJSON 的本地类将通过内部 JSON 实施序列化为 JSON。请尽可能避免替换此内置功能。如果您定义 toJSON() 成员，则 JSON 类将使用您的逻辑而不是其自己的函数。

使用 JSON.stringify() replacer 参数

如果需要在整个应用程序中更改类的 JSON 导出行为，则可以选择覆盖原型的 toJSON()。但是，在某些情况下，导出逻辑可能只适用于瞬态条件下的特殊情况。要适应此小范围的更改，您可以使用 JSON.stringify() 方法的 replacer 参数。

stringify() 方法应用从 replacer 参数传递到要编码的对象的函数。此函数的签名与 toJSON() 的签名类似：

```
function (k,v):*
```

与 toJSON() 不同， replacer 函数需要值 v 和密钥 k。此差异是必要的，因为 stringify() 是在静态 JSON 对象而不是要编码的对象上定义的。当 JSON.stringify() 调用 replacer(k,v) 时，它将遍历原始输入对象。传递到 replacer 函数的隐式 this 参数表示保存密钥和值的对象。由于 JSON.stringify() 不修改原始输入对象，因此该对象在要遍历的容器中保持不变。因此，您可以使用代码 this[k] 查询原始对象上的密钥。 v 参数保存 toJSON() 转换的值。

与 toJSON() 相同， replacer 函数可以返回任何类型的值。如果 replacer 返回字符串，则 JSON 引擎将对引号中的内容进行转义，然后使用引号包装转义内容。此包装可确保 stringify() 接收到有效的 JSON 字符串对象，该对象在对 JSON.parse() 的后续调用中仍然为一个字符串。

以下代码使用 replacer 参数和隐式 this 参数返回 Date 对象的 time 和 hours 值：

```
JSON.stringify(d, function (k,v):* {
    return "any date format you like via replacer: "+
        "holder[k].time:"+this[k].time + " holder[k].hours:"+this[k].hours;
});
```

使用 JSON.parse() reviver 参数

JSON.parse() 方法的 reviver 参数与 replacer 函数相反：它将 JSON 字符串转换为可使用的 ActionScript 对象。 reviver 参数是一个具有两个参数并返回任何类型的函数：

```
function (k,v):*
```

在此函数中，k 是一个密钥，v 是 k 的值。与 stringify() 一样，parse() 遍历 JSON 密钥 - 值对，并将 reviver 函数 （如果存在）应用于每个对。一个潜在的问题是：事实上，JSON 类不输出对象的 ActionScript 类名称。因此，了解要恢复的对象类型具有一定的难度。如果是嵌套对象，此问题将特别麻烦。在设计 toJSON()、 replacer 和 reviver 函数时，您可以想办法在保持原对象完整性的同时识别导出的 ActionScript 对象。

解析示例

下面的示例显示了恢复从 JSON 字符串解析的对象的策略。此示例定义 JSONGenericDictExample 和 JSONDictionaryExtnExample 两个类。JSONGenericDictExample 类是一个自定义 Dictionary 类。每个记录都包含一个人的姓名、生日以及唯一 ID。每次调用 JSONGenericDictExample 构造函数时，均会将新创建的对象添加到内部静态数组，并使用一个静态递增的整数作为其 ID。JSONGenericDictExample 类还定义了 revive() 方法，该方法只从较长的 id 成员提取整数部分。 revive() 方法使用此整数来查找并返回正确的可恢复对象。

JSONDictionaryExtnExample 类扩展了 ActionScript Dictionary 类。其记录没有固定的结构，可以包含任何数据。在构建 JSONDictionaryExtnExample 对象之后即为其指定数据，而不是作为类定义的属性来指定。JSONDictionaryExtnExample 记录将 JSONGenericDictExample 对象用作密钥。JSONDictionaryExtnExample 对象恢复时， JSONGenericDictExample.revive() 函数使用与 JSONDictionaryExtnExample 关联的 ID 来检索正确的密钥对象。

最重要的是， JSONDictionaryExtnExample.toJSON() 方法将返回包含 JSONDictionaryExtnExample 对象在内的标记字符串。此字符串将 JSON 输出标识为属于 JSONDictionaryExtnExample 类。通过此标记，我们可以清楚地了解 JSON.parse() 期间处理的对象类型。

```
package {
    // Generic dictionary example:
    public class JSONGenericDictExample {
        static var revivableObjects = [];
        static var nextId = 10000;
        public var id;
        public var dname:String;
        public var birthday;

        public function JSONGenericDictExample(name, birthday) {
            revivableObjects[nextId] = this;
            this.id       = "id_class_JSONGenericDictExample_" + nextId;
            this.dname     = name;
            this.birthday = birthday;
            nextId++;
        }
        public function toString():String { return this.dname; }
        public static function revive(id:String):JSONGenericDictExample {
            var r:RegExp = /^id_class_JSONGenericDictExample_([0-9]*)$/;
            var res = r.exec(id);
            return JSONGenericDictExample.revivableObjects[res[1]];
        }
    }
}
```

```
package {
    import flash.utils.Dictionary;
    import flash.utils.ByteArray;

    // For this extension of dictionary, we serialize the contents of the
    // dictionary by using toJSON
    public final class JSONDictionaryExtnExample extends Dictionary {
        public function toJSON(k):* {
            var contents = {};
            for (var a in this) {
                contents[a.id] = this[a];
            }

            // We also wrap the contents in an object so that we can
            // identify it by looking for the marking property "class E"
            // while in the midst of JSON.parse.
            return {"class JSONDictionaryExtnExample": contents};
        }

        // This is just here for debugging and for illustration
        public function toString():String {
            var retval = "[JSONDictionaryExtnExample <";
            var printed_any = false;
            for (var k in this) {
                retval += k.toString() + "=" +
                "[e="+this[k].earnings +
                ",v="+this[k].violations + "], "
                printed_any = true;
            }
            if (printed_any)
                retval = retval.substring(0, retval.length-2);
            retval += ">]"
            return retval;
        }
    }
}
```

下列运行时脚本调用 JSONDictionaryExtnExample 对象上的 JSON.parse() 时， reviver 函数将调用 JSONDictionaryExtnExample 中每个对象上的 JSONGenericDictExample revive()。此调用将提取表示对象密钥的 ID。JSONGenericDictExample.revive() 函数使用此 ID 从私有静态数组中检索并返回存储的 JSONDictionaryExtnExample 对象。

```
import flash.display.MovieClip;
import flash.text.TextField;

var a_bob1:JSONGenericDictExample = new JSONGenericDictExample("Bob", new Date(Date.parse("01/02/1934")));
var a_bob2:JSONGenericDictExample = new JSONGenericDictExample("Bob", new Date(Date.parse("05/06/1978")));
var a_jen:JSONGenericDictExample = new JSONGenericDictExample("Jen", new Date(Date.parse("09/09/1999")));

var e = new JSONDictionaryExtnExample();
e[a_bob1] = {earnings: 40, violations: 2};
e[a_bob2] = {earnings: 10, violations: 1};
e[a_jen]  = {earnings: 25, violations: 3};

trace("JSON.stringify(e): " + JSON.stringify(e)); // {"class JSONDictionaryExtnExample":
                        //{"id_class_JSONGenericDictExample_10001":
                        //{"earnings":10,"violations":1},
                        //"id_class_JSONGenericDictExample_10002":
                        //{"earnings":25,"violations":3},
                        //"id_class_JSONGenericDictExample_10000":
                        // {"earnings":40,"violations":2}}}

var e_result = JSON.stringify(e);
```

```
var e1 = new JSONDictionaryExtnExample();
var e2 = new JSONDictionaryExtnExample();

// It's somewhat easy to convert the string from JSON.stringify(e) back
// into a dictionary (turn it into an object via JSON.parse, then loop
// over that object's properties to construct a fresh dictionary).
//
// The harder exercise is to handle situations where the dictionaries
// themselves are nested in the object passed to JSON.stringify and
// thus does not occur at the topmost level of the resulting string.
//
// (For example: consider roundtripping something like
//   var tricky_array = [e1, [[4, e2, 6]], {table:e3}]
// where e1, e2, e3 are all dictionaries.  Furthermore, consider
// dictionaries that contain references to dictionaries.)
//
// This parsing (or at least some instances of it) can be done via
// JSON.parse, but it's not necessarily trivial.  Careful consideration
// of how toJSON, replacer, and reviver can work together is
// necessary.

var e_roundtrip =
    JSON.parse(e_result,
               // This is a reviver that is focused on rebuilding JSONDictionaryExtnExample objects.
               function (k, v) {
                     if ("class JSONDictionaryExtnExample" in v) { // special marker tag;
                         //see JSONDictionaryExtnExample.toJSON().
                       var e = new JSONDictionaryExtnExample();
                       var contents = v["class JSONDictionaryExtnExample"];
                       for (var i in contents) {
                           // Reviving JSONGenericDictExample objects from string
                           // identifiers is also special;
                           // see JSONGenericDictExample constructor and
                           // JSONGenericDictExample's revive() method.
                           e[JSONGenericDictExample.revive(i)] = contents[i];
                       }
                       return e;
                     } else {
                       return v;
                     }
               });

trace("// == Here is an extended Dictionary that has been round-tripped  ==");
trace("// == Note that we have revived Jen/Jan during the roundtrip.    ==");
trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                            //Jen=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jen=[e=25,v=3]>]
trace("Is e_roundtrip a JSONDictionaryExtnExample? " + (e_roundtrip is JSONDictionaryExtnExample)); //true
trace("Name change: Jen is now Jan");
a_jen.dname = "Jan"

trace("e:           " + e); //[JSONDictionaryExtnExample <Bob=[e=40,v=2], Bob=[e=10,v=1],
                            //Jan=[e=25,v=3]>]
trace("e_roundtrip: " + e_roundtrip); //[JSONDictionaryExtnExample <Bob=[e=40,v=2],
                                      //Bob=[e=10,v=1], Jan=[e=25,v=3]>]
```

第 8 章：处理事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

利用事件处理系统，程序员可以十分方便地响应用户输入和系统事件。 ActionScript 3.0 事件模型不仅方便，而且符合标准，并且它还与显示列表完美集成在一起。新的事件模型基于文档对象模型 (DOM) 第 3 级事件规范，是业界标准的事件处理体系结构，为 ActionScript 程序员提供了强大而直观的事件处理工具。

ActionScript 3.0 事件处理系统与显示列表密切交互。若要对显示列表有基本的了解，请阅读第 126 页的 " 显示编程 "。

更多帮助主题

flash.events 包

文档对象模型 (DOM) 第 3 级事件规范

事件处理基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以将事件视为 SWF 文件中发生的程序员感兴趣的任何类型的事件。例如，大多数 SWF 文件都支持某些类型的用户交互，无论是像响应鼠标单击这样简单的用户交互，还是像接受和处理表单中输入的数据这样复杂的用户交互。与 SWF 文件进行的任何此类用户交互都可以视为事件。也可能会在没有任何直接用户交互的情况下发生事件，例如，从服务器加载完数据或者连接的摄像头变为活动状态时。

在 ActionScript 3.0 中，每个事件都由一个事件对象表示。事件对象是 Event 类或其某个子类的实例。事件对象不但存储有关特定事件的信息，还包含便于操作此事件对象的方法。例如，当 Flash Player 或 AIR 检测到鼠标单击时，它会创建一个事件对象 （MouseEvent 类的实例）以表示该特定鼠标单击事件。

创建事件对象之后， Flash Player 或 AIR 即 " 调度 " 该事件对象，这意味着将该事件对象传递给作为事件目标的对象。作为被调度事件对象的目标的对象称为 " 事件目标 "。例如，当连接的摄像头变为活动状态时， Flash Player 会向事件目标直接调度一个事件对象，此时，该事件对象就是代表摄像头的对象。但是，如果事件目标位于显示列表中，则事件对象会沿显示列表层次向下传递，直到到达事件目标为止。在某些情况下，事件对象随后会沿着相同路线在显示列表层次中向上 " 冒泡 " 回去。这种在显示列表层次中遍历的活动称为事件流。

您可以使用事件侦听器 " 侦听 " 代码中的事件对象。 " 事件侦听器 " 是您编写的用于响应特定事件的函数或方法。若要确保您的程序响应事件，必须将事件侦听器添加到事件目标，或添加到作为事件对象事件流的一部分的任何显示列表对象。

无论何时编写事件侦听器代码，该代码都会采用以下基本结构 （以粗体显示的元素是占位符，您将针对具体情况对其进行填写）：

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

eventTarget.addEventListener(EventType.EVENT_NAME, eventResponse);
```

此代码完成两项任务。首先，它定义一个函数，这是指定为响应事件而执行的动作的方法。接下来，调用源对象的 addEventListener() 方法，实际上就是为指定事件 " 订阅 " 该函数，以便当该事件发生时，执行该函数的操作。当事件实际发生时，事件目标将检查其注册为事件侦听器的所有函数和方法的列表。然后，它依次调用每个函数或方法，同时将事件对象作为参数传递。

您需要在此代码中更改四项内容以创建自己的事件侦听器。第一，必须将函数名称更改为要使用的名称 （必须在两个位置更改此内容，代码将在此处显示 **eventResponse**）。第二，必须为要侦听的事件 （代码中的 **EventType**）所调度的事件对象指定相应的类名称，并且必须为特定事件 （列表中的 **EVENT_NAME**）指定相应的常量。第三，必须针对调度事件 （此代码中的 **eventTarget**）的对象调用 addEventListener() 方法。您可以选择更改用作函数参数 （此代码中的 **eventObject**）的变量的名称。

重要概念和术语

以下参考列表包括您在编写事件处理例程时会遇到的重要术语。

冒泡 一些事件会发生冒泡，以使父显示对象可以响应其子项调度的事件。

冒泡阶段 事件流中向上传播到父显示对象的事件所在部分。冒泡阶段发生在捕获和目标阶段之后。

捕获阶段 事件流中从最常规的目标向下传播到最具体的目标对象的事件所在部分。捕获阶段发生在目标和冒泡阶段之前。

默认行为 某些事件包含通常与事件同时发生的行为，称为默认行为。例如，当用户在文本字段中键入文本时，将引发文本输入事件。该事件的默认行为是实际显示在文本字段中键入的字符，但您可以覆盖该默认行为 （如果由于某种原因，您不希望显示键入的字符）。

调度 通知事件侦听器发生了某事件。

事件 在某对象上发生的情况，该对象可以将此情况告知其他对象。

事件流 当显示列表上的对象 （屏幕上显示的对象）发生事件时，将通知包括该对象在内的所有对象发生了此事件，并依次通知它们的事件侦听器。此过程从舞台开始，并在显示列表中一直进行到发生事件的实际对象，然后再返回到舞台。此过程称为事件流。

事件对象 一个包含发生的特定事件的相关信息的对象，当调度事件时，此信息将被发送到所有侦听器。

事件目标 实际调度事件的对象。例如，如果用户单击位于 Sprite （位于舞台内）内的按钮，则所有这些对象将调度事件，但事件目标是指实际发生事件的对象，此处指单击的按钮。

侦听器 一个已将自身注册到某对象的对象或函数，以指示当特定事件发生时应该通知它。

目标阶段 事件已到达最具体的可能目标时所在的事件流点。目标阶段发生在捕获和冒泡阶段之间。

ActionScript 3.0 事件处理与早期版本事件处理的不同之处

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 中的事件处理与早期 ActionScript 版本中的事件处理之间的一个最显著的区别是: 在 ActionScript 3.0 中，只有一个事件处理系统，而在早期的 ActionScript 版本中，则有几个不同的事件处理系统。本部分先概述早期 ActionScript 版本中的事件处理的工作原理，然后讨论 ActionScript 3.0 中的事件处理的变化情况。

早期 ActionScript 版本中的事件处理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 之前的 ActionScript 版本中提供许多不同的方法来处理事件:

- on() 事件处理函数，可以直接放在 Button 和 MovieClip 实例上
- onClipEvent() 处理函数，可以直接放在 MovieClip 实例上
- 回调函数属性，例如 XML.onload 和 Camera.onActivity
- 使用 addListener() 方法注册的事件侦听器

- 部分实现了 DOM 事件模型的 UIEventDispatcher 类。

其中的每一种机制都有其自己的若干优点和局限性。on() 和 onClipEvent() 处理函数易于使用，但使随后对项目的维护变得较为困难，因为很难查找直接放在按钮和影片剪辑上的代码。回调函数也很容易实现，但对于任何指定事件，仅限于使用一个回调函数。事件侦听器较难实现：它们不但要求创建侦听器对象和函数，而且要求向生成事件的对象注册侦听器。这虽然增加了开销，但您可以创建若干侦听器对象，并针对同一个事件注册这些对象。

对 ActionScript 2.0 组件的开发形成了另一个事件模型。该新模型包含在 UIEventDispatcher 类中，并且基于 DOM 事件规范的子集。熟悉组件事件处理的开发人员将会发现过渡到新的 ActionScript 3.0 事件模型相对来说较为容易。

遗憾的是，各个事件模型使用的语法以不同的方式相互重叠，并且在其他方面各自不同。例如，在 ActionScript 2.0 中，某些属性 （例如 TextField.onChanged）可用作回调函数或事件侦听器。但是，根据您是否在使用支持侦听器或六个类之一的 UIEventDispatcher 类，用于注册侦听器对象的语法有所不同。对于 Key、Mouse、MovieClipLoader、Selection、Stage 和 TextField 类；请使用 addListener() 方法，但对于组件事件处理，请使用名为 addEventListener() 的方法。

不同事件处理模型所导致的另一个复杂性是：根据所使用的机制的不同，事件处理函数的范围大不相同。也就是说，关键字 this 的含义在各个事件处理系统中并不一致。

ActionScript 3.0 中的事件处理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 引入了单一事件处理模型，以替代以前各语言版本中存在的众多不同的事件处理机制。该新事件模型基于文档对象模型 (DOM) 第 3 级事件规范。虽然 SWF 文件格式并不专门遵循文档对象模型标准，但显示列表和 DOM 结构之间存在的相似性足以使 DOM 事件模型的实现成为可能。显示列表中的对象类似于 DOM 层次结构中的节点，在本讨论中，术语 “ 显示列表对象 ” 和 “ 节点 ” 可互换使用。

Flash Player 和 AIR 实现的 DOM 事件模型包括一个名为 “ 默认行为 ” 的概念。“ 默认行为 ” 是 Flash Player 或 AIR 作为特定事件的正常后果而执行的操作。

默认行为

开发人员通常负责编写响应事件的代码。但在某些情况下，行为通常与某一事件关联，使得 Flash Player 或 AIR 会自动执行该行为，除非开发人员添加了取消该行为的代码。由于 Flash Player 或 AIR 会自动表现该行为，因此这类行为称为默认行为。

例如，当用户在 TextField 对象中输入文本时，普遍期待文本显示在该 TextField 对象中，因此该行为被内置到 Flash Player 和 AIR 中。如果您不希望该默认行为发生，可以使用新的事件处理系统来取消它。当用户在 TextField 对象中输入文本时，Flash Player 或 AIR 会创建 TextEvent 类的实例以表示该用户输入。若要阻止 Flash Player 或 AIR 显示 TextField 对象中的文本，必须访问该特定 TextEvent 实例并调用该实例的 preventDefault() 方法。

并非所有默认行为都可以被阻止。例如，当用户双击 TextField 对象中的单词时， Flash Player 和 AIR 会生成一个 MouseEvent 对象。无法阻止的默认行为是：加亮鼠标点击的单词。

许多类型的事件对象没有关联的默认行为。例如，当建立网络连接时， Flash Player 调度一个连接事件对象，但没有与该对象关联的默认行为。 Event 类及其子类的 API 文档列出了每一类型的事件，并说明所有关联的默认行为，以及是否可以阻止该行为。

默认行为仅与由 Flash Player 或 AIR 所调度的事件对象关联，但通过 ActionScript 以编程方式调度的事件对象则不存在默认行为。了解这一点很重要。例如，可以使用 EventDispatcher 类的方法来调度类型为 textInput 的事件对象，但该事件对象没有关联的默认行为。也就是说， Flash Player 和 AIR 不会因为您以编程方式调度了 textInput 事件而在 TextField 对象中显示字符。

ActionScript 3.0 中事件侦听器的新增功能

对于使用 ActionScript 2.0 addListener() 方法的开发人员来说，了解 ActionScript 2.0 事件侦听器模型和 ActionScript 3.0 事件模型之间的差别可能会有所帮助。下表说明两个事件模型之间的几个主要差别：

- 若要在 ActionScript 2.0 中添加事件侦听器，请在某些情况下使用 addListener()，在其他情况下使用 addEventListener()；而在 ActionScript 3.0 中，则始终使用 addEventListener()。
- ActionScript 2.0 中没有事件流，这意味着，只能对广播事件的对象调用 addListener() 方法；而在 ActionScript 3.0 中，可以对属于事件流一部分的任何对象调用 addEventListener() 方法。
- 在 ActionScript 2.0 中，事件侦听器可以是函数、方法或对象，而在 ActionScript 3.0 中，只有函数或方法可以是事件侦听器。

事件流

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

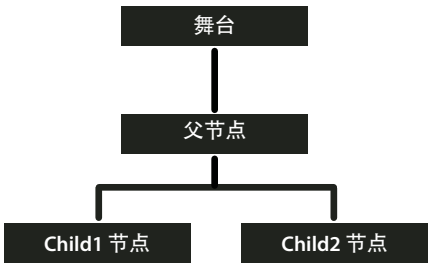
只要发生事件，Flash Player 或 AIR 就会调度事件对象。如果事件目标不在显示列表中，则 Flash Player 或 AIR 将事件对象直接调度到事件目标。例如，Flash Player 将 progress 事件对象直接调度到 URLStream 对象。但是，如果事件目标在显示列表中，则 Flash Player 将事件对象调度到显示列表，事件对象将在显示列表中穿行，直到到达事件目标。

“事件流”说明事件对象如何在显示列表中穿行。显示列表以一种可以用树结构来描述的层次形式进行组织。位于显示列表层次顶部的是舞台，它是一个特殊的显示对象容器，用作显示列表的根。舞台由 flash.display.Stage 类表示，且只能通过显示对象访问。每个显示对象都有一个名为 stage 的属性，该属性表示应用程序的舞台。

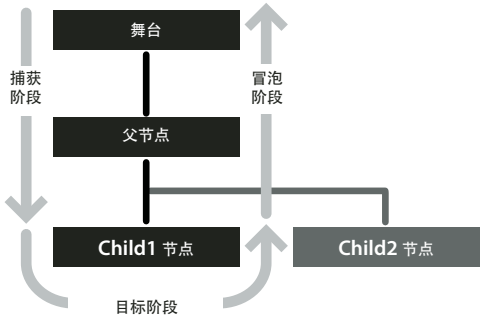
当 Flash Player 或 AIR 为显示列表相关的事件调度事件对象时，该事件对象将进行一次从舞台到“目标节点”的往返行程。DOM 事件规范将目标节点定义为代表事件目标的节点。也就是说，目标节点是发生了事件的显示列表对象。例如，如果用户单击名为 child1 的显示列表对象， Flash Player 或 AIR 将使用 child1 作为目标节点来调度事件对象。

从概念上来说，事件流分为三部分。第一部分称为捕获阶段，该阶段包括从舞台到目标节点的父节点范围内的所有节点。第二部分称为目标阶段，该阶段仅包括目标节点。第三部分称为冒泡阶段。冒泡阶段包括从目标节点的父节点返回到舞台的行程中遇到的节点。

如果将显示列表想像为一个垂直的层次，其中舞台位于顶层 （如下图显示），那么这些阶段的名称就更容易理解了：



如果用户单击 Child1 Node，Flash Player 或 AIR 将向事件流调度一个事件对象。如下面的图像所示，对象的行程从舞台开始，向下移到父节点，然后移到 Child1 节点，再“冒泡”返回到舞台（即在行程中重新经过父节点，再返回到舞台）。



在此示例中，捕获阶段在首次向下行程中包括舞台和父节点。目标阶段包括在 Child1 花费的时间。冒泡阶段包括在向上返回到根节点的行程中遇到的父节点和舞台。

事件流使现在的事件处理系统比 ActionScript 程序员以前使用的事件处理系统功能更为强大。早期版本的 ActionScript 中没有事件流，这意味着事件侦听器只能添加到生成事件的对象。在 ActionScript 3.0 中，您不但可以将事件侦听器添加到目标节点，还可以将它们添加到事件流中的任何节点。

当用户界面组件包含多个对象时，沿事件流添加事件侦听器的功能十分有用。例如，按钮对象通常包含一个用作按钮标签的文本对象。如果无法将侦听器添加到事件流，您将必须将侦听器添加到按钮对象和文本对象，以确保您收到有关在按钮上任何位置发生的单击事件的通知。而事件流的存在则使您可以将一个事件侦听器放在按钮对象上，以处理文本对象上发生的单击事件或按钮对象上未被文本对象遮住的区域上发生的单击事件。

不过，并非每个事件对象都参与事件流的所有三个阶段。某些类型的事件（例如 enterFrame 和 init 类型的事件）会直接调度到目标节点，并不参与捕获阶段和冒泡阶段。其他事件可能以不在显示列表中的对象为目标，例如调度到 Socket 类的实例的事件。这些事件对象也将直接流至目标对象，而不参与捕获和冒泡阶段。

要查明特定事件类型的行为，可以查看 API 文档或检查事件对象的属性。下面的部分介绍了如何检查事件对象的属性。

事件对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在新的事件处理系统中，事件对象有两个主要用途。首先，事件对象通过将特定事件的有关信息存储在一组属性中来表示实际事件。其次，事件对象包含一组方法，可用于操作事件对象和影响事件处理系统的行为。

为方便对这些属性和方法的访问，Flash Player API 定义了一个 Event 类，作为所有事件对象的基类。Event 类定义所有事件对象共有的一组基本属性和方法。

本部分首先讨论 Event 类属性，然后介绍 Event 类方法，最后说明 Event 类的子类存在的意义。

了解 Event 类的属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Event 类定义许多只读属性和常量，以提供有关事件对象的重要信息。以下内容尤其重要：

- 事件对象类型由常量表示，并存储在 Event.type 属性中。
- 事件的默认行为是否可以被阻止由布尔值表示，并存储在 Event.cancelable 属性中。

- 事件流信息包含在其余属性中。

事件对象类型

每个事件对象都有关联的事件类型。数据类型以字符串值的形式存储在 Event.type 属性中。知道事件对象的类型是非常有用的，这样您的代码就可以区分不同类型的对象。例如，下面的代码指定 clickHandler() 侦听器函数应响应传递给 myDisplayObject 的任何鼠标单击事件对象。

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

大约有 20 多种事件类型与 Event 类自身关联并由 Event 类常量表示，其中某些数据类型显示在摘自 Event 类定义的以下代码中：

```
package flash.events
{
    public class Event
    {
        // class constants
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String= "added";
        // remaining constants omitted for brevity
    }
}
```

这些常量提供了引用特定事件类型的简便方法。您应使用这些常量而不是它们所代表的字符串。如果您的代码中拼错了某个常量名称，编译器将捕获到该错误，但如果您改为使用字符串，则编译时可能不会出现拼写错误，这可能导致难以调试的意外行为。例如，添加事件侦听器时，使用以下代码：

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

而不是使用：

```
myDisplayObject.addEventListener("click", clickHandler);
```

默认行为信息

代码可通过访问 cancelable 属性来检查是否可以阻止任何给定事件对象的默认行为。cancelable 属性包含一个布尔值，用于指示是否可以阻止默认行为。您可以使用 preventDefault() 方法阻止或取消与少量事件关联的默认行为。有关详细信息，请参阅第 112 页的 “ 了解 Event 类的方法 ” 下的 “ 取消默认事件行为 ”。

事件流信息

其余 Event 类属性包含有关事件对象及其与事件流的关系的重要信息，如以下列表所述：

- bubbles 属性包含有关事件流中事件对象参与的部分的信息。
- eventPhase 属性指示事件流中的当前阶段。
- target 属性存储对事件目标的引用。
- currentTarget 属性存储对当前正在处理事件对象的显示列表对象的引用。

bubbles 属性

如果事件对象参与事件流的冒泡阶段，则将该事件称为 “ 冒泡 ”，这指的是从目标节点将事件对象往回传递，经过目标节点的父节点，直到到达舞台。 Event.bubbles 属性存储一个布尔值，用于指示事件对象是否参与冒泡阶段。由于冒泡的所有事件还参与捕获和目标阶段，因此这些事件参与事件流的所有三个阶段。如果值为 true，则事件对象参与所有三个阶段。如果值为 false，则事件对象不参与冒泡阶段。

eventPhase 属性

您可以通过调查任何事件对象的 eventPhase 属性来确定事件阶段。 eventPhase 属性包含一个无符号整数值，该值代表三个事件流阶段中的一个阶段。 Flash Player API 定义了单独的 EventPhase 类，该类包含三个对应于三个无符号整数值的常量，如以下摘录代码中所示：

```
package flash.events
{
    public final class EventPhase
    {
        public static const CAPTURING_PHASE:uint = 1;
        public static const AT_TARGET:uint = 2;
        public static const BUBBLING_PHASE:uint= 3;
    }
}
```

这些常量对应于 eventPhase 属性的三个有效值。使用这些常量可以使您的代码可读性更好。例如，如果要确保仅当事件目标在目标阶段中时才调用名为 myFunc() 的函数，您可以使用以下代码来测试此条件：

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

target 属性

target 属性包含对作为事件目标的对象的引用。在某些情况下，这很简单，例如当麦克风变为活动状态时，事件对象的目标是 Microphone 对象。但是，如果目标在显示列表中，就必须考虑显示列表层次。例如，如果用户在包括重叠的显示列表对象的某一点输入一个鼠标单击，则 Flash Player 和 AIR 始终会选择距离舞台层次最深的对象作为事件目标。

对于复杂的 SWF 文件，特别是那些通常使用更小的子对象来修饰按钮的 SWF 文件，target 属性可能并不常用，因为它通常指向按钮的子对象，而不是按钮。在这些情况下，常见的做法是将事件侦听器添加到按钮并使用 currentTarget 属性，因为该属性指向按钮，而 target 属性可能指向按钮的子对象。

currentTarget 属性

currentTarget 属性包含对当前正在处理事件对象的对象的引用。您并不知道哪个节点当前正在处理您要检查的事件对象，虽然这似乎很奇怪，但请记住，您可以向该事件对象的事件流中的任何显示对象添加侦听器函数，并且可以将侦听器函数放在任何位置。而且，可以将相同的侦听器函数添加到不同的显示对象。随着项目大小和复杂性的增加， currentTarget 属性会变得越来越有用。

了解 Event 类的方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有三种类别的 Event 类方法：

- 实用程序方法：可以创建事件对象的副本或将其转换为字符串
- 事件流方法：用于从事件流中删除事件对象
- 默认行为方法：可阻止默认行为或检查是否已阻止默认行为

Event 类实用程序方法

Event 类中有两个实用程序方法。clone() 方法用于创建事件对象的副本。toString() 方法用于生成事件对象属性的字符串表示形式以及它们的值。这两个方法都由事件模型系统在内部使用，但对开发人员公开以用于一般用途。

对于创建 Event 类的子类的高级开发人员来说，必须覆盖和实现两个实用程序方法的版本，以确保事件子类正常使用。

停止事件流

可以调用 Event.stopPropagation() 方法或 Event.stopImmediatePropagation() 方法来阻止在事件流中继续执行事件对象。这两种方法几乎相同，唯一的不同之处在于是否允许执行当前节点的其他事件侦听器：

- Event.stopPropagation() 方法可阻止事件对象移到下一个节点，但只有在允许执行当前节点上的任何其他事件侦听器之后才起作用。
- Event.stopImmediatePropagation() 方法也可阻止事件对象移到下一个节点，但不允许执行当前节点上的任何其他事件侦听器。

调用其中任何一个方法对是否发生与事件关联的默认行为没有影响。使用 Event 类的默认行为方法可以阻止默认行为。

取消事件默认行为

与取消默认行为有关的两个方法是 preventDefault() 方法和 isDefaultPrevented() 方法。调用 preventDefault() 方法可取消与事件关联的默认行为。若要查看是否已针对事件对象调用了 preventDefault()，请调用 isDefaultPrevented() 方法；如果已经调用，该方法将返回值 true，否则返回值 false。

preventDefault() 方法仅在可以取消事件的默认行为时才起作用。可通过参考该事件类型的 API 文档或使用 ActionScript 检查事件对象的 cancelable 属性来确定是否属于这种情况。

取消默认行为对事件对象通过事件流的进度没有影响。使用 Event 类的事件流方法可以从事件流中删除事件对象。

Event 类的子类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

对于很多事件， Event 类中定义的一组公共属性已经足够了。但是， Event 类中可用的属性无法捕获其他事件具有的独特的特性。 ActionScript 3.0 为这些事件定义了 Event 类的几个子类。

每个子类提供了对该类别的事件唯一的附加属性和事件类型。例如，与鼠标输入相关的事件具有若干独特的特性，无法被 Event 类中定义的属性捕获。MouseEvent 类添加了 10 个属性，扩展了 Event 类。这 10 个属性包含诸如鼠标事件的位置和在鼠标事件过程中是否按下了特定键等信息。

Event 子类还包含代表与子类关联的事件类型的常量。例如， MouseEvent 类定义几种鼠标事件类型的内容，包括 click、doubleClick、 mouseDown 和 mouseUp 事件类型。

正如第 110 页的 “ 事件对象 ” 下的 “Event 类实用程序方法 ” 一节所述，创建 Event 子类时，必须覆盖 clone() 和 toString() 方法以实现特定于该子类的功能。

事件侦听器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

事件侦听器也称为事件处理函数，是 Flash Player 和 AIR 为响应特定事件而执行的函数。添加事件侦听器的过程分为两步。首先，为 Flash Player 或 AIR 创建一个为响应事件而执行的函数或类方法。这有时称为侦听器函数或事件处理函数。然后，使用 addEventListener() 方法，在事件的目标或位于适当事件流上的任何显示列表对象中注册侦听器函数。

创建侦听器函数

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

创建侦听器函数是 ActionScript 3.0 事件模型与 DOM 事件模型不同的一个方面。在 DOM 事件模型中，事件侦听器和侦听器函数之间有一个明显的不同：即事件侦听器是实现 EventListener 接口的类的实例，而侦听器是该类的名为 handleEvent() 的方法。在 DOM 事件模型中，您注册的是包含侦听器函数的类实例，而不是实际的侦听器函数。

在 ActionScript 3.0 事件模型中，事件侦听器和侦听器函数之间没有区别。ActionScript 3.0 没有 EventListener 接口，侦听器函数可以在类外部定义，也可以定义为类的一部分。此外，无需将侦听器函数命名为 handleEvent() — 可以将它们命名为任何有效的标识符。在 ActionScript 3.0 中，您注册的是实际侦听器函数的名称。

在类外部定义的侦听器函数

以下代码创建一个显示红色正方形的简单 SWF 文件。名为 clickHandler() 的侦听器函数 （不是类的一部分）侦听红色正方形上的鼠标单击事件。

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

当用户通过单击正方形与生成的 SWF 文件交互时， Flash Player 或 AIR 生成以下跟踪输出：

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

请注意，事件对象作为参数传递到 clickHandler()。这就允许您的侦听器函数检查事件对象。在该示例中，使用事件对象的 type 属性来确定该事件是否为单击事件。

该示例还检查 this 关键字的值。在本例中， this 表示全局对象，这是因为函数是在任何自定义类或对象外部定义的。

定义为类方法的侦听器函数

下面的示例与前面定义 ClickExample 类的示例相同，只是将 clickHandler() 函数定义为 ChildSprite 类的方法：

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
    }
}
```

当用户通过单击红色正方形与生成的 SWF 文件交互时， Flash Player 或 AIR 生成以下跟踪输出：

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

请注意， this 关键字引用名为 child 的 ChildSprite 实例。这是与 ActionScript 2.0 相比行为方面的一个变化。如果您使用过 ActionScript 2.0 中的组件，您可能会记得，将类方法传递给 UIEventDispatcher.addEventListener() 时，方法的作用域被绑定到广播事件的组件，而不是在其中定义侦听器方法的类。也就是说，如果在 ActionScript 2.0 中使用该技术， this 关键字将引用广播事件的组件，而不是 ChildSprite 实例。

这对于某些程序员来说是一个重要问题，因为这意味着他们无法访问包含侦听器方法的类的其他方法和属性。过去，ActionScript 2.0 程序员可以使用 mx.util.Delegate 类更改侦听器方法的作用域以解决该问题。不过，现在已不再需要这样做了，因为 ActionScript 3.0 在调用 addEventListener() 时会创建一个绑定方法。这样， this 关键字引用名为 child 的 ChildSprite 实例，且程序员可以访问 ChildSprite 类的其他方法和属性。

不应使用的事件侦听器

还有第三种技术可用于创建一个通用对象，该对象具有指向动态分配的侦听器函数的属性，但不推荐使用该技术。在此讨论这种技术是因为它在 ActionScript 2.0 中经常使用，但在 ActionScript 3.0 中不应使用。建议不要使用此项技术，因为 this 关键字将引用全局对象，而不引用您的侦听器对象。

下面的示例与前面的 ClickExample 类示例相同，只是将侦听器函数定义为名为 myListenerObj 的通用对象的一部分：

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

跟踪的结果将类似如下：

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

您可能以为 this 引用 myListenerObj，且跟踪输出应为 [object Object]，但实际上它引用的是全局对象。将动态属性名称作为参数传递给 addEventListener() 时，Flash Player 或 AIR 无法创建绑定方法。这是因为作为 listener 参数传递的只不过是侦听器函数的内存地址， Flash Player 和 AIR 无法将该内存地址与 myListenerObj 实例关联起来。

管理事件侦听器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 IEventDispatcher 接口的方法来管理侦听器函数。IEventDispatcher 接口是 ActionScript 3.0 版本的 DOM 事件模型的 EventTarget 接口。虽然名称 IEventDispatcher 似乎暗示着其主要用途是发送（调度）事件对象，但该类的方法实际上更多用于注册、检查和删除事件侦听器。 IEventDispatcher 接口定义五个方法，如以下代码中所示：

```
package flash.events
{
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                        listener:Object,
                        useCapture:Boolean=false,
                        priority:Integer=0,
                        useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

Flash Player API 使用 EventDispatcher 类来实现 IEventDispatcher 接口，该类用作可以是事件目标或事件流一部分的所有类的基类。例如， DisplayObject 类继承自 EventDispatcher 类。这意味着，显示列表中的所有对象都可以访问 IEventDispatcher 接口的方法。

添加事件侦听器

addEventListener() 方法是 IEventDispatcher 接口的主要函数。使用它来注册侦听器函数。两个必需的参数是 type 和 listener。type 参数用于指定事件的类型。 listener 参数用于指定发生事件时将执行的侦听器函数。 listener 参数可以是对函数或类方法的引用。

指定 listener 参数时，不要使用括号。例如，在下面的 addEventListener() 方法调用中，指定 clickHandler() 函数时没有使用括号：

```
addEventListener(MouseEvent.CLICK, clickHandler)
```

通过使用 addEventListener() 方法的 useCapture 参数，可以控制侦听器将处于活动状态的事件流阶段。如果 useCapture 设置为 true，侦听器将在事件流的捕获阶段成为活动状态。如果 useCapture 设置为 false，侦听器将在事件流的目标阶段和冒泡阶段处于活动状态。若要在事件流的所有阶段侦听某一事件，您必须调用 addEventListener() 两次，第一次调用时将 useCapture 设置为 true，第二次调用时将 useCapture 设置为 false。

addEventListener() 方法的 priority 参数并不是 DOM Level 3 事件模型的正式部分。 ActionScript 3.0 中包括它是为了在组织事件侦听器时提供更大的灵活性。调用 addEventListener() 时，可以将一个整数值作为 priority 参数传递，以设置该事件侦听器的优先级。默认值为 0，但您可以将它设置为负整数值或正整数值。将优先执行此数字较大的事件侦听器。对于具有相同优先级的事件侦听器，则按它们的添加顺序执行，因此将优先执行较早添加的侦听器。

可以使用 useWeakReference 参数来指定对侦听器函数的引用是弱引用还是正常引用。通过将此参数设置为 true，可避免侦听器函数在不再需要时仍然存在于内存中的情况。Flash Player 和 AIR 使用一项称为 “ 垃圾回收 ” 的技术从内存中清除不再使用的对象。如果不存在对某个对象的引用，则该对象被视为不再使用。垃圾回收器不考虑弱引用，这意味着如果侦听器函数仅具有指向它的弱引用，则符合垃圾回收条件。

删除事件侦听器

可以使用 removeEventListener() 方法删除不再需要的事件侦听器。建议删除将不再使用的所有侦听器。必需的参数包括 eventName 和 listener 参数，这些参数与 addEventListener() 方法的必需参数相同。回想一下，您可以通过调用 addEventListener() 两次 （第一次调用时将 useCapture 设置为 true，第二次调用时将其设置为 false），在所有事件阶段侦听事件。若要删除这两个事件侦听器，您需要调用 removeEventListener() 两次，第一次调用时将 useCapture 设置为 true，第二次调用时将其设置为 false。

调度事件

高级程序员可以使用 dispatchEvent() 方法将自定义事件对象调度到事件流。该方法唯一接受的参数是对事件对象的引用，此事件对象必须是 Event 类的实例或子类。调度后，事件对象的 target 属性将设置为对其调用了 dispatchEvent() 的对象。

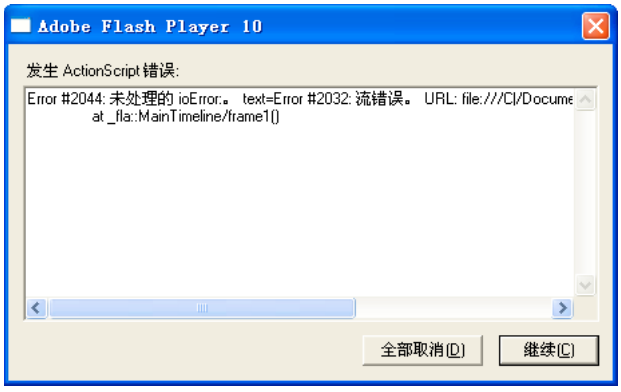
检查有无现有的事件侦听器

IEventDispatcher 接口的最后两个方法提供有关是否存在事件侦听器的有用信息。如果在特定显示列表对象上发现特定事件类型的事件侦听器， hasEventListener() 方法将返回 true。如果发现特定显示列表对象的侦听器， willTrigger() 方法也会返回 true。但 willTrigger() 不但检查该显示对象上的侦听器，还会检查该显示列表对象在事件流所有阶段中的所有始祖上的侦听器。

没有侦听器的错误事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 中处理错误的主要机制是异常而不是事件，但对于异步操作 （例如加载文件），异常处理不起作用。如果在这样的异步操作中发生错误，Flash Player 和 AIR 会调度一个错误事件对象。如果不为错误事件创建侦听器，Flash Player 和 AIR 的调试器版本将打开一个对话框，其中包含有关该错误的信息。例如，调试版的 Flash Player 会在应用程序试图从无效 URL 加载文件时生成以下对话框来描述错误：



大多数错误事件基于 ErrorEvent 类，而且同样具有一个名为 text 的属性，它用于存储 Flash Player 或 AIR 显示的错误消息。两个异常是 StatusEvent 和 NetStatusEvent 类。这两个类都具有一个 level 属性 （StatusEvent.level 和 NetStatusEvent.info.level）。当 level 属性的值为 "error" 时，这些事件类型被视为错误事件。

错误事件将不会导致 SWF 文件停止运行。它仅在浏览器插件和独立播放器的调试器版本上显示为对话框，在创作播放器的输出面板中显示为消息，在 Adobe Flash Builder 的日志文件中显示为条目。在 Flash Player 或 AIR 的发行版中根本不会显示。

事件处理示例： 闹钟

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Alarm Clock 示例包含一个时钟，供用户指定闹铃响起的时间，还包含一个在该时间显示的消息。Alarm Clock 示例是在第 1 页的 “ 使用日期和时间 ” 中的 SimpleClock 应用程序的基础上构建的。Alarm Clock 说明了在 ActionScript 3.0 中使用事件的几个方面，其中包括：

- 侦听和响应事件
- 向侦听器通知事件
- 创建自定义事件类型

若要获取此范例的 Flash Professional 应用程序文件，请参阅 http://www.adobe.com/go/learn_programmingAS3samples_flash_cn。若要获取此范例的 Flex 应用程序文件，请参阅 http://www.adobe.com/go/as3examples_cn。可以在 Samples/AlarmClock 文件夹中找到 Alarm Clock 应用程序文件。该应用程序包括以下文件：

文件	说明
AlarmClockApp.mxml 或 AlarmClockApp.fla	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/clock/AlarmClock.as	一个扩展 SimpleClock 类的类，添加了闹钟功能。
com/example/programmingas3/clock/AlarmEvent.as	自定义事件类 （flash.events.Event 的子类），用作 AlarmClock 类的 alarm 事件的事件对象。
com/example/programmingas3/clock/AnalogClockFace.as	绘制一个圆的时钟形状以及基于时间的时针、分针和秒针 （如 SimpleClock 示例中所述）。
com/example/programmingas3/clock/SimpleClock.as	具有简单走时功能的时钟界面组件 （如 SimpleClock 示例中所述）。

Alarm Clock 概述

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在此示例中，时钟的主要功能 （包括跟踪时间和显示时钟形状）重复使用 SimpleClock 应用程序代码，相关介绍请参阅第 5 页的 “ 日期和时间示例：简单模拟时钟 ”。 AlarmClock 类添加了闹钟所需的功能 （包括设置闹铃时间和在闹铃 “ 响起 ” 时显示通知），从而扩展了该示例中的 SimpleClock 类。

在发生事情时提供通知，是创建事件的目的。 AlarmClock 类公开 Alarm 事件，其他对象可侦听该事件以执行所需操作。此外， AlarmClock 类使用 Timer 类的实例来确定何时触发闹铃。和 AlarmClock 类一样， Timer 类提供一个事件，用于在经过特定时间时通知其他对象 （在本例中为 AlarmClock 实例）。就像大多数 ActionScript 应用程序一样，事件构成了 Alarm Clock 范例应用程序功能的重要部分。

触发闹铃

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如前所述， AlarmClock 类实际提供的唯一功能与设置和触发闹铃有关。内置的 Timer 类 (flash.utils.Timer) 为开发人员提供了定义要在指定时间之后执行的代码的方法。 AlarmClock 类使用 Timer 实例来确定何时触发闹铃。

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * The Timer that will be used for the alarm.
 */
public var alarmTimer:Timer;
...
/**
 * Instantiates a new AlarmClock of a given size.
 */
public override function initClock(faceSize:Number = 200):void
{
    super.initClock(faceSize);
    alarmTimer = new Timer(0, 1);
    alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

AlarmClock 类中定义的 Timer 实例被命名为 alarmTimer。 initClock() 方法执行 AlarmClock 实例的所需设置操作，使用 alarmTimer 变量执行两个任务。首先，使用指示 Timer 实例等待 0 毫秒且仅触发其 timer 事件一次的参数实例化变量。实例化 alarmTimer 后，代码调用变量的 addEventListener() 方法，指示它要监听该变量的 timer 事件。 Timer 实例的工作方式是：在经过指定时间后调度其 timer 事件。 AlarmClock 类需要了解何时调度 timer 事件，以便触发自己的闹铃。通过调用 addEventListener()， AlarmClock 代码将自身作为侦听器在 alarmTimer 中进行注册。两个参数指示 AlarmClock 类要侦听 timer 事件 （由常量 TimerEvent.TIMER 指示），并且当事件发生时，应调用 AlarmClock 类的 onAlarm() 方法以响应事件。

为了实际设置闹铃，代码调用了 AlarmClock 类的 setAlarm() 方法，如下所示：

```
/**
 * Sets the time at which the alarm should go off.
 * @param hour The hour portion of the alarm time.
 * @param minutes The minutes portion of the alarm time.
 * @param message The message to display when the alarm goes off.
 * @return The time at which the alarm will go off.
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = "Alarm!"):Date
{
    this.alarmMessage = message;
    var now:Date = new Date();
    // Create this time on today's date.
    alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

    // Determine if the specified time has already passed today.
    if (alarmTime <= now)
    {
        alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
    }

    // Stop the alarm timer if it's currently set.
    alarmTimer.reset();
    // Calculate how many milliseconds should pass before the alarm should
    // go off (the difference between the alarm time and now) and set that
    // value as the delay for the alarm timer.
    alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
    alarmTimer.start();

    return alarmTime;
}
```

此方法执行了几项操作，包括存储闹铃消息和创建一个 Date 对象 (alarmTime)，该对象表示触发闹铃的实际时间。在该方法的最后几行中，与当前讨论最相关的操作是设置和激活了 alarmTimer 变量的计时器。首先，调用其 reset() 方法，如果计时器已运行，则将其停止并进行重置。接下来，从 alarmTime 变量值中减去当前时间 （由 now 变量表示），以确定需要经过多少毫秒后才会触发闹铃。Timer 类并不会在某个绝对时间触发其 timer 事件，因此，分配给 alarmTimer 的 delay 属性的是该相对时间差。最后，调用 start() 方法以实际启动计时器。

一旦经过指定时间，alarmTimer 将调度 timer 事件。由于 AlarmClock 类已将其 onAlarm() 方法注册为该事件的侦听器，因此发生 timer 事件时，将调用 onAlarm()。

```
/**
 * Called when the timer event is dispatched.
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

注册为事件侦听器的方法必须使用适当的签名 （即，方法的参数集和返回类型）来定义。若要侦听 Timer 类的 timer 事件，方法必须定义一个数据类型为 TimerEvent (flash.events.TimerEvent) 的参数，该参数是 Event 类的子类。当 Timer 实例调用其事件侦听器时，会传递一个 TimerEvent 实例作为事件对象。

向其他代码通知闹铃

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

和 Timer 类一样， AlarmClock 类提供了一个事件，以允许其他代码在闹铃响起时收到通知。对于类而言，要使用内置于 ActionScript 中的事件处理框架，必须实现 flash.events.IEventDispatcher 接口。通常，这是通过扩展 flash.events.EventDispatcher 类（提供 IEventDispatcher 的标准实现）或 EventDispatcher 的某个子类来完成的。如前所述，AlarmClock 类扩展了 SimpleClock 类，SimpleClock 类（通过继承链）扩展了 EventDispatcher 类。所有这些意味着 AlarmClock 类已经具有内置功能以提供自己的事件。

其他代码可通过调用 AlarmClock 从 EventDispatcher 继承的 addEventListener() 方法进行注册，以获得 AlarmClock 类的 alarm 事件的通知。当 AlarmClock 实例准备通知其他代码已引发其 alarm 事件时，它会调用 dispatchEvent() 方法进行通知，该方法同样是从 EventDispatcher 继承的。

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

这些代码行摘自 AlarmClock 类的 onAlarm() 方法 （前面完整介绍过）。调用 AlarmClock 实例的 dispatchEvent() 方法，该方法接下来通知所有注册的侦听器：已触发 AlarmClock 实例的 alarm 事件。传递给 dispatchEvent() 的参数是要一直传递到侦听器方法的事件对象。在本例中，它是 AlarmEvent 类的实例，即为本示例专门创建的 Event 子类。

提供自定义 Alarm 事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

所有事件侦听器都接收一个事件对象参数，该参数提供有关要触发的特定事件的信息。在许多情况下，事件对象是 Event 类的实例。但在某些情况下，向事件侦听器提供其他信息很有用。实现该目的的一个常用方法是定义一个新类 （Event 类的子类），并将该类的实例用作事件对象。在本示例中，当调度 AlarmClock 类的 alarm 事件时，会将一个 AlarmEvent 实例用作事件对象。在此介绍的 AlarmEvent 类提供有关 alarm 事件的其他信息，具体来说是闹铃消息：

```
import flash.events.Event;

/**
 * This custom Event class adds a message property to a basic Event.
 */
public class AlarmEvent extends Event
{
    /**
     * The name of the new AlarmEvent type.
     */
    public static const ALARM:String = "alarm";

    /**
     * A text message that can be passed to an event handler
     * with this event object.
     */
    public var message:String;

    /**
     *Constructor.
     *@param message The text to display when the alarm goes off.
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

要创建自定义事件对象类，最好的方法是定义一个扩展 Event 类的类，如前面的示例中所示。为了补充继承的功能，AlarmEvent 类定义一个属性 message，该属性包含与事件关联的闹铃消息的文本； message 是作为 AlarmEvent 构造函数中的参数传入的。 AlarmEvent 类还定义常量 ALARM，当调用 AlarmClock 类的 addEventListener() 方法时，该常量可用于引用特定事件 (alarm)。

除了添加自定义功能外，作为 ActionScript 事件处理框架的一部分，每个 Event 子类还必须覆盖继承的 clone() 方法。 Event 子类还可以选择性地覆盖继承的 toString() 方法，以便在调用 toString() 方法时返回的值中包括自定义事件的属性。

```
/**
 * Creates and returns a copy of the current instance.
 * @return A copy of the current instance.
 */
public override function clone():Event
{
    return new AlarmEvent(message);
}

/**
 * Returns a String containing all the properties of the current
 * instance.
 * @return A string representation of the current instance.
 */
public override function toString():String
{
    return formatToString("AlarmEvent", "type", "bubbles", "cancelable", "eventPhase", "message");
}
```

被覆盖的 clone() 方法需要返回自定义 Event 子类的新实例，并且设置了所有自定义属性以匹配当前实例。在被覆盖的 toString() 方法中，实用程序方法 formatToString() （从 Event 继承）用于提供一个字符串，包括自定义类型的名称以及所有属性的名称和值。

第 9 章：使用应用程序域

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ApplicationDomain 类的用途是存储 ActionScript 3.0 定义表。 SWF 文件中的所有代码被定义为存在于应用程序域中。可以使用应用程序域划分位于同一个安全域中的类。这允许同一个类存在多个定义，并且还允许子级重用父级定义。

在使用 Loader 类 API 加载用 ActionScript 3.0 编写的外部 SWF 文件时，可以使用应用程序域。（请注意，在加载图像或用 ActionScript 1.0 或 ActionScript 2.0 编写的 SWF 文件时不能使用应用程序域。）包含在已加载类中的所有 ActionScript 3.0 定义都存储在应用程序域中。加载 SWF 文件时，通过将 LoaderContext 对象的 applicationDomain 参数设置为 ApplicationDomain.currentDomain，可以指定文件包含在 Loader 对象所在的相同应用程序域中。通过将加载的 SWF 文件放在同一个应用程序域中，可以直接访问它的类。如果加载的 SWF 文件包含嵌入的媒体 （可通过其关联的类名称访问），或者您要访问加载的 SWF 文件的方法，则这种方式会很有用。

以下示例假定可以访问定义了名为 welcome() 的公共方法的单独 Greeter.swf 文件。

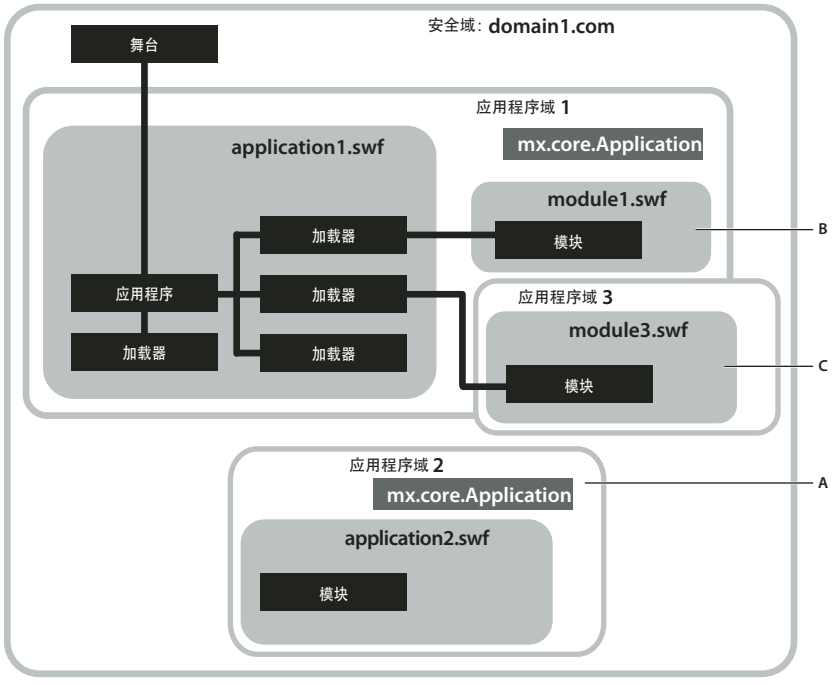
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            var myGreeter:Class = ApplicationDomain.currentDomain.getDefinition("Greeter") as Class;
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

另请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考的 ApplicationDomain 类示例。

使用应用程序域时，还要记住以下几点：

- SWF 文件中的所有代码被定义为存在于应用程序域中。主应用程序在 “ 当前域 ” 中运行。“ 系统域 ” 中包含所有应用程序域（包括当前域），也就是包含所有 Flash Player 类。
- 所有应用程序域 （除系统域外）都有关联的父域。主应用程序的应用程序域的父域是系统域。已加载的类仅在其父级中没有相关定义时才进行定义。不能用较新的定义覆盖已加载类的定义。

下图显示了某个应用程序在单个域 (domain1.com) 中加载多个 SWF 文件的内容。根据加载内容的不同，可以使用不同的应用程序域。紧跟的文本说明用于为应用程序中的每个 SWF 文件设置适当应用程序域的逻辑。



A. 用法 A **B.** 用法 B **C.** 用法 C

主应用程序文件为 application1.swf。它包含从其他 SWF 文件加载内容的 Loader 对象。在此方案下，当前域为 Application domain 1。用法 A、用法 B 和用法 C 说明了为应用程序中的每个 SWF 文件设置适当应用程序域的不同方法。

用法 A 通过创建系统域的子级划分子级 SWF 文件。在示意图中， Application domain 2 创建为系统域的子级。application2.swf 文件在 Application domain 2 中加载，因此其类定义从 application1.swf 中定义的类中划分出来。

此方法的一个用处是使旧版应用程序能够动态加载相同应用程序的更新版本，而不会发生冲突。之所以不发生冲突，是因为尽管使用的是同样的类名称，但它们划分到不同的应用程序域中。

下面的代码创建作为系统域子级的一个应用程序域，并使用该应用程序域开始加载一个 SWF：

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

用法 B: 在当前类定义中添加新的类定义。module1.swf 的应用程序域设置为当前域 (Application domain 1)。这可让您将新的类定义添加到应用程序当前的一组类定义中。这可用于主应用程序的运行时共享库。加载的 SWF 被视为运行时共享库 (RSL)。使用此方法可以在应用程序启动之前使用预加载器加载 RSL。

下面的代码加载一个 SWF，同时将其应用程序域设置为当前域：

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

用法 C: 通过创建当前域的新子域，使用父级的类定义。 module3.swf 的应用程序域是当前域的子级，并且子级使用所有类的父级的版本。此方法的一个用处可能是作为一个使用主应用程序的类型的多屏幕丰富 Internet 应用程序 (RIA) 模块，该模块作为主应用程序的子级加载。如果能够确保所有类始终更新为向后兼容，并且正在加载的应用程序始终比其加载的软件的版本新，则子级将使用父级版本。如果可以确保不继续拥有对子级 SWF 的引用，则拥有了新的应用程序域还使您能够卸载所有的类定义以便于垃圾回收。

此方法使加载的模块可以共享加载者的 singleton 对象和静态类成员。

下面的代码创建当前域的一个新子域，并使用该应用程序域开始加载一个 SWF：

```
var appDomainC:ApplicationDomain = new ApplicationDomain(ApplicationDomain.currentDomain);

var contextC:LoaderContext = new LoaderContext(false, appDomainC);
var loaderC:Loader = new Loader();
loaderC.load(new URLRequest("module3.swf"), contextC);
```

第 10 章：显示编程

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

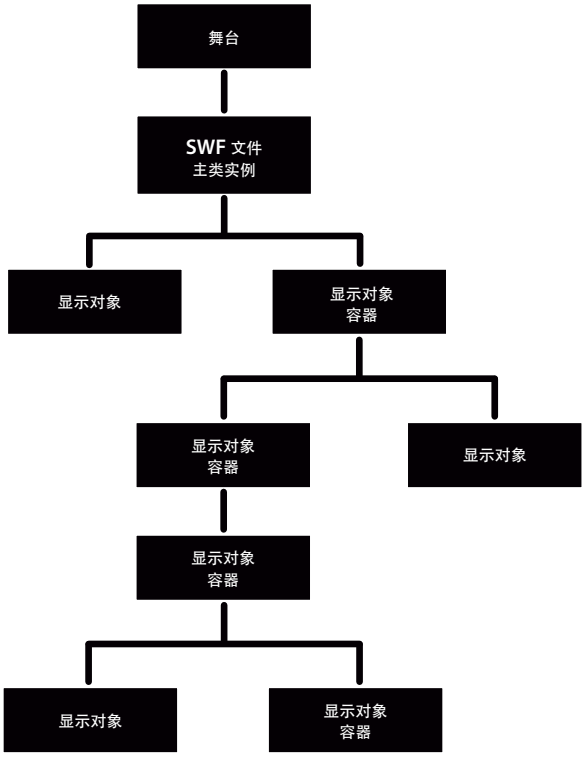
通过使用显示舞台上的显示对象，在 Adobe® ActionScript® 3.0 中对可视元素进行编程。例如，您可以使用 ActionScript 显示编程 API 执行以下操作：添加、移动、删除并对显示对象排序、应用滤镜和蒙版、绘制矢量和位图图形以及执行三维转换。显示编程使用的主类是 flash.display 包的一部分。

注：Adobe® AIR™ 提供了用于呈现和显示 HTML 内容的 HTMLoader 对象。 HTMLLoader 将 HTML DOM 的可视元素呈现为单个显示对象。您不能通过 ActionScript 显示列表层次结构直接访问 DOM 的各个元素。但是，您可使用由 HTMLLoader 提供的单独的 DOM API 来访问这些 DOM 元素。

显示编程的基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 ActionScript 3.0 构建的每个应用程序都有一个由显示对象构成的层次，这个层次称为显示列表，如下所示。显示列表包含应用程序中的所有可视元素。



如图所示，显示元素分为以下一个或多个组：

- Stage

  Stage 是包括显示对象的基础容器。每个应用程序都有一个 Stage 对象，其中包含所有的屏幕显示对象。舞台是顶级容器，位于显示列表层次的顶部：

  每个 SWF 文件都有一个关联的 ActionScript 类，该类称为“SWF 文件的主类”。在 Flash Player 或 Adobe AIR 中打开 SWF 文件时，Flash Player 或 AIR 将调用该类的构造函数，并添加所创建的实例（始终是一种显示对象）作为 Stage 对象的子级。 SWF 文件的主类始终用于扩展 Sprite 类 （有关详细信息，请参阅第 130 页的“ 显示列表方法的优点 ”）。

  可以通过任何 DisplayObject 实例的 stage 属性来访问舞台。有关详细信息，请参阅第 137 页的“ 设置舞台属性 ”。

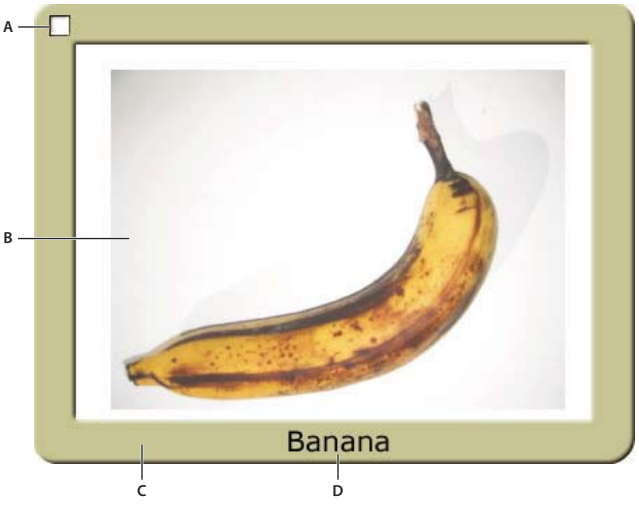
- 显示对象

  在 ActionScript 3.0 中，在应用程序屏幕上出现的所有元素都属于“ 显示对象 ”类型。 flash.display 包中包含一个 DisplayObject 类，该类是一个由许多其他类扩展的基类。这些不同的类表示一些不同类型的显示对象，如矢量形状、影片剪辑和文本字段等。有关这些类的概述，请参阅第 130 页的“ 显示列表方法的优点 ”。

- 显示对象容器

  显示对象容器是一些特殊类型的显示对象，这些显示对象除了有自己的可视表示形式之外，还可以包含也是显示对象的子对象。

  DisplayObjectContainer 类是 DisplayObject 类的子类。 DisplayObjectContainer 对象可以在其“ 子级列表 ”中包含多个显示对象。例如，下图显示一种称为 Sprite 的 DisplayObjectContainer 对象，其中包含各种显示对象：



A. SimpleButton 对象。此类显示对象有不同的“ 弹起 ”、“ 按下 ”和“ 指针经过 ”状态。 **B.** Bitmap 对象。本例中，Bitmap 对象是通过 Loader 对象从外部 JPEG 加载的。 **C.** Shape 对象。“ 图片帧 ”包含一个在 ActionScript 中绘制的圆角矩形。此 Shape 对象有一个应用于它的 Drop Shadow 滤镜。 **D.** 一个 TextField 对象。

  在讨论显示对象的上下文中， DisplayObjectContainer 对象又称为“ 显示对象容器 ”或简称为“ 容器 ”。如前所述，舞台是显示对象容器。

  尽管所有可视显示对象都从 DisplayObject 类继承，但每类显示对象都是 DisplayObject 类的一个特定子类。例如，有 Shape 类或 Video 类的构造函数，但没有 DisplayObject 类的构造函数。

重要概念和术语

以下参考列表包含您在对 ActionScript 图形进行编程时会遇到的重要术语：

Alpha 表示颜色透明度（或者更准确地说，是不透明度）的颜色值。例如，Alpha 通道值为 60% 的颜色只显示其最大强度的 60%，即有 40% 是透明的。

位图图形 在计算机中定义为彩色像素网格 （行和列）的图形。通常，位图图形包括数码照片和类似图像。

混合模式 说明两个重叠的图像的内容应该如何交互的规范。通常，一个图像上面的另一个不透明图像会遮盖住下面的图像，因此根本看不到该图像；但是，不同的混合模式会导致图像颜色以不同方式混合在一起，因此，生成的内容是两个图像的某种组合形式。

显示列表 由 Flash Player 和 AIR 呈现为可见屏幕内容的显示对象的层次。舞台是显示列表的根，附加到舞台或其一个子项上的所有显示对象构成了显示列表 （即使对象实际上并未呈现，例如，对象位于舞台边界以外）。

显示对象 表示 Flash Player 或 AIR 中的某些可视类型的内容的对象。显示列表中只能包含显示对象，所有显示对象类都是 DisplayObject 类的子类。

显示对象容器 一种特殊类型的显示对象，除了 （通常）具有自己的可视表示形式以外，还可以包含子显示对象。

SWF 文件的主类 定义 SWF 文件中最外层显示对象的行为的类，从概念上说，这是 SWF 文件自身的类。例如，对于在 Flash 创作工具中创建的 SWF，主类为文档类。它具有一个包含所有其他时间轴的 “ 主时间轴 ”； SWF 文件的主类是将主时间轴作为其实例的类。

蒙版 一种隐藏视图中图像的特定部分 （或相反，只允许显示图像的特定部分）的技术。遮罩图像部分将变为透明，因此，将显示其下面的内容。此术语与画家所使用的遮蔽胶带非常相似，遮蔽胶带用于防止将颜料喷到某些区域上。

舞台 作为 SWF 中的所有可视内容的库或背景的可视容器。

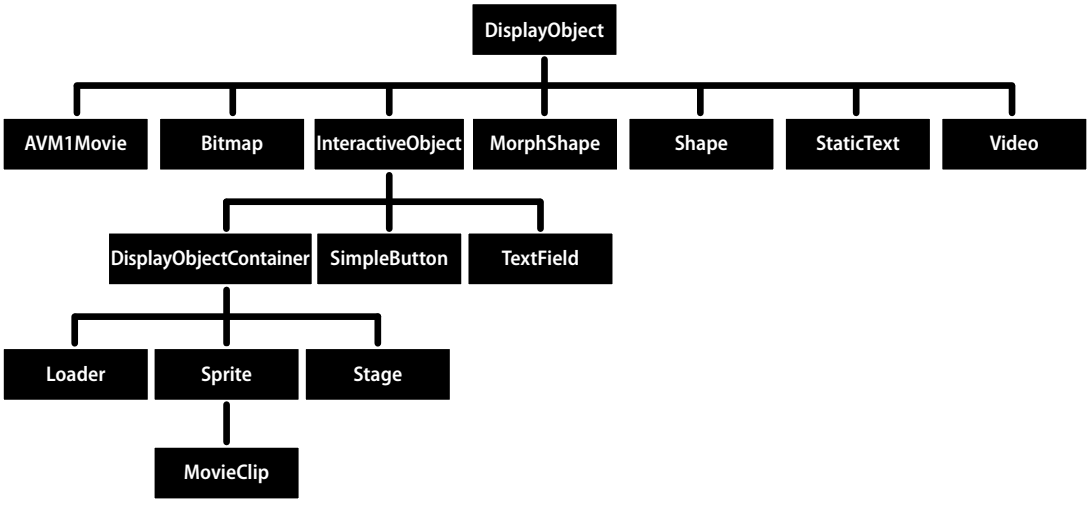
转换 对图形的视觉特性的调整，例如旋转对象、更改其比例、倾斜或扭曲其形状或改变其颜色。

矢量图形 在计算机中定义为使用特定特性 （如粗细、长度、大小、角度以及位置）绘制的线条和形状的图形。

核心显示类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 的 flash.display 包中包括可在 Flash Player 或 AIR 中显示的可视对象的类。下图说明了这些核心显示对象类的子类关系。



该图说明了显示对象类的类继承。请注意，其中某些类，尤其是 StaticText、 TextField 和 Video 类，不在 flash.display 包中，但它们仍然是从 DisplayObject 类继承的。

扩展 DisplayObject 类的所有类都继承该类的方法和属性。有关详细信息，请参阅 第 132 页的 “DisplayObject 类的属性和方法 ”。

可以实例化包含在 flash.display 包中的下列类的对象：

- Bitmap — 使用 Bitmap 类可定义从外部文件加载或通过 ActionScript 呈示的位图对象。可以通过 Loader 类从外部文件加载位图。可以加载 GIF、JPG 或 PNG 文件。还可以创建包含自定义数据的 BitmapData 对象，然后创建使用该数据的 Bitmap 对象。可以使用 BitmapData 类的方法来更改位图，无论这些位图是加载的还是在 ActionScript 中创建的。有关详细信息，请参阅第 165 页的 “ 加载显示对象 ” 和第 203 页的 “ 使用位图 ”。
- Loader — 使用 Loader 类可加载外部资源 （SWF 文件或图形）。有关详细信息，请参阅第 164 页的 “ 动态加载显示内容 ”。
- Shape — 使用 Shape 类可创建矢量图形，如矩形、直线、圆等。有关详细信息，请参阅第 185 页的 “ 使用绘图 API”。
- SimpleButton — SimpleButton 对象是在 Flash 创作工具中创建的按钮元件的 ActionScript 表示形式。SimpleButton 实例有四种按钮状态：弹起、按下、指针经过和点击测试 （响应鼠标和键盘事件的区域）。
- Sprite — Sprite 对象可包含它自己的图形，也可包含子显示对象。（Sprite 类用于扩展 DisplayObjectContainer 类）。有关详细信息，请参阅第 133 页的 “ 使用显示对象容器 ” 和第 185 页的 “ 使用绘图 API”。
- MovieClip — MovieClip 对象是在 Flash 创作工具中创建的影片剪辑元件的 ActionScript 形式。实际上，MovieClip 与 Sprite 对象类似，不同的是它还有一个时间轴。有关详细信息，请参阅第 270 页的 “ 使用影片剪辑 ”。

下列类不在 flash.display 包中，这些类是 DisplayObject 类的子类：

- TextField 类包括在 flash.text 包中，它是用于文本显示和输入的显示对象。有关详细信息，请参阅第 311 页的 “ 文本使用基础知识 ”。
- flash.text.engine 包中包含的 TextLine 类是用于显示由 Flash 文本引擎和 Text Layout Framework 组成的文本行的显示对象。有关详细信息，请参阅第 334 页的 “ 使用 Flash 文本引擎 ” 和 第 360 页的 “ 使用 Text Layout Framework”。
- Video 类包括在 flash.media 包中，它是用于显示视频文件的显示对象。有关详细信息，请参阅第 401 页的 “ 使用视频 ”。

flash.display 包中的下列类用于扩展 DisplayObject 类，但您不能创建这些类的实例。这些类而是用作其他显示对象的父类，因此可将通用功能合并到一个类中。

- AVM1Movie — AVM1Movie 类用于表示在 ActionScript 1.0 和 2.0 中创作的已加载 SWF 文件。
- DisplayObjectContainer — Loader、Stage、Sprite 和 MovieClip 类均用于扩展 DisplayObjectContainer 类。有关详细信息，请参阅第 133 页的 “ 使用显示对象容器 ”。
- InteractiveObject — InteractiveObject 是用于与鼠标和键盘交互的所有对象的基类。 SimpleButton、 TextField、 Loader、 Sprite、 Stage 和 MovieClip 对象是 InteractiveObject 类的所有子类。有关创建鼠标和键盘交互的详细信息，请参阅第 473 页的 “ 用户交互的基础知识 ”。
- MorphShape — 这些对象是在 Flash 创作工具中创建补间形状时创建的。无法使用 ActionScript 实例化这些对象，但可以从显示列表中访问它们。
- Stage — Stage 类用于扩展 DisplayObjectContainer 类。一个应用程序有一个 Stage 实例，该实例位于显示列表层次的顶部。要访问 Stage，请使用任何 DisplayObject 实例的 stage 属性。有关详细信息，请参阅第 137 页的 “ 设置舞台属性 ”。

此外，flash.text 包中的 StaticText 类也用于扩展 DisplayObject 类，但不能在代码中创建它的实例。只能在 Flash 中创建静态文本字段。

以下类不是显示对象或显示对象容器，也不会出现在显示列表中，但是会在舞台上显示图形。这些类将绘制成一个称为视口的矩形，相对于舞台放置。

- StageVideo — StageVideo 类用于在可能时使用硬件加速显示视频内容。此类从 Flash Player 10.2 开始可用。有关详细信息，请参阅第 433 页的 “ 使用 StageVideo 类来实现硬件加速呈现 ”。
- StageWebView — StageWebView 类用于显示 HTML 内容。此类从 AIR 2.5 开始可用。有关详细信息，请参阅 第 880 页的 “StageWebView 对象 ”。

下面的 fl.display 类提供与 flash.display.Loader 和 LoaderInfo 类并行的功能。如果您在 Flash Professional 环境 (CS5.5 或更高版本）中进行开发，请使用这些类而不是其 flash.display 中的对应类。在该环境中，使用这些类可以解决与 TLF 的 RSL 预加载有关的问题。有关详细信息，请参阅第 168 页的 " 使用 ProLoader 和 ProLoaderInfo 类 "。

- fl.display.ProLoader — 与 flash.display.Loader 类似
- fl.display.ProLoaderInfo — 与 flash.display.LoaderInfo 类似

显示列表方法的优点

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中，不同类型的显示对象有不同的类。在 ActionScript 1.0 和 2.0 中，很多相同类型的对象都包括在一个类 （即 MovieClip 类）中。

类的这种个性化处理方式和显示列表的分层次结构具有下列优点：

- 呈示方式更为有效且减少了内存使用
- 改进了深度管理
- 完整遍历显示列表
- 列表外的显示对象
- 更便于创建显示对象的子类

呈示方式更为有效且文件较小

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 1.0 和 2.0 中，只可以在 MovieClip 对象中绘制形状。在 ActionScript 3.0 中，提供了可在其中绘制形状的更简单的显示对象类。由于这些 ActionScript 3.0 显示对象类并不包括 MovieClip 对象中包含的全部方法和属性，因此给内存和处理器资源造成的负担比较小。

例如，每个 MovieClip 对象都包括用于影片剪辑时间轴的属性，而 Shape 对象则不包括。用于管理时间轴的属性会使用大量的内存和处理器资源。在 ActionScript 3.0 中，使用 Shape 对象可提高性能。与更复杂的 MovieClip 对象相比， Shape 对象的开销更少。Flash Player 和 AIR 并不需要管理未使用的 MovieClip 属性，因此提高了速度，还减少了对象使用的内存空间。

改进了深度管理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 1.0 和 2.0 中，通过线性深度管理方案和方法 （如 getNextHighestDepth()）进行深度管理。

在 ActionScript 3.0 中提供了 DisplayObjectContainer 类，该类提供用于管理显示对象深度的更便捷的方法和属性。

在 ActionScript 3.0 中，当您将显示对象移到 DisplayObjectContainer 实例的子级列表中的新位置时，显示对象容器中的其他子级会自动重新定位并在显示对象容器中分配相应的子索引位置。

此外，在 ActionScript 3.0 中，总是可以发现任何显示对象容器中的所有子对象。每个 DisplayObjectContainer 实例都有 numChildren 属性，用于列出显示对象容器中的子级数。由于显示对象容器的子级列表始终是索引列表，因此可以检查列表中从索引位置 0 到最后一个索引位置 (numChildren - 1) 的所有对象。这不适用于 ActionScript 1.0 和 2.0 中 MovieClip 对象的方法和属性。

在 ActionScript 3.0 中，可以按顺序轻松遍历显示列表；显示对象容器子级列表的索引编号没有中断。遍历显示列表和管理对象深度比在 ActionScript 1.0 和 2.0 中更容易。在 ActionScript 1.0 和 2.0 中，影片剪辑可以包含深度顺序有间歇中断的对象，这可能导致难以遍历对象列表。在 ActionScript 3.0 中，显示对象容器的每个子级列表都在内部缓存为一个数组，这样按索引查找的速度就非常快。遍历显示对象容器所有子级的速度也非常快。

在 ActionScript 3.0 中，还可以通过使用 DisplayObjectContainer 类的 getChildByName() 方法来访问显示对象容器中的子级。

完整遍历显示列表

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 1.0 和 2.0 中，无法访问在 Flash 创作工具中绘制的某些对象（如矢量形状）。在 ActionScript 3.0 中，可以访问显示列表中的所有对象，包括使用 ActionScript 创建的对象以及在 Flash 创作工具中创建的所有显示对象。有关详细信息，请参阅第 136 页的 “ 遍历显示列表 ”。

列表外的显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中，可以创建不在可视显示列表中的显示对象。这些对象称为 “ 列表外 ” 显示对象。仅当调用已添加到显示列表中的 DisplayObjectContainer 实例的 addChild() 或 addChildAt() 方法时，才会将显示对象添加到可视显示列表中。

可以使用列表外的显示对象来组合复杂的显示对象，如有多个显示对象容器（包含多个显示对象）的那些对象。通过将显示对象放在列表外，可以组合复杂的对象，而不需要占用处理时间来呈示这些显示对象。然后在需要时可以在显示列表中添加列表外的显示对象。此外，可以随意将显示对象容器的子级移入和移出显示列表以及移到显示列表中的任何需要位置。

更便于创建显示对象的子类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 1.0 和 2.0 中，通常必须在 SWF 文件中添加新的 MovieClip 对象才能创建基本形状或显示位图。在 ActionScript 3.0 中，DisplayObject 类包括许多内置子类，包括 Shape 和 Bitmap。由于 ActionScript 3.0 中的类更专用于特定类型的对象，因此更易于创建内置类的基本子类。

例如，要在 ActionScript 2.0 中绘制一个圆，可以在实例化自定义类的对象时创建用于扩展 MovieClip 类的 CustomCircle 类。但是，该类还另外包括 MovieClip 类中不应用于该类的许多属性和方法（如 totalFrames）。但在 ActionScript 3.0 中，可以创建用于扩展 Shape 对象的 CustomCircle 类，但该类不包括 MovieClip 类中包含的不相关的属性和方法。下面的代码显示了 CustomCircle 类的一个示例：

```
import flash.display.*;

public class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                     yInput:Number,
                                     rInput:Number,
                                     colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

使用显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

现在您已了解了舞台、显示对象、显示对象容器和显示列表的基本概念，本部分将为您提供有关在 ActionScript 3.0 中使用显示对象的一些更具体的信息。

DisplayObject 类的属性和方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

所有显示对象都是 DisplayObject 类的子类，同样它们还会继承 DisplayObject 类的属性和方法。继承的属性是适用于所有显示对象的基本属性。例如，每个显示对象都有 x 属性和 y 属性，用于指定对象在显示对象容器中的位置。

您不能使用 DisplayObject 类构造函数来创建 DisplayObject 实例。必须创建另一种对象 （属于 DisplayObject 类的子类的对象，如 Sprite）才能使用 new 运算符来实例化对象。此外，如果要创建自定义显示对象类，还必须创建具有可用构造函数的其中一个显示对象子类的子类 （如 Shape 类或 Sprite 类）。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 DisplayObject 类说明。

在显示列表中添加显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

实例化显示对象时，在将显示对象实例添加到显示列表上的显示对象容器之前，显示对象不会出现屏幕上 （即在舞台上）。例如，在下面的代码中，如果省略了最后一行代码，则 myText TextField 对象不可见。在最后一行代码中， this 关键字必须引用已添加到显示列表中的显示对象容器。

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

当在舞台上添加任何可视元素时，该元素会成为 Stage 对象的 " 子级 "。应用程序中加载的第一个 SWF 文件（例如，HTML 页中嵌入的文件）会自动添加为 Stage 的子级。它可以是扩展 Sprite 类的任何类型的对象。

不是 使用 ActionScript （例如，通过在 Flex MXML 文件中添加 MXML 标签或在 Flash Professional 的舞台上放置项目）创建的任何显示对象都会添加到显示列表中。尽管没有通过 ActionScript 添加这些显示对象，但仍可通过 ActionScript 访问它们。例如，下面的代码将调整在创作工具中 （不是通过 ActionScript）添加的名为 button1 的对象的宽度：

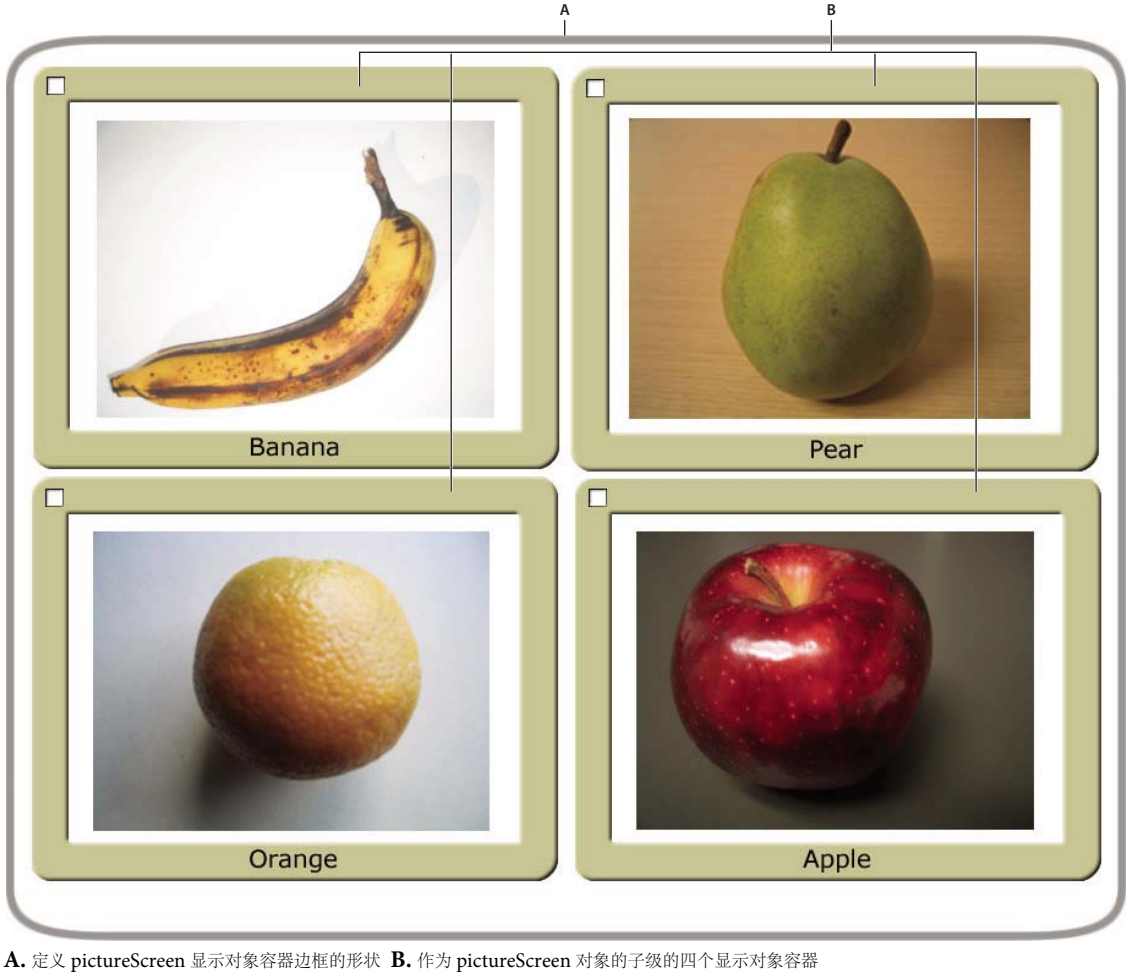
```
button1.width = 200;
```

使用显示对象容器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果从显示列表中删除某个 DisplayObjectContainer 对象，或者以其他某种方式移动该对象或对其进行变形处理，则会同时删除、移动 DisplayObjectContainer 中的每个显示对象或对其进行变形处理。

显示对象容器本身就是一种显示对象，它可以添加到其他显示对象容器中。例如，下图显示的是显示对象容器 pictureScreen，它包含一个轮廓形状和四个其他显示对象容器 （类型为 PictureFrame）：



A. 定义 pictureScreen 显示对象容器边框的形状 **B.** 作为 pictureScreen 对象的子级的四个显示对象容器

要使某一显示对象出现在显示列表中，必须将该显示对象添加到显示列表上的显示对象容器中。使用容器对象的 addChild() 方法或 addChildAt() 方法可执行此操作。例如，如果下面的代码没有最后一行，将不会显示 myTextField 对象：

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

在此代码范例中， this.root 指向包含该代码的 MovieClip 显示对象容器。在实际代码中，可以指定其他容器。

使用 addChildAt() 方法可将子级添加到显示对象容器的子级列表中的特定位置。子级列表中这些从 0 开始的索引位置与显示对象的分层 （从前到后顺序）有关。例如，请考虑下列三个显示对象。每个对象都是从称为 Ball 的自定义类创建的。

使用 addChildAt() 方法可以调整这些显示对象在容器中的分层。例如，请看以下代码：

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

执行此代码后，显示对象在 container DisplayObjectContainer 对象中的定位如下所示。请注意对象的分层。

要重新将对象定位到显示列表的顶部，只需重新将其添加到列表中。例如，在前面的代码后，要将 ball_A 移到堆栈的顶部，请使用下面的代码行：

```
container.addChild(ball_A);
```

此代码可有效地将 ball_A 从它在 container 的显示列表中的位置删除，然后将它重新添加到列表的顶部，最终的结果是将它移到堆栈的顶部。

可以使用 getChildAt() 方法来验证显示对象的图层顺序。getChildAt() 方法根据您向容器传递的索引编号返回容器的子对象。例如，下面的代码显示 container DisplayObjectContainer 对象的子级列表中不同位置的显示对象的名称：

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

如果从父容器的子级列表中删除了一个显示对象，则列表中更高位置的每一个元素在子索引中会分别下移一个位置。例如，接着前面的代码，下面的代码显示如果删除子级列表中位置较低的一个显示对象，container DisplayObjectContainer 中位置 2 的显示对象如何移到位置 1：

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

removeChild() 和 removeChildAt() 方法并不完全删除显示对象实例。这两种方法只是从容器的子级列表中删除显示对象实例。该实例仍可由另一个变量引用。（请使用 delete 运算符完全删除对象。）

由于显示对象只有一个父容器，因此只能在一个显示对象容器中添加显示对象的实例。例如，下面的代码说明了显示对象 tf1 只能存在于一个容器中 （本例中为 Sprite，它扩展 DisplayObjectContainer 类）：

```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

如果将包含在一个显示对象容器中的显示对象添加到另一个显示对象容器中，则会从第一个显示对象容器的子级列表中删除该显示对象。

除了上面介绍的方法之外， DisplayObjectContainer 类还定义了用于使用子显示对象的几个方法，其中包括：

- contains()：确定显示对象是否是 DisplayObjectContainer 的子级。
- getChildByName()：按名称检索显示对象。
- getChildIndex()：返回显示对象的索引位置。
- setChildIndex()：更改子显示对象的位置。
- removeChildren()：删除多个子显示对象。
- swapChildren()：交换两个显示对象的前后顺序。
- swapChildrenAt()：交换两个显示对象的前后顺序 （由其索引值指定）。

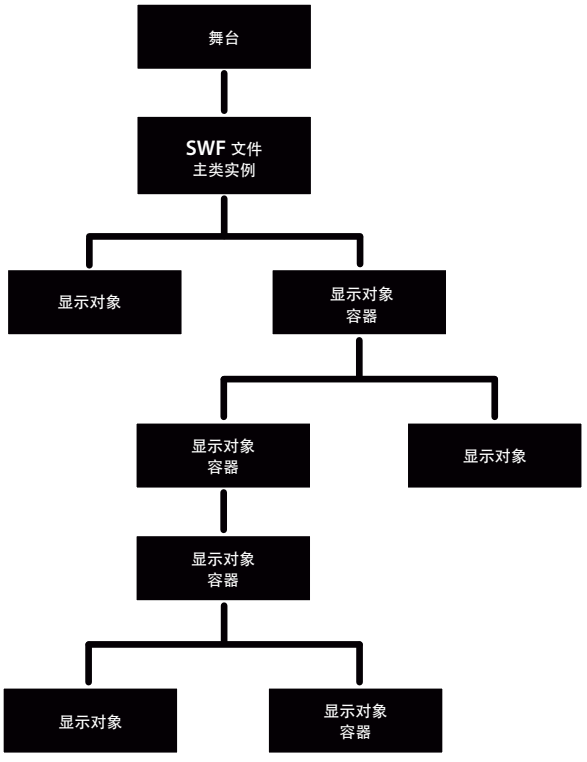
有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的相关条目。

回想一下，不在显示列表中的显示对象 （即不包括在作为舞台子级的显示对象容器中的显示对象）称为 “ 列表外 ” 显示对象。

遍历显示列表

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如您所见，显示列表是一个树结构。树的顶部是舞台，它可以包含多个显示对象。那些本身就是显示对象容器的显示对象可以包含其他显示对象或显示对象容器。



DisplayObjectContainer 类包括通过显示对象容器的子级列表遍历显示列表的属性和方法。例如，考虑下面的代码，其中在 container 对象 （该对象为 Sprite， Sprite 类用于扩展 DisplayObjectContainer 类）中添加了两个显示对象 title 和 pict：

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

getChildAt() 方法返回显示列表中特定索引位置的子级：

```
trace(container.getChildAt(0) is TextField); // true
```

您也可以按名称访问子对象。每个显示对象都有一个名称属性；如果没有指定该属性， Flash Player 或 AIR 会指定一个默认值，如 "instance1"。例如，下面的代码说明了如何使用 getChildByName() 方法来访问名为 "banana loader" 的子显示对象：

```
trace(container.getChildByName("banana loader") is Loader); // true
```

与使用 getChildAt() 方法相比，使用 getChildByName() 方法会导致性能降低。

由于显示对象容器可以包含其他显示对象容器作为其显示列表中的子对象，因此您可将应用程序的完整显示列表作为树来遍历。例如，在前面说明的代码摘录中，完成 pict Loader 对象的加载操作后， pict 对象将加载一个子显示对象，即位图。要访问此位图显示对象，可以编写 pict.getChildAt(0)。还可以编写 container.getChildAt(0).getChildAt(0) （由于 container.getChildAt(0) == pict）。

下面的函数提供了显示对象容器中显示列表的缩进式 trace() 输出：

```
function traceDisplayList(container:DisplayObjectContainer,indentString:String = "") :void
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "")
        }
    }
}
```

Adobe Flex

如果使用 Flex，您应了解 Flex 定义了许多组件显示对象类，这些类会覆盖 DisplayObjectContainer 类的显示列表访问方法。例如， mx.core 包的 Container 类会覆盖 DisplayObjectContainer 类 （Container 类所扩展的类）的 addChild() 方法和其他方法。就 addChild() 方法而言，该类覆盖该方法的结果是，在 Flex 中，所有类型的显示对象都不能添加到 Container 实例。在本例中，被覆盖的方法要求所添加的子对象为 mx.core.UIComponent 对象类型。

设置舞台属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Stage 类用于覆盖 DisplayObject 类的大多数属性和方法。如果调用其中一个被覆盖的属性或方法，Flash Player 和 AIR 会引发异常。例如，Stage 对象不具有 x 或 y 属性，因为作为应用程序的主容器，该对象的位置是固定的。x 和 y 属性是指显示对象相对于其容器的位置，因为舞台没有包含在其他显示对象容器中，所以这些属性不适用。

注：Stage 类的某些属性和方法只适用于与加载的第一个 SWF 文件在同一个安全沙箱中的显示对象。有关详细信息，请参阅第 912 页的 “Stage 安全性 ”。

控制播放帧速率

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Stage 类的 frameRate 属性用于设置加载到应用程序中的所有 SWF 文件的帧速率。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

控制舞台缩放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当调整呈示 Flash Player 或 AIR 的屏幕部分的大小时，运行时会自动调整舞台内容来加以补偿。Stage 类的 scaleMode 属性可确定如何调整舞台内容。此属性可以设置为 4 个不同值，如 flash.display.StageScaleMode 类中的常量所定义：

- StageScaleMode.EXACT_FIT 缩放 SWF 以填满新舞台尺寸，而不考虑原来的内容高宽比。宽度和高度的缩放系数可能会有所不同，因此当舞台的高宽比发生更改时，显示的内容有可能随之压扁或拉长。

- StageScaleMode.SHOW_ALL 会在保持内容高宽比的前提下，缩放 SWF 以适应新舞台。这种缩放模式可完整显示所有内容，但是可能导致出现“邮箱”式边框，就像使用标准电视收看宽屏电影时出现的黑色长条。
- StageScaleMode.NO_BORDER 会在保持内容高宽比的前提下，缩放 SWF 以填满新舞台。这种缩放模式可以充分利用舞台显示区域，但可能导致裁切。
- StageScaleMode.NO_SCALE — 不缩放 SWF。如果新舞台较小，内容将遭到裁切；如果较大，所增加的空间将显示为空白。

  仅在 StageScaleMode.NO_SCALE 缩放模式中，Stage 类的 stageWidth 和 stageHeight 属性才能用于确定窗口调整大小后的实际像素尺寸。（在其他缩放模式中， stageWidth 和 stageHeight 属性始终反映的是 SWF 的原始宽度和高度。）此外，当 scaleMode 设置为 StageScaleMode.NO_SCALE 并且调整了 SWF 文件大小时，将调度 Stage 类的 resize 事件，以允许您进行相应地调整。

  因此，将 scaleMode 设置为 StageScaleMode.NO_SCALE 可以更好地控制如何根据需要调整屏幕内容以适合窗口大小。例如，在包含视频和控制栏的 SWF 中，您可能希望在调整舞台大小时控制栏的大小保持不变，而仅更改视频窗口大小以适应舞台大小的更改。以下示例中演示了这一点：

```
// mainContent is a display object containing the main content;
// it is positioned at the top-left corner of the Stage, and
// it should resize when the SWF resizes.

// controlBar is a display object (e.g. a Sprite) containing several
// buttons; it should stay positioned at the bottom-left corner of the
// Stage (below mainContent) and it should not resize when the SWF
// resizes.

import flash.display.Stage;
import flash.display.StageAlign;
import flash.display.StageScaleMode;
import flash.events.Event;

var swfStage:Stage = mainContent.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;
swfStage.addEventListener(Event.RESIZE, resizeDisplay);

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the main content area
    var newContentHeight:Number = swfHeight - controlBar.height;
    mainContent.height = newContentHeight;
    mainContent.scaleX = mainContent.scaleY;

    // Reposition the control bar.
    controlBar.y = newContentHeight;
}
```

设置 **AIR** 窗口的舞台缩放模式

舞台 scaleMode 属性确定在调整窗口大小时舞台如何缩放和剪裁子显示对象。在 AIR 中只应使用 noScale 模式。在此模式中不缩放舞台。舞台的大小直接随窗口的范围变化。如果将窗口调小，则可能会剪裁对象。

舞台缩放模式旨在用于您无法始终控制舞台大小或高宽比的环境，例如 Web 浏览器。当舞台与应用程序的理想大小或高宽比不匹配时，使用这些模式可以选择最好的折衷方案。在 AIR 中，您始终能够控制舞台，因此在大多数情况下，重新放置您的内容或调整窗口尺寸可以获得比启用舞台缩放更好的结果。

在浏览器中，以及对于初始 AIR 窗口，窗口大小与初始缩放系数之间的关系是从所加载的 SWF 文件中读取的。然而，在创建 NativeWindow 对象时，AIR 选择窗口大小和缩放系数 72:1 之间的任意关系。因此如果窗口为 72x72 像素，则以 10x10 像素的正确大小绘制添加到窗口的 10x10 矩形。但是，如果窗口为 144x144 像素，则 10x10 像素的矩形会缩放为 20x20 像素。如果您坚持对窗口舞台使用 scaleMode 而不是 noScale，则可以通过将窗口中的任何显示对象的缩放系数都设置为 72 像素与舞台的当前宽度和高度之比加以补偿。例如，以下代码计算名为 client 的显示对象所需的缩放系数：

```
if(newWindow.stage.scaleMode != StageScaleMode.NO_SCALE){
client.scaleX = 72/newWindow.stage.stageWidth;
client.scaleY = 72/newWindow.stage.stageHeight;
}
```

注：Flex 和 HTML 窗口自动将舞台 scaleMode 设置为 noScale。更改 scaleMode 会打乱这些窗口类型中所使用的自动布局机制。

使用全屏模式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用全屏模式可以将影片的舞台设置为填充查看者的整个显示器，而不包含任何边框或菜单。 Stage 类的 displayState 属性用于切换 SWF 的全屏模式。可以将 displayState 属性设置为由 flash.display.StageDisplayState 类中的常量定义的其中一个值。若要打开全屏模式，请将 displayState 属性设置为 StageDisplayState.FULL_SCREEN：

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

若要启用全屏交互模式 （Flash Player 11.3 中的新增功能），请将 displayState 属性设置为 StageDisplayState.FULL_SCREEN_INTERACTIVE：

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

在 Flash Player 中，只能通过 ActionScript 响应鼠标单击 （包括右键单击）或按键来启动全屏模式。在应用程序安全沙箱中运行的 AIR 内容不要求在响应用户手势时进入全屏模式。

若要退出全屏模式，请将 displayState 属性设置为 StageDisplayState.NORMAL。

```
stage.displayState = StageDisplayState.NORMAL;
```

此外，用户可以通过将焦点切换到其他窗口或使用以下组合键之一退出全屏模式：Esc 键 （所有平台）、 Ctrl-W (Windows)、 Command-W (Mac) 或 Alt-F4 (Windows)。

启用 **Flash Player** 中的全屏模式

若要为 HTML 页中嵌入的 SWF 文件启用全屏模式，必须在嵌入 Flash Player 的 HTML 代码中加入含有名称 allowFullScreen 和值 true 的 param 标签和 embed 属性，如下所示：

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowFullScreen="true" />
</object>
```

在 Flash 创作工具中选择 “ 文件 ”->“ 发布设置 ”，并在 “ 发布设置 ” 对话框中的 “HTML” 选项卡上选择 “ 仅 Flash - 允许全屏 ” 模板。

在 Flex 中，确保 HTML 模板包含支持全屏的 <object> 和 <embed> 标签。

如果要在网页中使用 JavaScript 来生成 SWF 嵌入标签，则必须更改 JavaScript 以添加 allowFullScreen param 标签和属性。例如，如果 HTML 页使用 AC_FL_RunContent() 函数（在由 Flash Professional 和 Flash Builder 生成的 HTML 页中使用），则应在该函数调用中添加 allowFullScreen 参数，如下所示：

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

这不适用于在独立 Flash Player 中运行的 SWF 文件。

注：如果将窗口模式 （HTML 中的 wmode）设置为不透明无窗口 (opaque) 或透明无窗口 (transparent)，则全屏窗口始终是不透明的

对浏览器中的 Flash Player 使用全屏模式时还有一些安全方面的限制。第 894 页的 " 安全性 " 中对这些限制进行了说明。

在 **Flash Player 11.3** 和更高版本中启用全屏交互模式

Flash Player 11.3 和更高版本支持全屏交互模式，该模式可实现对所有键盘键的完全支持 （**Esc** 除外，使用该键将退出全屏交互模式）。全屏交互模式对于游戏非常有用 （例如，支持多人游戏聊天或第一人射击游戏中的 WASD 键盘控件。）

若要为 HTML 页中嵌入的 SWF 文件启用全屏交互模式，必须在嵌入 Flash Player 的 HTML 代码中加入含有名称 allowFullScreenInteractive 和值 true 的 param 标签和 embed 属性，如下所示：

```
<object>
    ...
    <param name="allowFullScreenInteractive" value="true" />
    <embed ... allowFullScreenInteractive="true" />
</object>
```

在 Flash 创作工具中选择 " 文件 "->" 发布设置 "，并在 " 发布设置 " 对话框中的 "HTML" 选项卡上选择 " 仅 Flash - 允许全屏 " 模板。

在 Flash Builder 和 Flex 中，确保 HTML 模板包括支持全屏交互模式的 <object> 和 <embed> 标签。

如果要在网页中使用 JavaScript 来生成 SWF 嵌入标签，则必须更改 JavaScript 以添加 allowFullScreenInteractive param 标签和属性。例如，如果 HTML 页使用 AC_FL_RunContent() 函数 （在由 Flash Professional 和 Flash Builder 生成的 HTML 页中使用），则应在该函数调用中添加 allowFullScreenInteractive 参数，如下所示：

```
AC_FL_RunContent(
    ...
    'allowFullScreenInteractive','true',
    ...
    ); //end AC code
```

这不适用于在独立 Flash Player 中运行的 SWF 文件。

全屏舞台大小和缩放

Stage.fullScreenHeight 和 Stage.fullScreenWidth 属性返回在转为全屏大小时所使用的显示器的高度和宽度 （如果是立即进入全屏状态）。在您检索到这些值之后，但在进入全屏模式之前，如果用户有机会将浏览器从一台显示器移至另一台显示器，则这些值可能不正确。如果是在将 Stage.displayState 属性设置为 StageDisplayState.FULL_SCREEN 的同一个事件处理函数中检索这些值，则这些值正确。对于拥有多台显示器的用户， SWF 内容扩大时只会填充一台显示器。 Flash Player 和 AIR 使用度量信息来确定哪个显示器包含 SWF 的最大部分内容，然后使用该显示器提供全屏模式。fullScreenHeight 和 fullScreenWidth 属性仅反映出显示器用于全屏模式的大小。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 Stage.fullScreenHeight 和 Stage.fullScreenWidth。

全屏模式的舞台缩放行为与正常模式下的相同；缩放比例由 Stage 类的 scaleMode 属性控制。如果 scaleMode 属性设置为 StageScaleMode.NO_SCALE，则舞台的 stageWidth 和 stageHeight 属性会发生更改，以反映 SWF 所占用的屏幕区域的大小 （在本例中为整个屏幕）；如果在浏览器中查看，则此属性的 HTML 参数用于控制该设置。

打开或关闭全屏模式时，可以使用 Stage 类的 fullScreen 事件来进行检测和响应。例如，进入或退出全屏模式时，您可能需要重新定位、添加或删除屏幕中的项目，如本例中所示：

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
    if (event.fullScreen)
    {
        // Remove input text fields.
        // Add a button that closes full-screen mode.
    }
    else
    {
        // Re-add input text fields.
        // Remove the button that closes full-screen mode.
    }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

正如此代码所示， fullScreen 事件的事件对象是 flash.events.FullScreenEvent 类的实例，它包含指示是启用 (true) 还是禁用 (false) 全屏模式的 fullScreen 属性。

全屏模式下的键盘支持

当 Flash Player 在浏览器中运行时，全屏模式下将禁用所有与键盘相关的 ActionScript，如 TextField 实例中的键盘事件和文本输入。但有一些例外 （启用的键）：

- 经过挑选的非打印键，尤其是方向键、空格键和 Tab 键
- 用于终止全屏模式的快捷键：Esc （Windows 和 Mac）、 Ctrl-W (Windows)、 Command-W (Mac) 和 Alt-F4

对于在独立 Flash Player 或 AIR 中运行的 SWF 内容，不存在这些限制。 AIR 支持允许键盘输入的交互式全屏模式。

全屏模式下的鼠标支持

默认情况下，全屏模式下的鼠标事件的工作方式与未处于全屏模式时相同。但是，在全屏模式下，您可以选择设置 Stage.mouseLock 属性，以启用鼠标锁定。鼠标锁定将禁用光标，且启用没有限制的鼠标移动。

注：您只能在桌面应用程序的全屏模式下启用鼠标锁定。如果在未处于全屏模式的应用程序上或为移动设备上的应用程序设置鼠标锁定，则会引发异常。

在下列情况下鼠标锁定被自动禁用，鼠标光标再次显示：

- 用户使用 Esc 键 （所有平台）、 Ctrl-W (Windows)、 Command-W (Mac) 或 Alt-F4 (Windows) 退出全屏模式。
- 应用程序窗口丢失焦点。
- 任何设置 UI 都可见，包括所有隐私对话框。
- 显示本机对话框，如文件上传对话框。

与鼠标移动相关的事件，例如 mouseMove 事件，使用 MouseEvent 类表示事件对象。禁用鼠标锁定时，使用 MouseEvent.localX 和 MouseEvent.localY 属性确定鼠标的位置。启用鼠标锁定时，使用 MouseEvent.movementX 和 MouseEvent.movementY 属性确定鼠标的位置。 movementX 和 movementY 属性包含自上一个事件以来的鼠标位置的变化，而不是鼠标位置的绝对坐标。

全屏模式下的硬件缩放

使用 Stage 类的 fullScreenSourceRect 属性可以设置 Flash Player 或 AIR 以将舞台的特定区域放大为全屏幕模式。 Flash Player 和 AIR 使用用户计算机上的图形和视频卡进行硬件缩放 （如果可用），一般来说显示内容的速度要快于软件缩放。

若要利用硬件缩放功能，请将整个舞台或部分舞台设置为全屏模式。以下 ActionScript 3.0 代码将整个舞台设置为全屏模式：

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

如果将此属性设置为有效矩形并将 displayState 属性设置为全屏模式， Flash Player 和 AIR 对指定区域进行缩放。ActionScript 中的实际舞台大小 （以像素为单位）不会发生改变。Flash Player 和 AIR 对矩形大小强制规定了最小限制，以容纳标准的 “ 按 Esc 退出全屏模式 ” 消息。通常，此限制大约为 260 x 30 个像素，但可能因平台和 Flash Player 版本而异。

仅当 Flash Player 或 AIR 处于非全屏模式时，才能设置 fullScreenSourceRect 属性。若要正确使用此属性，请先设置此属性，然后再将 displayState 属性设置为全屏模式。

若要启用缩放功能，请将 fullScreenSourceRect 属性设置为矩形对象。

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

若要禁用缩放功能，请将 fullScreenSourceRect 属性设置为 null。

```
stage.fullScreenSourceRect = null;
```

若要利用 Flash Player 中的所有硬件加速功能，请通过 Flash Player 的 “ 设置 ” 对话框将其启用。若要加载该对话框，请在浏览器的 Flash Player 内容中单击右键 (Windows) 或按住 Control 的同时单击 (Mac)。选择 “ 显示 ” 选项卡 （第一个选项卡），然后选中复选框：“ 启用硬件加速 ”。

直接窗口模式和 **GPU** 合成窗口模式

Flash Player 10 引入了两种窗口模式：直接模式和 GPU 合成模式。可以通过 Flash 创作工具中的发布设置启用这些模式。AIR 中不支持这两种模式。若要利用这两种模式，必须为 Flash Player 启用硬件加速。

直接模式使用最快、最直接的路径将图形推送至屏幕，这有利于视频播放。

GPU 合成模式使用显卡中的图形处理单元来加速合成。视频合成是层叠多幅图像以创建单幅视频图像的过程。当采用 GPU 加速合成时，这可提高 YUV 转换、颜色校正、旋转或缩放以及混合的性能。YUV 转换是指将用于传输的合成模拟信号转换为视频摄像头和显示器所使用的 RGB （红、绿、蓝）颜色模型的颜色转换。使用 GPU 加速合成可减少内存占用量以及 CPU 的计算负担。这还促成了标准清晰度视频更平滑的播放。

在实施这些窗口模式时应谨慎小心。使用 GPU 合成可能消耗很多内存资源和 CPU 资源。如果某些操作 （如混合模式、过滤、剪裁或遮罩）无法在 GPU 中执行，则可以通过软件来完成。 Adobe 建议在使用这些模式时，应限制为每个 HTML 页面一个 SWF 文件，并且不应对横幅启用这些模式。 Flash“ 测试影片 ” 工具不使用硬件加速，但是可通过 “ 发布预览 ” 选项来使用硬件加速。

将 SWF 文件中的帧速率设置为高于 60 （最大屏幕刷新率）将不起作用。将帧速率设置为 50 到 55 之间将允许放弃的帧，这种情况经常由于各种原因而发生。

使用直接模式对于 Windows 需要 Microsoft DirectX 9 以及 128 MB VRAM，对于 Apple Macintosh Mac OS X 10.2 版或更高版本需要 OpenGL。GPU 合成模式需要在具有 128 MB VRAM 的 Windows 上支持 Microsoft DirectX 9 和 Pixel Shader 2.0。在 Mac OS X 和 Linux 上，GPU 合成模式需要 OpenGL 1.5 以及多种 OpenGL 扩展（帧缓冲区对象、多纹理、着色器对象、着色语言和片段着色器）。

可通过 Flash“ 发布设置 ” 对话框，使用 “Flash” 选项卡上的 “ 硬件加速 ” 菜单，为每个 SWF 单独激活 direct 和 gpu 加速模式。如果选择 “ 无 ”，则窗口模式将按照 “HTML” 选项卡上 “ 窗口模式 ” 设置所指定的项，转换为 default、 transparent 或 opaque。

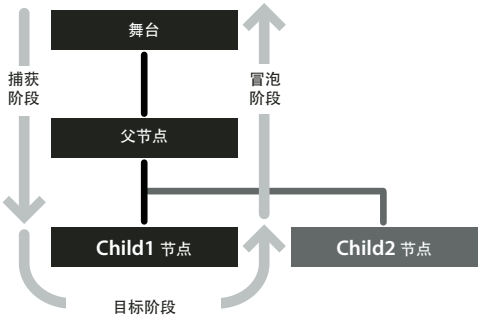
处理显示对象的事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

DisplayObject 类从 EventDispatcher 类继承。这意味着，每个显示对象都可完全参与到事件模型中 （在第 106 页的 " 处理事件 " 中介绍）。每个显示对象都可使用其 addEventListener() 方法 （继承自 EventDispatcher 类）来侦听特定的事件，但仅当侦听对象是该事件的事件流的一部分时才能实现此功能。

当 Flash Player 或 AIR 调度某个事件对象时，该事件对象会执行从舞台到发生事件的显示对象的往返行程。例如，如果用户单击名为 child1 的显示对象， Flash Player 会沿显示列表层次将事件对象从舞台向下调度到 child1 显示对象。

从概念上说，事件流分为三个阶段，如下图所示：



有关详细信息，请参阅第 106 页的 " 处理事件 "。

使用显示对象事件时需要记住的一个重要问题是：从显示列表中删除显示对象时，事件侦听器的存在将会对是否从内存中自动删除显示对象 （垃圾回收）产生影响。如果显示对象拥有订阅为其事件的侦听器的对象，即使从显示列表中删除了显示对象，也不会从内存中删除显示对象，因为显示对象仍然拥有对这些侦听器对象的引用。有关详细信息，请参阅第 116 页的 " 管理事件侦听器 "。

选择 DisplayObject 子类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可选的子类有多个，使用显示对象时要做出的一个重要决策是：每个显示对象的用途是什么。以下原则可以帮助您作出决策。无论是需要类实例，还是选择要创建的类的基类，这些建议都适用：

- 如果不需要可作为其他显示对象的容器的对象 （即只需要用作独立屏幕元素的对象），请根据用途选择 DisplayObject 或 InteractiveObject 两个子类中的一个：
  - 用于显示位图图像的 Bitmap。
  - 用于添加文本的 TextField。
  - 用于显示视频的 Video。
  - 用于绘制屏幕内容的 " 画布 " 的 Shape。特别是，如果要创建用于在屏幕上绘制形状的实例，而该实例不是其他显示对象的容器，则使用 Shape 比使用 Sprite 或 MovieClip 有明显的性能优势。
  - 用于由 Flash 创作工具创建的项的 MorphShape、 StaticText 或 SimpleButton。（无法以编程方式创建这些类的实例，但可以通过创建具有这些数据类型的变量来引用使用 Flash 创作工具创建的项。）
- 如果需要使用变量来引用主舞台，请使用 Stage 类作为其数据类型。

- 如果需要容器来加载外部 SWF 文件或图像文件，请使用 Loader 实例。加载的内容将作为 Loader 实例的子级添加到显示列表中。其数据类型将取决于加载内容的性质，如下所示：
  - 加载的图像将是 Bitmap 实例。
  - 使用 ActionScript 3.0 编写的已加载 SWF 文件将是 Sprite 或 MovieClip 实例（或这些类的子类的实例，由内容创建者指定）。
  - 使用 ActionScript 1.0 或 ActionScript 2.0 编写的已加载 SWF 文件将是 AVM1Movie 实例。
- 如果需要将一个对象用作其他显示对象的容器（无论是否还要使用 ActionScript 在显示对象上进行绘制），请选择其中一个 DisplayObjectContainer 子类：
  - 如果对象是只使用 ActionScript 创建的，或者如果对象作为只使用 ActionScript 创建和处理的自定义显示对象的基类，请选择 Sprite。
  - 如果要通过创建变量来引用在 Flash 创作工具中创建的影片剪辑元件，请选择 MovieClip。
- 如果要创建的类与 Flash 库中的影片剪辑元件关联，请选择其中一个 DisplayObjectContainer 子类作为该类的基类：
  - 如果关联的影片剪辑元件在多个帧上有内容，请选择 MovieClip
  - 如果关联的影片剪辑元件仅在第一帧上有内容，请选择 Sprite

处理显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

无论选择使用哪个显示对象，都会有许多的操作，这些操作作为屏幕上显示的一些元素是所有显示对象共有的。例如，可以在屏幕上确定所有显示对象的位置、前后移动显示对象的堆叠顺序、缩放或旋转显示对象等。因为所有显示对象都从它们共有的基类 (DisplayObject) 继承了此功能，所以无论是要操作 TextField 实例、Video 实例、Shape 实例还是其他任何显示对象，此功能的行为都相同。以下部分将详细说明几个常用显示对象操作。

改变位置

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

对任何显示对象进行的最基本操作是确定显示对象在屏幕上的位置。若要设置显示对象的位置，请更改对象的 x 和 y 属性。

```
myShape.x = 17;
myShape.y = 212;
```

显示对象定位系统将舞台视为一个笛卡尔坐标系（带有水平 x 轴和垂直 y 轴的常见网格系统）。坐标系的原点（x 和 y 轴相交的 0,0 坐标）位于舞台的左上角。从原点开始， x 轴的值向右为正，向左为负，而 y 轴的值向下为正，向上为负（与典型的图形系统相反）。例如，通过前面的代码行可以将对象 myShape 移到 x 轴坐标 17（原点向右 17 个像素）和 y 轴坐标 212（原点向下 212 个像素）。

默认情况下，当使用 ActionScript 创建显示对象时， x 和 y 属性均设置为 0，从而可将对象放在其父内容的左上角。

改变相对于舞台的位置

x 和 y 属性始终是指显示对象相对于其父显示对象坐标轴的 0,0 坐标的位置，记住这一点很重要。因此，对于包含在 Sprite 实例内的 Shape 实例（如圆），如果将 Shape 对象的 x 和 y 属性设置为 0，则会将圆放在 Sprite 的左上角，该位置不一定是舞台的左上角。若要确定对象相对于全局舞台坐标的位置，可以使用任何显示对象的 globalToLocal() 方法将坐标从全局（舞台）坐标转换为本地（显示对象容器）坐标，如下所示：

```
// Position the shape at the top-left corner of the Stage,
// regardless of where its parent is located.

// Create a Sprite, positioned at x:200 and y:200.
var mySprite:Sprite = new Sprite();
mySprite.x = 200;
mySprite.y = 200;
this.addChild(mySprite);

// Draw a dot at the Sprite's 0,0 coordinate, for reference.
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.beginFill(0x000000);
mySprite.graphics.moveTo(0, 0);
mySprite.graphics.lineTo(1, 0);
mySprite.graphics.lineTo(1, 1);
mySprite.graphics.lineTo(0, 1);
mySprite.graphics.endFill();

// Create the circle Shape instance.
var circle:Shape = new Shape();
mySprite.addChild(circle);

// Draw a circle with radius 50 and center point at x:50, y:50 in the Shape.
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0xff0000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();

// Move the Shape so its top-left corner is at the Stage's 0, 0 coordinate.
var stagePoint:Point = new Point(0, 0);
var targetPoint:Point = mySprite.globalToLocal(stagePoint);
circle.x = targetPoint.x;
circle.y = targetPoint.y;
```

同样，也可以使用 DisplayObject 类的 localToGlobal() 方法将本地坐标转换为舞台坐标。

使用鼠标移动显示对象

您可以在 ActionScript 中使用两种技术使用户可以使用鼠标来移动显示对象。在这两种情况下，会使用两个鼠标事件：按下鼠标按键时，通知对象跟随鼠标光标；松开鼠标按键时，通知对象停止跟随鼠标光标。

注：Flash Player 11.3 及更高版本，AIR 3.3 及更高版本：您也可以使用 MouseEvent.RELEASE_OUTSIDE 事件涵盖用户在包含 Sprite 的边界外释放鼠标按钮的情况。

第一种技术使用 startDrag() 方法，比较简单，但限制较多。按下鼠标按键时，将调用要拖动的显示对象的 startDrag() 方法。松开鼠标按键时，将调用 stopDrag() 方法。 Sprite 类定义这两个功能，因此移动的对象必须是 Sprite 或它的子类。

```
// This code creates a mouse drag interaction using the startDrag()
// technique.
// square is a MovieClip or Sprite instance).

import flash.events.MouseEvent;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    square.startDrag();
}

// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    square.stopDrag();
}

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

这种方法有一个非常大的限制：使用 startDrag() 时，每次只能拖动一个项目。如果正在拖动一个显示对象，然后对另一个显示对象调用了 startDrag() 方法，则第一个显示对象会立即停止跟随鼠标。例如，如果 startDragging() 函数如下发生了更改，则只拖动 circle 对象，而不管 square.startDrag() 方法调用：

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

由于每次只能使用 startDrag() 拖动一个对象，因此，可以对任何显示对象调用 stopDrag() 方法，这会停止当前正在拖动的任何对象。

如果需要拖动多个显示对象，或者为了避免因多个对象可能使用 startDrag() 而发生冲突，最好使用鼠标跟随方法来创建拖动效果。通过这种技术，当按下鼠标按键时，会将函数作为舞台的 mouseMove 事件的侦听器来订阅。然后，每次鼠标移动时都会调用此函数，它将使所拖动的对象跳到鼠标所在的 x,y 坐标。松开鼠标按键后，取消此函数作为侦听器的订阅，这意味着鼠标移动时不再调用该函数且对象停止跟随鼠标光标。下面是演示说明此技术的一些代码：

```
// This code moves display objects using the mouse-following
// technique.
// circle is a DisplayObject (e.g. a MovieClip or Sprite instance).

import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // Record the difference (offset) between where
    // the cursor was when the mouse button was pressed and the x, y
    // coordinate of the circle when the mouse button was pressed.
    offsetX = event.stageX - circle.x;
    offsetY = event.stageY - circle.y;

    // tell Flash Player to start listening for the mouseMove event
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called when the mouse button is released.
```

```
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragCircle);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragCircle(event:MouseEvent):void
{
    // Move the circle to the location of the cursor, maintaining
    // the offset between the cursor's location and the
    // location of the dragged object.
    circle.x = event.stageX - offsetX;
    circle.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

除使显示对象跟随鼠标光标之外，经常需要将拖动的对象移动到显示的前方，以使其像是浮动在所有其他对象上。例如，假设您有两个对象 （一个圆和一个正方形），都可以跟随鼠标移动。如果圆在显示列表中出现在正方形之下，您单击并拖动圆时光标会出现在正方形之上，圆好像在正方形之后滑动，这样中断了拖放视觉效果。您可以使用拖放交互组件避免这一点，以便在单击圆时圆移到显示列表的顶部，使圆会始终出现在其他任何内容的顶部。

以下代码 （根据上一示例改写）使两个显示对象 （一个圆和一个正方形）可跟随鼠标移动。只要在任一个显示对象上按下鼠标按键，该显示对象就会移到舞台显示列表的顶部，所以拖动的项目始终出现在顶部。（新代码或根据以前代码更改的代码显示为粗体。）

```
// This code creates a drag-and-drop interaction using the mouse-following
// technique.
// circle and square are DisplayObjects (e.g. MovieClip or Sprite
// instances).

import flash.display.DisplayObject;
import flash.events.MouseEvent;

var offsetX:Number;
var offsetY:Number;
var draggedObject:DisplayObject;

// This function is called when the mouse button is pressed.
function startDragging(event:MouseEvent):void
{
    // remember which object is being dragged
    draggedObject = DisplayObject(event.target);

    // Record the difference (offset) between where the cursor was when
    // the mouse button was pressed and the x, y coordinate of the
    // dragged object when the mouse button was pressed.
    offsetX = event.stageX - draggedObject.x;
    offsetY = event.stageY - draggedObject.y;

    // move the selected object to the top of the display list
    stage.addChild(draggedObject);

    // Tell Flash Player to start listening for the mouseMove event.
    stage.addEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}
```

```
// This function is called when the mouse button is released.
function stopDragging(event:MouseEvent):void
{
    // Tell Flash Player to stop listening for the mouseMove event.
    stage.removeEventListener(MouseEvent.MOUSE_MOVE, dragObject);
}

// This function is called every time the mouse moves,
// as long as the mouse button is pressed down.
function dragObject(event:MouseEvent):void
{
    // Move the dragged object to the location of the cursor, maintaining
    // the offset between the cursor's location and the location
    // of the dragged object.
    draggedObject.x = event.stageX - offsetX;
    draggedObject.y = event.stageY - offsetY;

    // Instruct Flash Player to refresh the screen after this event.
    event.updateAfterEvent();
}

circle.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
circle.addEventListener(MouseEvent.MOUSE_UP, stopDragging);

square.addEventListener(MouseEvent.MOUSE_DOWN, startDragging);
square.addEventListener(MouseEvent.MOUSE_UP, stopDragging);
```

若要进一步扩展这种效果，如在几副纸牌 （或几组标记）之间移动纸牌 （或标记）的游戏中，您可以在 “ 拿出 ” 拖动对象时将拖动对象添加到舞台的显示列表中，然后在 “ 放入 ” 拖动对象时 （通过松开鼠标按键）将拖动对象添加到另一个显示列表中 （如 “ 那副纸牌 ” 或 “ 那组标记 ”）。

最后，要增强效果，您可以在单击显示对象时 （开始拖动显示对象时）对显示对象应用投影滤镜，然后在松开对象时删除投影。有关在 ActionScript 中使用投影滤镜和其他显示对象滤镜的详细信息，请参阅第 224 页的 “ 过滤显示对象 ”。

平移和滚动显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果显示对象太大，不能在要显示它的区域中完全显示出来，则可以使用 scrollRect 属性定义显示对象的可查看区域。此外，通过更改 scrollRect 属性响应用户输入，可以使内容左右平移或上下滚动。

scrollRect 属性是 Rectangle 类的实例，Rectangle 类包括将矩形区域定义为单个对象所需的有关值。最初定义显示对象的可查看区域时，请创建一个新的 Rectangle 实例并为该实例分配显示对象的 scrollRect 属性。以后进行滚动或平移时，请将 scrollRect 属性读入单独的 Rectangle 变量，然后更改所需的属性 （例如，更改 Rectangle 实例的 x 属性进行平移，或更改 y 属性进行滚动）。然后将该 Rectangle 实例重新分配给 scrollRect 属性，将更改的值通知显示对象。

例如，下面的代码定义名为 bigText 的 TextField 对象的可查看区域，该对象因太高而无法容纳在 SWF 文件的边界内。单击名为 up 和 down 的两个按钮时，它们调用的函数通过修改 scrollRect Rectangle 实例的 y 属性而使 TextField 对象的内容向上或向下滚动。

```
import flash.events.MouseEvent;
import flash.geom.Rectangle;

// Define the initial viewable area of the TextField instance:
// left: 0, top: 0, width: TextField's width, height: 350 pixels.
bigText.scrollRect = new Rectangle(0, 0, bigText.width, 350);

// Cache the TextField as a bitmap to improve performance.
bigText.cacheAsBitmap = true;

// called when the "up" button is clicked
function scrollUp(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Decrease the y value of the rectangle by 20, effectively
    // shifting the rectangle down by 20 pixels.
    rect.y -= 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

// called when the "down" button is clicked
function scrollDown(event:MouseEvent):void
{
    // Get access to the current scroll rectangle.
    var rect:Rectangle = bigText.scrollRect;
    // Increase the y value of the rectangle by 20, effectively
    // shifting the rectangle up by 20 pixels.
    rect.y += 20;
    // Reassign the rectangle to the TextField to "apply" the change.
    bigText.scrollRect = rect;
}

up.addEventListener(MouseEvent.CLICK, scrollUp);
down.addEventListener(MouseEvent.CLICK, scrollDown);
```

正如此示例所示，使用显示对象的 scrollRect 属性时，最好指定 Flash Player 或 AIR 应使用 cacheAsBitmap 属性将显示对象的内容缓存为位图。这样，每次滚动显示对象时，Flash Player 和 AIR 就不必重绘显示对象的整个内容，而只需改用缓存的位图即可将所需部分直接呈示到屏幕上。有关详细信息，请参阅第 152 页的 “ 缓存显示对象 ”。

处理大小和缩放对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有两种方式可测量和操作显示对象的大小：使用尺寸属性 （width 和 height）或缩放属性 （scaleX 和 scaleY）。

每个显示对象都有 width 属性和 height 属性，它们最初设置为对象的大小，以像素为单位。您可以通过读取这些属性的值来确定显示对象的大小。还可以指定新值来更改对象的大小，如下所示：

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

更改显示对象的 height 或 width 会导致缩放对象，这意味着对象内容将经过伸展或挤压以适合新区域的大小。如果显示对象仅包含矢量形状，将按新缩放比例重绘这些形状，而品质不变。此时将缩放显示对象中的所有位图图形元素，而不是重绘。例如，缩放图形时，如果数码照片的宽度和高度增加后超出图像中像素信息的实际大小，数码照片将被像素化，使数码照片显示带有锯齿。

当更改显示对象的 width 或 height 属性时， Flash Player 和 AIR 也会更新对象的 scaleX 和 scaleY 属性。

注： TextField 对象是此缩放行为的例外。文本字段需要调整自身大小，以适应文本自动换行和字体大小，因此文本字段在调整大小之后将其 scaleX 或 scaleY 值重置为 1。但是，如果调整 TextField 对象的 scaleX 或 scaleY 值，则宽度和高度值会更改，以适应您提供的缩放值。

这些属性表示显示对象与其原始大小相比的相对大小。 scaleX 和 scaleY 属性使用小数 （十进制）值来表示百分比。例如，如果某个显示对象的 width 已更改，其宽度是原始大小的一半，则该对象的 scaleX 属性的值为 .5，表示 50%。如果其高度加倍，则其 scaleY 属性的值为 2，表示 200%。

```
// circle is a display object whose width and height are 150 pixels.
// At original size, scaleX and scaleY are 1 (100%).
trace(circle.scaleX); // output: 1
trace(circle.scaleY); // output: 1

// When you change the width and height properties,
// Flash Player changes the scaleX and scaleY properties accordingly.
circle.width = 100;
circle.height = 75;
trace(circle.scaleX); // output: 0.6622516556291391
trace(circle.scaleY); // output: 0.4966887417218543
```

此时，大小更改不成比例。换句话说，如果更改一个正方形的 height 但不更改其 width，则其边长不再相同，它将是一个矩形而不是一个正方形。如果要更改显示对象的相对大小，则可以通过设置 scaleX 和 scaleY 属性的值来调整该对象的大小，另一种方法是设置 width 或 height 属性。例如，下面的代码将更改名为 square 的显示对象的 width，然后更改垂直缩放 (scaleY) 以匹配水平缩放，所以正方形的大小成比例。

```
// Change the width directly.
square.width = 150;

// Change the vertical scale to match the horizontal scale,
// to keep the size proportional.
square.scaleY = square.scaleX;
```

控制缩放时的扭曲

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通常，缩放显示对象 （例如水平伸展）时，引起的扭曲在整个对象上是均匀分布的，所以各部分的伸展量是相同的。对于图形和设计元素，这可能是您所希望的结果。但是，有时更希望能控制显示对象的某些部分伸展、某些部分保持不变。这种情况的一个常见示例是有圆角的矩形按钮。进行正常缩放时，按钮的角将伸展，从而使角半径随按钮大小的调整而改变。

但在这种情况下，最好对缩放进行控制，即能够指定应缩放的某些区域 （直边和中间）和不应缩放的区域 （角），以便在缩放后不会出现可见的扭曲。

可以使用 9 切片缩放 (Scale-9) 来创建在其中控制如何缩放对象的显示对象。使用 9 切片缩放时，显示对象被分成 9 个单独的矩形 （一个 3 x 3 的网格，就像一个 “ 井 ” 字）。矩形的大小不必一定相同，您可以指定放置网格线的位置。缩放显示对象时，四个角矩形中的任何内容 （如按钮的圆角）不伸展也不压缩。上中矩形和下中矩形将进行水平缩放，但不进行垂直缩放，而左中矩形和右中矩形将进行垂直缩放，但不进行水平缩放。中心矩形既进行水平缩放又进行垂直缩放。

请记住，如果要创建显示对象且希望某些内容从不缩放，只需要通过放置 9 切片缩放网格的划分线来确保有关内容完全放在其中一个角矩形中即可。

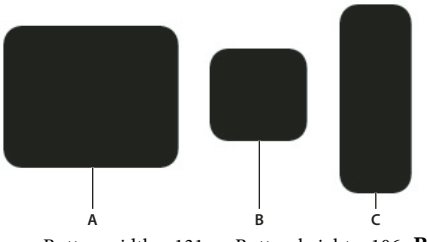
在 ActionScript 中，如果为显示对象的 scale9Grid 属性设置一个值，就会打开该对象的 9 切片缩放并定义该对象的缩放 9 网格中矩形的大小。可以使用 Rectangle 类的实例作为 scale9Grid 属性的值，如下所示：

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Rectangle 构造函数的四个参数是 x 坐标、 y 坐标、 width 和 height。在此示例中，矩形的左上角放在名为 myButton 的显示对象上的点 x: 32，y: 27 处。矩形宽 71 个像素，高 64 个像素（因此其右边位于显示对象上 x 坐标为 103 的位置，其下边位于显示对象上 y 坐标为 92 的位置）。

包含在由 Rectangle 实例定义的区域中的实际区域表示 Scale-9 网格的中心矩形。其他矩形是由 Flash Player 和 AIR 通过扩展 Rectangle 实例的各边计算出来的，如下所示：

在本例中，当按钮放大或缩小时，圆角不拉伸也不压缩，但其他区域将通过调整来适应缩放。



A. myButton.width = 131;myButton.height = 106; **B.** myButton.width = 73;myButton.height = 69; **C.** myButton.width = 54;myButton.height = 141;

缓存显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

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。如果内容保持为静态（如矩形 Shape 实例），则 Flash Player 和 AIR 不会优化内容。因此，更改矩形的位置时，Flash Player 或 AIR 会重绘整个 Shape 实例。

可以通过缓存指定的显示对象来提高 SWF 文件的性能。显示对象是一个 " 表面 "，实际上是位图版本的实例矢量数据，而矢量数据是 SWF 文件中不需要进行太多更改的一种数据。因此，打开缓存的实例不会随 SWF 文件的播放而不断地重绘，这样便可快速呈示 SWF 文件。

注：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。

将显示对象的 cacheAsBitmap 属性设置为 true 会使显示对象缓存其自身的位图表示形式。Flash Player 或 AIR 为该实例创建一个表面对象，该对象是一个缓存的位图，而不是矢量数据。如果要更改显示对象的边框，则重新创建表面而不是调整其大小。表面可以嵌套在其他表面之内。子表面会将其位图复制到它的父表面上。有关详细信息，请参阅第 153 页的 " 启用位图缓存 "。

DisplayObject 类的 opaqueBackground 属性和 scrollRect 属性与使用 cacheAsBitmap 属性的位图缓存有关。尽管这三个属性彼此互相独立，但当对象缓存为位图时， opaqueBackground 和 scrollRect 属性的作用最佳；只有将 cacheAsBitmap 设置为 true 时，才能看到 opaqueBackground 和 scrollRect 属性带来的性能优势。有关滚动显示对象内容的详细信息，请参阅第 148 页的 " 平移和滚动显示对象 "。有关设置不透明背景的详细信息，请参阅第 154 页的 " 设置不透明背景颜色 "。

有关 Alpha 通道遮罩 （要求将 cacheAsBitmap 属性设置为 true）的信息，请参阅第 158 页的 " 遮罩显示对象 "。

何时启用缓存

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

对显示对象启用缓存可创建表面，表面具有助于更快地呈示复杂的矢量动画等优点。有几种情形需要启用缓存。可能您总是希望通过启用缓存来提高 SWF 文件的性能；但是，某些情况下启用缓存并不能提高性能，甚至还会降低性能。本部分介绍在哪些情况下应使用缓存，以及何时使用常规显示对象。

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况。

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® 中创建的插图）。您可能会在背景上设计动画人物，这会降低动画的速度，因为背景需要持续地重新生成矢量数据。要提高性能，可以将背景显示对象的 opaqueBackground 属性设置为 true。背景将呈示为位图，可以迅速地重新绘制，以便更快地播放动画。
- 滚动文本字段：应用程序在滚动文本字段中显示大量的文本。可以将文本字段放置在您设置为可滚动的具有滚动框 （使用 scrollRect 属性）的显示对象中。这可以使指定的实例进行快速像素滚动。当用户滚动显示对象实例时， Flash Player 或 AIR 会通过将滚动的像素向上移来生成新的看得见的区域，而不是重新生成整个文本字段。
- 窗口排列秩序：应用程序具有秩序复杂的重叠窗口。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。
- Alpha 通道遮罩：当使用 Alpha 通道遮罩时，必须将 cacheAsBitmap 属性设置为 true。有关详细信息，请参阅第 158 页的 " 遮罩显示对象 "。

所有这些情况下，启用位图缓存后都通过优化矢量图来提高应用程序的响应能力和互动性。

此外，只要对显示对象应用滤镜， cacheAsBitmap 就会自动设置为 true，即使将其明确设置为 false 也是如此。如果清除了显示对象的所有滤镜，则 cacheAsBitmap 属性会返回最后设置的值。

何时避免使用位图缓存

在错误的环境中使用此功能可能会给 SWF 文件带来负面影响。使用位图缓存时，请记住下面的准则：

- 不要过度使用表面 （启用了缓存的显示对象）。每个表面使用的内存都比常规显示对象多，这意味着只在需要提高呈示性能时才启用表面。

  缓存的位图使用的内存比常规显示对象多很多。例如，如果舞台上 Sprite 实例的大小为 250 x 250 个像素，缓存它可能会使用 250 KB 内存；如果它是常规 （未缓存的） Sprite 实例，则使用 1 KB 内存。
- 避免放大缓存的表面。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 （请参阅上一段落）。
- 将表面用于通常为静态 （非动画）的显示对象实例。可以拖放或移动实例，但实例的内容不能为动画或更改太多。（动画或变化的内容更可能包含在包含动画的 MovieClip 实例或 Video 实例中。）例如，如果旋转或转换实例，实例将在表面和矢量数据之间进行变化，这种情况难于处理，并会对 SWF 文件产生负面影响。
- 如果将表面与矢量数据混在一起，将增加 Flash Player 和 AIR（有时还包括计算机）的工作量。应尽量将表面归为一组 — 例如，创建窗口应用程序时。
- 请不要缓存图形更改频繁的对象。每一次缩放、倾斜、旋转显示对象，更改 alpha 或颜色转换，移动子显示对象，或使用图形属性绘图时，位图缓存都会重绘。如果每一帧都发生这种情况，运行时必须将对象绘制为位图，然后将该位图复制到舞台 — 与仅将未缓存对象绘制到舞台相比，这会导致额外的工作量。缓存和更新频率之间的性能权衡取决于显示对象的复杂性和大小，并且只能通过测试具体内容来确定。

启用位图缓存

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要为显示对象启用位图缓存，请将它的 cacheAsBitmap 属性设置为 true：

```
mySprite.cacheAsBitmap = true;
```

将 cacheAsBitmap 属性设置为 true 后，您可能会注意到，显示对象的像素会自动与整个坐标对齐。测试 SWF 文件时，您还会注意到，在复杂矢量图像上执行的任何动画的呈示速度都快得多。

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。
- 位图分配不成功 （由于内存不足而出现的错误）。

缓存的位图转换矩阵

Adobe AIR 2.0 和更高版本 （移动配置文件）

在用于移动设备的 AIR 应用程序中，应在每次设置 cacheAsBitmap 属性的同时设置 cacheAsBitmapMatrix 属性。设置此属性可让您在不触发重新呈现的前提下，将更加丰富的转换应用到显示对象。

```
mySprite.cacheAsBitmap = true;
mySprite.cacheAsBitmapMatrix = new Matrix();
```

设置此矩阵属性后，可在不重新缓存对象的前提下，向显示对象应用以下额外转换：

- 在不发生像素贴紧的前提下进行移动或平移
- 旋转
- 缩放
- 倾斜效果

- 更改 alpha （0 至 100% 的透明度）

这些转换会直接应用到缓存的位图。

设置不透明背景颜色

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以为显示对象设置不透明背景。例如，如果 SWF 的背景中包含复杂的矢量图片，则可以将 opaqueBackground 属性设置为指定的颜色 （通常与舞台颜色相同）。将颜色指定为一个数字 （通常为十六进制的颜色值）。然后可将背景视作位图，这样有助于优化性能。

当将 cacheAsBitmap 设置为 true 并将 opaqueBackground 属性设置为指定的颜色时，opaqueBackground 属性可以使内部位图不透明而加快呈示速度。如果不将 cacheAsBitmap 设置为 true，opaqueBackground 属性将在显示对象的背景中添加一个不透明的矢量正方形形状。不会自动创建位图。

下面的示例说明了如何设置显示对象的背景以优化性能。

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

在本例中，将名为 myShape 的 Shape 的背景颜色设置为红色 (0xFF0000)。假定 Shape 实例在白色背景的舞台上包含一个绿色三角形绘图，这将在 Shape 实例的边框 （完全包含 Shape 的矩形）内显示一个绿色三角形，且空白区域为红色。

当然，如果此代码用于纯红色背景的舞台，则更合理。在其他颜色的背景上，则改为指定该颜色。例如，在白色背景的 SWF 中， opaqueBackground 属性最适合设置为 0xFFFFFF 或纯白色。

应用混合模式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

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）的颜色进行组合来生成第三个图像，所得的图像是实际在屏幕上显示的图像。图像中的每个像素值都会被使用其他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。

每个显示对象都有 blendMode 属性，可以将其设置为下列混合模式之一。以下是在 BlendMode 类中定义的常量。此外，还可以使用 String 值 （在括号中），这些值是常量的实际值。

- BlendMode.ADD ("add")：通常用于创建两个图像之间的动画变亮模糊效果。
- BlendMode.ALPHA ("alpha")：通常用于在背景上应用前景的透明度。（在 GPU 呈现下不支持。）
- BlendMode.DARKEN ("darken")：通常用于重叠类型。（在 GPU 呈现下不支持。）
- BlendMode.DIFFERENCE ("difference")：通常用于创建更多变动的颜色。
- BlendMode.ERASE ("erase")：通常用于使用前景 Alpha 剪掉 （擦除）背景的一部分。（在 GPU 呈现下不支持。）
- BlendMode.HARDLIGHT ("hardlight")：通常用于创建阴影效果。（在 GPU 呈现下不支持。）
- BlendMode.INVERT ("invert")：用于反转背景。
- BlendMode.LAYER ("layer")：用于强制为特定显示对象的预构成创建临时缓冲区。（在 GPU 呈现下不支持。）

- BlendMode.LIGHTEN ("lighten")：通常用于重叠类型。（在 GPU 呈现下不支持。）
- BlendMode.MULTIPLY ("multiply")：通常用于创建阴影和深度效果。
- BlendMode.NORMAL ("normal")：用于指定混合图像的像素值覆盖基本图像的像素值。
- BlendMode.OVERLAY ("overlay")：通常用于创建阴影效果。（在 GPU 呈现下不支持。）
- BlendMode.SCREEN ("screen")：通常用于创建亮点和镜头眩光。
- BlendMode.SHADER ("shader")：用于指定用于创建自定义混合效果的 Pixel Bender 着色器。有关使用着色器的详细信息，请参阅第 251 页的 " 使用 Pixel Bender 着色器 "。（在 GPU 呈现下不支持。）
- BlendMode.SUBTRACT ("subtract")：通常用于创建两个图像之间的动画变暗模糊效果。

调整 DisplayObject 颜色

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 ColorTransform 类的方法 (flash.geom.ColorTransform) 来调整显示对象的颜色。每个显示对象都有 transform 属性 （它是 Transform 类的实例），还包含有关应用到显示对象的各种变形的信息 （如旋转、缩放或位置的更改等）。除了有关几何变形的信息之外， Transform 类还包括 colorTransform 属性，它是 ColorTransform 类的实例，并提供访问来对显示对象进行颜色调整。要访问显示对象的颜色转换信息，可以使用如下代码：

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

创建 ColorTransform 实例后，可以通过读取其属性值来查明已应用了哪些颜色转换，也可以通过设置这些值来更改显示对象的颜色。要在进行任何更改后更新显示对象，必须将 ColorTransform 实例重新分配给 transform.colorTransform 属性。

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

使用代码设置颜色值

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ColorTransform 类的 color 属性可用于为显示对象分配具体的红、绿、蓝 (RGB) 颜色值。在下面的示例中，当用户单击名为 blueBtn 的按钮时，将使用 color 属性将名为 square 的显示对象的颜色更改为蓝色：

```
// square is a display object on the Stage.
// blueBtn, redBtn, greenBtn, and blackBtn are buttons on the Stage.

import flash.events.MouseEvent;
import flash.geom.ColorTransform;

// Get access to the ColorTransform instance associated with square.
var colorInfo:ColorTransform = square.transform.colorTransform;

// This function is called when blueBtn is clicked.
function makeBlue(event:MouseEvent):void
{
    // Set the color of the ColorTransform object.
    colorInfo.color = 0x003399;
    // apply the change to the display object
    square.transform.colorTransform = colorInfo;
}

blueBtn.addEventListener(MouseEvent.CLICK, makeBlue);
```

请注意，使用 color 属性更改显示对象的颜色时，将会完全更改整个对象的颜色，无论该对象以前是否有多种颜色。例如，如果某个显示对象包含一个顶部有黑色文本的绿色圆，将该对象的关联 ColorTransform 实例的 color 属性设置为红色阴影时，会使整个对象 （圆和文本）变为红色 （因此无法再将文本与该对象的其余部分区分开来）。

使用代码更改颜色和亮度效果

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

假设显示对象有多种颜色 （例如，数码照片），但是您不想完全重新调整对象的颜色，只想根据现有颜色来调整显示对象的颜色。这种情况下， ColorTransform 类包括一组可用于进行此类调整的乘数属性和偏移属性。乘数属性的名分别为 redMultiplier、 greenMultiplier、 blueMultiplier 和 alphaMultiplier，它们的作用像彩色照片滤镜 （或彩色太阳镜）一样，可以增强或削弱显示对象上的某些颜色。偏移属性 （redOffset、 greenOffset、 blueOffset 和 alphaOffset）可用于额外增加对象上某种颜色的值，或用于指定特定颜色可以具有的最小值。

在 “ 属性 ” 检查器上的 “ 颜色 ” 弹出菜单中选择 “ 高级 ” 时，这些乘数和偏移属性与 Flash 创作工具中影片剪辑元件可用的高级颜色设置相同。

下面的代码加载一个 JPEG 图像并为其应用颜色转换，当鼠标指针沿 x 轴和 y 轴移动时，将调整红色和绿色通道值。在本例中，因为未指定偏移值，所以屏幕上显示的每个颜色通道的颜色值将表示图像中原始颜色值的一个百分比，这意味着任何给定像素上显示的大部分红色或绿色都是该像素上红色或绿色的原始效果。

```
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.geom.Transform;
import flash.geom.ColorTransform;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// This function is called when the mouse moves over the loaded image.
function adjustColor(event:MouseEvent):void
{
    // Access the ColorTransform object for the Loader (containing the image)
    var colorTransformer:ColorTransform = loader.transform.colorTransform;

    // Set the red and green multipliers according to the mouse position.
    // The red value ranges from 0% (no red) when the cursor is at the left
    // to 100% red (normal image appearance) when the cursor is at the right.
    // The same applies to the green channel, except it's controlled by the
    // position of the mouse in the y axis.
    colorTransformer.redMultiplier = (loader.mouseX / loader.width) * 1;
    colorTransformer.greenMultiplier = (loader.mouseY / loader.height) * 1;

    // Apply the changes to the display object.
    loader.transform.colorTransform = colorTransformer;
}

loader.addEventListener(MouseEvent.MOUSE_MOVE, adjustColor);
```

旋转对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 rotation 属性可以旋转显示对象。可以通过读取此值来了解是否旋转了某个对象，如果要旋转该对象，可以将此属性设置为一个数字 （以度为单位），表示要应用于该对象的旋转量。例如，下面的代码行将名为 square 的对象旋转 45 度 （一整周旋转的 1/8）：

```
square.rotation = 45;
```

或者，也可以使用转换矩阵来旋转显示对象，第 175 页的 “ 使用几何结构 ” 中对此进行了介绍。

淡化对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以通过控制显示对象的透明度来使显示对象部分透明 （或完全透明），也可以通过更改透明度来使对象淡入或淡出。DisplayObject 类的 alpha 属性用于定义显示对象的透明度（更确切地说是不透明度）。可以将 alpha 属性设置为介于 0 和 1 之间的任何值，其中 0 表示完全透明， 1 表示完全不透明。例如，当使用鼠标单击名为 myBall 的对象时，下面的代码行将使该对象变得部分 (50%) 透明：

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

还可以使用通过 ColorTransform 类提供的颜色调整来更改显示对象的透明度。有关详细信息，请参阅第 155 页的 " 调整 DisplayObject 颜色 "。

遮罩显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以通过将一个显示对象用作遮罩来创建一个孔洞，透过该孔洞使另一个显示对象的内容可见。

定义遮罩

若要指示一个显示对象将是另一个显示对象的遮罩，请将遮罩对象设置为被遮罩的显示对象的 mask 属性：

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

被遮罩的显示对象显示在用作遮罩的显示对象的全部不透明区域之内。例如，下面的代码将创建一个包含 100 x 100 个像素的红色正方形的 Shape 实例和一个包含半径为 25 个像素的蓝色圆的 Sprite 实例。单击圆时，它被设置为正方形的遮罩，所以显示的正方形部分只是由圆完整部分覆盖的那一部分。换句话说，只有红色圆可见。

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.Shape;

// Draw a square and add it to the display list.
var square:Shape = new Shape();
square.graphics.lineStyle(1, 0x000000);
square.graphics.beginFill(0xff0000);
square.graphics.drawRect(0, 0, 100, 100);
square.graphics.endFill();
this.addChild(square);

// Draw a circle and add it to the display list.
var circle:Sprite = new Sprite();
circle.graphics.lineStyle(1, 0x000000);
circle.graphics.beginFill(0x0000ff);
circle.graphics.drawCircle(25, 25, 25);
circle.graphics.endFill();
this.addChild(circle);

function maskSquare(event:MouseEvent):void
{
    square.mask = circle;
    circle.removeEventListener(MouseEvent.CLICK, maskSquare);
}

circle.addEventListener(MouseEvent.CLICK, maskSquare);
```

用作遮罩的显示对象可拖动、设置动画，并可动态调整大小，可以在单个遮罩内使用单独的形状。遮罩显示对象不一定需要添加到显示列表中。但是，如果希望在缩放舞台时也缩放遮罩对象，或者如果希望支持用户与遮罩对象的交互 （如用户控制的拖动和调整大小），则必须将遮罩对象添加到显示列表中。只要遮罩对象已添加到显示列表中，显示对象的实际 z 索引 （从前到后的顺序）就无关紧要了。（除了显示为遮罩对象外，遮罩对象将不会出现在屏幕上。）如果遮罩对象是包含多个帧的一个 MovieClip 实例，则遮罩对象会沿其时间轴播放所有帧，如果没有用作遮罩对象，也会出现同样的情况。通过将 mask 属性设置为 null 可以删除遮罩：

```
// remove the mask from mySprite
mySprite.mask = null;
```

不能使用一个遮罩对象来遮罩另一个遮罩对象。不能设置遮罩显示对象的 alpha 属性。只有填充可用于作为遮罩的显示对象中；笔触都会被忽略。

AIR 2

如果通过设置 cacheAsBitmap 和 cacheAsBitmapMatrix 属性来缓存被遮罩显示对象，则遮罩必须是被遮罩显示对象的子级。类似地，如果被遮罩显示对象是被缓存的显示对象容器的子项，则遮罩和显示对象都必须是该容器的子项。如果被遮罩对象是多个缓存显示对象容器的子项，则遮罩必须是距离显示列表中被遮罩对象最近的缓存容器的子项。

关于遮蔽设备字体

您可以使用显示对象遮罩用设备字体设置的文本。当使用显示对象遮罩用设备字体设置的文本时，遮罩的矩形边框会用作遮罩形状。也就是说，如果为设备字体文本创建了非矩形的显示对象遮罩，则 SWF 文件中显示的遮罩将是遮罩的矩形边框的形状，而不是遮罩本身的形状。

Alpha 通道遮罩

如果遮罩显示对象和被遮罩的显示对象都使用位图缓存，则支持 Alpha 通道遮罩，如下所示：

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

例如， Alpha 通道遮罩的一个应用是对遮罩对象使用应用于被遮罩显示对象之外的滤镜。

在下面的示例中，将一个外部图像文件加载到舞台上。该图像 （更确切地说，是加载图像的 Loader 实例）将是被遮罩的显示对象。渐变椭圆 （中心为纯黑色，边缘淡变为透明）绘制在图像上；这就是 Alpha 遮罩。两个显示对象都打开了位图缓存。椭圆设置为图像的遮罩，然后使其可拖动。

```
// This code assumes it's being run within a display object container
// such as a MovieClip or Sprite instance.

import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Sprite;
import flash.geom.Matrix;
import flash.net.URLRequest;

// Load an image and add it to the display list.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create a Sprite.
var oval:Sprite = new Sprite();
// Draw a gradient oval.
var colors:Array = [0x000000, 0x000000];
var alphas:Array = [1, 0];
var ratios:Array = [0, 255];
```

```
var matrix:Matrix = new Matrix();
matrix.createGradientBox(200, 100, 0, -100, -50);
oval.graphics.beginGradientFill(GradientType.RADIAL,
                                colors,
                                alphas,
                                ratios,
                                matrix);
oval.graphics.drawEllipse(-100, -50, 200, 100);
oval.graphics.endFill();
// add the Sprite to the display list
this.addChild(oval);

// Set cacheAsBitmap = true for both display objects.
loader.cacheAsBitmap = true;
oval.cacheAsBitmap = true;
// Set the oval as the mask for the loader (and its child, the loaded image)
loader.mask = oval;

// Make the oval draggable.
oval.startDrag(true);
```

对象动画

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

动画是使内容移动或者使内容随时间发生变化的过程。脚本动画是视频游戏的基础部分，通常用于将优美、有用的交互线索添加到其他应用程序中。

脚本动画的基本概念是变化一定要发生，而且变化一定要分时间逐步完成。使用常见的循环语句，可以很容易地在 ActionScript 中使内容重复。但是，在更新显示之前，循环将遍历其所有迭代。要创建脚本动画，需要编写 ActionScript，它随时间重复执行某个动作，每次运行时还更新屏幕。

例如，假设要创建一个简单的动画，如使球沿着屏幕运动。 ActionScript 提供了一个用于跟踪时间和相应更新屏幕的简单机制，这意味着您可以编写代码，使球每次移动一点点，直到球到达目标为止。每次移动后，屏幕都会更新，从而使跨舞台的运动在查看器中可见。

从实用的观点来看，让脚本动画与 SWF 文件的帧速率同步 （换句话说，每次显示或要显示新帧时都产生一个动画变化）才有意义，因为帧速率定义了 Flash Player 或 AIR 更新屏幕的频率。每个显示对象都有 enterFrame 事件，它根据 SWF 文件的帧速率来调度，即每帧一个事件。创建脚本动画的大多数开发人员都使用 enterFrame 事件作为一种方法来创建随时间重复的动作。可以编写代码以侦听 enterFrame 事件，每一帧都让动画球移动一定的量，当屏幕更新时 （每一帧），将会在新位置重新绘制该球，从而产生了运动。

注：另一种随时间重复执行某个动作的方法是使用 Timer 类。每次过了指定的时间时，Timer 实例都会触发事件通知。可以编写通过处理 Timer 类的 timer 事件来执行动画的代码，将时间间隔设置为一个很小的值 （几分之几秒）。有关使用 Timer 类的详细信息，请参阅第 3 页的 “ 控制时间间隔 ”。

在下面的示例中，将在舞台上创建一个名为 circle 的圆 Sprite 实例。当用户单击圆时，脚本动画序列开始，从而使 circle 淡化（其 alpha 属性值减少），直到完全透明：

```
import flash.display.Sprite;
import flash.events.Event;
import flash.events.MouseEvent;

// draw a circle and add it to the display list
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0x990000);
circle.graphics.drawCircle(50, 50, 50);
circle.graphics.endFill();
addChild(circle);

// When this animation starts, this function is called every frame.
// The change made by this function (updated to the screen every
// frame) is what causes the animation to occur.
function fadeCircle(event:Event):void
{
    circle.alpha -= .05;

    if (circle.alpha <= 0)
    {
        circle.removeEventListener(Event.ENTER_FRAME, fadeCircle);
    }
}

function startAnimation(event:MouseEvent):void
{
    circle.addEventListener(Event.ENTER_FRAME, fadeCircle);
}

circle.addEventListener(MouseEvent.CLICK, startAnimation);
```

当用户单击圆时，将函数 fadeCircle() 订阅为 enterFrame 事件的侦听器，这意味着每一帧都会开始调用一次该函数。通过更改 circle 的 alpha 属性，该函数会淡化圆，因此对于每个帧，圆的 alpha 都会减少 .05 (5%) 并会更新屏幕。最后，当 alpha 值为 0（circle 完全透明）时， fadeCircle() 函数作为事件侦听器将被删除，从而结束动画。

以上代码还可用来创建动画运动而不是淡化。通过用不同属性替换函数中表示 enterFrame 事件侦听器的 alpha，就可获得该属性的动画效果。例如，将以下行

```
circle.alpha -= .05;
```

更改为以下代码

```
circle.x += 5;
```

将获得 x 属性的动画效果，使圆移到舞台的右侧。当到达需要的 x 坐标时，通过更改结束动画的条件就可结束动画 （即取消订阅 enterFrame 侦听器）。

舞台方向

AIR 2.0 和更高版本

移动设备通常会重新定向用户界面，以便在用户旋转设备时保持垂直显示。如果您在应用程序中启用了自动方向，则设备会自动保持适当的显示方向，但您需要自行确保舞台的高宽比发生更改后，内容的显示效果仍可接受。如果禁用自动方向，则除非您手动更改显示方向，否则设备的显示方向将保持不变。

AIR 应用程序可在各种不同的移动设备和操作系统中运行。不同操作系统，甚至不同设备上的同一操作系统的基本方向行为可能有所不同。一种适用于所有设备和操作系统的简单设计策略是启用自动方向并侦听 Stage 对象的 resize 事件，以确定何时需要刷新应用程序布局。

或者，如果您的应用程序仅支持纵向高宽比或仅支持横向高宽比，可以禁用自动方向，并在 AIR 应用程序描述符中设置支持的高宽比。此设计策略可提供一致的行为并可针对所选高宽比选择“最佳的”方向。例如，如果指定横向高宽比，则选择的方向适合使用横向模式滑出键盘的设备。

获取当前舞台方向和高宽比

所报告的方向是相对于设备的正常位置而言。大多数设备都具有明确的垂直位置。该位置将被视为默认 方向。其他可能的方向分别为：向左旋转、向右旋转 和向下翻转。 StageOrientation 类定义在设置或比较方向值时使用的字符串常量。

Stage 类定义两个用于报告方向的属性：

- Stage.deviceOrientation — 报告设备相对于默认位置的物理方向。

  注：在某些情况下， deviceOrientation 可能会不可用，例如应用程序初次启动或设备平放时。在这些情况下，设备方向将报告为未知。

- Stage.orientation — 报告舞台相对于默认方向的方向。启用自动方向后，当设备保持垂直时，舞台将向相反方向旋转。因此，orientation 属性报告的右和左的位置将与 deviceOrientation 属性报告的位置相反。例如，如果 deviceRotation 报告向右旋转，则 orientation 将报告向左旋转。

舞台的高宽比可以通过简单的比较舞台当前的宽度和高度推导出来。

```
var aspect:String = this.stage.stageWidth >= this.stage.stageHeight ? StageAspectRatio.LANDSCAPE :
StageAspectRatio.PORTRAIT;
```

自动方向

启用自动方向后，当用户旋转其设备时，操作系统会重新定向整个用户界面，包括系统任务栏和应用程序。因此，舞台的高宽比将会从纵向变为横向，或者从横向变为纵向。高宽比发生更改时，舞台尺寸也会更改。

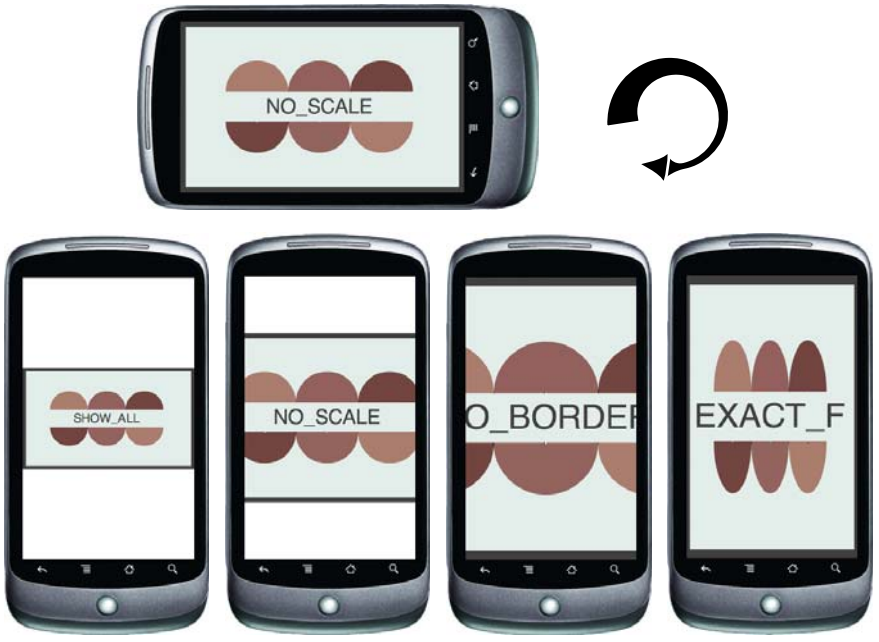
通过将 Stage 对象的 autoOrients 属性设置为 true 或 false，可在运行时启用或禁用自动方向。可以在 AIR 应用程序描述符中使用 <autoOrients> 元素来设置此属性的初始值。（请注意，对于 AIR 2.6 之前的版本， autoOrients 是一个只读属性，并且只能在应用程序描述符中设置。）

如果您指定一个横向或纵向的高宽比，同时启用了自动方向， AIR 将自动方向限制为指定的高宽比。

舞台尺寸更改

舞台尺寸发生更改时，舞台内容将按照 Stage 对象的 scaleMode 和 align 属性的指定进行缩放和重新定位。在大多数情况下，依赖由 Stage 对象的 scaleMode 设置提供的自动行为不会产生理想的效果。您必须重新布局或重新绘制图形和组件以支持多种高宽比。（提供灵活的布局逻辑也意味着您的应用程序可以更好地适应具有不同屏幕大小和高宽比的各种设备。）

下图演示了在旋转典型移动设备时，不同 scaleMode 设置的效果：



从横向高宽比旋转到纵向高宽比

该图演示了从横向高宽比旋转到纵向高宽比时不同缩放模式发生的缩放行为。从纵向旋转到横向会引起一系列相似的效果。

方向更改事件

Stage 对象会调度两种类型的事件，您可以将其用于检测方向更改并做出响应。启用自动方向时，会调度舞台 resize 和 orientationChange 事件。

如果您依赖自动方向来保持垂直显示，则 resize 事件是您的最佳选择。当舞台调度 resize 事件时，您的内容会根据需要重新布局或重新绘制。仅当舞台缩放模式设置为 noScale 时，才会调度 resize 事件。

orientationChange 事件也可以用于检测方向更改。仅当自动方向启用时，才会调度 orientationChange 事件。

注：在某些移动平台上，舞台会在调度 resize 或 orientationChange 事件之前调度一个可取消的 orientationChanging 事件。由于并非所有平台都支持该事件，因此请避免依赖该事件。

手动方向

AIR 2.6 和更高版本

可以使用舞台的 setOrientation() 或 setAspectRatio() 方法控制舞台方向。

设置舞台方向

可以使用 Stage 对象的 setOrientation() 方法在运行时设置舞台方向。使用 StageOrientation 类指定的字符串常量指定所需的方向：

```
this.stage.setOrientation( StageOrientation.ROTATED_RIGHT );
```

并不是所有设备和操作系统都支持每个可能的方向。例如， Android 2.2 在纵向标准设备上不支持以编程方式选择向左旋转方向，并且根本不支持向下翻转方向。舞台的 supportedOrientations 属性提供了一个可以传递到 setOrientation() 方法的方向列表：

```
var orientations:Vector.<String> = this.stage.supportedOrientations;
for each( var orientation:String in orientations )
{
    trace( orientation );
}
```

设置舞台高宽比

如果您非常在意舞台的高宽比，则可以将高宽比设置为纵向或横向。可以使用舞台的 setAspectRatio() 方法在 AIR 应用程序描述符中或在运行时设置高宽比：

```
this.stage.setAspectRatio( StageAspectRatio.LANDSCAPE );
```

运行时为指定的高宽比选择两个可能的方向之一。这可能与当前设备的方向不匹配。例如，默认的方向选定为首选上下翻转方向 （AIR 3.2 及早期版本），适用于滑动式键盘的方向选定为首选反向。

（**AIR 3.3** 及更高版本）从 AIR 3.3 （SWF 版本 16）开始，您也可以使用 StageAspectRatio.ANY 常数。如果 Stage.autoOrients 设置为 true，而且您调用了 setAspectRatio(StageAspectRatio.ANY)，您的应用程序能够将方向重新调整为所有方向 （横向 - 左、横向 - 右、纵向以及纵向和上下翻转）。同样是 AIR 3.3 的新功能，高宽比保持不变，进一步旋转设备限制为指定方向。

示例：将舞台方向设置为与设备方向相匹配

下面的示例演示了更新舞台方向以匹配当前设备方向的功能。舞台的 deviceOrientation 属性指示设备的物理方向，即使关闭了自动方向也会指示。

```
function refreshOrientation( theStage:Stage ):void
{
    switch ( theStage.deviceOrientation )
    {
        case StageOrientation.DEFAULT:
            theStage.setOrientation( StageOrientation.DEFAULT );
            break;
        case StageOrientation.ROTATED_RIGHT:
            theStage.setOrientation( StageOrientation.ROTATED_LEFT );
            break;
        case StageOrientation.ROTATED_LEFT:
            theStage.setOrientation( StageOrientation.ROTATED_RIGHT );
            break;
        case StageOrientation.UPSIDE_DOWN:
            theStage.setOrientation( StageOrientation.UPSIDE_DOWN );
            break;
        default:
            //No change
    }
}
```

方向更改是异步执行的。可以侦听舞台调度的 orientationChange 事件以检测更改的完成情况。如果某个设备不支持某个方向，setOrientation() 调用将失败且不引发错误。

动态加载显示内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以将下列任何外部显示资源加载到 ActionScript 3.0 应用程序中：

- 在 ActionScript 3.0 中创作的 SWF 文件 — 此文件可以是 Sprite、MovieClip 或扩展 Sprite 的任何类。在 iOS 上的 AIR 应用程序中，只能加载不包含 ActionScript 字节代码的 SWF 文件。这意味着可以加载包含嵌入数据 （如图像和声音）的 SWF 文件，但不能加载包含可执行代码的 SWF 文件。

- 图像文件 — 包括 JPG、 PNG 和 GIF 文件。
- AVM1 SWF 文件 — 用 ActionScript 1.0 或 2.0 编写的 SWF 文件。（在移动 AIR 应用程序中不受支持）

使用 Loader 类可以加载这些资源。

加载显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Loader 对象用于将 SWF 文件和图形文件加载到应用程序中。 Loader 类是 DisplayObjectContainer 类的子类。 Loader 对象在其显示列表中只能包含一个子显示对象，该显示对象表示它加载的 SWF 或图形文件。如下面的代码所示，在显示列表中添加 Loader 对象时，还可以在加载后将加载的子显示对象添加到显示列表中：

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

加载 SWF 文件或图像后，即可将加载的显示对象移到另一个显示对象容器中，如本示例中的 container DisplayObjectContainer 对象：

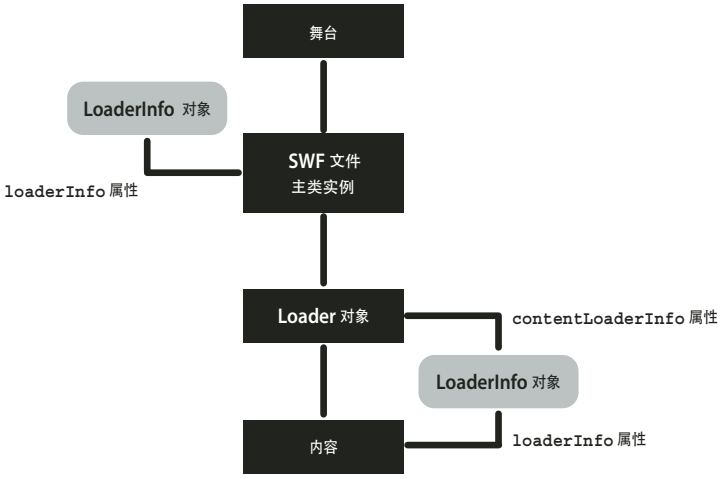
```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(event:Event):void
{
    container.addChild(pictLdr.content);
}
```

监视加载进度

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

文件开始加载后，就创建了 LoaderInfo 对象。 LoaderInfo 对象用于提供加载进度、加载者和被加载者的 URL、媒体的字节总数及媒体的标称高度和宽度等信息。 LoaderInfo 对象还调度用于监视加载进度的事件。

下图说明 LoaderInfo 对象的不同用途 — 用于 SWF 文件的主类的实例、用于 Loader 对象以及用于由 Loader 对象加载的对象：



可以将 LoaderInfo 对象作为 Loader 对象和加载的显示对象的属性进行访问。加载一开始，就可以通过 Loader 对象的 contentLoaderInfo 属性访问 LoaderInfo 对象。加载完显示对象后，也可以通过显示对象的 loaderInfo 属性，将 LoaderInfo 对象作为已加载显示对象的属性进行访问。已加载显示对象的 loaderInfo 属性是指与 Loader 对象的 contentLoaderInfo 属性相同的 LoaderInfo 对象。换句话说，LoaderInfo 对象是加载的对象与加载它的 Loader 对象之间（加载者和被加载者之间）的共享对象。

要访问加载的内容的属性，需要在 LoaderInfo 对象中添加事件侦听器，如下面的代码所示：

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
    var content:Sprite = event.target.content;
    content.scaleX = 2;
}
```

有关详细信息，请参阅第 106 页的 “ 处理事件 ”。

指定加载上下文

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过 Loader 类的 load() 或 loadBytes() 方法将外部文件加载到 Flash Player 或 AIR 中时，可以选择指定 context 参数。此参数是一个 LoaderContext 对象。

LoaderContext 类包括三个属性，用于定义如何使用加载的内容的上下文：

- checkPolicyFile：仅当加载图像文件 （不是 SWF 文件）时才会使用此属性。如果将此属性设置为 true， Loader 将检查策略文件的原始服务器 （请参阅第 901 页的 “ 网站控制 （策略文件） ”）。只有内容的来源域不是包含 Loader 对象的 SWF

文件所在的域时才需要此属性。如果服务器授予 Loader 域权限，Loader 域中 SWF 文件的 ActionScript 就可以访问加载图像中的数据；换句话说，可以使用 BitmapData.draw() 命令访问加载的图像中的数据。

请注意，来自 Loader 对象所在域以外的其他域的 SWF 文件可以通过调用 Security.allowDomain() 来允许特定的域。

- securityDomain：仅当加载 SWF 文件 （不是图像）时才会使用此属性。如果 SWF 文件所在的域与包含 Loader 对象的文件所在的域不同，则指定此属性。指定此选项时， Flash Player 将检查策略文件是否存在，如果存在，来自跨策略文件中允许的域的 SWF 文件可以对加载的 SWF 内容执行跨脚本操作。可以将 flash.system.SecurityDomain.currentDomain 指定为此参数。
- applicationDomain: 仅当加载使用 ActionScript 3.0 编写的 SWF 文件（不是图像或使用 ActionScript 1.0 或 2.0 编写的 SWF 文件）时才会使用此属性。加载文件时，通过将 applicationDomain 参数设置为 flash.system.ApplicationDomain.currentDomain，可以指定将该文件包括在与 Loader 对象相同的应用程序域中。通过将加载的 SWF 文件放在同一个应用程序域中，可以直接访问它的类。如果要加载的 SWF 文件中包含嵌入的媒体，这会很有帮助，您可以通过其关联的类名访问嵌入的媒体。有关详细信息，请参阅第 123 页的 “ 使用应用程序域 ”。

下面的示例在从另一个域加载位图时检查策略文件：

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

下面的示例在从另一个域加载 SWF 时检查策略文件，以便将该文件与 Loader 对象放在同一个安全沙箱中。此外，该代码还将加载的 SWF 文件中的类添加到与 Loader 对象的类相同的应用程序域中：

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 LoaderContext 类。

在 AIR for iOS 中加载 SWF 文件

Adobe AIR 3.6 和更高版本，仅针对 **iOS**

在 iOS 设备上，对于在运行时加载和编译代码存在限制。由于这些限制，将外部 SWF 文件加载到您的应用程序中势必会有一些不同：

- 所有包含 ActionScript 代码的 SWF 文件都必须包括在应用程序包中。不能从外部源 （如通过网络）加载包含代码的 SWF。在对应用程序进行打包时，对于 iOS 设备，应用程序包中所有 SWF 文件中的所有 ActionScript 代码都将编译成本机代码。
- 您不能加载 SWF 文件，然后卸载它再重新加载。如果这样做，会发生错误。
- 加载到内存中然后又卸载的行为与在桌面平台上这样操作的结果相同。如果加载 SWF 文件然后卸载它，包含在该 SWF 中的所有可视资源都将从内存中卸载掉。不过，对所加载 SWF 中的 ActionScript 类的任何类引用将保留在内存中，可以在 ActionScript 代码中访问这些类引用。
- 加载的所有 SWF 文件必须与主 SWF 文件使用相同的应用程序域。这并非默认行为，因此对于每个要加载的 SWF，您必须创建一个 LoaderContext 对象来指定主应用程序域，并将该 LoaderContext 对象传递给 Loader.load() 方法调用。如果要加载的 SWF 所处的应用程序域与主 SWF 应用程序域不同，便会发生错误。即使加载的 SWF 只包含可视资源而不包含 ActionScript 代码，也是这样。

  下例中的代码将一个 SWF 从应用程序包中加载到主 SWF 应用程序域中：

```
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("swfs/SecondarySwf.swf");
var loaderContext:LoaderContext = new LoaderContext(false, ApplicationDomain.currentDomain, null);
loader.load(url, loaderContext);
```

对于只包含资源而不包含代码的 SWF 文件，可以从应用程序包加载，也可以通过网络加载。无论哪一种情况，都必须仍将 SWF 文件加载到主应用程序域中。

对于 AIR 3.6 之前的版本，在编译过程期间，所有代码都是从非主应用程序 SWF 中去除。只包含可视资源的 SWF 文件可以包括在应用程序包中并在运行时加载，但对于包含代码的 SWF 不是这样。如果要加载的 SWF 包含 ActionScript 代码，便会发生错误。该错误会导致应用程序中出现一个 “ 未编译的 ActionScript” 错误对话框。

另请参见

在 iOS 上的 AIR 应用程序中打包并加载多个 SWF

使用 ProLoader 和 ProLoaderInfo 类

Flash Player 9 和更高版本、 **Adobe AIR 1.0** 和更高版本，并且需要 **Flash Professional CS5.5**

为了帮助预加载运行时共享库 (RSL)，Flash Professional CS5.5 引入了 fl.display.ProLoader 和 fl.display.ProLoaderInfo 类。这些类会镜像 flash.display.Loader 和 flash.display.LoaderInfo 类，但提供了更加一致的加载体验。

特别是， ProLoader 可帮助您加载使用 Text Layout Framework (TLF) 执行 RSL 预加载的 SWF 文件。在运行时，预加载其他 SWF 文件或 SWZ 文件（例如 TLF）的 SWF 文件需要仅供内部使用的 SWF 包装文件。SWF 包装文件会增加调用结构的复杂性，并可能产生不必要的行为。 ProLoader 可以消除这种复杂关系，它可以像加载普通 SWF 文件那样加载这些文件。ProLoader 类使用的解决方法对于用户来说是透明的，不需要在 ActionScript 中执行任何特殊处理。此外， ProLoader 还可以正确加载普通 SWF 内容。

在 Flash Professional CS 5.5 和更高版本中，您可以安全地将所有 Loader 类替换为 ProLoader 类。然后，将您的应用程序导出到 Flash Player 10.2 或更高版本，以便 ProLoader 能够访问所需的 ActionScript 功能。您也可以在面向支持 ActionScript 3.0 的 Flash Player 早期版本的应用程序中使用 ProLoader。但是只有在 Flash Player 10.2 或更高版本中才能充分利用 ProLoader 的强大功能。当您在 Flash Professional CS5.5 或更高版本中使用 TLF 时，请始终使用 ProLoader。在 Flash Professional 之外的环境中，不需要使用 ProLoader。

重要说明：对于在 Flash Professional CS5.5 和更高版本中发布的 SWF 文件，您可以始终使用 fl.display.ProLoader 和 fl.display.ProLoaderInfo 类，而不要使用 flash.display.Loader 和 flash.display.LoaderInfo。

ProLoader 类解决的问题

ProLoader 类解决了此前 Loader 类无法处理的问题。这些问题是在对 TLF 库执行 RSL 预加载的过程中产生的。具体来说，在使用 Loader 对象加载其他 SWF 文件的 SWF 文件中会碰到这些问题。解决的问题包括：

- 加载文件和被加载文件之间的脚本处理无法按预期方式执行。 ProLoader 类会自动将加载 SWF 文件设置为被加载 SWF 文件的父项。因此，来自加载 SWF 文件的通信会直接转到被加载 SWF 文件。
- **SWF** 应用程序必须主动管理加载过程。要进行主动管理，需要实现额外的事件，例如 added、 removed、 addedToStage 和 removedFromStage。如果您的应用程序是面向 Flash Player 10.2 或更高版本，则 ProLoader 可消除这些额外的工作。

更新代码，使用 **ProLoader** 来替代 **Loader**

由于 ProLoader 会镜像 Loader 类，您可以轻松地在代码中切换这两个类。下面的示例显示了如何更新现有代码以使用新类：

```
import flash.display.Loader;
import flash.events.Event;
var l:Loader = new Loader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

可将此代码更新为使用 ProLoader，如下所示：

```
import fl.display.ProLoader;
import flash.events.Event;
var l:ProLoader = new ProLoader();

addChild(l);
l.contentLoaderInfo.addEventListener(Event.COMPLETE, loadComplete);
l.load("my.swf");
function loadComplete(e:Event) {
    trace('load complete!');
}
```

显示对象示例：SpriteArranger

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

SpriteArranger 示例应用程序构建在 《学习 ActionScript 3.0》中单独介绍的几何形状示例应用程序的基础上。

SpriteArranger 范例应用程序演示说明了处理显示对象的许多概念：

- 扩展显示对象类
- 在显示列表中添加对象
- 分层显示对象和使用显示对象容器
- 响应显示对象事件
- 使用显示对象的属性和方法

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可在文件夹 Examples/SpriteArranger 中找到 SpriteArranger 应用程序文件。该应用程序包含以下文件：

文件	说明
SpriteArranger.mxml 或 SpriteArranger.fla	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/SpriteArranger/CircleSprite.as	定义一种 Sprite 对象的类，该种对象在屏幕上呈示为圆。
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	定义画布的类，画布是包含 GeometricSprite 对象的显示对象容器。
com/example/programmingas3/SpriteArranger/SquareSprite.as	定义一种 Sprite 对象的类，该对象在屏幕上呈示为正方形。

文件	说明
com/example/programmingas3/SpriteArranger/TriangleSprite.as	定义一种 Sprite 对象的类，该对象在屏幕上呈示为三角形。
com/example/programmingas3/SpriteArranger/GeometricSprite.as	扩展 Sprite 对象的类，用于定义屏幕形状。CircleSprite、 SquareSprite 和 TriangleSprite 都扩展此类。
com/example/programmingas3/geometricshapes/IGeometricShape.as	一个基接口，用于定义要由所有几何形状类实现的方法。
com/example/programmingas3/geometricshapes/IPolygon.as	一个接口，用于定义要由具有多条边的几何形状类实现的方法。
com/example/programmingas3/geometricshapes/RegularPolygon.as	一种几何形状，这种几何形状的边长相等，并且这些边围绕形状中心对称分布。
com/example/programmingas3/geometricshapes/Circle.as	一种用于定义圆的几何形状。
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	RegularPolygon 的子类，用于定义所有边长相等的三角形。
com/example/programmingas3/geometricshapes/Square.as	RegularPolygon 的子类，用于定义所有四条边相等的矩形。
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	包含 " 工厂方法 " 的一个类，用于创建给定了形状类型和尺寸的形状。

定义 SpriteArranger 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用户可以使用 SpriteArranger 应用程序在屏幕 " 画布 " 上添加各种显示对象。

DrawingCanvas 类用于定义绘制区 （一种显示对象容器），用户可以在其中添加屏幕形状。这些屏幕形状是 GeometricSprite 类的其中一个子类的实例。

DrawingCanvas 类

在 Flex 中，添加到 Container 对象的所有子显示对象都必须属于源自 mx.core.UIComponent 类的类。此应用程序将 DrawingCanvas 类的实例添加为 mx.containers.VBox 对象的子级，该对象在 SpriteArranger.mxml 文件的 MXML 代码中定义。此继承在 DrawingCanvas 类声明中定义，如下所示：

```
public class DrawingCanvas extends UIComponent
```

UIComponent 类继承自 DisplayObject、DisplayObjectContainer 和 Sprite 类，DrawingCanvas 类中的代码使用这些类的方法和属性。

DrawingCanvas 类扩展了 Sprite 类，此继承是在 DrawingCanvas 类声明中定义的，如下所示：

```
public class DrawingCanvas extends Sprite
```

Sprite 类是 DisplayObjectContainer 和 DisplayObject 类的子类， DrawingCanvas 类使用这些类的方法和属性。

DrawingCanvas() 构造函数方法设置 Rectangle 对象 bounds，它是以后在绘制画布轮廓时使用的属性。然后调用 initCanvas() 方法，如下所示：

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

如下面的示例所示， initCanvas() 方法用于定义 DrawingCanvas 对象的各种属性，这些属性作为参数传递给构造函数：

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

initCanvas() 方法随后调用 drawBounds() 方法，后者使用 DrawingCanvas 类的 graphics 属性绘制画布。graphics 属性继承自 Shape 类

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
                       bounds.top - 1,
                       bounds.width + 2,
                       bounds.height + 2);
this.graphics.endFill();
```

DrawingCanvas 类的下列附加方法是根据用户与应用程序的交互进行调用的：

- addShape() 和 describeChildren() 方法，在第 171 页的 " 在画布上添加显示对象 " 中介绍
- moveToBack()、 moveDown()、 moveToFront() 和 moveUp() 方法，在第 173 页的 " 重新排列显示对象层 " 中介绍
- onMouseUp() 方法，在第 173 页的 " 单击并拖动显示对象 " 中介绍

GeometricSprite 类及其子类

用户在画布上可以添加的每个显示对象都是 GeometricSprite 类以下某个子类的实例：

- CircleSprite
- SquareSprite
- TriangleSprite

GeometricSprite 类扩展了 flash.display.Sprite 类：

```
public class GeometricSprite extends Sprite
```

GeometricSprite 类包括所有 GeometricSprite 对象所共有的一些属性。这些属性是根据传递到构造函数的参数，在该函数中设置的。例如：

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

GeometricSprite 类的 geometricShape 属性用于定义 IGeometricShape 接口，该接口定义形状的数学属性，但不定义其可视属性。《学习 ActionScript 3.0》中介绍的 GeometricShapes 示例应用程序中定义了实现 IGeometricShape 接口的类。

GeometricSprite 类用于定义 drawShape() 方法，该方法在 GeometricSprite 的各子类的覆盖定义中已进一步精确定义。有关详细信息，请参阅后面的 " 在画布上添加显示对象 " 一节。

GeometricSprite 类还提供下列方法：

- onMouseDown() 和 onMouseUp() 方法，在第 173 页的 " 单击并拖动显示对象 " 中介绍
- showSelected() 和 hideSelected() 方法，在第 173 页的 " 单击并拖动显示对象 " 中介绍

在画布上添加显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当用户单击 " 添加形状 " 按钮时，应用程序将调用 DrawingCanvas 类的 addShape() 方法。它通过调用其中一个 GeometricSprite 子类的相应构造函数来实例化新的 GeometricSprite，如下面的示例所示：

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;

        case "Square":
            newShape = new SquareSprite(len);
            break;

        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

每个构造函数方法都调用 drawShape() 方法，该方法使用类 （继承自 Sprite 类）的 graphics 属性来绘制相应的矢量图形。例如， CircleSprite 类的 drawShape() 方法包括下列代码：

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

addShape() 函数的倒数第二行用于设置显示对象 （继承自 DisplayObject 类）的 alpha 属性，所以画布上添加的每个显示对象都有一点儿透明，这样用户就可看见它后面的对象。

addChild() 方法的最后一行用于在 DrawingCanvas 类实例的子级列表中添加新的显示对象，该实例已经在显示列表中。这样会使新的显示对象出现在舞台上。

应用程序界面上包括两个文本字段 selectedSpriteTxt 和 outputTxt。这些文本字段的文本属性由已添加到画布中或由用户选择的 GeometricSprite 对象的信息更新。 GeometricSprite 类通过覆盖 toString() 方法来处理这种信息报告任务，如下所示：

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

shapeType 属性设置为每个 GeometricSprite 子类的构造函数方法中的相应值。例如， toString() 方法可能返回最近添加到 DrawingCanvas 实例中的 CircleSprite 实例的下列值：

```
Circle of size 50 at 0, 0
```

DrawingCanvas 类的 describeChildren() 方法通过使用 numChildren 属性 （继承自 DisplayObjectContainer 类）设置 for 循环的限制，来循环访问画布的子级列表。它会生成一个列出了每一子级的字符串，如下所示：

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

所得的字符串用于设置 outputTxt 文本字段的 text 属性。

单击并拖动显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当用户单击 GeometricSprite 实例时，应用程序将调用 onMouseDown() 事件处理函数。如下所示，此事件处理函数设置为侦听 GeometricSprite 类的构造函数中的鼠标按下事件：

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

onMouseDown() 方法随后调用 GeometricSprite 对象的 showSelected() 方法。如果是首次调用该对象的此方法，该方法将创建名为 selectionIndicator 的新 Shape 对象，并且使用 Shape 对象的 graphics 属性来绘制红色加亮矩形，如下所示：

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1, this.size + 1);
this.addChild(this.selectionIndicator);
```

如果不是首次调用 onMouseDown() 方法，该方法仅设置 selectionIndicator 形状的 visible 属性 （继承自 DisplayObject 类），如下所示：

```
this.selectionIndicator.visible = true;
```

hideSelected() 方法通过将其 visible 属性设置为 false 来隐藏以前所选对象的 selectionIndicator 形状。

onMouseDown() 事件处理函数方法还会调用 startDrag() 方法 （继承自 Sprite 类），该方法包括下列代码：

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

这样，用户就可以在由 boundsRect 矩形设置的边界内在画布各处拖动选择的对象。

当用户松开鼠标按键时，将调度 mouseUp 事件。 DrawingCanvas 的构造函数方法设置了下列事件侦听器：

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

此事件侦听器是针对 DrawingCanvas 对象设置的，而不是针对单个 GeometricSprite 对象设置的。这是因为当拖动 GeometricSprite 对象时，松开鼠标后它可能会在另一个显示对象 （另一个 GeometricSprite 对象）之后结束前景中的显示对象会收到鼠标弹起事件，但用户正在拖动的显示对象则不会收到。在 DrawingCanvas 对象中添加侦听器可以确保始终处理该事件。

onMouseUp() 方法调用 GeometricSprite 对象的 onMouseUp() 方法，后者又调用 GeometricSprite 对象的 stopDrag() 方法。

重新排列显示对象层

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序用户界面上包括标有 “ 后移 ”、 “ 下移 ”、 “ 上移 ” 和 “ 移到最前 ” 的按钮。当用户单击其中一个按钮时，应用程序将调用 DrawingCanvas 类的相应方法：moveToBack()、 moveDown()、 moveUp() 或 moveToFront()。例如， moveToBack() 方法包括下面的代码：

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

该方法使用 setChildIndex() 方法（继承自 DisplayObjectContainer 类）确定显示对象位置在 DrawingCanvas 实例 (this) 的子级列表中的索引位置 0 处。

moveDown() 方法的作用类似，不同的是它将显示对象在 DrawingCanvas 实例的子级列表中的索引位置减少 1：

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

moveUp() 和 moveToFront() 方法的作用与 moveToBack() 和 moveDown() 方法类似。

第 11 章：使用几何结构

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

flash.geom 包中包含用于定义几何对象 （如，点、矩形和转换矩阵）的类。您可使用这些类来定义在其他类中使用的对象的属性。

更多帮助主题

flash.geom 包

几何结构基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

flash.geom 包中包含用于定义几何对象 （如，点、矩形和转换矩阵）的类。这些类本身并不一定提供功能，但它们用于定义在其他类中使用的对象的属性。

所有几何类都基于以下概念：将屏幕上的位置表示为二维平面。可以将屏幕看作是具有水平 (x) 轴和垂直 (y) 轴的平面图形。屏幕上的任何位置 （或 " 点 "）可以表示为 x 和 y 值对，即该位置的 " 坐标 "。

每个显示对象 （包括 Stage）都有自己的坐标空间。坐标空间是对象自己的图形，用于绘制子显示对象、绘图等的位置。原点的坐标位置为 0, 0 （X 和 Y 轴在此处相交），位于显示对象的左上角。此原点位置始终适用于舞台，但对于其他显示对象则不一定适用。 X 轴上的值越大越偏向右侧，越小越偏向左侧。对于原点左侧的位置， X 坐标是负数。然而，与传统的坐标系相反， Flash 运行时在 Y 轴的坐标值越大越偏向屏幕下方，越小越偏向屏幕上方。原点上侧的值为负的 Y 坐标值。因为舞台的左上角是其坐标空间的原点，所以舞台上大多数对象的 X 坐标值大于 0 但小于舞台宽度。而且同一个对象的 Y 坐标值大于 0 但小于舞台高度。

可以使用 Point 类实例来表示坐标空间中的各个点。您可以创建一个 Rectangle 实例来表示坐标空间中的矩形区域。对于高级用户，可以使用 Matrix 实例将多个或复杂变形应用于显示对象。通过使用显示对象的属性，可以将很多简单变形 （如旋转、位置以及缩放变化）直接应用于该对象。有关使用显示对象属性应用变形的详细信息，请参阅第 144 页的 " 处理显示对象 "。

重要概念和术语

以下参考列表包含重要的几何术语：

笛卡尔坐标 坐标通常写为一对数字 （例如 5, 12 或 17, -23）。两个数字分别是 x 坐标和 y 坐标。

坐标空间 包含在显示对象中的坐标的图形，显示对象的子元素位于该坐标位置处。

原点 坐标空间中位于 X 轴和 Y 轴相交处的点。该点的坐标为 0, 0。

点 坐标空间中的一个位置。在 ActionScript 使用的二维坐标系中，沿 X 轴和 Y 轴的位置 （点的坐标）定义点。

注册点 显示对象中，坐标空间的原点 （0, 0 坐标）。

缩放 对象的大小，相对于其原始大小。用作动词时，对象缩放是指伸展或缩小对象以更改其大小。

转换 将点的坐标从一个坐标空间更改到另一个坐标空间。

转换 对图形的视觉特性的调整，例如旋转对象、更改其比例、倾斜或扭曲其形状或改变其颜色。

X 轴 ActionScript 中使用的二维坐标系中的横轴。

Y 轴 ActionScript 中使用的二维坐标系中的纵轴。

使用 Point 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Point 对象定义一对笛卡尔坐标。它表示二维坐标系中的某个位置。其中 x 表示水平轴， y 表示垂直轴。

要定义 Point 对象，请设置它的 x 和 y 属性，如下所示：

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

确定两点之间的距离

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 Point 类的 distance() 方法确定坐标空间两点之间的距离。例如，下面的代码确定同一显示对象容器中两个显示对象（circle1 和 circle2）的注册点之间的距离：

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

平移坐标空间

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果两个显示对象位于不同的显示对象容器中，则两者可能位于不同的坐标空间中。您可以使用 DisplayObject 类的 localToGlobal() 方法将坐标平移到舞台中相同 （全局）坐标空间。例如，下面的代码确定不同显示对象容器中两个显示对象（circle1 和 circle2）的注册点之间的距离：

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle2.x, circle2.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

同样，要查找名为 target 的显示对象的注册点与舞台上特定点间的距离，请使用 DisplayObject 类的 localToGlobal() 方法：

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

按指定的角度和距离移动显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以使用 Point 类的 polar() 方法将显示对象按特定角度移动特定距离。例如，下列代码按 60° 角度将 myDisplayObject 对象移动 100 像素：

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Point 类的其他用法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以将 Point 对象用于以下方法和属性：

类	方法或属性	说明
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()getObjectsUnderPoint()`	用于返回显示对象容器中某个点下的对象的列表。
BitmapData	hitTest()	用于定义 BitmapData 对象中的像素以及要检查点击的点。
BitmapData	applyFilter() copyChannel() merge() paletteMap() pixelDissolve() threshold()	用于定义那些定义操作的矩形的位置。
Matrix	deltaTransformPoint() transformPoint()	用于定义您要对其应用变形的点。
Rectangle	bottomRight size topLeft	用于定义这些属性。

使用 Rectangle 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Rectangle 对象定义一个矩形区域。Rectangle 对象有具体的位置，该位置由其左上角的 x 和 y 坐标以及 width 属性和 height 属性定义。您可以按照如下方式通过调用 Rectangle() 构造函数来为新 Rectangle 对象定义这些属性：

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

调整 Rectangle 对象的大小和进行重新定位

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有多种方法调整 Rectangle 对象的大小和进行重新定位。

您可以通过更改 Rectangle 对象的 x 和 y 属性直接重新定位该对象。此更改不会影响 Rectangle 对象的宽度或高度。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

如下面的代码所示，当更改 Rectangle 对象的 left 或 top 属性时，将重新定位该矩形。矩形的 x 和 y 属性分别与 left 和 top 属性匹配。然而， Rectangle 对象的左下角的位置不发生改变，因此调整了该对象的大小。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=20, y=30, w=80, h=20)
```

同样，如下面的示例所示，如果更改 Rectangle 对象的 bottom 或 right 属性，该对象左上角的位置不发生改变。矩形相应地调整了大小：

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

也可以使用 offset() 方法重新定位 Rectangle 对象，如下所示：

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

offsetPt() 方法工作方式类似，只不过它是将 Point 对象作为参数，而不是将 x 和 y 偏移量值作为参数。

还可以使用 inflate() 方法调整 Rectangle 对象的大小，该方法包含两个参数， dx 和 dy。 dx 参数表示矩形的左侧和右侧距离中心的像素数。 dy 参数表示矩形的顶部和底部距离中心的像素数：

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

inflatePt() 方法作方式类似，只不过它是将 Point 对象作为参数，而不是将 dx 和 dy 的值作为参数。

确定 Rectangle 对象的联合和交集

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 union() 方法来确定由两个矩形的边界形成的矩形区域：

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

可以使用 intersection() 方法来确定由两个矩形重叠区域形成的矩形区域：

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

使用 intersects() 方法查明两个矩形是否相交。也可以使用 intersects() 方法查明显示对象是否在舞台的某个区域中。对于下列代码示例，假设包含 circle 对象的显示对象容器的坐标空间与舞台的坐标空间相同。本示例说明如何使用 intersects() 方法来确定显示对象 circle 是否与由 target1 和 target2 Rectangle 对象定义的指定舞台区域相交：

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

同样，可以使用 intersects() 方法查明两个显示对象的边界矩形是否重叠。使用 DisplayObject 类的 getRect() 方法来包括显示对象的笔触添加到边界区域的其他任何空间。

Rectangle 对象的其他用法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Rectangle 对象可用于以下方法和属性：

类	方法或属性	说明
BitmapData	applyFilter()、colorTransform()、copyChannel()、copyPixels()、draw()、drawWithQuality()、encode()、fillRect()、generateFilterRect()、getColorBoundsRect()、getPixels()、merge()、paletteMap()、pixelDissolve()、setPixels() 和 threshold()	用作某些参数的类型以定义 BitmapData 对象的区域。
DisplayObject	getBounds()、getRect()、scrollRect、scale9Grid	用作属性的数据类型或返回的数据类型。
PrintJob	addPage()	用于定义 printArea 参数。
Sprite	startDrag()	用于定义 bounds 参数。
TextField	getCharBoundaries()	用作返回值类型。
Transform	pixelBounds	用作数据类型。

使用 Matrix 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Matrix 类表示一个转换矩阵，它确定如何将点从一个坐标空间映射到另一个坐标空间。您可以对一个显示对象执行不同的图形转换，方法是设置 Matrix 对象的属性，将该 Matrix 对象应用于 Transform 对象的 matrix 属性，然后应用该 Transform 对象作为显示对象的 transform 属性。这些转换函数包括平移 （x 和 y 重新定位）、旋转、缩放和倾斜。

虽然可以通过直接调整 Matrix 对象的属性（a、b、c、d、tx 和 ty）来定义矩阵，但更简单的方法是使用 createBox() 方法。使用此方法提供的参数可以直接定义生成的矩阵的缩放、旋转和平移效果。例如，以下代码创建一个 Matrix 对象，该对象可将某个对象水平缩放 2.0、垂直缩放 3.0、旋转 45°、向右移动 （转换） 10 像素并向下移动 20 像素：

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

还可以使用 scale()、 rotate() 和 translate() 方法调整 Matrix 对象的缩放、旋转和平移效果。请注意，这些方法合并了现有 Matrix 对象的值。例如，以下代码设置一个 Matrix 对象，该对象可将某个对象按缩放系数 4 进行缩放并旋转 60°，因为 scale() 和 rotate() 方法会调用两次：

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

要将倾斜转换应用到 Matrix 对象，请调整该对象的 b 或 c 属性。调整 b 属性将矩阵垂直倾斜，并调整 c 属性将矩阵水平倾斜。以下代码使用系数 2 垂直倾斜 myMatrix Matrix 对象：

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

可以将矩阵转换应用到显示对象的 transform 属性。例如，以下代码将矩阵转换应用于名为 myDisplayObject 的显示对象：

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

第一行将 Matrix 对象设置为 myDisplayObject 显示对象所使用的现有转换矩阵 （myDisplayObject 显示对象的 transformation 属性的 matrix 属性）。这样，调用的 Matrix 类方法对显示对象的现有位置、缩放和旋转会产生累积效果。

注：ColorTransform 类还包含在 flash.geometry 包中。该类用于设置 Transform 对象的 colorTransform 属性。因为它不应用任何几何转换，所以在此处不做详细讨论。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 ColorTransform 类。

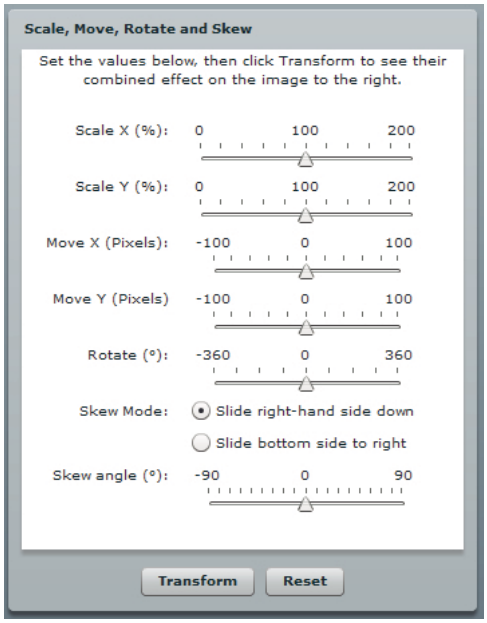
几何形状示例：对显示对象应用矩阵转换

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

DisplayObjectTransformer 范例应用程序说明许多使用 Matrix 类来变换显示对象的功能，包括：

- 旋转显示对象
- 缩放显示对象
- 平移 （重新定位）显示对象
- 倾斜显示对象

该应用程序提供了接口，以调整矩阵转换的参数，如下所示：



当用户单击“变形”按钮时，应用程序将应用适当的变形。

原始显示对象和旋转了 -45° 并缩放 50% 的显示对象

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/DisplayObjectTransformer 中找到 DisplayObjectTransformer 应用程序文件。该应用程序包含以下文件：

文件	说明
DisplayObjectTransformer.mxml 或 DisplayObjectTransformer.fla	Flash (FLA) 或 Flex (MXML) 中的主应用程序文件
com/example/programmingas3/geometry/MatrixTransformer.as	一个类，包含用于应用矩阵转换的方法。
img/	一个目录，包含应用程序使用的范例图像文件。

定义 MatrixTransformer 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

MatrixTransformer 类包含应用 Matrix 对象的几何变形的静态方法。

transform() 方法

transform() 方法包含以下属性的参数：

- sourceMatrix — 此方法转换的输入矩阵
- xScale 和 yScale — x 和 y 缩放系数
- dx 和 dy — x 和 y 平移量 （以像素为单位）
- rotation — 旋转量 （以度为单位）
- skew — 倾斜系数 （以百分比表示）
- skewType — 倾斜的方向， "right" 或 "left"

返回值为生成的矩阵。

transform() 方法调用下列类的静态方法：

- skew()
- scale()
- translate()
- rotate()

每种方法都返回应用了转换的源矩阵。

skew() 方法

skew() 方法通过调整矩阵的 b 和 c 属性来倾斜矩阵。可选参数 unit 确定用于定义倾斜角度的单位，如果必要，该方法会将 angle 值转换为弧度：

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

创建并调整 skewMatrix Matrix 对象以应用倾斜转换。最初，它是恒等矩阵，如下所示：

```
var skewMatrix:Matrix = new Matrix();
```

skewSide 参数确定倾斜应用到的边。如果该参数设置为 "right"，则以下代码设置矩阵的 b 属性：

```
skewMatrix.b = Math.tan(angle);
```

否则，将通过调整矩阵的 c 属性来倾斜底边，如下所示：

```
skewMatrix.c = Math.tan(angle);
```

然后，通过将两个矩阵连接起来以将所产生的倾斜应用到现有矩阵，如下面的示例所示：

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

scale() 方法

下面的示例显示 scale() 方法首先会调整缩放系数 （如果提供的缩放系数为百分比），然后使用矩阵对象的 scale() 方法：

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

translate() 方法

translate() 方法只需通过调用矩阵对象的 translate() 方法即可应用 dx 和 dy 平移系数，如下所示：

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

rotate() 方法

rotate() 方法将输入的旋转系数转换为弧度 （如果提供的是角度或渐变），然后调用矩阵对象的 rotate() 方法：

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

从应用程序中调用 MatrixTransformer.transform() 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序提供了一个用户界面，以便从用户处获得转换参数。然后将这些值与显示对象的 transform 属性的 matrix 属性一起传递给 Matrix.transform() 方法，如下所示：

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

应用程序随后将返回值应用到显示对象的 transform 属性的 matrix 属性，从而触发转换：

```
img.content.transform.matrix = tempMatrix;
```

第 12 章：使用绘图 API

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

虽然导入的图像和插图非常重要，但您可以使用一项称为绘图 API 的功能 （用于在 ActionScript 中绘制线条和形状）随时启动计算机中的应用程序，这就相当于一个空白画布，您可以在上面创建所需的任何图像。能够创建自己的图形可为您的应用程序提供广阔的前景。使用此处介绍的方法，您可以完成许多工作，如创建绘图程序、制作交互的动画效果，或以编程方式创建您自己的用户界面元素，等等。

更多帮助主题
flash.display.Graphics

绘制 API 的基础

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

绘图 API 是 ActionScript 中一项内置功能的名称，您可以使用该功能来创建矢量图形 （直线、曲线、形状、填充和渐变），并使用 ActionScript 在屏幕上显示它们。 flash.display.Graphics 类提供了这一功能。您可以在任何 Shape、 Sprite 或 MovieClip 实例中使用 ActionScript 进行绘制（使用在这每个类中定义的 graphics 属性）。（实际上，这每个类的 graphics 属性都是 Graphics 类的实例。）

如果刚刚开始学习使用代码进行绘制，可以使用 Graphics 类中包含的几种方法来简化常见形状 （如圆、椭圆、矩形以及带圆角的矩形）的绘制过程。您可以将它们作为空线条或填充形状进行绘制。当您需要更高级的功能时， Graphics 类还提供了用于绘制直线和二次贝塞尔曲线的方法，您可以将这些方法与 Math 类中的三角函数配合使用，创建所需的任何形状。

Flash 运行时（如 Flash Player 10 和 Adobe AIR 1.5 及更高版本）增加了一个绘图 API，通过该 API，只需一个命令即可用编程方式绘制完整的形状。熟悉 Graphics 类以及 “ 绘图 API 使用基础知识 ” 中介绍的任务后，请继续学习第 196 页的 “ 绘图 API 高级用法 ”，了解有关这些绘图 API 功能的详细信息。

重要概念和术语
以下参考列表包含使用绘图 API 时会遇到的重要术语：

锚点 二次贝塞尔曲线的两个端点之一。

控制点 该点定义二次贝塞尔曲线的弯曲方向和弯曲量。弯曲的线绝不会到达控制点；但曲线就好像朝着控制点方向进行绘制的。

坐标空间 包含在显示对象中的坐标的图形，显示对象的子元素位于该坐标位置处。

填充 用颜色填充了线条的形状的内部实体部分，或没有外框的整个形状。

渐变 此颜色是指从一种颜色逐渐过渡到一种或多种其他颜色 （与纯色相对）。

点 坐标空间中的一个位置。在 ActionScript 使用的二维坐标系中，点是按其 x 轴和 y 轴位置 （点坐标）来定义的。

二次贝塞尔曲线 由特定的数学公式定义的曲线类型。在这种类型的曲线中，曲线形状根据锚点 （曲线端点）和控制点 （定义曲线的弯曲方向和弯曲量）的位置来计算。

缩放 对象的大小，相对于其原始大小。用作动词时，对象缩放是指伸展或缩小对象以更改其大小。

笔触 用颜色填充了线条的形状的外框部分，或未填充形状的线条。

转换 将点的坐标从一个坐标空间更改到另一个坐标空间。

X 轴 ActionScript 中使用的二维坐标系中的横轴。

Y 轴 ActionScript 中使用的二维坐标系中的纵轴。

Graphics 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

每个 Shape、Sprite 和 MovieClip 对象都具有一个 graphics 属性，它是 Graphics 类的一个实例。Graphics 类包含用于绘制线条、填充和形状的属性和方法。如果要将显示对象仅用作内容绘制画布，则可以使用 Shape 实例。Shape 实例的性能优于其他用于绘制的显示对象，因为它不会产生 Sprite 和 MovieClip 类中的附加功能的开销。如果希望能够在显示对象上绘制图形内容，并且还希望该对象包含其他显示对象，则可以使用 Sprite 实例。有关确定用于各种任务的显示对象的详细信息，请参阅第 143 页的 " 选择 DisplayObject 子类 "。

绘制线条和曲线

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Graphics 实例进行的所有绘制均基于包含线条和曲线的基本绘制。因此，必须使用一系列相同的步骤来执行所有 ActionScript 绘制：

- 定义线条和填充样式
- 设置初始绘制位置
- 绘制线条、曲线和形状 （可选择移动绘制点）
- 如有必要，完成创建填充

定义线条和填充样式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要使用 Shape、 Sprite 或 MovieClip 实例的 graphics 属性进行绘制，您必须先定义在绘制时使用的样式 （线条大小和颜色、填充颜色）。就像使用 Adobe® Flash® Professional 或其他绘图应用程序中的绘制工具一样，使用 ActionScript 进行绘制时，可以使用笔触进行绘制，也可以不使用笔触；可以使用填充颜色进行绘制，也可以不使用填充颜色。您可以使用 lineStyle() 或 lineGradientStyle() 方法来指定笔触的外观。要创建纯色线条，请使用 lineStyle() 方法。调用此方法时，您指定的最常用的值是前三个参数：线条粗细、颜色以及 Alpha。例如，该行代码指示名为 myShape 的 Shape 对象绘制 2 个像素粗、红色 (0x990000) 以及 75% 不透明的线条：

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

Alpha 参数的默认值为 1.0 (100%)，因此，如果需要完全不透明的线条，可以将该参数的值保持不变。 lineStyle() 方法还接受另外两个参数，分别对应于像素提示和缩放模式；有关使用这些参数的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中 Graphics.lineStyle() 方法的说明。

要创建渐变线条，请使用 lineGradientStyle() 方法。关于此方法的介绍请参阅第 189 页的 " 创建渐变线条和填充 "。

如果要创建填充形状，请在开始绘制之前调用 beginFill()、beginGradientFill()、beginBitmapFill() 或 beginShaderFill() 方法。其中的最基本方法 beginFill() 接受以下两个参数：填充颜色以及填充颜色的 Alpha 值 （可选）。例如，如果要绘制具有纯绿色填充的形状，应使用以下代码 （假设在名为 myShape 的对象上进行绘制）：

```
myShape.graphics.beginFill(0x00FF00);
```

调用任何填充方法时，将隐式地结束任何以前的填充，然后再开始新的填充。调用任何指定笔触样式的方法时，将替换以前的笔触，但不会改变以前指定的填充，反之亦然。

指定了线条样式和填充属性后，下一步是指示绘制的起始点。 Graphics 实例具有一个绘制点，就像在一张纸上的钢笔尖一样。无论绘制点位于什么位置，它都是开始执行下一个绘制动作的位置。最初， Graphics 对象将它绘制时所在对象的坐标空间中的点 (0, 0) 作为起始绘制点。要在其他点开始进行绘制，您可以先调用 moveTo() 方法，然后再调用绘制方法之一。这类似于将钢笔尖从纸上抬起，然后将其移到新位置。

确定绘制点后，可通过使用对绘制方法 lineTo() （用于绘制直线）和 curveTo() （用于绘制曲线）的一系列调用来进行绘制。

在进行绘制时，可随时调用 moveTo() 方法，将绘制点移到新位置而不进行绘制。

进行绘制时，如果您已指定填充颜色，可以通过调用 endFill() 方法关闭填充。如果绘制的不是闭合的形状 （即，调用 endFill() 时绘制点不在形状的起始点），则调用 endFill() 方法时， Flash 运行时将自动绘制一条直线以使形状闭合，该直线从当前绘制点到最近一次 moveTo() 调用中指定的位置。如果已开始填充并且没有调用 endFill()，调用 beginFill() （或其他填充方法之一）时，将关闭当前填充并开始新的填充。

绘制直线

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

调用 lineTo() 方法时，Graphics 对象将绘制一条直线，该直线从当前绘制点到指定为方法调用中的两个参数的坐标，以便使用指定的线条样式进行绘制。例如，该行代码将绘制点放在点 (100, 100) 上，然后绘制一条到点 (200, 200) 的直线：

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

以下示例绘制红色和绿色三角形，其高度为 100 个像素：

```
var triangleHeight:uint = 100;
var triangle:Shape = new Shape();

// red triangle, starting at point 0, 0
triangle.graphics.beginFill(0xFF0000);
triangle.graphics.moveTo(triangleHeight / 2, 0);
triangle.graphics.lineTo(triangleHeight, triangleHeight);
triangle.graphics.lineTo(0, triangleHeight);
triangle.graphics.lineTo(triangleHeight / 2, 0);

// green triangle, starting at point 200, 0
triangle.graphics.beginFill(0x00FF00);
triangle.graphics.moveTo(200 + triangleHeight / 2, 0);
triangle.graphics.lineTo(200 + triangleHeight, triangleHeight);
triangle.graphics.lineTo(200, triangleHeight);
triangle.graphics.lineTo(200 + triangleHeight / 2, 0);

this.addChild(triangle);
```

绘制曲线

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

curveTo() 方法可以绘制二次贝塞尔曲线。这将绘制一个连接两个点 （称为锚点）的弧，同时向第三个点 （称为控制点）弯曲。 Graphics 对象使用当前绘制位置作为第一个锚点。调用 curveTo() 方法时，将传递以下四个参数：控制点的 x 和 y 坐标，后跟第二个锚点的 x 和 y 坐标。例如，以下代码绘制一条曲线，它从点 (100, 100) 开始，到点 (200, 200) 结束。由于控制点位于点 (175, 125)，因此，这会创建一条曲线，它先向右移动，然后向下移动：

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

以下示例绘制红色和绿色圆形对象，其宽度和高度均为 100 个像素。请注意，由于二次贝塞尔方程式所具有的特性，这些对象并不是完美的圆：

```
var size:uint = 100;
var roundObject:Shape = new Shape();

// red circular shape
roundObject.graphics.beginFill(0xFF0000);
roundObject.graphics.moveTo(size / 2, 0);
roundObject.graphics.curveTo(size, 0, size, size / 2);
roundObject.graphics.curveTo(size, size, size / 2, size);
roundObject.graphics.curveTo(0, size, 0, size / 2);
roundObject.graphics.curveTo(0, 0, size / 2, 0);

// green circular shape
roundObject.graphics.beginFill(0x00FF00);
roundObject.graphics.moveTo(200 + size / 2, 0);
roundObject.graphics.curveTo(200 + size, 0, 200 + size, size / 2);
roundObject.graphics.curveTo(200 + size, size, 200 + size / 2, size);
roundObject.graphics.curveTo(200, size, 200, size / 2);
roundObject.graphics.curveTo(200, 0, 200 + size / 2, 0);

this.addChild(roundObject);
```

使用内置方法绘制形状

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

为了便于绘制常见形状 （如圆、椭圆、矩形以及带圆角的矩形）， ActionScript 3.0 中提供了用于绘制这些常见形状的方法。它们是 Graphics 类的 drawCircle()、 drawEllipse()、 drawRect() 和 drawRoundRect() 方法。这些方法可用于替代 lineTo() 和 curveTo() 方法。但要注意，在调用这些方法之前，您仍需指定线条和填充样式。

以下示例重新创建绘制红色、绿色以及蓝色正方形的示例，其宽度和高度均为 100 个像素。以下代码使用 drawRect() 方法，并且还指定了填充颜色的 Alpha 为 50% (0.5)：

```
var squareSize:uint = 100;
var square:Shape = new Shape();
square.graphics.beginFill(0xFF0000, 0.5);
square.graphics.drawRect(0, 0, squareSize, squareSize);
square.graphics.beginFill(0x00FF00, 0.5);
square.graphics.drawRect(200, 0, squareSize, squareSize);
square.graphics.beginFill(0x0000FF, 0.5);
square.graphics.drawRect(400, 0, squareSize, squareSize);
square.graphics.endFill();
this.addChild(square);
```

在 Sprite 或 MovieClip 对象中，使用 graphics 属性创建的绘制内容始终出现在该对象包含的所有子级显示对象的后面。另外，graphics 属性内容不是单独的显示对象，因此，它不会出现在 Sprite 或 MovieClip 对象的子级列表中。例如，以下 Sprite 对象使用其 graphics 属性来绘制圆，并且其子级显示对象列表中包含一个 TextField 对象：

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.width = 200;
label.text = "They call me mellow yellow...";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

请注意， TextField 将出现在使用 graphics 对象绘制的圆的上面。

创建渐变线条和填充

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

graphics 对象也可以绘制渐变笔触和填充，而不是纯色笔触和填充。渐变笔触是使用 lineGradientStyle() 方法创建的；渐变填充是使用 beginGradientFill() 方法创建的。

这两种方法接受相同的参数。前四个参数是必需的，即类型、颜色、 Alpha 以及比率。其余四个参数是可选的，但对于高级自定义非常有用。

- 第一个参数指定要创建的渐变类型。可接受的值为 GradientType.LINEAR 或 GradientType.RADIAL。
- 第二个参数指定要使用的颜色值的数组。在线性渐变中，将从左向右排列颜色。在放射状渐变中，将从内到外排列颜色。数组颜色的顺序表示在渐变中绘制颜色的顺序。
- 第三个参数指定前一个参数中相应颜色的 Alpha 透明度值。
- 第四个参数指定比率或每种颜色在渐变中的重要程度。可接受的值范围是 0-255。这些值并不表示任何宽度或高度，而是表示在渐变中的位置； 0 表示渐变开始，255 表示渐变结束。比率数组必须按顺序增加，并且包含的条目数与第二个和第三个参数中指定的颜色和 Alpha 数组相同。

虽然第五个参数 （转换矩阵）是可选的，但通常会使用该参数，因为它提供了一种简便且有效的方法来控制渐变外观。此参数接受 Matrix 实例。为渐变创建 Matrix 对象的最简单方法是使用 Matrix 类的 createGradientBox() 方法。

定义 Matrix 对象以用于渐变

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 flash.display.Graphics 类的 beginGradientFill() 和 lineGradientStyle() 方法来定义在形状中使用的渐变。定义渐变时，需要提供一个矩阵作为这些方法的其中一个参数。

定义矩阵的最简单方法是使用 Matrix 类的 createGradientBox() 方法，该方法创建一个用于定义渐变的矩阵。可以使用传递给 createGradientBox() 方法的参数来定义渐变的缩放、旋转和位置。 createGradientBox() 方法接受以下参数：

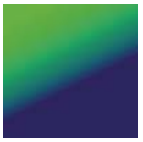
- 渐变框宽度：渐变扩展到的宽度 （以像素为单位）
- 渐变框高度：渐变扩展到的高度 （以像素为单位）
- 渐变框旋转：将应用于渐变的旋转角度 （以弧度为单位）
- 水平平移：将渐变水平移动的距离 （以像素为单位）
- 垂直平移：将渐变垂直移动的距离 （以像素为单位）

例如，假设渐变具有以下特性：

- GradientType.LINEAR
- 绿色和蓝色这两种颜色 （ratios 数组设置为 [0, 255]）
- SpreadMethod.PAD
- InterpolationMethod.LINEAR_RGB

下面的示例显示的是几种渐变，如图所示，它们的 createGradientBox() 方法的 rotation 参数不同，但所有其他设置是相同的：

width = 100;
height = 100;
rotation = 0;
tx = 0;
ty = 0;	

width = 100;
height = 100;
rotation = Math.PI/4; // 45°
tx = 0;
ty = 0;
	

width = 100;
height = 100;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;	

下面的示例显示的是绿到蓝线性渐变的效果，如图所示，它们的 createGradientBox() 方法的 rotation、 tx 和 ty 参数不同，但所有其他设置是相同的：

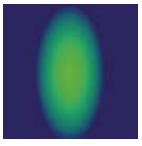
width = 50;
height = 100;
rotation = 0;
tx = 0;
ty = 0;	

width = 50;
height = 100;
rotation = 0
tx = 50;
ty = 0;	

width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;	

width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 50;	

createGradientBox() 方法的 width、 height、 tx 和 ty 参数也会影响 " 放射状 " 渐变填充的大小和位置，如下面的示例所示：

width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
	

下面的代码生成了所示的最后一个放射状渐变：

```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                            colors,
                            alphas,
                            ratios,
                            matrix,
                            spreadMethod,
                            interp,
                            focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

请注意，渐变填充的宽度和高度是由渐变矩阵的宽度和高度决定的，而不是由使用 Graphics 对象绘制的宽度和高度决定的。使用 Graphics 对象进行绘制时，您绘制的内容位于渐变矩阵中的这些坐标处。即使使用 Graphics 对象的形状方法之一 （如 drawRect()），渐变也不会将其自身伸展到绘制的形状的大小；必须在渐变矩阵本身中指定渐变的大小。

下面说明了渐变矩阵的尺寸和绘图本身的尺寸之间的视觉差异：

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(100, 40, 0, 0, 0);
myShape.graphics.beginGradientFill(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1, 1, 1], [0, 128,
255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 50, 40);
myShape.graphics.drawRect(0, 50, 100, 40);
myShape.graphics.drawRect(0, 100, 150, 40);
myShape.graphics.endFill();
this.addChild(myShape);
```

该代码绘制三个具有相同填充样式 （使用平均分布的红色、绿色和蓝色指定的）的渐变。这些渐变是使用 drawRect() 方法绘制的，像素宽度分别为 50、 100 和 150。 beginGradientFill() 方法中指定的渐变矩阵是使用像素宽度 100 创建的。这意味着，第一个渐变仅包含渐变色谱的一半，第二个渐变包含全部色谱，而第三个渐变包含全部色谱以及向右扩展的额外 50 蓝色像素。

lineGradientStyle() 方法的工作方式与 beginGradientFill() 类似，所不同的是，除了定义渐变外，您还必须在绘制之前使用 lineStyle() 方法指定笔触粗细。以下代码绘制一个带有红色、绿色和蓝色渐变笔触的框：

```
var myShape:Shape = new Shape();
var gradientBoxMatrix:Matrix = new Matrix();
gradientBoxMatrix.createGradientBox(200, 40, 0, 0, 0);
myShape.graphics.lineStyle(5, 0);
myShape.graphics.lineGradientStyle(GradientType.LINEAR, [0xFF0000, 0x00FF00, 0x0000FF], [1, 1, 1], [0, 128,
255], gradientBoxMatrix);
myShape.graphics.drawRect(0, 0, 200, 40);
this.addChild(myShape);
```

有关 Matrix 类的详细信息，请参阅第 180 页的 “ 使用 Matrix 对象 ”。

将 Math 类与绘制方法配合使用

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Graphics 对象可以绘制圆和正方形，但也可以绘制更复杂的形状，尤其是在将绘制方法与 Math 类的属性和方法配合使用时。Math 类包含人们通常很感兴趣的数学常量，如 Math.PI （约等于 3.14159265...），此常量表示圆的周长与其直径的比率。它还包含三角函数的方法，其中包括 Math.sin()、Math.cos() 和 Math.tan() 等。使用这些方法和常量绘制形状可产生更动态的视觉效果，尤其是用于重复或递归时。

Math 类的很多方法都要求以弧度为单位来测量圆弧，而不是使用角度。 Math 类的一个常见用途是在这两种类型的单位之间进行转换：

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

以下示例创建一个正弦波和余弦波以重点说明给定值的 Math.sin() 和 Math.cos() 方法之间的差异。

```
var sinWavePosition = 100;
var cosWavePosition = 200;
var sinWaveColor:uint = 0xFF0000;
var cosWaveColor:uint = 0x00FF00;
var waveMultiplier:Number = 10;
var waveStretcher:Number = 5;

var i:uint;
for(i = 1; i < stage.stageWidth; i++)
{
    var sinPosY:Number = Math.sin(i / waveStretcher) * waveMultiplier;
    var cosPosY:Number = Math.cos(i / waveStretcher) * waveMultiplier;

    graphics.beginFill(sinWaveColor);
    graphics.drawRect(i, sinWavePosition + sinPosY, 2, 2);
    graphics.beginFill(cosWaveColor);
    graphics.drawRect(i, cosWavePosition + cosPosY, 2, 2);
}
```

使用绘图 API 进行动画处理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用绘图 API 创建内容的一个优点是，您并不限于将内容放置一次。可通过保留和修改用于绘制的变量来修改所绘制的内容。您可以通过更改变量和重绘 （在一段帧上或使用计时器）来利用原有的动画。

例如，以下代码更改每个经过的帧 （通过侦听 Event.ENTER_FRAME 事件）的显示内容以增加当前度数，指示 graphics 对象清除内容并在更新位置进行重绘。

```
stage.frameRate = 31;

var currentDegrees:Number = 0;
var radius:Number = 40;
var satelliteRadius:Number = 6;

var container:Sprite = new Sprite();
container.x = stage.stageWidth / 2;
container.y = stage.stageHeight / 2;
addChild(container);
var satellite:Shape = new Shape();
container.addChild(satellite);

addEventListener(Event.ENTER_FRAME, doEveryFrame);

function doEveryFrame(event:Event):void
{
    currentDegrees += 4;
    var radians:Number = getRadians(currentDegrees);
    var posX:Number = Math.sin(radians) * radius;
    var posY:Number = Math.cos(radians) * radius;
    satellite.graphics.clear();
    satellite.graphics.beginFill(0);
    satellite.graphics.drawCircle(posX, posY, satelliteRadius);
}
function getRadians(degrees:Number):Number
{
return degrees * Math.PI / 180;
}
```

要产生明显不同的效果，您可以尝试修改代码开头的初始种子变量 （currentDegrees、 radius 和 satelliteRadius）。例如，尝试缩小 radius 变量和 / 或增大 totalSatellites 变量。这只是一个说明绘图 API 如何创建可视显示内容（其复杂性掩盖了创建简便性）的示例。

绘制 API 示例：算法可视化生成器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Algorithmic Visual Generator 示例在舞台上动态绘制几个 “ 卫星 ”，即在圆形轨道中运动的圆形物。要阐述的功能包括：

- 使用绘图 API 绘制具有动态外观的基本形状
- 将用户交互与绘图中使用的属性相关联
- 清除每个帧中的舞台内容并重绘以利用原有的动画

上一小节中的示例使用 Event.ENTER_FRAME 事件对唯一的 “ 卫星 ” 进行动画处理。该示例在此基础之上进一步扩展，生成一个包含一系列滑块的控制面板，这些滑块会立即更新若干卫星的可视显示内容。此示例将代码格式设置为外部类，并将卫星创建代码包装在循环中，以将对每个卫星的引用存储在 satellites 数组中。

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/AlgorithmicVisualGenerator 文件夹中找到应用程序文件。此文件夹包含以下文件：

文件	说明
AlgorithmicVisualGenerator.fla	Flash Professional 中的主应用程序文件 (FLA)。
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	此类提供应用程序的主要功能，其中包括在舞台上绘制卫星，以及从控制面板中响应事件以更新影响卫星绘制的变量。
com/example/programmingas3/algorithmic/ControlPanel.as	此类管理用户与几个滑块之间的交互并在发生此类交互时调度事件。
com/example/programmingas3/algorithmic/Satellite.as	此类表示在轨道中围绕中心点旋转的显示对象，并包含与其当前绘制状态有关的属性。

设置侦听器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序先创建三个侦听器。第一个侦听器侦听从控制面板中调度的事件：必须对卫星进行重新构建。第二个侦听器侦听对 SWF 文件的舞台大小的更改。第三个侦听器侦听 SWF 文件中每个经过的帧，并使用 doEveryFrame() 函数进行重绘。

创建卫星

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在设置这些侦听器后，将调用 build() 函数。此函数先调用 clear() 函数，后者清空 satellites 数组，并清除以前在舞台上绘制的任何内容。这是必要的，因为每当控制面板发送事件来执行此操作时，都可能会重新调用 build() 函数，例如在颜色设置已发生变化时。在这种情况下，必须删除并重新创建卫星。

该函数随后创建一些卫星，并设置创建所需的初始属性，如 position 变量 （从轨道中的随机位置开始）和 color 变量 （在本示例中，该变量在创建卫星后不会发生改变）。

创建每个卫星时，会将对它的引用添加到 satellites 数组中。调用 doEveryFrame() 函数时，它将更新此数组中的所有卫星。

更新卫星位置

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

doEveryFrame() 函数是应用程序动画过程的核心。将为每个帧调用该函数，频率等于 SWF 文件的帧频。由于绘制变量略微发生了变化，因此，这会利用原有的动画外观。

该函数先清除以前绘制的所有内容，然后再重绘背景。接下来，它循环访问每个卫星容器，增加每个卫星的 position 属性以及更新 radius 和 orbitRadius 属性 （这些属性可能由于用户与控制面板的交互而发生了变化）。最后，通过调用 Satellite 类的 draw() 方法，在新位置对卫星进行更新。

请注意，计数器 i 最多只增加到 visibleSatellites 变量。这是因为，如果用户通过控制面板限制了显示的卫星数，则不会重绘循环中的其余卫星，而应将其隐藏起来。这种情况发生在紧靠负责绘制的循环后面的循环中。

当 doEveryFrame() 函数完成时，将在屏幕上的位置中更新 visibleSatellites 数量。

响应用户交互

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用户交互是通过控制面板发生的，它是由 ControlPanel 类管理的。此类设置一个侦听器，并为每个滑块设置单独的最小、最大和默认值。当用户移动这些滑块时，将调用 changeSetting() 函数。此函数更新控制面板的属性。如果更改需要重新构建显示内容，则会调度一个事件，随后将在主应用程序文件中对该事件进行处理。当控制面板设置发生变化时， doEveryFrame() 函数将使用更新的变量绘制每个卫星。

进一步自定义

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

本示例只是概要介绍了使用绘图 API 生成可视内容方面的基础知识。它使用相对较少的几行代码来创建似乎非常复杂的交互式体验。尽管如此，您还是可以对本示例进行较小的改动以进行扩展。下面列出了一些很好的方法：

- doEveryFrame() 函数可以增加卫星的颜色值。
- doEveryFrame() 函数可能会随着时间的推移缩小或增大卫星半径。
- 卫星并不一定是圆形的，例如，可以使用 Math 类根据正弦波来移动其半径。
- 卫星可以使用碰撞检测来检查是否与其他卫星重叠。

可以将绘图 API 作为在 Flash 创作环境中创建视觉效果的替代方法，以便在运行时绘制基本形状。但是，它也可以用来创建涵盖范围很广且无法手动创建的各种视觉效果。通过使用绘图 API 和一些数学函数， ActionScript 作者可能实现很多意想不到的创作效果。

绘图 API 高级用法

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Flash Player 10 运行时、 Adobe AIR 1.5 运行时和更高版本的 Flash 运行时支持一组高级绘图功能。这些运行时的绘图 API 增强功能对早期版本中的绘图方法进行了扩展，使您通过建立数据集，即可生成形状、在运行时更改形状及创建三维效果。绘图 API 增强功能将现有方法合并为其他命令。这些命令利用矢量数组和枚举类为绘图方法提供数据集。使用矢量数组，可以快速呈示更复杂的形状，开发人员可以用编程方式为运行时呈示的动态形状更改数组值。

下面几节介绍 Flash Player 10 中引入的绘制功能：第 197 页的 “ 绘制路径 ”、第 198 页的 “ 定义缠绕规则 ”、第 200 页的 “ 使用图形数据类 ” 和第 202 页的 “ 关于使用 drawTriangles()”。

您可能需要使用 ActionScript 中的高级绘图 API 来完成以下任务：

- 使用 Vector 对象存储绘制方法的数据
- 以编程方式定义用于绘制形状的路径 （用单个操作）
- 定义缠绕规则，以确定如何填充重叠形状
- 读取显示对象的矢量图形内容，如序列化并保存图形数据、在运行时生成 Sprite 表以及绘制矢量图形内容的副本
- 使用三角形和绘制方法实现三维效果

重要概念和术语

以下参考列表包含本节中会遇到的重要术语：

- 矢量：数据类型完全相同的值组成的数组。 Vector 对象可存储绘制方法使用单个命令构建线条和形状时所用值的数组。有关 Vector 对象的详细信息，请参阅第 22 页的 " 索引数组 "。
- 路径：路径由一条或多条直线段或曲线段组成。每个线段的起点和终点都由坐标标记，就像用于固定线的针。路径可以是闭合的，例如圆；也可以是开放的，且具有不同的端点，例如波浪线。
- 缠绕：由渲染器解释的路径方向，包括正向 （顺时针）或负向 （逆时针）。
- GraphicsStroke: 用于设置线条样式的类。虽然 " 笔触 " 并非绘图 API 增强功能中的术语，但使用类以线条样式自身的填充属性来指定该线条样式却是新绘图 API 功能的一部分。您可以使用 GraphicsStroke 类动态调整线条的样式。
- Fill 对象：使用 flash.display.GraphicsBitmapFill 和 flash.display.GraphicsGradientFill 等传递给绘图命令 Graphics.drawGraphicsData() 的显示类所创建的对象。 Fill 对象和增强的绘图命令引入了一种面向对象程度更高的编程方法，用于复现 Graphics.beginBitmapFill() 和 Graphics.beginGradientFill() 的效果。

绘制路径

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

有关绘制线条和曲线的节中 （请参阅第 186 页的 " 绘制线条和曲线 "）介绍了一些命令，它们可用于绘制单个线条 (Graphics.lineTo()) 或曲线 (Graphics.curveTo())，然后将得到的线移至另一点 (Graphics.moveTo())，从而组成形状。 Graphics.drawPath() 和 Graphics.drawTriangles() 方法接受一组对象，它们将那些相同的绘图命令表示为一个参数。使用这些方法，您可以提供一系列 Graphics.lineTo()、Graphics.curveTo()、Graphics.moveTo() 命令，使 Flash 运行时在一个单独的语句中执行这些命令。

GraphicsPathCommand 枚举类定义一组对应于绘图命令的常量。您可以将这一系列常量 （包装在 Vector 实例中）作为对 Graphics.drawPath() 方法的一个参数来传递，然后通过一个单独的命令便可以呈现整个形状或多个形状。您还可以更改传递给这些方法的值，以更改现有形状。

除了这个绘图命令 Vector 之外，drawPath() 方法还需要一组坐标来对应每个绘图命令的坐标。创建一个包含坐标的 Vector 实例 （Number 实例），将其作为第二个参数 (data) 传递给 drawPath() 方法。

注：矢量中的值不是 Point 对象；该矢量是一系列数字，其中由两个数字组成的每个组表示一个 x/y 坐标对。

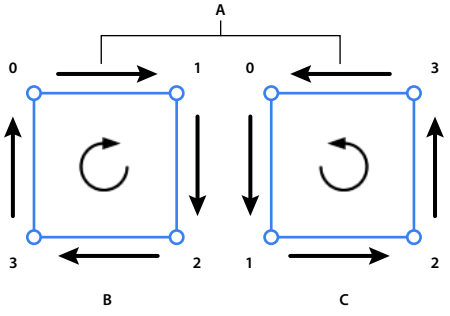
Graphics.drawPath() 方法将每个命令与其各自的点值 （由两个或四个数字组成的集合）相匹配，以在 Graphics 对象中生成路径：

```
package
{
    import flash.display.*;

    public class DrawPathExample extends Sprite
    {
        public function DrawPathExample(){

            var squareCommands:Vector.<int> = new Vector.<int>(5, true);
            squareCommands[0] = GraphicsPathCommand.MOVE_TO;
            squareCommands[1] = GraphicsPathCommand.LINE_TO;
            squareCommands[2] = GraphicsPathCommand.LINE_TO;
            squareCommands[3] = GraphicsPathCommand.LINE_TO;
            squareCommands[4] = GraphicsPathCommand.LINE_TO;

            var squareCoord:Vector.<Number> = new Vector.<Number>(10, true);
            squareCoord[0] = 20; //x
            squareCoord[1] = 10; //y
            squareCoord[2] = 50;
            squareCoord[3] = 10;
            squareCoord[4] = 50;
            squareCoord[5] = 40;
            squareCoord[6] = 20;
            squareCoord[7] = 40;
            squareCoord[8] = 20;
            squareCoord[9] = 10;

            graphics.beginFill(0x442266);//set the color
            graphics.drawPath(squareCommands, squareCoord);
        }
    }
}
```

定义缠绕规则

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

增强的绘图 API 还引入了路径 " 缠绕 " 的概念：路径的方向。路径的缠绕可以是正向的 （顺时针），也可以是负向的 （逆时针）。渲染器为 data 参数解释矢量所提供坐标的顺序确定了缠绕的方向。



正向缠绕和负向缠绕
A. 指示绘制方向的箭头 **B.** 正向缠绕 （顺时针） **C.** 负向缠绕 （逆时针）

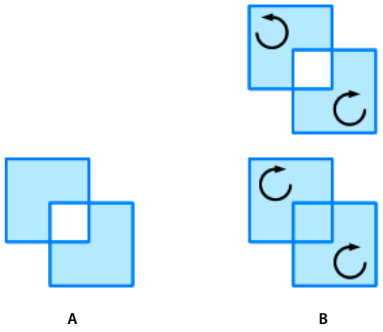
此外，请注意 Graphics.drawPath() 方法可选的第三个参数 "winding"：

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

在这种情况下，第三个参数是一个字符串或常量，用于指定相交路径的缠绕或填充规则。（这些常量值在 GraphicsPathWinding 类中定义为 GraphicsPathWinding.EVEN_ODD 或 GraphicsPathWinding.NON_ZERO。）当路径相交时，缠绕规则十分重要。

奇偶规则是标准的缠绕规则，早期的绘图 API 都使用此规则。奇偶规则也是 Graphics.drawPath() 方法的默认规则。使用奇偶缠绕规则时，任何相交路径都交替使用开放填充与闭合填充。如果使用同一填充绘制的两个正方形相交，则不会填充相交的区域。通常，相邻区域不会都填充或都不填充。

另一方面，非零规则依靠缠绕 （绘制方向）来确定是否填充相交路径定义的区域。当相对缠绕的路径相交时，不填充所定义的区域，这与奇偶规则十分类似。对于相同缠绕的路径，将填充本来不填充的区域：

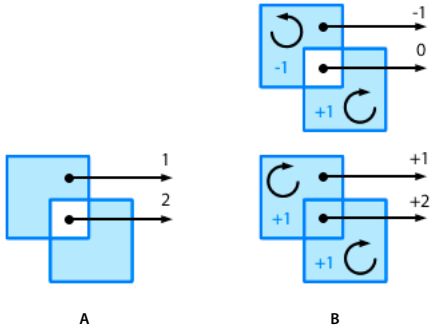


用于相交区域的缠绕规则
A. 奇偶缠绕规则 **B.** 非零缠绕规则

缠绕规则名称

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

这些名称系指用于定义如何管理填充的更具体规则。正向缠绕路径将得到赋值 +1 ；负向缠绕路径将得到赋值 -1。以形状上闭合区域中的一点为起点，绘制一个从该点向外无限延伸的线条。使用该线条与路径相交的次数以及这些路径的组合值来确定填充。对于奇偶缠绕，使用该线条与路径相交的次数。如果计数为奇数，则填充相交区域。如果计数为偶数，则不填充相交区域。对于非零缠绕，使用赋予路径的值。如果路径的组合值不为 0，则填充相交区域。如果组合值为 0，则不填充相交区域。



缠绕规则计数和填充
A. 奇偶缠绕规则 **B.** 非零缠绕规则

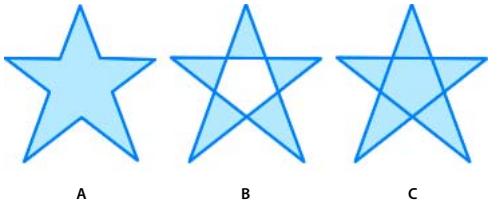
使用缠绕规则

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

这些填充规则很复杂，但有些情况下必须使用它们。例如绘制星形时。如果使用标准奇偶规则，该形状需要十个不同的线条。如果使用非零缠绕规则，所需十个线条将减少为五个。下面的 ActionScript 使用五个线条和非零缠绕规则来绘制星形：

```
graphics.beginFill(0x60A0FF);
graphics.drawPath( Vector.<int>([1,2,2,2,2]),  Vector.<Number>([66,10, 23,127, 122,50, 10,49, 109,127]),
GraphicsPathWinding.NON_ZERO);
```

星形如下所示：



使用不同缠绕规则的星形
A. 奇偶 10 个线条 **B.** 奇偶 5 个线条 **C.** 非零 5 个线条

如果对图像进行了动画处理，或图像用作三维对象上的纹理且发生重叠，则缠绕规则会变得更为重要。

使用图形数据类

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

增强的绘图 API 包括 flash.display 包中用于实现 IGraphicsData 接口的一组类。这些类用作表示绘图 API 的绘图方法的值对象 （数据容器）。

下面的类实现 IGraphicsData 接口：

- GraphicsBitmapFill
- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath
- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

通过这些类，可以将一个完整的绘图存储在一个 IGraphicsData 类型的 Vector 对象 (Vector.<IGraphicsData>) 中。之后您可以重新利用这些图形数据作为其他形状实例的数据源或存储绘图信息供以后使用。

请注意，每个填充样式有多个填充类，但只有一个笔触类。 ActionScript 只有一个笔触类 IGraphicsData，因为该笔触类使用填充类来定义自己的样式。因此每个笔触实际上是由一个笔触类和一个填充类的组合定义的。否则，这些图形数据类的 API 会镜像它们在 flash.display.Graphics 类中表示的方法：

Graphics 方法	对应的类
beginBitmapFill()	GraphicsBitmapFill
beginFill()	GraphicsSolidFill
beginGradientFill()	GraphicsGradientFill
beginShaderFill()	GraphicsShaderFill
lineBitmapStyle()	GraphicsStroke + GraphicsBitmapFill

Graphics 方法	对应的类
lineGradientStyle()	GraphicsStroke + GraphicsGradientFill
lineShaderStyle()	GraphicsStroke + GraphicsShaderFill
lineStyle()	GraphicsStroke + GraphicsSolidFill
moveTo() lineTo() curveTo() drawPath()	GraphicsPath
drawTriangles()	GraphicsTrianglePath

此外， GraphicsPath 类拥有自己的 GraphicsPath.moveTo()、 GraphicsPath.lineTo()、 GraphicsPath.curveTo()、 GraphicsPath.wideLineTo() 和 GraphicsPath.wideMoveTo() 实用程序方法，用于轻松为 GraphicsPath 实例定义这些命令。这些实用程序方法简化了直接定义或更新这些命令和数据值的过程。

使用矢量图形数据绘图

有了 IGraphicsData 实例集合之后，便可使用 Graphics 类的 drawGraphicsData() 方法来呈现图形了。drawGraphicsData() 方法将按顺序执行 IGraphicsData 实例矢量中的一组绘图指令：

```
// stroke object
var stroke:GraphicsStroke = new GraphicsStroke(3);
stroke.joints = JointStyle.MITER;
stroke.fill = new GraphicsSolidFill(0x102020);// solid stroke

// fill object
var fill:GraphicsGradientFill = new GraphicsGradientFill();
fill.colors = [0x0000FF, 0xEEFFEE];
fill.matrix = new Matrix();
fill.matrix.createGradientBox(70, 70, Math.PI/2);
// path object
var path:GraphicsPath = new GraphicsPath(new Vector.<int>(), new Vector.<Number>());
path.commands.push(GraphicsPathCommand.MOVE_TO, GraphicsPathCommand.LINE_TO, GraphicsPathCommand.LINE_TO);
path.data.push(125,0, 50,100, 175,0);

// combine objects for complete drawing
var drawing:Vector.<IGraphicsData> = new Vector.<IGraphicsData>();
drawing.push(stroke, fill, path);

// draw the drawing
graphics.drawGraphicsData(drawing);
```

通过修改示例中绘制所用路径的一个值，可以多次重绘形状，以生成更复杂的图像：

```
// draw the drawing multiple times
// change one value to modify each variation
graphics.drawGraphicsData(drawing);
path.data[2] += 200;
graphics.drawGraphicsData(drawing);
path.data[2] -= 150;
graphics.drawGraphicsData(drawing);
path.data[2] += 100;
graphics.drawGraphicsData(drawing);
path.data[2] -= 50;graphicsS.drawGraphicsData(drawing);
```

尽管 IGraphicsData 对象可以定义填充和笔触样式，但这两种样式并不是必需的。换句话说，可以在使用 IGraphicsData 对象绘制已保存路径集合的同时，使用 Graphics 类方法设置样式，反之亦然。

注：开始新的绘制之前，请使用 Graphics.clear() 方法清除以前的绘制；除非如上例所示，要在原始绘制的基础上继续绘制。在更改路径或 IGraphicsData 对象集合的某个部分时，请重绘整个绘制来查看所做的更改。

使用图形数据类时，只要绘制了三点或更多点，就会呈示填充，这是因为形状实际上在该点闭合。即使填充闭合，笔触也不会闭合；这一行为与使用多个 Graphics.lineTo() 或 Graphics.moveTo() 命令时不同。

读取矢量图形数据

Flash Player 11.6 和更高版本， **Adobe AIR 3.6** 和更高版本

除了将矢量内容绘制到显示对象之外，在 Flash Player 11.6 和 Adobe AIR 3.6 及更高版本中，您还可以使用 Graphics 类的 readGraphicsData() 方法来获取显示对象的矢量图形内容的数据表示。这可以用于创建一个图形快照以便在运行时保存、复制、创建 Sprite 表等等。

调用 readGraphicsData() 方法将返回一个包含 IGraphicsData 对象的 Vector 实例。这些对象就是使用 drawGraphicsData() 方法绘制矢量图形所用的对象。

使用 readGraphicsData() 方法读取矢量图形存在一些限制。有关详细信息，请参阅 《ActionScript 语言参考》中的 readGraphicsData() 条目。

关于使用 drawTriangles()

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Graphics.drawTriangles() 是 Flash Player 10 和 Adobe AIR 1.5 中引进的另一种高级方法，与 Graphics.drawPath() 方法相似。Graphics.drawTriangles() 方法还使用 Vector.<Number> 对象指定用于绘制路径的点位置。

但是， Graphics.drawTriangles() 方法的实际作用是便于通过 ActionScript 实现三维效果。有关使用 Graphics.drawTriangles() 生成三维效果的信息，请参阅第 304 页的 “ 通过三角形获得 3D 效果 ”。

第 13 章：使用位图

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

除了矢量图功能以外， ActionScript 3.0 还提供创建位图图像或操作加载到 SWF 中的外部位图图像的像素数据的功能。使用访问和更改各个像素值的功能，您可以创建自己的滤镜式图像效果并使用内置杂点功能创建纹理和随机杂点。

- Renaun Erickson：在 ActionScript 中，采用块传输技术渲染游戏资源
- 位图编程：Colin Moock 编著的 《Essential ActionScript 3》的第 26 章 （2007 年由 O'Reilly Media 出版）
- Mike Jones：在 Pushbutton 引擎中使用贴图定位
- Flash & Math：简单制作粒子像素系统
- Flixel

位图使用基本知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用数字图像时，您可能会遇到两种主要的图形类型：位图和矢量图形。位图图形也称为光栅图形，由排列为矩形网格形式的小方块 （像素）组成。矢量图形由以数学方式生成的几何形状 （如直线、曲线和多边形）组成。

位图图像用图像的宽度和高度来定义，以像素为量度单位，每个像素包含的位数表示像素包含的颜色数。在使用 RGB 颜色模型的位图图像中，像素由三个字节组成：红、绿和蓝。每个字节包含一个 0 至 255 之间的值。将字节与像素合并时，它们可以产生与艺术混合绘画颜色相似的颜色。例如，一个包含红色字节值 255、绿色字节值 102 和蓝色字节值 0 的像素可以形成明快的橙色。

位图图像的品质由图像分辨率和颜色深度位值共同确定。分辨率与图像中包含的像素数有关。像素数越大，分辨率越高，图像也就越精确。颜色深度与像素可包含的信息量有关。例如，颜色深度值为每像素 16 位的图像无法显示颜色深度为 48 位的图像所具有颜色数。因此， 48 位图像与 16 位图像相比，其阴影具有更高的平滑度。

由于位图图形与分辨率有关，因此不能很好地进行缩放。当放大位图图像时，这一特性显得尤为突出。通常，放大位图有损其细节和品质。

位图文件格式

位图图像可分为几种常见的文件格式。这些格式使用不同类型的压缩算法减小文件大小，并基于图像的最终用途优化图像品质。 Adobe 运行时支持的位图图像格式包括 BMP、 GIF、 JPG、 PNG 和 TIFF。

BMP

BMP （位映射）格式是 Microsoft Windows 操作系统使用的默认图像格式。这种格式不使用任何形式的压缩算法，因此文件大小通常较大。

GIF

图形交换格式 (GIF) 最初由 CompuServe 于 1987 年开发，作为一种传送 256 色 （8 位颜色）图像的方式。此格式提供较小的文件大小，是基于 Web 的图像的理想格式。受此格式的调色板所限， GIF 图像通常不适用于照片，照片通常需要高度的阴影和颜色渐变。 GIF 图像允许产生一位透明度，允许将颜色映射为清晰 （或透明）。这可以使网页的背景颜色通过已映射透明度的图像显示出来。

JPEG

由联合图像专家组 (JPEG) 开发， JPEG （通常写成 JPG）图像格式使用有损压缩算法允许 24 位颜色深度具有很小的文件大小。有损压缩意味着每次保存图像，都会损失图像品质和数据，但会生成更小的文件大小。由于 JPEG 能够显示数百万计的颜色，因此它是照片的理想格式。控制应用于图像的压缩程度的功能使您能够控制图像品质和文件大小。

PNG

可移植网络图形 (PNG) 格式是作为受专利保护的 GIF 文件格式的开放源替代格式而开发的。 PNG 最多支持 64 位颜色深度，允许使用最多 1600 万种颜色。由于 PNG 是一种比较新的格式，因此一些旧版本浏览器不支持 PNG 文件。与 JPG 不同，PNG 使用无损压缩，这意味着保存图像时不会丢失图像数据。PNG 文件还支持 Alpha 透明度，允许使用最多 256 级透明度。

TIFF

标签图像文件格式 (TIFF) 是在引入 PNG 之前的首选跨平台格式。 TIFF 格式的缺点是，因为 TIFF 有多种不同变体，但没有一种阅读器能够处理所有版本。此外，所有 Web 浏览器当前均不支持这种格式。 TIFF 可以使用有损或无损压缩，能够处理特定于设备的颜色空间 （如 CMYK）。

透明位图和不透明位图

使用 GIF 或 PNG 格式的位图图像可以对每个像素添加一个额外字节 （Alpha 通道）。此额外像素字节表示像素的透明度值。

GIF 图像允许使用一位透明度，这意味着您可以在 256 色调色板中指定一种透明的颜色。而 PNG 图像最多可以有 256 级透明度。当需要将图像或文本混合到背景中时，此功能特别有用。

ActionScript 3.0 在 BitmapData 类中复制了此额外透明度像素字节。与 PNG 透明度模型类似，ActionScript 最多提供 256 级透明度。

重要概念和术语

以下列表中包括您了解位图图形时会遇到的重要术语。

Alpha 颜色或图像中的透明度级别 （更准确地说是指不透明度）。 Alpha 量通常称为 “Alpha 通道 ” 值。

ARGB 颜色 一种配色方案，其中每个像素的颜色都是红、绿和蓝色值的混合颜色，并且其透明度被指定为一个 Alpha 值。

颜色通道 通常将颜色表示为几种基本颜色的混合颜色；对于计算机图形来说，基本颜色通常是红色、绿色和蓝色。每种基本颜色都视为一个颜色通道；每个颜色通道中的颜色量混合在一起可确定最终颜色。

颜色深度 也称为 “ 位深度 ”，指专门用于每个像素的计算机内存量，因而可以确定图像中可以显示的可能颜色数。

像素 位图图像中的最小信息单位，实际上就是颜色点。

分辨率 图像的像素尺寸，它决定图像中包含的精细细节的级别。分辨率通常表示为用像素数表示的宽度和高度。

RGB 颜色 一种配色方案，其中每个像素的颜色均表示为红、绿和蓝色值的混合颜色。

Bitmap 和 BitmapData 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用位图图像的主要 ActionScript 3.0 类是 Bitmap 类和 BitmapData 类，前者用于在屏幕上显示位图图像，后者用于访问和处理位图的原始图像数据。

更多帮助主题

flash.display.Bitmap

flash.display.BitmapData

了解 Bitmap 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

作为 DisplayObject 类的子类， Bitmap 类是用于显示位图图像的主要 ActionScript 3.0 类。这些图像可能已通过 flash.display.Loader 类加载或使用 Bitmap() 构造函数动态地创建。从外部源加载图像时，Bitmap 对象只能使用 GIF、JPEG 或 PNG 格式的图像。实例化后，可将 Bitmap 实例视为需要呈示在舞台上的 BitmapData 对象的包装。由于 Bitmap 实例是一个显示对象，因此可以使用显示对象的所有特性和功能来操作 Bitmap 实例。有关使用显示对象的详细信息，请参阅第 126 页的 " 显示编程 "。

像素贴紧和平滑

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

除了所有显示对象常见的功能外， Bitmap 类还提供了特定于位图图像的一些附加功能。

Bitmap 类的 pixelSnapping 属性可确定 Bitmap 对象是否贴紧最近的像素。此属性接受 PixelSnapping 类中定义的三个常量之一：ALWAYS、 AUTO 和 NEVER。

应用像素贴紧的语法为：

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

通常，缩放位图图像时，图像会变得模糊或扭曲。若要帮助减少这种扭曲，请使用 BitmapData 类的 smoothing 属性。如果将该布尔值属性设置为 true，当缩放图像时，可使图像中的像素平滑或消除锯齿。它可使图像更加清晰、更加自然。

了解 BitmapData 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

BitmapData 类位于 flash.display 包中，它可以看作是加载的或动态创建的位图图像中包含的像素的照片快照。此快照用对象中的像素数据的数组表示。 BitmapData 类还包含一系列内置方法，可用于创建和处理像素数据。

若要实例化 BitmapData 对象，请使用以下代码：

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

width 和 height 参数指定位图的大小。从 AIR 3 和 Flash Player 11 开始，取消了 BitmapData 对象的大小限制。位图的最大大小取决于操作系统。

在 AIR 1.5 和 Flash Player 10 中，BitmapData 对象的最大宽度或高度为 8,191 像素，并且像素总数不能超过 16,777,215 像素。（因此，如果 BitmapData 对象的宽度为 8,191 像素，则其高度只能为 2,048 像素。）在 Flash Player 9 及早期版本和 AIR 1.1 及早期版本中，高度最大为 2,880 像素，宽度最大为 2,880 像素。

transparent 参数指定位图数据是 (true) 否 (false) 包括 Alpha 通道。 fillColor 参数是一个 32 位颜色值，它指定背景颜色和透明度值 （如果设置为 true）。以下示例创建一个具有 50% 透明的橙色背景的 BitmapData 对象：

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

若要在屏幕上呈示新创建的 BitmapData 对象，请将此对象分配给或包装到 Bitmap 实例中。为此，可以将 BitmapData 对象作为 Bitmap 对象的构造函数参数传递，也可以将此对象分配给现有 Bitmap 实例的 bitmapData 属性。您还必须通过调用将包含该 Bitmap 实例的显示对象容器的 addChild() 或 addChildAt() 方法将该 Bitmap 实例添加到显示列表中。有关使用显示列表的详细信息，请参阅第 132 页的 " 在显示列表中添加显示对象 "。

以下示例创建一个具有红色填充的 BitmapData 对象，并在 Bitmap 实例中显示此对象：

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

处理像素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

BitmapData 类包含一组用于处理像素数据值的方法。

处理单个像素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在像素级别更改位图图像的外观时，您首先需要获取要处理的区域中包含的像素的颜色值。使用 getPixel() 方法可读取这些像素值。

getPixel() 方法从作为参数传递的一组 x, y （像素）坐标中检索 RGB 值。如果您要处理的像素包括透明度 （Alpha 通道）信息，则需要使用 getPixel32() 方法。此方法也可以检索 RGB 值，但与 getPixel() 不同， getPixel32() 返回的值包含表示所选像素的 Alpha 通道 （透明度）值的附加数据。

或者，如果只想更改位图中包含的某个像素的颜色或透明度，则可以使用 setPixel() 或 setPixel32() 方法。若要设置像素的颜色，只需将 x、 y 坐标和颜色值传递给这两种方法之一即可。

以下示例使用 setPixel() 在绿色 BitmapData 背景上绘制交叉形状。然后，此示例使用 getPixel() 从坐标 50, 50 处的像素中检索颜色值并跟踪返回的值。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapData:BitmapData = new BitmapData(100, 100, false, 0x009900);

for (var i:uint = 0; i < 100; i++)
{
    var red:uint = 0xFF0000;
    myBitmapData.setPixel(50, i, red);
    myBitmapData.setPixel(i, 50, red);
}

var myBitmapImage:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmapImage);

var pixelValue:uint = myBitmapData.getPixel(50, 50);
trace(pixelValue.toString(16));
```

如果要读取一组像素而不是单个像素的值，请使用 getPixels() 方法。此方法从作为参数传递的矩形像素数据区域中生成字节数组。字节数组的每个元素 （即像素值）都是无符号的整数 （32 位未经相乘的像素值）。

相反，若要更改 （或设置）一组像素的值，请使用 setPixels() 方法。此方法需要联合使用两个参数 （rect 和 inputByteArray）来输出像素数据 (inputByteArray) 的矩形区域 (rect)。

从 inputByteArray 中读取 （或写入）数据时，会为数组中的每个像素调用 ByteArray.readUnsignedInt() 方法。如果由于某些原因， inputByteArray 未包含像素数据的整个矩形，则该方法会停止处理该点处的图像数据。

必须记住的是，对于获取和设置像素数据，字节数组需要有 32 位 Alpha、红、绿、蓝 (ARGB) 像素值。

以下示例使用 getPixels() 和 setPixels() 方法将一组像素从一个 BitmapData 对象复制到另一个对象：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.utils.ByteArray;
import flash.geom.Rectangle;

var bitmapDataObject1:BitmapData = new BitmapData(100, 100, false, 0x006666FF);
var bitmapDataObject2:BitmapData = new BitmapData(100, 100, false, 0x00FF0000);

var rect:Rectangle = new Rectangle(0, 0, 100, 100);
var bytes:ByteArray = bitmapDataObject1.getPixels(rect);

bytes.position = 0;
bitmapDataObject2.setPixels(rect, bytes);

var bitmapImage1:Bitmap = new Bitmap(bitmapDataObject1);
addChild(bitmapImage1);
var bitmapImage2:Bitmap = new Bitmap(bitmapDataObject2);
addChild(bitmapImage2);
bitmapImage2.x = 110;
```

像素级别冲突检测

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

BitmapData.hitTest() 方法可以在位图数据和另一个对象或点之间执行像素级别冲突检测。

BitmapData.hitTest() 方法接受五个参数：

- firstPoint (Point)：此参数指在其上执行点击测试的第一个 BitmapData 的左上角的像素位置。
- firstAlphaThreshold (uint)：此参数指定对于此点击测试视为不透明的最高 Alpha 通道值。
- secondObject (Object)：此参数表示影响区域。secondObject 对象可以是 Rectangle、 Point、 Bitmap 或 BitmapData 对象。此对象表示在其上执行冲突检测的点击区域。
- secondBitmapDataPoint (Point)：此可选参数用于在第二个 BitmapData 对象中定义像素位置。只有当 secondObject 的值为 BitmapData 对象时，才使用此参数。默认值为 null。
- secondAlphaThreshold (uint)：此可选参数表示在第二个 BitmapData 对象中视为不透明的最高 Alpha 通道值。默认值为 1。仅当 secondObject 的值是 BitmapData 对象并且这两个 BitmapData 对象都透明时，才使用此参数。

在不透明图像上执行冲突检测时，请牢记， ActionScript 会将图像视为完全不透明的矩形 （或边框）。或者，在透明的图像上执行像素级别点击测试时，需要两个图像都是透明的。除此之外， ActionScript 还使用 Alpha 阈值参数来确定像素在哪点开始从透明变为不透明。

以下示例创建三个位图图像并使用两个不同冲突点 （一个返回 false，另一个返回 true）检查像素冲突：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.geom.Point;

var bmd1:BitmapData = new BitmapData(100, 100, false, 0x000000FF);
var bmd2:BitmapData = new BitmapData(20, 20, false, 0x00FF3300);

var bm1:Bitmap = new Bitmap(bmd1);
this.addChild(bm1);

// Create a red square.
var redSquare1:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare1);
redSquare1.x = 0;

// Create a second red square.
var redSquare2:Bitmap = new Bitmap(bmd2);
this.addChild(redSquare2);
redSquare2.x = 150;
redSquare2.y = 150;

// Define the point at the top-left corner of the bitmap.
var pt1:Point = new Point(0, 0);
// Define the point at the center of redSquare1.
var pt2:Point = new Point(20, 20);
// Define the point at the center of redSquare2.
var pt3:Point = new Point(160, 160);

trace(bmd1.hitTest(pt1, 0xFF, pt2)); // true
trace(bmd1.hitTest(pt1, 0xFF, pt3)); // false
```

复制位图数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要从一个图像向另一个图像复制位图数据，您可以使用以下几种方法：clone()、 copyPixels()、 copyChannel()、 draw() 和 drawWithQuality() （drawWithQuality 方法可在 Flash Player 11.3 和更高版本中使用；也可在 AIR 3.3 和更高版本中使用）。

正如名称的含义一样，clone() 方法允许您将位图数据从一个 BitmapData 对象克隆或采样到另一个对象。调用此方法时，此方法返回一个新的 BitmapData 对象，它是与被复制的原始实例完全一样的克隆。

以下示例克隆橙色 （父级）正方形的一个副本，并将克隆放在原始父级正方形的旁边：

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myParentSquareBitmap:BitmapData = new BitmapData(100, 100, false, 0x00ff3300);
var myClonedChild:BitmapData = myParentSquareBitmap.clone();

var myParentSquareContainer:Bitmap = new Bitmap(myParentSquareBitmap);
this.addChild(myParentSquareContainer);

var myClonedChildContainer:Bitmap = new Bitmap(myClonedChild);
this.addChild(myClonedChildContainer);
myClonedChildContainer.x = 110;
```

copyPixels() 方法是一种将像素从一个 BitmapData 对象复制到另一个 BitmapData 对象的快速且简便的方法。该方法会拍摄源图像的矩形快照 （由 sourceRect 参数定义），并将其复制到另一个矩形区域 （大小相等）。新 “ 粘贴 ” 的矩形的位置在 destPoint 参数中定义。

copyChannel() 方法从源 BitmapData 对象中采集预定义的颜色通道值 （Alpha、红、绿或蓝），并将此值复制到目标 BitmapData 对象的通道中。调用此方法不会影响目标 BitmapData 对象中的其他通道。

draw() 和 drawWithQuality() 方法将源子画面、影片剪辑或其他显示对象中的图形内容绘制或呈现在新位图上。使用 matrix、colorTransform、blendMode 和目标 clipRect 参数，可以修改新位图的呈示方式。此方法使用 Flash Player 和 AIR 中的矢量渲染器生成数据。

调用 draw() 或 drawWithQuality() 时，需要将源对象 （子画面、影片剪辑或其他显示对象）作为第一个参数传递，如下所示：

```
myBitmap.draw(movieClip);
```

如果源对象在最初加载后应用了变形 （颜色、矩阵等等），则不能将这些变形复制到新对象。如果想要将变形复制到新位图，则需要将 transform 属性的值从原始对象复制到使用新 BitmapData 对象的 Bitmap 对象的 transform 属性中。

压缩位图数据

Flash Player 11.3 和更高版本， **AIR 3.3** 和更高版本

flash.display.BitmapData.encode() 方法允许您将位图数据本机压缩为以下图像压缩格式之一：

- **PNG** - 使用 PNG 压缩，可以选择使用快速压缩，它强调的是压缩速度而不是文件大小。若要使用 PNG 压缩，请将新的 flash.display.PNGEncoderOptions 对象作为 BitmapData.encode() 方法的第二个参数传递。
- **JPEG** - 使用 JPEG 压缩，可以选择指定图像品质。若要使用 JPEG 压缩，请将新的 flash.display.JPEGEncoderOptions 对象作为 BitmapData.encode() 方法的第二个参数传递。
- **JPEGXR** - 使用 JPEG 扩展范围 (XR) 压缩，可以选择指定颜色通道、损耗和熵 (entropy) 编码设置。若要使用 JPEGXR 压缩，请将新的 flash.display.JPEGXREncoderOptions 对象作为 BitmapData.encode() 方法的第二个参数传递。

您可以将图像处理的此功能用作服务器上传或下载工作流程的一部分。

下面的示例代码片段使用 JPEGEncoderOptions 压缩 BitmapData 对象：

```
// Compress a BitmapData object as a JPEG file.
var bitmapData:BitmapData = new BitmapData(640,480,false,0x00FF00);
var byteArray:ByteArray = new ByteArray();
bitmapData.encode(new Rectangle(0,0,640,480), new flash.display.JPEGEncoderOptions(), byteArray);
```

使用杂点功能制作纹理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要修改位图的外观，可以使用 noise() 方法或 perlinNoise() 方法对位图应用杂点效果。可以把杂点效果比作未调谐的电视屏幕的静态外观。

若要对位图应用杂点效果，请使用 noise() 方法。此方法对位图图像的指定区域中的像素应用随机颜色值。

此方法接受五个参数：

- randomSeed (int)：决定图案的随机种子数。不管名称具有什么样的含义，只要传递的数字相同，此数字就会生成相同的结果。为了获得真正的随机结果，请使用 Math.random() 方法为此参数传递随机数字。
- low (uint)：此参数指要为每个像素生成的最低值 （0 至 255）。默认值为 0。将此参数设置为较低值会产生较暗的杂点图案，而将此参数设置为较高值会产生较亮的图案。

- high (uint)：此参数指要为每个像素生成的最高值 （0 至 255）。默认值为 255。将此参数设置为较低值会产生较暗的杂点图案，而将此参数设置为较高值会产生较亮的图案。
- channelOptions (uint)：此参数指定将向位图对象的哪个颜色通道应用杂点图案。此数字可以是四个颜色通道 ARGB 值的任意组合。默认值为 7。
- grayScale (Boolean)：设置为 true 时，此参数对位图像素应用 randomSeed 值，可有效地褪去图像中的所有颜色。此参数不影响 Alpha 通道。默认值为 false。

以下示例创建一个位图图像，并对它应用蓝色杂点图案：

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise1 extends Sprite
    {
        public function BitmapNoise1()
        {
            var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
            myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
            var image:Bitmap = new Bitmap(myBitmap);
            addChild(image);
        }
    }
}
```

如果要创建更好的有机外观纹理，请使用 perlinNoise() 方法。 perlinNoise() 方法可生成逼真、有机的纹理，是用于烟雾、云彩、水、火或爆炸的理想图案。

由于 perlinNoise() 方法是由算法生成的，因此它使用的内存比基于位图的纹理少。但还是会对处理器的使用有影响，特别是对于旧计算机，会降低内容的处理速度，使屏幕重绘的速度比帧速率慢。这主要是因为需要进行浮点计算，以便处理 Perlin 杂点算法。

此方法接受九个参数 （前六个是必需参数）：

- baseX (Number)：决定创建的图案的 x （大小）值。
- baseY (Number)：决定创建的图案的 y （大小）值。
- numOctaves (uint)：要组合以创建此杂点的 octave 函数或各个杂点函数的数目。 octave 数目越大，创建的图像越精细，但这需要更多的处理时间。
- randomSeed (int)：随机种子数的功能与在 noise() 函数中的功能完全相同。为了获得真正的随机结果，请使用 Math.random() 方法为此参数传递随机数字。
- stitch (Boolean)：如果设置为 true，则此方法尝试缝合 （或平滑）图像的过渡边缘以形成无缝的纹理，用于作为位图填充进行平铺。
- fractalNoise (Boolean)：此参数与此方法生成的渐变的边缘有关。如果设置为 true，则此方法生成的碎片杂点会对效果的边缘进行平滑处理。如果设置为 false，则将生成湍流。带有湍流的图像具有可见的不连续性渐变，可以使用它处理更接近锐化的视觉效果，例如，火焰或海浪。
- channelOptions (uint)：channelOptions 参数的功能与在 noise() 方法中的功能完全相同。它指定对哪个颜色通道 （在位图上）应用杂点图案。此数字可以是四个颜色通道 ARGB 值的任意组合。默认值为 7。
- grayScale (Boolean)：grayScale 参数的功能与在 noise() 方法中的功能完全相同。如果设置为 true，则对位图像素应用 randomSeed 值，可有效地褪去图像中的所有颜色。默认值为 false。

- offsets (Array): 对应于每个 octave 的 x 和 y 偏移的点数组。通过处理偏移值，可以平滑滚动图像层。偏移数组中的每个点将影响一个特定的 octave 杂点函数。默认值为 null。

以下示例创建一个 150 x 150 像素的 BitmapData 对象，该对象调用 perlinNoise() 方法来生成绿色和蓝色的云彩效果：

```
package
{
    import flash.display.Sprite;
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.BitmapDataChannel;

    public class BitmapNoise2 extends Sprite
    {
        public function BitmapNoise2()
        {
            var myBitmapDataObject:BitmapData =
                new BitmapData(150, 150, false, 0x00FF0000);

            var seed:Number = Math.floor(Math.random() * 100);
            var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
            myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

            var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
            addChild(myBitmap);
        }
    }
}
```

滚动位图

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

设想您创建了一个街道图应用程序，每次用户移动该图时，都需要您更新视图 （即使该图只移动了几个像素）。

创建此功能的一种方式是，每次用户移动街道图时，均重新呈示包含更新的街道图视图的新图像。或者，创建一个大型图像，并使用 scroll() 方法。

scroll() 方法可以复制屏幕上的位图，然后将它粘贴到由 (x, y) 参数指定的新偏移位置。如果位图的一部分恰巧在舞台以外，则会产生图像发生移位的效果。与计时器函数 （或 enterFrame 事件）配合使用时，可以使图像呈示动画或滚动效果。

以下示例采用前面的 Perlin 杂点示例并生成较大的位图图像 （其四分之三呈示在舞台外面）。然后应用 scroll() 方法和一个 enterFrame 事件侦听器，使图像在对角线向下方向偏移一个像素。每次进入帧时均会调用此方法，因此，随着图像向下滚动，图像位于屏幕以外的部分会呈现在舞台上。

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(1000, 1000, false, 0x00FF0000);
var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE;
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
myBitmap.x = -750;
myBitmap.y = -750;
addChild(myBitmap);

addEventListener(Event.ENTER_FRAME, scrollBitmap);

function scrollBitmap(event:Event):void
{
    myBitmapDataObject.scroll(1, 1);
}
```

利用 mipmap 处理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

“MIP map” （也称为 “mipmap”）是组合在一起并与纹理关联的位图，可改善运行时呈示品质和性能。 MIP map 中的每个位图图像分别是主位图图像的一个版本，但与主图像相比，其详细程度有所降低。

例如，您的 MIP map 可以包括 64 × 64 像素的最高品质的主图像。 MIP map 中的较低品质图像将为 32 × 32、 16 × 16、 8 × 8、 4 × 4、 2 × 2 和 1 × 1 像素。

“ 纹理流式处理 ” 是首先加载最低品质的位图，然后在加载位图时逐渐显示较高品质的位图的功能。因为较低品质的位图较小，所以它们的加载速度比主图像快。因此，应用程序用户可以在高品质的主位图加载之前，在应用程序中查看图像。

Flash Player 9.115.0 和更高版本及 AIR 通过创建每个位图的不同显示比例 （从 50% 开始）的优化版本来实现此技术 （该过程称为 “mipmap 处理 ”）。

Flash Player 11.3 和 AIR 3.3 支持通过 Context3D.createCubeTexture() 和 Context3D.createTexture() 方法的 streamingLevels 参数执行纹理流式处理。

使用纹理压缩可以将纹理图像以压缩格式直接存储在 GPU 中，从而节省 GPU 内存和内存带宽。通常，纹理的压缩是脱机进行的，且压缩后的纹理以压缩格式上传到 GPU。不过， Flash Player 11.4 和 AIR 3.4 支持运行时纹理压缩，这在某些情况下非常有用，比如渲染矢量图中的动态纹理时。要使用运行时纹理压缩，可执行以下步骤：

- 通过调用 Context3D.createTexture() 方法创建纹理对象，在第三个参数中传递 flash.display3D.Context3DTextureFormat.COMPRESSED 或 flash.display3D.Context3DTextureFormat.COMPRESSED_ALPHA。
- 使用 createTexture() 返回的 flash.display3D.textures.Texture 实例，调用 flash.display3D.textures.Texture.uploadFromBitmapData() 或 flash.display3D.textures.Texture.uploadFromByteArray()。这些方法会在单独一个步骤中执行纹理的上传和压缩。

为以下位图类型创建 MIP 映射：

- 使用 ActionScript 3.0 Loader 类显示的位图 （JPEG、 GIF 或 PNG 文件）
- Flash Professional 文档库中的位图
- BitmapData 对象

- 使用 ActionScript 2.0 loadMovie() 函数显示的位图

MIP map 不适用于应用滤镜的对象或缓存位图的影片剪辑。不过，如果应用滤镜的显示对象中包含位图转换，则即使位图位于被遮罩的内容中，也会应用 MIP map。

Mipmap 处理是自动执行的，但您可以遵循几条准则，以确保您的图像利用此优化技术：

- 对于视频播放，请将 Video 对象的 smoothing 属性设置为 true （请参阅 Video 类）。
- 对于位图，不一定要将 smoothing 属性设置为 true，但当位图使用平滑处理时品质的改善更为显著。
- 对于二维图像，请使用可被 4 或 8 整除的位图大小（如 640 x 128，可按如下方式递减：320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1）。

  对于三维纹理，使用其中每个图像的分辨率为 2 的幂 （即 2^n）的 MIP map。例如，主图像的分辨率为 1024 x 1024 像素。这样，MIP map 中较低品质的图像将为 512 x 512、256 x 256、128 x 128，直至 1 x 1 像素，分别针对 MIP map 中的总共 11 个图像。

  请注意，不会对宽度或高度为奇数的位图内容执行 Mipmap 处理。

位图示例：带动画效果的旋转的月亮

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

旋转的月球动画示例说明了位图对象和位图图像数据 （BitmapData 对象）的使用方法。该示例使用月球表面的平面图像作为原始图像数据来创建一个旋转的月球动画。将对以下方法进行说明：

- 加载外部图像并访问其原始图像数据
- 通过重复复制源图像不同部分的像素创建动画
- 通过设置像素值创建位图图像

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/SpinningMoon 文件夹中找到 " 旋转的月球动画 " 的应用程序文件。该应用程序包含以下文件：

文件	说明
SpinningMoon.mxml 或 SpinningMoon.fla	Flex (MXML) 或 Flash (FLA) 中的主应用程序文件。
com/example/programmingas3/moon/MoonSphere.as	用于执行加载、显示月球和创建月球动画的功能的类。
moonMap.png	包含月球表面照片的图像文件 （将加载此图像文件并使用它来创建旋转的月球动画）。

将外部图像作为位图数据加载

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此范例执行的第一项主要任务是加载外部图像文件，即月球表面照片。加载操作由 MoonSphere 类中的两个方法处理：MoonSphere() 构造函数 （启动加载过程）和 imageLoadComplete() 方法 （完成外部图像加载后调用该方法）。

加载外部图像与加载外部 SWF 类似，两者都使用 flash.display.Loader 类的实例执行加载操作。启动图像加载的 MoonSphere() 方法中的实际代码如下：

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

第一行声明名为 imageLoader 的 Loader 实例。第三行通过调用 Loader 对象的 load() 方法并传递一个 URLRequest 实例（表示要加载的图像的 URL）来实际启动加载过程。第二行设置在完成图像加载时将触发的事件侦听器。请注意：不是对 Loader 实例本身调用 addEventListener() 方法；而是对 Loader 对象的 contentLoaderInfo 属性调用该方法。Loader 实例本身不会调度与所加载内容相关的事件。然而，它的 contentLoaderInfo 属性包含对 LoaderInfo 对象的引用，该对象与加载到 Loader 对象的内容 （本例中为外部图像）相关联。该 LoaderInfo 对象提供与外部内容加载进度及加载完成有关的几个事件，其中包括 complete 事件 (Event.COMPLETE)，该事件将在完成图像加载时触发对 imageLoadComplete() 方法的调用。

启动外部图像加载是加载过程中的重要部分，而了解加载完成时要执行什么操作也同等重要。如以上代码所示，完成加载图像后将调用 imageLoadComplete() 函数。该函数对加载的图像数据执行一些操作，稍后将进行说明。然而，若要使用图像数据，还需要访问该数据。当使用某 Loader 对象来加载外部图像时，加载的图像将变为一个 Bitmap 实例，并附加为该 Loader 对象的一个子显示对象。在本例中， Loader 实例可作为事件对象的一部分用于事件侦听器方法，此事件对象将作为参数传递给该方法。 imageLoadComplete() 方法的前几行如下：

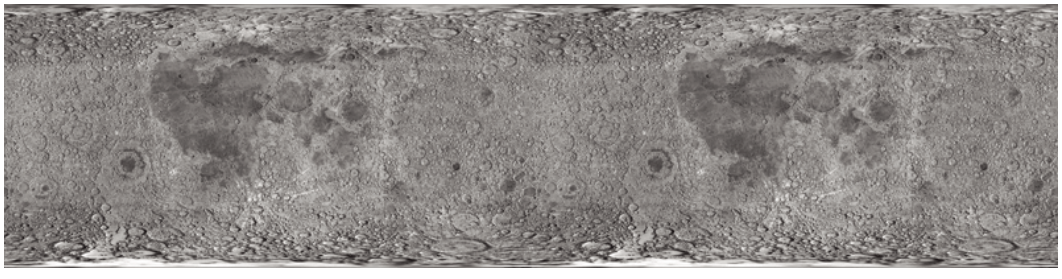
```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

请注意，事件对象参数名为 event，它是 Event 类的一个实例。Event 类的每个实例都具有一个 target 属性，该属性将引用触发事件的对象 （本例中为 LoaderInfo 实例，如前所述，将对该实例调用 addEventListener() 方法。）而 LoaderInfo 对象又具有一个 content 属性，加载过程完成后，该属性将包含 Bitmap 实例，其中具有加载的位图图像。如果要在屏幕上直接显示该图像，则可以将此 Bitmap 实例 (event.target.content) 附加到一个显示对象容器。（也可以将 Loader 对象附加到一个显示对象容器。）但是，在该示例中，加载的内容将用作原始图像数据的源而不是显示在屏幕上。因此， imageLoadComplete() 方法的第一行读取加载的 Bitmap 实例的 bitmapData 属性 (event.target.content.bitmapData)，并将其存储在名为 textureMap 的实例变量中，该变量用作创建旋转的月球动画的图像数据源。将稍后对此进行说明。

通过复制像素创建动画

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

动画的基本定义是通过随时间变化改变图像而产生的运动或变化的视觉效果。此示例的目标是创建月球绕其垂直轴旋转的视觉效果。然而，对于动画而言，您可以忽略该示例中球形扭曲方面的问题。假设已加载并用作月球图像数据源的实际图像如下：



如您所见，该图像并不是一个或几个球体；它是月球表面的一张矩形照片。因为该照片刚好是在月球赤道上拍摄的，所以图像中靠近图像顶部和底部的部分发生拉伸和扭曲。若要消除图像的扭曲使其具有球形外观，我们将使用置换图滤镜 （在后面进行介绍）。但是，因为该源图像是矩形，所以若要产生旋转球体的视觉效果，代码只要能完成水平滑动月球表面照片的操作即可。

请注意，该图像实际上由月球表面照片的两个副本彼此相接而成。该图像是要从中重复复制图像数据来创建动画外观的源图像。通过两个图像副本彼此相接，会更容易产生连续、不间断的滚动效果。让我们逐步浏览动画生成的过程来看看这是如何实现的。

该过程实际上涉及两个单独的 ActionScript 对象。首先，有加载的源图像，在代码中由名为 textureMap 的 BitmapData 实例表示。如前所述，外部图像加载后，将用图像数据来填充 textureMap，使用的代码如下：

```
textureMap = event.target.content.bitmapData;
```

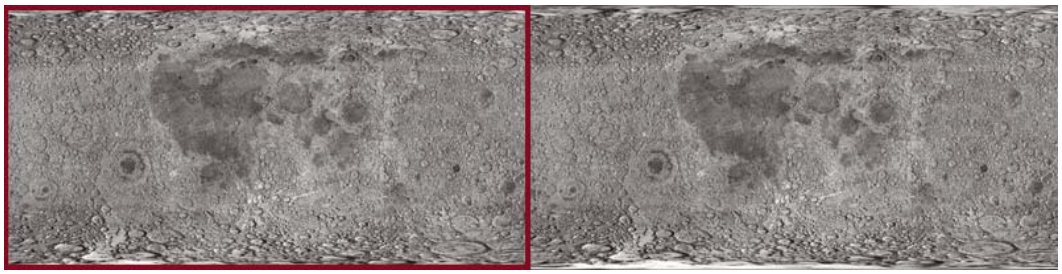
textureMap 的内容是矩形的月球图像。另外，为了产生旋转动画，该代码使用名为 sphere 的 Bitmap 实例，该实例是在屏幕上显示月球图像的实际显示对象。与 textureMap 一样，sphere 对象也是在 imageLoadComplete() 方法中创建并使用其初始图像数据填充，使用的代码如下：

```
sphere = new Bitmap();
sphere.bitmapData = new BitmapData(textureMap.width / 2, textureMap.height);
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(0, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

如代码所示，sphere 被实例化了。其 bitmapData 属性（通过 sphere 显示的原始图像数据）具有与 textureMap 相同的高度和一半的宽度。换句话说，sphere 的内容将是一幅月球照片的大小（因为 textureMap 图像包含并排的两幅月球照片）。接下来，用图像数据填充 bitmapData 属性，填充时使用的是 copyPixels() 方法。 copyPixels() 方法调用中的参数指示以下几点：

- 第一个参数指示从 textureMap 复制图像数据。
- 第二个参数（新的 Rectangle 实例）指定图像快照应该从 textureMap 的哪部分拍摄；在本例中，快照是从 textureMap 左上角开始的一个矩形（由前两个 Rectangle() 参数 0, 0 指示），矩形快照的宽度和高度与 sphere 的 width 和 height 属性一致。
- 第三个参数（新的 Point 实例）的 x 和 y 值都为 0，它定义了像素数据的目标位置 — 本例中为 sphere.bitmapData 的左上角 (0, 0)。

从视觉表示形式上看，该代码将复制下图中用轮廓线标出的、textureMap 的像素，并将其粘贴到 sphere 上。换句话说，sphere 的 BitmapData 内容是这里加亮的 textureMap 部分：



然而请记住，这只是 sphere 的初始状态 — 复制到 sphere 上的第一项图像内容。

加载源图像并创建 sphere 之后，由 imageLoadComplete() 方法执行的最终任务是设置动画。动画由名为 rotationTimer 的 Timer 实例驱动，该实例由以下代码创建并启动：

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

代码首先创建名为 rotationTimer 的 Timer 实例；传递给 Timer() 构造函数的参数指示 rotationTimer 应每 15 毫秒触发一次其 timer 事件。接下来将调用 addEventListener() 方法，以指定在发生 timer 事件 (TimerEvent.TIMER) 时调用 rotateMoon() 方法。最后，计时器实际上是通过调用其 start() 方法启动的。

根据 rotationTimer 的定义方式，约每 15 毫秒 Flash Player 调用一次 MoonSphere 类中的 rotateMoon() 方法（用于产生月球动画）。 rotateMoon() 方法的源代码如下：

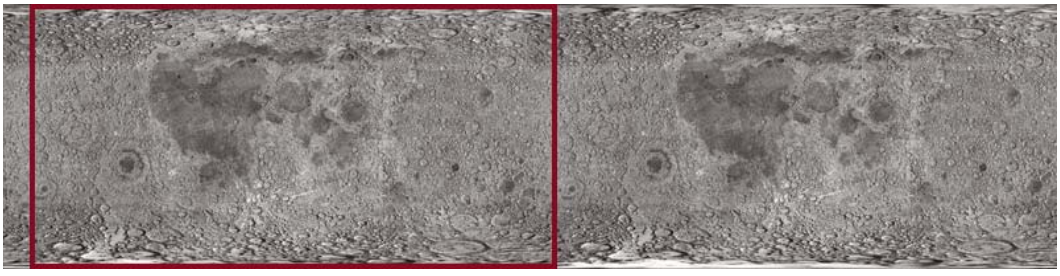
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.Data.copyPixels(textureMap,
                                    new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                    new Point(0, 0));

    event.updateAfterEvent();
}
```

该代码实现以下三方面的操作：

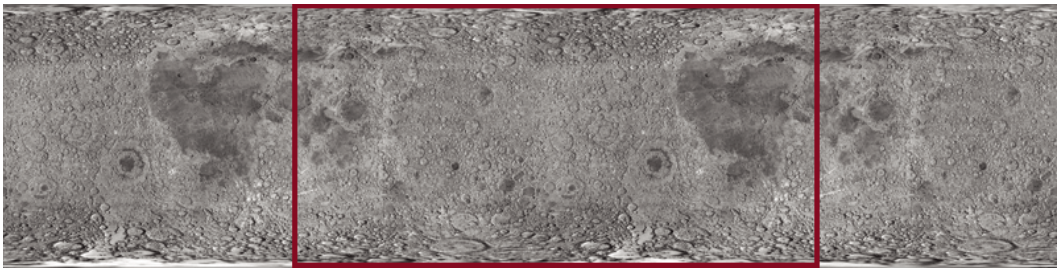
1 变量 sourceX 的值（最初设为 0）增加 1。

```
sourceX += 1;
```

您将看到，sourceX 用于确定 textureMap 中的位置（从该位置将像素复制到 sphere），因此该代码会产生在 textureMap 上将矩形向右移动一个像素的效果。返回到视觉表示形式，经过几个动画循环之后，源矩形将向右移动几个像素，如下所示：

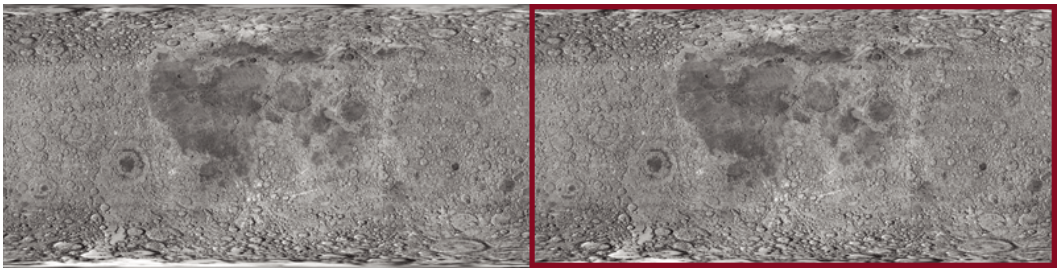


经过几个循环之后，矩形将进一步移动：



像素复制位置的这种平稳渐进式移动是动画制作的关键。通过缓慢、连续地将源位置移动到右侧，sphere 中显示在屏幕上的图像显示为连续地滑向左侧。这就是源图像 (textureMap) 需要两个月球表面照片副本的原因。由于矩形连续移动到右侧，因此大多数时间该矩形不是在一张月球照片上而是与两张月球照片发生重叠。

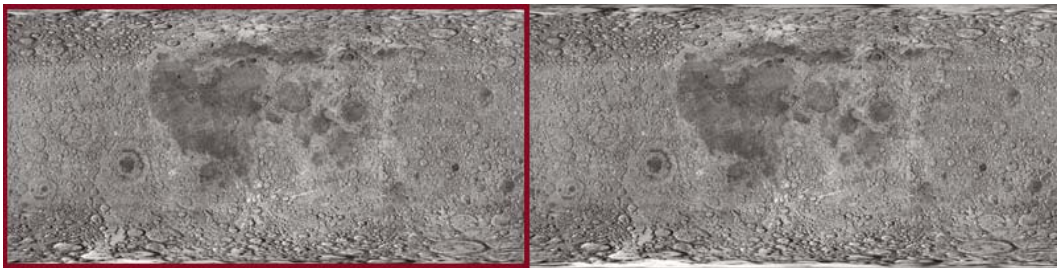
2 随着源矩形缓慢移到右侧，会出现一个问题。最后，矩形将到达 textureMap 的右边缘，它将用完要复制到 sphere 上的月球照片像素：



下一行代码将解决这个问题：

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

该代码检查 sourceX （矩形的左边缘）是否已到达 textureMap 的中部。如果是，它会将 sourceX 重置为 0，将其移回到 textureMap 的左边缘并重新开始循环：



3 计算出适当的 sourceX 值后，创建动画的最后一步是将新的源矩形像素实际复制到 sphere 上。实现这一操作的代码与最初填充 sphere 的代码 （如前面所述）非常类似；唯一的不同是：在本例的 new Rectangle() 构造函数调用中，矩形的左边缘位于 sourceX：

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

请记住，此代码每 15 毫秒重复调用一次。由于源矩形的位置是连续移动的，并且像素被复制到 sphere 上，因此在屏幕上显示为由 sphere 表示的月球照片图像发生连续滑动。换句话说，月球显示为连续旋转。

创建球形外观

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当然，月球是一个球体而不是一个矩形。由于要形成连续动画，因此该示例需要拍摄矩形的月球表面照片，并将其转换成球体。这涉及两个单独的步骤：一个用于隐藏月球表面照片中除圆形区域之外的所有内容的遮罩，以及一个用于扭曲月球照片外观使其具有三维外观的置换图滤镜。

首先，圆形遮罩用于隐藏 MoonSphere 对象中除通过滤镜创建的球体之外的所有内容。以下代码创建一个作为 Shape 实例的遮罩，并将其应用为 MoonSphere 实例的遮罩：

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

请注意，由于 MoonSphere 是一个显示对象 （它基于 Sprite 类），因此可以使用该显示对象继承的 mask 属性将遮罩直接应用于 MoonSphere 实例。

仅使用圆形遮罩隐藏照片的某些部分不足以创建逼真的旋转球体效果。由于月球表面照片拍摄方式的限制，导致照片的尺寸不成比例；与赤道上的部分相比，图像上越靠近图像顶部或底部的部分扭曲和拉伸得越严重。为了对月球照片的外观进行变形以使其具有三维效果，我们将使用置换图滤镜。

置换图滤镜是一种用于扭曲图像的滤镜。在本例中，将通过水平挤压图像的顶部和底部而保持中部不变来 “ 扭曲 ” 月球照片，从而使其看起来更加逼真。假设对照片的正方形部分执行滤镜操作，挤压顶部和底部而不挤压中部将使正方形变为圆形。为该扭曲图像添加动画效果所产生的另一效果是：与靠近顶部和底部的区域相比，图像的中部看起来所移动的实际像素距离更大，这将产生圆实际上是一个三维对象 （球体）的视觉效果。

以下代码用于创建名为 displaceFilter 的置换图滤镜：

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

第一个参数 fisheyeLens 称为置换图图像；在本例中，它是以编程方式创建的 BitmapData 对象。将在第 219 页的 “ 通过设置像素值创建位图图像 ”中介绍创建该图像的说明。其他参数说明过滤后的图像中应该应用滤镜的位置、使用哪些颜色通道来控制置换效果以及将在何种范围内对置换产生影响。一旦创建置换图滤镜，它将应用于仍位于 imageLoadComplete() 方法中的 sphere：

```
sphere.filters = [displaceFilter];
```

应用了遮罩和置换图滤镜的最终图像如下所示：

每个旋转月球动画循环之后， sphere 的 BitmapData 内容将由新的源图像数据快照覆盖。但是，无需每次都重新应用滤镜。这是因为滤镜应用到了 Bitmap 实例 （显示对象）而不是位图数据 （原始像素信息）。请记住， Bitmap 实例不是实际的位图数据；它是在屏幕上显示位图数据的显示对象。举例来说， Bitmap 实例就像用于在屏幕上显示照片幻灯片的幻灯片放映机，而 BitmapData 对象就像可以通过幻灯片放映机显示的实际照片幻灯片。滤镜可以直接应用于 BitmapData 对象，这与直接在照片幻灯片上绘图以更改图像相似。滤镜也可以应用于任何显示对象 （包括 Bitmap 实例）；这与在幻灯片放映机的镜头前面放一个滤镜以使屏幕上显示的输出变形 （丝毫不会更改原始幻灯片）相似。因为可通过 Bitmap 实例的 bitmapData 属性来访问原始位图数据，所以滤镜可能已直接应用于原始位图数据。然而，在本例中，将滤镜应用于 Bitmap 显示对象比应用于位图数据更有意义。

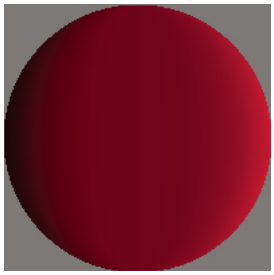
有关在 ActionScript 中使用置换图滤镜的详细信息，请参阅第 224 页的 “ 过滤显示对象 ”。

通过设置像素值创建位图图像

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

置换图滤镜的一个重要方面是它实际上涉及两个图像。一个图像为源图像，即由滤镜实际更改的图像。在该示例中，源图像是名为 sphere 的 Bitmap 实例。滤镜所用的另一个图像称为映射图像。映射图像不实际显示在屏幕上。相反，它的每个像素的颜色都用作置换函数的输入 — 置换图图像中位于特定 x、y 坐标的像素的颜色决定应用于源图像中位于该 x、y 坐标的像素的置换量 （物理位移）。

因此，为了使用置换图滤镜创建球体效果，该范例需要合适的置换图图像 — 一个包含灰色背景和用单一颜色 （红色）从暗到亮水平渐变填充的圆的图像，如下图所示：



因为该示例中仅使用一个映射图像和滤镜，所以在 imageLoadComplete() 方法中 （换句话说，外部图像完成加载时）只创建一次映射图像。通过调用 MoonSphere 类的 createFisheyeMap() 方法来创建名为 fisheyeLens 的置换图图像：

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

在 createFisheyeMap() 方法中，在使用 BitmapData 类的 setPixel() 方法绘制置换图图像时，实际上每次只绘制一个像素。下面列出了 createFisheyeMap() 方法的完整代码，并跟有操作方式的逐步说明：

```
private function createFisheyeMap(radius:int):BitmapData
{
    var diameter:int = 2 * radius;

    var result:BitmapData = new BitmapData(diameter,
                                           diameter,
                                           false,
                                           0x808080);

    // Loop through the pixels in the image one by one
    for (var i:int = 0; i < diameter; i++)
    {
        for (var j:int = 0; j < diameter; j++)
        {
            // Calculate the x and y distances of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctX:Number = (i - radius) / radius;
            var pctY:Number = (j - radius) / radius;

            // Calculate the linear distance of this pixel from
            // the center of the circle (as a percentage of the radius).
            var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);

            // If the current pixel is inside the circle,
            // set its color.
            if (pctDistance < 1)
            {
                // Calculate the appropriate color depending on the
                // distance of this pixel from the center of the circle.
                var red:int;
                var green:int;
                var blue:int;
                var rgb:uint;
                red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
                green = 0;
                blue = 0;
                rgb = (red << 16 | green << 8 | blue);
                // Set the pixel to the calculated color.
                result.setPixel(i, j, rgb);
            }
        }
    }
    return result;
}
```

首先，调用该方法时，将接收参数 radius，指示要创建的圆形图像的半径。接下来，该代码将创建 BitmapData 对象 （将在该对象上绘制圆）。该对象名为 result，最终将作为该方法的返回值传递回来。如以下代码片断所示， result BitmapData 实例将使用与圆直径相同的宽度和高度创建，且不透明 （第三个参数为 false），用颜色 0x808080 （中灰色）预先填充：

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

接下来，该代码使用两个循环遍历图像的每个像素。外部循环从左到右遍历图像的每一列 （使用变量 i 表示当前所操作的像素的水平位置），而内部循环从上到下遍历当前列的每个像素 （使用变量 j 表示当前像素的垂直位置）。下面显示了循环的代码（省略了内部循环的内容）：

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

由于循环逐个遍历像素，因此会为每个像素计算一个值 （映射图像中该像素的颜色值）。此过程包含四个步骤：

1 代码计算当前像素沿 x 轴距离圆心的距离 (i - radius)。该值除以半径得到半径的百分比而不是绝对距离 ((i - radius) / radius)。该百分比值存储在名为 pctX 的变量中，而且会计算沿 y 轴的等效值并存储在变量 pctY，如以下代码所示：

```
var pctX:Number = (i - radius) / radius;
var pctY:Number = (j - radius) / radius;
```

2 使用标准三角公式 （勾股定理），通过 pctX 和 pctY 计算圆心和当前点之间的直线距离。该值存储在名为 pctDistance 的变量中，如下所示：

```
var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
```

3 接下来，代码将检查距离百分比是否小于 1 （也就是半径的 100%，或者换句话说，所考虑的像素是否处于圆的半径范围之内。如果该像素落在圆内，将被指定一个计算得到的颜色值 （此处省略，在步骤 4 中介绍）；如果不落在圆内，该像素将不发生任何变化，其颜色保留为默认的中灰色：

```
if (pctDistance < 1)
{
    ...
}
```

4 对于落在圆内的那些像素，将为其计算一个颜色值。最终颜色将呈现红色色调，范围从圆左边缘的黑 (0%) 红到圆右边缘的亮 (100%) 红。颜色值最初按三部分 （红、绿和蓝）计算，如下所示：

```
red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
green = 0;
blue = 0;
```

请注意，实际上只有颜色的红色部分 （变量 red）有一个值。为清楚起见，此处显示了绿色和蓝色值 （变量 green 和 blue），不过可以省略。因为该方法的目的是创建一个包含红色渐变的圆，所以不需要绿色和蓝色。

三个颜色值都确定之后，将使用标准移位算法组合成一个整数颜色值，如以下代码所示：

```
rgb = (red << 16 | green << 8 | blue);
```

最后，计算出颜色值后，使用 result BitmapData 对象的 setPixel() 方法将该颜色值实际赋给当前像素，如下所示：

```
result.setPixel(i, j, rgb);
```

位图图像的异步解码

Flash Player 11 和更高版本， **Adobe AIR 2.6** 和更高版本

使用位图图像时，可以异步解码和加载位图图像，以改善应用程序的感知性能。在许多情况下，异步解码位图图像所需的时间可能和同步解码图像相同。但是，在关联的 Loader 对象发送 COMPLETE 事件之前，位图图像会在单独的线程中进行解码。因此，您可以在加载较大的图像后，对其进行异步解码。

借助 flash.system 包中的 ImageDecodingPolicy 类，您可以指定位图加载方案。默认加载方案为异步。

位图解码策略	位图加载方案	说明
ImageDecodingPolicy.ON_DEMAND	同步	在访问图像数据时，对加载的图像进行解码。 使用此策略可以解码较小的图像。如果您的应用程序不依赖复杂的效果或过渡，则也可以使用此策略。
ImageDecodingPolicy.ON_LOAD	异步	在加载时，在调度 COMPLETE 事件之前对加载的图像进行解码。 此策略是处理较大 （大于 10 MP）图像的理想之选。当开发具有页面过渡效果的基于 AIR 的移动应用程序时，使用此位图加载策略可以改善应用程序的感知性能。

注：如果加载的文件是位图图像，并且使用的解码策略为 ON_LOAD，则该图像将在 COMPLETE 事件调度之前进行异步解码。

以下代码显示了 ImageDecodingPolicy 类的用法：

```
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD
var loader:Loader = new Loader();
loader.load(new URLRequest("http://www.adobe.com/myimage.png"), loaderContext);
```

您仍然可以通过 Loader.load() 和 Loader.loadBytes() 方法使用 ON_DEMAND 解码。但是，所有其他使用 LoaderContext 对象作为参数的方法，将忽略传递的任何 ImageDecodingPolicy 值。

下面的示例显示了同步和异步解码位图图像的差异：

```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.system.ImageDecodingPolicy;
    import flash.system.LoaderContext;

    public class AsyncTest extends Sprite
    {
        private var loaderContext:LoaderContext;
        private var loader:Loader;
        private var urlRequest:URLRequest;
        public function AsyncTest()
        {
            //Load the image synchronously
            loaderContext = new LoaderContext();
            //Default behavior.
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_DEMAND;
            loader = new Loader();
            loadImageSync();

            //Load the image asynchronously
            loaderContext = new LoaderContext();
            loaderContext.imageDecodingPolicy = ImageDecodingPolicy.ON_LOAD;
            loader = new Loader();
            loadImageASync();
        }

        private function loadImageASync():void{
            trace("Loading image asynchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
```

```
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onAsyncLoadComplete);
        }

        private function onAsyncLoadComplete(event:Event):void{
            trace("Async. Image Load Complete");
        }

        private function loadImageSync():void{
            trace("Loading image synchronously...");
            urlRequest = new URLRequest("http://www.adobe.com/myimage.png");
            urlRequest.useCache = false;
            loader.load(urlRequest, loaderContext);
            loader.contentLoaderInfo.addEventListener
                (Event.COMPLETE, onSyncLoadComplete);
        }

        private function onSyncLoadComplete(event:Event):void{
            trace("Sync. Image Load Complete");
        }
    }
}
```

有关不同解码策略的效果说明，请参阅 Thibaud Imbert：Adobe Flash 运行时中的异步位图解码

第 14 章：过滤显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以往，对位图图像应用滤镜效果是专业图像编辑软件 （如 Adobe Photoshop® 和 Adobe Fireworks®）的范畴。ActionScript 3.0 包括 flash.filters 包，其中包含一系列位图效果滤镜类。使用这些效果，开发人员可以编程方式对位图应用滤镜并显示对象，以达到图形处理应用程序中所具有的许多相同效果。

过滤显示对象的基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

为应用程序添加优美效果的一种方式是添加简单的图形效果。您可以在图片后面添加投影以产生三维视觉效果，或在按钮周围添加发光效果以表示该按钮当前处于活动状态。 ActionScript 3.0 包括十种可应用于任何显示对象或 BitmapData 实例的滤镜。内置滤镜的范围从基本滤镜 （如投影和发光滤镜）到复杂滤镜 （如置换图滤镜和卷积滤镜）。

注：除了内置滤镜外，您还可以使用 Pixel Bender 来设计自定义滤镜和效果。请参阅第 251 页的 " 使用 Pixel Bender 着色器 "。

重要概念和术语

以下参考列表包含创建滤镜时可能会遇到的重要术语：

斜面 通过在两个面变使像素变亮并在相对两个面使像素变暗创建的一个边缘。此效果可产生三维边框的外观。该效果常用于凸起或凹进按钮和类似图形。

卷积 通过使用各种比率将每个像素的值与其周围的某些像素或全部像素的值合并，使图像中的像素发生扭曲。

置换 将图像中的像素偏移或移动到新位置。

Matrix 用于通过将网格中的数字应用到多个值然后合并这些结果来执行某些数学计算的数字网格。

更多帮助主题

flash.filters 包

flash.display.DisplayObject.filters

flash.display.BitmapData.applyFilter()

创建和应用滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用滤镜可以对位图和显示对象应用从投影到斜角和模糊等各种效果。由于将每个滤镜定义为一个类，因此应用滤镜涉及创建滤镜对象的实例，这与构造任何其他对象并没有区别。创建了滤镜对象的实例后，通过使用该对象的 filters 属性可以很容易地将此实例应用于显示对象；如果是 BitmapData 对象，可以使用 applyFilter() 方法。

创建滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要创建滤镜对象，只需调用所选的滤镜类的构造函数方法即可。例如，若要创建 DropShadowFilter 对象，请使用以下代码：

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

虽然此处没有显示参数，但 DropShadowFilter() 构造函数 （与所有滤镜类的构造函数一样）接受多个可用于自定义滤镜效果外观的可选参数。

应用滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

构造滤镜对象后，可以将其应用于显示对象或 BitmapData 对象；应用滤镜的方式取决于为之应用该滤镜的对象。

对显示对象应用滤镜

对显示对象应用滤镜效果时，可以通过 filters 属性应用这些效果。显示对象的 filters 属性是一个 Array 实例，其中的元素是应用于该显示对象的滤镜对象。若要对显示对象应用单个滤镜，请创建该滤镜实例，将其添加到 Array 实例，再将该 Array 对象分配给显示对象的 filters 属性：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.filters.DropShadowFilter;

// Create a bitmapData object and render it to screen
var myBitmapData:BitmapData = new BitmapData(100,100,false,0xFFFF3300);
var myDisplayObject:Bitmap = new Bitmap(myBitmapData);
addChild(myDisplayObject);

// Create a DropShadowFilter instance.
var dropShadow:DropShadowFilter = new DropShadowFilter();

// Create the filters array, adding the filter to the array by passing it as
// a parameter to the Array() constructor.
var filtersArray:Array = new Array(dropShadow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

如果要为该对象分配多个滤镜，只需在将 Array 实例分配给 filters 属性之前将所有滤镜添加到该实例即可。可以通过将多个对象作为参数传递给 Array 的构造函数，将多个对象添加到 Array。例如，以下代码对上面创建的显示对象应用斜角滤镜和发光滤镜：

```
import flash.filters.BevelFilter;
import flash.filters.GlowFilter;

// Create the filters and add them to an array.
var bevel:BevelFilter = new BevelFilter();
var glow:GlowFilter = new GlowFilter();
var filtersArray:Array = new Array(bevel, glow);

// Assign the filters array to the display object to apply the filter.
myDisplayObject.filters = filtersArray;
```

在创建包含滤镜的数组时，您可以使用 new Array() 构造函数创建该数组 （如前面示例所示），也可以使用 Array 文本语法将滤镜括在中括号 ([]) 中。例如，下面这行代码：

```
var filters:Array = new Array(dropShadow, blur);
```

与下面这行代码效果相同：

```
var filters:Array = [dropShadow, blur];
```

如果对显示对象应用多个滤镜，则会按顺序以累积方式应用这些滤镜。例如，如果滤镜数组有两个元素：先添加的斜角滤镜和后添加的投影滤镜，则投影滤镜既会应用于斜角滤镜，也会应用于显示对象。这是由于投影滤镜在滤镜数组中位于第二个位置。如果想要以非累积方式应用滤镜，可对显示对象的新副本应用每个滤镜。

如果只为显示对象分配一个或几个滤镜，则可以先创建该滤镜实例，然后在单个语句中将该实例分配给该对象。例如，下面的一行代码将模糊滤镜应用于名为 myDisplayObject 的显示对象：

```
myDisplayObject.filters = [new BlurFilter()];
```

上面的代码使用 Array 文本语法 （中括号）创建一个 Array 实例，并创建一个 BlurFilter 实例作为 Array 中的一个元素，然后将该 Array 分配给名为 myDisplayObject 的显示对象的 filters 属性。

删除显示对象中的滤镜

删除显示对象中的所有滤镜非常简单，只需为 filters 属性分配一个 null 值即可：

```
myDisplayObject.filters = null;
```

如果您已对一个对象应用了多个滤镜，并且只想删除其中一个滤镜，则必须完成多个步骤才能更改 filters 属性数组。有关详细信息，请参阅第 226 页的 “ 使用滤镜的潜在问题 ”。

对 **BitmapData** 对象应用滤镜

对 BitmapData 对象应用滤镜需要使用 BitmapData 对象的 applyFilter() 方法：

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

applyFilter() 方法会对源 BitmapData 对象应用滤镜，从而生成一个新的、应用滤镜的图像。此方法不会修改原始的源图像；而是将对源图像应用滤镜的结果存储在调用 applyFilter() 方法的 BitmapData 实例中。

滤镜的工作原理

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

显示对象过滤是通过将原始对象的副本缓存为透明位图来工作的。

将滤镜应用于显示对象后，只要此对象具有有效的滤镜列表，运行时会将该对象缓存为位图。然后，将此位图用作所有后续应用的滤镜效果的原始图像。

每个显示对象通常包含两个位图：一个包含原始未过滤的源显示对象，另一个用于过滤后的最终图像。呈示时使用最终图像。只要显示对象不发生更改，最终图像就不需要更新。

使用滤镜的潜在问题

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用滤镜时要记住几个导致混淆或问题的潜在根源。

滤镜和位图缓存

若要对显示对象应用滤镜，必须启用该对象的位图缓存。对其 cacheAsBitmap 属性设置为 false 的显示对象应用滤镜时，该对象的 cacheAsBitmap 属性会自动设置为 true。如果稍后删除了显示对象的所有滤镜，则 cacheAsBitmap 属性会重置为最后设置的值。

在运行时更改滤镜

如果显示对象已应用了一个或多个滤镜，则不能通过在 filters 属性数组中添加或删除滤镜来更改滤镜组。若要添加或更改要应用的滤镜组，必须对单独的数组进行更改，然后将该数组分配给显示对象的 filters 属性以对该对象应用滤镜。执行此操作最简单方法是将 filters 属性数组读入 Array 变量，然后对此临时数组进行修改。然后，将此数组重新分配回显示对象的 filters 属性。在较复杂的情况下，可能需要保留一个单独的主滤镜数组。可以对该主滤镜数组进行任何更改，并在每次更改后将该主数组重新分配给显示对象的 filters 属性。

添加其他滤镜

下面的代码演示向已应用一个或多个滤镜的显示对象添加其他滤镜的过程。首先，对名为 myDisplayObject 的显示对象应用发光滤镜，然后，在单击该显示对象时，调用 addFilters() 函数。在此函数中，另有两个滤镜应用于 myDisplayObject：

```
import flash.events.MouseEvent;
import flash.filters.*;

myDisplayObject.filters = [new GlowFilter()];

function addFilters(event:MouseEvent):void
{
    // Make a copy of the filters array.
    var filtersCopy:Array = myDisplayObject.filters;

    // Make desired changes to the filters (in this case, adding filters).
    filtersCopy.push(new BlurFilter());
    filtersCopy.push(new DropShadowFilter());

    // Apply the changes by reassigning the array to the filters property.
    myDisplayObject.filters = filtersCopy;
}

myDisplayObject.addEventListener(MouseEvent.CLICK, addFilters);
```

从一组滤镜中删除一个滤镜

如果显示对象已经应用了多个滤镜，您希望删除其中的一个滤镜并希望其他滤镜继续应用于该对象，请将这些滤镜复制到一个临时数组中，从该数组中删除不需要的滤镜，然后将该临时数组重新分配给该显示对象的 filters 属性。第 26 页的 “ 检索值和删除数组元素 ”一节介绍了从任意数组删除一个或多个元素的几种方法。

最简单的情况是删除对象上最顶层的滤镜 （最后一个应用于对象的滤镜）。可以使用 Array 类的 pop() 方法从数组中删除滤镜：

```
// Example of removing the top-most filter from a display object
// named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the last element from the Array (the top-most filter).
tempFilters.pop();

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

同样，若要删除最底层的滤镜 （第一个应用于对象的滤镜），也可以使用相同的代码，不过需要用 Array 类的 shift() 方法替换 pop() 方法。

若要从滤镜数组的中间删除滤镜 （假定该数组有两个以上的滤镜），则可以使用 splice() 方法。您必须知道要删除的滤镜的索引 （在数组中的位置）。例如，下面的代码从显示对象中删除第二个滤镜 （索引 1 处的滤镜）：

```
// Example of removing a filter from the middle of a stack of filters
// applied to a display object named "filteredObject".

var tempFilters:Array = filteredObject.filters;

// Remove the second filter from the array. It's the item at index 1
// because Array indexes start from 0.
// The first "1" indicates the index of the filter to remove; the
// second "1" indicates how many elements to remove.
tempFilters.splice(1, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

确定滤镜索引

您需要知道要从数组中删除哪个滤镜，才能确定该滤镜的索引。您必须知道 （根据应用程序的设计方式）或计算出要删除的滤镜的索引。

最佳方法是通过设计应用程序，确保要删除的滤镜始终位于滤镜组中的同一位置。例如，如果有一个显示对象先后应用了卷积滤镜和投影滤镜，您希望删除投影滤镜而保留卷积滤镜，由于滤镜位于已知位置 （最顶层的滤镜），因此您可以提前知道应使用哪种 Array 方法 （在本例中，使用 Array.pop() 删除投影滤镜）。

如果您要删除的滤镜始终属于某个特定类型，但不一定总在滤镜组中的同一位置，那么您可以检查该数组中每个滤镜的数据类型，从而确定将要删除哪个滤镜。例如，下面的代码确定滤镜组中的哪个滤镜是发光滤镜，然后从该组中删除该滤镜。

```
// Example of removing a glow filter from a set of filters, where the
//filter you want to remove is the only GlowFilter instance applied
// to the filtered object.

var tempFilters:Array = filteredObject.filters;

// Loop through the filters to find the index of the GlowFilter instance.
var glowIndex:int;
var numFilters:int = tempFilters.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (tempFilters[i] is GlowFilter)
    {
        glowIndex = i;
        break;
    }
}

// Remove the glow filter from the array.
tempFilters.splice(glowIndex, 1);

// Apply the new set of filters to the display object.
filteredObject.filters = tempFilters;
```

在更为复杂的情况下 （比如要删除的滤镜是在运行时选定的），最好为滤镜数组保留一个单独的永久副本，用作主滤镜列表。不论何时对滤镜组进行更改，均需更改主列表，然后按照显示对象的 filters 属性应用该滤镜数组。

例如，在下面的代码清单中，多个卷积滤镜应用于一个显示对象，以制造不同的视觉效果，随后在应用程序中删除其中一个滤镜而保留其余滤镜。在本例中，代码保留滤镜数组的主副本以及对于要删除的滤镜的引用。查找并删除特定滤镜的方法与前面的方法类似，但不是创建滤镜数组的临时副本，而是操作主副本，然后将其应用于显示对象。

```
// Example of removing a filter from a set of
// filters, where there may be more than one
// of that type of filter applied to the filtered
// object, and you only want to remove one.

// A master list of filters is stored in a separate,
// persistent Array variable.
var masterFilterList:Array;

// At some point, you store a reference to the filter you
// want to remove.
var filterToRemove:ConvolutionFilter;

// ... assume the filters have been added to masterFilterList,
// which is then assigned as the filteredObject.filters:
filteredObject.filters = masterFilterList;

// ... later, when it's time to remove the filter, this code gets called:

// Loop through the filters to find the index of masterFilterList.
var removeIndex:int = -1;
var numFilters:int = masterFilterList.length;
for (var i:int = 0; i < numFilters; i++)
{
    if (masterFilterList[i] == filterToRemove)
    {
        removeIndex = i;
        break;
    }
}

if (removeIndex >= 0)
{
    // Remove the filter from the array.
    masterFilterList.splice(removeIndex, 1);

    // Apply the new set of filters to the display object.
    filteredObject.filters = masterFilterList;
}
```

在此方法（将存储的滤镜引用与滤镜数组中的项目进行比较以确定要删除哪个滤镜）中，必须 为滤镜数组保留一个单独的副本 — 如果将存储的滤镜引用与从显示对象的 filters 属性复制的临时数组中的元素进行比较，则代码将不起作用。这是因为在内部将数组分配给 filters 属性时，运行时会为数组中的每个滤镜对象创建副本。这些副本 （而非原始对象）将应用于显示对象，而在将 filters 属性读入临时数组时，临时数组包含对复制的滤镜对象的引用，而不是对原始滤镜对象的引用。因此，如果在上一示例中尝试通过将 filterToRemove 与临时滤镜数组中的滤镜进行比较来确定其索引，则不会找到匹配的结果。

滤镜和对象变形

显示对象的边框矩形之外的任何过滤区域 （例如，投影）都不能视为可进行点击检测 （确定实例是否与其他实例重叠或交叉）的表面。由于 DisplayObject 类的点击检测方法是基于矢量的，因此无法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；如果过滤的显示对象本身进行了缩放 （如果 scaleX 和 scaleY 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。

滤镜和位图对象

对 BitmapData 对象应用滤镜时， cacheAsBitmap 属性会自动设置为 true。通过这种方式，滤镜实际上是应用于对象的副本而不是原始对象。

之后，会将此副本放在主显示 （原始对象）上，尽量接近最近的像素。如果原始位图的边框发生更改，则会从头重新创建过滤的副本位图，而不是被伸展或扭曲。

如果清除了显示对象的所有滤镜， cacheAsBitmap 属性会重置为应用滤镜之前的值。

可用的显示滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 提供十个可用于显示对象和 BitmapData 对象的滤镜类：

- 斜角滤镜 （BevelFilter 类）
- 模糊滤镜 （BlurFilter 类）
- 投影滤镜 （DropShadowFilter 类）
- 发光滤镜 （GlowFilter 类）
- 渐变斜角滤镜 （GradientBevelFilter 类）
- 渐变发光滤镜 （GradientGlowFilter 类）
- 颜色矩阵滤镜 （ColorMatrixFilter 类）
- 卷积滤镜 （ConvolutionFilter 类）
- 置换图滤镜 （DisplacementMapFilter 类）
- 着色器滤镜 （ShaderFilter 类）

前六个滤镜是简单滤镜，可用于创建一种特定效果，并可以对效果进行某种程度的自定义。可以使用 ActionScript 应用这六个滤镜，也可以在 Flash Professional 中使用 “ 滤镜 ” 面板将其应用于对象。因此，即使您要使用 ActionScript 应用滤镜，如果有 Flash Professional，也可以使用可视界面快速尝试不同的滤镜和设置，以弄清楚如何创建所需的效果。

最后四个滤镜仅在 ActionScript 中可用。这些滤镜 （颜色矩阵滤镜、卷积滤镜、置换图滤镜和着色器滤镜）能够制造的效果类型十分灵活。这些滤镜不是针对一种效果进行优化，而是具有强大的功能和灵活性。例如，如果为卷积滤镜的矩阵选择不同的值，则它可用于创建模糊、浮雕、锐化、查找颜色边缘、变形等效果。

不管是简单滤镜还是复杂滤镜，每个滤镜都可以使用其属性进行自定义。通常，您有两种方法用于设置滤镜属性。所有滤镜都允许通过向滤镜对象的构造函数传递参数值来设置属性。或者，不管您是否通过传递参数来设置滤镜属性，都可以在以后通过设置滤镜对象的属性值来调整滤镜。多数示例代码清单都直接设置属性，以便更易于按照示例进行操作。不过，您通常可以通过在滤镜对象的构造函数中以参数的形式传递值来获得同样的结果。有关每个滤镜及其属性和构造函数参数的更多详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 flash.filters 包列表。

斜角滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

BevelFilter 类可为过滤的对象添加 3D 斜角边缘。此滤镜可使对象的硬角或边缘具有硬角或边缘被凿削或呈斜面的效果。

BevelFilter 类属性允许您自定义斜角的外观。您可以设置加亮和阴影颜色、斜角边缘模糊、斜角角度和斜角边缘的位置，甚至可以创建挖空效果。

以下示例加载外部图像并对它应用斜角滤镜。

```
import flash.display.*;
import flash.filters.BevelFilter;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.net.URLRequest;

// Load an image onto the Stage.
var imageLoader:Loader = new Loader();
var url:String = "http://www.helpexamples.com/flash/images/image3.jpg";
var urlReq:URLRequest = new URLRequest(url);
imageLoader.load(urlReq);
addChild(imageLoader);

// Create the bevel filter and set filter properties.
var bevel:BevelFilter = new BevelFilter();

bevel.distance = 5;
bevel.angle = 45;
bevel.highlightColor = 0xFFFF00;
bevel.highlightAlpha = 0.8;
bevel.shadowColor = 0x666666;
bevel.shadowAlpha = 0.8;
bevel.blurX = 5;
bevel.blurY = 5;
bevel.strength = 5;
bevel.quality = BitmapFilterQuality.HIGH;
bevel.type = BitmapFilterType.INNER;
bevel.knockout = false;

// Apply filter to the image.
imageLoader.filters = [bevel];
```

模糊滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

BlurFilter 类可使显示对象及其内容具有涂抹或模糊的效果。模糊效果可以用于产生对象不在焦点之内的视觉效果，也可以用于模拟快速运动，比如运动模糊。通过将模糊滤镜的 quality 属性设置为非常低的值，可以模拟轻轻离开焦点的镜头效果。将 quality 属性设置为高会产生类似高斯模糊的平滑模糊效果。

以下示例使用 Graphics 类的 drawCircle() 方法创建一个圆形对象并对它应用模糊滤镜：

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.BlurFilter;

// Draw a circle.
var redDotCutout:Sprite = new Sprite();
redDotCutout.graphics.lineStyle();
redDotCutout.graphics.beginFill(0xFF0000);
redDotCutout.graphics.drawCircle(145, 90, 25);
redDotCutout.graphics.endFill();

// Add the circle to the display list.
addChild(redDotCutout);

// Apply the blur filter to the rectangle.
var blur:BlurFilter = new BlurFilter();
blur.blurX = 10;
blur.blurY = 10;
blur.quality = BitmapFilterQuality.MEDIUM;
redDotCutout.filters = [blur];
```

投影滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

投影给人一种目标对象上方有独立光源的印象。可以修改此光源的位置和强度，以产生各种不同的投影效果。

DropShadowFilter 类使用的算法与模糊滤镜的算法类似。主要区别是投影滤镜有更多的属性，您可以修改这些属性来模拟不同的光源属性 （如 Alpha、颜色、偏移和亮度）。

投影滤镜还允许您对投影的样式应用自定义变形选项，包括内侧或外侧阴影和挖空 （也称为剪切块）模式。

以下代码创建方框 sprite 并对它应用投影滤镜：

```
import flash.display.Sprite;
import flash.filters.DropShadowFilter;

// Draw a box.
var boxShadow:Sprite = new Sprite();
boxShadow.graphics.lineStyle(1);
boxShadow.graphics.beginFill(0xFF3300);
boxShadow.graphics.drawRect(0, 0, 100, 100);
boxShadow.graphics.endFill();
addChild(boxShadow);

// Apply the drop shadow filter to the box.
var shadow:DropShadowFilter = new DropShadowFilter();
shadow.distance = 10;
shadow.angle = 25;

// You can also set other properties, such as the shadow color,
// alpha, amount of blur, strength, quality, and options for
// inner shadows and knockout effects.

boxShadow.filters = [shadow];
```

发光滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

GlowFilter 类对显示对象应用加亮效果，使显示对象看起来像是被下方的灯光照亮，可创造出一种柔和发光效果。

与投影滤镜类似，发光滤镜包括的属性可修改光源的距离、角度和颜色，以产生各种不同效果。 GlowFilter 还有多个选项用于修改发光样式，包括内侧或外侧发光和挖空模式。

以下代码使用 Sprite 类创建一个交叉对象并对它应用发光滤镜：

```
import flash.display.Sprite;
import flash.filters.BitmapFilterQuality;
import flash.filters.GlowFilter;

// Create a cross graphic.
var crossGraphic:Sprite = new Sprite();
crossGraphic.graphics.lineStyle();
crossGraphic.graphics.beginFill(0xCCCC00);
crossGraphic.graphics.drawRect(60, 90, 100, 20);
crossGraphic.graphics.drawRect(100, 50, 20, 100);
crossGraphic.graphics.endFill();
addChild(crossGraphic);

// Apply the glow filter to the cross shape.
var glow:GlowFilter = new GlowFilter();
glow.color = 0x009922;
glow.alpha = 1;
glow.blurX = 25;
glow.blurY = 25;
glow.quality = BitmapFilterQuality.MEDIUM;

crossGraphic.filters = [glow];
```

渐变斜角滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

GradientBevelFilter 类可对显示对象或 BitmapData 对象应用增强的斜角效果。在斜角上使用渐变颜色可以大大改善斜角的空间深度，使边缘产生一种更逼真的三维外观效果。

以下代码使用 Shape 类的 drawRect() 方法创建一个矩形对象，并对它应用渐变斜角滤镜。

```
import flash.display.Shape;
import flash.filters.BitmapFilterQuality;
import flash.filters.GradientBevelFilter;

// Draw a rectangle.
var box:Shape = new Shape();
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Apply a gradient bevel to the rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter();

gradientBevel.distance = 8;
gradientBevel.angle = 225; // opposite of 45 degrees
gradientBevel.colors = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
gradientBevel.alphas = [1, 0, 1];
gradientBevel.ratios = [0, 128, 255];
gradientBevel.blurX = 8;
gradientBevel.blurY = 8;
gradientBevel.quality = BitmapFilterQuality.HIGH;

// Other properties let you set the filter strength and set options
// for inner bevel and knockout effects.

box.filters = [gradientBevel];

// Add the graphic to the display list.
addChild(box);
```

渐变发光滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

GradientGlowFilter 类可对显示对象或 BitmapData 对象应用增强的发光效果。该效果可使您更好地控制发光颜色，因而可产生一种更逼真的发光效果。另外，渐变发光滤镜还允许您对对象的内侧、外侧或上侧边缘应用渐变发光。

以下示例在舞台上绘制一个圆形，并对它应用渐变发光滤镜。当您进一步向右和向下移动鼠标时，会分别增加水平和垂直方向的模糊量。此外，只要您在舞台上单击，就会增加模糊的强度。

```
import flash.events.MouseEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.GradientGlowFilter;

// Create a new Shape instance.
var shape:Shape = new Shape();

// Draw the shape.
shape.graphics.beginFill(0xFF0000, 100);
shape.graphics.moveTo(0, 0);
shape.graphics.lineTo(100, 0);
shape.graphics.lineTo(100, 100);
shape.graphics.lineTo(0, 100);
shape.graphics.lineTo(0, 0);
shape.graphics.endFill();

// Position the shape on the Stage.
addChild(shape);
shape.x = 100;
```

```
shape.y = 100;

// Define a gradient glow.
var gradientGlow:GradientGlowFilter = new GradientGlowFilter();
gradientGlow.distance = 0;
gradientGlow.angle = 45;
gradientGlow.colors = [0x000000, 0xFF0000];
gradientGlow.alphas = [0, 1];
gradientGlow.ratios = [0, 255];
gradientGlow.blurX = 10;
gradientGlow.blurY = 10;
gradientGlow.strength = 2;
gradientGlow.quality = BitmapFilterQuality.HIGH;
gradientGlow.type = BitmapFilterType.OUTER;

// Define functions to listen for two events.
function onClick(event:MouseEvent):void
{
    gradientGlow.strength++;
    shape.filters = [gradientGlow];
}

function onMouseMove(event:MouseEvent):void
{
    gradientGlow.blurX = (stage.mouseX / stage.stageWidth) * 255;
    gradientGlow.blurY = (stage.mouseY / stage.stageHeight) * 255;
    shape.filters = [gradientGlow];
}
stage.addEventListener(MouseEvent.CLICK, onClick);
stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
```

示例：合并基本滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下代码示例使用与 Timer 合并的多个基本滤镜创建重复动作，以形成一种动画交通信号灯模拟效果。

```
import flash.display.Shape;
import flash.events.TimerEvent;
import flash.filters.BitmapFilterQuality;
import flash.filters.BitmapFilterType;
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;
import flash.filters.GradientBevelFilter;
import flash.utils.Timer;

var count:Number = 1;
var distance:Number = 8;
var angleInDegrees:Number = 225; // opposite of 45 degrees
var colors:Array = [0xFFFFCC, 0xFEFE78, 0x8F8E01];
var alphas:Array = [1, 0, 1];
var ratios:Array = [0, 128, 255];
var blurX:Number = 8;
var blurY:Number = 8;
var strength:Number = 1;
var quality:Number = BitmapFilterQuality.HIGH;
var type:String = BitmapFilterType.INNER;
var knockout:Boolean = false;

// Draw the rectangle background for the traffic light.
var box:Shape = new Shape();
```

```
box.graphics.lineStyle();
box.graphics.beginFill(0xFEFE78);
box.graphics.drawRect(100, 50, 90, 200);
box.graphics.endFill();

// Draw the 3 circles for the three lights.
var stopLight:Shape = new Shape();
stopLight.graphics.lineStyle();
stopLight.graphics.beginFill(0xFF0000);
stopLight.graphics.drawCircle(145,90,25);
stopLight.graphics.endFill();

var cautionLight:Shape = new Shape();
cautionLight.graphics.lineStyle();
cautionLight.graphics.beginFill(0xFF9900);
cautionLight.graphics.drawCircle(145,150,25);
cautionLight.graphics.endFill();

var goLight:Shape = new Shape();
goLight.graphics.lineStyle();
goLight.graphics.beginFill(0x00CC00);
goLight.graphics.drawCircle(145,210,25);
goLight.graphics.endFill();

// Add the graphics to the display list.
addChild(box);
addChild(stopLight);
addChild(cautionLight);
addChild(goLight);

// Apply a gradient bevel to the traffic light rectangle.
var gradientBevel:GradientBevelFilter = new GradientBevelFilter(distance, angleInDegrees, colors, alphas,
ratios, blurX, blurY, strength, quality, type, knockout);
box.filters = [gradientBevel];

// Create the inner shadow (for lights when off) and glow
// (for lights when on).
var innerShadow:DropShadowFilter = new DropShadowFilter(5, 45, 0, 0.5, 3, 3, 1, 1, true, false);
var redGlow:GlowFilter = new GlowFilter(0xFF0000, 1, 30, 30, 1, 1, false, false);
var yellowGlow:GlowFilter = new GlowFilter(0xFF9900, 1, 30, 30, 1, 1, false, false);
var greenGlow:GlowFilter = new GlowFilter(0x00CC00, 1, 30, 30, 1, 1, false, false);

// Set the starting state of the lights (green on, red/yellow off).
stopLight.filters = [innerShadow];
cautionLight.filters = [innerShadow];
goLight.filters = [greenGlow];

// Swap the filters based on the count value.
function trafficControl(event:TimerEvent):void
{
    if (count == 4)
    {
        count = 1;
    }

    switch (count)
    {
        case 1:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [yellowGlow];
            goLight.filters = [innerShadow];
```

```
            break;
        case 2:
            stopLight.filters = [redGlow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [innerShadow];
            break;
        case 3:
            stopLight.filters = [innerShadow];
            cautionLight.filters = [innerShadow];
            goLight.filters = [greenGlow];
            break;
    }

    count++;
}

// Create a timer to swap the filters at a 3 second interval.
var timer:Timer = new Timer(3000, 9);
timer.addEventListener(TimerEvent.TIMER, trafficControl);
timer.start();
```

颜色矩阵滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ColorMatrixFilter 类用于操作已应用滤镜的对象的颜色和 Alpha 值。它允许您进行饱和度更改、色相旋转 （将调色板从一个颜色范围移动到另一个颜色范围）、将亮度更改为 Alpha，以及生成其他颜色操作效果，方法是使用一个颜色通道中的值，并将这些值潜移默化地应用于其他通道。

从概念上来说，滤镜将逐一处理源图像中的像素，并将每个像素分为红、绿、蓝和 Alpha 组件。然后，用每个值乘以颜色矩阵中提供的值，将结果加在一起以确定该像素将显示在屏幕上的最终颜色值。滤镜的 matrix 属性是一个由 20 个数字组成的数组，用于计算最终颜色。有关用于计算颜色值的特定算法的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中描述 ColorMatrixFilter 类的 matrix 属性的条目。

卷积滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ConvolutionFilter 类可用于对 BitmapData 对象或显示对象应用广泛的图像变形，如模糊、边缘检测、锐化、浮雕和斜角。

从概念上来说，卷积滤镜会逐一处理源图像中的每个像素，并使用像素和它周围的像素的值来确定该像素的最终颜色。指定为数值数组的矩阵可以指示每个特定邻近像素的值对最终结果值具有何种程度的影响。

最常用的矩阵类型是 3 x 3 矩阵。此矩阵包括九个值：

```
N   N   N
N   P   N
N   N   N
```

对特定像素应用卷积滤镜时，会检查该像素本身的颜色值 （本示例中的 “P”）以及周围像素的值 （本示例中的 “N”）。而通过设置矩阵中的值，可以指定特定像素在影响生成的图像方面所具有的优先级。

例如，使用卷积滤镜时应用的以下矩阵，会保持图像原样：

```
0   0   0
0   1   0
0   0   0
```

图像保持不变的原因是，在决定最终像素颜色时，原始像素的值相对强度为 1，而周围像素的值相对强度为 0 （意味着它们的颜色不影响最终图像）。

同样，下面的这个矩阵会使图像的像素向左移动一个像素：

```
0   0   0
0   0   1
0   0   0
```

请注意，在本例中，像素本身不影响最终图像上显示在该位置的像素的最终值，而只使用右侧的像素值来确定像素的结果值。

在 ActionScript 中，您可以通过组合一个包含值的 Array 实例和两个指定矩阵中行数和列数的属性来创建矩阵。以下示例加载一个图像，完成加载该图像后，使用前面列表中的矩阵对该图像应用卷积滤镜：

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

function applyFilter(event:MouseEvent):void
{
    // Create the convolution matrix.
    var matrix:Array = [0, 0, 0,
                                      0, 0, 1,
                                      0, 0, 0];

    var convolution:ConvolutionFilter = new ConvolutionFilter();
    convolution.matrixX = 3;
    convolution.matrixY = 3;
    convolution.matrix = matrix;
    convolution.divisor = 1;

    loader.filters = [convolution];
}

loader.addEventListener(MouseEvent.CLICK, applyFilter);
```

此代码中没有明确显示使用除矩阵中 1 或 0 以外的值将会产生的效果。例如，同一矩阵中如果右侧位置的数字是 8 而不是 1，则它会执行同样的动作 （向左移动像素）。此外，它还影响图像的颜色，使颜色加亮了 8 倍。这是因为最终像素颜色值的计算方法是：用原始像素颜色乘以矩阵值，将这些值加在一起，再除以滤镜的 divisor 属性的值。请注意，在示例代码中， divisor 属性设置为 1。通常，如果想让颜色的明亮度与原始图像保持基本相同，应让 divisor 等于矩阵值之和。因此，如果矩阵的值相加为 8，并且除数为 1，结果图像将比原始图像明亮约 8 倍。

虽然此矩阵的效果不是很明显，但可以使用其他矩阵值以产生各种不同效果。以下是几组标准矩阵值集合，用于使用 3 x 3 矩阵产生不同效果：

- 基本模糊 （除数 5）：

```
 0 1 0
 1 1 1
 0 1 0
```

- 锐化 （除数 1）：

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- 边缘检测 （除数 1）：

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- 浮雕效果 （除数 1）：

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

  请注意，对于上述大部分效果， divisor 均为 1。这是因为负矩阵值加正矩阵值产生 1 (或边缘检测中的 0，但 divisor 属性的值不能为 0）。

置换图滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

DisplacementMapFilter 类使用 BitmapData 对象 （称为置换图图像）的像素值在新对象上执行置换效果。通常，置换图图像与将要应用滤镜的实际显示对象或 BitmapData 实例不同。置换效果包括置换经过过滤的图像中的像素，即让这些像素离开各自原始位置一定距离。此滤镜可用于产生移位、扭曲或斑点效果。

应用于给定像素的置换位置和置换量由置换图图像的颜色值确定。使用滤镜时，除了指定置换图图像外，还要指定以下值，以便控制置换图图像中计算置换的方式：

- 映射点：过滤图像上的位置，在该点将应用置换滤镜的左上角。如果只想对图像的一部分应用滤镜，可以使用此值。
- X 组件：影响像素的 x 位置的置换图图像的颜色通道。
- Y 组件：影响像素的 y 位置的置换图图像的颜色通道。
- X 缩放比例：指定 x 轴置换强度的乘数值。
- Y 缩放比例：指定 y 轴置换强度的乘数值。
- 滤镜模式：确定在移开像素后形成的空白区域中要执行什么操作。在 DisplacementMapFilterMode 类中定义为常量的选项可以显示原始像素 （滤镜模式 IGNORE）、从图像的另一侧环绕像素 （滤镜模式 WRAP，这是默认设置）、使用最近的移位像素 （滤镜模式 CLAMP）或用颜色填充空间 （滤镜模式 COLOR）。

若要了解置换图滤镜的工作原理，请查看下面的基本示例。在以下代码中，将加载一个图像，完成加载后使图像在舞台上居中并对它应用置换图滤镜，使整个图像中的像素向左水平移位。

```
import flash.display.BitmapData;
import flash.display.Loader;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.geom.Point;
import flash.net.URLRequest;

// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image3.jpg");
loader.load(url);
this.addChild(loader);

var mapImage:BitmapData;
var displacementMap:DisplacementMapFilter;

// This function is called when the image finishes loading.
function setupStage(event:Event):void
{
    // Center the loaded image on the Stage.
    loader.x = (stage.stageWidth - loader.width) / 2;
    loader.y = (stage.stageHeight - loader.height) / 2;

    // Create the displacement map image.
    mapImage = new BitmapData(loader.width, loader.height, false, 0xFF0000);

    // Create the displacement filter.
    displacementMap = new DisplacementMapFilter();
    displacementMap.mapBitmap = mapImage;
    displacementMap.mapPoint = new Point(0, 0);
    displacementMap.componentX = BitmapDataChannel.RED;
    displacementMap.scaleX = 250;
    loader.filters = [displacementMap];
}

loader.contentLoaderInfo.addEventListener(Event.COMPLETE, setupStage);
```

用于定义置换的属性有：

- 置换图位图：置换位图是由代码创建的新的 BitmapData 实例。它的尺寸与加载的图像的尺寸匹配 （因此会将置换应用于整个图像）。用纯红色像素填充此实例。
- 映射点：将此值设置为点 0, 0，使置换再次应用于整个图像。
- X 组件：此值设置为常量 BitmapDataChannel.RED，表示置换图位图的红色值将决定沿着 x 轴置换像素的程度 （像素的移动程度）。
- X 缩放比例：此值设置为 250。由于全部置换量 （与全红的置换图图像的距离）仅使图像置换很小的量 （大约为一个像素的一半），因此，如果将此值设置为 1，图像只会水平移动 0.5 个像素。将此值设置为 250，图像将移动大约 125 个像素。

这些设置可使过滤图像的像素向左移动 250 个像素。移动的方向 （向左或向右）和移动量取决于置换图图像中的像素的颜色值。从概念上来说，滤镜会逐一处理过滤图像的像素 （至少处理将应用滤镜的区域中的像素，在本例中指所有像素），并对每个像素执行以下操作：

1. Flash Player 在置换图图像中查找相应的像素。例如，当滤镜计算过滤图像左上角像素的置换量时，会在置换图图像左上角中查找像素。
2. Flash Player 确定置换图像素中指定颜色通道的值。在本例中， x 组件颜色通道是红色通道，因此滤镜将查看映射图像中该问题像素所在位置处的红色通道所对应的值。由于置换图图像是纯红色的，所以像素的红色通道为 0xFF （即 255）。该值将用作置换值。

3 比较置换值和“中间”值（127，它是 0 和 255 之间的中间值）。如果置换值低于中间值，则像素正向移位（x 置换向右； y 置换向下）。另一方面，如果置换值高于中间值 （如本示例），则像素负向移位 （x 置换向左； y 置换向上）。为更精确起见，滤镜会从 127 中减去置换值，结果 （正或负）即是应用的相对置换量。

4 最后，Flash Player 通过确定相对置换值所表示的完全置换量的百分比来确定实际置换量。在本例中，全红色意味着 100% 置换。然后用 x 缩放比例值或 y 缩放比例值乘以该百分比，以确定将应用的置换像素数。在本示例中，100% 乘以一个乘数 250 可以确定置换量 （大约为向左移动 125 个像素）。

因为没有为 y 分量和 y 缩放比例指定值，所以使用默认值 （不发生置换），这就是图像在垂直方向不移位的原因。

由于在本示例中使用了默认滤镜模式设置 WRAP，因此在像素向左移位时，会用移到图像左边缘以外的像素填充右侧空白区域。您可以对此设置试用不同的值以查看不同的效果。例如，如果您在设置置换属性的代码部分中 （在 loader.filters = [displacementMap] 行之前）添加以下一行内容，则会使图像在舞台上产生涂抹的效果：

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

下面是一个更为复杂的示例，代码清单中使用置换图滤镜在图像上创建放大镜效果：

```
import flash.display.Bitmap;
import flash.display.BitmapData;
import flash.display.BitmapDataChannel;
import flash.display.GradientType;
import flash.display.Loader;
import flash.display.Shape;
import flash.events.MouseEvent;
import flash.filters.DisplacementMapFilter;
import flash.filters.DisplacementMapFilterMode;
import flash.geom.Matrix;
import flash.geom.Point;
import flash.net.URLRequest;

// Create the gradient circles that will together form the
// displacement map image
var radius:uint = 50;

var type:String = GradientType.LINEAR;
var redColors:Array = [0xFF0000, 0x000000];
var blueColors:Array = [0x0000FF, 0x000000];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var xMatrix:Matrix = new Matrix();
xMatrix.createGradientBox(radius * 2, radius * 2);
var yMatrix:Matrix = new Matrix();
yMatrix.createGradientBox(radius * 2, radius * 2, Math.PI / 2);

var xCircle:Shape = new Shape();
xCircle.graphics.lineStyle(0, 0, 0);
xCircle.graphics.beginGradientFill(type, redColors, alphas, ratios, xMatrix);
xCircle.graphics.drawCircle(radius, radius, radius);

var yCircle:Shape = new Shape();
yCircle.graphics.lineStyle(0, 0, 0);
yCircle.graphics.beginGradientFill(type, blueColors, alphas, ratios, yMatrix);
yCircle.graphics.drawCircle(radius, radius, radius);

// Position the circles at the bottom of the screen, for reference.
this.addChild(xCircle);
xCircle.y = stage.stageHeight - xCircle.height;
this.addChild(yCircle);
yCircle.y = stage.stageHeight - yCircle.height;
yCircle.x = 200;
```

```
// Load an image onto the Stage.
var loader:Loader = new Loader();
var url:URLRequest = new URLRequest("http://www.helpexamples.com/flash/images/image1.jpg");
loader.load(url);
this.addChild(loader);

// Create the map image by combining the two gradient circles.
var map:BitmapData = new BitmapData(xCircle.width, xCircle.height, false, 0x7F7F7F);
map.draw(xCircle);
var yMap:BitmapData = new BitmapData(yCircle.width, yCircle.height, false, 0x7F7F7F);
yMap.draw(yCircle);
map.copyChannel(yMap, yMap.rect, new Point(0, 0), BitmapDataChannel.BLUE, BitmapDataChannel.BLUE);
yMap.dispose();

// Display the map image on the Stage, for reference.
var mapBitmap:Bitmap = new Bitmap(map);
this.addChild(mapBitmap);
mapBitmap.x = 400;
mapBitmap.y = stage.stageHeight - mapBitmap.height;

// This function creates the displacement map filter at the mouse location.
function magnify():void
{
    // Position the filter.
    var filterX:Number = (loader.mouseX) - (map.width / 2);
    var filterY:Number = (loader.mouseY) - (map.height / 2);
    var pt:Point = new Point(filterX, filterY);
    var xyFilter:DisplacementMapFilter = new DisplacementMapFilter();
    xyFilter.mapBitmap = map;
    xyFilter.mapPoint = pt;
    // The red in the map image will control x displacement.
    xyFilter.componentX = BitmapDataChannel.RED;
    // The blue in the map image will control y displacement.
    xyFilter.componentY = BitmapDataChannel.BLUE;
    xyFilter.scaleX = 35;
    xyFilter.scaleY = 35;
    xyFilter.mode = DisplacementMapFilterMode.IGNORE;
    loader.filters = [xyFilter];
}

// This function is called when the mouse moves. If the mouse is
// over the loaded image, it applies the filter.
function moveMagnifier(event:MouseEvent):void
{
    if (loader.hitTestPoint(loader.mouseX, loader.mouseY))
    {
        magnify();
    }
}
loader.addEventListener(MouseEvent.MOUSE_MOVE, moveMagnifier);
```

代码首先生成两个渐变圆，它们合并在一起构成置换图图像。红色圆创建 x 轴置换 (xyFilter.componentX = BitmapDataChannel.RED)，蓝色圆创建 y 轴置换 (xyFilter.componentY = BitmapDataChannel.BLUE)。为帮助您理解置换图图像的外观，代码向屏幕底部添加了原始圆形以及合并后作为置换图图像的圆形。

然后，代码加载一个图像，当鼠标移动时，将置换滤镜应用于鼠标下方的图像部分。用作置换图图像的渐变圆使置换区域从指针处向外扩展。请注意，置换图图像的灰色区域不会发生置换。灰色为 0x7F7F7F。该灰色的蓝色和红色通道与这些颜色通道中的中间色度完全匹配，因此在置换图图像的灰色区域不会发生置换。同样，在圆的中心也不会发生置换。由于蓝色和红色是引起置换的颜色，虽然颜色中没有灰色，但该颜色的蓝色通道和红色通道与中度灰的蓝色通道和红色通道完全相同，因此该处不发生置换。

着色器滤镜

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

ShaderFilter 类可使用定义为 Pixel Bender 着色器的自定义滤镜效果。由于该滤镜效果是以 Pixel Bender 着色器形式编写的，因此可完全自定义。过滤内容将作为图像输入传递给着色器，着色器操作的结果就是滤镜结果。

注：从 Flash Player 10 和 Adobe AIR 1.5 开始，在 ActionScript 中提供着色器滤镜。

若要对某个对象应用着色器滤镜，首先应创建一个表示要使用的 Pixel Bender 着色器的 Shader 实例。有关 Shader 实例创建过程以及如何指定输入图像和参数值的详细信息，请参阅第 251 页的 “ 使用 Pixel Bender 着色器 ”。

将着色器用作滤镜时，请记住以下三个要点：

- 必须将着色器定义为至少接受一个输入图像。
- 将过滤对象 （为之应用滤镜的显示对象或 BitmapData 对象）作为第一个输入图像值传递给着色器。因此，不要为第一个图像输入手动指定值。
- 如果着色器定义了多个输入图像，则必须手动指定其他输入（即，为属于 Shader 实例的任何 ShaderInput 实例设置 input 属性）。

创建着色器的 Shader 对象后，即创建了一个 ShaderFilter 实例。这就是使用方法与其他所有滤镜相同的实际滤镜对象。若要创建使用 Shader 对象的 ShaderFilter，请调用 ShaderFilter() 构造函数，并将 Shader 对象作为参数传递，如下所示：

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

有关使用着色器滤镜的完整示例，请参阅第 266 页的 “ 使用着色器作为滤镜 ”。

筛选显示对象示例：Filter Workbench

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Filter Workbench 提供了一个用户界面，用于对图像和其他可视内容应用不同的滤镜以及查看结果代码，这些代码可用于在 ActionScript 中生成相同的效果。除了提供试验滤镜的工具外，此应用程序还展示了以下技巧：

- 创建各种滤镜的实例
- 对显示对象应用多种滤镜

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/FilterWorkbench 文件夹中找到 Filter Workbench 应用程序文件。该应用程序包含以下文件：

文件	说明
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	提供应用程序主要功能的类，主要功能包括切换要应用滤镜的内容以及将滤镜应用于内容。
com/example/programmingas3/filterWorkbench/IFilterFactory.as	用来定义由各滤镜工厂类所实现的常用方法的接口。此接口定义 FilterWorkbenchController 类用来与各个滤镜工厂类交互的常见功能。
在文件夹 com/example/programmingas3/filterWorkbench/ 中： BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	类集，每个类都实现 IFilterFactory 接口。每个类均可为某一类型的滤镜提供创建和设置值的功能。应用程序中的滤镜属性面板使用这些工厂类来创建其特定滤镜的实例， FilterWorkbenchController 类将检索这些实例并将它们应用于图像内容。
com/example/programmingas3/filterWorkbench/IFilterPanel.as	定义某些类 （这些类定义用于操作应用程序中滤镜值的用户界面面板）所实现的常见方法的接口。
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	实用程序类，包含将数字颜色值转换为十六进制字符串格式的方法
com/example/programmingas3/filterWorkbench/GradientColor.as	作为值对象的类，此类将与 GradientBevelFilter 和 GradientGlowFilter 中各颜色相关联的三个值 （颜色、Alpha 和比例）合并到单个对象中
用户界面 (Flex)	
FilterWorkbench.mxml	定义应用程序用户界面的主要文件。
flexapp/FilterWorkbench.as	为主应用程序用户界面提供功能的类；此类用作应用程序 MXML 文件的代码隐藏类。

文件	说明
在 flexapp/filterPanels 文件夹中： BevelPanel.mxml BlurPanel.mxml ColorMatrixPanel.mxml ConvolutionPanel.mxml DropShadowPanel.mxml GlowPanel.mxml GradientBevelPanel.mxml GradientGlowPanel.mxml	MXML 组件集，为各面板 （用于为单个滤镜设置选项）提供功能。
flexapp/ImageContainer.as	作为屏幕上已加载图像的容器的显示对象
flexapp/controls/BGColorCellRenderer.as	用于更改 DataGrid 组件中单元格的背景颜色的自定义单元格渲染器
flexapp/controls/QualityComboBox.as	自定义控件，负责定义可用于多个滤镜面板中 “ 品质 ” 设置的组合框。
flexapp/controls/TypeComboBox.as	自定义控件，负责定义可用于多个滤镜面板中 “ 类型 ” 设置的组合框。
用户界面 (Flash)	
FilterWorkbench.fla	定义应用程序用户界面的主要文件。
flashapp/FilterWorkbench.as	为主应用程序用户界面提供功能的类；此类用作应用程序 FLA 文件的文档类。
在 flashapp/filterPanels 文件夹中： BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	类集，为各面板 （用于为单个滤镜设置选项）提供功能。 对于每个类而言，在主应用程序 FLA 文件库中，还有一个关联的 MovieClip 元件，其名称与类的名称相匹配（例如，“BlurPanel” 元件链接到 BlurPanel.as 中定义的类）。构成用户界面的组件在这些元件中进行定位和命名。
flashapp/ImageContainer.as	作为屏幕上已加载图像的容器的显示对象
flashapp/BGColorCellRenderer.as	用于更改 DataGrid 组件中单元格的背景颜色的自定义单元格渲染器
flashapp/ButtonCellRenderer.as	用于将 Button 组件包含在 DataGrid 组件的单元格中的自定义单元格渲染器

文件	说明
套用滤镜的图像内容	
com/example/programmingas3/filterWorkbench/ImageType.as	此类用作值对象，它包含应用程序可加载并对其应用滤镜的单个图像文件的类型和 URL。此类还包含一组代表实际可用图像文件的常量。
images/sampleAnimation.swf images/sampleImage1.jpg images/sampleImage2.jpg	应用程序中应用滤镜的图像和其他可视内容。

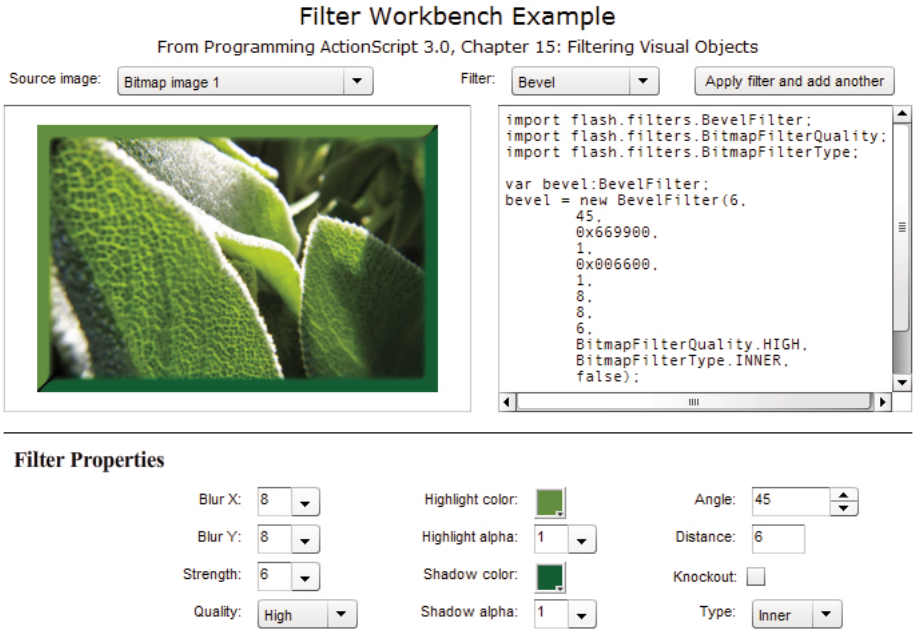
试验 ActionScript 滤镜

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Filter Workbench 应用程序旨在帮助您试验各种滤镜效果，并为该效果生成相关的 ActionScript 代码。使用该应用程序，您可以从包含可视内容 （包括位图图像和 Flash 创建的动画）的三种不同文件中进行选择，并可采用单独或与其他滤镜相结合的方式，将八种不同的 ActionScript 滤镜应用于选定的图像。此应用程序包含以下滤镜：

- 斜角 (flash.filters.BevelFilter)
- 模糊 (flash.filters.BlurFilter)
- 颜色矩阵 (flash.filters.ColorMatrixFilter)
- 卷积 (flash.filters.ConvolutionFilter)
- 投影 (flash.filters.DropShadowFilter)
- 发光 (flash.filters.GlowFilter)
- 渐变斜角 (flash.filters.GradientBevelFilter)
- 渐变发光 (flash.filters.GradientGlowFilter)

用户选定图像并将滤镜应用于图像后，应用程序将显示一个带有用于设置选定滤镜特定属性的控件的面板。例如，下列图像显示的是已选择 “ 斜角 ” 滤镜的应用程序：



如果用户调整滤镜属性，则预览将实时更新。用户还可以应用多个滤镜，方法是：自定义一个滤镜，单击 “ 应用 ” 按钮；自定义另一个滤镜，单击 “ 应用 ” 按钮，以此类推。

应用程序的滤镜面板具有一些功能和限制：

- 颜色矩阵滤镜包含一组用于直接操作常用图像属性 （包括亮度、对比度、饱和度和色相）的控件。另外，可以指定自定义颜色矩阵值。
- 卷积滤镜只有在使用 ActionScript 时才可用，此滤镜包含一组常用的卷积矩阵值，也可指定自定义值。但是，虽然 ConvolutionFilter 类可以接受任意大小的矩阵，但 Filter Workbench 应用程序使用的是固定的 3 x 3 矩阵，这是最常用的滤镜大小。
- 置换图滤镜和着色器滤镜仅在 ActionScript 中可用，在 Filter Workbench 应用程序中不可用。

创建滤镜实例

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Filter Workbench 应用程序包含一个类集，各可用滤镜均有一个对应的类，各面板用这些类来创建滤镜。用户选定滤镜后，与滤镜面板相关联的 ActionScript 代码就将创建相应滤镜工厂类的实例。（这些类称为工厂类，因为其用途在于创建其他对象的实例，这与实际工厂创建各产品极为相似）

只要用户更改面板上的属性值，面板的代码就会调用工厂类中的相应方法。各工厂类包含面板用于创建相应滤镜实例的特定方法。例如，如果用户选择 “ 模糊 ” 滤镜，则应用程序将创建 BlurFactory 实例。 BlurFactory 类包含一个 modifyFilter() 方法，该方法接受三个参数：blurX、 blurY 和 quality，这三个参数一起用于创建所需的 BlurFilter 实例：

```
private var _filter:BlurFilter;

public function modifyFilter(blurX:Number = 4, blurY:Number = 4, quality:int = 1):void
{
    _filter = new BlurFilter(blurX, blurY, quality);
    dispatchEvent(new Event(Event.CHANGE));
}
```

另一方面，如果用户选择“卷积”滤镜，因为该滤镜允许更大的灵活性，所以需要控制的属性就会更多。在 ConvolutionFactory 类中，如果用户在滤镜面板上选择不同的值，则将调用以下代码：

```
private var _filter:ConvolutionFilter;

public function modifyFilter(matrixX:Number = 0,
                                                matrixY:Number = 0,
                                                matrix:Array = null,
                                                divisor:Number = 1.0,
                                                bias:Number = 0.0,
                                                preserveAlpha:Boolean = true,
                                                clamp:Boolean = true,
                                                color:uint = 0,
                                                alpha:Number = 0.0):void
{
    _filter = new ConvolutionFilter(matrixX, matrixY, matrix, divisor, bias, preserveAlpha, clamp, color,
alpha);
    dispatchEvent(new Event(Event.CHANGE));
}
```

请注意，在各例中，如果滤镜值发生变化，则工厂对象将调度 Event.CHANGE 事件，从而向侦听器 ⊠ 通知滤镜的值已经改变。将滤镜实际应用于过滤内容的 FilterWorkbenchController 类将侦听该事件，以确定何时需检索滤镜的新副本并将其重新应用于过滤的内容。

FilterWorkbenchController 类无需知道各滤镜工厂类的特定详细信息，它只需知道滤镜已更改，并能够访问滤镜副本即可。为支持此目的，应用程序包含了一个 IFilterFactory 接口，该接口定义滤镜工厂类需提供的行为，以便应用程序的 FilterWorkbenchController 实例能够完成自己的工作。 IFilterFactory 定义在 FilterWorkbenchController 类中使用的 getFilter() 方法：

```
function getFilter():BitmapFilter;
```

请注意，getFilter() 接口方法定义指定该方法返回一个 BitmapFilter 实例，而非特定类型的滤镜。BitmapFilter 类不定义特定类型的滤镜，而是用来构建所有滤镜类的基类。各滤镜工厂类定义 getFilter() 方法的特定实现，此时将返回对所创建滤镜对象的引用。例如，以下就是 ConvolutionFactory 类源代码的缩写版本：

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

ConvolutionFactory 类在实现 getFilter() 方法时将返回一个 ConvolutionFilter 实例，尽管调用 getFilter() 的任何对象无需知道此返回结果。根据 ConvolutionFactory 所遵循的 getFilter() 方法的定义，此方法必须返回任意 BitmapFilter 实例，这些实例可能是任意 ActionScript 滤镜类的实例。

将滤镜应用于显示对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如前所述，Filter Workbench 应用程序使用 FilterWorkbenchController 类的实例（下文称为“控制器实例”），该实例执行将滤镜应用于所选可视对象的实际任务。控制器实例必须首先知道滤镜需应用于哪些图像或可视内容，然后才能应用滤镜。用户选定图像后，应用程序将调用 FilterWorkbenchController 类中的 setFilterTarget() 方法，传入在 ImageType 类中定义的一个常量：

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

控制器实例使用该信息来加载指定的文件，并将所加载的文件存储在一个名为 _currentTarget 的实例变量中：

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

用户选择滤镜后，应用程序将调用控制器实例的 setFilter() 方法，为控制器提供对相关滤镜工厂对象的引用，此引用存储在名为 _filterFactory 的实例变量中。

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

请注意，正如上文所说明的那样，控制器实例并不知道为其提供的滤镜工厂实例的具体数据类型，它只知道对象实现 IFilterFactory 实例，这意味着它具有 getFilter() 方法，并可在滤镜更改时调度 change (Event.CHANGE) 事件。

如果用户改变滤镜面板中的滤镜属性，则控制器实例将通过滤镜工厂的 change 事件发现滤镜已发生更改，该事件调用控制器实例的 filterChange() 方法。而该方法又将调用 applyTemporaryFilter() 方法：

```
private function filterChange(event:Event):void
{
    applyTemporaryFilter();
}

private function applyTemporaryFilter():void
{
    var currentFilter:BitmapFilter = _filterFactory.getFilter();

    // Add the current filter to the set temporarily
    _currentFilters.push(currentFilter);

    // Refresh the filter set of the filter target
    _currentTarget.filters = _currentFilters;

    // Remove the current filter from the set
    // (This doesn't remove it from the filter target, since
    // the target uses a copy of the filters array internally.)
    _currentFilters.pop();
}
```

将滤镜应用于显示对象的工作在 applyTemporaryFilter() 方法中进行。首先，控制器将通过调用滤镜工厂的 getFilter() 方法来检索对滤镜对象的引用。

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

控制器实例有一个名为 _currentFilters 的数组实例变量，所有曾应用于显示对象的滤镜均存储在该变量中。下一步就是将新近更新的滤镜添加到该数组：

```
_currentFilters.push(currentFilter);
```

然后，代码将滤镜数组分配给显示对象的 filters 属性，这会将滤镜实际应用于图像：

```
_currentTarget.filters = _currentFilters;
```

最后，由于此最近添加的滤镜仍旧是“工作”滤镜，不应将其永久应用于显示对象，因此将其从 _currentFilters 数组中删除：

```
_currentFilters.pop();
```

将此滤镜从数组中删除并不会影响已过滤的显示对象，因为在将滤镜数组分配给 filters 属性时，显示对象将生成滤镜数组的副本，而且显示对象使用的是内部数组而非原始数组。因此，对滤镜数组所做的任何更改均不会影响显示对象，直至数组被再次分配给显示对象的 filters 属性。

第 15 章：使用 Pixel Bender 着色器

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

利用 Adobe Pixel Bender 工具包，开发人员可以编写用于创建图形效果、执行其他图像和数据处理的着色器。 Pixel Bender 字节码可以在 ActionScript 中执行，从而将效果应用于图像数据或可视内容。通过在 ActionScript 中使用 Pixel Bender 着色器，不仅可以运用 ActionScript 的内置功能，而且还可以创建自定义的视觉效果并执行数据处理。

注: 从 Flash Player 10 和 Adobe AIR 1.5 开始支持 Pixel Bender。在 GPU 呈现下不支持 Pixel Bender 混合、滤镜和填充。

更多帮助主题

Adobe Pixel Bender 技术中心

Pixel Bender 开发人员指南

Pixel Bender 参考

flash.display.Shader

flash.filters.ShaderFilter

针对 Flash 的 Pixel Bender 基础知识

针对 Flex 的 Pixel Bender 基础知识

Pixel Bender 着色器基础知识

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Adobe Pixel Bender 是一种编程语言，用于创建或操作图像内容。使用 Pixel Bender （也称为着色器）创建内核。着色器定义了一个可对图像的每个像素单独执行的单一函数。对该函数的每次调用都将得到图像中该像素坐标处的输出颜色。可通过指定输入图像和参数值来自定义该操作。在着色器的单次执行中，输入值和参数值是不变的。唯一发生变化的是像素 （其颜色是函数调用的结果）的坐标。

对多个输出像素坐标调用着色器函数时，将尽可能采用并行方式。这样会改进着色器性能，提供高性能的处理能力。

在 ActionScript 中，可使用着色器轻松地创建三种类型的效果：

- 绘制填充
- 混合模式
- 滤镜

着色器也可以按独立模式执行。使用独立模式时，将直接访问着色器的结果，而非预先指定着色器的用途。结果可以按图像数据或者二进制或数值数据的形式访问。该数据完全不必是图像数据。这样一来，您可以为着色器输入一组数据。着色器将处理该数据，然后您可以访问着色器返回的结果数据。

从 Flash Player 10 和 Adobe AIR 1.5 开始支持 Pixel Bender。在 GPU 呈现下不支持 Pixel Bender 混合、滤镜和填充。在移动设备上， Pixel Bender 着色器在 CPU 呈现下运行。但是，其性能与在桌面计算机上相差甚远。许多着色器程序每秒只能执行几个帧。

重要概念和术语

以下参考列表包含创建和使用 Pixel Bender 着色器时会遇到的重要术语：

内核 对于 Pixel Bender，内核就相当于着色器。通过 Pixel Bender，您的代码定义了一个内核，它定义了可对图像的每个像素单独执行的单一函数。

Pixel Bender 字节码 编译 Pixel Bender 内核时，它会转换为 Pixel Bender 字节码。在运行时访问并执行字节码。

Pixel Bender 语言 用于创建 Pixel Bender 内核的编程语言。

Pixel Bender 工具包 用于根据 Pixel Bender 源代码创建 Pixel Bender 字节码文件的应用程序。您可以使用该工具包编写、测试和编译 Pixel Bender 源代码。

Shader 对于此文档而言，着色器是用 Pixel Bender 语言编写的一组功能。着色器的代码会创建视觉效果或执行计算。在任一情况下，着色器都返回一组数据 （通常为图像的像素）。着色器在每个数据点上执行的操作都相同，唯一的区别是输出像素的坐标。着色器不是用 ActionScript 编写的。它用 Pixel Bender 语言编写，并编译为 Pixel Bender 字节代码。着色器可在编译时嵌入 SWF 文件，也可在运行时作为外部文件加载。无论采用上述哪一种方式，都要在 ActionScript 中访问着色器，方法是先创建一个 Shader 对象，然后将其链接到着色器字节代码。

着色器输入 一种复杂的输入，通常为位图图像数据，提供给着色器供其计算之用。对于着色器中定义的每个输入变量，着色器的整个执行过程都使用单一变量值 （即，一个图像或一组二进制数据）。

着色器参数 提供给着色器的单个值 （或限定的一组值），供着色器进行计算用。着色器的每次执行中都会定义各个参数值，该值在着色器的整个执行过程中保持不变。

完成代码示例

您可能想测试提供的示例代码列表。测试代码包括运行代码，并在创建的 SWF 中查看结果。所有示例都使用绘图 API 创建内容，这类 API 使用着色器效果，或由该效果进行修改。

大多数示例代码清单都包含两个部分。一部分是此示例中所用着色器的 Pixel Bender 源代码。您必须首先使用 Pixel Bender 工具包将源代码编译成 Pixel Bender 字节代码文件。请遵循以下步骤创建 Pixel Bender 字节代码文件：

1. 打开 Adobe Pixel Bender 工具包。如有必要，从 “ 生成 ” 菜单中选择 “ 打开 Flash Player 警告和错误 ”。
2. 复制 Pixel Bender 代码清单，然后粘贴到 Pixel Bender 工具包的代码编辑器窗格中。
3. 从 “ 文件 ” 菜单中，选择 “ 为 Flash Player 导出内核滤镜 ”。
4. 将 Pixel Bender 字节代码文件保存到 Flash 文档所在的目录。文件名应与示例说明中指定的名称一致。

每个示例的 ActionScript 部分都编写为一个类文件。要在 Flash Professional 中测试示例，请执行下列操作：

1. 创建一个空的 Flash 文档并将它保存到您的计算机上。
2. 创建一个新的 ActionScript 文件，并将它保存到 Flash 文档所在的目录中。文件名应与代码清单中的类的名称一致。例如，如果代码清单定义了一个名为 MyApplication 的类，则使用 MyApplication.as 名称保存 ActionScript 文件。
3. 将代码清单复制到 ActionScript 文件中并保存该文件。
4. 在 Flash 文档中，单击舞台或工作区的空白部分，以激活文档的 “ 属性 ” 检查器。
5. 在 “ 属性 ” 检查器的 “ 文档类 ” 字段中，输入您从文本中复制的 ActionScript 类的名称。
6. 使用 “ 控制 ”>“ 测试影片 ” 来运行程序

   您将在预览窗口中看到示例的结果。

第 939 页的 “ 如何使用 ActionScript 示例 ” 中详细介绍了测试示例代码清单的方法。

加载或嵌入着色器

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在 ActionScript 中使用 Pixel Bender 着色器的第一步是在 ActionScript 代码中访问着色器。因为着色器是用 Adobe Pixel Bender 工具包创建并以 Pixel Bender 语言编写的，所以不能在 ActionScript 中直接访问。您需要创建 Shader 类的一个实例，用于向 ActionScript 表示 Pixel Bender 着色器。通过 Shader 对象可以查明着色器的有关信息，如着色器是否需要参数或输入图像值。将 Shader 对象传递给其他对象即可实际使用着色器。例如，若要将着色器用作滤镜，可将 Shader 对象分配给 ShaderFilter 对象的 shader 属性。若要将着色器用作绘制填充，可将 Shader 对象作为参数传递给 Graphics.beginShaderFill() 方法。

ActionScript 代码可以通过两种方式访问由 Adobe Pixel Bender 工具包创建的着色器 （.pbj 文件）：

- 在运行时加载：可以使用 URLLoader 对象将着色器文件作为外部资源进行加载。这种方法类似于加载外部资源 （如文本文件）。下面的示例演示如何在运行时加载着色器字节码文件并将其链接到一个 Shader 实例：

```
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.BINARY;
loader.addEventListener(Event.COMPLETE, onLoadComplete);
loader.load(new URLRequest("myShader.pbj"));

var shader:Shader;

function onLoadComplete(event:Event):void {
    // Create a new shader and set the loaded data as its bytecode
    shader = new Shader();
    shader.byteCode = loader.data;

    // You can also pass the bytecode to the Shader() constructor like this:
    // shader = new Shader(loader.data);

     // do something with the shader
}
```

- 嵌入在 SWF 文件中：使用 [Embed] 元数据标签可以在编译时将着色器文件嵌入在 SWF 文件中。只有在使用 Flex SDK 编译 SWF 文件时， [Embed] 元数据标签才可用。 [Embed] 标签的 source 参数指向着色器文件，其 mimeType 参数为 "application/octet-stream"，如以下示例中所示：

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shader:Shader = new Shader();
shader.byteCode = new MyShaderClass();

// You can also pass the bytecode to the Shader() constructor like this:
// var shader:Shader = new Shader(new MyShaderClass());

// do something with the shader
```

在任何一种情况下，都可以将原始着色器字节码 （URLLoader.data 属性或 [Embed] 数据类的实例）链接到 Shader 实例。如以上示例所示，可以通过两种方式将字节码分配给 Shader 实例。可以将着色器字节代码作为参数传递到 Shader() 构造函数。或者，可以将其设置为 Shader 实例的 byteCode 属性。

在创建 Pixel Bender 着色器并将其链接到 Shader 对象后，即可使用着色器以多种方式创建效果。您可以将着色器用作滤镜、混合模式、位图填充，也可以用于位图或其他数据的独立处理。您还可以使用 Shader 对象的 data 属性访问着色器的元数据、指定输入图像以及设置参数值。

访问着色器元数据

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在创建 Pixel Bender 着色器内核时，作者可以在 Pixel Bender 源代码中指定着色器的相关元数据。在 ActionScript 中使用着色器时，可以检查着色器和提取其元数据。

在创建 Shader 实例并将其链接到 Pixel Bender 着色器时，会创建一个包含着色器相关数据的 ShaderData 对象并将该对象存储在 Shader 对象的 data 属性中。ShaderData 类不定义自身的任何属性。但是，对于在着色器源代码中定义的每个元数据值，都会在运行时将一个属性动态添加到 ShaderData 对象。为每个属性提供的名称与在元数据中指定的名称相同。例如，假设 Pixel Bender 着色器的源代码包括下面的元数据定义：

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

则为该着色器创建 ShaderData 对象时将使用以下属性和值：

- namespace （字符串）："Adobe::Example"
- vendor （字符串）："Bob Jones"
- version （字符串）："1"
- description （字符串）："Creates a version of the specified image with the specified brightness"

因为元数据属性是动态添加到 ShaderData 对象的，所以可以使用 for..in 循环检查 ShaderData 对象。通过这种方法，可以确定着色器是否具有元数据以及元数据是何值。除了元数据属性之外， ShaderData 对象还可能具有表示着色器中定义的输入和参数的属性。在使用 for..in 循环检查 ShaderData 对象时，检查每个属性的数据类型以确定属性是输入 （ShaderInput 实例）、参数 （ShaderParameter 实例）还是元数据值 （String 实例）。下面的示例演示如何使用 for..in 循环检查着色器的 data 属性的动态属性。每个元数据值都会添加到一个名为 metadata 的 Vector 实例。请注意，此示例假设已创建了名为 myShader 的 Shader 实例：

```
var shaderData:ShaderData = myShader.data;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (!(shaderData[prop] is ShaderInput) && !(shaderData[prop] is ShaderParameter))
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the metadata
```

有关增加着色器输入和参数提取的此示例版本，请参阅第 255 页的 “ 识别着色器输入和参数 ”。有关输入和参数属性的详细信息，请参阅第 254 页的 “ 指定着色器输入和参数值 ”。

指定着色器输入和参数值

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

许多 Pixel Bender 着色器定义为使用一个或多个在着色器处理中使用的输入图像。例如，一种常见情况是，着色器接受一个源图像，然后输出应用了特定效果的该图像。根据着色器的使用方式，可以自动指定输入值，也可能需要明确提供值。同样，许多着色器指定用于自定义着色器输出的参数。在使用着色器之前，还必须为每个参数明确设置一个值。

使用 Shader 对象的 data 属性可以设置着色器输入和参数，还可以确定特定着色器是否需要输入或参数。 data 属性是一个 ShaderData 实例。

识别着色器输入和参数

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

指定着色器输入和参数值的第一步是确定要使用的特定着色器是否需要输入图像或参数。每个 Shader 实例都有一个包含 ShaderData 对象的 data 属性。如果着色器定义了任何输入或参数，则这些输入或参数可作为该 ShaderData 对象的属性进行访问。属性的名称与在着色器源代码中为输入和参数指定的名称相匹配。例如，如果着色器定义了一个名为 src 的输入，则 ShaderData 对象具有一个名为 src 的属性用于表示该输入。每个表示输入的属性都是一个 ShaderInput 实例，而每个表示参数的属性都是一个 ShaderParameter 实例。

理想情况下，着色器的创作者会提供着色器文档，用于指示着色器所需的输入图像值和参数、其所表示的内容以及正确的值等等。

但是，如果着色器不带有文档 （并且您没有其源代码），则可以检查着色器数据以确定输入和参数。表示输入和参数的属性会动态添加到 ShaderData 对象。因此，可以使用 for..in 循环检查 ShaderData 对象，以确定其关联着色器是否定义了输入或参数。如第 254 页的 " 访问着色器元数据 " 中所述，为着色器定义的任何元数据值还可以作为添加到 Shader.data 属性的动态属性进行访问。在使用这种方法识别着色器输入和参数时，请检查动态属性的数据类型。如果属性是 ShaderInput 实例，则表示输入。如果是 ShaderParameter 实例，则表示参数。其他情况下，则是元数据值。下面的示例演示如何使用 for..in 循环检查着色器的 data 属性的动态属性。每个输入 （ShaderInput 对象）都会添加到一个名为 inputs 的 Vector 实例。每个参数（ShaderParameter 对象）都会添加到一个名为 parameters 的 Vector 实例。最后，所有元数据属性都会添加到一个名为 metadata 的 Vector 实例。请注意，此示例假设已创建了名为 myShader 的 Shader 实例：

```
var shaderData:ShaderData = myShader.data;
var inputs:Vector.<ShaderInput> = new Vector.<ShaderInput>();
var parameters:Vector.<ShaderParameter> = new Vector.<ShaderParameter>();
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in shaderData)
{
    if (shaderData[prop] is ShaderInput)
    {
        inputs[inputs.length] = shaderData[prop];
    }
    else if (shaderData[prop] is ShaderParameter)
    {
        parameters[parameters.length] = shaderData[prop];
    }
    else
    {
        metadata[metadata.length] = shaderData[prop];
    }
}

// do something with the inputs or properties
```

指定着色器输入值

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

许多着色器需要一个或多个在着色器处理中使用的输入图像。但是，在很多情况下，输入是在使用 Shader 对象时自动指定的。例如，假定着色器需要一个输入，并且该着色器用作滤镜。在滤镜应用于显示对象或 BitmapData 对象时，该对象会自动设置为输入。在这种情况下，无需明确设置输入值。

但是，在某些情况下，特别是在着色器定义多个输入时，一定要明确设置输入值。在 ActionScript 中，在着色器中定义的每个输入都是由一个 ShaderInput 对象表示的。如第 255 页的 " 识别着色器输入和参数 " 中所述，ShaderInput 对象是 Shader 对象的 data 属性中的 ShaderData 实例的属性。例如，假定着色器定义一个名为 src 的输入，并且该着色器链接到一个名为 myShader 的 Shader 对象。在这种情况下，可使用下面的标识符访问对应于 src 输入的 ShaderInput 对象：

```
myShader.data.src
```

每个 ShaderInput 对象都有一个 input 属性，该属性用于设置输入值。将 input 属性设置为 BitmapData 实例可指定图像数据。此外，还可以将 input 属性设置为 BitmapData 或 Vector。 <Number> 实例以指定二进制或数字数据。有关使用 BitmapData 或 Vector.<Number> 实例作为输入的详细信息和限制，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 ShaderInput.input 列表。

除了 input 属性之外，ShaderInput 对象还具有可用于确定输入所需图像类型的属性。这些属性包括 width、height 和 channels 属性。每个 ShaderInput 对象还有一个 index 属性，该属性用于确定是否必须为输入提供显式值。如果着色器需要的输入数量大于自动设置的输入数量，则需要为这些输入设置值。有关使用着色器的不同方式，以及是否自动设置输入值的详细信息，请参阅第 259 页的 " 使用着色器 "。

指定着色器参数值

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

某些着色器定义了参数值，着色器在创建其结果时会使用这些参数值。例如，更改图像亮度的着色器可能会指定一个亮度参数，该参数确定操作影响亮度的程度。根据着色器中的参数定义，着色器中定义的单个参数可能需要单个值或多个值。在 ActionScript 中，在着色器中定义的每个参数都由一个 ShaderParameter 对象表示。如第 255 页的 " 识别着色器输入和参数 " 中所述，ShaderParameter 对象是 Shader 对象的 data 属性中的 ShaderData 实例的属性。例如，假定着色器定义一个名为 brightness 的参数，并且该着色器由一个名为 myShader 的 Shader 对象表示。在这种情况下，可使用下面的标识符访问对应于 brightness 参数的 ShaderParameter：

```
myShader.data.brightness
```

若要为该参数设置一个 （或多个）值，请创建包含这个值 （或这些值）的 ActionScript 数组，并将该数组分配给 ShaderParameter 对象的 value 属性。 value 属性定义为 Array 实例，是因为单个着色器参数可能需要多个值。即使着色器参数只需要一个值，也必须将该值包含在 Array 对象中才能将它分配给 ShaderParameter.value 属性。下面的代码演示如何将单个值设置为 value 属性：

```
myShader.data.brightness.value = [75];
```

如果着色器的 Pixel Bender 源代码为参数定义默认值，则在创建 Shader 对象时，会创建包含默认值的数组，并将其分配给 ShaderParameter 对象的 value 属性。将数组分配给 value 属性之后 （包括该数组是默认数组的情况），可以通过更改数组元素的值来更改参数值。您不需要创建新数组并将它分配给 value 属性。

下面的示例演示如何在 ActionScript 中设置着色器的参数值。在此示例中，着色器定义一个名为 color 的参数。在 Pixel Bender 源代码中，color 参数声明为 float4 变量，这意味着该参数是包含四个浮点数的数组。在该示例中，color 参数值会连续更改，该参数每次更改时，都会使用着色器在屏幕上绘制一个有色的矩形。产生的结果是具有动画效果的颜色更改。

注：此示例的代码由 Ryan Taylor 编写。感谢 Ryan 分享此示例。若要查看 Ryan 的个人资料并阅读他的文章，请访问 www.boostworthy.com/。

这段 ActionScript 代码以下面三个方法为中心：

- init()：在 init() 方法中，代码加载包含着色器的 Pixel Bender 字节代码文件。加载文件后，调用 onLoadComplete() 方法。
- onLoadComplete()：在 onLoadComplete() 方法中，代码创建名为 shader 的 Shader 对象。同时还创建名为 texture 的 Sprite 实例。在 renderShader() 方法中，代码将着色器结果逐帧绘制在 texture 中。
- onEnterFrame()：onEnterFrame() 方法按每帧一次的频率进行调用，用于创建动画效果。在此方法中，代码将着色器参数值设置为新颜色，然后调用 renderShader() 方法将着色器结果绘制为矩形。

- renderShader()：在 renderShader() 方法中，代码调用 Graphics.beginShaderFill() 方法来指定着色器填充。然后，绘制一个矩形，该矩形的填充由着色器输出 （生成的颜色）定义。有关以此方法使用着色器的详细信息，请参阅第 259 页的 “ 使用着色器作为绘制填充 ”。

下面是此示例的 ActionScript 代码。使用此类作为 Flash Builder 中纯 ActionScript 项目的主应用程序类，或者作为 Flash Professional 中 FLA 文件的文档类：

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ColorFilterExample extends Sprite
    {
        private const DELTA_OFFSET:Number = Math.PI * 0.5;
        private var loader:URLLoader;
        private var shader:Shader;
        private var texture:Sprite;
        private var delta:Number = 0;

        public function ColorFilterExample()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ColorFilter.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            texture = new Sprite();

            addChild(texture);

            addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void
        {
```

```
            shader.data.color.value[0] = 0.5 + Math.cos(delta - DELTA_OFFSET) * 0.5;
            shader.data.color.value[1] = 0.5 + Math.cos(delta) * 0.5;
            shader.data.color.value[2] = 0.5 + Math.cos(delta + DELTA_OFFSET) * 0.5;
            // The alpha channel value (index 3) is set to 1 by the kernel's default
            // value. This value doesn't need to change.

            delta += 0.1;

            renderShader();
        }

        private function renderShader():void
        {
            texture:graphics.clear();
            texture.graphics.beginShaderFill(shader);
            texture.graphics.drawRect(0, 0, stage.stageWidth, stage.stageHeight);
            texture.graphics.endFill();
        }
    }
}
```

下面是 ColorFilter 着色器内核的源代码，用于创建 “ColorFilter.pbj”Pixel Bender 字节代码文件：

```
<languageVersion : 1.0;>
kernel ColorFilter
<
    namespace : "boostworthy::Example";
    vendor : "Ryan Taylor";
    version : 1;
    description : "Creates an image where every pixel has the specified color value.";
>
{
    output pixel4 result;

    parameter float4 color
    <
        minValue:float4(0, 0, 0, 0);
        maxValue:float4(1, 1, 1, 1);
        defaultValue:float4(0, 0, 0, 1);
    >;

    void evaluatePixel()
    {
        result = color;
    }
}
```

如果所使用的着色器的参数没有文档说明，则通过检查 ShaderParameter 对象的 type 属性，便可确定数组中必须包含的元素数量及其类型。 type 属性指示着色器本身定义的参数数据类型。有关每种参数类型需要的元素数量和类型的列表，请参阅 “ActionScript 3.0 参考 ” 中的 ShaderParameter.value 属性列表。

每个 ShaderParameter 对象还有一个 index 属性，该属性指示参数在着色器参数顺序中的位置。除了这些属性之外，ShaderParameter 对象还可以具有包含由着色器创作者提供的元数据值的其他属性。例如，创作者可以指定元数据值，如参数的最小值、最大值和默认值。创作者所指定的所有元数据值都会作为动态属性添加到 ShaderParameter 对象中。若要检查这些属性，请使用 for..in 循环来循环访问 ShaderParameter 对象的动态属性，以识别其元数据。下面的示例演示如何使用 for..in 循环来识别 ShaderParameter 对象的元数据。每个元数据值都会添加到一个名为 metadata 的 Vector 实例。请注意，此示例假定已创建了一个名为 myShader 的 Shader 实例，并且该实例具有一个名为 brightness 的参数：

```
var brightness:ShaderParameter = myShader.data.brightness;
var metadata:Vector.<String> = new Vector.<String>();

for (var prop:String in brightness)
{
    if (brightness[prop] is String)
    {
        metadata[metadata.length] = brightness[prop];
    }
}

// do something with the metadata
```

使用着色器

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

只要 Pixel Bender 着色器可以在 ActionScript 中用作 Shader 对象，就可以通过多种方式使用：

- 着色器绘制填充：着色器定义使用绘图 API 绘制的形状的填充部分。
- 混合模式：着色器定义两个重叠的显示对象之间的混合。
- 滤镜：着色器定义用于修改可视内容外观的滤镜。
- 独立着色器处理：着色器处理在不指定输出的目标用途的情况下运行。着色器也可以在后台运行，结果在处理完成时可用。这种方法可用于生成位图数据，还可用于处理非可视数据。

使用着色器作为绘制填充

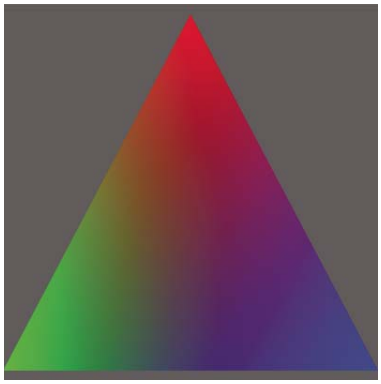
Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在使用着色器创建绘制填充时，您将使用绘图 API 方法来创建矢量形状。正如使用绘图 API 可将任何位图图像用作位图填充一样，着色器的输出将用于填充该形状。若要创建着色器填充，可在代码中开始绘制形状的位置处，调用 Graphics 对象的 beginShaderFill() 方法。将 Shader 对象作为第一个参数传递给 beginShaderFill() 方法，如以下清单所示：

```
var canvas:Sprite = new Sprite();
canvas.graphics.beginShaderFill(myShader);
canvas.graphics.drawRect(10, 10, 150, 150);
canvas.graphics.endFill();
// add canvas to the display list to see the result
```

在使用着色器作为绘制填充时，您将设置着色器需要的所有输入图像值和参数值。

下面的示例演示如何使用着色器作为绘制填充。在此示例中，着色器创建了一个三点渐变。此渐变具有三种颜色，分别位于三角形的三个顶点，之间的颜色为渐变混合。此外，颜色进行转动，产生旋转色的动画效果。



注：此示例的代码由 Petri Leskinen 编写。感谢 Petri 分享此示例。若要查看 Petri 的更多示例和教程，请访问 http://pixelero.wordpress.com/。

这段 ActionScript 代码用到三个方法：

- init()：应用程序加载时调用 init() 方法。在此方法中，代码为代表三角形三个顶点的 Point 对象设置初始值。此外还创建名为 canvas 的 Sprite 实例。稍后，在 updateShaderFill() 方法中，代码将着色器结果逐帧绘制在 canvas 中。最后，代码加载着色器字节代码文件。
- onLoadComplete()：在 onLoadComplete() 方法中，代码创建名为 shader 的 Shader 对象。此外还设置初始参数值。最后，代码添加 updateShaderFill() 方法作为 enterFrame 事件的侦听器，表示逐帧调用该方法来创建动画效果。
- updateShaderFill()：updateShaderFill() 方法是逐帧调用的，用于创建动画效果。在此方法中，代码计算并设置着色器的参数值。然后，代码调用 beginShaderFill() 方法来创建着色器填充，调用其他绘图 API 方法在三角形中绘制着色器结果。

下面是此示例的 ActionScript 代码。使用此类作为 Flash Builder 中纯 ActionScript 项目的主应用程序类，或者作为 Flash Professional 中 FLA 文件的文档类：

```
package
{
    import flash.display.Shader;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Point;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class ThreePointGradient extends Sprite
    {
        private var canvas:Sprite;
        private var shader:Shader;
        private var loader:URLLoader;

        private var topMiddle:Point;
        private var bottomLeft:Point;
        private var bottomRight:Point;

        private var colorAngle:Number = 0.0;
        private const d120:Number = 120 / 180 * Math.PI; // 120 degrees in radians


        public function ThreePointGradient()
```

```
        {
            init();
        }

        private function init():void
        {
            canvas = new Sprite();
            addChild(canvas);

            var size:int = 400;
            topMiddle = new Point(size / 2, 10);
            bottomLeft = new Point(0, size - 10);
            bottomRight = new Point(size, size - 10);

            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("ThreePointGradient.pbj"));
        }

        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            shader.data.point1.value = [topMiddle.x, topMiddle.y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];

            addEventListener(Event.ENTER_FRAME, updateShaderFill);
        }

        private function updateShaderFill(event:Event):void
        {
            colorAngle += .06;

            var c1:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle);
            var c2:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle + d120);
            var c3:Number = 1 / 3 + 2 / 3 * Math.cos(colorAngle - d120);

            shader.data.color1.value = [c1, c2, c3, 1.0];
            shader.data.color2.value = [c3, c1, c2, 1.0];
            shader.data.color3.value = [c2, c3, c1, 1.0];

            canvas.graphics.clear();
            canvas.graphics.beginShaderFill(shader);

            canvas.graphics.moveTo(topMiddle.x, topMiddle.y);
            canvas.graphics.lineTo(bottomLeft.x, bottomLeft.y);
            canvas.graphics.lineTo(bottomRight.x, bottomLeft.y);

            canvas.graphics.endFill();
        }
    }
}
```

下面是 ThreePointGradient 着色器内核的源代码，用于创建 “ThreePointGradient.pbj”Pixel Bender 字节代码文件：

```
<languageVersion : 1.0;>
kernel ThreePointGradient
<
    namespace : "Petri Leskinen::Example";
    vendor : "Petri Leskinen";
    version : 1;
    description : "Creates a gradient fill using three specified points and colors.";
>
{
    parameter float2 point1 // coordinates of the first point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 0);
    >;

    parameter float4 color1 // color at the first point, opaque red by default
    <
        defaultValue:float4(1.0, 0.0, 0.0, 1.0);
    >;

    parameter float2 point2 // coordinates of the second point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color2 // color at the second point, opaque green by default
    <
        defaultValue:float4(0.0, 1.0, 0.0, 1.0);
    >;

    parameter float2 point3 // coordinates of the third point
    <
        minValue:float2(0, 0);
        maxValue:float2(4000, 4000);
        defaultValue:float2(0, 500);
    >;

    parameter float4 color3 // color at the third point, opaque blue by default
    <
        defaultValue:float4(0.0, 0.0, 1.0, 1.0);
    >;
```

```
    output pixel4 dst;

    void evaluatePixel()
    {
        float2 d2 = point2 - point1;
        float2 d3 = point3 - point1;

        // transformation to a new coordinate system
        // transforms point 1 to origin, point2 to (1, 0), and point3 to (0, 1)
        float2x2 mtrx = float2x2(d3.y, -d2.y, -d3.x, d2.x) / (d2.x * d3.y - d3.x * d2.y);
        float2 pNew = mtrx * (outCoord() - point1);

        // repeat the edge colors on the outside
        pNew.xy = clamp(pNew.xy, 0.0, 1.0); // set the range to 0.0 ... 1.0

        // interpolating the output color or alpha value
        dst = mix(mix(color1, color2, pNew.x), color3, pNew.y);
    }
}
```

注：如果在图形处理单元 (GPU) 下呈现时使用着色器填充，则填充区域将以蓝绿色着色。

有关使用绘图 API 绘制形状的详细信息，请参阅第 185 页的 “ 使用绘图 API”。

使用着色器作为混合模式

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

使用着色器作为混合模式与使用其他混合模式类似。着色器定义的外观取决于将两个显示对象混合在一起的视觉效果。若要将着色器用于混合模式，请将 Shader 对象指派给前景显示对象的 blendShader 属性。如果为 blendShader 属性指定非 null 值，显示对象的 blendMode 属性将自动设置为 BlendMode.SHADER。下面的清单演示如何使用着色器作为混合模式。注意，此示例假定存在名为 foreground 的显示对象。该对象与其他显示内容包含在显示列表的同一父级中，而 foreground 与其他内容重叠：

```
foreground.blendShader = myShader;
```

使用着色器作为混合模式时，着色器必须由至少两个输入定义。如示例所示，您未在代码中设置输入值。而是将两个混合后的图像自动用作着色器的输入。前景图像设置为第二个图像。（此显示对象便是要对其应用混合模式的对象。）背景图像由前景图像边框后的所有像素组合而成。背景图像设置为第一个输入图像。如果所用着色器要求两个以上的输入，则还需为前两个之外的其他输入提供值。

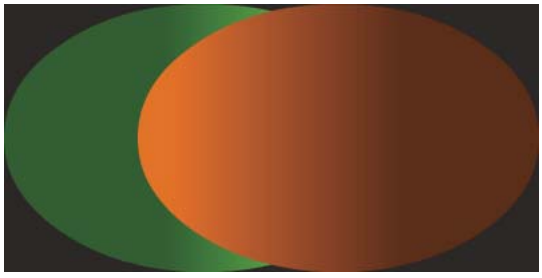
下面的示例演示如何使用着色器作为混合模式。此示例使用基于亮度的加亮混合模式。混合的结果是以任一混合对象中最亮的像素值显示该像素。

注：此示例的代码由 Mario Klingemann 编写。感谢 Mario 分享此示例。若要查看 Mario 的更多作品及阅读他的文章，请访问 www.quasimondo.com/。

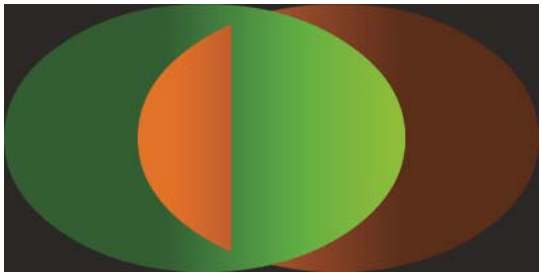
这段重要的 ActionScript 代码用到下面两个方法：

- init()：应用程序加载时调用 init() 方法。在此方法中，代码加载着色器字节代码文件。
- onLoadComplete()：在 onLoadComplete() 方法中，代码创建名为 shader 的 Shader 对象。然后绘制三个对象。第一个对象 backdrop 是混合对象后的深灰色背景。第二个对象 backgroundShape 是绿色渐变椭圆。第三个对象 foregroundShape 是橙色渐变椭圆。

foregroundShape 椭圆是混合的前景对象。混合的背景图像由部分 backdrop 和部分 backgroundShape 构成，并叠加了 foregroundShape 对象的边框。 foregroundShape 对象是位于显示列表最前面的对象。它与 backgroundShape 部分重叠，与 backdrop 全部重叠。由于上述重叠的存在，如果不应用混合模式，橙色椭圆 (foregroundShape) 将全部显示，并遮盖住绿色椭圆 (backgroundShape) 的一部分：



而在应用混合模式之后，绿色椭圆的较亮部分会 " 穿透 " 前景显示出来，因为该部分比遮盖它的那部分 foregroundShape 更亮。



下面是此示例的 ActionScript 代码。使用此类作为 Flash Builder 中纯 ActionScript 项目的主应用程序类，或者作为 Flash Professional 中 FLA 文件的文档类：

```
package
{
    import flash.display.BlendMode;
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class LumaLighten extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function LumaLighten()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
```

```
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("LumaLighten.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var backdrop:Shape = new Shape();
            var g0:Graphics = backdrop.graphics;
            g0.beginFill(0x303030);
            g0.drawRect(0, 0, 400, 200);
            g0.endFill();
            addChild(backdrop);

            var backgroundShape:Shape = new Shape();
            var g1:Graphics = backgroundShape.graphics;
            var c1:Array = [0x336600, 0x80ff00];
            var a1:Array = [255, 255];
            var r1:Array = [100, 255];
            var m1:Matrix = new Matrix();
            m1.createGradientBox(300, 200);
            g1.beginGradientFill(GradientType.LINEAR, c1, a1, r1, m1);
            g1.drawEllipse(0, 0, 300, 200);
            g1.endFill();
            addChild(backgroundShape);

            var foregroundShape:Shape = new Shape();
            var g2:Graphics = foregroundShape.graphics;
            var c2:Array = [0xff8000, 0x663300];
            var a2:Array = [255, 255];
            var r2:Array = [100, 255];
            var m2:Matrix = new Matrix();
            m2.createGradientBox(300, 200);
            g2.beginGradientFill(GradientType.LINEAR, c2, a2, r2, m2);
            g2.drawEllipse(100, 0, 300, 200);
            g2.endFill();
            addChild(foregroundShape);

            foregroundShape.blendShader = shader;
            foregroundShape.blendMode = BlendMode.SHADER;
        }
    }
}
```

下面是 LumaLighten 着色器内核的源代码，用于创建 “LumaLighten.pbj”Pixel Bender 字节代码文件：

```
<languageVersion : 1.0;>
kernel LumaLighten
<
    namespace : "com.quasimondo.blendModes";
    vendor : "Quasimondo.com";
    version : 1;
    description : "Luminance based lighten blend mode";
>
{
    input image4 background;
    input image4 foreground;

    output pixel4 dst;

    const float3 LUMA = float3(0.212671, 0.715160, 0.072169);

    void evaluatePixel()
    {
        float4 a = sampleNearest(foreground, outCoord());
        float4 b = sampleNearest(background, outCoord());
        float luma_a = a.r * LUMA.r + a.g * LUMA.g + a.b * LUMA.b;
        float luma_b = b.r * LUMA.r + b.g * LUMA.g + b.b * LUMA.b;

        dst = luma_a > luma_b ? a : b;
    }
}
```

有关使用混合模式的详细信息，请参阅第 154 页的 “ 应用混合模式 ”。

注：当 Pixel Bender 着色器程序以混合模式在 Flash Player 或 AIR 中运行时，采样和 outCoord() 函数的行为与在其他上下文中不同。在混合模式中，采样函数将始终返回着色器计算的当前像素。例如，您不能为了采集邻近像素而向 outCoord() 中添加偏移。同样，如果您使用 outCoord() 函数而不使用采样函数，则其坐标的计算结果将始终为 0。例如，您无法利用像素的位置来影响混合图像的合并方式。

使用着色器作为滤镜

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

使用着色器作为滤镜与在 ActionScript 中使用任何其他滤镜类似。使用着色器作为滤镜时，过滤出的图像 （显示对象或 BitmapData 对象）将传递给着色器。着色器使用输入图像来创建滤镜输出，该输出通常为原始图像经过修改的版本。如果过滤出的对象是显示对象，着色器输出将显示在屏幕上，代替过滤出的显示对象。如果过滤出的对象是 BitmapData 对象，着色器输出将成为 BitmapData 对象的内容，并调用该对象的 applyFilter() 方法。

若要使用着色器作为滤镜，您首先要创建 Shader 对象，如第 253 页的 “ 加载或嵌入着色器 ”中所述。接下来，您需要创建链接到该 Shader 对象的 ShaderFilter 对象。ShaderFilter 对象便是将应用到所过滤对象的滤镜。将滤镜应用于对象的方式与应用任何滤镜的方式相同。将其传递给显示对象的 filters 属性，或者对 BitmapData 对象调用 applyFilter() 方法。例如，下面的代码创建 ShaderFilter 对象，并将此滤镜应用到名为 homeButton 的显示对象。

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

在使用着色器作为滤镜时，着色器必须由至少一个输入定义。如示例所示，您未在代码中设置输入值。而是将过滤出的显示对象或 BitmapData 对象设置为输入图像。如果所用着色器要求一个以上的输入，则还需为第一个之外的其他输入提供值。

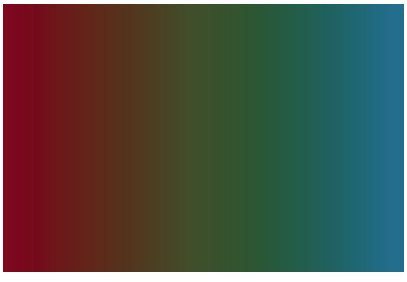
在某些情况下，滤镜会改变原始图像的尺寸。例如，典型的投影效果会添加额外的像素，这些像素组成为图像添加的阴影。在使用更改图像尺寸的着色器时，需设置 leftExtension、rightExtension、topExtension 和 bottomExtension 属性，指明所需图像尺寸更改量。

下面的示例演示如何使用着色器作为滤镜。此示例中的滤镜将反转图像红绿蓝三色通道的值。其结果为该图像的 “ 负片 ” 版本。

注：此示例使用的着色器为 Pixel Bender 工具包中附带的 invertRGB.pbk Pixel Bender 内核。您可以从 Pixel Bender 工具包安装目录中加载此内核的源代码。编译源代码，然后将字节代码文件保存到源代码所在的目录。

这段重要的 ActionScript 代码用到下面两个方法：

- init()：应用程序加载时调用 init() 方法。在此方法中，代码加载着色器字节代码文件。
- onLoadComplete()：在 onLoadComplete() 方法中，代码创建名为 shader 的 Shader 对象。然后创建并绘制名为 target 的对象的内容。 target 对象是用线性渐变色填充的矩形：左边是红色，中间是黄绿色，右边是淡蓝色。未过滤的对象看上去是这样：



应用了滤镜后，颜色反转，矩形变成这样：

此示例使用的着色器为 Pixel Bender 工具包中附带的 “invertRGB.pbk”Pixel Bender 范例内核。源代码位于 Pixel Bender 工具包安装目录的 “invertRGB.pbk” 文件中。编译源代码，然后以文件名 “invertRGB.pbj” 将字节代码文件保存在 ActionScript 源代码所在的目录中。

下面是此示例的 ActionScript 代码。使用此类作为 Flash Builder 中纯 ActionScript 项目的主应用程序类，或者作为 Flash Professional 中 FLA 文件的文档类：

```
package
{
    import flash.display.GradientType;
    import flash.display.Graphics;
    import flash.display.Shader;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.filters.ShaderFilter;
    import flash.events.Event;
    import flash.geom.Matrix;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class InvertRGB extends Sprite
    {
        private var shader:Shader;
        private var loader:URLLoader;

        public function InvertRGB()
        {
            init();
        }

        private function init():void
        {
            loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.BINARY;
            loader.addEventListener(Event.COMPLETE, onLoadComplete);
            loader.load(new URLRequest("invertRGB.pbj"));
        }


        private function onLoadComplete(event:Event):void
        {
            shader = new Shader(loader.data);

            var target:Shape = new Shape();
            addChild(target);

            var g:Graphics = target.graphics;
            var c:Array = [0x990000, 0x445500, 0x007799];
            var a:Array = [255, 255, 255];
            var r:Array = [0, 127, 255];
            var m:Matrix = new Matrix();
            m.createGradientBox(w, h);
            g.beginGradientFill(GradientType.LINEAR, c, a, r, m);
            g.drawRect(10, 10, w, h);
            g.endFill();

            var invertFilter:ShaderFilter = new ShaderFilter(shader);
            target.filters = [invertFilter];
        }
    }
}
```

有关应用滤镜的详细信息，请参阅第 224 页的 “ 创建和应用滤镜 ”。

在独立模式下使用着色器

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在独立模式下使用着色器时，着色器处理的运行独立于其输出结果的用途。指定要执行的着色器、设置输入值和参数值，及指定用于放置结果数据的对象。在以下两种情形中，可以考虑以独立模式使用着色器：

- 处理非图像数据：在独立模式下，您可以选择将任意二进制数据或数值数据 （而非位图图像数据）传递给着色器。除位图图像数据外，您还可以选择将着色器结果以二进制数据或数值数据的形式返回。
- 背景处理：以独立模式运行着色器时，着色器默认为异步运行。这表示，在您的应用程序继续运行的同时，着色器以后台方式运行，并在其处理结束时通知您的代码。您可以使用运行耗时的着色器，它在运行时不会导致应用程序用户界面或其他处理响应迟缓。

使用 ShaderJob 对象以独立模式执行着色器。首先创建 ShaderJob 对象，并将其链接到代表要执行的着色器的 Shader 对象：

```
var job:ShaderJob = new ShaderJob(myShader);
```

接下来，设置着色器需要的所有输入值或参数值。如果着色器在后台运行，还需要为 ShaderJob 对象的 complete 事件注册一个侦听器。着色器完成其处理时，将调用该侦听器：

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

接下来，创建着色器操作完成时，向其中写入操作结果的对象。将该对象指派给 ShaderJob 对象的 target 属性：

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

如果使用 ShaderJob 执行图像处理，则为 target 属性指派一个 BitmapData 实例。如果要处理二进制数据或数值数据，则指派 ByteArray 对象或 Vector.<Number> 实例给 target 属性。在该情形下，您必须设置 ShaderJob 对象的 width 和 height 属性，以指定输出到 target 对象的数据量。

注：您可以一步完成 ShaderJob 对象 shader、 target、 width 和 height 属性的设置，方法是将相应的参数传递给 ShaderJob() 构造函数，如：var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);

准备好执行着色器时，调用 ShaderJob 对象的 start() 方法：

```
job.start();
```

默认情况下，调用 start() 导致 ShaderJob 以异步方式执行。在这种情况下，程序立即继续执行下一行代码，而不等待着色器完成处理。着色器操作完成时， ShaderJob 对象调用其 complete 事件的侦听器，通知它们操作已完成。在这里 （即 complete 事件侦听器代码中）， target 对象获得着色器操作结果。

注：也可以不使用 target 属性对象，直接从传递给侦听器方法的事件对象处取得着色器结果。该事件对象是一个 ShaderEvent 实例。根据设置为 target 属性的对象的数据类型，ShaderEvent 对象有三个可用于访问结果的属性：ShaderEvent.bitmapData、ShaderEvent.byteArray 和 ShaderEvent.vector。

或者，可以将 true 参数传递给 start() 方法。在该情形下，着色器操作将同步执行。所有代码 （包括与用户界面及所有其他事件的交互）在着色器执行时暂停。着色器完成处理后， target 对象包含着色器结果，程序继续执行下一行代码。

```
job.start(true);
```

第 16 章：使用影片剪辑

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

MovieClip 类是在 Adobe® Flash® 开发环境中创建的动画和影片剪辑元件的核心类。它具有显示对象的所有行为和功能，还具有用于控制其时间轴的其他属性和方法。

影片剪辑基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

影片剪辑对于使用 Flash 创作工具创建动画内容并想要通过 ActionScript 来控制该内容的人来说是一个重要元素。只要在 Flash 中创建影片剪辑元件， Flash 就会将该元件添加到该 Flash 文档的库中。默认情况下，此元件会成为 MovieClip 类等具有 MovieClip 类的属性和方法。

在将某个影片剪辑元件的实例放置在舞台上时，如果该影片剪辑具有多个帧，它会自动按其时间轴进行播放，除非使用 ActionScript 更改其播放。此时间轴使 MovieClip 类与其他类区别开来，允许您在 Flash 创作工具中通过补间动画或补间形状来创建动画。相反，对于作为 Sprite 类的实例的显示对象，您只需以编程方式更改该对象的值即可创建动画。

在 ActionScript 的早期版本中， MovieClip 类是舞台上所有实例的基类。在 ActionScript 3.0 中，影片剪辑只是可以在屏幕上显示的众多显示对象中的一个。如果使用显示对象时不需要时间轴，则使用 Shape 类或 Sprite 类替代 MovieClip 类可能会提高呈示性能。有关为任务选择合适的显示对象的详细信息，请参阅第 143 页的 “ 选择 DisplayObject 子类 ”。

重要概念和术语

以下参考列表包含与影片剪辑相关的重要术语：

AVM1 SWF 使用 ActionScript 1.0 或 ActionScript 2.0 创建的 SWF 文件，通常以 Flash Player 8 或更早期版本为目标播放器。

AVM2 SWF 使用 Adobe Flash Player 9 或更高版本的 ActionScript 3.0 或者 Adobe AIR 创建的 SWF 文件。

外部 SWF 单独从项目 SWF 文件创建的 SWF 文件，将加载到项目 SWF 文件中并在该 SWF 文件中播放。

帧 时间轴上划分时间的最小单位。与运动图像电影胶片一样，每个帧都类似于动画在特定时间的快照，当快速按顺序播放各个帧时，会产生动画的效果。

时间轴 组成影片剪辑动画序列的一系列帧的比喻性表示形式。 MovieClip 对象的时间轴等同于 Flash 创作工具中的时间轴。

播放头 一个标记，用于标识在给定时刻在时间轴中所处的位置 （帧）。

使用 MovieClip 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在发布 SWF 文件时， Flash 会将舞台上的所有影片剪辑元件实例转换为 MovieClip 对象。通过在属性检查器的 “ 实例名称 ” 字段中指定影片剪辑元件的实例名称，您可以在 ActionScript 中使用该元件。在创建 SWF 文件时， Flash 会生成在舞台上创建该 MovieClip 实例的代码并使用该实例名称声明一个变量。如果您已经命名了嵌套在其他已命名影片剪辑内的影片剪辑，则会将这些子级影片剪辑视为父级影片剪辑的属性，这样您便可以使用点语法访问该子影片剪辑。例如，如果实例名称为 childClip 的影片剪辑嵌套在实例名称为 parentClip 的另一个剪辑内，则可以通过调用以下代码来播放子剪辑的时间轴动画：

```
parentClip.childClip.play();
```

注：在 Flash 创作工具中放到舞台上的子实例无法由父实例构造函数中的代码进行访问，因为在执行代码时尚未在该位置创建这些实例。在访问子实例之前，父实例必须通过代码创建子实例，或者延迟访问用于侦听子实例以调度其 Event.ADDED_TO_STAGE 事件的回调函数。

尽管 ActionScript 2.0 MovieClip 类的一些旧方法和属性仍保持不变，但其他方法和属性已发生了变化。所有前缀为下划线的属性均已被重新命名。例如，_width 和 _height 属性现在分别作为 width 和 height 被访问，而 _xscale 和 _yscale 则作为 scaleX 和 scaleY 被访问。有关 MovieClip 类的属性和方法的完整列表，请参考用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

控制影片剪辑播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash 利用时间轴来形象地表示动画或状态改变。任何使用时间轴的可视元素都必须为 MovieClip 对象或从 MovieClip 类扩展而来。尽管 ActionScript 可控制任何影片剪辑的停止、播放或转至时间轴上的另一点，但不能用于动态创建时间轴或在特定帧添加内容，这项工作仅能使用 Flash 创作工具来完成。

MovieClip 在播放时将以 SWF 文件的帧速率决定的速度沿着其时间轴推进。或者，您也可以通过在 ActionScript 中设置 Stage.frameRate 属性来覆盖此设置。

播放影片剪辑和停止播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

play() 和 stop() 方法允许对时间轴上的影片剪辑进行基本控制。例如，假设舞台上有一个影片剪辑元件，其中包含一个自行车横穿屏幕的动画，其实例名称设置为 bicycle。如果将以下代码附加到主时间轴上的关键帧，

```
bicycle.stop();
```

自行车将不会移动 （将不播放其动画）。自行车的移动可以通过其他某种用户交互来开始。例如，如果您有一个名为 startButton 的按扭，则主时间轴上某一关键帧上的以下代码会使单击该按扭时播放该动画：

```
// This function will be called when the button is clicked. It causes the
// bicycle animation to play.
function playAnimation(event:MouseEvent):void
{
    bicycle.play();
}
// Register the function as a listener with the button.
startButton.addEventListener(MouseEvent.CLICK, playAnimation);
```

快进和后退

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在影片剪辑中，play() 和 stop() 方法并非是控制播放的唯一方法。也可以使用 nextFrame() 和 prevFrame() 方法手动向前或向后沿时间轴移动播放头。调用这两种方法中的任一方法均会停止播放并分别使播放头向前或向后移动一帧。

使用 play() 方法类似于每次触发影片剪辑对象的 enterFrame 事件时调用 nextFrame()。使用此方法，您可以为 enterFrame 事件创建一个事件侦听器并在侦听器函数中让 bicycle 回到前一帧，从而使 bicycle 影片剪辑向后播放，如下所示：

```
// This function is called when the enterFrame event is triggered, meaning
// it's called once per frame.
function everyFrame(event:Event):void
{
    if (bicycle.currentFrame == 1)
    {
        bicycle.gotoAndStop(bicycle.totalFrames);
    }
    else
    {
        bicycle.prevFrame();
    }
}
bicycle.addEventListener(Event.ENTER_FRAME, everyFrame);
```

在正常播放过程中，如果影片剪辑包含多个帧，播放时将会无限循环播放，也就是说在经过最后一帧后将返回到第 1 帧。使用 prevFrame() 或 nextFrame() 时，不会自动发生此行为 （在播放头位于第 1 帧时调用 prevFrame() 不会将播放头移动到最后一帧）。以上示例中的 if 条件将检查播放头是否已返回至第一帧，并将播放头设置为处于最后一帧前面，从而有效地使影片剪辑向后持续循环播放。

跳到不同帧和使用帧标签

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

向新帧发送影片剪辑非常简单。调用 gotoAndPlay() 或 gotoAndStop() 将使影片剪辑跳到指定为参数的帧编号。或者，您可以传递一个与帧标签名称匹配的字符串。可以为时间轴上的任何帧分配一个标签。为此，选择时间轴上的某一帧，然后在属性检查器的 “ 帧标签 ” 字段中输入一个名称。

当创建复杂的影片剪辑时，使用帧标签比使用帧编号具有明显优势。当动画中的帧、图层和补间变得很多时，应考虑给重要的帧加上具有解释性说明的标签来表示影片剪辑中的行为转换 （例如，“ 离开 ”、“ 行走 ” 或 “ 跑 ”）。这可提高代码的可读性，同时使代码更加灵活，因为转到添加了标签的帧的 ActionScript 调用是指向单个引用 （即 “ 标签 ”，而不是特定帧编号）的指针。如果您以后决定将动画的特定片段移动到不同的帧，则无需更改 ActionScript 代码，只要将这些帧的相同标签保持在新位置即可。

为便于在代码中表示帧标签， ActionScript 3.0 包括了 FrameLabel 类。此类的每个实例均代表一个帧标签，并具有一个 name 属性 （表示在属性检查器中指定的帧标签的名称）和一个 frame 属性 （表示该标签在时间轴上所处帧的帧编号）。

为了访问与影片剪辑实例相关联的 FrameLabel 实例， MovieClip 类包括了两个可直接返回 FrameLabel 对象的属性。currentLabels 属性返回一个包含影片剪辑整个时间轴上所有 FrameLabel 对象的数组。 currentLabel 属性返回一个字符串，该字符串包含在时间轴上最近遇到的帧标签的名称。

假设创建了一个名为 robot 的影片剪辑并已经为其动画的各个状态加上了标签。可以设置一个用于检查 currentLabel 属性的条件以访问 robot 的当前状态，如下面的代码所示：

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Flash Player 11.3 和 AIR 3.3 向 FrameLabel 类中添加了 frameLabel 事件。您可以向表示帧标签的 FrameLabel 实例分配事件处理函数。在播放头进入帧时调度事件。

下面的示例为 MovieClip 的帧标签数组中的第二个帧标签创建 FrameLabel 实例。然后它为 frameLabel 事件注册事件处理函数：

```
var myFrameLabel:FrameLabel = robot.currentLabels[1];
myFrameLabel.addEventListener(Event.FRAME_LABEL, onFrameLabel);

function onFrameLabel(e:Event):void {
    //do something
}
```

使用场景

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 Flash 创作环境中，您可以使用场景来区分 SWF 文件播放时将要经过的一系列时间轴。使用 gotoAndPlay() 或 gotoAndStop() 方法的第二个参数，可以指定要向其发送播放头的场景。所有 FLA 文件开始时都只有初始场景，但您可以创建新的场景。

使用场景并非始终是最佳方法，因为场景有许多缺点。包含多个场景的 Flash 文档可能很难维护，尤其是在存在多个作者的环境中。多个场景也会使带宽效率降低，因为发布过程会将所有场景合并为一个时间轴。这样将使所有场景进行渐进式下载，即使从不会播放这些场景。因此，除非是组织冗长的基于多个时间轴的动画，否则通常不鼓励使用多个场景。

MovieClip 类的 scenes 属性返回表示 SWF 文件中所有场景的 Scene 对象的数组。currentScene 属性返回一个表示当前正在播放的场景的 Scene 对象。

Scene 类具有多个提供有关场景信息的属性。labels 属性返回表示该场景中帧标签的 FrameLabel 对象的数组。name 属性以字符串形式返回场景的名称。 numFrames 属性返回一个表示场景中帧的总数的整数。

使用 ActionScript 创建 MovieClip 对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 Flash 中向屏幕中添加内容的一个方法是将资源从库中拖放到舞台上，但不是仅有这一种方法。对于复杂项目，经验丰富的开发人员通常更喜欢以编程方式创建影片剪辑。这种方法具有多个优点：代码更易于重用、编译时速度加快，以及仅可在 ActionScript 中进行的更复杂的修改。

ActionScript 3.0 的显示列表 API 简化了动态创建 MovieClip 对象的过程。直接实例化 MovieClip 实例的功能从向显示列表中添加该实例的过程中分离出来，从而更加灵活、简单，而不会牺牲控制性能。

在 ActionScript 3.0 中，当以编程方式创建影片剪辑 （或任何其他显示对象）实例时，只有通过对显示对象容器调用 addChild() 或 addChildAt() 方法将该实例添加到显示列表中后，才能在屏幕上看到该实例。这允许您创建影片剪辑、设置其属性，甚至可以在向屏幕呈示该影片剪辑之前调用方法。有关使用显示列表的详细信息，请参阅第 133 页的 “ 使用显示对象容器 ”。

为 ActionScript 导出库元件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下， Flash 文档库中的影片剪辑元件实例不能以动态方式创建 （即只使用 ActionScript 创建）。这是因为导出供 ActionScript 使用的每个元件都会增加 SWF 文件的大小，而且众所周知，有些元件可能不适合在舞台上使用。因此，为了使元件可以在 ActionScript 中使用，必须指定为 ActionScript 导出该元件。

为 **ActionScript** 导出元件：

1 在 “ 库 ” 面板中选择该元件并打开其 “ 元件属性 ” 对话框。

2 必要时激活 “ 高级 ” 设置。

3 在“链接”部分中，激活“为 ActionScript 导出”复选框。

这将激活“类”和“基类”字段。

默认情况下，“类”字段会用删除了空格的元件名称进行填充（例如，名为“Tree House”的元件会变为“TreeHouse”）。若要指定该元件对其行为使用自定义类，请在此字段中输入该类的完整名称，包括它所在的包。如果希望在 ActionScript 中创建该元件的实例，但不需要添加任何其他行为，则可以使类名称保持原样。

“基类”字段的值默认为 flash.display.MovieClip。如果想让元件扩展另一个自定义类的功能，可以指定该类的名称替代这个值，只要该类扩展 Sprite（或 MovieClip）类即可。

4 按“确定”按钮以保存所做的更改。

此时，如果 Flash 无法找到链接的 SWC 文件，或包含指定类的定义的外部 ActionScript 文件（例如，如果不需要为元件添加其它行为），则会显示警告：

无法在类路径中找到对此类的定义，因此将在导出时自动在 SWF 文件中生成相应的定义。

如果库元件不需要超出 MovieClip 类功能的独特功能，则可以忽略此警告消息。

如果没有为元件提供类， Flash 将为元件创建一个等同于下面所示类的类：

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

如果想要向元件中添加额外的 ActionScript 功能，请向下面的代码结构中添加相应的属性和方法。例如，假如有一个包含 50 像素宽和 50 像素高的圆形的影片剪辑元件，并用名为 Circle 的类指定为 ActionScript 导出该元件。以下代码在放入 Circle.as 文件后将扩展 MovieClip 类，同时为此元件提供额外的方法 getArea() 和 getCircumference()：

```
package
{
    import flash.display.MovieClip;

    public class Circle extends MovieClip
    {
        public function Circle()
        {
        }

        public function getArea():Number
        {
        // The formula is Pi times the radius squared.
        return Math.PI * Math.pow((width / 2), 2);
        }

        public function getCircumference():Number
        {
        // The formula is Pi times the diameter.
        return Math.PI * width;
        }
    }
}
```

放置在 Flash 文档第 1 帧的关键帧上的以下代码将创建该元件的一个实例，并在屏幕上显示该实例：

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

此代码演示了基于 ActionScript 的实例化可作为将单个资源拖放到舞台上的替代方法。它所创建的圆形具有影片剪辑的所有属性，同时还具有 Circle 类中定义的自定义方法。这是一个非常简单的示例 — 您的库元件可在其类中指定任意数目的属性和方法。

基于 ActionScript 的实例化功能强大，因为允许动态创建大量实例，而如果采用手动方式来创建将是一项繁重的任务。同时还很灵活，因为您可以在创建每个实例时自定义该实例的属性。您可以通过使用循环动态创建多个 Circle 实例来体会上述优点。在 Flash 文档库中存在上述 Circle 元件和类的情况下，将下面的代码放在第 1 帧的关键帧上：

```
import flash.geom.ColorTransform;

var totalCircles:uint = 10;
var i:uint;
for (i = 0; i < totalCircles; i++)
{
    // Create a new Circle instance.
    var c:Circle = new Circle();
    // Place the new Circle at an x coordinate that will space the circles
    // evenly across the Stage.
    c.x = (stage.stageWidth / totalCircles) * i;
    // Place the Circle instance at the vertical center of the Stage.
    c.y = stage.stageHeight / 2;
    // Change the Circle instance to a random color
    c.transform.colorTransform = getRandomColor();
    // Add the Circle instance to the current timeline.
    addChild(c);
}

function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

此代码演示了如何使用代码快速创建和自定义元件的多个实例。每个实例都根据循环内的当前计数进行定位，并且每个实例都通过设置 transform 属性 （Circle 通过扩展 MovieClip 类而继承该属性）获得了一种随机颜色。

加载外部 SWF 文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中，SWF 文件是使用 Loader 类来加载的。若要加载外部 SWF 文件，ActionScript 需要执行以下 4 个操作：

1 用文件的 URL 创建一个新的 URLRequest 对象。

2 创建一个新的 Loader 对象。

3 调用 Loader 对象的 load() 方法，并以参数形式传递 URLRequest 实例。

4 对显示对象容器 （如 Flash 文档的主时间轴）调用 addChild() 方法，将 Loader 实例添加到显示列表中。

最后，代码如下所示：

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

通过指定图像文件的 URL 而不是 SWF 文件的 URL，可以使用上述同样的代码加载外部图像文件，如 JPEG、 GIF 或 PNG 图像。 SWF 文件不同于图像文件，可能包含 ActionScript。因此，虽然加载 SWF 文件的过程可能与加载图像的过程完全相同，但如果 Flash Player 或 AIR 在播放 SWF，并且您计划使用 ActionScript 以某种方式与外部 SWF 文件通信，则在加载该外部 SWF 文件时，执行加载的 SWF 文件和要加载的 SWF 文件必须位于同一个安全沙箱中。另外，如果外部 SWF 文件包含了与执行加载的 SWF 文件中的类共享同一命名空间的类，可能需要为被加载的 SWF 文件创建新的应用程序域才能避免命名空间冲突。有关安全性和应用程序域注意事项的详细信息，请参阅第 123 页的 “ 使用应用程序域 ”和第 908 页的 “ 加载内容 ”。

当成功加载外部 SWF 文件后，可通过 Loader.content 属性访问该文件。如果该外部 SWF 文件是针对 ActionScript 3.0 发布的，则加载的文件将为影片剪辑或 sprite，具体取决于所扩展的类。

与其他平台相比，在 Adobe AIR for iOS 中加载 SWF 文件存在一些不同之处。有关详细信息，请参阅第 167 页的 “ 在 AIR for iOS 中加载 SWF 文件 ”。

加载早期 SWF 文件的注意事项

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果已使用早期版本的 ActionScript 发布了外部 SWF 文件，则需要考虑一些重要的限制条件。与在 AVM2 (ActionScript Virtual Machine 2) 中运行的 ActionScript 3.0 SWF 文件不同，针对 ActionScript 1.0 或 2.0 发布的 SWF 文件在 AVM1 (ActionScript Virtual Machine 1) 中运行。

将 ActionScript 1.0 或 2.0 SWF 文件加载到 ActionScript 3.0 SWF 文件时，（与加载 ActionScript 3.0 SWF 文件相比）有重要区别。 Flash Player 提供与以前发布的内容的完全后向兼容性。在以前版本的 Flash Player 中运行的任何内容可在支持 ActionScript 3.0 的 Flash Player 版本中运行。但是，存在以下限制：

- ActionScript 3.0 代码可以加载使用 ActionScript 1.0 或 2.0 编写的 SWF 文件。如果 ActionScript 1.0 或 2.0 SWF 文件成功加载，加载的对象 （Loader.content 属性）是 AVM1Movie 对象。 AVM1Movie 实例不同于 MovieClip 实例。而是显示对象，但不同于影片剪辑，它不包括与时间轴相关的方法或属性。父 AVM2 SWF 文件无法访问加载的 AVM1Movie 对象的属性、方法或对象。
- 以 ActionScript 1.0 或 2.0 编写的 SWF 文件无法加载以 ActionScript 3.0 编写的 SWF 文件。这意味着，在 Flash 8 或 Flex Builder 1.5 或更早版本中创作的 SWF 文件无法加载 ActionScript 3.0 SWF 文件。

  此规则的唯一例外情况是，只要 ActionScript 2.0 SWF 文件以前没有向它的任何级别加载任何内容， ActionScript 2.0 SWF 文件就可以用 ActionScript 3.0 SWF 文件来替换它自身。ActionScript 2.0 SWF 文件可通过调用 loadMovieNum() 并将值 0 传递给 level 参数来实现此目的。
- 通常，如果使用 ActionScript 1.0 或 2.0 编写的 SWF 文件要与使用 ActionScript 3.0 编写的 SWF 文件一起使用，则必须迁移前者。例如，假设您使用 ActionScript 2.0 创建了一个媒体播放器。此媒体播放器加载的也是使用 ActionScript 2.0 创建的多种内容。您无法使用 ActionScript 3.0 创建新内容并在此媒体播放器中加载新内容。您必须将视频播放器迁移到 ActionScript 3.0。

  但是，如果您在 ActionScript 3.0 中创建一个媒体播放器，则该媒体播放器可以执行 ActionScript 2.0 内容的简单加载。

下面的表总结了以前版本的 Flash Player 在加载较新内容和执行代码方面的限制，以及在使用不同版本的 ActionScript 编写的 SWF 文件之间进行跨脚本访问的限制。

支持的功能	Flash Player 7	Flash Player 8	Flash Player 9 和 10
可以加载针对以下版本发布的 SWF	7 和更早版本	8 和更早版本	9 （或 10）及更低版本
包含此 AVM	AVM1	AVM1	AVM1 和 AVM2
运行在以下 ActionScript 版本中编写的 SWF	1.0 和 2.0	1.0 和 2.0	1.0、 2.0 和 3.0

在下表中，“ 支持的功能 ”指在 Flash Player 9 或更高版本中运行的内容。运行在 Flash Player 8 或更早版本中的内容只能在 ActionScript 1.0 和 2.0 中加载、显示、执行以及跨脚本编写。

支持的功能	以 ActionScript 1.0 和 2.0 创建的内容	以 ActionScript 3.0 创建的内容
可以加载在以下版本中创建的内容并在其中执行代码	仅 ActionScript 1.0 和 2.0	ActionScript 1.0、 2.0 和 ActionScript 3.0
可以对在以下版本中创建的内容进行跨脚本编写	仅 ActionScript 1.0 和 2.0 （通过本地连接的 ActionScript 3.0）	ActionScript 1.0 和 2.0 （通过本地连接）。 ActionScript 3.0

影片剪辑示例：RuntimeAssetsExplorer

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

“ 为 ActionScript 导出 ”功能非常适合用于在多个项目间使用库的情况。如果 Flash Player 或 AIR 执行 SWF 文件，则任何 SWF 文件只要与加载它的 SWF 处于同一安全沙箱中就可以使用已导出到 ActionScript 的元件。这样，单个 Flash 文档可生成仅用于保存图形资源的 SWF 文件。此技术对大型项目特别有用，在这样的项目中，处理可视资源的设计人员可与创建 “ 包装 ”SWF 文件的开发人员同时工作，然后在运行时加载图形资源 SWF 文件。您可以使用此方法维护一系列图形资源与编程开发进度无关的版本文件。

RuntimeAssetsExplorer 应用程序加载属于 RuntimeAsset 子类的任何 SWF 文件，并允许您浏览该 SWF 文件的可用资源。示例说明了以下过程：

- 使用 Loader.load() 加载外部 SWF 文件
- 动态创建为 ActionScript 导出的库元件
- 使用 ActionScript 控制 MovieClip 播放

开始之前，注意每个要在 Flash Player 中运行的 SWF 文件必须位于同一个安全沙箱中。有关详细信息，请参阅第 895 页的 “ 安全沙箱 ”。

若要获取此范例的应用程序文件，请下载 Flash Professional 范例。 RuntimeAssetsExplorer 应用程序文件位于文件夹 Samples/RuntimeAssetsExplorer 中。该应用程序包含以下文件：

文件	说明
RuntimeAssetsExample.mxml 或 RuntimeAssetsExample.fla	适用于 Flex (MXML) 或 Flash (FLA) 的应用程序的用户界面。
RuntimeAssetsExample.as	Flash (FLA) 应用程序的文档类。
GeometricAssets.as	用于实现 RuntimeAsset 接口的示例类。
GeometricAssets.fla	链接到 GeometricAssets 类的 FLA 文件 （该类是 FLA 的文档类）包含为 ActionScript 导出的元件。

文件	说明
com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as	用于定义将加载到浏览器容器中的所有运行时资源 SWF 文件所需方法的接口。
com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as	形状为旋转方框的库元件的类。
com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as	形状为旋转星形的库元件的类。

建立运行时库界面

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要使浏览器与 SWF 库正确交互，运行时资源库的结构就必须具有一定程式。我们将通过创建一个接口来完成此过程 （接口类似于类，因为接口也是用于区分预期结构的方法的蓝图；同时接口又与类不同，因为接口不包含方法体）。接口提供了在运行时库和浏览器之间相互通信的方法。加载到浏览器中的运行时资源的每个 SWF 都将实现此接口。有关接口及其使用方法的更多信息，请参阅 《学习 ActionScript 3.0》中的 " 接口 "。

RuntimeLibrary 接口非常简单 — 我们只需要一个函数，这个函数能够为浏览器提供类路径的数组以导出元件并使元件在运行时库中可用。为此，该接口具有单个方法：getAssets()。

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

创建资源库 SWF 文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过定义 RuntimeLibrary 接口，可以创建能够加载到另一个 SWF 文件中的多个资源库 SWF 文件。制作资源的单个 SWF 库包括四个任务：

- 为资源库 SWF 文件创建一个类
- 为库中包含的单个资源创建类
- 创建实际图形资源
- 将图形元素与类关联并发布库 SWF

创建一个类以实现 **RuntimeLibrary** 接口

下一步，我们将创建要实现 RuntimeLibrary 接口的 GeometricAssets 类。这个类将成为 FLA 的文档类。此类的代码与 RuntimeLibrary 接口非常相似，不同之处在于，在类定义中， getAssets() 方法有方法体。

```
package
{
    import flash.display.Sprite;
    import com.example.programmingas3.runtimeassetexplorer.RuntimeLibrary;

    public class GeometricAssets extends Sprite implements RuntimeLibrary
    {
        public function GeometricAssets() {

        }
        public function getAssets():Array {
            return [ "com.example.programmingas3.runtimeassetexplorer.AnimatingBox",
                     "com.example.programmingas3.runtimeassetexplorer.AnimatingStar" ];
        }
    }
}
```

如果要创建第二个运行时库，可以另外创建一个基于另一个类 （例如 AnimationAssets）的 FLA，该类可提供自带的 getAssets() 实现。

为每个 **MovieClip** 资源创建类

对于本示例，我们只扩展 MovieClip 类而不为自定义资源添加任何功能。以下 AnimatingStar 的代码类似于 AnimatingBox 的代码：

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

发布库

现在将基于 MovieClip 的资源连接到新类，方法是创建一个新的 FLA 并在 “ 属性 ” 检查器的 “ 文档类 ” 字段中输入 GeometricAssets。为实现本示例的目的，我们将创建两个使用时间轴补间的非常简单的形状，其中一个形状顺时针旋转超过 360 帧。 animatingBox 和 animatingStar 元件都设为 “ 为 ActionScript 导出 ”，并将 “ 类 ” 字段设置为 getAssets() 实现中指定的对应类路径。保留 flash.display.MovieClip 的默认基类，因为我们希望对标准 MovieClip 方法进行子分类。

在设置了元件的导出设置后，可发布 FLA。您现在便拥有了第一个运行时库。该 SWF 文件可以加载到另一个 AVM2 SWF 文件中，且 AnimatingBox 和 AnimatingStar 元件可用于新的 SWF 文件。

将库加载到另一个 SWF 文件中

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要处理的最后一个功能性部分是资源浏览器的用户界面。在本示例中，运行时库的路径硬编码为一个名为 ASSETS_PATH 的变量。或者，您也可以使用 FileReference 类，例如，用来创建在硬盘驱动器上浏览特定 SWF 文件的接口。

成功加载运行时库后， Flash Player 会调用 runtimeAssetsLoadComplete() 方法：

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

在此方法中，变量 rl 表示已加载的 SWF 文件。代码将调用已加载的 SWF 文件的 getAssets() 方法，获取可用资源的列表，并通过调用 populateDropDown() 方法用这些资源填充具有可用资源列表的 ComboBox 组件。该方法会依次存储每个资源的完整类路径。单击用户界面上的 “ 添加 ” 按扭即会触发 addAsset() 方法：

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

此方法获取 ComboBox 中当前所选资源的类路径 (assetNameCbo.selectedItem.data)，并使用 getDefinitionByName() 函数（来自 flash.utils 包）获取对该资源的类的实际引用，以创建该资源的新实例。

第 17 章：使用补间动画

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

第 160 页的 " 对象动画 " 介绍如何在 ActionScript 中实现脚本动画。

我们在这里介绍另一种创建动画的技术：补间动画。使用此技术，您可以通过使用 Adobe® Flash® Professional 在文档中交互设置动画来创建移动。然后您可以在运行时将该动画用于基于 ActionScript 的动态动画。

Flash Professional 将自动生成实现补间动画的 ActionScript，并使其具有可复制性和重复使用性。

若要创建补间动画，您必须拥有 Flash Professional 的许可证。

更多帮助主题

fl.motion 包

补间动画基础知识

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

补间动画提供了创建动画的简单方法。

补间动画以帧到帧的方式修改显示对象属性 （如位置或旋转）。在显示对象移动期间，补间动画还可以通过应用各种滤镜和其他属性来更改显示对象的外观。使用 Flash Professional 交互创建补间动画，会生成补间动画的 ActionScript。在 Flash 中，可使用 " 将动画复制为 ActionScript 3.0 脚本 " 命令复制创建补间动画的 ActionScript。随后您便可以在您自己的动态动画中重复使用该 ActionScript，以便在运行时创建移动效果。

有关创建补间动画的信息，请参阅 《使用 Flash Professional》中的 " 补间动画 " 一节。

重要概念和术语

以下是一个与此功能相关的重要术语：

补间动画 一种构造，它生成显示对象在不同时间不同状态下的中间帧；提供使第一个状态平滑过渡到第二个状态的外观。用于在舞台上移动显示对象，并使显示对象随时间而增大、缩小、旋转、淡化或更改颜色。

在 Flash 中复制补间动画脚本

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

补间生成的中间帧可显示处于时间轴上两个不同帧中的不同状态下的显示对象。它创建的外观可使第一个帧中的图像平滑过渡到第二个帧中的图像。在补间动画中，外观的更改通常涉及显示对象位置的更改，因而可创建移动效果。除了调整显示对象的位置之外，补间动画还可以对显示对象进行旋转、倾斜、调整大小或应用滤镜。

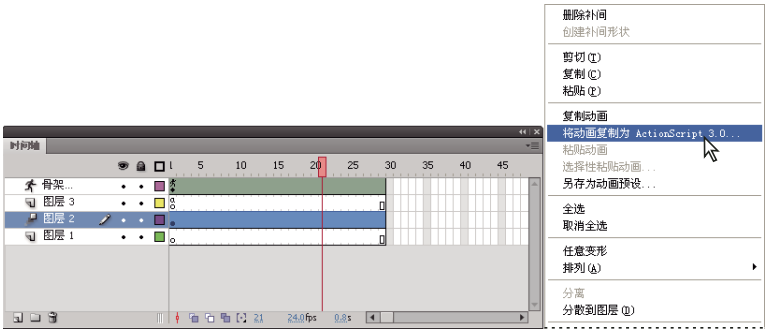
在 Flash 中，通过沿时间轴在两个关键帧之间移动显示对象来创建补间动画。 Flash 可自动生成描述补间的 ActionScript 代码，您可以复制这些代码并将其保存在文件中。有关创建补间动画的信息，请参阅 《使用 Flash Professional》中的 " 补间动画 " 一节。

您可以通过两种方式访问 Flash 中的 “ 将动画复制为 ActionScript 3.0 脚本 ” 命令。第一种方式是通过舞台上的补间上下文菜单：

1 在舞台上选择补间动画。

2 右键单击 (Windows) 或按住 Control 单击 (Macintosh)。

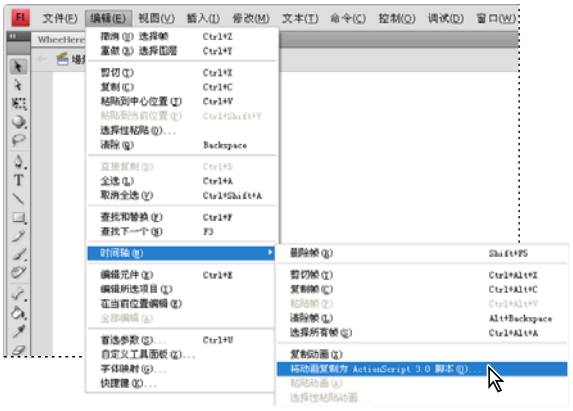
3 选择 “ 将动画复制为 ActionScript 3.0 脚本 . . .”



第二种方式是直接从 Flash“ 编辑 ” 菜单选择该命令：

1 在舞台上选择补间动画。

2 选择 “ 编辑 ”>“ 时间轴 ”>“ 将动画复制为 ActionScript 3.0 脚本 ”。



在复制了脚本后，将脚本粘贴到文件中并保存。

在创建某个补间动画并复制和保存脚本后，您可以按原样重复使用该脚本，也可以在您自己的基于 ActionScript 的动态动画中修改该脚本。

合并补间动画脚本

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

您从 Flash 复制的 ActionScript 代码中的标头会列出支持补间动画所需的所有模块。

补间动画类

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

基本的补间动画类是 fl.motion 包中的 AnimatorFactory 类、MotionBase 类和 Motion 类。根据补间动画所操作的属性，您可能还需要其他类。例如，如果补间动画对显示对象进行变形或旋转，则需要导入相应的 flash.geom 类。如果补间动画应用了滤镜，则需导入 flash.filter 类。在 ActionScript 中，补间动画是 Motion 类的实例。 Motion 类存储可应用于可视对象的关键帧动画序列。动画数据包括位置、缩放、旋转、倾斜、颜色、滤镜和缓动。

下面的 ActionScript 的复制源是在 Flash 中创建的补间动画，用于对实例名称为 Symbol1_2 的显示对象进行动画处理。这段代码为一个名为 __motion_Symbol1_2 的 MotionBase 对象声明了一个变量。 MotionBase 类是 Motion 类的父级。

```
var __motion_Symbol1_2:MotionBase;
```

随后脚本创建 Motion 对象：

```
__motion_Symbol1_2 = new Motion();
```

Motion 对象名称

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

上例中，Flash 自动为 Motion 对象生成名称 __motion_Symbol1_2。它将前缀 __motion_ 附加到显示对象名称。因此，自动生成的名称基于 Flash 中补间动画的目标对象的实例名称。 Motion 对象的 duration 属性指示补间动画中的总帧数：

```
__motion_Symbol1_2.duration = 200;
```

默认情况下，如果复制的补间动画所属的显示对象实例还没有实例名称，则 Flash 会自动对该实例进行命名。

当您在自己的动画中重用 Flash 创建的 ActionScript 时，可以保留 Flash 自动为补间生成的名称，也可以替换为其他名称。如果您更改了补间名称，请确保在整个脚本中都更改该名称。

或者，您可以在 Flash 中将您选择的名称分配给补间动画的目标对象。然后创建补间动画并复制脚本。无论您使用哪种命名方式，都要确保您的 ActionScript 代码中的每个 Motion 对象都有唯一的名称。

描述动画

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

MotionBase 类的 addPropertyArray() 方法添加用于描述每个补间属性的值数组。

该数组可能包含用于补间动画中每个关键帧的数组项。通常，这些数组中的一些数组包含的项数少于补间动画中的关键帧总数。当数组中的最后一个值对于剩下的帧没有更改时，会发生这种情况。

如果数组参数的长度大于 Motion 对象的 duration 属性，则 addPropertyArray() 会相应地调整 duration 属性的值。它不会为以前添加的属性添加关键帧。会为动画的额外帧保持新添加的关键帧。

Motion 对象的 x 和 y 属性描述补间对象在动画运行时不断改变的位置。如果显示对象的位置发生更改，则这些坐标是在每个关键帧中最可能发生更改的值。您可以使用 addPropertyArray() 方法添加其他动画属性。例如，如果调整补间对象的大小，可以添加 scaleX 和 scaleY 值。如果倾斜补间对象，则添加 scewX 和 skewY 值。如果旋转补间对象，则添加 rotationConcat 属性。

使用 addPropertyArray() 方法可定义以下补间属性：

x	对象的变形点在其父级的坐标空间中的水平位置
y	对象的变形点在其父级的坐标空间中的垂直位置
z	对象的变形点在其父级的坐标空间中的深度 （z 轴）位置
scaleX	从变形点开始应用的对象的水平缩放比例 （百分比）
scaleY	从变形点开始应用的对象的垂直缩放比例 （百分比）
skewX	从变形点开始应用的对象的水平倾斜角度 （以度为单位）
skewY	从变形点开始应用的对象的垂直倾斜角度 （以度为单位）
rotationX	对象相对于其原始方向围绕 x 轴的旋转
rotationY	对象相对于其原始方向围绕 y 轴的旋转
rotationConcat	动画中的对象的旋转 （z 轴）值，相对于前一个方向且从变形点开始应用
useRotationConcat	如果设置此属性，则会在 addPropertyArray() 提供动画数据时导致目标对象旋转
blendMode	BlendMode 类值，指定对象的颜色与底层图形的混合
matrix3D	matrix3D 属性 （如果对于关键帧存在此属性）；用于 3D 补间；如果使用，则会忽略以前所有的变形属性
rotationZ	对象相对于 3D 父容器从其原始方向开始的 z 轴旋转 （以度为单位）；用于 3D 补间 （而不是 rotationConcat）

在自动生成的脚本中添加的属性取决于在 Flash 中分配给补间动画的属性。在自定义自己的脚本版本时，您可以添加、删除或修改其中一些属性。

下面的代码向名为 __motion_Wheel 的补间动画的属性赋值。在此示例中，补间显示对象在补间动画中的全部 29 个帧中都不更改位置，而是在当前位置上旋转。分配给 rotationConcat 数组的多个值对旋转进行了定义。此补间动画的其他属性值无变化。

```
__motion_Wheel = new Motion();
__motion_Wheel.duration = 29;
__motion_Wheel.addPropertyArray("x", [0]);
__motion_Wheel.addPropertyArray("y", [0]);
__motion_Wheel.addPropertyArray("scaleX", [1.00]);
__motion_Wheel.addPropertyArray("scaleY", [1.00]);
__motion_Wheel.addPropertyArray("skewX", [0]);
__motion_Wheel.addPropertyArray("skewY", [0]);
__motion_Wheel.addPropertyArray("rotationConcat",
    [
        0,-13.2143,-26.4285,-39.6428,-52.8571,-66.0714,-79.2857,-92.4999,-105.714,
        -118.929,-132.143,-145.357,-158.571,-171.786,-185,-198.214,-211.429,-224.643,
        -237.857,-251.071,-264.286,-277.5,-290.714,-303.929,-317.143,-330.357,
        -343.571,-356.786,-370
    ]
);
__motion_Wheel.addPropertyArray("blendMode", ["normal"]);
```

在下一个示例中，名为 Leaf_1 的显示对象会在舞台上移动。该对象的 x 和 y 属性数组为动画的 100 个帧中的每个帧包含不同的值。此外，该对象在舞台上移动时同时在其 z 轴上旋转。 rotationZ 属性数组中的多个项确定了旋转方式。

```
__motion_Leaf_1 = new MotionBase();
__motion_Leaf_1.duration = 100;
__motion_Symbol1_4.addPropertyArray("y",
    [
        0,5.91999,11.84,17.76,23.68,29.6,35.52,41.44,47.36,53.28,59.2,65.12,71.04,
        76.96,82.88,88.8,94.72,100.64,106.56,112.48,118.4,124.32,130.24,136.16,142.08,
        148,150.455,152.909,155.364,157.818,160.273,162.727,165.182,167.636,170.091,
        172.545,175,177.455,179.909,182.364,184.818,187.273,189.727,192.182,194.636,
        197.091,199.545,202,207.433,212.865,218.298,223.73,229.163,234.596,240.028,
        245.461,250.893,256.326,261.759,267.191,272.624,278.057,283.489,
        288.922,294.354,299.787,305.22,310.652,316.085,321.517,326.95,330.475,334,
        337.525,341.05,344.575,348.1,351.625,355.15,358.675,362.2,365.725,369.25,
        372.775,376.3,379.825,383.35,386.875,390.4,393.925,397.45,400.975,404.5,
        407.5,410.5,413.5,416.5,419.5,422.5,425.5
    ]
);
__motion_Symbol1_4.addPropertyArray("scaleX", [1.00]);
__motion_Symbol1_4.addPropertyArray("scaleY", [1.00]);
__motion_Symbol1_4.addPropertyArray("skewX", [0]);
__motion_Symbol1_4.addPropertyArray("skewY", [0]);
__motion_Symbol1_4.addPropertyArray("z",
    [
        0,0,0,
```

初始化滤镜数组

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

initFilters() 方法将初始化滤镜。该方法的第一个参数是应用于显示对象的所有滤镜的完全限定类名称的数组。此滤镜名称数组是从 Flash 中的补间动画滤镜列表生成的。在您的脚本副本中，您可以在此数组中删除或添加 flash.filters 包中的任何滤镜。下面的调用初始化目标显示对象的滤镜列表。该调用应用 DropShadowFilter、GlowFilter 和 BevelFilter，并将列表复制到 Motion 对象中的每个关键帧。

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

添加滤镜

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

addFilterPropertyArray() 方法使用以下参数描述已初始化滤镜的属性：

1. 该方法的第一个参数通过索引标识滤镜。该索引指示滤镜名称在滤镜类名称数组中的位置，该数组是在前一个 initFilters() 调用中传递的。
2. 该方法的第二个参数是在每个关键帧中为该滤镜存储的滤镜属性。
3. 该方法的第三个参数是指定滤镜属性的值。

在进行了前一个 initFilters() 调用后，随后的 addFilterPropertyArray() 调用将值 5 分配给 DropShadowFilter 的 blurX 和 blurY 属性。 DropShadowFilter 是已初始化的滤镜数组中的第一个 （索引为 0）项：

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

接下来的三个调用向已初始化滤镜数组中的第二个项 （索引为 1） GlowFilter 的 quality、 alpha 和 color 属性赋值。

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

接下来的四个调用向已初始化滤镜数组的第三个项（索引为 2）BevelFilter 的 shadowAlpha、shadowColor、highlightAlpha 和 highlightColor 赋值。

```
__motion_Box.addFilterPropertyArray(2, "shadowAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "shadowColor", [0x000000]);
__motion_Box.addFilterPropertyArray(2, "highlightAlpha", [1.00]);
__motion_Box.addFilterPropertyArray(2, "highlightColor", [0xffffff]);
```

使用 ColorMatrixFilter 调整颜色

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

在初始化 ColorMatrixFilter 之后，您可以设置相应的 AdjustColor 属性来调整补间显示对象的亮度、对比度、饱和度和色相。通常，在 Flash 中创建补间动画时，会应用 AdjustColor 滤镜；您可以在 ActionScript 的副本中对其进行微调。下面的示例在显示对象移动期间转换该对象的色相和饱和度。

```
__motion_Leaf_1.initFilters(["flash.filters.ColorMatrix"], [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorBrightness", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorContrast", [0], -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorSaturation",
    [
        0,-0.589039,1.17808,-1.76712,-2.35616,-2.9452,-3.53424,-4.12328,
        -4.71232,-5.30136,-5.89041, 6.47945,-7.06849,-7.65753,-8.24657,
        -8.83561,-9.42465,-10.0137,-10.6027,-11.1918,11.7808,-12.3699,
        -12.9589,-13.5479,-14.137,-14.726,-15.3151,-15.9041,-16.4931,
        17.0822,-17.6712,-18.2603,-18.8493,-19.4383,-20.0274,-20.6164,
        -21.2055,-21.7945,22.3836,-22.9726,-23.5616,-24.1507,-24.7397,
        -25.3288,-25.9178,-26.5068,-27.0959,27.6849,-28.274,-28.863,-29.452,
        -30.0411,-30.6301,-31.2192,-31.8082,-32.3973,32.9863,-33.5753,
        -34.1644,-34.7534,-35.3425,-35.9315,-36.5205,-37.1096,-37.6986,
        38.2877,-38.8767,-39.4657,-40.0548,-40.6438,-41.2329,-41.8219,
        -42.411,-43
    ],
    -1, -1);
__motion_Leaf_1.addFilterPropertyArray(0, "adjustColorHue",
    [
        0,0.677418,1.35484,2.03226,2.70967,3.38709,4.06451,4.74193,5.41935,
        6.09677,6.77419,7.45161,8.12903,8.80645,9.48387,10.1613,10.8387,11.5161,
        12.1935,12.871,13.5484,14.2258,14.9032,15.5806,16.2581,16.9355,17.6129,
        18.2903,18.9677,19.6452,20.3226,21,22.4286,23.8571,25.2857,26.7143,28.1429,
        29.5714,31,32.4286,33.8571,35.2857,36.7143,38.1429,39.5714,41,42.4286,43.8571,
        45.2857,46.7143,48.1429,49.5714,51,54,57,60,63,66,69,72,75,78,81,84,87,
        90,93,96,99,102,105,108,111,114
    ],
    -1, -1);
```

将补间动画与其显示对象关联

Flash Player 9 及更高版本， **Adobe AIR 1.0** 及更高版本，需要 **Flash CS3** 或更高版本

最后一项任务是将补间动画与它所操作的一个或多个显示对象关联起来。

AnimatorFactory 类管理补间动画与其目标显示对象之间的关联。 AnimatorFactory 构造函数的参数为 Motion 对象：

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

使用 AnimatorFactory 类的 addTarget() 方法可将目标显示对象与其补间动画关联起来。从 Flash 复制的 ActionScript 注释掉了 addTarget() 代码行，并且未指定实例名称：

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

在您的副本中，指定要与补间动画关联的显示对象。在下面的示例中，目标指定为 greenWheel 和 redWheel：

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

您可以使用多个 addTarget() 调用，将多个显示对象与同一个补间动画关联起来。

第 18 章：使用反向运动

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

反向运动 (IK) 是一种用于创建逼真运动的重要方法。

使用 IK 可以在一系列连接的部分 （称为 IK 骨架）中创建协调运动，以便各部分以逼真的方式一起移动。骨架的各部分是其骨骼和连接。如果给定骨架的终点， IK 便可计算出达到该终点所需的连接的角度。

自己手动计算这些角度难度很大。此功能的优点在于您可以使用 Adobe® Flash® Professional 以交互方式创建骨架。然后可以使用 ActionScript 对骨架进行动画处理。 Flash Professional 中附带的 IK 引擎可进行计算，以描述骨架的移动。您可以在 ActionScript 代码中使用某些参数限制移动。

此 Flash Professional CS5 版本的 IK 新增了骨骼弹簧概念，通常与高端动画应用程序相关联。与新增动态物理引擎配合使用，此功能可将动作配置得栩栩如生。并且，无论在运行时还是创作期间都可达到这种效果。

若要创建反向运动骨架，您必须拥有 Flash Professional 的许可证。

更多帮助主题

fl.ik 包

反向运动的基础知识

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

使用反向运动 (IK) 可以将不同的部分链接起来，使它们以逼真的方式进行相对移动，从而创建逼真的动画效果。

例如，使用 IK 时，您可以根据需要移动某条腿的连接将那条腿移动到特定的位置，从而获得所需的姿势。 IK 使用一种骨骼框架，这些骨骼采用称为 “IK 骨架 ” 的结构连接在一起。fl.ik 包可帮助您创建模仿自然运动的动画。使用该包可以对多个 IK 骨架进行无缝动画处理，而不必了解很多 IK 算法所依赖的物理知识。

使用 Flash Professional 创建 IK 骨架及其辅助骨骼和连接。然后您便可以在运行时访问 IK 类以对它们进行动画处理。

有关如何创建 IK 骨架的详细介绍，请参阅 《使用 Flash Professional》中的 “ 使用反向运动 ” 一节。

重要概念和术语
以下参考列表中包含与此功能相关的重要术语：

骨架 一种由骨骼和联结点组成的运动链，用于在计算机动画中模拟真实运动。

骨骼 骨架中的刚性段，类似于动物骨架中的骨骼。

反向运动 (IK) 确定连接的灵活对象 （称为运动链或骨架）的参数的过程。

联结点 两块骨骼接合的位置，构建在一起从而使骨骼可以移动；类似于动物的关节。

物理引擎 与物理相关联的算法包，用于为动画提供逼真的动作。

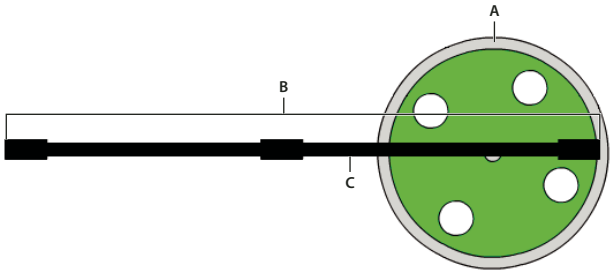
弹簧 当父骨骼移动时移动并响应、并随时间变化以增量方式下降的骨骼质量。

IK 骨架动画处理概述

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

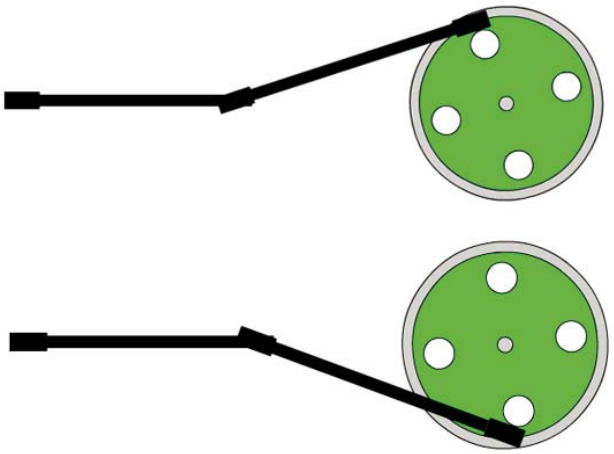
在 Flash Professional 中创建 IK 骨架后，可使用 fl.ik 类在运行时限制骨架的移动、跟踪其事件并对其进行动画处理。

下图演示了一个名为 Wheel 的影片剪辑。轴是一个名为 Axle 的 IKArmature 实例。IKMover 类会在轮子旋转时同步移动骨架。骨架中的 IKBone (ikBone2) 在其尾部连接处附加到轮子。



A. Wheel **B.** Axle **C.** ikBone2

在运行时，轮子结合 __motion_Wheel 补间动画 （在第 283 页的 " 描述动画 " 中进行了介绍）进行旋转。由一个 IKMover 对象启动和控制轴的移动。下图演示了附加到旋转轮上的轴骨架在旋转中的不同帧上的两个快照。



在运行时，下面的 ActionScript 可完成以下功能：

- 获取有关骨架及其组件的信息
- 实例化 IKMover 对象
- 结合轮子的旋转移动轴

```
import fl.ik.*

var tree:IKArmature = IKManager.getArmatureByName("Axle");
var bone:IKBone = tree.getBoneByName("ikBone2");
var endEffector:IKJoint = bone.tailJoint;
var pos:Point = endEffector.position;

var ik:IKMover = new IKMover(endEffector, pos);
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;

Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90, 0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

用于移动轴的 IK 类为：

- IKArmature：描述骨架 （由骨骼和连接组成的树结构）；必须使用 Flash 创建 Professional
- IKManager：文档中所有 IK 骨架的容器类；必须使用 Flash Professional 创建
- IKBone：IK 骨架的一段
- IKJoint：两个 IK 骨骼之间的连接
- IKMover：启动和控制骨架的 IK 移动

有关这些类的完整和详细描述，请参阅 ik 包。

获取有关 IK 骨架的信息

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

首先，为组成要移动的各部分的骨架、骨骼和连接声明变量。

下面的代码使用 IKManager 类的 getArmatureByName() 方法将 Axle 骨架的值分配给 IKArmature 变量 tree。Axle 骨架是先前使用 Flash Professional 创建的。

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

同样，下面的代码使用 IKArmature 类的 getBoneByName() 方法将 ikBone2 骨骼的值分配给 IKBone 变量。

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

ikBone2 骨骼的尾部连接是附加到旋转轮的骨架部分。

下行代码声明变量 endEffector 并将 ikBone2 骨骼的 tailjoint 属性分配给该变量：

```
var endEffector:IKJoint = home.tailjoint;
```

变量 pos 是用于存储 endEffector 连接的当前位置的点。

```
var pos:Point = endEffector.position;
```

在此示例中， pos 是处于轴尾部 （轴在此处与轮子连接）的连接的位置。此变量的原始值是通过 IKJoint 的 position 属性获取的。

实例化 IKMover 并限制其移动

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

由 IKMover 类的实例移动轴。

下行代码实例化 IKMover 对象 ik，将要移动的元素和移动的起始点传递给该对象的构造函数：

```
var ik:IKMover = new IKMover(endEffector, pos);
```

使用 IKMover 类的属性可以限制骨架的移动。可以基于移动的距离、迭代和时间来限制移动。

以下几对属性可强制执行这些限制。这几对属性包括一个用于指示是否限制移动的布尔值和一个用于指定限制量的整数：

布尔属性	整数属性	限制设置
limitByDistance:Boolean	distanceLimit:int	设置 IK 引擎对每次迭代移动的最大距离 （以像素为单位）。
limitByIteration:Boolean	iterationLimit:int	设置 IK 引擎对每个移动执行的最大迭代次数。
limitByTime:Boolean	timeLimit:int	设置分配给 IK 引擎用于执行移动的最长时间 （以毫秒为单位）。

默认情况下，所有布尔值都设置为 false，因此除非您明确地将布尔值设置为 true，否则移动不会受限制。若要强制执行限制，可将适当的属性设置为 true，然后为相应的整数属性指定一个值。如果您将限制设置为某个值，却没有设置其对应的 Boolean 属性，则会忽略该限制。在这种情况下， IK 引擎继续移动对象，直至达到其他限制或 IKMover 的目标位置。

在下面的示例中，骨架移动的最大距离设置为每次迭代 0.1 个像素。每个移动的最大迭代次数设置为 10。

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

移动 IK 骨架

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

IKMover 可在轮子的事件侦听器中移动轴。在轮子的每个 enterFrame 事件中，都会计算骨架的新目标位置。 IKMover 使用其 moveTo() 方法将尾部连接移动到其目标位置，或是在通过其 limitByDistance、limitByIteration 和 limitByTime 属性设置的限制下移动到尽可能远的位置。

```
Wheel.addEventListener(Event.ENTER_FRAME, frameFunc);

function frameFunc(event:Event)
{
    if (Wheel != null)
    {
        var mat:Matrix = Wheel.transform.matrix;
        var pt = new Point(90,0);
        pt = mat.transformPoint(pt);

        ik.moveTo(pt);
    }
}
```

使用弹簧

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS5** 或更高版本

Flash Professional CS5 中的反向运动支持骨骼弹簧。可在创作期间设置骨骼弹簧，并且可在运行时添加或修改骨骼弹簧属性。弹簧是骨骼及其关节的属性。它包含两个属性：一个是 IKJoint.springStrength，用于设置弹簧的量；另一个是 IKJoint.springDamping，用于将向强度值添加阻力并更改弹簧的衰减率。

弹簧强度是介于 0-100 的百分比值， 0 表示完全刚性 （默认值）， 100 表示很松散且由物理特性控制。具有弹簧的骨骼响应其关节的移动。如果未启用其他转换 （旋转、 x 或 y），弹簧设置无效。

弹簧阻尼是介于 0-100 的百分比值， 0 表示没有阻力 （默认值）， 100 表示强阻尼的。阻尼会影响从骨骼初始移动到返回静止状态所用的时间。

要了解 IKArmature 对象是否启用了弹簧，请查看对象的 IKArmature.springsEnabled 属性。其他弹簧属性和方法属于各个 IKJoint 对象。可以为关节启用角度旋转和沿 X 轴和 Y 轴的平移。您可使用 IKJoint.setSpringAngle 指定旋转关节的弹簧角度位置，使用 IKJoint.setSpringPt 指定平移关节的弹簧位置。

在以下示例中，通过名称选择了骨骼并标识了其尾关节。代码测试父骨架，以查看是否已启用弹簧，然后为关节设置弹簧属性。

```
var arm:IKArmature = IKManager.getArmatureAt(0);
var bone:IKBone = arm.getBoneByName("c");
var joint:IKJoint = bone.tailJoint;
if (arm.springsEnabled) {
    joint.springStrength = 50; //medium spring strength
    joint.springDamping = 10; //light damping resistance
    if (joint.hasSpringAngle) {
        joint.setSpringAngle(30); //set angle for rotational spring
    }
}
```

使用 IK 事件

Flash Player 10 及更高版本， **Adobe AIR 1.5** 及更高版本，需要 **Flash CS4** 或更高版本

使用 IKEvent 类可以创建包含有关 IK 事件的信息的事件对象。IKEvent 信息描述由于超过指定时间、距离或迭代限制而终止的运动。

下面的代码演示用于跟踪时间限制事件的一个事件侦听器和处理函数。此事件处理函数在超过 IKMover 的时间限制时，报告所引发的事件的时间、距离、迭代计数和连接属性。

```
var ikmover:IKMover = new IKMover(endjoint, pos);
ikMover.limitByTime = true;
ikMover.timeLimit = 1000;

ikmover.addEventListener(IKEvent.TIME_LIMIT, timeLimitFunction);

function timeLimitFunction(evt:IKEvent):void
{
    trace("timeLimit hit");
    trace("time is " + evt.time);
    trace("distance is " + evt.distance);
    trace("iterationCount is " + evt.iterationCount);
    trace("IKJoint is " + evt.joint.name);
}
```

第 19 章：在三维 (3D) 环境中工作

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Flash Player 和 AIR 运行时可通过两种方式支持 3D 图形。您可以使用 Flash 显示列表上的三维显示对象。此方法适用于为 Flash 内容添加三维效果以及适用于多边形数量很少的对象。在 Flash Player 11 和 AIR 3 或更高版本中，可以使用 Stage3D API 呈现复杂 3D 场景。

Stage3D 视口不是显示对象。相反， 3D 图形会呈现于显示在 Flash 显示列表下方 （并且在任何 StageVideo 视口平面上方）的视口中。您可以使用可编程的 3D 管道（与 OpenGL 和 Direct3D 类似）替代 Flash DisplayObject 类来创建场景。此管道将三角形数据和纹理作为输入并使用提供的着色器程序来渲染场景。如果客户端计算机上存在兼容的图形处理单元 (GPU) 并具有受支持的驱动程序，则将使用硬件加速。

Stage3D 提供最底层的 API。在应用程序中，我们鼓励您使用支持 Stage3D 的 3D 框架。您可以创建自已的框架，或者使用已经提供的几个商业和开放源框架之一。

有关使用 Stage3D 开发 3D 应用程序以及可用 3D 框架的详细信息，请访问 Flash Player 开发人员中心：Stage 3D。

3D 显示对象的基础知识

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

二维 (2D) 对象和投影在二维屏幕上的三维 (3D) 对象之间的区别在于，三维对象增加了第三维。第三维使对象能够靠近或远离用户的视点。

如果将某个显示对象的 z 属性明确设置为数值，则该对象会自动创建一个 3D 转换矩阵。您可以通过更改此矩阵来修改该对象的 3D 转换设置。

此外， 3D 旋转与 2D 旋转也有所不同。在 2D 中，旋转轴始终垂直于 x/y 平面，即位于 z 轴上。在 3D 中，旋转轴可以位于 x、 y 或 z 轴中的任一轴上。通过设置显示对象的旋转属性和缩放属性，可以让该对象在 3D 空间中移动。

重要概念和术语

以下参考列表包含进行三维图形编程时会遇到的重要术语：

透视 在 2D 平面中将平行线表示成聚合于一个消失点，以创建深度和距离的视觉效果。

投影 制作高维对象的 2D 图像； 3D 投影将 3D 点映射到 2D 平面。

旋转 通过按圆周运动方向移动对象内的每个点来更改对象的方向 （通常也会更改其位置）。

转换 通过平移、旋转、缩放、倾斜或这些操作的组合更改 3D 点或点集。

平移 通过将对象内的每个点向同一方向移动相同的距离来更改对象的位置。

消失点 在以线性透视法表示平行线时，逐渐远离的平行线看似聚合在一起的点。

Vector 三维向量使用笛卡尔坐标 x、 y 和 z 轴表示三维空间中的点或位置。

顶点 角点。

纹理式网格 用于在 3D 空间中定义对象的任意点。

UV 映射 对 3D 表面应用纹理或位图的一种方法。UV 映射将值分配给图像上的坐标，以水平 (U) 轴和垂直 (V) 轴的百分比值形式表示。

T 值 当对象向当前观察点移近或远离当前观察点时，用来确定 3D 对象大小的比例因子。

剔除 呈现或不呈现带有特定缠绕的曲面。通过使用剔除，您可以隐藏对当前视点不可见的表面。

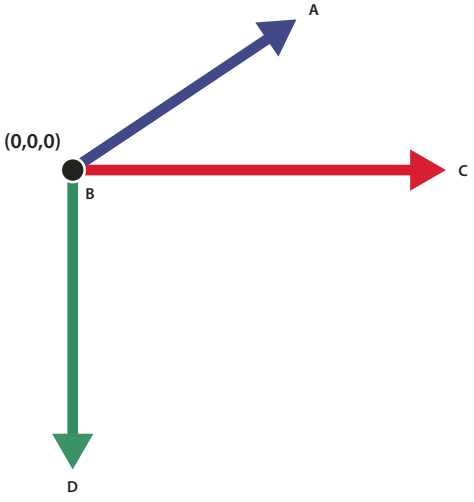
了解 Flash Player 和 AIR 运行时中的 3D 显示对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在 Flash Player 10 之前的 Flash Player 版本以及 Adobe AIR 1.5 之前的 Adobe AIR 版本中，显示对象有 x 和 y 两个属性，用于在 2D 平面上放置显示对象。从 Flash Player 10 和 Adobe AIR 1.5 开始，每个 ActionScript 显示对象都有一个 z 属性，利用该属性可以沿 z 轴放置显示对象， z 轴一般用于指示深度或距离。

Flash Player 10 和 Adobe AIR 1.5 引入了对 3D 效果的支持。但是显示对象本质上还是平面的。每个显示对象 （例如 MovieClip 对象或 Sprite 对象）最终会呈示于二维的单一平面上。通过 3D 功能，可在三个维中放置、移动、旋转以及按其他方式转换这些平面对象，还可管理 3D 点以及将这些点转换为 2D x、y 坐标，这样您就能将 3D 对象投影到 2D 视图上。通过这些功能，可以模拟出丰富的 3D 效果。

ActionScript 使用的 3D 坐标系与其他坐标系不同。在 ActionScript 中使用 2D 坐标系时，沿 x 轴向右移动过程中， x 的值增大，而沿 y 轴向下移动过程中， y 的值增大。 3D 坐标系仍遵从这一惯例，但另外添加了 z 轴，该轴的值随着远离视点而增大。



ActionScript 3D 坐标系中 x、 y 和 z 三条轴的正向。
A. +Z 轴 **B.** 原点 **C.** +X 轴 **D.** +Y 轴

注：请注意，Flash Player 和 AIR 始终在图层中表示 3D。也就是说，如果对象 A 在显示列表中位于对象 B 的前面，则无论这两个对象的 z 轴值为多少，Flash Player 或 AIR 都始终将 A 呈示在 B 的前面。若要解决显示列表顺序与 z 轴顺序之间的冲突，可使用 transform.getRelativeMatrix3D() 方法进行保存，然后重新对 3D 显示对象的图层排序。有关详细信息，请参阅第 303 页的 “ 使用 Matrix3D 对象重新排序显示 ”。

以下 ActionScript 类支持与 3D 相关的新功能：

1 flash.display.DisplayObject 类包含 z 属性和新的旋转和缩放属性，这些属性用于操作 3D 空间中的显示对象。DisplayObject.local3DToGlobal() 方法提供了一种将 3D 几何图形投影到 2D 平面的简单方式。

2 flash.geom.Vector3D 类可用作管理 3D 点的数据结构。该类还支持矢量数学运算。

3 flash.geom.Matrix3D 类支持复杂的 3D 几何转换，例如旋转、缩放和平移。

4 flash.geom.PerspectiveProjection 类控制着将 3D 几何图形映射到 2D 视图的相关参数。

在 ActionScript 中，有两种不同的模拟 3D 图像的方式：

1 在 3D 空间中排列平面对象并进行动画处理。这种方式需要使用显示对象的 x、 y 和 z 属性来对显示对象进行动画处理，或者使用 DisplayObject 类设置旋转和缩放属性。使用 DisplayObject.transform.matrix3D 对象可以实现更为复杂的运动。 DisplayObject.transform.perspectiveProjection 对象可自定义显示对象在 3D 透视中的绘制方式。如果需要对主要包含平面的 3D 对象进行动画处理，可以使用这种方式。这种方式的例子包括 3D 图库或者 3D 空间中排列的 2D 动画对象。

2 从 3D 几何图形生成 2D 三角形，然后用纹理呈示这些三角形。要使用这种方式，必须首先定义和管理有关 3D 对象的数据，然后将这些数据转换成要呈示的 2D 三角形。可以将位图纹理映射到这些三角形，然后使用 Graphics.drawTriangles() 方法将三角形绘制为图形对象。这种方式的例子包括从文件中加载 3D 模型数据并将模型呈示到屏幕上，或以三角形网格形式生成和绘制 3D 图形。

创建和移动 3D 显示对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

若要将 2D 显示对象转换为 3D 显示对象，可将其 z 属性明确设置为一个数值。如果为 z 属性指定一个值，则会为显示对象创建一个新的 Transform 对象。设置 DisplayObject.rotationX 或 DisplayObject.rotationY 属性也会创建新的 Transform 对象。Transform 对象包含 Matrix3D 属性，该属性控制显示对象在 3D 空间中的表示方式。

下面的代码设置名为 “leaf” 的显示对象的坐标：

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

在 leaf 的 Transform 对象的 matrix3D 属性中，可以查看这些值以及从这些值派生的属性：

```
var leafMatrix:Matrix3D  = leaf.transform.matrix3D;

trace(leafMatrix.position.x);
trace(leafMatrix.position.y);
trace(leafMatrix.position.z);
trace(leafMatrix.position.length);
trace(leafMatrix.position.lengthSquared);
```

有关 Transform 对象的属性的信息，请参阅 Transform 类。有关 Matrix3D 对象的属性的信息，请参阅 Matrix3D 类。

在 3D 空间中移动对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

通过更改对象的 x、 y 或 z 属性的值，可以在 3D 空间中移动对象。如果更改 z 属性的值，对象看起来就会相应地靠近或远离观察者。

下面的代码在响应事件时，通过更改两个椭圆的 z 属性的值，沿着各自的 z 轴前后移动这两个椭圆。 ellipse2 的移动速度比 ellipse1 快：其 z 属性针对每个 Frame 事件增加 20 倍，而 ellipse1 的 z 属性增加 10 倍：

```
var depth:int = 1000;

function ellipse1FrameHandler(e:Event):void
{
    ellipse1Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 10;
}
function ellipse2FrameHandler(e:Event):void
{
    ellipse2Back = setDepth(e, ellipse1Back);
    e.currentTarget.z += ellipse1Back * 20;
}
function setDepth(e:Event, d:int):int
{
    if(e.currentTarget.z > depth)
    {
        e.currentTarget.z = depth;
        d = -1;
    }
    else if (e.currentTarget.z <  0)
    {
        e.currentTarget.z = 0;
        d = 1;
    }
}
```

在 3D 空间中旋转对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

可以通过三种方式旋转对象，具体采用哪一种取决于设置对象的 rotationX、 rotationY 和 rotationZ 这三个旋转属性的方式。

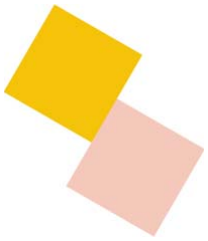
下图显示两个未旋转的正方形：

下一个图显示两个正方形，它们在正方形容器的 rotationY 属性增大时在 y 轴上旋转。通过旋转这两个正方形的容器或父显示对象，可以旋转这两个正方形：

```
container.rotationY += 10;
```

下一个图显示当设置正方形容器的 rotationX 属性时发生的变化。此操作会在 x 轴上旋转正方形。

下一个图显示当增大正方形容器的 rotationZ 属性时发生的变化。此操作会在 z 轴上旋转正方形。



显示对象可以在 3D 空间中同时移动和旋转。

将 3D 对象投影到 2D 视图上

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

flash.geom 包中的 PerspectiveProjection 类提供了一种简单在 3D 空间中移动显示对象时应用基本透视的简单方式。

如果未明确创建 3D 空间的透视投影，3D 引擎将使用默认的 PerspectiveProjection 对象，该对象存在于根上并会传播到其所有子项上。

用于定义 PerspectiveProjection 对象如何显示 3D 空间的三个属性是：

- fieldOfView
- projectionCenter
- focalLength

修改 fieldOfView 的值会导致自动修改 focalLength 的值，反之亦然，因为这两个属性相互依赖。

如果给定 fieldOfView 值，用于计算 focalLength 的值的公式为：

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

通常，您需要明确修改 fieldOfView 属性。

视野

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

通过操作 PerspectiveProjection 类的 fieldOfView 属性，可以使逐渐靠近观察者的 3D 显示对象变大，而使逐渐远离观察者的对象变小。

fieldOfView 属性指定一个介于 0 到 180 度的角度，该角度确定透视投影的强度。该值越大，沿 z 轴移动的显示对象的扭曲程度就越大。如果 fieldOfView 值较小，则缩放程度较低，使对象看起来在空间中只是稍稍后移。如果 fieldOfView 值较大，则会导致较大的扭曲，并显示为较大的移动。如果达到最大值 179.9999... 度，则会出现极端的鱼眼摄像头镜头效果。 fieldOfView 的最大值为 179.9999...，最小值为 0.00001...。精确的 0 和 180 值是非法值。

投影中心

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

projectionCenter 属性表示透视投影的消失点。该属性作为相对于舞台左上角默认注册点 (0,0) 的偏移量。

当对象逐渐远离观察者时，该对象将朝消失点倾斜直到最终消失。想象一下无限长的走廊。当朝走廊远处看时，两边的墙壁将聚合到走廊远处的消失点。

如果消失点位于舞台的中心，则走廊将消失于该中心点。 projectionCenter 属性的默认值是舞台的中心。例如，如果希望元素出现在舞台的左边，而 3D 区域出现在右边，可将 projectionCenter 设置为舞台右边的点，使之成为 3D 查看区域的消失点。

焦距

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

focalLength 属性表示视点原点 (0,0,0) 与显示对象在 z 轴上的位置之间的距离。

较长的焦距相当于视野较窄、对象间距离经过压缩的摄远镜头。较短的焦距相当于广角镜头，可获得较宽的视野和较大的扭曲。中等的焦距相当于肉眼所见的效果。

通常，当显示对象移动时， focalLength 属性会在透视转换过程中动态重新进行计算，不过您可以明确设置该属性。

默认透视投影值

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在根上创建的默认 PerspectiveProjection 对象具有以下值：

- fieldOfView：55
- perspectiveCenter：stagewidth/2, stageHeight/2
- focalLength：stageWidth/ 2 * (cos(fieldOfView/2) / sin(fieldOfView/2))

如果您没有创建自己的 PerspectiveProjection 对象，则默认使用这些值。

如果要自行修改 projectionCenter 和 fieldOfView 属性，则可以实例化您自己的 PerspectiveProjection 对象。在这种情况下，新建对象的默认值如下 （假设默认舞台大小为 500 x 500）：

- fieldOfView：55
- perspectiveCenter：250,250
- focalLength: 480.24554443359375

示例：透视投影

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

下面的示例演示如何使用透视投影来创建 3D 空间。该示例演示如何通过 projectionCenter 属性来修改消失点和更改空间的透视投影。进行这种修改后，将强制重新计算 focalLength 和 fieldOfView （及其相关的 3D 空间扭曲）。

此示例：

1 创建一个名为 center 的 sprite，作为包含十字准线的圆

2 将 center sprite 的坐标指定给根的 transform 属性的 perspectiveProjection 属性的 projectionCenter 属性

3 为鼠标事件添加事件侦听器，用于调用修改 projectionCenter 的处理程序，以便跟踪 center 对象的位置

4 创建四个折叠样式的框，这四个框将形成透视空间的墙壁

在测试此示例时， ProjectionDragger.swf 将圆拖动到不同的位置。消失点将跟随圆移动，直到释放圆时才固定下来。在移动投影中心使其远离舞台中心时，观察包围空间的框所发生的拉伸和出现的扭曲。

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。在 Samples/ProjectionDragger 文件夹中可以找到 ProjectionDragger 应用程序文件。

```
package
{
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.geom.Point;
    import flash.events.*;
    public class ProjectionDragger extends Sprite
    {
        private var center : Sprite;
        private var boxPanel:Shape;
        private var inDrag:Boolean = false;

        public function ProjectionDragger():void
        {
            createBoxes();
            createCenter();
        }
        public function createCenter():void
        {
            var  centerRadius:int = 20;

            center = new Sprite();

            // circle
            center.graphics.lineStyle(1, 0x000099);
            center.graphics.beginFill(0xCCCCCC, 0.5);
            center.graphics.drawCircle(0, 0, centerRadius);
            center.graphics.endFill();
            // cross hairs
            center.graphics.moveTo(0, centerRadius);
            center.graphics.lineTo(0, -centerRadius);
            center.graphics.moveTo(centerRadius, 0);
            center.graphics.lineTo(-centerRadius, 0);
            center.x = 175;
            center.y = 175;
            center.z = 0;
            this.addChild(center);

            center.addEventListener(MouseEvent.MOUSE_DOWN, startDragProjectionCenter);
            center.addEventListener(MouseEvent.MOUSE_UP, stopDragProjectionCenter);
            center.addEventListener( MouseEvent.MOUSE_MOVE, doDragProjectionCenter);
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x, center.y);
        }
        public function createBoxes():void
        {
            // createBoxPanel();
            var boxWidth:int = 50;
            var boxHeight:int = 50;
            var numLayers:int = 12;
            var depthPerLayer:int = 50;

            // var boxVec:Vector.<Shape> = new Vector.<Shape>(numLayers);
            for (var i:int = 0; i < numLayers; i++)
```

```
            {
                this.addChild(createBox(150, 50, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xCCCCFF));
                this.addChild(createBox(50, 150, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xFFCCCC));
                this.addChild(createBox(250, 150, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xCCFFCC));
                this.addChild(createBox(150, 250, (numLayers - i) * depthPerLayer, boxWidth, boxHeight,
0xDDDDDD));
            }
        }

        public function createBox(xPos:int = 0, yPos:int = 0, zPos:int = 100, w:int = 50, h:int = 50,
color:int = 0xDDDDDD):Shape
        {
            var box:Shape = new Shape();
            box.graphics.lineStyle(2, 0x666666);
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, w, h);
            box.graphics.endFill();
            box.x = xPos;
            box.y = yPos;
            box.z = zPos;
            return box;
        }
        public function startDragProjectionCenter(e:Event)
        {
            center.startDrag();
            inDrag = true;
        }

        public function doDragProjectionCenter(e:Event)
        {
            if (inDrag)
            {
                root.transform.perspectiveProjection.projectionCenter = new Point(center.x, center.y);
            }
        }

        public function stopDragProjectionCenter(e:Event)
        {
            center.stopDrag();
            root.transform.perspectiveProjection.projectionCenter = new Point(center.x, center.y);
            inDrag = false;
        }
    }
}
```

对于更加复杂的透视投影，请使用 Matrix3D 类。

执行复杂的 3D 转换

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

使用 Matrix3D 类可以转换坐标空间内的 3D 点，也可以将 3D 点从一个坐标空间映射到另一个坐标空间。

无需了解矩阵数学，即可使用 Matrix3D 类。大多数常见的转换操作都可以通过该类的方法进行处理。您不必担心如何明确设置或计算矩阵中每个元素的值。

将显示对象的 z 属性设置为数值后，可以使用该显示对象的 Transform 对象的 Matrix3D 属性来检索显示对象的转换矩阵：

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

您可以用 Matrix3D 对象的方法对显示对象执行平移、旋转、缩放和透视投影。

使用 Vector3D 类及其 x、 y 和 z 属性可管理 3D 点。该类还可以表示具有方向和大小的物理空间矢量。通过 Vector3D 类的方法，可以执行有关空间矢量的常见计算，例如加法、点积和叉积计算。

注: Vector3D 类与 ActionScript Vector 类无关。Vector3D 类包含的属性和方法用于定义和操作 3D 点，而 Vector 类则支持类型对象数组。

创建 Matrix3D 对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

有三种主要的创建或检索 Matrix3D 对象的方式：

- 使用 Matrix3D() 构造函数方法实例化新的矩阵。 Matrix3D() 构造函数使用包含 16 个数值的 Vector 对象，并将每个值分别放入一个矩阵单元格中。例如：

  ```
  var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
  ```

- 设置显示对象的 z 属性的值。然后，从该对象的 transform.matrix3D 属性检索转换矩阵。
- 通过对根显示对象调用 perspectiveProjection.tomatrix3D() 方法，检索用于控制舞台上 3D 对象显示的 Matrix3D 对象。

应用多种 3D 转换

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

通过 Matrix3D 对象，可以一次应用多种 3D 转换。例如，如果要旋转、缩放然后移动立方体，可以对立方体的每个点分别应用这三种转换。但是，还有一种高效得多的方法，即在一个 Matrix3D 对象中预先计算多种转换，然后对每个点执行一次矩阵转换。

注：矩阵转换的应用顺序非常重要。矩阵运算的顺序是不能交换的。例如，先应用旋转再应用平移与先应用平移再应用旋转，二者的结果是不同的。

下面的示例演示执行多种 3D 转换的两种方式。

```
package {
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;
    import flash.geom.*;

public class Matrix3DTransformsExample extends Sprite
    {
        private var rect1:Shape;
        private var rect2:Shape;

public function Matrix3DTransformsExample():void
        {
            var pp:PerspectiveProjection = this.transform.perspectiveProjection;
            pp.projectionCenter = new Point(275,200);
            this.transform.perspectiveProjection = pp;

            rect1 = new Shape();
            rect1.x = -70;
            rect1.y = -40;
```

```
            rect1.z = 0;
            rect1.graphics.beginFill(0xFF8800);
            rect1.graphics.drawRect(0,0,50,80);
            rect1.graphics.endFill();
            addChild(rect1);

            rect2 = new Shape();
            rect2.x = 20;
            rect2.y = -40;
            rect2.z = 0;
            rect2.graphics.beginFill(0xFF0088);
            rect2.graphics.drawRect(0,0,50,80);
            rect2.graphics.endFill();
            addChild(rect2);

            doTransforms();
        }

        private function doTransforms():void
        {
            rect1.rotationX = 15;
            rect1.scaleX = 1.2;
            rect1.x += 100;
            rect1.y += 50;
            rect1.rotationZ = 10;

            var matrix:Matrix3D = rect2.transform.matrix3D;
            matrix.appendRotation(15, Vector3D.X_AXIS);
            matrix.appendScale(1.2, 1, 1);
            matrix.appendTranslation(100, 50, 0);
            matrix.appendRotation(10, Vector3D.Z_AXIS);
            rect2.transform.matrix3D = matrix;
        }
    }
}
```

在 doTransforms() 方法中，第一个代码块使用 DisplayObject 属性更改矩形形状的旋转、缩放和位置。第二个代码块使用 Matrix3D 类的方法执行相同的转换。

使用 Matrix3D 方法的主要优点在于，所有计算都是在矩阵中提前执行的，然后这些计算只需对显示对象应用一次 （前提是设置了显示对象的 transform.matrix3D 属性）。通过设置 DisplayObject 属性可使源代码更易于阅读。但是，每次设置旋转和缩放属性后，会导致进行大量计算并更改显示对象的多个属性。

如果您的代码要多次对显示对象应用相同的复杂转换，应将 Matrix3D 对象保存为变量，然后反复应用该变量。

使用 Matrix3D 对象重新排序显示

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

如前所述，显示列表中显示对象的层叠顺序确定了对象的显示顺序，这些对象的相对 z 轴值对显示顺序没有影响。如果您的动画将显示对象的属性转换为一种与显示列表中的顺序不同的顺序，则观察者看到的显示对象层叠顺序与 z 轴层叠顺序不一致。因此，在视觉上应当远离观察者的对象可能反而会靠近观察者。

为确保 3D 显示对象的层叠顺序对应于对象的相对深度，请使用如下方法：

1 使用 Transform 对象的 getRelativeMatrix3D() 方法获取子级 3D 显示对象的相对 z 轴值。

2 使用 removeChild() 方法从显示列表中删除对象。

3 根据显示对象的相对 z 轴值对显示对象进行排序。

4 使用 addChild() 方法以相反顺序将子对象添加到显示列表中。

这种重新排序方法可确保您的对象按照各自的相对 z 轴值进行显示。

下面的代码将 3D 框的六面设置为正确的显示顺序。这段代码对经过旋转的框的各面进行重新排序：

```
public var faces:Array; . . .

public function ReorderChildren()
{
    for(var ind:uint = 0; ind < 6; ind++)
    {
        faces[ind].z = faces[ind].child.transform.getRelativeMatrix3D(root).position.z;
        this.removeChild(faces[ind].child);
    }
    faces.sortOn("z", Array.NUMERIC | Array.DESCENDING);
    for (ind = 0; ind < 6; ind++)
    {
        this.addChild(faces[ind].child);
    }
}
```

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。在 Samples/ReorderByZ 文件夹中可以找到这些应用程序文件。

通过三角形获得 3D 效果

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在 ActionScript 中，使用 Graphics.drawTriangles() 方法执行位图转换，因为 3D 模型由空间中的一系列三角形表示。（但是，Flash Player 和 AIR 不支持深度缓冲，因此显示对象在本质上仍然是平面的，即 2D。这一点在第 295 页的 “ 了解 Flash Player 和 AIR 运行时中的 3D 显示对象 ”中进行了说明。）Graphics.drawTriangles() 方法通过一组坐标来绘制三角形路径，它与 Graphics.drawPath() 方法类似。

若要熟悉 Graphics.drawPath() 方法的使用，请参阅第 197 页的 “ 绘制路径 ”。

Graphics.drawTriangles() 方法使用 Vector.<Number> 来指定三角形路径的点位置：

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number> = null,
culling:String = "none"):void
```

drawTriangles() 的第一个参数是唯一的必需参数：vertices 参数。该参数是由定义坐标的数字组成的矢量，通过该矢量即可绘制三角形。每三组坐标 （六个数字）表示一个三角形路径。如果没有 indices 参数，矢量的长度应始终为六的倍数，因为每个三角形都需要三个坐标对 （三组 x/y 值对）。例如：

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

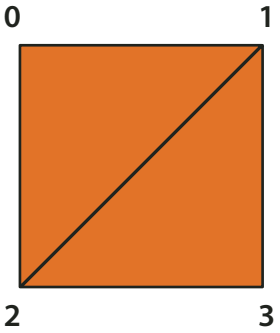
这些三角形不共享任何点，但是如果有共享点，就可以使用第二个 drawTriangles() 参数 indices 来对多个三角形重复使用 vertices 矢量中的值。

使用 indices 参数时，请注意， indices 值是点索引，而不是直接与 vertices 数组元素相关的索引。也就是说， vertices 矢量中由 indices 定义的索引实际上是除以 2 后的实际索引。例如，对于 vertices 矢量的第三个点，即使该点的第一个数值从矢量索引 4 开始，也使用 indices 值 2。

例如，使用 indices 参数合并两个三角形使二者共享对边：

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

请注意，尽管正方形现在是通过两个三角形绘制的，但在 vertices 矢量中只指定了四个点。通过使用 indices，两个三角形共享的两点将由每个三角形重复使用。这样可以将顶点总数从 6 （12 个数字）减少为 4 （8 个数字）：



使用 vertices 参数通过两个三角形绘制的正方形

对于较大的三角形网格，这种方法非常有用，因为这种情况下大多数点由多个三角形共享。

所有填充方式都可应用于三角形。填充应用于三角形网格的方式与应用于其他形状的方式是相同的。

转换位图

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

位图转换可在三维对象上提供透视视觉效果或 “ 纹理 ”。具体而言，您可以朝消失点的方向扭曲位图，这样图像在朝消失点方向移动时会出现收缩效果。或者，您可以使用二维位图为三维对象创建表面，从而提供纹理视觉效果或将三维对象 “ 包裹 ” 起来。

使用消失点的二维表面和用位图包裹的三维对象。

UV 映射

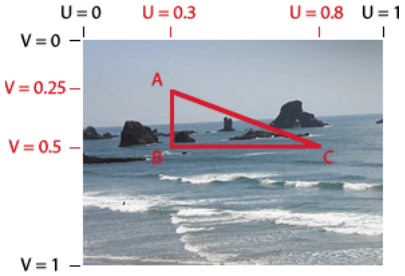
Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

一旦开始处理纹理，您就要使用 drawTriangles() 的 uvtData 参数。此参数用于为位图填充设置 UV 映射。

UV 映射是一种纹理化对象的方法。它依赖于两个值：U 水平 (x) 值和 V 垂直 (y) 值。这两个值不是基于像素值，而是基于百分比。 0 U 和 0 V 表示图像的坐上角， 1 U 和 1 V 表示右下角：

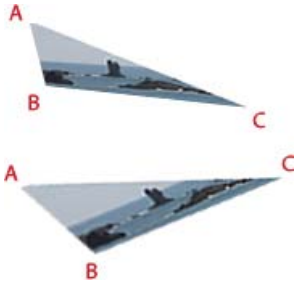
位图图像上的 UV 0 和 1 位置

可以为三角形的矢量指定 UV 坐标，从而将矢量自身关联到图像上的相应位置：



位图图像的三角形区域的 UV 坐标

UV 值与三角形的点保持一致：



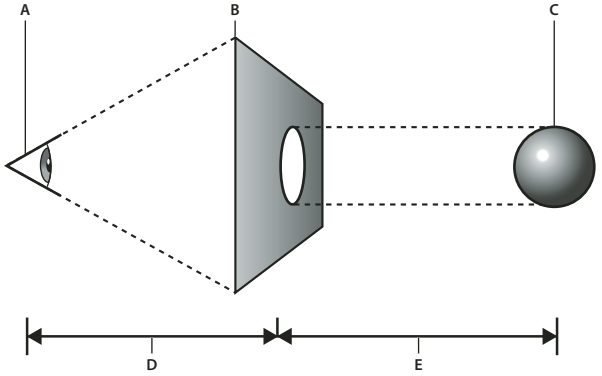
三角形的顶点移动且位图发生扭曲，从而使单个点的 UV 值保持不变

由于 ActionScript 3D 转换应用于与位图关联的三角形，位图图像将根据 UV 值应用于三角形。因此，不要使用矩阵运算，而应通过设置或调整 UV 值来实现三维效果。

Graphics.drawTriangles() 方法也接受关于三维转换的一条可选信息：T 值。 uvtData 中的 T 值表示 3D 透视，更具体地说，表示相关顶点的缩放系数。 UVT 映射向 UV 映射加入了透视修正处理。例如，如果将对象放置在 3D 空间中远离视点的位置，使之显示为原始大小的 50%，则该对象的 T 值为 0.5。因为在 3D 空间中是通过绘制三角形来表示对象的，所以对象在 z 轴上的位置确定对象的 T 值。用于确定 T 值的等式为：

```
T = focalLength/(focalLength + z);
```

在该等式中， focalLength 表示焦距或计算得出的 “ 屏幕 ” 位置，它决定了视图中提供的透视大小。



焦距与 z 值
A. 视点 **B.** 屏幕 **C.** 3D 对象 **D.** focalLength 值 **E.** z 值

T 值用于缩放基本形状，使这些形状看起来更远。该值通常用于将 3D 点转换为 2D 点。而对于 UVT 数据，它也用于在透视三角形内的点之间缩放位图。

在定义 UVT 值时，T 值紧跟在为顶点定义的 UV 值后面。纳入 T 之后，uvtData 参数中的每三个值（U、V 和 T）与 vertices 参数中的每两个值 （x 和 y）相匹配。在只有 UV 值的情况下， uvtData.length == vertices.length。在加入 T 值的情况下，uvtData.length = 1.5*vertices.length。

下面的示例演示使用 UVT 数据在 3D 空间中旋转平面。本例使用了一幅名为 ocean.jpg 的图像和一个 " 帮助器 " 类 ImageLoader，该类用于加载 ocean.jpg 图像以便将其分配给 BitmapData 对象。

下面是 ImageLoader 类的源代码 （请将此代码保存到命名为 ImageLoader.as 的文件中）：

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.net.URLRequest;
    public class ImageLoader extends Sprite {
        public var url:String;
        public var bitmap:Bitmap;
    public function ImageLoader(loc:String = null) {
        if (loc != null){
            url = loc;
            loadImage();
        }
    }
    public function loadImage():void{
        if (url != null){
            var loader:Loader = new Loader();
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
            loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_ERROR, onIoError);

                var req:URLRequest = new URLRequest(url);
                loader.load(req);
            }
        }

    private function onComplete(event:Event):void {
            var loader:Loader = Loader(event.target.loader);
            var info:LoaderInfo = LoaderInfo(loader.contentLoaderInfo);
            this.bitmap = info.content as Bitmap;
            this.dispatchEvent(new Event(Event.COMPLETE));
    }

    private function onIoError(event:IOErrorEvent):void {
            trace("onIoError: " + event);
    }
    }
}
```

下面的 ActionScript 用三角形、 UV 映射和 T 值使图像获得这样的显示效果：图像朝着消失点逐渐收缩并不断旋转。将此代码保存到命名为 Spinning3dOcean.as 的文件中：

```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    public class Spinning3dOcean extends Sprite {
        // plane vertex coordinates (and t values)
        var x1:Number = -100,y1:Number = -100,z1:Number = 0,t1:Number = 0;
        var x2:Number = 100,y2:Number = -100,z2:Number = 0,t2:Number = 0;
        var x3:Number = 100,y3:Number = 100,z3:Number = 0,t3:Number = 0;
        var x4:Number = -100,y4:Number = 100,z4:Number = 0,t4:Number = 0;
        var focalLength:Number = 200;
        // 2 triangles for 1 plane, indices will always be the same
        var indices:Vector.<int>;

        var container:Sprite;

        var bitmapData:BitmapData; // texture
        var imageLoader:ImageLoader;
        public function Spinning3dOcean():void {
            indices =  new Vector.<int>();
            indices.push(0,1,3, 1,2,3);
```

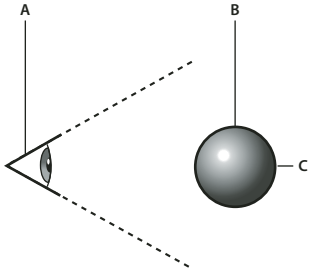
```
            container = new Sprite(); // container to draw triangles in
            container.x = 200;
            container.y = 200;
            addChild(container);

            imageLoader = new ImageLoader("ocean.jpg");
            imageLoader.addEventListener(Event.COMPLETE, onImageLoaded);
        }
        function onImageLoaded(event:Event):void {
            bitmapData = imageLoader.bitmap.bitmapData;
            // animate every frame
            addEventListener(Event.ENTER_FRAME, rotatePlane);
        }
        function rotatePlane(event:Event):void {
            // rotate vertices over time
            var ticker = getTimer()/400;
            z2 = z3 = -(z1 = z4 = 100*Math.sin(ticker));
            x2 = x3 = -(x1 = x4 = 100*Math.cos(ticker));

            // calculate t values
            t1 = focalLength/(focalLength + z1);
            t2 = focalLength/(focalLength + z2);
            t3 = focalLength/(focalLength + z3);
            t4 = focalLength/(focalLength + z4);

            // determine triangle vertices based on t values
            var vertices:Vector.<Number> = new Vector.<Number>();
            vertices.push(x1*t1,y1*t1, x2*t2,y2*t2, x3*t3,y3*t3, x4*t4,y4*t4);
            // set T values allowing perspective to change
            // as each vertex moves around in z space
            var uvtData:Vector.<Number> = new Vector.<Number>();
            uvtData.push(0,0,t1, 1,0,t2, 1,1,t3, 0,1,t4);

            // draw
            container.graphics.clear();
            container.graphics.beginBitmapFill(bitmapData);
            container.graphics.drawTriangles(vertices, indices, uvtData);
        }
    }
}
```

若要测试此示例，请在名为 “ocean.jpg” 的图像所在的目录中保存这两个类文件。您可以看到原始位图如何转换为在 3D 空间中消失于远处并不断旋转的效果。

剔除

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

剔除是一个过程，用于确定三维对象的哪些表面因对当前视点不可见而不应被渲染器所呈示。在 3D 空间中，三维对象 “ 背面 ” 的表面对视点不可见：



3D 对象的背面对视点不可见。
A. 视点 **B.** 3D 对象 **C.** 三维对象的背面

从本质上讲，不管三角形的大小、形状或位置如何，所有三角形都会始终呈示。剔除能确保 Flash Player 或 AIR 正确呈示 3D 对象。此外，为了减少呈示周期，有时您会希望渲染器跳过某些三角形。考虑在空间中旋转的立方体。在任何给定时间，您看到的立方体的面数不会超过三个面，这是因为不可见的面朝向立方体背面的其他方向。因为这些面不可见，所以渲染器不应绘制它们。如果不用剔除， Flash Player 或 AIR 就既要绘制正面也要绘制背面。

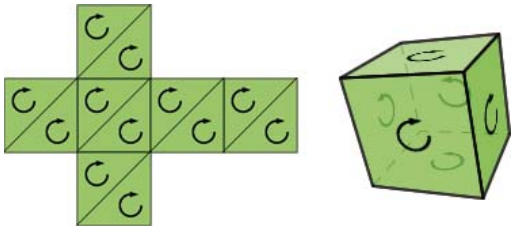
立方体的某些面对当前视点不可见

因此， Graphics.drawTriangles() 方法采用第四个参数来建立剔除值：

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

该剔除参数是来自 TriangleCulling 枚举类的值：TriangleCulling.NONE、 TriangleCulling.POSITIVE 和 TriangleCulling.NEGATIVE。这些值与定义对象表面的三角形路径的方向有关。用于确定剔除的 ActionScript API 假设 3D 形状的所有外向三角形都是以同一路径方向绘制的。一旦三角形面经过旋转后，其路径方向也会改变。此时可以剔除 （不呈示）该三角形。

因此，如果 TriangleCulling 值为 POSITIVE，则会移除正向路径方向 （顺时针）的三角形。如果 TriangleCulling 值为 NEGATIVE，则会移除负向路径方向 （逆时针）的三角形。对于立方体，朝前的表面具有正向路径方向，而朝后的表面具有负向路径方向：



“ 展开 ” 的立方体，用以显示路径方向。当立方体 “ 恢复原状 ” 后，背面路径方向变为相反方向。

若要了解剔除的工作原理，请从前面 第 305 页的 “UV 映射 ” 中的示例开始，将 drawTriangles() 方法的剔除参数设置为 TriangleCulling.NEGATIVE：

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

请注意，在对象旋转时，未呈现图像的 “ 背面 ”。

第 20 章：文本使用基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 Adobe® Flash® Player 或 Adobe® AIR™ 中，若要在屏幕上显示文本，可以使用 TextField 类的实例或使用 Flash 文本引擎类。这些类可用于执行文本的创建、显示和格式设置。或者，您可以使用 Text Layout Framework (TLF)，一种便于使用的基于 Flash 文本引擎类的组件库。在移动设备上，您可以使用 StageText 类进行文本输入。

您可以为文本字段确定具体内容，或者指定文本来源，然后设置该文本的外观。还可以在用户输入文本或单击超文本链接时响应用户事件。

您可以使用 TextField 类，也可以使用 Flash 文本引擎类在 Flash Player 和 AIR 中显示和管理文本。可使用 TextField 类创建文本对象以显示和输入文本。 TextField 类为其他基于文本的组件提供了基础，例如 TextArea 和 TextInput。可以使用 TextFormat 类来设置 TextField 对象的字符和段落格式，可以使用 Textfield.styleSheet 属性和 StyleSheet 类来应用层叠样式表 (CSS)。可将 HTML 格式的文本直接分配给文本字段， HTML 格式的文本可包含嵌入的媒体 （影片剪辑、 SWF 文件、 GIF 文件、 PNG 文件和 JPEG 文件）。

从 Flash Player 10 和 Adobe AIR 1.5 开始提供的 Flash 文本引擎为文本度量、格式设置和双向文本的复杂控制提供底层支持。它还提供改进的文本流和增强的语言支持。尽管可以使用 Flash 文本引擎创建和管理文本元素，但设计 Flash 文本引擎的主要目的在于为创建文本处理组件提供基础，并且它要求较高的编程技术。 Text Layout Framework 提供了一种比较容易的使用该新文本引擎的高级功能的方法，它包括一个基于 Flash 文本引擎的文本处理组件。Text Layout Framework 是完全内置于 ActionScript 3.0 的可扩展库。您可以使用现有的 TLF 组件，或者使用框架构建您自己的文本组件。

AIR 3 中将开始引入 StageText 类，以提供本地文本输入字段。由于此字段是设备操作系统提供的，它将为最熟悉设备的用户提供相关经验。 StageText 实例不是显示对象。您不需要将其添加到显示列表，而是为实例分配一个舞台和一个位于此舞台上的称为视口的显示区域。 StageText 实例显示在显示对象的前面。

有关这些主题的详细信息，请参阅：

- 第 313 页的 " 使用 TextField 类 "
- 第 334 页的 " 使用 Flash 文本引擎 "
- 第 360 页的 " 使用 Text Layout Framework"
- 使用 StageText 的本机文本输入

重要概念和术语

以下参考列表包含处理文本涉及的重要术语：

级联样式表 标准语法，用于指定以 XML （或 HTML）格式构建的内容的样式和格式设置。

设备字体 用户计算机中安装的一种字体。

动态文本字段 其内容可由 ActionScript 更改，而不能通过用户输入进行更改的文本字段。

嵌入字体 其字符轮廓数据存储在应用程序 SWF 文件中的一种字体。

HTML 文本 使用 ActionScript 输入到文本字段中的文本内容，包括 HTML 格式标签和实际文本内容。

输入文本字段 其内容既可通过用户输入进行更改也可通过 ActionScript 进行更改的文本字段。

字距微调 调整两个字符之间的间距，使字与字的间距比例更佳，文字更易于阅读。

静态文本字段 在创作工具中创建的文本字段，无法在运行 SWF 文件时更改其内容。

文本行量度 文本字段中文本内容不同部分的大小的量度，如文本的基线、字符顶部的高度、下行字符 （某些小写字母延伸到基线以下的部分）的大小，等等。

跟踪 调整字母组或文本块之间的间距，以增大或减小密度，使文本更容易阅读。

第 21 章：使用 TextField 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 Adobe® Flash® Player 或 Adobe® AIR™ 中，您可以使用 TextField 类的实例在屏幕上显示文本或创建文本输出字段。TextField 类是其他基于文本的组件 （如 TextArea 组件或 TextInput 组件）的基础。

文本字段内容可以在 SWF 文件中预先指定、从文本文件或数据库中加载，或由用户在与应用程序交互时输入。在文本字段内，文本可以显示为呈示的 HTML 内容，并可在其中嵌入图像。创建文本字段的实例后，可以使用 TextFormat 和 StyleSheet 等 flash.text 类控制文本的外观。 flash.text 包几乎包含所有与在 ActionScript 中创建、管理文本和设置文件格式相关的类。

可以用 TextFormat 对象定义格式设置并将此对象分配给文本字段，以此来设置文本格式。如果文本字段包含 HTML 文本，则可以对文本字段应用 StyleSheet 对象，以便将样式分配给文本字段内容的特定片段。TextFormat 对象或 StyleSheet 对象包含定义文本外观 （例如颜色、大小和粗细）的属性。 TextFormat 对象可以将属性分配给文本字段中的所有内容，也可以分配给某个范围的文本。例如，在同一文本字段中，一个句子可以是粗体的红色文本，而下一个句子可以是斜体的蓝色文本。

除了 flash.text 包中的类以外，您还可以使用 flash.events.TextEvent 类响应与文本相关的用户操作。

更多帮助主题

第 319 页的 “ 指定文本格式 ”

第 314 页的 “ 显示 HTML 文本 ”

第 320 页的 “ 应用层叠样式表 ”

显示文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

虽然 Adobe Flash Builder 和 Flash Professional 等创作工具提供了多种用于显示文本的选项（包括与文本相关的组件或文本工具），但是以编程方式显示文本的最简单方法还是通过文本字段。

文本类型

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

文本字段中文本的类型根据其来源进行划分：

- 动态文本

  动态文本包含从外部源 （例如文本文件、 XML 文件以及远程 Web 服务）加载的内容。

- 输入文本

  输入文本是指用户输入的任何文本或用户可以编辑的动态文本。可以设置样式表来设置输入文本的格式，或使用 flash.text.TextFormat 类为输入内容指定文本字段的属性。有关详细信息，请参阅第 317 页的 “ 捕获文本输入 ”。

- 静态文本

静态文本只通过 Flash Professional 创建。不能使用 ActionScript 3.0 创建静态文本实例。但是，可以使用 ActionScript 类（例如 StaticText 和 TextSnapshot）来操作现有的静态文本实例。有关详细信息，请参阅第 324 页的 “ 使用静态文本 ”。

修改文本字段内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可通过将字符串分配给 flash.text.TextField.text 属性来定义动态文本。可以直接将字符串赋予该属性，如下所示：

```
myTextField.text = "Hello World";
```

还可以为 text 属性赋予在脚本中定义的变量值，如下例所示：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello World";

        public function TextWithImage()
        {
            addChild(myTextBox);
            myTextBox.text = myText;
        }
    }
}
```

或者，可以将一个远程变量的值赋予 text 属性。从远程源加载文本值有三种方式：

- flash.net.URLLoader 和 flash.net.URLRequest 类可以从本地或远程位置为文本加载变量。
- FlashVars 属性被嵌入到承载 SWF 文件的 HTML 页中，可以包含文本变量的值。
- flash.net.SharedObject 类管理值的永久存储。有关详细信息，请参阅第 602 页的 “ 存储本地数据 ”。

显示 HTML 文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

flash.text.TextField 类具有一个 htmlText 属性，可使用它将您的文本字符串标识为包含用于设置内容格式的 HTML 标签。如下例所示，必须将您的字符串值赋予 htmlText 属性（而不是 text 属性），以便 Flash Player 或 AIR 将文本呈示为 HTML：

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Flash Player 和 AIR 支持用于 htmlText 属性的 HTML 标签和实体的一个子集。“ActionScript 3.0 参考 ” 中的 flash.text.TextField.htmlText 属性描述提供了关于支持的 HTML 标记和实体的详细信息。

一旦您使用 htmlText 属性指定了内容，就可以使用样式表或 textformat 标签来管理内容的格式设置。有关详细信息，请参阅第 319 页的 “ 设置文本格式 ”。

在文本字段中使用图像

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

将内容显示为 HTML 文本的另一个好处是可以在文本字段中包括图像。可以使用 img 标签引用一个本地或远程图像，并使其显示在关联的文本字段内。

以下示例将创建一个名为 myTextBox 的文本字段，并在显示的文本中包括一个内容为眼睛的 JPG 图像，该图像与 SWF 文件存储在同一目录下：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class TextWithImage extends Sprite
    {
        private var myTextBox:TextField;
        private var myText:String = "<p>This is <b>some</b> content to <i>test</i> and <i>see</i></p><p><img
src='eye.jpg' width='20' height='20'></p><p>what can be rendered.</p><p>You should see an eye image and some
<u>HTML</u> text.</p>";

        public function TextWithImage()
        {
            myTextBox.width = 200;
            myTextBox.height = 200;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.border = true;

            addChild(myTextBox);
            myTextBox.htmlText = myText;
        }
    }
}
```

img 标签支持 JPEG、 GIF、 PNG 和 SWF 文件。

在文本字段中滚动文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在许多情况下，文本可能比显示该文本的文本字段长。或者，某个输入字段允许用户输入比字段一次可显示的文本内容更多的文本。您可以使用 flash.text.TextField 类的与滚动相关的属性来管理过长的内容 （垂直或水平方向）。

与滚动有关的属性包括 TextField.scrollV、 TextField.scrollH、 maxScrollV 和 maxScrollH。可使用这些属性来响应鼠标单击或按键等事件。

以下示例将创建一个已设置大小的文本字段，其中包含的文本内容超过了该字段一次可以显示的内容。当用户单击文本字段时，文本垂直滚动。

```
package
{
    import flash.display.Sprite;
    import flash.text.*;
    import flash.events.MouseEvent;

    public class TextScrollExample extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myText:String = "Hello world and welcome to the show. It's really nice to meet you.
Take your coat off and stay a while. OK, show is over. Hope you had fun. You can go home now. Don't forget
to tip your waiter. There are mints in the bowl by the door. Thank you. Please come again.";

        public function TextScrollExample()
        {
            myTextBox.text = myText;
            myTextBox.width = 200;
            myTextBox.height = 50;
            myTextBox.multiline = true;
            myTextBox.wordWrap = true;
            myTextBox.background = true;
            myTextBox.border = true;

            var format:TextFormat = new TextFormat();
            format.font = "Verdana";
            format.color = 0xFF0000;
            format.size = 10;

            myTextBox.defaultTextFormat = format;
            addChild(myTextBox);
            myTextBox.addEventListener(MouseEvent.MOUSE_DOWN, mouseDownScroll);
        }

        public function mouseDownScroll(event:MouseEvent):void
        {
            myTextBox.scrollV++;
        }
    }
}
```

选择和操作文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以选择动态文本或输入文本。由于 TextField 类的文本选择属性和方法使用索引位置来设置要操作的文本的范围，因此即使不知道内容，您也可以以编程方式选择动态文本或输入文本。

注：在 Flash Professional 中，如果对静态文本字段选择了可选选项，则导出并置于显示列表中的文本字段为常规的动态文本字段。

选择文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下， flash.text.TextField.selectable 属性为 true，您可以使用 setSelection() 方法以编程方式选择文本。

例如，您可以将某个文本字段中的特定文本设置成用户单击该文本字段时处于选定状态：

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS is selected.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, selectText);

function selectText(event:MouseEvent):void
{
    myTextField.setSelection(49, 65);
}
```

同样，如果您想让文本字段中的文本一开始显示时就处于选定状态，可以创建一个在向显示列表中添加该文本字段时调用的事件处理函数。

捕获用户选择的文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

TextField 的 selectionBeginIndex 和 selectionEndIndex 属性可用于捕获用户当前选择的任何内容，这两个属性为 " 只读 " 属性，因此不能设置为以编程方式选择文本。此外，输入文本字段也可以使用 caretIndex 属性。

例如，以下代码将跟踪用户所选文本的索引值：

```
var myTextField:TextField = new TextField();
myTextField.text = "Please select the TEXT IN ALL CAPS to see the index values for the first and last
letters.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.MOUSE_UP, selectText);

function selectText(event:MouseEvent):void
{
    trace("First letter index position: " + myTextField.selectionBeginIndex);
    trace("Last letter index position: " + myTextField.selectionEndIndex);
}
```

您可以对所选内容应用 TextFormat 对象属性的集合来更改文本的外观。有关将 TextFormat 属性的集合应用于选定文本的详细信息，请参阅第 322 页的 " 设置文本字段内文本范围的格式 "。

捕获文本输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下，文本字段的 type 属性设置为 dynamic。如果使用 TextFieldType 类将 type 属性设置为 input，则可以收集用户输入并保存该值以便在应用程序的其他部分使用。对于表单以及希望用户定义可用于程序中其他位置的文本值的任何应用程序而言，输入文本字段都十分有用。

例如，以下代码会创建一个名为 myTextBox 的输入文本字段。当用户在字段中输入文本时，会触发 textInput 事件。名为 textInputCapture 的事件处理函数会捕获输入的文本字符串，并将其赋予一个变量。 Flash Player 或 AIR 会在另一个名为 myOutputBox 的文本字段中显示新文本。

```
package
{
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.text.*;
    import flash.events.*;

    public class CaptureUserInput extends Sprite
    {
        private var myTextBox:TextField = new TextField();
        private var myOutputBox:TextField = new TextField();
        private var myText:String = "Type your text here.";

        public function CaptureUserInput()
        {
            captureText();
        }

        public function captureText():void
        {
            myTextBox.type = TextFieldType.INPUT;
            myTextBox.background = true;
            addChild(myTextBox);
            myTextBox.text = myText;
            myTextBox.addEventListener(TextEvent.TEXT_INPUT, textInputCapture);
        }

        public function textInputCapture(event:TextEvent):void
        {
            var str:String = myTextBox.text;
            createOutputBox(str);
        }

        public function createOutputBox(str:String):void
        {
            myOutputBox.background = true;
            myOutputBox.x = 200;
            addChild(myOutputBox);
            myOutputBox.text = str;
        }

    }
}
```

限制文本输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

由于输入文本字段经常用于表单或应用程序中的对话框，所以您可能想要限制用户在文本字段中输入的字符的类型，或者甚至希望将文本隐藏（例如，密码文本）。可以设置 flash.text.TextField 类的 displayAsPassword 属性和 restrict 属性来控制用户输入。

displayAsPassword 属性只是在用户键入文本时将其隐藏（显示为一系列星号）。当 displayAsPassword 设置为 true 时，“ 剪切 ” 和 “ 复制 ” 命令及其对应的键盘快捷键将不起作用。如下例所示，为 displayAsPassword 属性赋值的过程与为其他属性 （如背景和颜色）赋值类似:

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

restrict 属性更复杂些，您必须指定允许用户在输入文本字段中键入的字符。可以允许特定字母、数字或字母、数字和字符的范围。以下代码只允许用户在文本字段中输入大写字母 （不包括数字或特殊字符）：

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0 使用连字符来定义范围，使用尖号来定义被排除的字符。有关定义输入文本字段中的受限内容的详细信息，请参阅 “ActionScript 3.0 参考 ” 中的 flash.text.TextField.restrict 属性条目。

注：如果使用 flash.text.TextField.restrict 属性，运行时会将受限字母自动转换为允许的大小写。如果使用 fl.text.TLFTextField.restrict 属性 （即如果使用 TLF 文本字段），运行时将忽略受限字母。

设置文本格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以编程方式设置文本显示的格式设置有多种方式。可以直接在 TextField 实例中设置属性，例如， TextFIeld.thickness、TextField.textColor 和 TextField.textHeight 属性。也可以使用 htmlText 属性指定文本字段的内容，并使用受支持的 HTML 标签，如 b、i 和 u。但是您也可以将 TextFormat 对象应用于包含纯文本的文本字段，或将 StyleSheet 对象应用于包含 htmlText 属性的文本字段。使用 TextFormat 和 StyleSheet 对象可以对整个应用程序的文本外观提供最有力的控制和最佳的一致性。可以定义 TextFormat 或 StyleSheet 对象并将其应用于应用程序中的部分或所有文本字段。

指定文本格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以使用 TextFormat 类设置多个不同的文本显示属性，并将它们应用于 TextField 对象的整个内容或一定范围的文本。

以下示例对整个 TextField 对象应用一个 TextFormat 对象，并对 TextField 对象中一定范围的文本应用另一个 TextFormat 对象：

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

TextField.setTextFormat() 方法只影响已显示在文本字段中的文本。如果 TextField 中的内容发生更改，则应用程序可能需要重新调用 TextField.setTextFormat() 方法以便重新应用格式设置。您也可以设置 TextField 的 defaultTextFormat 属性来指定用户输入文本所用的格式。

应用层叠样式表

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

文本字段可以包含纯文本或 HTML 格式的文本。纯文本存储在实例的 text 属性中，而 HTML 文本存储在 htmlText 属性中。

您可以使用 CSS 样式声明来定义可应用于多种不同文本字段的文本样式。CSS 样式声明可以在应用程序代码中进行创建，也可以在运行时从外部 CSS 文件中加载。

flash.text.StyleSheet 类用于处理 CSS 样式。 StyleSheet 类可识别有限的 CSS 属性集合。有关 StyleSheet 类支持的样式属性的详细列表，请参阅 "ActionScript 3.0 参考 " 中的 flash.textStylesheet 条目。

如以下示例所示，您可以在代码中创建 CSS，并使用 StyleSheet 对象对 HTML 文本应用这些样式：

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

创建 StyleSheet 对象后，示例代码创建一个简单对象以容纳一组样式声明属性。然后该代码调用 StyleSheet.setStyle() 方法，该方法将名为 ".darkred" 的新样式添加到样式表中。接着，代码通过将 StyleSheet 对象分配给 TextField styleSheet 属性来应用样式表格式设置。

要使 CSS 样式生效，应在设置 htmlText 属性之前对 TextField 对象应用样式表。

根据设计，带有样式表的文本字段是不可编辑的。如果您有一个输入文本字段并为其分配一个样式表，则该文本字段将显示样式表的属性，但不允许用户在其中输入新的文本。而且，您也无法在分配有样式表的文本字段上使用以下 ActionScript API：

- TextField.replaceText() 方法
- TextField.replaceSelectedText() 方法
- TextField.defaultTextFormat 属性
- TextField.setTextFormat() 方法

如果某个文本字段已经分配了一个样式表，但后来将 TextField.styleSheet 属性设置为 null，则 TextField.text 和 TextField.htmlText 属性的内容会向它们的内容中添加标签和属性，以结合先前分配的样式表设定的格式。若要保留原始 htmlText 属性，应在将样式表设置为 null 之前将其保存在变量中。

加载外部 CSS 文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用于设置格式的 CSS 方法的功能更加强大，您可以在运行时从外部文件加载 CSS 信息。当 CSS 数据位于应用程序本身以外时，您可以更改应用程序中的文本的可视样式，而不必更改 ActionScript 3.0 源代码。部署完应用程序后，可以通过更改外部 CSS 文件来更改应用程序的外观，而不必重新部署应用程序 SWF 文件。

StyleSheet.parseCSS() 方法可将包含 CSS 数据的字符串转换为 StyleSheet 对象中的样式声明。以下示例显示如何读取外部 CSS 文件并对 TextField 对象应用其样式声明。

首先，下面是要加载的 CSS 文件 （名为 example.css）的内容：

```
p {
    font-family: Times New Roman, Times, _serif;
    font-size: 14;
}

h1 {
    font-family: Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
}

.bluetext {
    color: #0000CC;
}
```

接下来是加载该 example.css 文件并对 TextField 内容应用样式的类的 ActionScript 代码：

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLLoader;
    import flash.net.URLRequest;
    import flash.text.StyleSheet;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;

    public class CSSFormattingExample extends Sprite
    {
        var loader:URLLoader;
        var field:TextField;
        var exampleText:String = "<h1>This is a headline</h1>" +
            "<p>This is a line of text. <span class='bluetext'>" +
            "This line of text is colored blue.</span></p>";

        public function CSSFormattingExample():void
        {
            field = new TextField();
            field.width = 300;
            field.autoSize = TextFieldAutoSize.LEFT;
            field.wordWrap = true;
            addChild(field);

            var req:URLRequest = new URLRequest("example.css");

            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            var sheet:StyleSheet = new StyleSheet();
            sheet.parseCSS(loader.data);
            field.styleSheet = sheet;
            field.htmlText = exampleText;
        }
    }
}
```

加载 CSS 数据后，会执行 onCSSFileLoaded() 方法并调用 StyleSheet.parseCSS() 方法，将样式声明传送给 StyleSheet 对象。

设置文本字段内文本范围的格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

flash.text.TextField 类的一个很有用的方法是 setTextFormat() 方法。使用 setTextFormat()，您可以将特定属性分配给文本字段的部分内容以响应用户输入，例如，需要提醒用户必须输入特定条目的表单，或在用户选择部分文本时提醒用户更改文本字段内文本段落小节的重点的表单。

以下示例对某一范围的字符使用 TextField.setTextFormat()，以在用户单击文此本字段时更改 myTextField 的部分内容的外观：

```
var myTextField:TextField = new TextField();
myTextField.text = "No matter where you click on this text field the TEXT IN ALL CAPS changes format.";
myTextField.autoSize = TextFieldAutoSize.LEFT;
addChild(myTextField);
addEventListener(MouseEvent.CLICK, changeText);

var myformat:TextFormat = new TextFormat();
myformat.color = 0xFF0000;
myformat.size = 18;
myformat.underline = true;

function changeText(event:MouseEvent):void
{
    myTextField.setTextFormat(myformat, 49, 65);
}
```

高级文本呈现

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 在 flash.text 包中提供多个类来控制所显示文本的属性，包括嵌入字体、消除锯齿设置、 alpha 通道控制及其他特定设置。“ActionScript 3.0 参考 ”提供了对这些类和属性（其中包括 CSMSettings、Font 和 TextRenderer 类）的详细描述。

使用嵌入字体

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当您在应用程序中为 TextField 指定特定字体时， Flash Player 或 AIR 会查找同名的设备字体 （位于用户计算机上的一种字体）。如果在系统上没有找到该字体，或者用户的字体版本与具有该名称的字体略有差异，则文本显示外观会与预想的情况差别很大。默认情况下，文本显示为 Times Roman 字体。

若要确保用户看到完全正确的字体，您可以将该字体嵌入到应用程序 SWF 文件中。嵌入字体有很多好处：

- 嵌入字体字符是消除锯齿的，特别是对于较大的文本，该字体可以使文本边缘看起来更平滑。
- 可以旋转使用嵌入字体的文本。
- 嵌入字体文本可以产生透明或半透明效果。
- 可以对嵌入字体使用字距调整的 CSS 样式。

使用嵌入字体的最大限制是嵌入字体会增加文件大小或应用程序的下载大小。

将字体文件嵌入到应用程序 SWF 文件中的具体方法因开发环境而异。

嵌入字体后，可以确保 TextField 使用正确的嵌入字体：

- 将 TextField 的 embedFonts 属性设置为 true。
- 创建一个 TextFormat 对象，将其 fontFamily 属性设置为嵌入字体的名称，并对 TextField 应用 TextFormat 对象。指定嵌入字体时， fontFamily 属性应只包含一个名称；该名称不能是用逗号分隔的由多个字体名称构成的列表。
- 如果使用 CSS 样式为 TextField 或组件设置字体，请将 font-family CSS 属性设置为嵌入字体的名称。如果要指定一种嵌入字体，则 font-family 属性必须包含单一名称，而不能是多个名称的列表。

在 **Flash** 中嵌入字体

Flash Professional 允许您嵌入系统中已安装的几乎所有字体，包括 TrueType 字体和 Type 1 Postscript 字体。

可以利用多种方法将字体嵌入应用程序，其中包括：

- 在舞台上设置 TextField 的字体和样式属性，然后单击 " 嵌入字体 " 复选框
- 创建并引用字体元件
- 创建并使用包含嵌入字体元件的运行时共享库

有关如何在应用程序中嵌入字体的详细信息，请参阅 《使用 Flash》中的 " 动态或输入文本字段的嵌入字体 "。

在 **Flex** 中嵌入字体

可以利用多种方法将字体嵌入 Flex 应用程序，其中包括：

- 在脚本中使用 [Embed] 元数据标签
- 使用 @font-face 样式声明
- 建立此类字体的类并使用 [Embed] 标签将其嵌入。

在 Flex 应用程序中只能直接嵌入 TrueType 字体。 Type 1 Postscript 等其他格式的字体可以先嵌入到使用 Flash Professional 的 SWF 文件，然后便可在 Flex 应用程序中使用此 SWF 文件。有关在 Flex 中使用 SWF 文件中的嵌入字体的详细信息，请参阅 《使用 Flex 4》中的 " 嵌入 SWF 文件中的字体 "。

更多帮助主题

嵌入字体以实现一致的文本外观

Peter deHaan：嵌入字体

Divillysausages.com：AS3 字体嵌入主类

控制清晰度、粗细和消除锯齿

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下，在文本调整大小、更改颜色或在不同背景上显示时， Flash Player 或 AIR 可以确定文本显示控件的设置 （如清晰度、粗细和消除锯齿）。在某些情况下，如文本很小、很大或显示在各种特别的背景上时，您可能需要保持对这些设置的控制。可以使用 flash.text.TextRenderer 类及其相关类 （如 CSMSettings 类）来覆盖 Flash Player 或 AIR 的设置。使用这些类可以精确控制嵌入文本的呈示品质。有关嵌入字体的详细信息，请参阅第 322 页的 " 使用嵌入字体 "。

注：为了设置清晰度、粗细或 gridFitType 属性，或者使用 TextRenderer.setAdvancedAntiAliasingTable() 方法，flash.text.TextField.antiAliasType 属性的值必须是 AntiAliasType.ADVANCED。

以下示例使用名为 myFont 的嵌入字体对显示的文本应用自定义连续笔触调制 (CSM) 属性和格式设置。用户单击显示的文本时， Flash Player 或 Adobe AIR 会应用自定义设置：

```
var format:TextFormat = new TextFormat();
format.color = 0x336699;
format.size = 48;
format.font = "myFont";

var myText:TextField = new TextField();
myText.embedFonts = true;
myText.autoSize = TextFieldAutoSize.LEFT;
myText.antiAliasType = AntiAliasType.ADVANCED;
myText.defaultTextFormat = format;
myText.selectable = false;
myText.mouseEnabled = true;
myText.text = "Hello World";
addChild(myText);
myText.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:Event):void
{
    var myAntiAliasSettings = new CSMSettings(48, 0.8, -0.8);
    var myAliasTable:Array = new Array(myAntiAliasSettings);
    TextRenderer.setAdvancedAntiAliasingTable("myFont", FontStyle.ITALIC, TextColorType.DARK_COLOR,
myAliasTable);
}
```

使用静态文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

仅在 Flash Professional 中创建静态文本。不能使用 ActionScript 以编程方式对静态文本进行实例化。静态文本非常适用于简短且不会更改 （动态文本则会更改）的文本。可以将静态文本看作一种图形元素，类似于 Flash Professional 中在舞台上绘制的圆形或正方形。由于静态文本比动态文本受到更多的限制，ActionScript 3.0 允许使用 StaticText 类读取静态文本的属性值。另外还可使用 TextSnapshot 类从静态文本中读取值。

使用 StaticText 类访问静态文本字段

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通常，可使用 Flash Professional 的 " 动作 " 面板中的 flash.text.StaticText 类与舞台上放置的静态文本实例交互。也可以在与包含静态文本的 SWF 文件进行交互的 ActionScript 文件中执行类似工作。但是这两种情况下都不能以编程方式对静态文本实例进行实例化。在 Flash Professional 中创建静态文本。

若要创建对现有静态文本字段的引用，可以遍历显示列表中的项目并分配一个变量。例如：

```
for (var i = 0; i < this.numChildren; i++) {
var displayitem:DisplayObject = this.getChildAt(i);
if (displayitem instanceof StaticText) {
trace("a static text field is item " + i + " on the display list");
        var myFieldLabel:StaticText = StaticText(displayitem);
        trace("and contains the text: " + myFieldLabel.text);
}
}
```

引用某个静态文本字段之后，您可以在 ActionScript 3.0 中使用该字段的属性。下面的代码附加到时间轴上的一个帧，并假设一个静态文本引用分配有一个名为 myFieldLabel 的变量。相对于 myFieldLabel 的 x 和 y 值放置名为 myField 的动态文本字段，并再次显示 myFieldLabel 的值。

```
var myField:TextField = new TextField();
addChild(myField);
myField.x = myFieldLabel.x;
myField.y = myFieldLabel.y + 20;
myField.autoSize = TextFieldAutoSize.LEFT;
myField.text = "and " + myFieldLabel.text
```

使用 TextSnapshot 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果要以编程方式使用现有静态文本实例，可以使用 flash.text.TextSnapshot 类来与 flash.display.DisplayObjectContainer 的 textSnapshot 属性配合工作。也就是说，通过 DisplayObjectContainer.textSnapshot 属性创建 TextSnapshot 实例。然后，可以将方法应用于该实例，以检索值或选择部分静态文本。

例如，请在舞台上放置一个包含文本 “TextSnapshot Example” 的静态文本字段。将下面的 ActionScript 添加到时间轴中的第 1 帧：

```
var mySnap:TextSnapshot = this.textSnapshot;
var count:Number = mySnap.charCount;
mySnap.setSelected(0, 4, true);
mySnap.setSelected(1, 2, false);
var myText:String = mySnap.getSelectedText(false);
trace(myText);
```

如果要在应用程序的其他部分将该文本作为值使用，则 TextSnapshot 类对于从所加载的 SWF 文件中的静态文本字段中获取文本非常有用。

TextField 示例：报纸风格的文本格式设置

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

“ 新闻布局 ” 示例设置文本的格式，使文本的外观看起来有点象印刷报纸中的素材。输入文本可以包含标题、副标题和素材正文。在给定显示宽度和高度的情况下，此 “ 新闻布局 ” 示例将会设置标题和副标题的格式，使其占据整个显示区域的宽度。素材文本分布在两列或更多列中。

此示例演示以下 ActionScript 编程技巧：

- 扩展 TextField 类
- 加载并应用外部 CSS 文件
- 将 CSS 样式转换为 TextFormat 对象
- 使用 TextLineMetrics 类获取有关文本显示大小的信息

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。 “ 新闻布局 ” 应用程序文件位于 Samples/NewsLayout 文件夹中。该应用程序包含以下文件：

文件	说明
NewsLayout.mxml 或 NewsLayout.fla	适用于 Flex (MXML) 或 Flash (FLA) 的应用程序的用户界面。
com/example/programmingas3/newslayout/StoryLayoutComponent.as	放置 StoryLayout 实例的 Flex UIComponent 类。
com/example/programmingas3/newslayout/StoryLayout.as	排列用于显示的所有新闻素材组件的主要 ActionScript 类。
com/example/programmingas3/newslayout/FormattedTextField.as	管理本身的 TextFormat 对象的 TextField 类的子类。
com/example/programmingas3/newslayout/HeadlineTextField.as	调整字体大小以适合需要的宽度的 FormattedTextField 类的子类。
com/example/programmingas3/newslayout/MultiColumnTextField.as	在两列或多列之间拆分文本的 ActionScript 类。
story.css	为布局定义文本样式的 CSS 文件。

读取外部 CSS 文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

“ 新闻布局 ” 应用程序开始时读取本地 XML 文件中的素材文本。然后，它读取提供标题、副标题和主体文本的格式设置信息的外部 CSS 文件。

CSS 文件定义三种样式：用于素材的标准段落样式和分别用于标题和副标题的 h1 和 h2 样式。

```
p {
    font-family: Georgia, "Times New Roman", Times, _serif;
    font-size: 12;
    leading: 2;
    text-align: justify;
    indent: 24;
}

h1 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 20;
    font-weight: bold;
    color: #000099;
    text-align: left;
}

h2 {
    font-family: Verdana, Arial, Helvetica, _sans;
    font-size: 16;
    font-weight: normal;
    text-align: left;
}
```

用于读取外部 CSS 文件的方法与第 320 页的 “ 加载外部 CSS 文件 ”中所述的方法相同。加载 CSS 文件后，应用程序执行 onCSSFileLoaded() 方法，如下所示。

```
public function onCSSFileLoaded(event:Event):void
{
    this.sheet = new StyleSheet();
    this.sheet.parseCSS(loader.data);

    h1Format = getTextStyle("h1", this.sheet);
    if (h1Format == null)
    {
        h1Format = getDefaultHeadFormat();
    }
    h2Format = getTextStyle("h2", this.sheet);
    if (h2Format == null)
    {
        h2Format = getDefaultHeadFormat();
        h2Format.size = 16;
    }
    pFormat = getTextStyle("p", this.sheet);
    if (pFormat == null)
    {
        pFormat = getDefaultTextFormat();
        pFormat.size = 12;
    }
    displayText();
}
```

onCSSFileLoaded() 方法创建一个 StyleSheet 对象并使之分析输入 CSS 数据。素材的主体文本将显示在 MultiColumnTextField 对象中，该对象可以直接使用 StyleSheet 对象。不过，标题字段使用 HeadlineTextField 类，该类使用 TextFormat 对象进行格式设置。

onCSSFileLoaded() 方法调用两次 getTextStyle() 方法，将 CSS 样式声明转换为 TextFormat 对象，以便与两个 HeadlineTextField 对象中的每个对象配合使用。

```
public function getTextStyle(styleName:String, ss:StyleSheet):TextFormat
{
    var format:TextFormat = null;

    var style:Object = ss.getStyle(styleName);
    if (style != null)
    {
        var colorStr:String = style.color;
        if (colorStr != null && colorStr.indexOf("#") == 0)
        {
            style.color = colorStr.substr(1);
        }
        format = new TextFormat(style.fontFamily,
                        style.fontSize,
                        style.color,
                        (style.fontWeight == "bold"),
                        (style.fontStyle == "italic"),
                        (style.textDecoration == "underline"),
                        style.url,
                        style.target,
                        style.textAlign,
                        style.marginLeft,
                        style.marginRight,
                        style.indent,
                        style.leading);

        if (style.hasOwnProperty("letterSpacing"))
        {
            format.letterSpacing = style.letterSpacing;
        }
    }
    return format;
}
```

CSS 样式声明和 TextFormat 对象的属性名称和属性值的含义不同。 getTextStyle() 方法可以将 CSS 属性值转换为 TextFormat 对象预期的值。

在页面上排列素材元素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

StoryLayout 类可以设置标题、副标题和主体文本字段的格式并将这些字段的布局排列为报纸样式。 displayText() 方法一开始会创建并放置各个字段。

```
public function displayText():void
{
    headlineTxt = new HeadlineTextField(h1Format);
    headlineTxt.wordWrap = true;
    headlineTxt.x = this.paddingLeft;
    headlineTxt.y = this.paddingTop;
    headlineTxt.width = this.preferredWidth;
    this.addChild(headlineTxt);

    headlineTxt.fitText(this.headline, 1, true);

    subtitleTxt = new HeadlineTextField(h2Format);
    subtitleTxt.wordWrap = true;
    subtitleTxt.x = this.paddingLeft;
    subtitleTxt.y = headlineTxt.y + headlineTxt.height;
    subtitleTxt.width = this.preferredWidth;
    this.addChild(subtitleTxt);

    subtitleTxt.fitText(this.subtitle, 2, false);

    storyTxt = new MultiColumnText(this.numColumns, 20,
                        this.preferredWidth, 400, true, this.pFormat);
    storyTxt.x = this.paddingLeft;
    storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
    this.addChild(storyTxt);

    storyTxt.text = this.content;
...
```

将一个字段的 y 属性设置为等于上一个字段的 y 属性加上其自身高度，这样即可将每个字段放置在上一个字段的下面。由于 HeadlineTextField 对象和 MultiColumnTextField 对象可以更改其高度以适应内容，因此需要进行这种动态位置计算。

更改字体大小以适合字段大小

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在给定要显示内容的宽度 （以像素为单位）和最多行数的情况下， HeadlineTextField 会更改字体大小以使文本适合字段大小。如果文本短，字体大小很大，就会生成 tabloid 样式的标题。如果文本很长，那么字体大小较小。

下面所示的 HeadlineTextField.fitText() 方法的作用就是调整字体大小：

```
public function fitText(msg:String, maxLines:uint = 1, toUpper:Boolean = false, targetWidth:Number = -
1):uint
{
    this.text = toUpper ? msg.toUpperCase() : msg;

    if (targetWidth == -1)
    {
        targetWidth = this.width;
    }

    var pixelsPerChar:Number = targetWidth / msg.length;

    var pointSize:Number = Math.min(MAX_POINT_SIZE, Math.round(pixelsPerChar * 1.8 * maxLines));

    if (pointSize < 6)
    {
        // the point size is too small
        return pointSize;
    }
```

```
    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(--pointSize, maxLines);
    }
    else
    {
        return growText(pointSize, maxLines);
    }
}

public function growText(pointSize:Number, maxLines:uint = 1):Number
{
    if (pointSize >= MAX_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize + 1);

    if (this.numLines > maxLines)
    {
        // set it back to the last size
        this.changeSize(pointSize);
        return pointSize;
    }
    else
    {
        return growText(pointSize + 1, maxLines);
    }
}

public function shrinkText(pointSize:Number, maxLines:uint=1):Number
{
    if (pointSize <= MIN_POINT_SIZE)
    {
        return pointSize;
    }

    this.changeSize(pointSize);

    if (this.numLines > maxLines)
    {
        return shrinkText(pointSize - 1, maxLines);
    }
    else
    {
        return pointSize;
    }
}
```

HeadlineTextField.fitText() 方法使用简单的递归技术来调整字体大小。首先，该方法推测文本中每个字符的平均像素数，并据此计算起始点大小。然后，它更改字体大小并检查文本中的文字是否换行，从而产生比最大值更多的文本行。如果文本行过多，则它会调用 shrinkText() 方法减小字体大小并重试。如果文本行不太多，则它会调用 growText() 方法增大字体大小并重试。当字体大小再增加一点即会产生过多行时，该过程即会停止。

在多列之间拆分文本

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

MultiColumnTextField 类会在多个 TextField 对象中分布文本，然后将这些对象排列成报纸专栏的样式。

MultiColumnTextField() 构造函数首先创建一个由 TextField 对象构成的数组，每列一个对象，如下所示：

```
for (var i:int = 0; i < cols; i++)
{
    var field:TextField = new TextField();
    field.multiline = true;
    field.autoSize = TextFieldAutoSize.NONE;
    field.wordWrap = true;
    field.width = this.colWidth;
    field.setTextFormat(this.format);
    this.fieldArray.push(field);
    this.addChild(field);
}
```

每个 TextField 对象均使用 addChild() 方法添加到该数组中，并添加到显示列表中。

只要 StoryLayout 的 text 属性或 styleSheet 属性发生更改，该对象就会调用 layoutColumns() 方法来重新显示文本。layoutColumns() 方法会调用 getOptimalHeight() 方法，以计算在给定布局宽度内适合所有文本所需的正确像素高度。

```
public function getOptimalHeight(str:String):int
{
    if (field.text == "" || field.text == null)
    {
        return this.preferredHeight;
    }
    else
    {
        this.linesPerCol = Math.ceil(field.numLines / this.numColumns);

        var metrics:TextLineMetrics = field.getLineMetrics(0);
        this.lineHeight = metrics.height;
        var prefHeight:int = linesPerCol * this.lineHeight;

        return prefHeight + 4;
    }
}
```

首先， getOptimalHeight() 方法计算每列的宽度。然后设置数组中第一个 TextField 对象的宽度和 htmlText 属性。getOptimalHeight() 方法使用第一个 TextField 对象计算文本中自动换行文本的总行数，并据此确定每列中应有多少行。接下来，它调用 TextField.getLineMetrics() 方法以检索 TextLineMetrics 对象，该对象包含有关第一行的文本大小的详细信息。TextLineMetrics.height 属性用像素表示文本行的完整高度，包括上缘、下缘和前导。 MultiColumnTextField 对象的最佳高度即为行高度乘以每列行数再加上 4 （TextField 对象顶部边框和底部边框各 2 个像素）。

以下是完整 layoutColumns() 方法的代码：

```
public function layoutColumns():void
{
    if (this._text == "" || this._text == null)
    {
        return;
    }

    var field:TextField = fieldArray[0] as TextField;
    field.text = this._text;
    field.setTextFormat(this.format);

    this.preferredHeight = this.getOptimalHeight(field);

    var remainder:String = this._text;
    var fieldText:String = "";
    var lastLineEndedPara:Boolean = true;

    var indent:Number = this.format.indent as Number;

    for (var i:int = 0; i < fieldArray.length; i++)
    {
        field = this.fieldArray[i] as TextField;

        field.height = this.preferredHeight;
        field.text = remainder;

        field.setTextFormat(this.format);

        var lineLen:int;
        if (indent > 0 && !lastLineEndedPara && field.numLines > 0)
        {
            lineLen = field.getLineLength(0);
            if (lineLen > 0)
            {
                field.setTextFormat(this.firstLineFormat, 0, lineLen);
                }
            }

        field.x = i * (colWidth + gutter);
        field.y = 0;

        remainder = "";
        fieldText = "";

        var linesRemaining:int = field.numLines;
        var linesVisible:int = Math.min(this.linesPerCol, linesRemaining);

        for (var j:int = 0; j < linesRemaining; j++)
        {
            if (j < linesVisible)
            {
                fieldText += field.getLineText(j);
            }
            else
            {
                remainder +=field.getLineText(j);
            }
        }

        field.text = fieldText;

        field.setTextFormat(this.format);
```

```
        if (indent > 0 && !lastLineEndedPara)
        {
            lineLen = field.getLineLength(0);
            if (lineLen > 0)
            {
                field.setTextFormat(this.firstLineFormat, 0, lineLen);
            }
        }

        var lastLine:String = field.getLineText(field.numLines - 1);
        var lastCharCode:Number = lastLine.charCodeAt(lastLine.length - 1);

        if (lastCharCode == 10 || lastCharCode == 13)
        {
        lastLineEndedPara = true;
        }
        else
        {
        lastLineEndedPara = false;
        }

        if ((this.format.align == TextFormatAlign.JUSTIFY) &&
                (i < fieldArray.length - 1))
        {
        if (!lastLineEndedPara)
        {
            justifyLastLine(field, lastLine);
        }
    }
    }
}
```

通过调用 getOptimalHeight() 方法设置 preferredHeight 属性后，layoutColumns() 方法会循环访问各个 TextField 对象，将每个对象的高度设置为 preferredHeight 值。然后， layoutColumns() 方法会为每个字段分布刚好足够的文本行，以使每个字段中都不会发生滚动，并且每个连续字段中的文本均会从前一个字段的文本结束处开始。如果文本对齐样式已设置为 “justify”，则会调用 justifyLastLine() 方法以对齐字段中最后一行文本。否则，最后一行将被视为段落结束行，而不予对齐。

第 22 章：使用 Flash 文本引擎

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

从 Flash Player 10 和 Adobe® AIR™1.5 开始提供的 Adobe® Flash® 文本引擎 (FTE) 为文本度量、格式设置和双向文本的复杂控制提供了底层支持。它提供了改进的文本流和增强的语言支持。尽管可以使用 FTE 创建和管理简单的文本元素，但设计 FTE 的主要目的是为开发人员创建文本处理组件提供基础。因此， Flash 文本引擎采用了更高级的编程技术。要显示简单文本元素，请参阅第 313 页的 " 使用 TextField 类 "。

Text Layout Framework 包括一个基于 FTE 的文本处理组件，提供了一种比较容易的使用其高级功能的方法。Text Layout Framework 是完全内置于 ActionScript 3.0 的可扩展库。您可以使用现有的 TLF 组件，或者使用框架构建您自己的文本组件。有关更多信息，请参阅第 360 页的 " 使用 Text Layout Framework"。

更多帮助主题

flash.text.engine 包

创建和显示文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

组成 Flash 文本引擎的类可用于创建文本、设置文本格式以及控制文本。下面的类是通过 Flash 文本引擎创建和显示文本时所用的基本构造块：

- TextElement/GraphicElement/GroupElement：包含 TextBlock 实例的内容
- ElementFormat：指定 TextBlock 实例内容的格式设置属性
- TextBlock：用于构建文本段落的工厂
- TextLine：依据 TextBlock 创建的文本行

要显示文本，可从字符串创建一个 TextElement 对象，使用 ElementFormat 对象指定格式设置特征。将 TextElement 分配给 TextBlock 对象的 content 属性。通过调用 TextBlock.createTextLine() 方法创建将要显示的文本行。createTextLine() 方法返回一个 TextLine 对象，该对象包含的字符串长度与将适合的指定宽度相同。重复调用此方法，直到整个字符串格式设置为行。当没有更多的行要创建时， TextBlock 对象的 textLineCreationResult 属性将分配值：TextLineCreationResult.COMPLETE。要显示行，请将它们添加到显示列表 （x 和 y 的位置值要适当）。

例如，以下代码使用这些 FTE 类显示 "Hello World！ This is Flash Text Engine!"，显示时使用默认的格式和字体值。在此简单示例中，仅创建了一行文本。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class HelloWorldExample extends Sprite
    {
        public function HelloWorldExample()
        {
            var str = "Hello World! This is Flash Text Engine!";
            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;

            var textLine1:TextLine = textBlock.createTextLine(null, 300);
            addChild(textLine1);
            textLine1.x = 30;
            textLine1.y = 30;
        }
    }
}
```

createTextLine() 的参数指定新行从哪一行开始，以及新行的宽度 （单位为像素）。新行通常从前一行开始，但如果新行是第一行，则为 null。

添加 GraphicElement 和 GroupElement 对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

可以将 GraphicElement 对象分配给 TextBlock 对象以显示图像或图形元素。直接从图形或图像创建一个 GraphicElement 类的实例，然后将该实例分配给 TextBlock.content 属性。按照常规的方法调用 TextBlock.createTextline() 来创建文本行。下面的示例创建两个文本行，一个用 GraphicElement 对象创建，另一个用 TextElement 对象创建。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GraphicElementExample extends Sprite
    {
        public function GraphicElementExample()
        {
            var str:String = "Beware of Dog!";

            var triangle:Shape = new Shape();
            triangle.graphics.beginFill(0xFF0000, 1);
            triangle.graphics.lineStyle(3);
            triangle.graphics.moveTo(30, 0);
            triangle.graphics.lineTo(60, 50);
            triangle.graphics.lineTo(0, 50);
            triangle.graphics.lineTo(30, 0);
            triangle.graphics.endFill();

            var format:ElementFormat = new ElementFormat();
```

```
            format.fontSize = 20;

            var graphicElement:GraphicElement = new GraphicElement(triangle, triangle.width,
triangle.height, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = graphicElement;
            var textLine1:TextLine = textBlock.createTextLine(null, triangle.width);
            textLine1.x = 50;
            textLine1.y = 110;
            addChild(textLine1);

            var textElement:TextElement = new TextElement(str, format);
            textBlock.content = textElement;
            var textLine2 = textBlock.createTextLine(null, 300);
            addChild(textLine2);
            textLine2.x = textLine1.x - 30;
            textLine2.y = textLine1.y + 15;
        }
    }
}
```

您可以创建 GroupElement 对象，以创建一组 TextElement 对象、 GraphicElement 对象和其他 GroupElement 对象。GroupElement 可以分配给 TextBlock 对象的 content 属性。GroupElement() 构造函数的参数是一个 Vector，它指向构成组的文本、图形以及组元素。下面的示例将两个图形元素和一个文本元素组成一组，然后将它们作为一个单元分配给某个文本块。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;
    import flash.display.Shape;
    import flash.display.Graphics;

    public class GroupElementExample extends Sprite
    {
        public function GroupElementExample()
        {
            var str:String = "Beware of Alligators!";

            var triangle1:Shape = new Shape();
            triangle1.graphics.beginFill(0xFF0000, 1);
            triangle1.graphics.lineStyle(3);
            triangle1.graphics.moveTo(30, 0);
            triangle1.graphics.lineTo(60, 50);
            triangle1.graphics.lineTo(0, 50);
            triangle1.graphics.lineTo(30, 0);
            triangle1.graphics.endFill();

            var triangle2:Shape = new Shape();
            triangle2.graphics.beginFill(0xFF0000, 1);
            triangle2.graphics.lineStyle(3);
            triangle2.graphics.moveTo(30, 0);
            triangle2.graphics.lineTo(60, 50);
            triangle2.graphics.lineTo(0, 50);
            triangle2.graphics.lineTo(30, 0);
            triangle2.graphics.endFill();
```

```
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 20;
            var graphicElement1:GraphicElement = new GraphicElement(triangle1, triangle1.width,
triangle1.height, format);
            var textElement:TextElement = new TextElement(str, format);
            var graphicElement2:GraphicElement = new GraphicElement(triangle2, triangle2.width,
triangle2.height, format);
            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>();
            groupVector.push(graphicElement1, textElement, graphicElement2);
            var groupElement = new GroupElement(groupVector);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = groupElement;
            var textLine:TextLine = textBlock.createTextLine(null, 800);
            addChild(textLine);
            textLine.x = 100;
            textLine.y = 200;
        }
    }
}
```

替换文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

您可以调用 TextElement.replaceText() 来替换您分配给 TextBlock.content 属性的 TextElement 中的文本，从而替换 TextBlock 实例中的文本。

以下示例首先使用 replaceText() 在行首插入文本，然后在行尾追加文本，最后在行的中间位置替换文本。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class ReplaceTextExample extends Sprite
    {
        public function ReplaceTextExample()
        {

            var str:String = "Lorem ipsum dolor sit amet";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription);
            format.fontSize = 14;
            var textElement:TextElement = new TextElement(str, format);
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = textElement;
            createLine(textBlock, 10);
            textElement.replaceText(0, 0, "A text fragment: ");
            createLine(textBlock, 30);
            textElement.replaceText(43, 43, "...");
            createLine(textBlock, 50);
            textElement.replaceText(23, 28, "(ipsum)");
            createLine(textBlock, 70);
        }

        function createLine(textBlock:TextBlock, y:Number):void {
            var textLine:TextLine = textBlock.createTextLine(null, 300);
            textLine.x = 10;
            textLine.y = y;
            addChild(textLine);
        }
    }
}
```

该 replaceText() 方法将 beginIndex 和 endIndex 参数指定的文本替换为 newText 参数指定的文本。如果 beginIndex 和 endIndex 参数的值相同，replaceText() 将在相应的位置插入指定的文本。否则，将用新文本替换 beginIndex 和 endIndex 所指定的字符。

处理 FTE 中的事件

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

您可以将事件侦听器添加到 TextLine 实例中，就像添加到其他显示对象一样。例如，您可以检测用户何时将鼠标移动到某个文本行的上方，或者用户何时单击此行。下面的示例检测这两种事件。如果将鼠标移动到此行的上方，光标将变为按钮光标，并且如果单击此行，此行将会变色。

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventHandlerExample extends Sprite
    {
        var textBlock:TextBlock = new TextBlock();

        public function EventHandlerExample():void
        {
            var str:String = "I'll change color if you click me.";
            var fontDescription:FontDescription = new FontDescription("Arial");
            var format:ElementFormat = new ElementFormat(fontDescription, 18);
            var textElement = new TextElement(str, format);
            textBlock.content = textElement;
            createLine(textBlock);
        }

        private function createLine(textBlock:TextBlock):void
        {
            var textLine:TextLine = textBlock.createTextLine(null, 500);
            textLine.x = 30;
            textLine.y = 30;
            addChild(textLine);
            textLine.addEventListener("mouseOut", mouseOutHandler);
            textLine.addEventListener("mouseOver", mouseOverHandler);
            textLine.addEventListener("click", clickHandler);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
            Mouse.cursor = "arrow";
        }

        function clickHandler(event:MouseEvent):void {
            if(textBlock.firstLine)
                removeChild(textBlock.firstLine);
            var newFormat:ElementFormat = textBlock.content.elementFormat.clone();
```

```
            switch(newFormat.color)
            {
                case 0x000000:
                    newFormat.color = 0xFF0000;
                    break;
                case 0xFF0000:
                    newFormat.color = 0x00FF00;
                    break;
                case 0x00FF00:
                    newFormat.color = 0x0000FF;
                    break;
                case 0x0000FF:
                    newFormat.color = 0x000000;
                    break;
            }
            textBlock.content.elementFormat = newFormat;
            createLine(textBlock);
        }
    }
}
```

镜像事件

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

另外，还可将文本块或文本块局部的事件映射到事件调度程序。首先，创建一个 EventDispatcher 实例，然后将其分配给 TextElement 实例的 eventMirror 属性。如果文本块由一个文本元素组成，那么文本引擎将映射整个文本块的事件。如果文本块由多个文本元素组成，那么文本引擎只映射设置了 eventMirror 属性的 TextElement 实例的事件。下面示例中的文本由三个元素组成：单词 “Click”、单词 “here” 以及字符串 “to see me in italic”。此示例将事件调度程序分配给第二个文本元素，即单词 “here”，然后添加事件侦听器 clickHandler() 方法。该 clickHandler() 方法将文本更改为斜体。另外，它还将第三个文本元素的内容替换为 “Click here to see me in normal font!”。

```
package
{
    import flash.text.engine.*;
    import flash.ui.Mouse;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.EventDispatcher;

    public class EventMirrorExample extends Sprite
    {
        var fontDescription:FontDescription = new FontDescription("Helvetica", "bold");
        var format:ElementFormat = new ElementFormat(fontDescription, 18);
        var textElement1 = new TextElement("Click ", format);
        var textElement2 = new TextElement("here ", format);
        var textElement3 = new TextElement("to see me in italic! ", format);
        var textBlock:TextBlock = new TextBlock();

        public function EventMirrorExample()
        {
            var myEvent:EventDispatcher = new EventDispatcher();

            myEvent.addEventListener("click", clickHandler);
            myEvent.addEventListener("mouseOut", mouseOutHandler);
            myEvent.addEventListener("mouseOver", mouseOverHandler);

            textElement2.eventMirror=myEvent;

            var groupVector:Vector.<ContentElement> = new Vector.<ContentElement>;
```

```
            groupVector.push(textElement1, textElement2, textElement3);
            var groupElement:GroupElement = new GroupElement(groupVector);

            textBlock.content = groupElement;
            createLines(textBlock);
        }

        private function clickHandler(event:MouseEvent):void
        {
            var newFont:FontDescription = new FontDescription();
            newFont.fontWeight = "bold";

            var newFormat:ElementFormat = new ElementFormat();
            newFormat.fontSize = 18;
            if(textElement3.text == "to see me in italic! ") {
                newFont.fontPosture = FontPosture.ITALIC;
                textElement3.replaceText(0,21, "to see me in normal font! ");
            }
            else {
                newFont.fontPosture = FontPosture.NORMAL;
                textElement3.replaceText(0, 26, "to see me in italic! ");
            }
            newFormat.fontDescription = newFont;
            textElement1.elementFormat = newFormat;
            textElement2.elementFormat = newFormat;
            textElement3.elementFormat = newFormat;
            createLines(textBlock);
        }

        private function mouseOverHandler(event:MouseEvent):void
        {
            Mouse.cursor = "button";
        }

        private function mouseOutHandler(event:MouseEvent):void
        {
                Mouse.cursor = "arrow";
        }

        private function createLines(textBlock:TextBlock):void
        {
            if(textBlock.firstLine)
                removeChild (textBlock.firstLine);
            var textLine:TextLine = textBlock.createTextLine (null, 300);
            textLine.x = 15;
            textLine.y = 20;
            addChild (textLine);
        }
    }
}
```

mouseOverHandler() 和 mouseOutHandler() 函数在光标位于单词 “here” 上方时将光标设置为按钮光标，在光标不在 “here” 上方时将光标恢复为箭头。

设置文本格式

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

TextBlock 对象是用于创建文本行的工厂。 TextBlock 的内容是通过 TextElement 对象分配的。 ElementFormat 对象负责处理文本的格式设置。 ElementFormat 类定义基线对齐、字距调整、间距、文本旋转以及字体大小、颜色和大小写等属性。它还包含 FontDescription，相关内容在第 345 页的 " 使用字体 " 中详细介绍。

使用 ElementFormat 对象

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

ElementFormat 对象的构造函数可以接受一长串可选参数中的任何一个参数，包括 FontDescription。也可以在构造函数之外设置这些属性。下面的示例演示了在定义和显示简单文本行时各个对象的关系：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function ElementFormatExample()
        {
            fd.fontName = "Garamond";
            ef = new ElementFormat(fd);
            ef.fontSize = 30;
            ef.color = 0xFF0000;
            str = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

字体颜色和透明度 (alpha)

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

ElementFormat 对象的 color 属性设置字体颜色。该值是一个表示 RGB 颜色分量的整数；例如， 0xFF0000 表示红色，0x00FF00 表示绿色。默认值是黑色 (0x000000)。

alpha 属性设置元素 （TextElement 和 GraphicElement）的 alpha 透明度值。值范围介于 0 （完全透明）到 1 （完全不透明，默认值）之间。 alpha 为 0 的元素不可见，但仍为活动元素。此值乘以继承的 alpha 值，因而元素更为透明。

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

基线对齐和移位

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

行中最大文本的字体和大小决定了行的主要基线。可通过设置 TextBlock.baselineFontDescription 和 TextBlock.baselineFontSize 来覆盖这些值。可以将主要基线与文本内多个基线中的一个对齐。这些基线包括上缘线和下缘线或表意字顶部、中心或者底部。

A. 上缘 **B.** 基线 **C.** 下缘 **D.** x 高度

在 ElementFormat 对象中，三个属性决定了基线和对齐特征。alignmentBaseline 属性设置 TextElement 或 GraphicElement 的主基线。此基线是元素的 “ 贴紧 ” 线，所有文本的主要基线均对齐此位置。

dominantBaseline 属性指定要使用各个元素基线中的哪个基线，这决定了元素在行中的垂直位置。默认值为 TextBaseline.ROMAN，但也可以设置为使用 IDEOGRAPHIC_TOP 基线或 IDEOGRAPHIC_BOTTOM 基线作为主要基线。

baselineShift 属性将基线沿 Y 轴移动一定像素数的距离。在正常 （非旋转）文本中，正数值使基线下移，负数值使基线上移。

印刷大小写

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

ElementFormat 的 TypographicCase 属性指定文本大小写，如大写、小写或小型大写字母。

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

旋转文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

可按 90 度的增量旋转文本块或文本段中的字形。 TextRotation 类定义了以下常量，用于设置文本块和字型旋转：

常量	值	说明
AUTO	“auto”	指定 90 度逆时针旋转。通常用于垂直的亚洲文字，以仅旋转需要旋转的字型。
ROTATE_0	“rotate_0”	指定不进行旋转。

常量	值	说明
ROTATE_180	“rotate_180”	指定 180 度旋转。
ROTATE_270	“rotate_270”	指定 270 度旋转。
ROTATE_90	“rotate_90”	指定 90 度顺时针旋转。

若要旋转文本块中的多行文本，请在调用 TextBlock.createTextLine() 方法创建文本行之前设置 TextBlock.lineRotation 属性。

若要旋转文本块或文本段中的字型，请将 ElementFormat.textRotation 属性设置为字型要旋转的度数。字型是构成字符的形状，或者是由多个字型组成的一部分字符。例如，字母 “a” 和 “i” 上的点都是字形。

在某些亚洲语言中会涉及旋转字型。在这些语言中，需要将行旋转到垂直方向，但不旋转行中的字符。有关旋转亚洲文字的详细信息，请参阅第 348 页的 “ 对齐东亚文本 ”。

下面的示例同时旋转文本块以及其中的字型，就像处理亚洲文字那样。该示例也使用日文字体：

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class RotationExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef:ElementFormat;
        private var fd:FontDescription = new FontDescription();
        private var str:String;
        private var tl:TextLine;

        public function RotationExample()
        {
            fd.fontName = "MS Mincho";
            ef = new ElementFormat(fd);
            ef.textRotation = TextRotation.AUTO;
            str = "This is rotated Japanese text";
            te = new TextElement(str, ef);
            tb.lineRotation = TextRotation.ROTATE_90;
            tb.content = te;
            tl = tb.createTextLine(null,600);
            addChild(tl);
        }
    }
}
```

锁定和克隆 ElementFormat

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

当 ElementFormat 对象分配给任何类型的 ContentElement 时，其 locked 属性自动设置为 true。试图修改锁定的 ElementFormat 对象将会引发 IllegalOperationError。最佳做法是先完全定义这类对象，然后将其分配给 TextElement 实例。

如果要修改某个现有 ElementFormat 实例，请首先检查其 locked 属性。如果该属性为 true，请使用 clone() 方法创建对象的未锁定副本。此未锁定对象的属性可以更改，然后可以将其指派给 TextElement 实例。依据此对象创建的所有新行均具有新的格式设置。前面依据此对象创建并使用旧格式的行保持不变。

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class ElementFormatCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd:FontDescription = new FontDescription();

        public function ElementFormatCloneExample()
        {
            fd.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            ef1.fontSize = 24;
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontSize = 32;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

使用字体

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

FontDescription 对象与 ElementFormat 配合使用可以标识字型，以及定义其部分特征。这些特征包括字体名称、粗细、形态、呈示以及如何查找字体 （设备字体与嵌入字体）。

注： FTE 不支持 Type 1 字体或 Type 3、 ATC、 sfnt 换行 CID 或 Naked CID 等位图字体。

定义字体特征 （FontDescription 对象）

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

FontDescription 对象的 fontName 属性可以是单个名称，也可以是逗号分隔的名称列表。例如，在 “Arial, Helvetica, _sans” 等列表中，文本引擎先查找 “Arial”，然后查找 “Helvetica”，如果这两种字体都找不到，则最后查找 “_sans”。在字体名称集中有三个通用设备字体名称：“_sans”、 “_serif” 和 “_typewriter”。这些名称根据播放系统映射到具体的设备字体。在使用设备字体的所有字体说明中指定此类默认名称是一种很好的做法。如果没有指定任何 fontName，则默认使用 “_serif”。

fontPosture 属性可设置为默认值 (FontPosture.NORMAL)，也可设置为斜体 (FontPosture.ITALIC)。 fontWeight 属性可设置为默认值 (FontWeight.NORMAL)，也可设置为粗体 (FontWeight.BOLD)。

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

嵌入字体和设备字体

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

FontDescription 对象的 fontLookup 属性指定文本引擎是查找设备字体还是嵌入字体来呈现文本。如果指定设备字体 (FontLookup.DEVICE)，则运行时将在播放系统中查找该字体。如果指定嵌入字体 (FontLookup.EMBEDDED_CFF)，运行时将在 SWF 文件中查找具有指定名称的嵌入字体。只有嵌入的 CFF （压缩字体格式）字体才可使用此项设置。如果没有找到指定的字体，则会使用后备设备字体。

使用设备字体，得到的 SWF 文件较小。嵌入字体提供了更高的跨平台保真度。

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

呈现模式和提示

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

从 Flash Player 10 和 Adobe AIR 1.5 开始提供 CFF （压缩字体格式）呈示。以此方式呈示字体能使文本更加清晰可辨，并为小字号的字体提供更高品质的显示效果。此项设置只适用于嵌入字体。对于 renderingMode 属性， FontDescription 默认设为此项设置 (RenderingMode.CFF)。可以将此属性设置为 RenderingMode.NORMAL，从而与 Flash Player 7 或更低版本所使用的呈示类型保持一致。

选择 CFF 呈示之后，将由另一个属性 cffHinting 控制字体水平线适合子像素网格的方式。默认值 CFFHinting.HORIZONTAL_STEM 使用 CFF 提示。将此属性设置为 CFFHinting.NONE 将会消除提示，对于动画或大字号应当如此设置。

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

锁定和克隆 FontDescription

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

当 FontDescription 对象分配给 ElementFormat 时，其 locked 属性自动设置为 true。试图修改锁定的 FontDescription 对象将会引发 IllegalOperationError。最佳做法是先完全定义这类对象，然后将其分配给 ElementFormat。

如果要修改某个现有 FontDescription，请首先检查其 locked 属性。如果该属性为 true，请使用 clone() 方法创建对象的未锁定副本。此未锁定对象的属性可以更改，然后可以将其指派给 ElementFormat。根据此 TextElement 创建的所有新行均具有新的格式设置。以前根据这个对象创建的行保持不变。

```
package
{
    import flash.display.Sprite;
    import flash.text.*;

    public class FontDescriptionCloneExample extends Sprite
    {
        private var tb:TextBlock = new TextBlock();
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var ef2:ElementFormat;
        private var fd1:FontDescription = new FontDescription();
        private var fd2:FontDescription;

        public function FontDescriptionCloneExample()
        {
            fd1.fontName = "Garamond";
            ef1 = new ElementFormat(fd);
            var str:String = "This is flash text";
            te = new TextElement(str, ef);
            tb.content = te;
            var tx1:TextLine = tb.createTextLine(null,600);
            addChild(tx1);

            fd2 = (fd1.locked) ? fd1.clone() : fd1;
            fd2.fontName = "Arial";
            ef2 = (ef1.locked) ? ef1.clone() : ef1;
            ef2.fontDescription = fd2;
            tb.content.elementFormat = ef2;
            var tx2:TextLine = tb.createTextLine(null,600);
            addChild(tx2);
        }
    }
}
```

控制文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

FTE 提供了一组新的文本格式控件，用以处理对齐和字符间距 （字距调整和间距）。还有一些属性可用于控制各行断开方式以及在行中设置 Tab 停靠位。

对齐文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

通过调整单词之间 （有的时候是字母之间）的间距，对齐文本能让段落中各行的长度相等。其效果是文本两端对齐，而单词和字母之间的间距发生变化。报刊杂志中的文本栏通常是对齐的。

SpaceJustifier 类中的 lineJustfication 属性可用于控制文本块中各行的对齐情况。 LineJustification 类定义了可用于指定对齐选项的常量：ALL_BUT_LAST 对齐除文本最后一行之外的所有各行； ALL_INCLUDING_LAST 对齐所有文本，包括最后一行； UNJUSTIFIED 是默认值，保持文本的不对齐状态。

若要对齐文本，请将 lineJustification 属性设置为 SpaceJustifier 类的实例，并将该实例指派给 TextBlock 实例的 textJustifier 属性。下面的示例创建一个段落。在此段落中，除最后一行文本之外，所有各行均对齐。

```
package
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class JustifyExample extends Sprite
    {
        public function JustifyExample()
        {
            var str:String = "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.";

            var format:ElementFormat = new ElementFormat();
            var textElement:TextElement=new TextElement(str,format);
            var spaceJustifier:SpaceJustifier=new SpaceJustifier("en",LineJustification.ALL_BUT_LAST);

            var textBlock:TextBlock = new TextBlock();
            textBlock.content=textElement;
            textBlock.textJustifier=spaceJustifier;
            createLines(textBlock);
        }

        private function createLines(textBlock:TextBlock):void {
            var yPos=20;
            var textLine:TextLine=textBlock.createTextLine(null,150);

            while (textLine) {
                addChild(textLine);
                textLine.x=15;
                yPos+=textLine.textHeight+2;
                textLine.y=yPos;
                textLine=textBlock.createTextLine(textLine,150);
            }
        }
    }
}
```

若要更改字母以及单词之间的间距，请将 SpaceJustifier.letterspacing 属性设置为 true。应用字母间距调整功能可以减少单词间出现不美观间隙 （简单对齐有时会出现这种情况）的几率。

对齐东亚文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

对齐东亚文字时，需要额外注意一些事项。东亚文字可能是从上至下书写，且某些字符 （称为避头尾）不能出现在行首或行尾。 JustificationStyle 类定义了以下常量，这些常量指定了用于处理这些字符的选项。 PRIORITIZE_LEAST_ADJUSTMENT 通过扩展行或压缩行进行对齐，具体采用哪种方式取决于能否产生最理想的效果。 PUSH_IN_KINSOKU 通过行尾压缩避头尾或扩展行 （如果没有避头尾或该空间不足）进行对齐。

PUSH_OUT_ONLY 通过扩展行进行对齐。若要创建垂直的亚洲文本块，请将 TextBlock.lineRotation 属性设置为 TextRotation.ROTATE_90，并将 ElementFormat.textRotation 属性设置为 TextRotation.AUTO （默认值）。将 textRotation 属性设置为 AUTO 将使文本中的字型保持垂直，而不是在整行发生旋转的时候侧向旋转。根据字形的 Unicode 属性， AUTO 设置仅将全宽字形和宽字形逆时针旋转 90°。下面的示例显示垂直的日文文本块，并使用 PUSH_IN_KINSOKU 选项使其对齐。

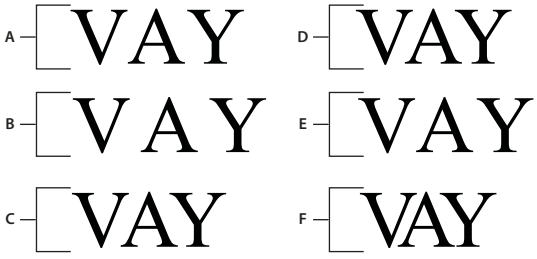
```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    public class EastAsianJustifyExample extends Sprite
    {
        public function EastAsianJustifyExample()
        {
            var Japanese_txt:String = String.fromCharCode(
            0x5185, 0x95A3, 0x5E9C, 0x304C, 0x300C, 0x653F, 0x5E9C, 0x30A4,
            0x30F3, 0x30BF, 0x30FC, 0x30CD, 0x30C3, 0x30C8, 0x30C6, 0x30EC,
            0x30D3, 0x300D, 0x306E, 0x52D5, 0x753B, 0x914D, 0x4FE1, 0x5411,
            0x3051, 0x306B, 0x30A2, 0x30C9, 0x30D3, 0x30B7, 0x30B9, 0x30C6,
            0x30E0, 0x30BA, 0x793E, 0x306E)
            var textBlock:TextBlock = new TextBlock();
            var font:FontDescription = new FontDescription();
            var format:ElementFormat = new ElementFormat();
            format.fontSize = 12;
            format.color = 0xCC0000;
            format.textRotation = TextRotation.AUTO;
            textBlock.baselineZero = TextBaseline.IDEOGRAPHIC_CENTER;
            var eastAsianJustifier:EastAsianJustifier = new EastAsianJustifier("ja",
LineJustification.ALL_BUT_LAST);
            eastAsianJustifier.justificationStyle = JustificationStyle.PUSH_IN_KINSOKU;
            textBlock.textJustifier = eastAsianJustifier;
            textBlock.lineRotation = TextRotation.ROTATE_90;
            var linePosition:Number = this.stage.stageWidth - 75;
            if (Capabilities.os.search("Mac OS") > -1)
                // set fontName: Kozuka Mincho Pro R
                font.fontName = String.fromCharCode(0x5C0F, 0x585A, 0x660E, 0x671D) + " Pro R";
            else
                font.fontName = "Kozuka Mincho Pro R";
            textBlock.content = new TextElement(Japanese_txt, format);
            var previousLine:TextLine = null;

            while (true)
            {
                var textLine:TextLine = textBlock.createTextLine(previousLine, 200);
                if (textLine == null)
                    break;
                textLine.y = 20;
                textLine.x = linePosition;
                linePosition -= 25;
                addChild(textLine);
                previousLine = textLine;
            }
        }
    }
}
```

字距调整和间距

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

字距调整和间距影响文本块中相邻两个字符之间的距离。字距调整控制字符对如何组合在一起，例如字符对 “WA” 或 “Va”。字距调整在 ElementFormat 对象中设置。字距调整默认为启用 (Kerning.ON)。它可以设置为 OFF 或 AUTO，在此情形中，只有两个字符都不是日本汉字、平假名和片假名时，才在它们之间应用字距调整。

间距处理在文本块中的所有字符之间增加或减少一定数量的像素，并且也是在 ElementFormat 对象中进行设置。嵌入字体和设备字体均可使用间距处理。 FTE 支持两种间距属性：trackingLeft，增加 / 减少字符左侧的像素；以及 trackingRight，增加 / 减少字符右侧的像素。如果使用间距，那么每个字符对的字距调整值将会加上或减去间距值。



A. Kerning.OFF **B.** TrackingRight=5，Kerning.OFF **C.** TrackingRight=-5，Kerning.OFF **D.** Kerning.ON **E.** TrackingRight=-5，Kerning.ON **F.** TrackingRight=-5，Kerning.ON

```
var ef1:ElementFormat = new ElementFormat();
ef1.kerning = Kerning.OFF;

var ef2:ElementFormat = new ElementFormat();
ef2.kerning = Kerning.ON;
ef2.trackingLeft = 0.8;
ef2.trackingRight = 0.8;

var ef3:ElementFormat = new ElementFormat();
ef3.trackingRight = -0.2;
```

换行文本的换行符

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

ElementFormat 对象的 breakOpportunity 属性确定在换行文本分成多行时，哪些字符可用于断行。默认值 BreakOpportunity.AUTO 使用标准 Unicode 属性，例如在不同单词之间断行，以及在连字符的位置断行。如果使用 BreakOpportunity.ALL，那么在任何一个字符均可断行，这有助于制造某些效果 （例如沿某个路径排列文本）。

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

Tab 停靠位

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

若要在文本块中设置 Tab 停靠位，请通过创建 TabStop 类的实例来定义 Tab 停靠位。TabStop() 构造函数的参数指定文本如何与 Tab 停靠位对齐。这些参数指定 Tab 停靠位的位置，以及对于十进制数对齐，指定要在哪个值对齐 （表示为字符串）。通常，该值是小数点，但也可以是逗号、美元符号或日元或欧元符号等。下面的代码行创建一个名为 tab1 的 Tab 停靠位。

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

为一个文本块创建 Tab 停靠位之后，请将这些停靠位分配给 TextBlock 实例的 tabStops 属性。由于 tabStops 属性需要一个 Vector，首先创建一个 Vector，然后将 Tab 停靠位添加到此 Vector 中。此 Vector 允许您将一组 Tab 停靠位分配给文本块。下面的示例创建一个 Vector<TabStop> 实例，并将一组 TabStop 对象添加到其中。然后，代码将这些 Tab 停靠位分配给 TextBlock 实例的 tabStops 属性。

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

有关矢量的详细信息，请参阅第 21 页的 “ 使用数组 ”。

下面的示例演示每个 TabStop 对齐选项的效果。

```
package {

    import flash.text.engine.*;
    import flash.display.Sprite;

    public class TabStopExample extends Sprite
    {
        public function TabStopExample()
        {
            var format:ElementFormat = new ElementFormat();
            format.fontDescription = new FontDescription("Arial");
            format.fontSize = 16;

            var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
            tabStops.push(
                new TabStop(TabAlignment.START, 20),
                new TabStop(TabAlignment.CENTER, 140),
                new TabStop(TabAlignment.DECIMAL, 260, "."),
                new TabStop(TabAlignment.END, 380));
            var textBlock:TextBlock = new TextBlock();
            textBlock.content = new TextElement(
                "\tt1\tt2\tt3\tt4\n" +
                "\tThis line aligns on 1st tab\n" +
                "\t\t\t\tThis is the end\n" +
                "\tThe following fragment centers on the 2nd tab:\t\t\n" +
                "\t\tit's on me\t\t\n" +
                "\tThe following amounts align on the decimal point:\n" +
                "\t\t\t45.00\t\n" +
                "\t\t\t75,320.00\t\n" +
                "\t\t\t6,950.00\t\n" +
                "\t\t\t7.01\t\n", format);

            textBlock.tabStops = tabStops;
            var yPosition:Number = 60;
            var previousTextLine:TextLine = null;
            var textLine:TextLine;
            var i:int;
            for (i = 0; i < 10; i++) {
                textLine = textBlock.createTextLine(previousTextLine, 1000, 0);
                textLine.x = 20;
                textLine.y = yPosition;
                addChild(textLine);
                yPosition += 25;
                previousTextLine = textLine;
            }
        }
    }
}
```

Flash 文本引擎示例：新闻版面布局

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

此编程示例演示在为简单的新闻页面布局时，如何使用 Flash 文本引擎。该页包含大标题、副标题和包含多列的正文部分。

首先，创建一个 FLA 文件，并将以下代码附加到默认图层的第 2 帧：

```
import com.example.programmingas3.newslayout.StoryLayout ;
// frame sc ript - create  a 3-columned arti cle layout
var story:StoryLayout = new StoryLayout(720, 500, 3, 10);
story.x = 20;
story.y = 80;
addChild(story);
stop();
```

StoryLayout.as 是本示例的控制器脚本。它设置内容，从外部样式表读取样式信息，将这些样式分配给 ElementFormat 对象。然后，它创建标题、副标题和多列文本元素。

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.StyleSheet;
    import flash.text.engine.*;

    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.display.Sprite;
    import flash.display.Graphics;

    public class StoryLayout extends Sprite
    {
        public var headlineTxt:HeadlineTextField;
        public var subtitleTxt:HeadlineTextField;
        public var storyTxt:MultiColumnText;
        public var sheet:StyleSheet;
        public var h1_ElFormat:ElementFormat;
        public var h2_ElFormat:ElementFormat;
        public var p_ElFormat:ElementFormat;

        private var loader:URLLoader;

        public var paddingLeft:Number;
        public var paddingRight:Number;
        public var paddingTop:Number;
        public var paddingBottom:Number;

        public var preferredWidth:Number;
        public var preferredHeight:Number;

        public var numColumns:int;

        public var bgColor:Number = 0xFFFFFF;

        public var headline:String = "News Layout Example";
        public var subtitle:String = "This example formats text like a newspaper page using the Flash Text
Engine API. ";

        public var rawTestData:String =
        "From the part Mr. Burke took in the American Revolution, it was natural that I should consider him
a friend to mankind; and as our acquaintance commenced on that ground, it would have been more agreeable to
```

```
me to have had cause to continue in that opinion than to change it. " +
        "At the time Mr. Burke made his violent speech last winter in the English Parliament against the
French Revolution and the National Assembly, I was in Paris, and had written to him but a short time before
to inform him how prosperously matters were going on. Soon after this I saw his advertisement of the Pamphlet
he intended to publish: As the attack was to be made in a language but little studied, and less understood
in France, and as everything suffers by translation, I promised some of the friends of the Revolution in
that country that whenever Mr. Burke's Pamphlet came forth, I would answer it. This appeared to me the more
necessary to be done, when I saw the flagrant misrepresentations which Mr. Burke's Pamphlet contains; and
that while it is an outrageous abuse on the French Revolution, and the principles of Liberty, it is an
imposition on the rest of the world. " +
        "I am the more astonished and disappointed at this conduct in Mr. Burke, as (from the circumstances
I am going to mention) I had formed other expectations. " +
        "I had seen enough of the miseries of war, to wish it might never more have existence in the world,
and that some other mode might be found out to settle the differences that should occasionally arise in the
neighbourhood of nations. This certainly might be done if Courts were disposed to set honesty about it, or
if countries were enlightened enough not to be made the dupes of Courts. The people of America had been bred
up in the same prejudices against France, which at that time characterised the people of England; but
experience and an acquaintance with the French Nation have most effectually shown to the Americans the
falsehood of those prejudices; and I do not believe that a more cordial and confidential intercourse exists
between any two countries than between America and France. ";

        public function StoryLayout(w:int = 400, h:int = 200, cols:int = 3, padding:int = 10):void
        {
            this.preferredWidth = w;
            this.preferredHeight = h;

            this.numColumns = cols;

            this.paddingLeft = padding;
            this.paddingRight = padding;
            this.paddingTop = padding;
            this.paddingBottom = padding;

            var req:URLRequest = new URLRequest("story.css");
            loader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, onCSSFileLoaded);
            loader.load(req);
        }

        public function onCSSFileLoaded(event:Event):void
        {
            this.sheet = new StyleSheet();
            this.sheet.parseCSS(loader.data);

            // convert headline styles to ElementFormat objects
            h1_ElFormat = getElFormat("h1", this.sheet);
            h1_ElFormat.typographicCase = TypographicCase.UPPERCASE;
            h2_ElFormat = getElFormat("h2", this.sheet);
            p_ElFormat = getElFormat("p", this.sheet);
            displayText();
        }

        public function drawBackground():void
        {
            var h:Number = this.storyTxt.y + this.storyTxt.height +
                            this.paddingTop + this.paddingBottom;
            var g:Graphics = this.graphics;
            g.beginFill(this.bgColor);
            g.drawRect(0, 0, this.width + this.paddingRight + this.paddingLeft, h);
            g.endFill();
        }
```

```
/**
 * Reads a set of style properties for a named style and then creates
 * a TextFormat object that uses the same properties.
 */
public function getElFormat(styleName:String, ss:StyleSheet):ElementFormat
{
var style:Object = ss.getStyle(styleName);
if (style != null)
{
var colorStr:String = style.color;
if (colorStr != null && colorStr.indexOf("#") == 0)
{
    style.color = colorStr.substr(1);
}
        var fd:FontDescription = new FontDescription(
                                style.fontFamily,
                                style.fontWeight,
                                FontPosture.NORMAL,
                                FontLookup.DEVICE,
                                RenderingMode.NORMAL,
                                CFFHinting.NONE);
var format:ElementFormat = new ElementFormat(fd,
                        style.fontSize,
                        style.color,
                        1,
                        TextRotation.AUTO,
                        TextBaseline.ROMAN,
                        TextBaseline.USE_DOMINANT_BASELINE,
                        0.0,
                        Kerning.ON,
                        0.0,
                        0.0,
                        "en",
                                BreakOpportunity.AUTO,
                                DigitCase.DEFAULT,
                                DigitWidth.DEFAULT,
                        LigatureLevel.NONE,
                                TypographicCase.DEFAULT);

if (style.hasOwnProperty("letterSpacing"))
{
    format.trackingRight = style.letterSpacing;
}
}
return format;
}

public function displayText():void
{
headlineTxt = new HeadlineTextField(h1_ElFormat,headline,this.preferredWidth);
headlineTxt.x = this.paddingLeft;
```

```
        headlineTxt.y = 40 + this.paddingTop;
            headlineTxt.fitText(1);
            this.addChild(headlineTxt);

        subtitleTxt = new HeadlineTextField(h2_ElFormat,subtitle,this.preferredWidth);
        subtitleTxt.x = this.paddingLeft;
        subtitleTxt.y = headlineTxt.y + headlineTxt.height;
            subtitleTxt.fitText(2);
        this.addChild(subtitleTxt);

        storyTxt = new MultiColumnText(rawTestData, this.numColumns,
                        20, this.preferredWidth, this.preferredHeight, p_ElFormat);
        storyTxt.x = this.paddingLeft;
        storyTxt.y = subtitleTxt.y + subtitleTxt.height + 10;
        this.addChild(storyTxt);

            drawBackground();
        }
    }
}
```

FormattedTextBlock.as 用作文本块创建的基类。它还包含用于更改字体大小和大小写的实用程序函数。

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    import flash.display.Sprite;

    public class FormattedTextBlock extends Sprite
    {
        public var tb:TextBlock;
        private var te:TextElement;
        private var ef1:ElementFormat;
        private var textWidth:int;
        public var totalTextLines:int;
        public var blockText:String;
        public var leading:Number = 1.25;
        public var preferredWidth:Number = 720;
        public var preferredHeight:Number = 100;

        public function FormattedTextBlock(ef:ElementFormat,txt:String, colW:int = 0)
        {
            this.textWidth = (colW==0) ? preferredWidth : colW;
            blockText = txt;
            ef1 = ef;
            tb = new TextBlock();
            tb.textJustifier = new SpaceJustifier("en",LineJustification.UNJUSTIFIED,false);
            te = new TextElement(blockText,this.ef1);
            tb.content = te;
            this.breakLines();
        }

        private function breakLines()
        {
            var textLine:TextLine = null;
            var y:Number = 0;
            var lineNum:int = 0;
            while (textLine = tb.createTextLine(textLine,this.textWidth,0,true))
            {
                textLine.x = 0;
                textLine.y = y;
                y += this.leading*textLine.height;
                this.addChild(textLine);
```

```
            }
            for (var i:int = 0; i < this.numChildren; i++)
            {
                TextLine(this.getChildAt(i)).validity = TextLineValidity.STATIC;
            }
            this.totalTextLines = this.numChildren;
        }

        private function rebreakLines()
        {
            this.clearLines();
            this.breakLines();
        }

        private function clearLines()
        {
            while(this.numChildren)
            {
                this.removeChildAt(0);
            }
        }

        public function changeSize(size:uint=12):void
        {
            if (size > 5)
            {
                var ef2:ElementFormat = ef1.clone();
                ef2.fontSize = size;
                te.elementFormat = ef2;
                this.rebreakLines();
            }
        }

        public function changeCase(newCase:String = "default"):void
        {
            var ef2:ElementFormat = ef1.clone();
            ef2.typographicCase = newCase;
            te.elementFormat = ef2;
        }
    }
}
```

HeadlineTextBlock.as 扩展 FormattedTextBlock 类，用于创建标题。它包含用于将文本放入已定义页面区域的函数。

```
package com.example.programmingas3.newslayout
{
    import flash.text.engine.*;
    public class HeadlineTextField extends FormattedTextBlock
    {

        public static var MIN_POINT_SIZE:uint = 6;
        public static var MAX_POINT_SIZE:uint = 128;

        public function HeadlineTextField(te:ElementFormat,txt:String,colW:int = 0)
        {
            super(te,txt);
        }

        public function fitText(maxLines:uint = 1, targetWidth:Number = -1):uint
        {
            if (targetWidth == -1)
            {
                targetWidth = this.width;
            }

            var pixelsPerChar:Number = targetWidth / this.blockText.length;
            var pointSize:Number = Math.min(MAX_POINT_SIZE,
                            Math.round(pixelsPerChar * 1.8 * maxLines));

            if (pointSize < 6)
            {
                // the point size is too small
                return pointSize;
            }

            this.changeSize(pointSize);
            if (this.totalTextLines > maxLines)
            {
                return shrinkText(--pointSize, maxLines);
            }
            else
            {
                return growText(pointSize, maxLines);
            }
        }

        public function growText(pointSize:Number, maxLines:uint = 1):Number
        {
            if (pointSize >= MAX_POINT_SIZE)
            {
                return pointSize;
            }

            this.changeSize(pointSize + 1);
            if (this.totalTextLines > maxLines)
            {
                // set it back to the last size
                this.changeSize(pointSize);
                return pointSize;
            }
            else
            {
                return growText(pointSize + 1, maxLines);
            }
        }
```

```
        public function shrinkText(pointSize:Number, maxLines:uint=1):Number
        {
            if (pointSize <= MIN_POINT_SIZE)
            {
                return pointSize;
            }
            this.changeSize(pointSize);

            if (this.totalTextLines > maxLines)
            {
                return shrinkText(pointSize - 1, maxLines);
            }
            else
            {
                return pointSize;
            }
        }
    }
}
```

MultiColumnText.as 处理多列设计中的文本格式设置。它演示了 TextBlock 对象的灵活用法，该对象可作为用于创建文本行、设置文本行格式以及放置文本行的工厂。

```
package com.example.programmingas3.newslayout
{
    import flash.display.Sprite;
    import flash.text.engine.*;

    public class MultiColumnText extends Sprite
    {
        private var tb:TextBlock;
        private var te:TextElement;
        private var numColumns:uint = 2;
        private var gutter:uint = 10;
        private var leading:Number = 1.25;
        private var preferredWidth:Number = 400;
        private var preferredHeight:Number = 100;
        private var colWidth:int = 200;

        public function MultiColumnText(txt:String = "",cols:uint = 2,
            gutter:uint = 10, w:Number = 400, h:Number = 100,
            ef:ElementFormat = null):void
        {
            this.numColumns = Math.max(1, cols);
            this.gutter = Math.max(1, gutter);

            this.preferredWidth = w;
            this.preferredHeight = h;

            this.setColumnWidth();

            var field:FormattedTextBlock = new FormattedTextBlock(ef,txt,this.colWidth);
            var totLines:int = field.totalTextLines;
            field = null;
            var linesPerCol:int = Math.ceil(totLines/cols);

            tb = new TextBlock();
            te = new TextElement(txt,ef);
            tb.content = te;
            var textLine:TextLine = null;
            var x:Number = 0;
            var y:Number = 0;
```

```
            var i:int = 0;
            var j:int = 0;
            while (textLine = tb.createTextLine(textLine,this.colWidth,0,true))
            {
                textLine.x = Math.floor(i/(linesPerCol+1))*(this.colWidth+this.gutter);
                textLine.y = y;
                y += this.leading*textLine.height;
                j++;
                if(j>linesPerCol)
                {
                    y = 0;
                    j = 0;
                }
                i++;

                this.addChild(textLine);
            }
        }

        private function setColumnWidth():void
        {
        this.colWidth = Math.floor( (this.preferredWidth -
            ((this.numColumns - 1) * this.gutter)) / this.numColumns);
        }

    }
}
```

第 23 章：使用 Text Layout Framework

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Text Layout Framework 概述

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Text Layout Framework (TLF) 是一种可扩展的 ActionScript 库。TLF 是在 Adobe® Flash® Player 10 和 Adobe® AIR® 1.5 中的文本引擎基础上构建而成的。 TLF 提供的高级排版和文本布局功能可以实现新颖的 Web 排版创意。该框架可与 Adobe® Flex® 或 Adobe® Flash® Professional 一起使用。开发人员可以使用或扩展现有组件，也可以使用该框架创建自己的文本组件。

TLF 包含以下功能：

- 双向文本、垂直文本以及 30 多种写作文种，包括阿拉伯文、希伯来文、中文、日文、韩文、泰文、老挝文、越南文和其他文种
- 跨多个列和链接容器选择、编辑和流动文本
- 垂直文本、直排内横排 （在垂直文本内显示水平文本）以及在东亚排版规则中对齐文本
- 丰富的排版控件，包括字距微调、连字、印刷大小写、数字大小写、数字宽度和自由连字符
- 剪切、复制、粘贴、撤消以及标准的键盘和鼠标编辑手势
- 丰富的开发人员 API，用于操纵文本内容、布局、和标记以及创建自定文本组件
- 强大的列表支持，包括自定义标记和编号格式
- 内嵌图像和定位规则

TLF 是在 Flash Player 10 中引入的 Flash 文本引擎 (FTE) 基础上构建的一种 ActionScript 3.0 库。可以通过 flash.text.engine 包访问 FTE，该包是 Flash Player 10 应用程序编程接口 (API) 的一部分。

但是，Flash Player API 提供了对文本引擎的低级别访问，这意味着为了执行某些任务可能需要使用的代码相对比较多。TLF 将底层代码封装到更简单的 API 中。 TLF 还提供了一种概念体系结构，可以将 FTE 定义的基础构建块组织成易于使用的系统。

与 FTE 不同， TLF 并未内置在 Flash Player 中，而是完全用 ActionScript 3.0 编写的一个独立组件库。由于此框架可扩展，因此可针对特定环境对其进行自定义。 Flash Professional 和 Flex SDK 都包括基于 TLF 框架的组件。

更多帮助主题

"Flow"TLF 标记应用程序

复杂文种支持

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

TLF 提供了复杂文种支持。复杂文种支持包括显示和编辑从右至左文种的功能。TLF 还提供了显示和编辑从左至右和从右至左混合文种 （如阿拉伯文和希伯来文）的功能。此框架不仅支持适用于中文、日文和韩文的垂直文本布局，而且还支持直排内横排 （TCY 元素）。 TCY 元素是嵌入到垂直文本串的水平文本块。支持以下文种：

- 拉丁文 （英文、西班牙文、法文、越南文等）
- 希腊文、西里尔文、亚美尼亚文、格鲁吉亚文和埃塞俄比亚文
- 阿拉伯文和希伯来文
- 汉字象形和假名 （中文、日文和韩文）及 Hangul Johab （韩文）
- 泰文、老挝文和高棉文
- 梵文、孟加拉文、果鲁穆奇文、马拉雅拉姆文、泰卢固文、泰米尔文、古吉特拉文、奥里雅文、卡纳达文和藏文
- 提非纳文、彝文、切罗基文、加拿大音节、德塞莱特文、萧伯纳字母、瓦伊文、塔加路文、哈努诺文、布迪文和塔格巴努亚文

在 Flash Professional 和 Flex 中使用 Text Layout Framework

在 Flash 中，您可以直接使用 TLF 类创建自定义组件。此外， Flash Professional CS5 还提供了一个新的 fl.text.TLFTextField 类，该类封装了 TLF 功能。使用 TLFTextField 类可在 ActionScript 中创建使用 TLF 的高级文本显示功能的文本字段。以与使用 TextField 类创建文本字段相同的方式创建 TLFTextField 对象。然后，使用 textFlow 属性从 TLF 类分配高级格式设置。

利用文本工具，还可以使用 Flash Professional 在舞台上创建 TLFTextField 实例。然后，可以使用 ActionScript 通过 TLF 类控制文本字段内容的格式和布局。有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 TLFTextField。

在 Flex 中工作时，请使用 TLF 类。有关更多信息，请参阅第 361 页的 “ 使用 Text Layout Framework”。

使用 Text Layout Framework

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在 Flex 中工作时或在构建自定义文本组件时，请使用 TLF 类。 TLF 是一种 ActionScript 3.0 库，它完全包含在 textLayout.swc 库中。 TLF 库包含约 100 个 ActionScript 3.0 类和接口，这些类和接口被组织到 10 个包中。这些包是 flashx.textLayout 包的子包。

Text Layout Framework 类

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

TLF 类划分为三种类别：

- 数据结构和格式设置类
- 呈现类
- 用户交互类

数据结构和格式设置类

下列包包含 TLF 的数据结构和格式设置类：

- flashx.textLayout.elements
- flashx.textLayout.formats
- flashx.textLayout.conversion

TLF 的主要数据结构为文本流层次结构，在元素包中定义。在该结构中，可以使用格式包为文本串指定样式和属性，也可以使用转换包控制在数据结构中如何导入和导出文本。

呈现类

下列包包含 TLF 的呈现类：

- flashx.textLayout.factory
- flashx.textLayout.container
- flashx.textLayout.compose

利用这些包中的类，可以方便地呈现文本以便在 Flash Player 中显示。工厂包提供显示静态文本的简便方式。容器包中的类和接口用于定义动态文本的显示容器。组合包定义在容器中放置和显示动态文本的技术。

用户交互类

下列包包含 TLF 的用户交互类：

- flashx.textLayout.edit
- flashx.textLayout.operations
- flashx.textLayout.events

编辑包和操作包定义了一些类，使用这些类可编辑在数据结构中存储的文本。事件包中包含事件处理类。

使用 Text Layout Framework 创建文本的常规步骤

下列步骤介绍了创建带有文本布局格式的文本的常规过程：

1. 将带格式文本导入到 TLF 数据结构中。有关更多信息，请参阅第 365 页的 “ 使用 TLF 构建文本结构 ” 和第 369 页的 “ 使用 TLF 设置文本格式 ”。
2. 为文本创建一个或多个链接的显示对象容器。有关更多信息，请参阅第 370 页的 “ 使用 TLF 管理文本容器 ”。
3. 将数据结构中的文本与容器关联并设置编辑和滚动选项。有关更多信息，请参阅第 371 页的 “ 使用 TLF 启用文本选择、编辑和撤消 ”。
4. 在响应 resize （或其他）事件时，创建事件处理函数以重排文本。有关更多信息，请参阅第 372 页的 “ 使用 TLF 处理事件 ”。

Text Layout Framework 示例：新闻版面布局

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

以下示例演示了如何使用 TLF 设置简单的新闻页面的布局。该页包含大标题、副标题和包含多列的正文部分：

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.Event;
    import flash.geom.Rectangle;

    import flashx.textLayout.compose.StandardFlowComposer;
    import flashx.textLayout.container.ContainerController;
    import flashx.textLayout.container.ScrollPolicy;
    import flashx.textLayout.conversion.TextConverter;
    import flashx.textLayout.elements.TextFlow;
    import flashx.textLayout.formats.TextLayoutFormat;

    public class TLFNewsLayout extends Sprite
    {
        private var hTextFlow:TextFlow;
        private var headContainer:Sprite;
        private var headlineController:ContainerController;
        private var hContainerFormat:TextLayoutFormat;

        private var bTextFlow:TextFlow;
        private var bodyTextContainer:Sprite;
        private var bodyController:ContainerController;
        private var bodyTextContainerFormat:TextLayoutFormat;

        private const headlineMarkup:String = "<flow:TextFlow
xmlns:flow='http://ns.adobe.com/textLayout/2008'><flow:p textAlign='center'><flow:span
fontFamily='Helvetica' fontSize='18'>TLF News Layout Example</flow:span><flow:br/><flow:span
fontFamily='Helvetica' fontSize='14'>This example formats text like a newspaper page with a headline, a
subtitle, and multiple columns</flow:span></flow:p></flow:TextFlow>";

        private const bodyMarkup:String = "<flow:TextFlow xmlns:flow='http://ns.adobe.com/textLayout/2008'
fontSize='12' textIndent='10' marginBottom='15' paddingTop='4' paddingLeft='4'><flow:p
marginBottom='inherit'><flow:span>There are many </flow:span><flow:span
fontStyle='italic'>such</flow:span><flow:span> lime-kilns in that tract of country, for the purpose of
burning the white marble which composes a large part of the substance of the hills. Some of them, built
years ago, and long deserted, with weeds growing in the vacant round of the interior, which is open to the
sky, and grass and wild-flowers rooting themselves into the chinks of the stones, look already like relics
of antiquity, and may yet be overspread with the lichens of centuries to come. Others, where the lime-burner
still feeds his daily and nightlong fire, afford points of interest to the wanderer among the hills, who
seats himself on a log of wood or a fragment of marble, to hold a chat with the solitary man. It is a
lonesome, and, when the character is inclined to thought, may be an intensely thoughtful occupation; as it
proved in the case of Ethan Brand, who had mused to such strange purpose, in days gone by, while the fire
in this very kiln was burning.</flow:span></flow:p><flow:p marginBottom='inherit'><flow:span>The man who
now watched the fire was of a different order, and troubled himself with no thoughts save the very few that
were requisite to his business. At frequent intervals, he flung back the clashing weight of the iron door,
and, turning his face from the insufferable glare, thrust in huge logs of oak, or stirred the immense brands
with a long pole. Within the furnace were seen the curling and riotous flames, and the burning marble, almost
molten with the intensity of heat; while without, the reflection of the fire quivered on the dark intricacy
of the surrounding forest, and showed in the foreground a bright and ruddy little picture of the hut, the
spring beside its door, the athletic and coal-begrimed figure of the lime-burner, and the half-frightened
child, shrinking into the protection of his father's shadow. And when again the iron door was closed, then
reappeared the tender light of the half-full moon, which vainly strove to trace out the indistinct shapes
of the neighboring mountains; and, in the upper sky, there was a flitting congregation of clouds, still
faintly tinged with the rosy sunset, though thus far down into the valley the sunshine had vanished long
and long ago.</flow:span></flow:p></flow:TextFlow>";

        public function TLFNewsLayout()
        {
            //wait for stage to exist
```

```
            addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
        }

        private function onAddedToStage(evtObj:Event):void
        {
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;

            // Headline text flow and flow composer
            hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);

            // initialize the headline container and controller objects
            headContainer = new Sprite();
            headlineController = new ContainerController(headContainer);
            headlineController.verticalScrollPolicy = ScrollPolicy.OFF;
            hContainerFormat = new TextLayoutFormat();
            hContainerFormat.paddingTop = 4;
            hContainerFormat.paddingRight = 4;
            hContainerFormat.paddingBottom = 4;
            hContainerFormat.paddingLeft = 4;

            headlineController.format = hContainerFormat;
            hTextFlow.flowComposer.addController(headlineController);
            addChild(headContainer);
            stage.addEventListener(flash.events.Event.RESIZE, resizeHandler);

            // Body text TextFlow and flow composer
            bTextFlow = TextConverter.importToFlow(bodyMarkup, TextConverter.TEXT_LAYOUT_FORMAT);

            // The body text container is below, and has three columns
            bodyTextContainer = new Sprite();
            bodyController = new ContainerController(bodyTextContainer);
            bodyTextContainerFormat = new TextLayoutFormat();
            bodyTextContainerFormat.columnCount = 3;
            bodyTextContainerFormat.columnGap = 30;

            bodyController.format = bodyTextContainerFormat;
            bTextFlow.flowComposer.addController(bodyController);
            addChild(bodyTextContainer);
            resizeHandler(null);
        }

        private function resizeHandler(event:Event):void
        {
            const verticalGap:Number = 25;
            const stagePadding:Number = 16;
            var stageWidth:Number = stage.stageWidth - stagePadding;
            var stageHeight:Number = stage.stageHeight - stagePadding;
            var headlineWidth:Number = stageWidth;
            var headlineContainerHeight:Number = stageHeight;

            // Initial compose to get height of headline after resize
            headlineController.setCompositionSize(headlineWidth,
headlineContainerHeight);
            hTextFlow.flowComposer.compose();
```

```
            var rect:Rectangle = headlineController.getContentBounds();
            headlineContainerHeight = rect.height;

            // Resize and place headline text container
            // Call setCompositionSize() again with updated headline height
            headlineController.setCompositionSize(headlineWidth, headlineContainerHeight );
            headlineController.container.x = stagePadding / 2;
            headlineController.container.y = stagePadding / 2;
            hTextFlow.flowComposer.updateAllControllers();

            // Resize and place body text container
            var bodyContainerHeight:Number = (stageHeight - verticalGap - headlineContainerHeight);
            bodyController.format = bodyTextContainerFormat;
            bodyController.setCompositionSize(stageWidth, bodyContainerHeight );
            bodyController.container.x = (stagePadding/2);
            bodyController.container.y = (stagePadding/2) + headlineContainerHeight + verticalGap;
            bTextFlow.flowComposer.updateAllControllers();
        }
    }
}
```

TLFNewsLayout 类使用两个文本容器。一个容器显示大标题和副标题，另一个容器显示包含三列的正文文本。为简洁起见，文本作为 TLF 标记文本硬编码到示例中。headlineMarkup 变量包含标题和副标题，bodyMarkup 变量包含正文文本。有关 TLF 标记的更多信息，请参阅第 365 页的 “ 使用 TLF 构建文本结构 ”。

在完成一些初始化操作之后，onAddedToStage() 函数将大标题文本导入到 TextFlow 对象中，该对象是 TLF 的主要数据结构：

```
hTextFlow = TextConverter.importToFlow(headlineMarkup, TextConverter.TEXT_LAYOUT_FORMAT);
```

接下来，为容器创建 Sprite 对象，然后为容器创建控制器并将该控制器与容器相关联：

```
headContainer = new Sprite();
headlineController = new ContainerController(headContainer);
```

初始化控制器以设置格式、滚动和其他选项。控制器包含相关的几何尺寸设置，用于定义文本流入的容器的边界。TextLayoutFormat 对象包含格式选项：

```
hContainerFormat = new TextLayoutFormat();
```

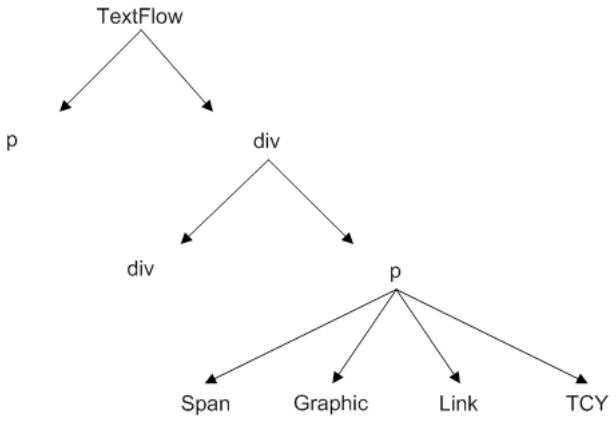
控制器分配给了流合成器，该函数将容器添加到显示列表。容器的实际构成和显示会延迟到 resizeHandler() 方法。按相同顺序执行相关步骤以初始化正文的 TextFlow 对象。

resizeHandler() 方法测量可供呈现容器用的空间并相应地设置容器大小。首先，调用 compose() 方法，以便计算标题容器的合适高度。随后， resizeHandler() 方法可以使用 updateAllControllers() 方法添加和显示标题容器。最后， resizeHandler() 方法使用标题容器的大小确定正文文本容器的位置。

使用 TLF 构建文本结构

TLF 使用层次树来表示文本。树中的每个节点都是元素包中定义的类的实例。例如，树的根节点始终是 TextFlow 类的实例。TextFlow 类表示整个文本素材。文章是文本和其他元素的集合，可将其视为一个单元或流。一篇文章可能需要使用多个列或文本容器进行显示。

除了根节点，其余元素大致基于 XHTML 元素。下图显示了该框架的层次结构：



TextFlow 层次结构

Text Layout Framework 标记

了解 TLF 的结构在处理 TLF 标记时也很有用。 TLF 标记是文本的 XML 表示形式，作为 TLF 一部分包含在其中。虽然该框架也支持其他 XML 格式，但 TLF 标记是唯一专门基于 TextFlow 层次结构的一种格式。如果您使用此标记格式从 TextFlow 导出 XML，则导出 XML 时会保持此层次结构的完整性。

TLF 标记为 TextFlow 层次结构中的文本提供了最高保真度的表示形式。标记语言为 TextFlow 层次结构的每个基本元素提供标签，并为 TextLayoutFormat 类中的所有可用格式设置属性提供特性。

下表包含可以在 TLF 标记中使用的标签。

元素	说明	子代	类
textflow	标记的根元素。	div、 p	TextFlow
div	TextFlow 内的分割块。可以包含一组段落。	div、 list、 p	DivElement
p	段落。	a、 tcy、 span、 img、 tab、 br、 g	ParagraphElement
a	链接。	tcy、 span、 img、 tab、 br、 g	LinkElement
tcy	水平文本串 （用于垂直 TextFlow 中）。	a、 span、 img、 tab、 br、 g	TCYElement
span	段落内的文本串。		SpanElement
img	段落中的图像。		InlineGraphicElement
tab	制表符字符。		TabElement
br	分隔符。用于在段落内结束一行；文本在下一行中继续，仍位于同一段落中。		BreakElement
linkNormalFormat	定义用于处于正常状态的链接的格式属性。	TextLayoutFormat	TextLayoutFormat
linkActiveFormat	在链接上按下鼠标时，定义用于处于活动状态的链接的格式属性。	TextLayoutFormat	TextLayoutFormat
linkHoverFormat	当鼠标位于链接边界内时 （滚过），定义用于处于悬停状态的链接的格式属性。	TextLayoutFormat	TextLayoutFormat

元素	说明	子代	类
li	一个列表项目元素。必须位于一个列表元素内。	div、li、list、p	ListItemElement
list	一个列表。列表可以嵌套，或者放置得距离彼此很近。列表项目可以应用不同的标签或编号方案。	div、li、list、p	ListElement
g	组元素。用于对段落中的元素进行分组。允许您在段落级别以下嵌套元素。	a、tcy、span、img、tab、br、g	SubParagraphGroup Element

更多帮助主题

TLF 2.0 列表标记

TLF 2.0 SubParagraphGroupElements 和 typeName

使用编号列表和项目列表

可以使用 ListElement 和 ListItemElement 类将项目列表添加到文本控件。可以嵌套项目列表，也可以对其进行自定义以使用不同的项目符号 （或标记）和自动编号以及轮廓样式编号。

若要在文本流中创建列表，请使用 <list> 标签。然后，对于列表中的每个列表项，您可以在 <list> 标签内使用 <li> 标签。可以使用 ListMarkerFormat 类自定义项目符号的外观。

下面的示例会创建简单的列表：

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

您可以在其他列表中嵌套列表，如下面的示例所示：

```
<flow:list paddingRight="24" paddingLeft="24">
    <flow:li>Item 1</flow:li>
    <flow:list paddingRight="24" paddingLeft="24">
        <flow:li>Item 1a</flow:li>
        <flow:li>Item 1b</flow:li>
        <flow:li>Item 1c</flow:li>
    </flow:list>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

若要自定义列表中标记的类型，请使用 ListElement 的 listStyleType 属性。此属性可以是 ListStyleType 类定义的任意值（如 check、 circle、 decimal 和 box）。下面的示例使用多种标记类型和一个自定义计数器增量创建列表：

```
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperAlpha"> <flow:li>upperAlpha
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerAlpha"> <flow:li>lowerAlpha item</flow:li> <flow:li>another</flow:li> </flow:list>
<flow:list paddingRight="24" paddingLeft="24" listStyleType="upperRoman"> <flow:li>upperRoman
item</flow:li> <flow:li>another</flow:li> </flow:list> <flow:list paddingRight="24" paddingLeft="24"
listStyleType="lowerRoman"> <flow:listMarkerFormat> <!-- Increments the list by 2s rather than 1s. -->
<flow:ListMarkerFormat counterIncrement="ordered 2"/> </flow:listMarkerFormat> <flow:li>lowerRoman
item</flow:li> <flow:li>another</flow:li> </flow:list>
```

使用 ListMarkerFormat 类可以定义计数器。除了定义计数器的增量，您还可以利用 counterReset 属性重新设置计数器，从而对其进行自定义。

可以使用 ListMarkerFormat 的 beforeContent 和 afterContent 属性对列表中标记的外观进行进一步自定义。这些属性会应用到显示在标记内容之前和之后的内容。

下面的示例会在标记之前添加字符串 “XX”，在标记之后添加字符串 “YY”：

```
<flow:list listStyleType="upperRoman" paddingLeft="36" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="XX"
            afterContent="YY"
            counterIncrement="ordered -1"/>
        </flow:listMarkerFormat>
    <flow:li>Item 1</flow:li>
    <flow:li>Item 2</flow:li>
    <flow:li>Item 3</flow:li>
</flow:list>
```

content 属性本身可以定义标记格式的更多自定义内容。下面的示例会显示一个有序的、大写 Roman 数字标记：

```
<flow:list listStyleType="disc"  paddingLeft="96" paddingRight="24">
    <flow:listMarkerFormat>
        <flow:ListMarkerFormat fontSize="16"
            beforeContent="Section "
            content="counters(ordered,&quot;*&quot;,upperRoman)"
            afterContent=": "/>
    </flow:listMarkerFormat>
    <flow:li>Item 1</li>
    <flow:li>Item 2</li>
    <flow:li>Item 3</li>
</flow:list>
```

如上个示例所示， content 属性还可以插入一个后缀：一个在标记后、但在 afterContent 之前显示的字符串。若要在向流提供 XML 内容时插入此字符串，请使用 &quote; HTML 实体而不是引号 ("<***string***>") 来包括此字符串。

更多帮助主题

TLF 2.0 列表标记

在 TLF 中使用填充

每个 FlowElement 支持您使用的填充属性，来控制每个元素的内容区域的位置以及内容区域之间的间距。

某个元素的总宽度是其内容宽度的总和加上 paddingLeft 和 paddingRight 属性。某个元素的总高度是其内容高度的总和加上 paddingTop 和 paddingBottom 属性。

填充是指边框和内容之间的间距。填充属性包括有 paddingBottom、paddingTop、paddingLeft 和 paddingRight。可以将填充应用于 TextFlow 对象和以下子元素：

- div
- img
- li
- list
- p

填充属性不能应用到范围元素。

下面的示例将设置 TextFlow 的填充属性：

```
<flow:TextFlow version="2.0.0" xmlns:flow="http://ns.adobe.com/textLayout/2008" fontSize="14"
textIndent="15" paddingTop="4" paddingLeft="4" fontFamily="Times New Roman">
```

填充属性的有效值包括数值 （以像素为单位）、“auto” 或 “inherit”。默认值为 “auto”，表示该值是自动计算得出，对所有元素均设置为 0，ListElement 除外。对于 ListElements，“auto” 为 0，使用 listAutoPadding 属性的值所在列表的起始侧除外。listAutoPadding 的默认值为 40，这是为列表指定的默认缩进。

默认情况下，填充属性不会继承。“auto” 和 “inherit” 值是由 FormatValue 类定义的常数。

填充属性可以为负值。

更多帮助主题

TLF 2.0 中对填充所做的更改

使用 TLF 设置文本格式

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

flashx.textLayout.formats 包中包含一些接口和类，使用这些接口和类，您可以为文本流层次树中的任何 FlowElement 指定格式。可通过两种方式应用格式设置。您可以分别指定某特定格式，也可以利用特殊格式设置对象同时指定一组格式。

ITextLayoutFormat 接口包含可以应用于 FlowElement 的所有格式。有些格式应用于文本的某一完整容器或段落，但不会在逻辑上应用于各个字符。例如，字距调整和制表符这样的格式应用于全部段落，但不应用于各个字符。

使用属性为 FlowElement 指定格式

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

您可以通过属性分配对任何 FlowElement 设置格式。FlowElement 类实现 ITextLayoutFormat 接口，因此 FlowElement 类的任一子类也必须实现该接口。

例如，以下代码显示如何为 ParagraphElement 实例指定单种格式：

```
var p:ParagraphElement = new ParagraphElement();
p.fontSize = 18;
p.fontFamily = "Arial";
```

使用 TextLayoutFormat 类为 FlowElement 指定格式

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

您可以使用 TextLayoutFormat 类向 FlowElement 应用格式。可使用此类创建包含所需的所有格式设置值的特定格式设置对象。然后，将该对象指定给任何 FlowElement 对象的 format 属性。 TextLayoutFormat 和 FlowElement 都实现 ITextLayoutFormat 接口。这种安排确保两个类都包含同样的格式属性。

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 TextLayoutFormat。

格式继承

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

通过文本流层次结构继承格式。如果将 TextLayoutFormat 的实例指定给带有子代的 FlowElement 实例，框架将启动一个称为级联的过程。在级联过程中，框架递归检查从您的 FlowElement 继承的层次中的每个节点。然后，它确定是否将继承的值指定给各格式设置属性。在级联过程中应用下列规则：

1 属性值只从直接祖先 （有时称为父代）继承。

2 仅当属性尚未设置值 （即该值为 undefined）时，才继承属性值。

3 一些属性在未定义时不继承值，除非将属性的值设置为 " 继承 " 或常量 flashx.textLayout.formats.FormatValue.INHERIT。

例如，如果在 TextFlow 级别设置 fontSize 值，则该设置将应用于 TextFlow 中的所有元素。换句话说，这些值按照文本流层次向下级联。不过，您可以通过直接对给定元素指定新值来覆盖该元素中的值。举个反例，如果您在 TextFlow 级别设置 backgroundColor 值， TextFlow 的子项不继承该值。backgroundColor 属性不是级联期间从其父项继承的。通过将每一子项中的 backgroundColor 属性设置为 flashx.textLayout.formats.FormatValue.INHERIT，可以覆盖此行为 .

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 TextLayoutFormat。

使用 TLF 导入和导出文本

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

flashx.textLayout.conversion 中的 TextConverter 类。* 包允许您将文本导入到 TLF 以及从 TLF 导出文本。如果打算在运行时加载文本，而不是将文本编译到 SWF 文件中，请使用此类。您也可以使用此类将存储在 TextFlow 实例中的文本导出到一个字符串或 XML 对象。

导入和导出过程都直接完成。可以调用 export() 方法或 importToFlow() 方法，这两个方法都属于 TextConverter 类。这两个方法都是静态方法，意味着您对 TextConverter 类调用这两个方法，而非对 TextConverter 类的实例调用它们。

flashx.textLayout.conversion 包中的类在选择文本存储位置方面提供了极大的灵活性。例如，如果将文本存储在数据库中，您可以将文本导入该框架中以供显示。然后，可以使用 flashx.textLayout.edit 包中的类更改文本，并将更改后的文本导回到数据库中。

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 flashx.textLayout.conversion。

使用 TLF 管理文本容器

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

将文本存储在 TLF 数据结构中后， Flash Player 可以显示这些文本。必须将流层次结构中存储的文本转换为 Flash Player 可以显示的格式。 TLF 提供了两种从流中创建显示对象的方法。第一种方法较为简单，适用于显示静态文本。第二种方法较为复杂，允许您创建可进行选择和编辑的动态文本。使用这两种方法，文本最终都会转换为 TextLine 类的实例， TextLine 类保存在 Flash Player 10 中的 flash.text.engine.* 包中。

创建静态文本

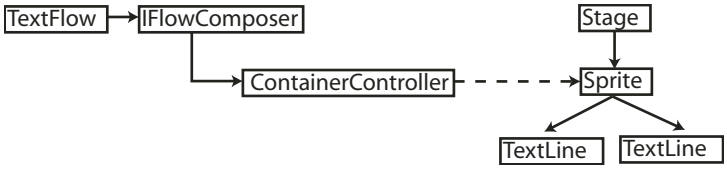
此简单方法使用 TextFlowTextLineFactory 类，该类位于 flashx.textLayout.factory 包中。此方法不仅简单，比 FlowComposer 方法占用的内存也少。建议对不需要用户编辑、选择或滚动的静态文本使用此方法。

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 TextFlowTextLineFactory。

创建动态文本和容器

如果您要加大对文本显示的控制程度 （而不是由 TextFlowTextLineFactory 提供的控制程度），请使用流合成器。例如，使用流合成器，用户可以选择和编辑文本。有关更多信息，请参阅第 371 页的 " 使用 TLF 启用文本选择、编辑和撤消 "。

流合成器是 flashx.textLayout.compose 包中的 StandardFlowComposer 类的实例。流合成器管理 TextFlow 到 TextLine 实例的转换，并负责将这些 TextLine 实例放置到一个或多个容器中。



IFlowComposer 包含零个或更多 ContainerController

每个 TextFlow 实例都有一个可实现 IFlowComposer 接口的相应对象。可以通过 TextFlow.flowComposer 属性访问此 IFlowComposer 对象。可以通过此属性调用 IFlowComposer 接口定义的方法。使用这些方法，可以将文本与一个或多个容器关联并准备好要在容器中显示的文本。

容器是 Sprite 类 （该类是 DisplayObjectContainer 类的子类）的实例。这两个类都包含在 Flash Player 显示列表 API 中。容器是定界矩形的更高级形式，与 TextLineFactory 类一起使用。与定界矩形类似，容器定义显示 TextLine 实例的区域。与定界矩形不同，容器具有对应的 " 控制器 " 对象。控制器管理一个容器或一组容器的滚动、合成、格式设置和事件处理。每个容器有一个对应的控制器对象，该对象是 flashx.textLayout.container 包中 ContainerController 类的实例。

若要显示文本，请创建一个控制器对象来管理该容器，并将其与流合成器关联。关联容器之后，编写文本以使文本能显示出来。相应地，容器有两种状态：编写和显示。编写是将文本流层次中的文本转换为 TextLine 实例并计算这些实例是否适合容器的过程。显示是更新 Flash Player 显示列表的过程。

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 IFlowComposer、StandardFlowComposer 和 ContainerController。

使用 TLF 启用文本选择、编辑和撤消

Flash Player 9.0 和更高版本， **Adobe AIR 1.0** 和更高版本

在文本流级别控制选择或编辑文本的能力。 TextFlow 类的每个实例都有一个关联的交互管理器。可通过对象的 TextFlow.interactionManager 属性访问 TextFlow 对象的交互管理器。要启用文本选择功能，可将 SelectionManager 类的实例指定给 interactionManager 属性。要启用文本选择和编辑功能，可指定 EditManager 类的实例，而不是指定 SelectionManager 类的实例。要启用撤消操作，可创建 UndoManager 类的一个实例，并将其纳为调用 EditManager 的构造函数时的参数。 UndoManager 类维护用户最新编辑活动的历史记录，并允许用户撤消或重做特定的编辑操作。上述所有三个类都包含在编辑包中。

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 SelectionManager、 EditManager 和 UndoManager。

使用 TLF 处理事件

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

在许多情况下， TextFlow 对象都会调度事件，包括：

- 当文本或布局发生更改时
- 在操作开始之前或操作完成之后
- 当 FlowElement 对象的状态发生更改时
- 当合成操作完成时

有关更多信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 flashx.textLayout.events。

更多帮助主题

TLF FlowElement 和 LinkElement 事件以及 EventMirrors

在文本内定位图像

若要在文本内定位 InlineGraphicElement，您可以使用以下属性：

- InlineGraphicElement 类的 float 属性
- FlowElement 的 clearFloats 属性

float 属性控制着图形及其周围文本的放置。 clearFloats 属性控制着段落元素相对于 float 的放置。

要控制文本元素内某个图像的位置，可以使用 float 属性。下面的示例向段落中添加一个图像，并将其与左侧对齐，从而使文本在右侧自动换行：

```
<flow:p paragraphSpaceAfter="15" >Images in a flow are a good thing. For example, here is a float. It should
show on the left: <flow:img float="left" height="50" width="19" source="../assets/bulldog.jpg"></flow:img>
Don't you agree? Another sentence here. Another sentence here. Another sentence here. Another sentence here.
Another sentence here. Another sentence here. Another sentence here. Another sentence here.</flow:p>
```

float 属性的有效值包括：“left”、“right”、“start”、“end” 和 “none”。 Float 类定义了这些常数。默认值为 “none”。

要为那些通常环绕在图像周围的后续段落调整起始位置， clearFloats 属性很有用。例如，假设您有一个图像，其尺寸大于第一个段落。要确保第二个段落起始于该图像后面，请设置 clearFloats 属性。

下面的示例使用的图像高于第一个段落中的文本。为使第二个段落起始于文本区块中的图像之后，此示例将第二个段落的 clearFloats 属性设置为 “end”。

```
<flow:p paragraphSpaceAfter="15" >Here is another float, it should show up on the right: <flow:img
float="right" height="50" elementHeight="200" width="19" source="../assets/bulldog.jpg"></flow:img>We'll
add another paragraph that should clear past it.</flow:p><flow:p clearFloats="end" >This should appear after
the previous float on the right.</flow:p>
```

clearFloats 属性的有效值包括：“left”、“right”、“end”、“start”、“none” 和 “both”。 ClearFloats 类定义这些常数。您还可以将 clearFloats 属性设置为 “inherit”，这是一个由 FormatValue 类定义的常数。默认值为 “none”。

更多帮助主题

TLF 浮动

第 24 章：处理声音

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 是为开发引人入胜的交互式应用程序而设计的，这种极其引人入胜的应用程序中经常被忽略的一个元素是声音。您可以在视频游戏中添加声音效果，在应用程序用户界面中添加音频回馈，甚至创建一个分析通过 Internet 加载的 mp3 文件的程序 （在这些情况下，声音是应用程序的核心）。

您可以加载外部音频文件并使用在 SWF 中嵌入的音频。您可以控制音频，创建声音信息的可视表示形式，以及从用户的麦克风捕获声音。

更多帮助主题

flash.media 包

flash.events.SampleDataEvent

声音处理基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

计算机可以捕获和编码数字音频 （声音信息的计算机表示形式），还可以存储和检索声音，从而通过扬声器播放。您可以使用 Adobe® Flash® Player 或 Adobe® AIR™ 以及 ActionScript 来播放声音。

将声音数据转换为数字形式后，它具有各种不同的特性，如声音的音量以及它是立体声还是单声道声音。在 ActionScript 中播放声音时，您也可以调整这些特性；例如，使声音变得更大，或者使其像是来自某个方向。

您需要先将声音信息加载到 Flash Player 或 AIR 中，才能在 ActionScript 中控制声音。可以使用五种方式将音频数据加载到 Flash Player 或 AIR 中，以便通过 ActionScript 使用这些数据。

- 将外部声音文件 （如 mp3 文件）加载到 SWF 中。
- 在创建 SWF 文件时将声音信息直接嵌入到其中。
- 使用连接到用户计算机的麦克风来获取音频。
- 从服务器流式传输音频。
- 动态生成并播放音频。

从外部声音文件加载声音数据时，您可以在仍加载其余声音数据的同时开始播放声音文件的开头部分。

虽然可以使用各种不同的声音文件格式对数字音频进行编码，但是 ActionScript 3.0、Flash Player 和 AIR 支持以 mp3 格式存储的声音文件。它们不能直接加载或播放 WAV 或 AIFF 等其他格式的声音文件。

在 ActionScript 中处理声音时，可能会使用 flash.media 包中的某些类。可以使用 Sound 类来访问音频信息：加载声音文件或为对声音数据进行采样的事件分配函数，然后开始播放。开始播放声音后， Flash Player 和 AIR 为您提供对 SoundChannel 对象的访问。因为已加载的音频文件只能是您在用户计算机上播放的几种声音之一，所以，所播放的每种单独的声音都使用自己的 SoundChannel 对象；混合在一起的所有 SoundChannel 对象的组合输出是实际通过计算机扬声器播放的声音。可以使用此 SoundChannel 实例来控制声音的属性以及使其停止播放。最后，如果要控制组合音频，您可以通过 SoundMixer 类对混合输出进行控制。

也可以使用其他几个类，在 ActionScript 中处理声音时执行更具体的任务；要了解与声音有关的所有类的详细信息，请参阅第 374 页的 “ 了解声音体系结构 ”。

重要概念和术语

以下参考列表包含您可能遇到的重要术语：

波幅 声音波形上的点与零或平衡线之间的距离。

比特率 每秒为声音文件编码或流式传输的数据量。对于 mp3 文件，比特率通常是以每秒千位数 (kbps) 来表述的。较高的比特率通常意味着较高品质的声音波形。

缓冲 在播放之前接收和存储声音数据。

mp3 MPEG-1 Audio Layer 3 或 mp3 是常见的声音压缩格式。

平移 音频信号在立体声声场中左右两声道之间的位置。

峰值 波形中的最高点。

采样率 定义每秒从模拟音频信号采集的用来组成数字信号的样本数。标准光盘音频的采样率为 44.1 kHz 或每秒 44,100 个样本。

流 此过程是指，在从服务器加载声音文件或视频文件的后面部分的同时播放该文件的前面部分。

卷 声音的响度。

波形 声音信号波幅随时间推移不断变化的图形形状。

了解声音体系结构

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可以从以下五种主要来源加载声音数据：

- 在运行时加载的外部声音文件
- 在应用程序的 SWF 文件中嵌入的声音资源
- 来自连接到用户系统上的麦克风的声音数据
- 从 Flash Media Server 等远程媒体服务器流式传输的声音数据
- 通过使用 sampleData 事件处理函数动态生成的声音数据

可以在播放之前完全加载声音数据，也可以进行流式传输，即在仍进行加载的同时播放这些数据。

ActionScript 3.0 声音类支持以 mp3 格式存储的声音文件。它们不能直接加载或播放其他格式 （如 WAV 或 AIFF）的声音文件。但从 Flash Player 9.0.115.0 开始，可使用 NetStream 类加载和播放 AAC 音频文件。这与加载和播放视频内容的方法相同。有关此方法的详细信息，请参阅第 401 页的 “ 使用视频 ”。

使用 Adobe Flash Professional，可以导入 WAV 或 AIFF 声音文件，然后将其以 mp3 格式嵌入应用程序的 SWF 文件中。Flash 创作工具还可压缩嵌入的声音文件以减小文件大小，但会降低声音的品质。有关详细信息，请参阅 《使用 Flash》中的 “ 导入声音 ”。

ActionScript 3.0 声音体系结构使用 flash.media 包中的以下类。

类	说明
flash.media.Sound	Sound 类处理声音加载、管理基本声音属性以及启动声音播放。
flash.media.SoundChannel	当应用程序播放 Sound 对象时，将创建一个新的 SoundChannel 对象来控制播放。SoundChannel 对象可控制声音的左和右播放声道的音量。播放的每种声音都具有其自己的 SoundChannel 对象。
flash.media.SoundLoaderContext	SoundLoaderContext 类指定在加载声音时使用的缓冲秒数，以及 Flash Player 或 AIR 在加载文件时是否从服务器中查找策略文件。 SoundLoaderContext 对象用作 Sound.load() 方法的参数。

类	说明
flash.media.SoundMixer	SoundMixer 类可控制与应用程序中的所有声音有关的播放和安全属性。实际上，可通过一个通用 SoundMixer 对象将多个声道混合在一起，因此，该 SoundMixer 对象中的属性值将影响当前播放的所有 SoundChannel 对象。
flash.media.SoundTransform	SoundTransform 类包含控制音量和声相的值。可以将 SoundTransform 对象应用于单个 SoundChannel 对象、全局 SoundMixer 对象或 Microphone 对象等。
flash.media.ID3Info	ID3Info 对象包含一些属性，它们表示通常存储在 mp3 声音文件中的 ID3 元数据信息。
flash.media.Microphone	Microphone 类表示连接到用户计算机上的麦克风或其他声音输入设备。可以将来自麦克风的音频输入传送到本地扬声器或发送到远程服务器。 Microphone 对象控制其自己的声音流的增益、采样率以及其他特性。
flash.media.AudioPlaybackMode	AudioPlaybackMode 类为 SoundMixer 类的 audioPlaybackMode 属性定义常量。

加载和播放的每种声音需要其自己的 Sound 类和 SoundChannel 类的实例。然后，全局 SoundMixer 类在播放期间将来自多个 SoundChannel 实例的输出混合在一起。

Sound、 SoundChannel 和 SoundMixer 类不能用于从麦克风或流媒体服务器 （如 Flash Media Server）中获取的声音数据。

加载外部声音文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Sound 类的每个实例都可加载并触发特定声音资源的播放。应用程序无法重复使用 Sound 对象来加载多种声音。如果它要加载新的声音资源，则应创建一个新的 Sound 对象。

如果要加载较小的声音文件 （如要附加到按钮上的单击声音），应用程序可以创建一个新的 Sound，并让其自动加载该声音文件，如下所示：

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

Sound() 构造函数接受一个 URLRequest 对象作为其第一个参数。在提供 URLRequest 参数的值后，新的 Sound 对象将自动开始加载指定的声音资源。

除了最简单的情况外，应用程序都应关注声音的加载进度，并监视在加载期间出现的错误。例如，如果单击声音非常大，在用户单击触发该声音的按钮时，该声音可能没有完全加载。尝试播放已卸载的声音可能会导致运行时错误。较为稳妥的做法是等待声音完全加载后，再让用户执行可能启动声音播放的动作。

Sound 对象将在声音加载过程中调度多种不同的事件。应用程序可以侦听这些事件以跟踪加载进度，并确保在播放之前完全加载声音。下表列出了可以由 Sound 对象调度的事件。

事件	说明
open (Event.OPEN)	在声音加载操作之前最后一刻进行调度。
progress (ProgressEvent.PROGRESS)	在从文件或流接收到数据之后，在声音加载过程中定期进行调度。
id3 (Event.ID3)	当存在可用于 mp3 声音的 ID3 数据时进行调度。
complete (Event.COMPLETE)	在加载所有声音资源的数据后进行调度。
ioError (IOErrorEvent.IO_ERROR)	在以下情况下进行调度：找不到声音文件，或者在收到所有声音数据之前加载过程中断。

以下代码说明了如何在完成加载后播放声音：

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(Event.COMPLETE, onSoundLoaded);
var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onSoundLoaded(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
```

首先，该代码范例创建一个新的 Sound 对象，但没有为其指定 URLRequest 参数的初始值。然后，它通过 Sound 对象侦听 Event.COMPLETE 事件，该对象导致在加载完所有声音数据后执行 onSoundLoaded() 方法。接下来，它使用新的 URLRequest 值为声音文件调用 Sound.load() 方法。

在加载完声音后，将执行 onSoundLoaded() 方法。Event 对象的 target 属性是对 Sound 对象的引用。如果调用 Sound 对象的 play() 方法，则会启动声音播放。

监视声音加载过程

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

声音文件可能很大，而需要花很长时间进行加载。尽管 Flash Player 和 AIR 允许应用程序甚至可以在完全加载声音之前播放声音，但您可能希望向用户指示已加载了多少声音数据以及已播放了多少声音。

Sound 类调度以下两个事件，它们可使声音加载进度显示变得相对比较简单：ProgressEvent.PROGRESS 和 Event.COMPLETE。以下示例说明了如何使用这些事件来显示有关所加载的声音的进度信息：

```
import flash.events.Event;
import flash.events.ProgressEvent;
import flash.media.Sound;
import flash.net.URLRequest;

var s:Sound = new Sound();
s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
s.addEventListener(Event.COMPLETE, onLoadComplete);
s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);

var req:URLRequest = new URLRequest("bigSound.mp3");
s.load(req);

function onLoadProgress(event:ProgressEvent):void
{
    var loadedPct:uint = Math.round(100 * (event.bytesLoaded / event.bytesTotal));
    trace("The sound is " + loadedPct + "% loaded.");
}

function onLoadComplete(event:Event):void
{
    var localSound:Sound = event.target as Sound;
    localSound.play();
}
function onIOError(event:IOErrorEvent)
{
    trace("The sound could not be loaded: " + event.text);
}
```

此代码先创建一个 Sound 对象，然后向该对象添加侦听器以侦听 ProgressEvent.PROGRESS 和 Event.COMPLETE 事件。在调用 Sound.load() 方法并从声音文件接收第一批数据后，将会发生 ProgressEvent.PROGRESS 事件并触发 onSoundLoadProgress() 方法。

已加载的声音数据百分比等于 ProgressEvent 对象的 bytesLoaded 属性值除以 bytesTotal 属性值。Sound 对象上也提供了相同的 bytesLoaded 和 bytesTotal 属性。以上示例仅显示了关于声音加载进度的消息，但是您可以轻松地使用 bytesLoaded 和 bytesTotal 值更新进度栏组件，例如 Adobe Flex 框架或 Adobe Flash 创作工具附带的组件。

此示例还说明了应用程序在加载声音文件时如何识别并响应出现的错误。例如，如果找不到具有给定文件名的声音文件，Sound 对象将调度一个 Event.IO_ERROR 事件。在上面的代码中，当发生错误时，将执行 onIOError() 方法并显示一条简短的错误消息。

处理嵌入的声音

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

对于在应用程序用户界面中用作指示器的较小声音 （如在单击按钮时播放的声音），最适合使用嵌入的声音 （而不是从外部文件加载声音）。

在应用程序中嵌入声音文件时，生成的 SWF 文件大小比原来增加了声音文件的大小。也就是说，如果在应用程序中嵌入较大的声音文件，可能会使 SWF 文件增大到难以接受的大小。

将声音文件嵌入到应用程序的 SWF 文件中的具体方法因开发环境而异。

在 Flash 中使用嵌入的声音文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash 创作工具可导入多种声音格式的声音并将其作为元件存储在库中。然后，您可以将其分配给时间轴上的帧或按钮状态的帧，通过行为来使用声音，或直接在 ActionScript 代码中使用它们。本节说明如何在 ActionScript 代码中通过 Flash 创作工具来使用嵌入的声音。有关如何以其他方法在 Flash 中使用嵌入的声音的信息，请参阅 《使用 Flash》中的 “ 导入声音 ”。

使用 **Flash** 创作工具嵌入声音文件：

1. 选择 “ 文件 ”>“ 导入 ”>“ 导入到库 ”，然后选择一个声音文件并导入它。
2. 在 “ 库 ” 面板中，右键单击导入的文件的名称，然后选择 “ 属性 ”。单击 “ 为 ActionScript 导出 ” 复选框。
3. 在 “ 类 ” 字段中，输入一个名称，以便在 ActionScript 中引用此嵌入的声音时使用。默认情况下，它将使用此字段中声音文件的名称。如果文件名包含句点 （如名称 “DrumSound.mp3”），则必须将其更改为类似于 “DrumSound” 这样的名称 ； ActionScript 不允许在类名称中出现句点字符。 “ 基类 ” 字段应仍显示 flash.media.Sound。
4. 单击 “ 确定 ”。可能出现一个对话框，指出无法在类路径中找到该类的定义。单击 “ 确定 ” 以继续。如果输入的类名称与应用程序的类路径中任何类的名称都不匹配，则会自动生成从 flash.media.Sound 类继承的新类。
5. 若要使用嵌入的声音，请在 ActionScript 中引用该声音的类名称。例如，通过创建自动生成的 DrumSound 类的一个新实例来启动以下代码：

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound 是 flash.media.Sound 类的子类，所以它继承了 Sound 类的方法和属性，包括上面显示的 play() 方法。

在 Flex 中使用嵌入的声音文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有多种在 Flex 应用程序中嵌入声音资源的方法，包括：

- 在脚本中使用 [Embed] 元数据标签
- 在 MXML 中使用 @Embed 指令将嵌入的资源分配为组件 （如 Button 或 SoundEffect）的属性。
- 在 CSS 文件中使用 @Embed 指令

本节介绍第一种方法：如何使用 [Embed] 元数据标签在 Flex 应用程序的 ActionScript 代码中嵌入声音。

若要在 ActionScript 代码中嵌入资源，请使用 [Embed] 元数据标签。

将声音文件放置在项目生成路径中的主资源文件夹或其他文件夹中。编译器遇到 Embed 元数据标签时，会创建嵌入资源类。通过紧接 [Embed] 元数据标签之后声明的数据类型 Class 的变量，可以访问该资源类。

下面的代码嵌入名为 smallSound.mp3 的声音，使用名为 soundClass 的变量来存储对该声音的关联嵌入资源类的引用。然后，该代码创建该嵌入资源类的一个实例，将其转换为 Sound 类的实例，并在该实例上调用 play() 方法：

```
package
{
    import flash.display.Sprite;
    import flash.media.Sound;
    import flash.media.SoundChannel;

    public class EmbeddedSoundExample extends Sprite
    {
        [Embed(source="smallSound.mp3")]
        public var soundClass:Class;

        public function EmbeddedSoundExample()
        {
            var smallSound:Sound = new soundClass() as Sound;
            smallSound.play();
        }
    }
}
```

若要使用嵌入的声音设置 Flex 组件的属性，应将该声音转换为 mx.core.SoundAsset 类的实例 （而不是 Sound 类的实例）。有关使用 SoundAsset 类的类似示例，请参阅 《学习 ActionScript 3.0》中的 " 嵌入资源类 "。

处理声音流文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果在声音文件或视频文件的数据加载过程中播放该文件，则称为 " 流式传输 "。通常，将对从远程服务器加载的外部声音文件进行流式传输，以使用户不必等待加载完所有声音数据再收听声音。

SoundMixer.bufferTime 属性表示 Flash Player 或 AIR 在允许播放声音之前应收集多长时间的声音数据 （以毫秒为单位）。也就是说，如果将 bufferTime 属性设置为 5000，则在开始播放声音之前，Flash Player 或 AIR 会从声音文件中加载至少相当于 5000 毫秒的数据。 SoundMixer.bufferTime 默认值为 1000。

通过在加载声音时明确地指定新的 bufferTime 值，应用程序可以覆盖单个声音的全局 SoundMixer.bufferTime 值。要覆盖默认缓冲时间，请先创建一个新的 SoundLoaderContext 类实例，设置其 bufferTime 属性，然后将其作为参数传递给 Sound.load() 方法，如下所示：

```
import flash.media.Sound;
import flash.media.SoundLoaderContext;
import flash.net.URLRequest;

var s:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
var context:SoundLoaderContext = new SoundLoaderContext(8000, true);
s.load(req, context);
s.play();
```

当播放继续进行时，Flash Player 和 AIR 尝试将声音缓冲保持在相同大小或更大。如果声音数据的加载速度比播放快，播放将继续进行而不会中断。但是，如果数据加载速率由于网络限制而减慢，播放头可能会到达声音缓冲区的结尾。如果发生这种情况，播放会暂停，但播放会在已加载更多声音数据后自动恢复。

若要查明暂停播放是否是由于 Flash Player 或 AIR 正在等待加载数据造成的，请使用 Sound.isBuffering 属性。

处理动态生成的音频

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

注：从 Flash Player 10 和 Adobe AIR 1.5 开始可以动态生成音频。

可以动态生成音频数据，而不是加载或流式传输现有声音。在为 Sound 对象的 sampleData 事件分配事件侦听器时，可以生成音频数据。（sampleData 事件在 flash.events 包的 SampleDataEvent 类中定义。）在这种情况下， Sound 对象不从文件中加载声音数据。相反，该对象将用作声音数据的套接字，声音数据通过使用您分配给此事件的函数流入该对象。

在您将 sampleData 事件侦听器添加到 Sound 对象后，该对象将定期请求数据以添加到声音缓冲区。此缓冲区包含 Sound 对象要播放的数据。在调用 Sound 对象的 play() 方法时，它会在请求新的声音数据时调度 sampleData 事件。（只有在 Sound 对象尚未从文件加载 mp3 数据时，此操作才生效。）

SampleDataEvent 对象包含 data 属性。在事件侦听器中，将 ByteArray 对象写入此 data 对象。写入此对象的字节数组将添加到 Sound 对象播放的缓冲声音数据中。缓冲区中的字节数组是由从 -1 到 1 的浮点值组成的流。各浮点值均代表声音样本的一个声道 （左声道或右声道）的幅度。声音按每秒 44,100 个样本进行采样。每个样本均包含左声道和右声道，在字节数组中以浮点数据的形式交错排列。

在处理函数中，使用 ByteArray.writeFloat() 方法写入 sampleData 事件的 data 属性。例如，以下代码将生成正弦波：

```
var mySound:Sound = new Sound();
mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, sineWaveGenerator);
mySound.play();
function sineWaveGenerator(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192; i++)
    {
        var n:Number = Math.sin((i + event.position) / Math.PI / 4);
        event.data.writeFloat(n);
        event.data.writeFloat(n);
    }
}
```

当您调用 Sound.play() 时，该应用程序将开始调用事件处理函数，并请求声音样本数据。在播放声音时，应用程序将继续发送事件，直至您停止提供数据或调用 SoundChannel.stop()。

事件的滞后时间因平台而异，在以后的 Flash Player 和 AIR 版本中也可能改变。请不要依赖某个特定的滞后时间；而应计算出相应的滞后时间。若要计算滞后时间，请使用以下公式：

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

向 SampleDataEvent 对象的 data 属性提供 2048 到 8192 个样本 （对于每次事件侦听器调用）。为了获得最佳性能，请尽可能多地提供样本 （最多可达 8192 个样本）。提供的样本越少，在播放过程中就越有可能出现单击和弹出事件。此行为对于不同的平台会有所不同，并且会在各种情况下发生。例如，在调整浏览器的大小时。在仅提供了 2048 个样本时，工作在某一个平台上的代码可能在运行于其他不同平台时将不能很好地工作。若要尽可能缩短滞后时间，请考虑允许用户选择数据量。

如果提供的样本少于 2048 个 （每次 sampleData 事件侦听器调用），则应用程序将在播放完剩余的样本后停止。然后，SoundChannel 对象调度 SoundComplete 事件。

对源自 mp3 数据的声音数据进行修改

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

使用 Sound.extract() 方法提取 Sound 对象中的数据。可以使用 （和修改）该数据，将其写入另一个 Sound 对象的动态流以进行播放。例如，以下代码使用加载的 MP3 文件的字节，并通过过滤函数 upOctave() 进行传递：

```
var mySound:Sound = new Sound();
var sourceSnd:Sound = new Sound();
var urlReq:URLRequest = new URLRequest("test.mp3");
sourceSnd.load(urlReq);
sourceSnd.addEventListener(Event.COMPLETE, loaded);
function loaded(event:Event):void
{
    mySound.addEventListener(SampleDataEvent.SAMPLE_DATA, processSound);
    mySound.play();
}
function processSound(event:SampleDataEvent):void
{
        var bytes:ByteArray = new ByteArray();
        sourceSnd.extract(bytes, 8192);
        event.data.writeBytes(upOctave(bytes));
}
function upOctave(bytes:ByteArray):ByteArray
{
    var returnBytes:ByteArray = new ByteArray();
    bytes.position = 0;
    while(bytes.bytesAvailable > 0)
    {
        returnBytes.writeFloat(bytes.readFloat());
        returnBytes.writeFloat(bytes.readFloat());
        if (bytes.bytesAvailable > 0)
        {
            bytes.position += 8;
        }
    }
    return returnBytes;
}
```

有关生成声音的限制

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

将 sampleData 事件侦听器与 Sound 对象一起使用时，启用的其他 Sound 方法仅包括 Sound.extract() 和 Sound.play()。调用任何其他方法或属性将导致异常。仍启用 SoundChannel 对象的所有方法和属性。

播放声音

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

播放加载的声音非常简便，您只需为 Sound 对象调用 Sound.play() 方法，如下所示：

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

使用 ActionScript 3.0 播放声音时，您可以执行以下操作：

- 从特定起始位置播放声音
- 暂停声音并稍后从相同位置恢复播放
- 准确了解何时播放完声音
- 跟踪声音的播放进度
- 在播放声音的同时更改音量或声相

若要在播放期间执行这些操作，请使用 SoundChannel、 SoundMixer 和 SoundTransform 类。

SoundChannel 类控制一种声音的播放。可以将 SoundChannel.position 属性视为播放头，以指示所播放的声音数据中的当前位置。

当应用程序调用 Sound.play() 方法时，将创建一个新的 SoundChannel 类实例来控制播放。

通过将特定起始位置 （以毫秒为单位）作为 Sound.play() 方法的 startTime 参数进行传递，应用程序可以从该位置播放声音。它也可以通过在 Sound.play() 方法的 loops 参数中传递一个数值，指定快速且连续地将声音重复播放固定的次数。

使用 startTime 参数和 loops 参数调用 Sound.play() 方法时，每次将从相同的起始点重复播放声音，如以下代码中所示：

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

在此示例中，从声音开始后的 1 秒起连续播放声音三次。

暂停和恢复播放声音

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果应用程序播放很长的声音 （如歌曲或播客），您可能需要让用户暂停和恢复播放这些声音。实际上，无法在 ActionScript 中的播放期间暂停声音；而只能将其停止。但是，可以从任何位置开始播放声音。您可以记录声音停止时的位置，并随后从该位置开始重放声音。

例如，假定代码加载并播放一个声音文件，如下所示：

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

在播放声音的同时， SoundChannel.position 属性指示当前播放到的声音文件位置。应用程序可以在停止播放声音之前存储位置值，如下所示：

```
var pausePosition:int = channel.position;
channel.stop();
```

若要恢复播放声音，请传递以前存储的位置值，以便从声音以前停止的相同位置重新启动声音。

```
channel = snd.play(pausePosition);
```

监视播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可能需要了解何时停止播放某种声音，以便开始播放另一种声音，或者清除在以前播放期间使用的某些资源。SoundChannel 类在其声音完成播放时将调度 Event.SOUND_COMPLETE 事件。应用程序可以侦听此事件并执行相应的动作，如下所示：

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("smallSound.mp3");
snd.load(req);

var channel:SoundChannel = snd.play();
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

public function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
}
```

SoundChannel 类在播放期间不调度 progress 事件。若要报告播放进度，应用程序可以设置其自己的计时机制并跟踪声音播放头的位置。

若要计算已播放的声音百分比，您可以将 SoundChannel.position 属性值除以所播放的声音数据长度：

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

但是，仅当在开始播放之前完全加载了声音数据时，此代码才会报告精确的播放百分比。 Sound.length 属性显示当前加载的声音数据的大小，而不是整个声音文件的最终大小。若要跟踪仍在加载的声音流的播放进度，应用程序应估计完整声音文件的最终大小，并在其计算中使用该值。您可以使用 Sound 对象的 bytesLoaded 和 bytesTotal 属性来估计声音数据的最终长度，如下所示：

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

下面的代码加载一个较大的声音文件，并使用 Event.ENTER_FRAME 事件作为其计时机制来显示播放进度。它定期报告播放百分比，这是作为当前位置值除以声音数据的总长度来计算的：

```
import flash.events.Event;
import flash.media.Sound;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new
    URLRequest("http://av.adobe.com/podcast/csbu_dev_podcast_epi_2.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

function onEnterFrame(event:Event):void
{
    var estimatedLength:int =
        Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
    var playbackPercent:uint =
        Math.round(100 * (channel.position / estimatedLength));
    trace("Sound playback is " + playbackPercent + "% complete.");
}

function onPlaybackComplete(event:Event)
{
    trace("The sound has finished playing.");
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

在开始加载声音数据后，此代码会调用 snd.play() 方法，并将生成的 SoundChannel 对象存储在 channel 变量中。随后，此代码在主应用程序中添加 Event.ENTER_FRAME 事件的事件侦听器，并在 SoundChannel 对象中添加另一个事件侦听器，用于侦听在播放完成时发生的 Event.SOUND_COMPLETE 事件。

每次应用程序到达其动画中的新帧时，将调用 onEnterFrame() 方法。onEnterFrame() 方法基于已加载的数据量来估计声音文件的总长度，然后计算并显示当前播放百分比。

当播放完整个声音后，将执行 onPlaybackComplete() 方法来移除 Event.ENTER_FRAME 事件的事件侦听器，以使其在完成播放后不会尝试显示进度更新。

可以每秒多次调度 Event.ENTER_FRAME 事件。在某些情况下，您不需要频繁显示播放进度。在这些情况下，应用程序可以使用 flash.util.Timer 类来设置其自己的计时机制；请参阅第 1 页的 “ 使用日期和时间 ”。

停止声音流

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在进行流式传输的声音 （即，在播放的同时仍在加载声音）的播放过程中，有一个奇怪的现象。当应用程序对播放声音流的 SoundChannel 实例调用 SoundChannel.stop() 方法时，声音播放在一个帧处停止，随后在下一帧处从声音开头重新播放。发生这种情况是因为，声音加载过程仍在进行当中。若要同时停止声音流加载和播放，请调用 Sound.close() 方法。

加载和播放声音时的安全注意事项

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以根据 Flash Player 或 AIR 安全模型来限制应用程序访问声音数据的能力。每种声音受两种不同的安全沙箱的限制：内容本身的沙箱 （“ 内容沙箱 ”）以及加载和播放声音的应用程序或对象的沙箱 （“ 所有者沙箱 ”）。对于应用程序安全沙箱中的 AIR 应用程序内容，应用程序安全沙箱中的内容可以访问所有声音 （包括从其他域加载的声音）。但是，其他安全沙箱中的内容与在 Flash Player 中运行的内容遵守相同的规则。有关 Flash Player 安全模型概述的详细信息以及沙箱定义，请参阅第 894 页的 “ 安全性 ”。

内容沙箱控制使用 id3 属性还是 SoundMixer.computeSpectrum() 方法从声音提取详细声音数据。它不会限制声音文件本身的加载或播放。

声音文件的原始域定义了内容沙箱的安全限制。一般来说，如果某个声音文件与加载该文件的应用程序或对象的 SWF 文件位于相同的域或文件夹中，则应用程序或对象具有该声音文件的完全访问权限。如果声音来自不同于应用程序所在域的域，仍可以使用策略文件将其加载到内容沙箱中。

应用程序可以将带有 checkPolicyFile 属性的 SoundLoaderContext 对象作为参数传递给 Sound.load() 方法。如果将 checkPolicyFile 属性设置为 true，则会通知 Flash Player 或 AIR 在从中加载声音的服务器上查找策略文件。如果存在策略文件，并且它为执行加载的 SWF 文件所在的域授予了访问权限，则该 SWF 文件可以加载声音文件，访问 Sound 对象的 id3 属性以及为加载的声音调用 SoundMixer.computeSpectrum() 方法。

所有者沙箱控制声音的本地播放。所有者沙箱是由开始播放声音的应用程序或对象定义的。

只要当前播放的所有 SoundChannel 对象中的声音符合以下条件， SoundMixer.stopAll() 方法就会将它们静音：

- 声音是由相同所有者沙箱中的对象启动的。
- 声音来自具有某一策略文件的源，该策略文件为调用 SoundMixer.stopAll() 方法的应用程序或对象所在的域授予访问权限。

但在 AIR 应用程序中，应用程序安全沙箱中的内容 （随 AIR 应用程序安装的内容）不受这些安全限制的约束。

要查明 SoundMixer.stopAll() 方法是否确实停止了所有播放的声音，应用程序可以调用 SoundMixer.areSoundsInaccessible() 方法。如果该方法返回值 true，则当前所有者沙箱无法控制播放的某些声音， SoundMixer.stopAll() 方法不会将其停止。

SoundMixer.stopAll() 方法还会阻止播放头继续播放从外部文件加载的所有声音。但是，如果动画移动到一个新帧， FLA 文件中嵌入的声音以及使用 Flash 创作工具附加到时间轴中的帧上的声音可能会重新开始播放。

控制音量和声相

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

单个 SoundChannel 对象可同时控制声音的左立体声声道和右立体声声道。如果 mp3 声音是单声道声音， SoundChannel 对象的左和右立体声声道将包含完全相同的波形。

可通过使用 SoundChannel 对象的 leftPeak 和 rightPeak 属性来查明所播放的声音的每个立体声声道的波幅。这些属性显示声音波形本身的峰值波幅。它们并不表示实际播放音量。实际播放音量是声音波形的波幅以及 SoundChannel 对象和 SoundMixer 类中设置的音量值的函数。

在播放期间，可以使用 SoundChannel 对象的 pan 属性为左声道和右声道分别指定不同的音量级别。 pan 属性可以具有范围从 -1 到 1 的值，其中，-1 表示左声道以最大音量播放，而右声道处于静音状态； 1 表示右声道以最大音量播放，而左声道处于静音状态。介于 -1 和 1 之间的数值为左和右声道值设置一定比例的值，值 0 表示两个声道以均衡的中音量级别播放。

以下代码示例使用 volume 值 0.6 和 pan 值 -1 创建一个 SoundTransform 对象（左声道为最高音量，右声道没有音量）。此代码将 SoundTransform 对象作为参数传递给 play() 方法，此方法将该 SoundTransform 对象应用于为控制播放而创建的新 SoundChannel 对象。

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var trans:SoundTransform = new SoundTransform(0.6, -1);
var channel:SoundChannel = snd.play(0, 1, trans);
```

可以在播放声音的同时更改音量和声相控制，方法是设置 SoundTransform 对象的 pan 或 volume 属性，然后将该对象作为 SoundChannel 对象的 soundTransform 属性进行应用。

也可以通过使用 SoundMixer 类的 soundTransform 属性，同时为所有声音设置全局音量和声相值，如以下示例所示：

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

也可以使用 SoundTransform 对象为 Microphone 对象设置 volume 和 pan 值（请参阅第 389 页的 “ 捕获声音输入 ”），并可以为 Sprite 对象和 SimpleButton 对象设置这些值。

以下示例在播放声音的同时将声音从左声道移到右声道，然后再移回来，并交替进行这一过程。

```
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var panCounter:Number = 0;

var trans:SoundTransform;
trans = new SoundTransform(1, 0);
var channel:SoundChannel = snd.play(0, 1, trans);
channel.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

addEventListener(Event.ENTER_FRAME, onEnterFrame);

function onEnterFrame(event:Event):void
{
    trans.pan = Math.sin(panCounter);
    channel.soundTransform = trans; // or SoundMixer.soundTransform = trans;
    panCounter += 0.05;
}

function onPlaybackComplete(event:Event):void
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

此代码先加载一个声音文件，然后将 volume 设置为 1 （最大音量）并将 pan 设置为 0 （声音在左和右声道之间均衡地平均分布）以创建一个新的 SoundTransform 对象。接下来，此代码调用 snd.play() 方法，以将 SoundTransform 对象作为参数进行传递。

在播放声音时，将反复执行 onEnterFrame() 方法。 onEnterFrame() 方法使用 Math.sin() 函数来生成介于 -1 和 1 之间的值，此范围对应于可接受的 SoundTransform.pan 属性值。此代码将 SoundTransform 对象的 pan 属性设置为新值，然后设置声道的 soundTransform 属性以使用更改后的 SoundTransform 对象。

要运行此示例，请用本地 mp3 文件的名称替换文件名 bigSound.mp3。然后，运行该示例。当右声道音量变小时，您应会听到左声道音量变大，反之亦然。

在此示例中，可通过设置 SoundMixer 类的 soundTransform 属性来获得同样的效果。但是，这会影响当前播放的所有声音的声相，而不是只影响此 SoundChannel 对象播放的一种声音。

处理声音元数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 mp3 格式的声音文件可以采用 ID3 标签格式来包含有关声音的其他数据。

并非每个 mp3 文件都包含 ID3 元数据。当 Sound 对象加载 mp3 声音文件时，如果该声音文件包含 ID3 元数据，它将调度 Event.ID3 事件。若要防止出现运行时错误，应用程序应等待接收 Event.ID3 事件后，再访问加载的声音的 Sound.id3 属性。

以下代码说明了如何识别何时加载了声音文件的 ID3 元数据：

```
import flash.events.Event;
import flash.media.ID3Info;
import flash.media.Sound;

var s:Sound = new Sound();
s.addEventListener(Event.ID3, onID3InfoReceived);
s.load("mySound.mp3");

function onID3InfoReceived(event:Event)
{
    var id3:ID3Info = event.target.id3;

    trace("Received ID3 Info:");
    for (var propName:String in id3)
    {
        trace(propName + " = " + id3[propName]);
    }
}
```

此代码先创建一个 Sound 对象并通知该对象侦听 Event.ID3 事件。加载声音文件的 ID3 元数据后，将调用 onID3InfoReceived() 方法。传递给 onID3InfoReceived() 方法的 Event 对象的目标是原始 Sound 对象，因此，该方法随后获取 Sound 对象的 id3 属性，然后循环访问其所有命名属性以跟踪这些属性的值。

访问原始声音数据

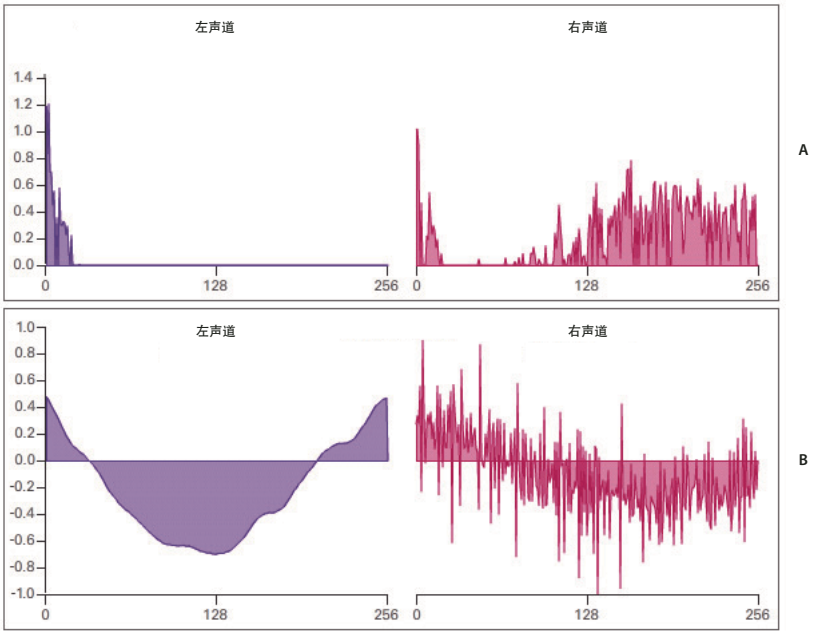
Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过使用 SoundMixer.computeSpectrum() 方法，应用程序可以读取当前所播放的波形的原始声音数据。如果当前播放多个 SoundChannel 对象， SoundMixer.computeSpectrum() 方法将显示混合在一起的每个 SoundChannel 对象的组合声音数据。

声音数据是作为 ByteArray 对象 （包含 512 个字节的数据）返回的，其中的每个字节包含一个介于 -1 和 1 之间的浮点值。这些值表示所播放的声音波形中的点的波幅。这些值是分为两个组 （每组包含 256 个值）提供的，第一个组用于左立体声声道，第二个组用于右立体声声道。

如果将 FFTMode 参数设置为 true， SoundMixer.computeSpectrum() 方法将返回频谱数据，而非波形数据。频谱显示按声音频率 （从最低频率到最高频率）排列的波幅。可以使用快速傅立叶变换 (FFT) 将波形数据转换为频谱数据。生成的频谱值范围介于 0 和约 1.414 （2 的平方根）之间。

下图比较了将 FFTMode 参数设置为 true 和 false 时从 computeSpectrum() 方法返回的数据。此图所用数据的声音在左声道中包含很大的低音；而在右声道中包含击鼓声。



SoundMixer.computeSpectrum() 方法返回的值
A. fftMode=true **B.** fftMode=false

computeSpectrum() 方法也可以返回已在较低比特率重新采样的数据。通常，这会产生更平滑的波形数据或频率数据，但会以牺牲细节为代价。 stretchFactor 参数控制 computeSpectrum() 方法数据的采样率。如果将 stretchFactor 参数设置为 0 （默认值），则以采样率 44.1 kHz 采集声音数据样本。stretchFactor 参数值每连续增加 1，采样率就减小一半，因此，值 1 指定采样率 22.05 kHz，值 2 指定采样率 11.025 kHz，依此类推。当使用较高的 stretchFactor 值时，computeSpectrum() 方法仍会为每个立体声声道返回 256 个字节。

SoundMixer.computeSpectrum() 方法具有一些限制：

- 由于来自麦克风或 RTMP 流的声音数据不是通过全局 SoundMixer 对象传递的，因此， SoundMixer.computeSpectrum() 方法不会从这些源返回数据。
- 如果播放的一种或多种声音来自当前内容沙箱以外的源，安全限制将导致 SoundMixer.computeSpectrum() 方法引发错误。有关 SoundMixer.computeSpectrum() 方法的安全限制的更多详细信息，请参阅第 384 页的 “ 加载和播放声音时的安全注意事项 ” 和第 913 页的 “ 作为数据访问加载的媒体 ”。

但在 AIR 应用程序中，应用程序安全沙箱中的内容 （随 AIR 应用程序安装的内容）不受这些安全限制的约束。

构建简单的声音可视化程序

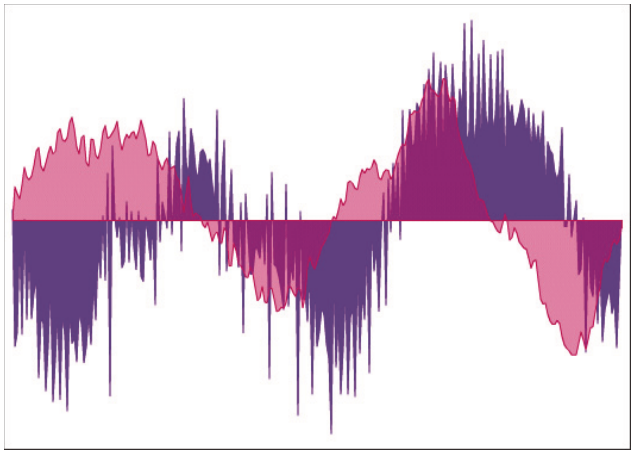
Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下示例使用 SoundMixer.computeSpectrum() 方法来显示声音波形图 （它对每个帧进行动画处理）：

```
import flash.display.Graphics;
import flash.events.Event;
import flash.media.Sound;
import flash.media.SoundChannel;
import flash.media.SoundMixer;
import flash.net.URLRequest;

const PLOT_HEIGHT:int = 200;
const CHANNEL_LENGTH:int = 256;

var snd:Sound = new Sound();
var req:URLRequest = new URLRequest("bigSound.mp3");
snd.load(req);

var channel:SoundChannel;
channel = snd.play();
addEventListener(Event.ENTER_FRAME, onEnterFrame);
snd.addEventListener(Event.SOUND_COMPLETE, onPlaybackComplete);

var bytes:ByteArray = new ByteArray();

function onEnterFrame(event:Event):void
{
    SoundMixer.computeSpectrum(bytes, false, 0);

    var g:Graphics = this.graphics;

    g.clear();
    g.lineStyle(0, 0x6600CC);
    g.beginFill(0x6600CC);
    g.moveTo(0, PLOT_HEIGHT);

    var n:Number = 0;

    // left channel
    for (var i:int = 0; i < CHANNEL_LENGTH; i++)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);
    g.endFill();

    // right channel
    g.lineStyle(0, 0xCC0066);
    g.beginFill(0xCC0066, 0.5);
    g.moveTo(CHANNEL_LENGTH * 2, PLOT_HEIGHT);

    for (i = CHANNEL_LENGTH; i > 0; i--)
    {
        n = (bytes.readFloat() * PLOT_HEIGHT);
        g.lineTo(i * 2, PLOT_HEIGHT - n);
    }
    g.lineTo(0, PLOT_HEIGHT);
    g.endFill();
}

function onPlaybackComplete(event:Event)
{
    removeEventListener(Event.ENTER_FRAME, onEnterFrame);
}
```

此示例先加载并播放一个声音文件，然后在播放声音的同时侦听将触发 onEnterFrame() 方法的 Event.ENTER_FRAME 事件。onEnterFrame() 方法先调用 SoundMixer.computeSpectrum() 方法，后者将声音波形数据存储在 bytes ByteArray 对象中。

声音波形是使用矢量绘图 API 绘制的。 for 循环将循环访问第一批 256 个数据值 （表示左立体声声道），然后使用 Graphics.lineTo() 方法绘制一条从每个点到下一个点的直线。第二个 for 循环将循环访问下一批 256 个值，此时按相反的顺序（从右到左）对它们进行绘制。生成的波形图可能会产生非常有趣的镜像图像效果，如以下图像中所示。



捕获声音输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可通过 Microphone 类连接到用户系统上的麦克风或其他声音输入设备，并将输入音频广播到该系统的扬声器，或者将音频数据发送到远程服务器 （如 Flash Media Server）。您可以从麦克风访问原始音频数据并进行记录或处理；还可以直接将音频发送到系统的扬声器或将压缩的音频数据发送到远程服务器。对于发送到远程服务器的数据，可以使用 Speex 或 Nellymoser 编解码器。（从 Flash Player 10 和 Adobe AIR 1.5 开始支持 Speex 编解码器。）

更多帮助主题

Michael Chaize：AIR、 Android 和麦克风

Christophe Coenraets：Android 语音注意事项

访问麦克风

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Microphone 类没有构造函数方法。相反，应使用静态 Microphone.getMicrophone() 方法来获取新的 Microphone 实例，如下所示：

```
var mic:Microphone = Microphone.getMicrophone();
```

不使用参数调用 Microphone.getMicrophone() 方法时，将返回在用户系统上发现的第一个声音输入设备。

系统可能连接了多个声音输入设备。应用程序可以使用 Microphone.names 属性来获取所有可用声音输入设备名称的数组。然后，它可以使用 index 参数 （与数组中的设备名称的索引值相匹配）来调用 Microphone.getMicrophone() 方法。

系统可能没有连接麦克风或其他声音输入设备。可以使用 Microphone.names 属性或 Microphone.getMicrophone() 方法来检查用户是否安装了声音输入设备，则 names 数组的长度为零，并且 getMicrophone() 方法返回值 null。

当应用程序调用 Microphone.getMicrophone() 方法时， Flash Player 将显示 “Flash Player 设置 ” 对话框，它提示用户允许或拒绝 Flash Player 对系统上的摄像头和麦克风的访问。在用户单击此对话框中的 “ 允许 ” 或 “ 拒绝 ” 按钮后，将调度 StatusEvent。该 StatusEvent 实例的 code 属性指示是允许还是拒绝对麦克风的访问，如下例所示：

```
import flash.media.Microphone;

var mic:Microphone = Microphone.getMicrophone();
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

function onMicStatus(event:StatusEvent):void
{
    if (event.code == "Microphone.Unmuted")
    {
        trace("Microphone access was allowed.");
    }
    else if (event.code == "Microphone.Muted")
    {
        trace("Microphone access was denied.");
    }
}
```

如果允许访问，则 StatusEvent.code 属性将包含 “Microphone.Unmuted”，如果拒绝访问，则该属性包含 “Microphone.Muted”。

当用户允许或拒绝对麦克风的访问时，Microphone.muted 属性将被分别设置为 true 或 false。但是，在调度 StatusEvent 之前，Microphone 实例上未设置 muted 属性，因此在检查 Microphone.muted 属性之前，应用程序还应等待调度 StatusEvent.STATUS 事件。

要使 Flash Player 显示设置对话框，应用程序窗口的大小必须足以显示它 （至少是 215 x 138 像素）。否则，自动拒绝访问。

在 AIR 应用程序沙箱中运行的内容不需要用户对麦克风的访问权限。因此，从来不调度对麦克风静音和取消静音的状态事件。在应用程序沙箱外的 AIR 中运行的内容要求用户权限，因此可以调度这些状态事件。

将麦克风音频传送到本地扬声器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用参数值 true 调用 Microphone.setLoopback() 方法，以将来自麦克风的音频输入传送到本地系统扬声器。

如果将来自本地麦克风的声音传送到本地扬声器，则会存在创建音频回馈循环的风险，这可能会导致非常大的振鸣声，并且可能会损坏声音硬件。使用参数值 true 调用 Microphone.setUseEchoSuppression() 方法可降低发生音频回馈的风险，但不会完全消除该风险。Adobe 建议您始终在调用 Microphone.setLoopback(true) 之前调用 Microphone.setUseEchoSuppression(true)，除非您确信用户使用耳机来播放声音，或者使用除扬声器以外的某种设备。

以下代码说明了如何将来自本地麦克风的音频传送到本地系统扬声器：

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

更改麦克风音频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可以使用两种方法更改来自麦克风的音频数据。第一，它可以更改输入声音的增益，这会有效地将输入值乘以指定的数值以创建更大或更小的声音。Microphone.gain 属性接受介于 0 和 100 之间的数值（含 0 和 100）。值 50 相当于乘数 1，它指定正常音量。值 0 相当于乘数 0，它可有效地将输入音频静音。大于 50 的值指定的音量高于正常音量。

应用程序也可以更改输入音频的采样率。较高的采样率可提高声音品质，但它们也会创建更密集的数据流 （使用更多的资源进行传输和存储）。Microphone.rate 属性表示以千赫 (kHz) 为单位测量的音频采样率。默认采样率是 8 kHz。如果麦克风支持较高的采样率，您可以将 Microphone.rate 属性设置为高于 8 kHz 的值。例如，如果将 Microphone.rate 属性设置为值 11，则会将采样率设置为 11 kHz；如果将该属性设置为 22，则会将采样率设置为 22 kHz，依此类推。采样率取决于所选择的编解码器。如果使用的是 Nellymoser 编解码器，则可以指定 5、 8、 11、 16、 22 和 44 kHz 作为采样率。使用 Speex 编解码器（Flash Player 10 和 Adobe AIR 1.5 中开始提供）时，只能使用 16 kHz。

检测麦克风活动

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

为节省带宽和处理资源， Flash Player 将尝试检测何时麦克风不传输声音。当麦克风的活动级别处于静音级别阈值以下一段时间后，Flash Player 将停止传输音频输入，并改为调度一个简单的 ActivityEvent。如果使用 Speex 编解码器（Flash Player 10 或更高版本和 Adobe AIR 1.5 或更高版本中提供），请将静音级别设置为 0，以确保应用程序持续传输音频数据。Speex 语音活动检测将自动减少带宽。

注：当应用程序监视麦克风时， Microphone 对象只调度 Activity 事件。因此，如果不调用 setLoopBack(true)、为示例数据事件添加侦听器、将麦克风附加到 NetStream 对象，则不调度任何活动事件。

Microphone 类的以下三个属性用于监视和控制活动检测：

- activityLevel 只读属性指示麦克风检测的音量，范围从 0 到 100。
- silenceLevel 属性指定激活麦克风并调度 ActivityEvent.ACTIVITY 事件所需的音量。silenceLevel 属性也使用从 0 到 100 的范围，默认值为 10。
- silenceTimeout 属性描述活动级别处于静音级别以下多长时间 （以毫秒为单位）后，才会调度 ActivityEvent.ACTIVITY 事件以指示麦克风现在处于静音状态。 silenceTimeout 默认值是 2000。

Microphone.silenceLevel 属性和 Microphone.silenceTimeout 属性都是只读的，但可以使用 Microphone.setSilenceLevel() 方法来更改它们的值。

在某些情况下，在检测到新活动时激活麦克风的过程可能会导致短暂的延迟。通过将麦克风始终保持活动状态，可以消除此类激活延迟。应用程序可以调用 Microphone.setSilenceLevel() 方法并将 silenceLevel 参数设置为零，以通知 Flash Player 将麦克风保持活动状态并持续收集音频数据，即使未检测到任何声音也是如此。反之，如果将 silenceLevel 参数设置为 100，则可以完全禁止激活麦克风。

以下示例显示了有关麦克风的信息，并报告 Microphone 对象调度的 activity 事件和 status 事件：

```
import flash.events.ActivityEvent;
import flash.events.StatusEvent;
import flash.media.Microphone;

var deviceArray:Array = Microphone.names;
trace("Available sound input devices:");
for (var i:int = 0; i < deviceArray.length; i++)
{
    trace(" " + deviceArray[i]);
}

var mic:Microphone = Microphone.getMicrophone();
mic.gain = 60;
mic.rate = 11;
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
mic.setSilenceLevel(5, 1000);

mic.addEventListener(ActivityEvent.ACTIVITY, this.onMicActivity);
mic.addEventListener(StatusEvent.STATUS, this.onMicStatus);

var micDetails:String = "Sound input device name: " + mic.name + '\n';
micDetails += "Gain: " + mic.gain + '\n';
micDetails += "Rate: " + mic.rate + " kHz" + '\n';
micDetails += "Muted: " + mic.muted + '\n';
micDetails += "Silence level: " + mic.silenceLevel + '\n';
micDetails += "Silence timeout: " + mic.silenceTimeout + '\n';
micDetails += "Echo suppression: " + mic.useEchoSuppression + '\n';
trace(micDetails);

function onMicActivity(event:ActivityEvent):void
{
    trace("activating=" + event.activating + ", activityLevel=" +
        mic.activityLevel);
}

function onMicStatus(event:StatusEvent):void
{
    trace("status: level=" + event.level + ", code=" + event.code);
}
```

在运行上面的示例时，对着系统麦克风说话或发出噪音，并观察所生成的、显示在控制台或调试窗口中的 trace 语句。

向媒体服务器发送音频以及从中接收音频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

将 ActionScript 与 Flash Media Server 等流媒体服务器配合使用时，可以使用额外的音频功能。

具体而言，应用程序可以将 Microphone 对象附加到 NetStream 对象，并将数据直接从用户的麦克风传输到服务器。音频数据也可以从服务器流式传输到应用程序，并作为 MovieClip 的一部分或通过使用 Video 对象进行播放。

从 Flash Player 10 和 Adobe AIR 1.5 开始提供 Speex 编解码器。若要设置对于发送到媒体服务器的压缩音频所使用的编解码器，请设置 Microphone 对象的 codec 属性。此属性可以包含两个值，可使用 SoundCodec 类对这两个值进行枚举。将 codec 属性设置为 SoundCodec.SPEEX 将选择使用 Speex 编解码器来压缩音频。将该属性设置为 SoundCodec.NELLYMOSER（默认值）将选择使用 Nellymoser 编解码器来压缩音频。

有关详细信息，请参阅 www.adobe.com/go/learn_fms_docs_cn 上提供的在线 Flash Media Server 文档。

捕获麦克风声音数据

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

在 Flash Player 10.1 和 AIR 2 或更高版本中，可以以浮点值的字节数组形式从麦克风数据捕获数据。每个值表示一个单声道音频数据样本。

要获得麦克风数据，可为 Microphone 对象的 sampleData 事件设置事件侦听器。 Microphone 对象会在用声音样本填充麦克风缓冲时定期调度 sampleData 事件。SampleDataEvent 对象包括一个 data 属性，它是声音样本的一个字节数组。样本均表示为浮点值，每个浮点值表示一个单声道声音样本。

以下代码将麦克风声音数据捕获到名为 soundBytes 的 ByteArray 对象：

```
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
function micSampleDataHandler(event:SampleDataEvent):void {
    while(event.data.bytesAvailable) {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
```

您可以在播放音频时为 Sound 对象重复使用该示例字节。如果要重复使用样本字节，您应将 Microphone 对象的 rate 属性设置为 44，它表示 Sound 对象所使用的采样率。（您还可以将以低采样率捕获的麦克风样本转化为 Sound 对象所需的 44 kHz 采样率。）另外，请记住， Microphone 对象捕获单声道样本，而 Sound 对象使用立体声；因此，应将 Microphone 对象捕获的每个字节写入 Sound 对象两次。以下示例捕获 4 秒钟长的麦克风数据并使用 Sound 对象播放它：

```
const DELAY_LENGTH:int = 4000;
var mic:Microphone = Microphone.getMicrophone();
mic.setSilenceLevel(0, DELAY_LENGTH);
mic.gain = 100;
mic.rate = 44;
mic.addEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);

var timer:Timer = new Timer(DELAY_LENGTH);
timer.addEventListener(TimerEvent.TIMER, timerHandler);
timer.start();

function micSampleDataHandler(event:SampleDataEvent):void
{
    while(event.data.bytesAvailable)
    {
        var sample:Number = event.data.readFloat();
        soundBytes.writeFloat(sample);
    }
}
var sound:Sound = new Sound();
var channel:SoundChannel;
function timerHandler(event:TimerEvent):void
{
    mic.removeEventListener(SampleDataEvent.SAMPLE_DATA, micSampleDataHandler);
    timer.stop();
    soundBytes.position = 0;
```

```
    sound.addEventListener(SampleDataEvent.SAMPLE_DATA, playbackSampleHandler);
    channel.addEventListener( Event.SOUND_COMPLETE, playbackComplete );
    channel = sound.play();
}

function playbackSampleHandler(event:SampleDataEvent):void
{
    for (var i:int = 0; i < 8192 && soundBytes.bytesAvailable > 0; i++)
    {
        trace(sample);
        var sample:Number = soundBytes.readFloat();
        event.data.writeFloat(sample);
        event.data.writeFloat(sample);
    }
}

function playbackComplete( event:Event ):void
{
    trace( "Playback finished.");
}
```

有关从声音样本数据播放声音的详细信息，请参阅第 379 页的 “ 处理动态生成的音频 ”。

声音示例：Podcast Player

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

播客是通过 Internet 以按需方式或订阅方式分发的声音文件。播客通常是作为系列的一部分发布的，此系列也称为播客频道。由于播客节目的持续时间从一分钟到数小时不等，因此，通常在播放的同时对其进行流式传输。播客节目 （也称为项目）通常是以 mp3 文件格式提供的。视频播客也非常受欢迎，但此范例应用程序仅播放使用 mp3 文件的音频播客。

此示例并不是一个功能完备的播客聚合器应用程序。例如，它不能管理对特定播客的订阅，或在下次运行应用程序时记住用户已收听的播客。它可用作功能更完备的播客聚合器的起点。

Podcast Player 示例说明了以下 ActionScript 编程方法：

- 读取外部 RSS 新闻频道并分析其 XML 内容
- 创建 SoundFacade 类以简化加载和播放声音文件的过程
- 显示声音播放进度
- 暂停和恢复声音播放

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。 Podcast Player 应用程序文件位于文件夹 Samples/PodcastPlayer 中。该应用程序包含以下文件：

文件	说明
PodcastPlayer.mxml 或 PodcastPlayer.fla	适用于 Flex (MXML) 或 Flash (FLA) 的应用程序的用户界面。
comp/example/programmingas3/podcastplayer/PodcastPlayer.as	包含 podcast 播放器用户界面逻辑的文档类 （仅限 Flash）。
SoundPlayer.mxml	一个显示播放按钮和进度栏并控制声音播放的 MXML 组件 （仅用于 Flex）。

文件	说明
main.css	应用程序用户界面的样式 （仅限 Flex）。
images/	用于为按钮增加样式的图标 （仅限 Flex）。
comp/example/programmingas3/podcastplayer/SoundPlayer.as	包含声音播放器用户界面逻辑的 SoundPlayer 影片剪辑元件的类 （仅限 Flash）。
comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as	用于在数据网格单元格中显示播放按钮的自定义单元格渲染器 （仅限 Flash）。
com/example/programmingas3/podcastplayer/RSSBase.as	为 RSSChannel 类和 RSSItem 类提供公共属性和方法的基类。
com/example/programmingas3/podcastplayer/RSSChannel.as	保存 RSS 频道的相关数据的 ActionScript 类。
com/example/programmingas3/podcastplayer/RSSItem.as	保存 RSS 项目的相关数据的 ActionScript 类。
com/example/programmingas3/podcastplayer/SoundFacade.as	应用程序的主 ActionScript 类。它封装 Sound 类和 SoundChannel 类的方法和事件，并添加对播放暂停和恢复的支持。
com/example/programmingas3/podcastplayer/URLService.as	从远程 URL 检索数据的 ActionScript 类。
playerconfig.xml	这是一个 XML 文件，其中包含表示播客频道的 RSS 新闻频道列表。
comp/example/programmingas3/utils/DateUtil.as	用于简化日期格式设置的类 （仅限 Flash）。

读取播客频道的 RSS 数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Podcast Player 应用程序先读取一些播客频道及其节目的相关信息：

1. 首先，应用程序读取包含播客频道列表的 XML 配置文件，并向用户显示该频道列表。

2. 当用户选择其中的一个播客频道时，应用程序会读取该频道的 RSS 新闻频道并显示频道节目列表。

此示例使用 URLLoader 实用程序类，从远程位置或本地文件检索基于文本的数据。Podcast Player 先创建一个 URLLoader 对象，以便从 playerconfig.xml 文件中获取采用 XML 格式的 RSS 新闻频道列表。接下来，当用户从列表中选择特定新闻频道时，将创建一个新的 URLLoader 对象以从该新闻频道的 URL 中读取 RSS 数据。

使用 SoundFacade 类简化声音加载和播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 声音体系结构功能强大，但非常复杂。如果应用程序只需要基本的声音加载和播放功能，可使用通过提供一组更简单的方法调用和事件来隐藏某些复杂性的类。在软件设计模式领域中，这样的类称为 “ 外观 ”。

SoundFacade 类表示用于执行以下任务的单个接口：

- 使用 Sound 对象、 SoundLoaderContext 对象以及 SoundMixer 类来加载声音文件
- 使用 Sound 对象和 SoundChannel 对象来播放声音文件
- 调度播放进度事件
- 使用 Sound 对象和 SoundChannel 对象来暂停和恢复声音播放

SoundFacade 类尝试以更简化的方式提供 ActionScript Sound 类的大部分功能。

以下代码显示了类声明、类属性以及 SoundFacade() 构造函数方法：

```
public class SoundFacade extends EventDispatcher
{
    public var s:Sound;
    public var sc:SoundChannel;
    public var url:String;
    public var bufferTime:int = 1000;

    public var isLoaded:Boolean = false;
    public var isReadyToPlay:Boolean = false;
    public var isPlaying:Boolean = false;
    public var isStreaming:Boolean = true;
    public var autoLoad:Boolean = true;
    public var autoPlay:Boolean = true;

    public var pausePosition:int = 0;

    public static const PLAY_PROGRESS:String = "playProgress";
    public var progressInterval:int = 1000;
    public var playTimer:Timer;

    public function SoundFacade(soundUrl:String, autoLoad:Boolean = true,
                                    autoPlay:Boolean = true, streaming:Boolean = true,
                                    bufferTime:int = -1):void
    {
        this.url = soundUrl;
```

```
        // Sets Boolean values that determine the behavior of this object
        this.autoLoad = autoLoad;
        this.autoPlay = autoPlay;
        this.isStreaming = streaming;

        // Defaults to the global bufferTime value
        if (bufferTime < 0)
        {
            bufferTime = SoundMixer.bufferTime;
        }

        // Keeps buffer time reasonable, between 0 and 30 seconds
        this.bufferTime = Math.min(Math.max(0, bufferTime), 30000);

        if (autoLoad)
        {
            load();
        }
    }
```

SoundFacade 类扩展了 EventDispatcher 类，以使其能够调度自己的事件。类代码先声明 Sound 对象和 SoundChannel 对象的属性。该类还会存储声音文件 URL 的值以及对声音进行流式传输时使用的 bufferTime 属性。此外，它还接受某些影响加载和播放行为的布尔参数值：

- autoLoad 参数通知对象，应在创建此对象后立即启动声音加载。
- autoPlay 参数指示在加载了足够多的声音数据后应立即启动声音播放。如果这是声音流，在加载了足够多的数据 （由 bufferTime 属性指定）后将立即开始播放。
- streaming 参数指示可以在加载完成之前开始播放此声音文件。

bufferTime 参数的默认值为 -1。如果构造函数方法在 bufferTime 参数中检测到负值，它会将 bufferTime 属性设置为 SoundMixer.bufferTime 的值。这样，应用程序便可根据需要默认使用全局 SoundMixer.bufferTime 值。

如果将 autoLoad 参数设置为 true，构造函数方法将立即调用以下 load() 方法来开始加载声音文件：

```
public function load():void
{
    if (this.isPlaying)
    {
        this.stop();
        this.s.close();
    }
    this.isLoaded = false;

    this.s = new Sound();

    this.s.addEventListener(ProgressEvent.PROGRESS, onLoadProgress);
    this.s.addEventListener(Event.OPEN, onLoadOpen);
    this.s.addEventListener(Event.COMPLETE, onLoadComplete);
    this.s.addEventListener(Event.ID3, onID3);
    this.s.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
    this.s.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onIOError);

    var req:URLRequest = new URLRequest(this.url);

    var context:SoundLoaderContext = new SoundLoaderContext(this.bufferTime, true);
    this.s.load(req, context);
}
```

load() 方法创建一个新的 Sound 对象，然后为所有重要的声音事件添加侦听器。接下来，它通知 Sound 对象使用 SoundLoaderContext 对象传入 bufferTime 值以加载声音文件。

由于可以更改 url 属性，因此，可以使用 SoundFacade 实例来连续播放不同的声音文件：只需更改 url 属性并调用 load() 方法，即可加载新的声音文件。

以下三个事件侦听器方法说明了 SoundFacade 对象如何跟踪加载进度并确定何时开始播放声音：

```
public function onLoadOpen(event:Event):void
{
    if (this.isStreaming)
    {
        this.isReadyToPlay = true;
        if (autoPlay)
        {
            this.play();
        }
    }
    this.dispatchEvent(event.clone());
}

public function onLoadProgress(event:ProgressEvent):void
{
    this.dispatchEvent(event.clone());
}

public function onLoadComplete(event:Event):void
{
    this.isReadyToPlay = true;
    this.isLoaded = true;
    this.dispatchEvent(evt.clone());

    if (autoPlay && !isPlaying)
    {
        play();
    }
}
```

在开始加载声音时，将执行 onLoadOpen() 方法。如果可以使用流模式播放声音， onLoadComplete() 方法会立即将 isReadyToPlay 标志设置为 true。 isReadyToPlay 标志确定应用程序能否开始播放声音，这可能为了响应用户动作，如单击 “ 播放 ” 按钮。 SoundChannel 类管理声音数据的缓冲，因此在调用 play() 方法之前，不需要明确地检查是否加载了足够多的数据。

在加载过程中，将定期执行 onLoadProgress() 方法。它仅调度其 ProgressEvent 对象的克隆，该对象由使用此 SoundFacade 对象的代码使用。

当完全加载了声音数据后，将执行 onLoadComplete() 方法，以便为非声音流调用 play() 方法 （如果需要）。下面显示了 play() 方法本身。

```
public function play(pos:int = 0):void
{
    if (!this.isPlaying)
    {
        if (this.isReadyToPlay)
        {
            this.sc = this.s.play(pos);
            this.sc.addEventListener(Event.SOUND_COMPLETE, onPlayComplete);
            this.isPlaying = true;

            this.playTimer = new Timer(this.progressInterval);
            this.playTimer.addEventListener(TimerEvent.TIMER, onPlayTimer);
            this.playTimer.start();
        }
    }
}
```

如果已准备好播放声音，play() 方法将调用 Sound.play() 方法。生成的 SoundChannel 对象存储在 sc 属性中。play() 方法随后创建一个 Timer 对象，该对象用于按固定间隔调度播放进度事件。

显示播放进度

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

创建 Timer 对象以实现播放监视是一个很复杂的操作，您只需对其编码一次。通过在可重用的类 （如 SoundFacade 类）中封装此 Timer 逻辑，应用程序可以在加载和播放声音时侦听相同类型的进度事件。

由 SoundFacade.play() 方法创建的 Timer 对象每秒调度一个 TimerEvent 实例。每当新的 TimerEvent 到达时，就会执行以下 onPlayTimer() 方法：

```
public function onPlayTimer(event:TimerEvent):void
{
    var estimatedLength:int =
        Math.ceil(this.s.length / (this.s.bytesLoaded / this.s.bytesTotal));
    var progEvent:ProgressEvent =
        new ProgressEvent(PLAY_PROGRESS, false, false, this.sc.position, estimatedLength);
    this.dispatchEvent(progEvent);
}
```

onPlayTimer() 方法可实现在第 382 页的 “ 监视播放 ”一节中介绍的大小估计技术。然后，它创建一个事件类型为 SoundFacade.PLAY_PROGRESS 的新 ProgressEvent 实例，并将 bytesLoaded 属性设置 SoundChannel 对象的当前位置，而将 bytesTotal 属性设置为估计的声音数据长度。

暂停和恢复播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以前显示的 SoundFacade.play() 方法接受 pos 参数，该参数与声音数据中的起始位置相对应。如果 pos 值为零，则从开头开始播放声音。

SoundFacade.stop() 方法也接受 pos 参数，如下所示：

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

每当调用 SoundFacade.stop() 方法时，它都会设置 pausePosition 属性，以便当用户要恢复播放相同的声音时应用程序知道将播放头放置在什么位置。

下面显示的 SoundFacade.pause() 和 SoundFacade.resume() 方法分别调用 SoundFacade.stop() 和 SoundFacade.play() 方法，以便每次传递 pos 参数值。

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

pause() 方法将当前的 SoundChannel.position 值传递给 play() 方法，后者将该值存储在 pausePosition 属性中。resume() 方法通过使用 pausePosition 值作为起点再次开始播放相同的声音。

扩展 Podcast Player 示例

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此示例呈示了一个框架 Podcast Player，用于说明如何使用可重用的 SoundFacade 类。您可以添加其他功能以改进此应用程序的用途，其中包括以下功能：

- 将新闻频道列表和有关每个节目的使用信息存储在 SharedObject 实例中，下次用户运行应用程序时便可以使用它们。
- 允许用户将其自己的 RSS 新闻频道添加到播客频道列表中。
- 当用户停止或离开节目时，记住播放头的位置，以便下次用户运行应用程序时能够从该位置重新开始播放。
- 下载节目的 mp3 文件，以便用户在没有连接到 Internet 时可以脱机收听。
- 添加订阅功能，以便定期检查播客频道中的新节目并自动更新节目列表。
- 使用来自播客托管服务 （如 Odeo.com）的 API 添加播客搜索和浏览功能。

第 25 章：使用视频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash 视频是 Internet 上的一项优秀技术。然而，视频的传统演示形式 （在一个矩形屏幕中演示，下方有一个进度栏和一些控制按钮）只是视频的可能使用方式之一。通过 ActionScript，您可以微调和控制视频的加载、演示和播放。

视频基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe® Flash® Player 和 Adobe® AIR™ 的一个重要功能是可以使用 ActionScript，像操作其他可视内容 （如图像、动画、文本等）一样显示和操作视频信息。在 Adobe Flash CS4 Professional 中创建 Flash 视频 (FLV) 文件时，您可以选择包含常用播放控件的外观。不过，您不一定要局限于可用的选项。使用 ActionScript 可以精确控制视频的加载、显示和播放，这意味着您可以创建自己的视频播放器外观，也可以按照所需的任何非传统方式使用视频。在 ActionScript 中使用视频涉及多个类的联合使用：

- Video 类：舞台上的传统视频内容框是 Video 类的一个实例。 Video 类是一种显示对象，因此可以使用适用于其他显示对象的同样的技术 （比如定位、应用变形、应用滤镜和混合模式等）进行操作。
- StageVideo 类：Video 类通常使用软件解码和呈现。当设备上的 GPU 硬件加速可用时，您的应用程序可以切换到 StageVideo 类以利用硬件加速呈现。 StageVideo API 包括一组事件，这些事件可告诉您的代码何时在 StageVideo 和 Video 对象之间进行切换。舞台视频在视频播放方面存在一些小的限制。如果您的应用程序接受这些限制，请实现 StageVideo API。请参阅第 435 页的 “ 指导准则和限制 ”。
- NetStream 类：当加载将由 ActionScript 控制的视频文件时， NetStream 实例表示视频内容的源 （在此示例中是视频数据流）。使用 NetStream 实例也涉及 NetConnection 对象的使用，该对象是到视频文件的连接，它好比是视频数据馈送的通道。
- Camera 类：当使用的视频数据来自与用户计算机相连接的摄像头时，Camera 实例表示视频内容的源，即用户的摄像头以及它所提供的视频数据。 Flash Player 11.4 和 AIR 3.4 中新增的类，可以将摄像头输入用于 StageVideo。

在加载外部视频时，您可以从标准 Web 服务器加载文件以便进行渐进式下载，也可以使用由专门的服务器 （如 Adobe 的 Flash® Media Server）传送的视频流。

重要概念和术语

提示点 一个可以放在视频文件内特定时刻的标记，用来提供特定功能，例如，可用作书签以便定位到该时刻或提供与该时刻相关联的其他数据。

编码 接收某一种格式的视频数据，然后将其转换为另一种视频数据格式的过程；例如，接收高分辨率源视频，然后将其转换成适合 Internet 传送的格式。

帧 单个视频信息段；每个帧就像表示某一时刻的快照的静止图像一样。通过按顺序高速播放各个帧，可产生动画视觉效果。

关键帧 包含帧的完整信息的视频帧。关键帧后面的其他帧仅包含有关它们与关键帧之间的差异的信息，而不包含完整的帧信息。

元数据 有关嵌入在视频文件中并可在加载视频时检索的视频文件的信息。

渐进式下载 从标准 Web 服务器传送视频文件时，会使用渐进式下载来加载视频数据，这意味着将按顺序加载视频信息。其好处是不必等待整个文件下载完毕即可开始播放视频；不过，它会阻止您向前跳到视频中尚未加载的部分。

流 渐进式下载的一种替代方法，使用流式传输 （有时称为“实流”）技术和专用视频服务器通过 Internet 传送视频。使用流式传输，用于查看视频的计算机不必一次下载整个视频。为了加快下载速度，在任何时刻，计算机均只需要整个视频信息的一部分。由于使用一台专用服务器来控制视频内容的传送，因此可以在任何时刻访问视频的任何部分，而无需等待其下载完毕后才能进行访问。

了解视频格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

除了 Adobe FLV 视频格式之外，Flash Player 和 Adobe AIR 还支持从 MPEG-4 标准文件格式中以 H.264 和 HE-AAC 编码的视频和音频。这些格式以更低的比特率提供高质量的视频流。开发人员可利用业界标准工具（包括 Adobe Premiere Pro 和 Adobe After Effects）来创建并提供引入注目的视频内容。

类型	格式	容器
视频	H.264	MPEG-4：MP4、 M4V、 F4V、 3GPP
视频	Sorenson Spark	FLV 文件
视频	ON2 VP6	FLV 文件
音频	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4：MP4、 M4V、 F4V、 3GPP
音频	Mp3	Mp3
音频	Nellymoser	FLV 文件
音频	Speex	FLV 文件

更多帮助主题

Flash Media Server: Supported codecs

Adobe HTTP Dynamic Streaming

针对移动设备对视频进行编码

Android 上的 AIR 可以对各种 H.264 视频进行解码。但是，只有一小部分 H.264 视频适合于在移动手机上流畅播放。这是因为许多移动手机受处理能力的限制。用于移动设备的 Adobe Flash Player 可以使用内置硬件加速对 H.264 视频进行解码。这种解码方式可以确保以较低的处理能力消耗实现较高的播放质量。

H.264 标准支持多种编码技术。只有高端设备才能够流畅播放具有复杂配置文件和级别的视频。但是，大多数设备均可播放采用基本配置文件编码的视频。在移动设备上，为上述部分技术提供了硬件加速功能。配置文件和级别参数定义了支持硬件加速的部分编码技术以及编码器所使用的设置。对于开发人员，可理解为采用能够在大多数设备上流畅播放的所选分辨率对视频进行编码。

尽管能够利用硬件加速的分辨率因设备而异，但大多数设备均支持下列标准分辨率。

高宽比	推荐分辨率		
4:3	640 × 480	512 × 384	480 × 360
16:9	640 × 360	512 x 288	480 × 272

注：Flash Player 支持 H.264 标准的所有级别和配置文件。遵循上述建议可确保在大多数设备上实现硬件加速和更佳的用户体验。这些建议并非强制性的。

有关 Adobe Media Encoder CS5 的详细介绍和编码设置，请参阅针对为移动设备上的 Flash Player 10.1 编码 H.264 视频提出的建议。

注：在 iOS 上，只有使用 Sorenson Spark 和 On2 VP6 编解码器编码的视频可以使用 Video 类播放。可以通过使用 flash.net.navigateToURL() 函数启动视频 URL ，从而在设备视频播放器中播放 H.264 编码的视频。还可以在 StageWebView 对象中显示的 html 页中使用 <video> 标签播放 H.264 视频。

Flash Player 和 AIR 与编码的视频文件的兼容性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

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。

Flash Player 9.0.115.0 及更高版本支持从标准 MPEG-4 容器格式派生的文件。这些文件包括 F4V、 MP4、 M4A、 MOV、 MP4V、3GP 和 3G2（如果这些文件包含 H.264 视频和 / 或 HEAAC v2 编码音频）。与 Sorenson 或 On2 中相同的编码配置文件相比，H.264 可以在更低的比特率下传送更高品质的视频。HE-AAC v2 是 AAC 的扩展，AAC 是在 MPEG-4 视频标准中定义的一种标准音频格式。 HE-AAC v2 使用频带复制 (SBR) 和参量立体声 (PS) 技术来提高低比特率下的编码效率。

下表列出了支持的编解码器。表中还显示了相应的 SWF 文件格式以及播放这些文件所需的 Flash Player 和 AIR 版本：

编解码器	SWF 文件格式版本 （支持的最早发布版本）	Flash Player 和 AIR （播放所需要的最早的版本）
Sorenson Spark	6	Flash Player 6、 Flash Lite 3
On2 VP6	6	Flash Player 8、 Flash Lite 3。 只有 Flash Player 8 及更高版本才支持 On2 VP6 视频的发布和播放。
H.264 (MPEG-4 Part 10)	9	Flash Player 9 Update 3、 AIR 1.0
ADPCM	6	Flash Player 6、 Flash Lite 3
Mp3	6	Flash Player 6、 Flash Lite 3
AAC (MPEG-4 Part 3)	9	Flash Player 9 Update 3、 AIR 1.0
Speex （音频）	10	Flash Player 10、 AIR 1.5
Nellymoser	6	Flash Player 6

了解 Adobe F4V 和 FLV 视频文件格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe 提供了 F4V 和 FLV 视频文件格式，以用于向 Flash Player 和 AIR 流式传输内容。有关这些视频文件格式的完整描述，请参阅 www.adobe.com/go/video_file_format_cn。

F4V 视频文件格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

从 Flash Player Update 3 (9.0.115.0) 和 AIR 1.0 开始，Flash Player 和 AIR 支持 Adobe F4V 视频格式，该格式基于 ISO MP4 格式， MP4 格式的不同子集支持不同的功能。 Flash Player 需要有效的 F4V 文件来从以下某个顶级框开始：

- ftyp

  ftyp 框标识程序为播放特定的文件格式而必须支持的功能。

- moov

  moov 框实际是 F4V 文件的标头。该框包含一个或多个其他框，这些框又包含定义 F4V 数据结构的其他框。 F4V 文件必须包含且只能包含一个 moov 框。

- mdat

  mdat 框包含 F4V 文件的数据负载。一个 FV 文件只包含一个 mdat 框。 moov 框也必须存在于文件中，因为 mdat 框在单独使用时没有意义。

F4V 文件支持 big-endian 字节顺序的多字节整数，按照该顺序，最高有效字节在最低地址中最先出现。

FLV 视频文件格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe FLV 文件格式包含可供 Flash Player 传送的音频和视频编码数据。可以使用编码器 （如 Adobe Media Encoder 或 Sorenson™ Squeeze）将 QuickTime 或 Windows Media 视频文件转换为 FLV 文件。

注：可通过将视频导入 Flash，然后再导出为 FLV 文件来创建 FLV 文件。可以使用 “FLV 导出 ” 插件从受支持的视频编辑应用程序中导出 FLV 文件。若要从 Web 服务器加载 FLV 文件，请向您的 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。有关详细信息，请参阅第 428 页的 “ 关于配置 FLV 文件以便在服务器上托管 ”。

有关 FLV 文件的详细信息，请参阅第 428 页的 “ 视频文件的高级主题 ”。

外部视频和嵌入视频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用外部视频文件可以提供使用导入的视频时不可用的某些功能：

- 可在应用程序中使用较长的视频剪辑，而不会降低播放速度。外部视频文件可使用缓存内存，这意味着大文件将分成小片断存储，并可以动态访问。因此，外部 F4V 和 FLV 文件所需要的内存比嵌入的视频文件要少。
- 外部视频文件的帧速率可以不同于它所播放的 SWF 文件。例如，可以将 SWF 文件帧速率设置为 30 帧 / 秒 (fps)，而将视频帧速率设置为 21 fps。与嵌入的视频相比，此项设置可使您更好地控制视频，确保视频顺畅地播放。此项设置还允许您以不同的帧速率播放视频文件，而无需更改现有 SWF 文件的内容。
- 如果使用外部视频文件，则不会在加载视频文件时中断 SWF 内容的播放。导入的视频文件有时可能需要中断文档播放来执行某些功能，例如，访问 CD-ROM 驱动器。视频文件可独立于 SWF 内容执行功能，而不会中断播放。
- 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元数据。

了解 Video 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Video 类可以直接在应用程序中显示实时视频流，而无需将其嵌入 SWF 文件中。可以使用 Camera.getCamera() 方法捕获并播放实时视频。还可以使用 Video 类通过 HTTP 或在本地文件系统中播放视频文件。在项目中使用 Video 有多种不同方法：

- 使用 NetConnection 和 NetStream 类动态加载视频文件并在 Video 对象中显示视频。
- 从用户摄像头捕获输入。有关详细信息，请参阅第 441 页的 " 使用摄像头 "。
- 使用 FLVPlayback 组件。
- 使用 VideoDisplay 控件。

注：舞台上 Video 对象的实例是 Video 类的实例。

尽管 Video 类位于 flash.media 包中，但它继承自 flash.display.DisplayObject 类。因此，所有显示对象功能 （如矩阵转换和滤镜）也适用于 Video 实例。

有关详细信息，请参阅第 144 页的 " 处理显示对象 "、第 175 页的 " 使用几何结构 " 和第 224 页的 " 过滤显示对象 "。

加载视频文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 NetStream 和 NetConnection 类加载视频是一个多步骤过程。对于将 Video 对象添加到显示列表、将 NetStream 对象附加到 Video 实例以及调用 NetStream 对象的 play() 方法，最佳的做法是应按指定顺序执行这些步骤：

1 创建一个 NetConnection 对象。如果要连接到本地视频文件或者未使用 Adobe Flash Media Server 2 之类的服务器的视频文件，请将 null 传给 connect() 方法，以从 HTTP 地址或本地驱动器上播放视频文件。如果要连接到服务器，请将该参数设置为包含服务器上视频文件的应用程序的 URI。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2 创建一个用来显示视频的新 Video 对象，将其添加到舞台显示列表，如以下代码片段所示：

```
var vid:Video = new Video();
addChild(vid);
```

3 创建一个 NetStream 对象，将 NetConnection 对象作为一个参数传递给构造函数。以下代码片段将 NetStream 对象连接到 NetConnection 实例并设置该流的事件处理函数：

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS,netStatusHandler);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);

function netStatusHandler(event:NetStatusEvent):void
{
    // handle netStatus events, described later
}

function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

4 使用 Video 对象的 attachNetStream() 方法将 NetStream 对象附加到 Video 对象，如以下代码片段所示：

```
vid.attachNetStream(ns);
```

5 调用 NetStream 对象的 play() 方法，同时将视频文件 url 作为开始视频播放的参数。以下代码片段将加载并播放与 SWF 文件位于同一目录下的视频文件 “video.mp4”：

```
ns.play("video.mp4");
```

更多帮助主题

Flex：Spark VideoPlayer 控件

spark.components.VideoDisplay

控制视频播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

NetStream 类提供了四个用于控制视频播放的主要方法：

pause()：暂停播放视频流。如果视频已经暂停，则调用此方法将不会执行任何操作。

resume()：恢复播放已暂停的视频流。如果视频已在播放，则调用此方法将不会执行任何操作。

seek()：搜索与指定位置 （从流的开始处算起的偏移，以秒为单位）最靠近的关键帧。

togglePause()：暂停或恢复播放流。

注：没有 stop() 方法。为了停止视频流，必须暂停播放并找到视频流的开始位置。

注：play() 方法不会恢复播放，它用于加载视频文件。

以下示例演示如何使用多个不同的按钮控制视频。若要运行下面的示例，请创建一个新文档，并在工作区中添加 4 个按钮实例（pauseBtn、 playBtn、 stopBtn 和 togglePauseBtn）：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

pauseBtn.addEventListener(MouseEvent.CLICK, pauseHandler);
playBtn.addEventListener(MouseEvent.CLICK, playHandler);
stopBtn.addEventListener(MouseEvent.CLICK, stopHandler);
togglePauseBtn.addEventListener(MouseEvent.CLICK, togglePauseHandler);

function pauseHandler(event:MouseEvent):void
{
    ns.pause();
}
function playHandler(event:MouseEvent):void
{
    ns.resume();
}
function stopHandler(event:MouseEvent):void
{
    // Pause the stream and move the playhead back to
    // the beginning of the stream.
    ns.pause();
    ns.seek(0);
}
function togglePauseHandler(event:MouseEvent):void
{
    ns.togglePause();
}
```

播放视频的同时单击 pauseBtn 按钮实例会导致视频文件暂停。如果视频已经暂停，则单击此按钮不会执行任何操作。如果之前暂停了播放，则单击 playBtn 按钮实例会恢复视频播放；如果视频已在播放，则单击该按钮不会执行任何操作。

检测视频流的末尾

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

为了侦听视频流的开始和末尾，需要向 NetStream 实例添加一个事件侦听器以侦听 netStatus 事件。以下代码演示如何在视频播放过程中侦听不同代码：

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

上面这段代码的输出如下：

```
NetStream.Play.Start
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Empty
NetStream.Buffer.Full
NetStream.Buffer.Flush
NetStream.Play.Stop
NetStream.Buffer.Empty
NetStream.Buffer.Flush
```

您要专门侦听的两段代码为 “NetStream.Play.Start” 和 “NetStream.Play.Stop”，它们会在视频播放的开始和末尾发出信号。下面的代码片断使用 switch 语句来过滤这两段代码并输出一条消息：

```
function statusHandler(event:NetStatusEvent):void
{
    switch (event.info.code)
    {
        case "NetStream.Play.Start":
            trace("Start [" + ns.time.toFixed(3) + " seconds]");
            break;
        case "NetStream.Play.Stop":
            trace("Stop [" + ns.time.toFixed(3) + " seconds]");
            break;
    }
}
```

通过侦听 netStatus 事件 (NetStatusEvent.NET_STATUS)，您可以生成一个视频播放器，它在当前视频完成播放后加载播放列表中的下一个视频。

在全屏模式下播放视频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 和 AIR 使您可以创建全屏应用程序来播放您的视频，并支持将视频放大至全屏。

对于桌面上以全屏模式运行的 AIR 内容，在播放过程中将禁用系统屏幕保护程序和节能选项，直至视频输入停止或用户退出全屏模式。

有关使用全屏模式的完整详细信息，请参阅第 139 页的 “ 使用全屏模式 ”。

在浏览器中为 **Flash Player** 启用全屏模式

应先通过应用程序的 “ 发布 ” 模板启用全屏模式，然后才能在浏览器中实现 Flash Player 的全屏模式。允许全屏的模板包含 <object> 和 <embed> 标签，这些标签包含 allowFullScreen 参数。下面的示例显示了 <embed> 标签中的 allowFullScreen 参数。

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="fullScreen" width="100%" height="100%"
    codebase="http://fpdownload.macromedia.com/get/flashplayer/current/swflash.cab">
    ...
    <param name="allowFullScreen" value="true" />
    <embed src="fullScreen.swf" allowFullScreen="true" quality="high" bgcolor="#869ca7"
        width="100%" height="100%" name="fullScreen" align="middle"
        play="true"
        loop="false"
        quality="high"
        allowScriptAccess="sameDomain"
        type="application/x-shockwave-flash"
        pluginspage="http://www.adobe.com/go/getflashplayer">
    </embed>
    ...
</object>
```

在 Flash 中，选择 “ 文件 ”->“ 发布设置 ”，然后在 “ 发布设置 ” 对话框中的 “HTML” 选项卡上，选择 “ 仅 Flash - 允许全屏 ” 模板。

在 Flex 中，确保 HTML 模板包含支持全屏的 <object> 和 <embed> 标签。

启动全屏模式

对于运行于浏览器中的 Flash Player 内容，可在响应鼠标单击或按键操作时为视频启动全屏模式。例如，您可以在用户单击标签为 “ 全屏 ” 的按钮或者从上下文菜单中选择 “ 全屏 ” 命令时，启动全屏模式。若要响应用户，请将一个事件侦听器添加到发生操作的对象。下面的代码将一个事件侦听器添加到用户单击后可进入全屏模式的按钮：

```
var fullScreenButton:Button = new Button();
fullScreenButton.label = "Full Screen";
addChild(fullScreenButton);
fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);

function fullScreenButtonHandler(event:MouseEvent)
{
    stage.displayState = StageDisplayState.FULL_SCREEN;

}
```

代码通过将 Stage.displayState 属性设置为 StageDisplayState.FULL_SCREEN 来启动全屏模式。这段代码将整个舞台放大为全屏，同时其中的视频根据它在舞台中所占空间的比例一同放大。

fullScreenSourceRect 属性可用于指定舞台中要放大为全屏的某块特定区域。首先，定义要放大为全屏的矩形。然后将其赋给 Stage.fullScreenSourceRect 属性。此版本的 fullScreenButtonHandler() 函数添加了另外两行代码，这些代码只是将视频放大为全屏。

```
private function fullScreenButtonHandler(event:MouseEvent)
{
    var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
    stage.fullScreenSourceRect = screenRectangle;
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

虽然此示例中为响应鼠标单击而调用了一个事件处理函数，但进入全屏模式的方法对于 Flash Player 和 AIR 都是相同的。定义要缩放的矩形，然后设置 Stage.displayState 属性。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

完整的示例 （如下所示）添加了额外的代码，这些代码创建连接以及视频的 NetStream 对象，并开始播放该视频。

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import fl.controls.Button;
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.events.FullScreenEvent;
    import flash.geom.Rectangle;

    public class FullScreenVideoExample extends Sprite
    {
        var fullScreenButton:Button = new Button();
        var video:Video = new Video();

        public function FullScreenVideoExample()
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("http://www.helpexamples.com/flash/video/water.flv");

            fullScreenButton.x = 100;
            fullScreenButton.y = 270;
            fullScreenButton.label = "Full Screen";
            addChild(fullScreenButton);
            fullScreenButton.addEventListener(MouseEvent.CLICK, fullScreenButtonHandler);
        }

        private function fullScreenButtonHandler(event:MouseEvent)
        {
            var screenRectangle:Rectangle = new Rectangle(video.x, video.y, video.width, video.height);
            stage.fullScreenSourceRect = screenRectangle;
            stage.displayState = StageDisplayState.FULL_SCREEN;
        }

        public function onMetaData(infoObject:Object):void
        {
            // stub for callback function
        }
    }
}
```

onMetaData() 函数是处理视频元数据 （如果存在）的回调函数。回调函数是运行时响应某种类型的事件而调用的函数。此示例中，onMetaData() 函数是满足要求以提供该函数的存根。有关详细信息，请参阅第 412 页的 “ 编写元数据和提示点的回调方法 ”

退出全屏模式

用户通过输入某个快捷键 （如 Esc 键）即可退出全屏模式。可以用 ActionScript 终止全屏模式，方法是将 Stage.displayState 属性设置为 StageDisplayState.NORMAL。下例中的代码在发生 NetStream.Play.Stop netStatus 事件时终止全屏模式。

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

全屏硬件加速

在将舞台的某个矩形区域放大为全屏模式时，如果硬件加速可用并已启用，则 Flash Player 或 AIR 将使用硬件加速。运行时使用计算机上的视频适配器加速将视频或舞台的某个局部放大为全屏大小。在这些情况下， Flash Player 应用程序经常可以通过从 Video 类 （或 Camera 类 - 对于 Flash Player 11.4/AIR 3.4 和更高版本而言）切换到 StageVideo 类而获益。

有关全屏模式下硬件加速的详细信息，请参阅第 139 页的 " 使用全屏模式 "。有关 StageVideo 的详细信息，请参阅第 433 页的 " 使用 StageVideo 类来实现硬件加速呈现 "。

流式传输视频文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要流式传输 Flash Media Server 中的文件，可以使用 NetConnection 和 NetStream 类连接到远程服务器实例并播放指定的流。若要指定实时消息传递协议 (RTMP) 服务器，请向 NetConnection.connect() 方法传递所需的 RTMP URL （例如 "rtmp://localhost/appName/appInstance"），而不是传递 null。若要播放指定 Flash Media Server 中的特定实时流或录制流，可为 NetStream.play() 方法传递一个由 NetStream.publish() 发布的实时数据的标识名称，或一个要播放的录制文件名称。

向服务器发送视频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果您要生成涉及 Video 或 Camera 对象的更为复杂的应用程序，可以使用 Flash Media Server 提供的流媒体功能和开发环境组合来创建媒体应用程序并将它提供给广泛的目标用户。开发人员可以使用这一组合来创建应用程序，如视频点播、实时 Web 事件广播和 Mp3 流式传输，以及视频博客、视频消息传送和多媒体聊天环境。有关详细信息，请参阅 www.adobe.com/go/learn_fms_docs_cn 上提供的在线 Flash Media Server 文档。

了解提示点

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可在编码期间将提示点嵌入 Adobe F4V 或 FLV 视频文件。过去，在影片中嵌入提示点是为了给放映员提供了一个可视信号，以指出胶片盘中的胶片即将放完。在 Adobe F4V 和 FLV 视频格式中，提示点的作用在于：当视频流中出现提示点时，在应用程序中触发一个或多个其他动作。

您可以对 Flash 视频使用几种不同类型的提示点。可以使用 ActionScript 与在创建视频文件时嵌入其中的提示点进行交互。

- 导航提示点：您可以在编码视频文件时，将导航提示点嵌入到视频流和元数据包中。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：您可以在编码视频文件时，将事件提示点嵌入到视频流和元数据包中。还可以编写代码来处理视频播放期间在指定点上触发的事件。

- ActionScript 提示点：ActionScript 提示点只对 Flash FLVPlayback 组件有用。 ActionScript 提示点是您使用 ActionScript 代码创建和访问的外部提示点。您可以编写代码来触发这些与视频播放有关的提示点。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。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 FLV 文件中，因为这些提示点更加精确。

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。您可以在视频文件中设置一些希望用户搜索的特定点。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。

有关使用提示点对 Adobe 视频文件进行编码的详细信息，请参阅 《使用 Flash》中的 " 嵌入提示点 "。

您可以通过编写 ActionScript 来访问提示点参数。提示点参数是由回调处理函数接收的事件对象的一部分。

若要在 FLV 文件到达特定提示点时在代码中触发特定动作，请使用 NetStream.onCuePoint 事件处理函数。

若要同步对 F4V 视频文件中某个提示点的操作，必须从 onMetaData() 或 onXMPData() 回调函数检索该提示点数据，并使用 ActionScript 3.0 中的 Timer 类触发该提示点。有关 F4V 提示点的详细信息，请参阅第 422 页的 " 使用 onXMPData()"。

有关处理提示点和元数据的详细信息，请参阅第 412 页的 " 编写元数据和提示点的回调方法 "。

编写元数据和提示点的回调方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当播放器收到特定元数据或到达特定提示点时，可以在应用程序中触发动作。当这些事件发生时，必须将特定回调方法用作事件处理函数。 NetStream 类指定了在播放期间可发生的以下元数据事件：onCuePoint （仅限 FLV 文件）、 onImageData、onMetaData、 onPlayStatus、 onTextData 和 onXMPData。

必须为这些处理程序编写回调方法，否则 Flash 运行时可能会引发错误。例如，以下代码播放 SWF 文件所在文件夹中名为 video.flv 的 FLV 文件：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.flv");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    trace(event.text);
}

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

上面的代码加载一个名为 video.flv 的本地视频文件并侦听要调度的 asyncError (AsyncErrorEvent.ASYNC_ERROR)。当本机异步代码中引发异常时调度此事件。在本例中，当视频文件中包含元数据或提示点信息，并且未定义相应的侦听器时，将调度此事件。如果您对视频文件的元数据或提示点信息不感兴趣，则可以使用上面的代码处理 asyncError 事件并忽略错误。如果 FLV 中有元数据和多个提示点，则 trace() 函数将显示以下错误消息：

```
Error #2095: flash.net.NetStream was unable to invoke callback onMetaData.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
Error #2095: flash.net.NetStream was unable to invoke callback onCuePoint.
```

发生错误的原因是 NetStream 对象找不到 onMetaData 或 onCuePoint 回调方法。在应用程序中定义这些回调方法有多种方式。

更多帮助主题

Flash Media Server：处理流中的元数据

将 NetStream 对象的 client 属性设置为一个 Object

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过将 client 属性设置为一个 Object 或设置为 NetStream 的一个子类，可以重新发送 onMetaData 和 onCuePoint 回调方法或彻底忽略这些方法。以下示例演示如何使用空的 Object 忽略这些回调方法而不侦听 asyncError 事件：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var customClient:Object = new Object();

var ns:NetStream = new NetStream(nc);
ns.client = customClient;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

如果想要侦听 onMetaData 或 onCuePoint 回调方法，则需要定义用于处理这些回调方法的方法，如以下代码片断所示：

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

上面的代码侦听 onMetaData 回调方法并调用 metaDataHandler() 方法，后者会输出一个字符串。如果 Flash 运行时遇到一个提示点，那么即使未定义 onCuePoint 回调方法，也不会生成错误。

创建自定义类并定义用于处理回调方法的方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下代码将 NetStream 对象的 client 属性设置为一个自定义类 CustomClient，该类为回调方法定义处理函数：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = new CustomClient();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

CustomClient 类如下所示：

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

CustomClient 类为 onMetaData 回调处理函数定义一个处理函数。如果遇到了提示点，并且调用了 onCuePoint 回调处理函数，则会调度一个 asyncError 事件 (AsyncErrorEvent.ASYNC_ERROR)，显示 “flash.net.NetStream 无法调用回调 onCuePoint”。为了防止发生此错误，需要在 CustomClient 类中定义一个 onCuePoint 回调方法，或者为 asyncError 事件定义一个事件处理函数。

扩展 NetStream 类并添加处理回调方法的方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下代码创建 CustomNetStream 类的一个实例， CustomNetStream 类在后面的代码清单中定义：

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

以下代码清单定义 CustomNetStream 类，该类扩展 NetStream 类、处理必要的 NetConnection 对象的创建，并处理 onMetaData 和 onCuePoint 回调处理函数方法：

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function CustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
        public function onCuePoint(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

如果要重命名 CustomNetStream 类中的 onMetaData() 和 onCuePoint() 方法，可以使用以下代码：

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public class CustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public var onMetaData:Function;
        public var onCuePoint:Function;
        public function CustomNetStream()
        {
            onMetaData = metaDataHandler;
            onCuePoint = cuePointHandler;
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
        private function metaDataHandler(infoObject:Object):void
        {
            trace("metadata");
        }
        private function cuePointHandler(infoObject:Object):void
        {
            trace("cue point");
        }
    }
}
```

扩展 NetStream 类并使其为动态类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以扩展 NetStream 类并使其子类为动态类，以便可以动态添加 onCuePoint 和 onMetaData 回调处理函数。以下代码清单演示了这一过程：

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

DynamicCustomNetStream 类如下所示：

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    public dynamic class DynamicCustomNetStream extends NetStream
    {
        private var nc:NetConnection;
        public function DynamicCustomNetStream()
        {
            nc = new NetConnection();
            nc.connect(null);
            super(nc);
        }
    }
}
```

由于 DynamicCustomNetStream 类为动态类，因此，即使 onMetaData 和 onCuePoint 回调处理函数没有处理函数，也不会引发错误。如果想要为 onMetaData 和 onCuePoint 回调处理函数定义方法，可以使用以下代码：

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.onMetaData = metaDataHandler;
ns.onCuePoint = cuePointHandler;
ns.play("http://www.helpexamples.com/flash/video/cuepoints.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
function cuePointHandler(infoObject:Object):void
{
    trace("cue point");
}
```

将 NetStream 对象的 client 属性设置为 this

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过将 client 属性设置为 this，应用程序将在当前作用域内查找 onMetaData() 和 onCuePoint() 方法。以下示例演示了这一效果：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

如果调用 onMetaData 或 onCuePoint 回调处理函数，在不存在处理该回调的方法时不会生成错误。若要处理这些回调处理函数，请在代码中创建 onMetaData() 和 onCuePoint() 方法，如以下代码片断所示：

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

使用提示点和元数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 NetStream 回调方法可在视频播放时捕获并处理提示点和元数据事件。

使用提示点

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

下表描述了可在 Flash Player 和 AIR 中用来捕获 F4V 和 FLV 提示点的回调方法。

运行时	F4V	FLV
Flash Player 9/ AIR1.0		OnCuePoint
		OnMetaData
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

下面的示例使用简单的 for..in 循环来遍历 onCuePoint() 函数收到的 infoObject 参数中的每一个属性。它在收到提示点数据时，调用 trace() 函数来显示消息：

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onCuePoint(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

显示以下输出：

```
parameters:
name: point1
time: 0.418
type: navigation
```

此代码使用多种技术之一来设置运行回调方法的对象。您可以使用其他技术；有关详细信息，请参阅第 412 页的 “ 编写元数据和提示点的回调方法 ”。

使用视频元数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 OnMetaData() 和 OnXMPData() 函数来访问视频文件中的元数据信息，包括提示点。

使用 OnMetaData()

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

元数据包含有关视频文件的信息，如持续时间、宽度、高度和帧速率。添加到视频文件中的元数据信息取决于您用来编码视频文件的软件。

```
var nc:NetConnection = new NetConnection();
nc.connect(null);

var ns:NetStream = new NetStream(nc);
ns.client = this;
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);

function onMetaData(infoObject:Object):void
{
    var key:String;
    for (key in infoObject)
    {
        trace(key + ": " + infoObject[key]);
    }
}
```

上面的代码生成如下输出：

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

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。

在上面的代码中，未显示提示点信息。为了显示提示点元数据，可以使用以下函数，该函数会以递归方式显示 Object 中的项目：

```
function traceObject(obj:Object, indent:uint = 0):void
{
    var indentString:String = "";
    var i:uint;
    var prop:String;
    var val:*;
    for (i = 0; i < indent; i++)
    {
        indentString += "\t";
    }
    for (prop in obj)
    {
        val = obj[prop];
        if (typeof(val) == "object")
        {
            trace(indentString + " " + prop + ": [Object]");
            traceObject(val, indent + 1);
        }
        else
        {
            trace(indentString + " " + prop + ": " + val);
        }
    }
}
```

使用上面的代码片断在 onMetaData() 方法中跟踪 infoObject 参数将产生以下输出：

```
width: 320
audiodatarate: 96
audiocodecid: 2
videocodecid: 4
videodatarate: 400
canSeekToEnd: true
duration: 16.334
audiodelay: 0.038
height: 213
framerate: 15
cuePoints: [Object]
    0: [Object]
        parameters: [Object]
            lights: beginning
        name: point1
        time: 0.418
        type: navigation
    1: [Object]
        parameters: [Object]
            lights: middle
        name: point2
        time: 7.748
        type: navigation
    2: [Object]
        parameters: [Object]
            lights: end
        name: point3
        time: 16.02
        type: navigation
```

下面的示例显示 MP4 视频的元数据。它假设存在一个名为 metaDataOut 的 TextArea 对象，它将把元数据写入其中。

```
package
{
    import flash.net.NetConnection;
    import flash.net.NetStream;
    import flash.events.NetStatusEvent;
    import flash.media.Video;
    import flash.display.StageDisplayState;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class onMetaDataExample extends Sprite
    {
        var video:Video = new Video();

        public function onMetaDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;

            addChild(video);
            video.x = 185;
            video.y = 5;

            video.attachNetStream(videoStream);

            videoStream.play("video.mp4");

            videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        }

        public function onMetaData(infoObject:Object):void
        {
            for(var propName:String in infoObject)
            {
                metaDataOut.appendText(propName + "=" + infoObject[propName] + "\n");
            }
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if(event.info.code == "NetStream.Play.Stop")
                stage.displayState = StageDisplayState.NORMAL;
        }
    }
}
```

对于此视频，onMetaData() 函数将生成以下输出：

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

使用信息对象

下表显示了在 onMetaData() 回调函数接收的 Object 中传递给 onMetaData() 回调函数的可能的视频元数据值。

参数	说明
aacaot	AAC 音频对象类型；支持 0、 1 或 2。
avclevel	AVC IDC 级别号，如 10、 11、 20、 21 等等。
avcprofile	AVC 配置文件号，如 55、 77、 100 等等。
audiocodecid	指示所用音频编解码器 （编码 / 解码技术）的字符串 -- 例如 “Mp3” 或 “mp4a”
audiodatarate	一个数字，指示音频的编码速率，以每秒千字节为单位。
audiodelay	一个数字，指示原始 FLV 文件的 “time 0” 在 FLV 文件中保持多长时间。为了正确同步音频，视频内容需要有少量的延迟。
canSeekToEnd	一个布尔值，如果 FLV 文件是用最后一帧 （它允许定位到渐进式下载视频文件的末尾）上的关键帧编码的，则该值为 true。如果 FLV 文件不是用最后一帧上的关键帧编码的，则该值为 false。
cuePoints	嵌入在 FLV 文件中的提示点对象组成的数组，每个提示点对应一个对象。如果 FLV 文件不包含任何提示点，则值是未定义的。每个对象都具有以下属性： • type：一个字符串，它将提示点的类型指定为 “navigation” 或 “event”。 • name：一个字符串，表示提示点的名称。 • time：一个数字，表示以秒为单位的提示点的时间，精确到 3 位小数 （毫秒）。 • parameters：一个可选对象，包含用户在创建提示点时指定的名称 - 值对。
duration	一个数字，以秒为单位指定视频文件的持续时间。
framerate	一个数字，表示 FLV 文件的帧速率。
height	一个数字，以像素为单位表示 FLV 文件的高度。
seekpoints	一个数组，其中将可用的关键帧作为时间戳列出，单位为毫秒。可选。
tags	一个键 - 值对数组，这些键 / 值对表示 “ilst” 原子中的信息，相当于 MP4 文件中的 ID3 标签。 iTunes 使用这些标签。可用于显示插图 （如果有插图）。
trackinfo	一个对象，提供有关 MP4 文件中所有轨道的信息 （包括其采样描述 ID）。
videocodecid	一个字符串，表示用于编码视频的编解码器版本。 - 例如， “avc1” 或 “VP6F”
videodatarate	一个数字，表示 FLV 文件的视频数据速率。
videoframerate	MP4 视频的帧速率。
width	一个数字，以像素为单位表示 FLV 文件的宽度。

下表显示 videocodecid 参数的可能值：

videocodecid	编解码器名称
2	Sorenson H.263
3	屏幕视频 （仅限 SWF 版本 7 和更高版本）
4	VP6 （仅限 SWF 版本 8 和更高版本）
5	带有 Alpha 通道的 VP6 视频 （仅限 SWF 版本 8 和更高版本）

下表显示 audiocodecid 参数的可能值：

audiocodecid	编解码器名称
0	未压缩
1	ADPCM
2	Mp3
4	Nellymoser，16 kHz 单声
5	Nellymoser，8kHz 单声
6	Nellymoser
10	AAC
11	Speex

使用 onXMPData()

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

onXMPData() 回调函数接收 Adobe F4V 或 FLV 视频文件中嵌入的 Adobe 可扩展元数据平台 (XMP) 专用信息。XMP 元数据包括提示点以及其他视频元数据。Flash Player 10 和 Adobe AIR 1.5 中引入了 XMP 元数据支持，并且后续版本都支持这种元数据。

下面的示例处理 XMP 元数据中的提示点数据：

```
package
{
    import flash.display.*;
    import flash.net.*;
    import flash.events.NetStatusEvent;
    import flash.media.Video;

    public class onXMPDataExample extends Sprite
    {
        public function onXMPDataExample():void
        {
            var videoConnection:NetConnection = new NetConnection();
            videoConnection.connect(null);

            var videoStream:NetStream = new NetStream(videoConnection);
            videoStream.client = this;
            var video:Video = new Video();

            addChild(video);

            video.attachNetStream(videoStream);

            videoStream.play("video.f4v");
        }

        public function onMetaData(info:Object):void {
            trace("onMetaData fired");
        }

        public function onXMPData(infoObject:Object):void
        {
            trace("onXMPData Fired\n");
            //trace("raw XMP =\n");
            //trace(infoObject.data);
```

```
            var cuePoints:Array = new Array();
            var cuePoint:Object;
            var strFrameRate:String;
            var nTracksFrameRate:Number;
            var strTracks:String = "";
            var onXMPXML = new XML(infoObject.data);
            // Set up namespaces to make referencing easier
            var xmpDM:Namespace = new Namespace("http://ns.adobe.com/xmp/1.0/DynamicMedia/");
            var rdf:Namespace = new Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#");
            for each (var it:XML in onXMPXML..xmpDM::Tracks)
            {
                 var strTrackName:String = it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::trackName;
                 var strFrameRateXML:String = it.rdf::Bag.rdf::li.rdf::Description.@xmpDM::frameRate;
                 strFrameRate = strFrameRateXML.substr(1,strFrameRateXML.length);

                 nTracksFrameRate = Number(strFrameRate);

                 strTracks += it;
            }
            var onXMPTracksXML:XML = new XML(strTracks);
            var strCuepoints:String = "";
            for each (var item:XML in onXMPTracksXML..xmpDM::markers)
            {
                strCuepoints += item;
            }
            trace(strCuepoints);
        }
    }
}
```

对于名为 startrekintro.f4v 的短视频文件，此示例生成以下跟踪行。这些行显示了 XMP 元数据中用于导航的提示点数据以及事件提示点：

```
onMetaData fired
onXMPData Fired

<xmpDM:markers xmlns:xmp="http://ns.adobe.com/xap/1.0/"
xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#" xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:x="adobe:ns:meta/">
  <rdf:Seq>
    <rdf:li>
      <rdf:Description xmpDM:startTime="7695905817600" xmpDM:name="Title1" xmpDM:type="FLVCuePoint"
xmpDM:cuePointType="Navigation">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="Title" xmpDM:value="Star Trek"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
    <rdf:li>
      <rdf:Description xmpDM:startTime="10289459980800" xmpDM:name="Title2" xmpDM:type="FLVCuePoint"
xmpDM:cuePointType="Event">
        <xmpDM:cuePointParams>
          <rdf:Seq>
            <rdf:li xmpDM:key="William Shatner" xmpDM:value="First Star"/>
            <rdf:li xmpDM:key="Color" xmpDM:value="Light Blue"/>
          </rdf:Seq>
        </xmpDM:cuePointParams>
      </rdf:Description>
    </rdf:li>
  </rdf:Seq>
</xmpDM:markers>
onMetaData fired
```

注： 在 XMP 数据中，时间是作为 DVA 刻度存储的，而不是秒。若要计算提示点时间，请将启动时间除以帧速率。例如，启动时间 7695905817600 除以帧速率 254016000000 等于 30:30。

若要查看全部原始 XMP 元数据，包括帧速率，请删除位于 onXMPData() 函数开头的第二和第三条 trace() 语句之前的注释标识符 (//)。

有关 XMP 的详细信息，请参阅：

- partners.adobe.com/public/developer/xmp/topic.html
- www.adobe.com/devnet/xmp/

使用图像元数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

onImageData 事件通过 AMF0 数据通道将图像数据作为字节数组发送。该数据可以是 JPEG、 PNG 或 GIF 格式。可定义 onImageData() 回调方法来处理此信息，这与定义 onCuePoint 和 onMetaData 的回调方法的方式相同。下面的示例使用 onImageData() 回调方法访问并显示图像数据。

```
public function onImageData(imageData:Object):void
{
    // display track number
    trace(imageData.trackid);
    var loader:Loader = new Loader();
    //imageData.data is a ByteArray object
    loader.loadBytes(imageData.data);
    addChild(loader);
}
```

使用文本元数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

onTextData 事件通过 AMF0 数据通道发送文本数据。文本数据为 UTF-8 格式，并且包含有关基于 3GP 计时文本规范的格式化的详细信息。此规范定义了标准化的字幕格式。可定义 onTextData() 回调方法来处理此信息，这与定义 onCuePoint 或 onMetaData 的回调方法的方式相同。在下面的示例中， onTextData() 方法显示曲目 ID 号和对应的曲目文本。

```
public function onTextData(textData:Object):void
{
    // display the track number
    trace(textData.trackid);
    // displays the text, which can be a null string, indicating old text
    // that should be erased
    trace(textData.text);
}
```

监控 NetStream 活动

Flash Player 10.3 和更高版本， **Adobe AIR 2.7** 和更高版本

可监控 NetStream 活动以收集所需的信息来支持媒体使用分析和报告。本节讨论的监控功能支持您创建媒体测量库，该库可不用断开与显示媒体的特定视频播放器的连接来收集数据。这可让您的客户端开发人员在使用您的库时选择自己最喜欢的视频播放器。使用 NetMonitor 类来监控应用程序中 NetStream 对象的创建和活动。 NetMonitor 类可提供在任何给定时间存在的活动 NetStream 的列表，并且一旦创建了 NetStream 对象就会调度一个事件。

NetStream 对象调度下表中列出的事件，具体取决于正在播放的媒体类型：

事件	渐进式下载	RTMP 流	HTTP 流
NetStream.Play.Start	是	是	否
NetStream.Play.Stop	是	是	否
NetStream.Play.Complete	是	是	否
NetStream.SeekStart.Notify	是	是	是
NetStream.Seek.Notify	是	是	是
NetStream.Unpause.Notify	是	是	是
NetStream.Unpause.Notify	是	是	是
NetStream.Play.Transition	不适用	是	不适用
NetStream.Play.TransitionComplete	不适用	是	不适用

事件	渐进式下载	RTMP 流	HTTP 流
NetStream.Buffer.Full	是	是	是
NetStream.Buffer.Flush	是	是	是
NetStream.Buffer.Empty	是	是	是

与 NetStream 实例关联的 NetStreamInfo 对象也存储最后的元数据以及在媒体中遇到的 XMP 数据对象。

当通过 HTTP 流播放媒体时，不会调度 NetStream.Play.Start、 NetStream.Play.Stop 以及 NetStream.Play.Complete，因为应用程序完全控制媒体流。视频播放器应该为 HTTP 流综合并调度这些事件。

相似地，不会为渐进的下载或 HTTP 媒体调度 NetStream.Play.Transition 和 NetStream.Play.TransitionComplete。动态位率切换是一个 RTMP 功能。如果视频播放器使用支持相似功能的 HTTP 流，播放器就可综合并调度转换事件。

更多帮助主题

Adobe Developer Connection：测量 Flash 中的视频消耗

监控 NetStream 事件

有两类事件可提供有价值的使用数据：netStatus 和 mediaTypeData。此外，可使用定时器来定期记录 NetStream 播放头的位置。

netStatus 事件提供您可用于确定用户观看了多少流的信息。缓冲区和 RTMFP 流转换事件也会导致 netStatus 事件。

mediaTypeData 事件可提供元数据和 XMP 数据信息。 Netstream.Play.Complete 事件作为 mediaTypeData 事件调度。嵌套在流中的其他数据也可通过 mediaTypeData 事件获取，这些数据包括提示点、文本和图像。

下面的示例展示了如何创建一个类，这个类可以监视应用程序中任意活动 NetStream 的状态和数据事件。通常，该类会将需要分析的数据上传到服务器进行收集。

```
package com.adobe.example
{
    import flash.events.NetDataEvent;
    import flash.events.NetMonitorEvent;
    import flash.events.NetStatusEvent;
    import flash.net.NetMonitor;
    import flash.net.NetStream;

    public class NetStreamEventMonitor
    {
        private var netmon:NetMonitor;
        private var heartbeat:Timer = new Timer( 5000 );

        public function NetStreamEventMonitor()
        {
            //Create NetMonitor object
            netmon = new NetMonitor();
            netmon.addEventListener( NetMonitorEvent.NET_STREAM_CREATE, newNetStream );

            //Start the heartbeat timer
            heartbeat.addEventListener( TimerEvent.TIMER, onHeartbeat );
            heartbeat.start();
        }

        //On new NetStream
        private function newNetStream( event:NetMonitorEvent ):void
        {
            trace( "New Netstream object");
```

```
            var stream:NetStream = event.netStream;
            stream.addEventListener(NetDataEvent.MEDIA_TYPE_DATA, onStreamData);
            stream.addEventListener(NetStatusEvent.NET_STATUS, onStatus);
        }

        //On data events from a NetStream object
        private function onStreamData( event:NetDataEvent ):void
        {

            var netStream:NetStream = event.target as NetStream;
            trace( "Data event from " + netStream.info.uri + " at " + event.timestamp );
            switch( event.info.handler )
            {
                case "onMetaData":
                    //handle metadata;
                    break;
                case "onXMPData":
                    //handle XMP;
                    break;
                case "onPlayStatus":
                    //handle NetStream.Play.Complete
                case "onImageData":
                    //handle image
                    break;
                case "onTextData":
                    //handle text
                    break;
                default:
                    //handle other events

            }
        }

        //On status events from a NetStream object
        private function onStatus( event:NetStatusEvent ):void
        {
            trace( "Status event from " + event.target.info.uri + " at " + event.target.time );
            //handle status events
        }
        //On heartbeat timer
        private function onHeartbeat( event:TimerEvent ):void
        {
            var streams:Vector.<NetStream> = netmon.listStreams();
            for( var i:int = 0; i < streams.length; i++ )
            {
                trace( "Heartbeat on " + streams[i].info.uri + " at " + streams[i].time );
                //handle heartbeat event
            }
        }

    }
}
```

检测播放器域

用户在上面观看媒体内容的网页的 URL 和域并非始终随时可用。如果托管网站允许，您可使用 ExternalInterface 类获取确切 URL。尽管如此，允许第三方视频播放器的一些网站不允许使用 ExternalInterface。在这种情况下，您可通过 Security 类的 pageDomain 属性获取当前网页的域。出于用户安全和隐私的原因，不会泄露完整的 URL。

页面域可通过 Security 类的静态 pageDomain 属性获取：

```
var domain:String = Security.pageDomain;
```

视频文件的高级主题

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下主题介绍使用 FLV 文件的一些特殊问题。

关于配置 FLV 文件以便在服务器上托管

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在处理 FLV 文件时，您可能需要配置服务器以便使用 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
- 必需参数：无
- 可选参数：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.adobe.com/go/video_file_format_cn

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 Server® 2003 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 Server 2003 上的所有 FLV 文件，包括用早期版本的 Flash 创作工具以及 Adobe 的 Macromedia Flash Video Kit for Dreamweaver MX 2004 制作的文件。如果在其他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 以流式传输 FLV 视频的信息，请参阅 www.adobe.com/go/tn_19439_cn。

关于在 Macintosh 上将本地 FLV 文件设定为目标

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果您尝试从 Apple® Macintosh® 计算机的非系统驱动器上，使用以斜杠 (/) 表示的相对路径来播放本地 FLV，将无法播放视频。非系统驱动器包括 （但不限于） CD-ROM、分区的硬盘、可移除的存储媒体以及连接的存储设备。

注：导致失败发生的原因是操作系统的局限，而非 Flash Player 或 AIR 的局限。

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：(Mac OS®) myDrive:myFolder:myFLV.flv

视频示例：视频自动唱片点唱机

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下示例生成一个简单的视频自动唱片点唱机，它可以动态加载一组视频并按顺序播放。您可以生成一个可以让用户浏览一系列视频教程的应用程序，也可以指定在传送用户请求的视频之前应播放的广告。此示例演示了 ActionScript 3.0 的下列功能：

- 根据视频文件的播放进度更新播放头
- 侦听并分析视频文件的元数据
- 处理网络流中的特定代码
- 加载、播放、暂停和停止动态加载的 FLV
- 根据网络流的元数据调整显示列表上视频对象的大小

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。“视频自动唱片点唱机”应用程序文件位于 Samples/VideoJukebox 文件夹中。该应用程序包含以下文件：

文件	说明
VideoJukebox.fla 或 VideoJukebox.mxml	Flex 或 Flash 的主应用程序文件（分别为 MXML 和 FLA 格式）。
VideoJukebox.as	此类提供了应用程序的主要功能。
playlist.xml	列出要向视频自动唱片点唱机中加载哪些视频文件的文件。

加载外部视频播放列表文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

外部 playlist.xml 文件指定要加载的视频以及播放这些文件的顺序。为了加载该 XML 文件，您需要使用一个 URLLoader 对象和一个 URLRequest 对象，如以下代码所示：

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

此代码放置在 VideoJukebox 类的构造函数内，因此，在运行其他任何代码之前，会首先加载该文件。一旦 XML 文件加载完成后，即会调用 xmlCompleteHandler() 方法，该方法会将该外部文件分析为一个 XML 对象，如以下代码所示：

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

playlist XML 对象包含来自外部文件的原始 XML，而视频 XML 是仅包含视频节点的 XMLList 对象。以下代码片断显示了一个示例 playlist.xml 文件：

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

最后， xmlCompleteHandler() 方法将调用 main() 方法，被调用的方法会在显示列表上设置各个组件实例以及用于加载外部 FLV 文件的 NetConnection 和 NetStream 对象。

创建用户界面

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要构建用户界面，您需要将 5 个 Button 实例拖到显示列表上并为其指定以下实例名称：playButton、 pauseButton、 stopButton、 backButton 和 forwardButton。

对于这些 Button 实例中的每个实例，您需要为 click 事件分配一个处理函数，如以下代码片断所示：

```
playButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
pauseButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
stopButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
backButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
forwardButton.addEventListener(MouseEvent.CLICK, buttonClickHandler);
```

buttonClickHandler() 方法使用 switch 语句来确定单击了哪个按钮实例，如以下代码所示：

```
private function buttonClickHandler(event:MouseEvent):void
{
    switch (event.currentTarget)
    {
        case playButton:
            ns.resume();
            break;
        case pauseButton:
            ns.togglePause();
            break;
        case stopButton:
            ns.pause();
            ns.seek(0);
            break;
        case backButton:
            playPreviousVideo();
            break;
        case forwardButton:
            playNextVideo();
            break;
    }
}
```

接下来，向显示列表中添加一个 Slider 实例，并为其指定实例名称 volumeSlider。以下代码将该滑块实例的 liveDragging 属性设置为 true，并为该滑块实例的 change 事件定义一个事件侦听器：

```
volumeSlider.value = volumeTransform.volume;
volumeSlider.minimum = 0;
volumeSlider.maximum = 1;
volumeSlider.snapInterval = 0.1;
volumeSlider.tickInterval = volumeSlider.snapInterval;
volumeSlider.liveDragging = true;
volumeSlider.addEventListener(SliderEvent.CHANGE, volumeChangeHandler);
```

向显示列表中添加一个 ProgressBar 实例，并为其指定实例名称 positionBar。将其 mode 属性设置为 manual，如以下代码片断所示：

```
positionBar.mode = ProgressBarMode.MANUAL;
```

最后，向显示列表中添加一个 Label 实例，并为其指定实例名称 positionLabel。此 Label 实例的值将由计时器实例设置

侦听视频对象的元数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当 Flash Player 遇到已加载的每个视频的元数据时，会对 NetStream 对象的 client 属性调用 onMetaData() 回调处理函数。以下代码初始化一个对象并设置指定的回调处理函数：

```
client = new Object();
client.onMetaData = metadataHandler;
```

metadataHandler() 方法将其数据复制到在代码前面部分中定义的 meta 属性中。这可使您在应用程序运行过程中随时访问当前视频的元数据。接下来，调整舞台上视频对象的大小，使其与从元数据中返回的尺寸相符。最后，根据当前正在播放的视频的尺寸移动并调整 positionBar 进度栏实例的大小。以下代码包含完整的 metadataHandler() 方法：

```
private function metadataHandler(metadataObj:Object):void
{
    meta = metadataObj;
    vid.width = meta.width;
    vid.height = meta.height;
    positionBar.move(vid.x, vid.y + vid.height);
    positionBar.width = vid.width;
}
```

动态加载视频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

为了动态加载每个视频，应用程序使用 NetConnection 和 NetStream 对象。以下代码创建一个 NetConnection 对象并将 null 传递给 connect() 方法。通过指定 null，Flash Player 会连接到本地服务器上的视频，而不连接到服务器（如 Flash Media Server）。

以下代码创建 NetConnection 和 NetStream 实例，为 netStatus 事件定义事件侦听器，并将 client 对象分配给 client 属性：

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

每当视频的状态发生改变时即会调用 netStatusHandler() 方法。这包括视频开始或停止播放时、缓冲时或找不到视频流时。以下代码列出了 netStatusHandler() 事件：

```
private function netStatusHandler(event:NetStatusEvent):void
{
    try
    {
        switch (event.info.code)
        {
            case "NetStream.Play.Start":
                t.start();
                break;
            case "NetStream.Play.StreamNotFound":
            case "NetStream.Play.Stop":
                t.stop();
                playNextVideo();
                break;
        }
    }
    catch (error:TypeError)
    {
        // Ignore any errors.
    }
}
```

上面的代码计算 info 对象的 code 属性，并过滤出为 “NetStream.Play.Start”、 “NetStream.Play.StreamNotFound” 或 “NetStream.Play.Stop” 的代码。其他所有代码均被忽略。如果网络流已开始，代码会启动用于更新播放头的 Timer 实例。如果找不到网络流或网络流已停止，则会停止 Timer 实例，应用程序将尝试播放播放列表中的下一个视频。

每次执行 Timer 时， positionBar 进度栏实例都会通过调用 ProgressBar 类的 setProgress() 方法来更新其当前位置，并且会用当前视频已经过的时间和总时间来更新 positionLabel Label 实例。

```
private function timerHandler(event:TimerEvent):void
{
    try
    {
        positionBar.setProgress(ns.time, meta.duration);
        positionLabel.text = ns.time.toFixed(1) + " of " meta.duration.toFixed(1) + " seconds";
    }
    catch (error:Error)
    {
        // Ignore this error.
    }
}
```

控制视频音量

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过在 NetStream 对象上设置 soundTransform 属性，您可以控制动态加载的视频的音量。视频自动唱片点唱机应用程序允许您通过更改 volumeSlider Slider 实例的值来修改音量级别。以下代码演示如何通过将 Slider 组件的值赋给 SoundTransform 对象 （在 NetStream 对象上设置为 soundTransform 属性）来更改音量级别：

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

控制视频播放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序的其余部分是在视频到达视频流末尾或用户跳到上一个或下一个视频时控制视频的播放。

下面的方法可以从当前所选索引的 XMLList 中检索视频 URL：

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

playVideo() 方法会调用 NetStream 对象上的 play() 方法以加载当前所选的视频：

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

playPreviousVideo() 方法使当前视频索引递减、调用 playVideo() 方法来加载新的视频文件并将进度栏设置为可见：

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

最后一个方法 playNextVideo() 使视频索引递增并调用 playVideo() 方法。如果当前视频是播放列表中的最后一个视频，则会对 Video 对象调用 clear() 方法，并将进度栏实例的 visible 属性设置为 false：

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

使用 StageVideo 类来实现硬件加速呈现

Flash Player 10.2 和更高版本

Flash Player 将使用用于 H.264 解码的硬件加速来优化视频性能。您可以使用 StageVideo API 进一步增强性能。舞台视频允许应用程序利用硬件加速呈现。

支持 StageVideo API 的运行时包括：

- Flash Player 10.2 和更高版本

注：在 Flash Player 11.4/AIR 3.4 和更高版本中，可以将摄像头输入用于 StageVideo。

舞台视频快速入门提供了舞台视频功能的可下载源代码和其他详细信息。

要获得 StageVideo 快速入门教程，请参阅使用舞台视频。

关于使用 StageVideo 实现硬件加速

硬件加速呈现 （包括视频缩放、颜色转换和位图传送）增强了硬件加速解码的性能优势。在提供 GPU （硬件）加速功能的设备上，可以使用 flash.media.StageVideo 对象直接在设备硬件上处理视频。直接处理会释放 CPU，令其可以在 GPU 处理视频时执行其他任务。另一方面，旧版 Video 类通常会使用软件呈现。软件呈现通过设备 CPU 来实现，会消耗大量共享的系统资源。

当前，只有很少的设备提供完全 GPU 加速。不过，舞台视频允许应用程序最大限度地利用可用硬件加速功能。

StageVideo 类不会令 Video 类废弃。组合使用这两个类，可以在任何给定时间提供设备资源所允许的最佳视频显示体验。应用程序通过侦听相应的事件并根据需要在 StageVideo 和 Video 之间切换，从而利用硬件加速功能。

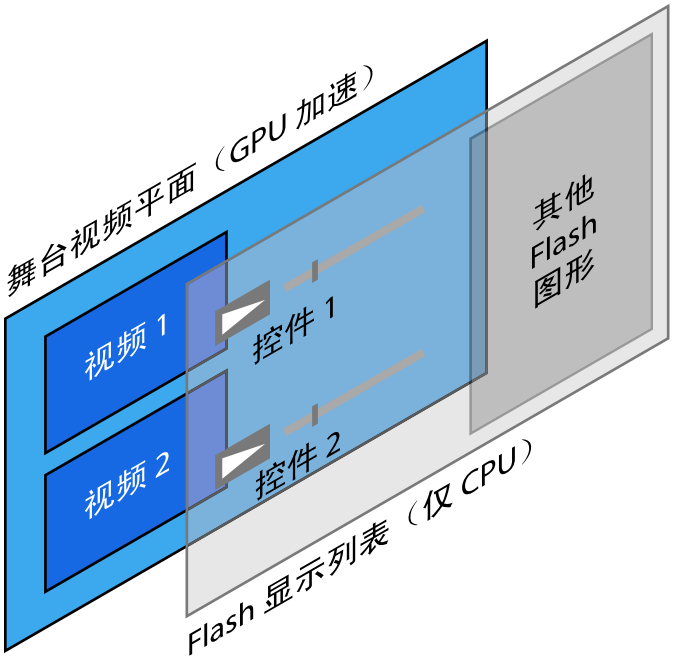
StageVideo 类对视频的使用施加某些限制。在实现 StageVideo 之前，请查看原则，并确保您的应用程序可以接受这些原则。如果您接受这些限制，则不论何时 Flash Player 检测到硬件加速呈现可用时，都请使用 StageVideo 类。请参阅第 435 页的 “指导准则和限制 ”。

平行面：舞台视频和 Flash 显示列表

利用舞台视频模型，Flash Player 可以将视频从显示列表中分离。Flash Player 在两个按 Z 轴排列的平面之间拆分复合显示。

舞台视频平面 舞台视频平面位于背景中。它仅显示硬件加速视频。按照此设计方案，当设备不支持硬件加速或硬件加速不可用时，该平面不可用。在 ActionScript 中， StageVideo 对象处理舞台视频平面上播放的视频。

Flash 显示列表平面 Flash 显示列表实体组合显示在位于舞台视频平面前方的平面上。显示列表实体包含运行时呈现的所有内容 （包括播放控件）。当硬件加速不可用时，只能使用 Video 类对象在此平面上播放视频。舞台视频始终显示在 Flash 显示列表图形的后面。



视频显示平面

StageVideo 对象显示在屏幕上窗口对齐的非旋转矩形区域中。您不能在舞台视频平面后分层显示对象。不过，可以使用 Flash 显示列表平面将其他图形分层显示在舞台视频平面的上方。舞台视频与显示列表并行运行。因此，您可以组合使用这两种机制利用两个不同平面创建统一的视觉效果。例如，可以使用前面的平面显示播放控件，用于操作在背景中运行的舞台视频。

舞台视频和 H.264 编解码器

在 Flash Player 应用程序中，实现视频加速涉及到两个步骤：

1 将视频编码为 H.264

2 实现 StageVideo API

为了获得最佳结果，请执行下面两个步骤：H.264 编解码器允许您最大限度利用硬件加速功能，包括从视频解码到呈现。

舞台视频不再执行从 GPU 到 CPU 回读。换言之，GPU 不再将已解码的帧发送回 CPU 以便与显示列表对象复合，而是将已解码和已呈现的帧直接以位块形式传输到屏幕上，并将其置于显示列表对象后面。此项技术会减少 CPU 和内存的使用，另外还提供更好的像素保真度。

更多帮助主题

第 402 页的 “ 了解视频格式 ”

指导准则和限制

当视频在全屏模式下运行时，如果设备支持硬件加速，则舞台视频始终可用。不过， Flash Player 还会在浏览器内运行。在浏览器上下文中， wmode 设置会影响舞台视频的可用性。如果您希望使用舞台视频，请始终尝试使用 wmode="direct"。在非全屏模式下，舞台视频与其他 wmode 设置不兼容。此项限制意味着在运行时，舞台视频可能会在可用和不可用之间摇摆不定。例如，在运行舞台视频时，如果用户退出全屏模式，则视频上下文将恢复为浏览器。如果浏览器的 wmode 参数未设置为 "direct"，则视频会突然变得不可用。 Flash Player 会通过一组事件将播放上下文更改发送给应用程序。如果您实现 StageVideo API，当舞台视频变得不可用时，请将 Video 对象保留为备份。

由于舞台视频直接与硬件建立关系，因此舞台视频会限制一些视频功能。舞台视频执行以下限制：

- 对于每个 SWF 文件， Flash Player 将可同时显示视频的 StageVideo 对象数量限制为四个。但是，实际限值可能会更低，具体取决于设备硬件资源。
- 视频计时与运行时显示的内容的计时不同步。
- 视频显示区域只能是矩形。您无法使用更先进的显示区域，如椭圆形或不规则形状。
- 无法旋转视频。
- 无法以位图形式缓存视频，也无法使用 BitmapData 对象访问视频。
- 无法对视频应用滤镜。
- 无法对视频应用颜色转化。
- 无法对视频应用 Alpha 值。
- 您应用到显示列表平面中对象的混合模式不会应用到舞台视频。
- 只能将视频放置在全像素边界上。
- 尽管 GPU 呈现模式是给定设备硬件的最佳可用模式，但不同设备并不能实现 100%“ 像素相同 ”。像素会因驱动程序和平台的不同而略有差异。
- 少数设备不支持所有必需的色彩空间。例如，一些设备不支持 H.264 标准 BT.709。在此类情况下，可使用 BT.601 进行快速显示。
- 如果 WMODE 设置为 “normal”、“opaque” 或 “transparent”，则无法使用舞台视频。在非全屏模式下，舞台视频仅支持 WMODE=direct。 WMODE 在 Safari 4 或更高版本和 IE 9 或更高版本中无效。

大多数情况下，上述限制对视频播放器应用程序没有影响。如果您能够接受上述限制，则应尽可能使用舞台视频。

更多帮助主题

第 139 页的 “ 使用全屏模式 ”

使用 StageVideo API

舞台视频是运行时中的一种视频呈现机制，可增强视频播放和设备性能。运行时负责创建并维护该机制；作为开发人员，您的职责是配置应用程序以利用该机制。

为了使用舞台视频，您需要实现事件处理函数框架，以检测舞台视频何时可用以及何时不可用。当接收到指示舞台视频可用的通知时，从 Stage.stageVideos 属性中检索 StageVideo 对象。运行时使用一个或多个 StageVideo 对象填充此 Vector 对象。然后，您就可以使用所提供的 StageVideo 对象替代 Video 对象来显示流式传输视频。

在 Flash Player 中，当您接收到指示舞台视频不再可用的通知时，请将视频流切换回 Video 对象。

注：您不能创建 StageVideo 对象。

Stage.stageVideos 属性

Stage.stageVideos 属性是一个 Vector 对象，通过该对象可访问 StageVideo 实例。此矢量最多可包含 4 个 StageVideo 对象，具体取决于硬件和系统资源。移动设备可以限制为 1 个对象或无对象。

当舞台视频不可用时，该矢量不包含任何对象。为了避免运行时错误，当接收到指示舞台视频可用的最近 StageVideoAvailability 事件时，请确保仅访问该矢量的成员。

StageVideo 事件

StageVideo API 框架提供了以下事件：

StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 当 Stage.stageVideos 属性发生更改时发送该事件。StageVideoAvailabilityEvent.availability 属性指示 AVAILABLE 或 UNAVAILABLE。使用该事件可确定 stageVideos 属性是否包含任何 StageVideo 对象，而不必直接检查 Stage.stageVideos 矢量的长度。

StageVideoEvent.RENDER_STATE 在 NetStream 或 Camera 对象已附加到 StageVideo 对象并正在播放时发送。指示当前使用的解码类型：硬件、软件或不可用 （不显示任何内容）。事件目标包含 videoWidth 和 videoHeight 属性，调整视频视口的大小时可以安全地使用这些属性。

重要说明：从 StageVideo 目标对象获取的坐标使用舞台坐标，这是因为他们不是标准显示列表的一部分。

VideoEvent.RENDER_STATE 当正在使用 Video 对象时发送该事件。指示当前使用的是软件加速解码还是硬件加速解码。如果该事件指示硬件加速解码，则在可能的情况下，应切换到 StageVideo 对象。 Video 事件目标包含 videoWidth 和 videoHeight 属性，调整视频视口大小时可以安全地使用这些属性。

实现 StageVideo 功能的工作流程

请执行这些顶级步骤来实现 StageVideo 功能：

1. 侦听 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件以了解 Stage.stageVideos 矢量何时更改。请参阅第 438 页的 " 使用 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件 "。
2. 如果 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件报告舞台视频可用，则可以在事件处理函数中使用 Stage.stageVideos Vector 对象来访问 StageVideo 对象。
3. 使用 StageVideo.attachNetStream() 附加一个 NetStream 对象，或使用 StageVideo.attachCamera() 附加一个 Camera 对象。
4. 使用 NetStream.play() 播放视频。
5. 侦听 StageVideo 对象上的 StageVideoEvent.RENDER_STATE 事件，以确定播放视频的状态。收到此事件还说明，视频的宽度和高度属性已初始化或更改。请参阅第 439 页的 " 使用 StageVideoEvent.RENDER_STATE 和 VideoEvent.RENDER_STATE 事件 "。
6. 侦听 Video 对象上的 VideoEvent.RENDER_STATE 事件。此事件提供与 StageVideoEvent.RENDER_STATE 相同的状态，因此您也可以使用它来确定 GPU 加速是否可用。收到此事件还说明，视频的宽度和高度属性已初始化或更改。请参阅第 439 页的 " 使用 StageVideoEvent.RENDER_STATE 和 VideoEvent.RENDER_STATE 事件 "。

初始化 StageVideo 事件侦听器

在应用程序初始化过程中设置 StageVideoAvailabilityEvent 和 VideoEvent 侦听器。例如，可以在 flash.events.Event.ADDED_TO_STAGE 事件处理函数中初始化这些侦听器。此事件可保证您的应用程序在舞台上可见；

```
public class SimpleStageVideo extends Sprite
    private var nc:NetConnection;
    private var ns:NetStream;

    public function SimpleStageVideo()
    {
        // Constructor for SimpleStageVideo class
        // Make sure the app is visible and stage available
        addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    }

    private function onAddedToStage(event:Event):void
    {
        //...
        // Connections
        nc = new NetConnection();
        nc.connect(null);
        ns = new NetStream(nc);
        ns.addEventListener(NetStatusEvent.NET_STATUS, onNetStatus);
        ns.client = this;

        // Screen
        video = new Video();
        video.smoothing = true;

        // Video Events
        // the StageVideoEvent.STAGE_VIDEO_STATE informs you whether
        // StageVideo is available
        stage.addEventListener(StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY,
            onStageVideoState);
        // in case of fallback to Video, listen to the VideoEvent.RENDER_STATE
        // event to handle resize properly and know about the acceleration mode running
        video.addEventListener(VideoEvent.RENDER_STATE, videoStateChange);
        //...
    }
```

使用 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件

在 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 处理函数中，根据 StageVideo 的可用性确定是使用 Video 对象还是使用 StageVideo 对象。如果 StageVideoAvailabilityEvent.availability 属性设置为 StageVideoAvailability.AVAILABLE，则使用 StageVideo。在这种情况下，可依赖于 Stage.stageVideos 矢量来包含一个或多个 StageVideo 对象。从 Stage.stageVideos 属性获取 StageVideo 对象，并将 NetStream 对象附加到它。由于 StageVideo 对象始终显示在背景中，因此需删除所有现有的 Video 对象 （始终位于前景中）。还可以使用此事件处理函数添加 StageVideoEvent.RENDER_STATE 事件的侦听器。

如果 StageVideoAvailabilityEvent.availability 属性设置为 StageVideoAvailability.UNAVAILABLE，请勿使用 StageVideo 或访问 Stage.stageVideos 矢量。在这种情况下，请将 NetStream 对象附加到 Video 对象。最后，请将 StageVideo 或 Video 对象添加到舞台，并调用 NetStream.play()。

下面的代码显示如何处理 StageVideoAvailabilityEvent.STAGE_VIDEO_AVAILABILITY 事件：

```
private var sv:StageVideo;
private var video:Video;

private function onStageVideoState(event:StageVideoAvailabilityEvent):void
{
    // Detect if StageVideo is available and decide what to do in toggleStageVideo
    toggleStageVideo(event.availability == StageVideoAvailability.AVAILABLE);
}

private function toggleStageVideo(on:Boolean):void
{
    // To choose StageVideo attach the NetStream to StageVideo
    if (on)
    {
        stageVideoInUse = true;
        if ( sv == null )
        {
            sv = stage.stageVideos[0];
            sv.addEventListener(StageVideoEvent.RENDER_STATE, stageVideoStateChange);
                sv.attachNetStream(ns);
        }

        if (classicVideoInUse)
        {
            // If you use StageVideo, remove from the display list the
            // Video object to avoid covering the StageVideo object
            // (which is always in the background)
            stage.removeChild ( video );
            classicVideoInUse = false;
        }
    } else
    {
        // Otherwise attach it to a Video object
        if (stageVideoInUse)
            stageVideoInUse = false;
        classicVideoInUse = true;
        video.attachNetStream(ns);
        stage.addChildAt(video, 0);
    }

    if ( !played )
    {
        played = true;
        ns.play(FILE_NAME);
    }
}
```

重要说明：当应用程序第一次在 Stage.stageVideos[0] 访问 vector 元素时，默认矩形设置为 0,0,0,0, 并使用默认值平移和缩放属性。请始终将这些值设置您首选的设置。您可以使用 StageVideoEvent.RENDER_STATE 或 VideoEvent.RENDER_STATE 事件的 videoWidth 和 videoHeight 属性来计算视频视口尺寸。

可从舞台视频快速入门下载本示例应用程序的完整源代码。

使用 StageVideoEvent.RENDER_STATE 和 VideoEvent.RENDER_STATE 事件

当显示环境变化时， StageVideo 和 Video 对象会发送事件以通知应用程序。这些事件为 StageVideoEvent.RENDER_STATE 和 VideoEvent.RENDER_STATE。

当 NetStream 对象已附加并开始播放时， StageVideo 或 Video 对象会调度一个呈现状态事件。当显示环境变化时，它还会发送此事件；例如，当视频视口的大小调整时。请使用这些通知将视口重新设置为事件目标对象的当前 videoHeight 和 videoWidth 值。

已报告的呈现状态包括：

- RENDER_STATUS_UNAVAILABLE
- RENDER_STATUS_SOFTWARE
- RENDER_STATUS_ACCELERATED

不论当前播放视频的是哪个类，使用硬件加速解码时，均会指示呈现状态。请查看 StageVideoEvent.status 属性了解必要的解码是否可用。如果此属性设置为 “unavailable”，则 StageVideo 对象不能播放此视频。此状态要求您立即将 NetStream 对象重新附加到 Video 对象。其他状态会将当前的呈现情况通知您的应用程序。

下表说明了 Flash Player 中 StageVideoEvent 和 VideoEvent 对象的所有呈现状态值的影响。

	VideoStatus.ACCELERATED	VideoStatus.SOFTWARE	VideoStatus.UNAVAILABLE
StageVideoEvent	解码和呈现均发生在硬件中。（最佳性能。）	通过硬件呈现，使用软件解码。（可接受的性能。）	没有可用的 GPU 资源来处理视频，不显示任何内容。回退到 **Video** 对象。
VideoEvent	使用软件呈现，通过硬件解码。（仅在现代桌面系统中可达到可接受的性能。全屏性能会下降。）	使用软件呈现，使用软件解码。（最差的性能表现。全屏性能会下降。）	（不适用）

色彩空间

舞台视频使用基础硬件功能提供色彩空间支持。 SWF 内容可提供指示其首选色彩空间的元数据。但是，设备图形硬件决定是否可使用该色彩空间。某一台设备可能会支持多种色彩空间，而另一台设备却不支持任何色彩空间。如果硬件不支持所请求的色彩空间，则 Flash Player 将尝试在所支持的色彩空间中查找匹配度最高的色彩空间。

若要查询硬件支持的色彩空间，请使用 StageVideo.colorSpaces 属性。此属性以 String 矢量的形式返回所支持的色彩空间列表：

```
var colorSpace:Vector.<String> = stageVideo.colorSpaces();
```

若要了解当前播放的视频所使用的色彩空间，请检查 StageVideoEvent.colorSpace 属性。在 StageVideoEvent.RENDER_STATE 事件的事件处理函数中检查此属性：

```
var currColorSpace:String;

//StageVideoEvent.RENDER_STATE event handler
private function stageVideoRenderState(event:Object):void
{
    //...
    currColorSpace = (event as StageVideoEvent).colorSpace;
    //...
}
```

如果 Flash Player 无法为不支持的色彩空间找到替代色彩空间，舞台视频将使用默认的 BT.601 色彩空间。例如，采用 H.264 编码的视频流通常使用 BT.709 色彩空间。如果设备硬件不支持 BT.709，则 colorSpace 属性将返回 "BT601"。如果 StageVideoEvent.colorSpace 的值为 "unknown"，则指示硬件未提供色彩空间查询方法。

如果您的应用程序认为当前色彩空间不可接受，可选择从 StageVideo 对象切换到 Video 对象。Video 类通过软件合成支持所有色彩空间。

第 26 章：使用摄像头

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

附加到用户计算机上的摄像头可以作为您使用 ActionScript 进行显示和操作的视频数据的来源。 Camera 类是构建在 ActionScript 中用于使用计算机或设备摄像头的机制。

在移动设备上，还可以使用 CameraUI 类。 CameraUI 类启动一个单独的摄像头应用程序，以便用户捕获静止图像或视频。当用户完成操作后，您的应用程序可以通过 MediaPromise 对象访问图像或视频。

更多帮助主题

Christian Cantrell：如何以跨平台方式使用 CameraUI

Michaël CHAIZE：Android、 AIR 和摄像头

了解 Camera 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Camera 对象可以连接到用户的本地摄像头并在本地广播视频 （播放给用户），或将其广播到远程服务器 （比如 Flash Media Server）。

使用 Camera 类可以访问有关用户摄像头的以下各种信息：

- 可以使用用户计算机或设备上安装的哪些摄像头
- 是否安装了摄像头
- 允许还是拒绝 Flash Player 访问用户的摄像头
- 哪个摄像头当前处于活动状态
- 正在捕获的视频的宽度和高度

Camera 类包括多个有用的方法和属性，通过这些方法和属性可以使用 Camera 对象。例如，静态 Camera.names 属性包含由当前安装在用户计算机上的摄像头的名称所组成的数组。您也可以使用 name 属性显示当前处于活动状态的摄像头的名称。

注：在通过网络传输摄像头视频流时，应始终处理网络中断。引发网络中断的原因有多种，尤其是在移动设备上。

在屏幕上显示摄像头内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

连接到摄像头所需的代码比使用 NetConnection 和 NetStream 类加载视频的代码要少。由于在使用 Flash Player 时，您需要先请求用户允许您连接其摄像头，然后才能访问这些摄像头，因此， Camera 类可能很可能变得非常麻烦。

以下代码演示如何使用 Camera 类连接到用户的本地摄像头：

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

注：Camera 类没有构造函数方法。若要创建新的 Camera 实例，请使用静态 Camera.getCamera() 方法。

设计摄像头应用程序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在编写需要连接到用户摄像头的应用程序时，需要在代码中考虑以下事项：

- 检查用户当前是否安装了摄像头。处理无摄像头可用时的情况。
- （只针对 Flash Player）检查用户是否明确允许访问其摄像头。出于安全原因，播放器会显示 “Flash Player 设置 ” 对话框，让用户选择允许还是拒绝对其摄像头的访问。这样可以防止 Flash Player 在未经用户许可的情况下连接到其摄像头并广播视频流。如果用户单击允许，则应用程序即可连接到用户的摄像头。如果用户单击拒绝，则应用程序将无法访问用户的摄像头。应用程序始终应适当地处理这两种情况。
- （只针对 AIR）检查您的应用程序所支持的设备配置文件是否支持 Camera 类。
- 移动浏览器不支持 Camera 类。
- 使用 GPU 呈现模式的移动 AIR 应用程序不支持 Camera 类。
- 在移动设备上，一次只能有一个摄像头处于活动状态。

更多帮助主题

Christophe Coenraets：多用户视频趣味游戏

Mark Doherty：Android Radar 应用程序 （源）

连接至用户的摄像头

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

连接到用户摄像头时，执行的第一步是通过创建一个类型为 Camera 的变量并将其初始化为静态 Camera.getCamera() 方法的返回值来创建一个新的 Camera 实例。

下一步是创建一个新的视频对象并向其附加 Camera 对象。

第三步是向显示列表中添加该视频对象。由于 Camera 类不会扩展 DisplayObject 类，它不能直接添加到显示列表中，因此需要执行第 2 步和第 3 步。若要显示摄像头捕获的视频，需要创建一个新的视频对象并调用 attachCamera() 方法。

以下代码演示这三个步骤：

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

注意，如果用户未安装摄像头，应用程序不会显示任何内容。

在实际情况下，您需要对应用程序执行其他步骤。有关详细信息，请参阅第 443 页的 “ 验证是否已安装摄像头 ” 和第 444 页的 “ 检测摄像头的访问权限 ”。

更多帮助主题

Lee Brimelow：如何访问 Android 设备上的摄像头

验证是否已安装摄像头

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在尝试对 Camera 实例使用任何方法或属性之前，您需要验证用户是否已安装了摄像头。检查用户是否已安装摄像头有两种方式：

- 检查静态 Camera.names 属性，该属性包含可用摄像头名称的数组。此数组通常具有一个或几个字符串，因为多数用户不太可能同时安装多个摄像头。以下代码演示如何检查 Camera.names 属性以查看用户是否具有可用的摄像头：

```
if (Camera.names.length > 0)
{
    trace("User has at least one camera installed.");
    var cam:Camera = Camera.getCamera(); // Get default camera.
}
else
{
    trace("User has no cameras installed.");
}
```

- 检查静态 Camera.getCamera() 方法的返回值。如果没有摄像头可用或未安装摄像头，则此方法将返回 null，否则返回对 Camera 对象的引用。以下代码演示如何检查 Camera.getCamera() 方法以查看用户是否具有可用的摄像头：

```
var cam:Camera = Camera.getCamera();
if (cam == null)
{
    trace("User has no cameras installed.");
}
else
{
    trace("User has at least 1 camera installed.");
}
```

由于 Camera 类未扩展 DisplayObject 类，因此无法使用 addChild() 方法将其直接添加到显示列表。为了显示摄像头捕获的视频，您需要创建一个新的 Video 对象并调用该 Video 实例的 attachCamera() 方法。

以下代码片断演示在存在摄像头的情况下如何附加摄像头；如果不存在摄像头，应用程序将不显示任何内容：

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

移动设备摄像头

在移动设备浏览器中， Flash Player 运行时不支持 Camera 类。

在移动设备上的 AIR 应用程序中，您可以访问该设备上的一个或多个摄像头。在移动设备上，可以使用前置和后置摄像头，但在任何给定时间只能显示一个摄像头输出。（附加第二个摄像头会断开与第一个摄像头的连接。）前置摄像头在 iOS 上采用水平镜像，但在 Android 上不采用水平镜像。

检测摄像头的访问权限

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 AIR 应用程序沙箱中，应用程序无需用户许可即可访问任何摄像头。但是，在 Android 上，应用程序必须在应用程序描述符中指定 Android CAMERA 权限。

在 Flash Player 可显示摄像头输出之前，用户必须明确允许 Flash Player 访问摄像头。在调用 attachCamera() 方法后，Flash Player 会显示 “Flash Player 设置 ” 对话框，提示用户允许或拒绝 Flash Player 访问摄像头和麦克风。如果用户单击 “ 允许 ” 按钮，Flash Player 将在舞台上的 Video 实例中显示摄像头输出。如果用户单击 “ 拒绝 ” 按钮，则 Flash Player 将无法连接到摄像头，且 Video 对象将不显示任何内容。

如果想要检测用户是否允许 Flash Player 访问其摄像头，可以侦听摄像头的 status 事件 (StatusEvent.STATUS)，如以下代码所示：

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
function statusHandler(event:StatusEvent):void
{
    // This event gets dispatched when the user clicks the "Allow" or "Deny"
    // button in the Flash Player Settings dialog box.
    trace(event.code); // "Camera.Muted" or "Camera.Unmuted"
}
```

一旦用户单击 “ 允许 ” 或 “ 拒绝 ”，即会调用 statusHandler() 函数。使用以下两种方法之一可以检测用户单击了哪个按钮：

- statusHandler() 函数的 event 参数包含一个 code 属性，其中含有字符串 “Camera.Muted” 或 “Camera.Unmuted”。如果该值为 “Camera.Muted”，表示用户单击了 “ 拒绝 ” 按钮，因此 Flash Player 无法访问摄像头。在下面的代码片断中您会看到此情况的一个示例：

  ```
  function statusHandler(event:StatusEvent):void
  {
      switch (event.code)
      {
          case "Camera.Muted":
              trace("User clicked Deny.");
              break;
          case "Camera.Unmuted":
              trace("User clicked Accept.");
              break;
      }
  }
  ```

- Camera 类包含一个名为 muted 的只读属性，它指明用户在 Flash Player“ 隐私 ” 面板中是拒绝访问摄像头 (true) 还是允许访问摄像头 (false)。在下面的代码片断中您会看到此情况的一个示例：

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

通过检查将要调度的 status 事件，您可以编写代码来处理用户接受或拒绝访问摄像头的操作并进行相应的清理。例如，如果用户单击“拒绝”按钮，则您可以向用户显示一条消息，说明他们如果想要参加视频聊天的话，需要单击“允许”；或者，您也可以确保删除显示列表中的 Video 对象以释放系统资源。

在 AIR 中，由于使用摄像头的权限不是动态的， Camera 对象不调度状态事件。

最优化摄像头视频品质

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下， Video 类的新实例为 320 像素宽乘以 240 像素高。为了最优化视频品质，应始终确保视频对象与 Camera 对象返回的视频具有相同的尺寸。使用 Camera 类的 width 和 height 属性，您可以获取 Camera 对象的宽度和高度，然后将该视频对象的 width 和 height 属性设置为与 Camera 对象的尺寸相符，也可以将 Camera 对象的宽度和高度传递给 Video 类的构造函数方法，如以下代码片断所示：

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

由于 getCamera() 方法返回对 Camera 对象的引用 （在没有可用摄像头时返回 null），因此，即使用户拒绝访问其摄像头，您也可以访问 Camera 对象的方法和属性。这样可以使用摄像头的本机高度和宽度设置视频实例的尺寸。

```
var vid:Video;
var cam:Camera = Camera.getCamera();

if (cam == null)
{
    trace("Unable to locate available cameras.");
}
else
{
    trace("Found camera: " + cam.name);
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("Unable to connect to active camera.");
    }
    else
    {
        // Resize Video object to match camera settings and
        // add the video to the display list.
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
    }
    // Remove the status event listener.
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}
```

有关全屏模式的信息，请参阅第 137 页的 “ 设置舞台属性 ”下的全屏模式部分。

监控摄像头状态

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Camera 类包含多个属性，这些属性允许您监视 Camera 对象的当前状态。例如，以下代码使用一个 Timer 对象和一个文本字段实例在显示列表上显示摄像头的若干属性：

```
var vid:Video;
var cam:Camera = Camera.getCamera();
var tf:TextField = new TextField();
tf.x = 300;
tf.autoSize = TextFieldAutoSize.LEFT;
addChild(tf);

if (cam != null)
{
    cam.addEventListener(StatusEvent.STATUS, statusHandler);
    vid = new Video();
    vid.attachCamera(cam);
}
function statusHandler(event:StatusEvent):void
{
    if (!cam.muted)
    {
        vid.width = cam.width;
        vid.height = cam.height;
        addChild(vid);
        t.start();
    }
    cam.removeEventListener(StatusEvent.STATUS, statusHandler);
}

var t:Timer = new Timer(100);
t.addEventListener(TimerEvent.TIMER, timerHandler);
function timerHandler(event:TimerEvent):void
{
    tf.text = "";
    tf.appendText("activityLevel: " + cam.activityLevel + "\n");
    tf.appendText("bandwidth: " + cam.bandwidth + "\n");
    tf.appendText("currentFPS: " + cam.currentFPS + "\n");
    tf.appendText("fps: " + cam.fps + "\n");
    tf.appendText("keyFrameInterval: " + cam.keyFrameInterval + "\n");
    tf.appendText("loopback: " + cam.loopback + "\n");
    tf.appendText("motionLevel: " + cam.motionLevel + "\n");
    tf.appendText("motionTimeout: " + cam.motionTimeout + "\n");
    tf.appendText("quality: " + cam.quality + "\n");
}
```

每 1/10 秒 （100 毫秒），即会调度一次 Timer 对象的 timer 事件，并且 timerHandler() 函数会更新显示列表上的文本字段。

第 27 章：使用数字权限管理

Flash Player 10.1 和更高版本， **Adobe AIR 1.5** 和更高版本

Adobe® Access™ 是一项内容保护解决方案。该解决方案通过提供对付费内容的无缝访问，使内容所有者、分销商和广告商实现了新的收入来源。发布者使用 Adobe Access 加密内容、创建策略并颁发许可证。Adobe Flash Player 和 Adobe AIR 合并为一个 DRM 库，即 Adobe Access 模块。此模块启用运行时，以便与 Adobe Access 许可证服务器通信并播放受保护的内容。因此，运行时完成受 Adobe Access 保护并由 Flash Media Server 分发的内容的生命周期。

使用 Adobe Access，内容提供者可以提供免费内容和付费内容。例如，某消费者要观看不穿插广告的电视节目。他可以注册并付费给内容发布者。然后，即可输入用户凭据来访问和播放不穿插广告的节目。

在另一个示例中，消费者要在旅途中无法访问 Internet 时脱机查看内容。 AIR 应用程序中支持此脱机工作流程。在注册并付费给内容发布者后，用户可以从发布者的网站访问并下载内容和关联的 AIR 应用程序。使用 AIR 应用程序，用户可以在允许的期限内脱机查看内容。该内容还可以与使用域的同一设备组中的其他设备共享 （**3.0** 中的新功能）。

Adobe Access 还支持匿名访问，不需要对用户进行身份验证。例如，发布者可以使用匿名访问来分发支持广告的内容。通过匿名访问，发布者还可以允许对当前内容进行指定天数的免费访问。内容提供者还可指定和限制播放其内容所需的播放器类型和版本。

用户尝试在 Flash Player 或 Adobe AIR 中播放受保护的内容时，您的应用程序必须调用 DRM API。DRM API 启动工作流程以播放受保护的内容。运行时通过 Adobe Access 模块与许可证服务器连接。如有必要，许可证服务器将对用户进行身份验证，并颁发许可证以允许播放受保护的内容。运行时将接收许可证并解密播放的内容。

此处介绍了如何启用应用程序以播放受 Adobe Access 保护的内容。无需了解如何使用 Adobe Access 加密内容或维护策略。但本文假设您正在与 Adobe Access 许可证服务器通信以对用户进行身份验证并检索许可证。还假设您正在设计一种应用程序，专门播放受 Adobe Access 保护的内容。

有关包括创建策略在内的 Adobe Access 的概述，请参阅 Adobe Access 随附的文档。

更多帮助主题

flash.net.drm 包

flash.net.NetConnection

flash.net.NetStream

了解受保护的内容工作流程

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

重要说明：Flash Player 11.5 和更高版本集成了 Adobe Access 模块，因此无需执行更新步骤 （调用 SystemUpdater.update(SystemUp aterType.DRM)）。其中包括下列浏览器和平台：

- Flash Player 11.5 ActiveX 控件，适用于除使用 Intel 处理器的 Windows 8 上的 Internet Explorer 之外所有平台
- Flash Player 11.5 插件，适用于所有浏览器
- Adobe AIR （桌面版和移动版）

这意味着在下列情况下，仍需要执行更新步骤：

- 使用 Intel 处理器的 Windows 8 上的 Internet Explorer

- Flash Player 11.4 和更低版本，在 Google Chrome 22 和更高版本 （针对所有平台）或 21 和更高版本 （针对 Windows）上除外

注：您仍可在安装 Flash Player 11.5 或更高版本的系统上安全调用 SystemUpdater.update(SystemUpdaterType.DRM)，但不会下载任何文件。

以下高级工作流程说明了应用程序如何检索并播放受保护的内容。该工作流程假设应用程序专门设计用于播放受 Adobe Access 保护的内容：

1 获取内容元数据。

2 处理对 Flash Player 的更新 （如果需要）。

3 检查许可证在本地是否可用。如果可用，则加载该许可证并转到步骤 7。否则，转到步骤 4。

4 检查是否需要进行身份验证。如果不需要，可以转到步骤 7。

5 如果需要进行身份验证，则从用户处获取身份验证凭据并传递给许可证服务器。

6 如果需要域注册，请加入域 （AIR 3.0 和更高版本）。

7 一旦身份验证成功，即可从服务器下载许可证。

8 播放内容。

如果未出错并且成功授权用户查看内容， NetStream 对象将调度 DRMStatusEvent 对象。应用程序将开始播放。DRMStatusEvent 对象保留相关凭证信息，这些信息用于标识用户的策略和权限。例如，将保留有关是否可以脱机使用内容或许可证何时到期的信息。应用程序可以使用此数据来通知用户其策略的状态。例如，应用程序可在状态栏上显示用户可以观看相应内容的剩余天数。

如果允许用户脱机访问，则将缓存凭证，并且将加密的内容下载到用户的计算机上。在许可证缓存期间定义的持续时间内可以访问该内容。事件中的 detail 属性包含 "DRM.voucherObtained"。应用程序决定内容的本地存储位置，以使内容可脱机使用。您还可以使用 DRMManager 类预加载凭证。

注：AIR 和 Flash Player 中都支持缓存和预加载凭证。不过，仅在 AIR 中支持下载和存储加密的内容。

应用程序负责明确处理错误事件。这些事件包括以下情况：用户输入有效凭据，但是保护加密内容的凭证限制对内容的访问。例如，如果未针对权限支付费用，已经过身份验证的用户将无法访问该内容。当同一发布者的两个注册成员试图共享仅其中一人付费的内容时，也可能发生这种情况。应用程序必须向用户通知该错误，并提供替代建议。典型的替代建议是提供有关如何注册查看权限并进行付费的说明。

详细的 API 工作流程

Flash Player 10.1 和更高版本， **AIR 2.0** 和更高版本

此工作流程提供了保护内容工作流程的更详细视图。该工作流程介绍了用于播放受 Adobe Access 保护的内容的特定 API。

1 使用 URLLoader 对象，加载受保护内容的元数据文件的字节。将此对象设置为变量，例如 metadata_bytes。

   由 Adobe Access 控制的所有内容都包含 Adobe Access 元数据。打包内容时，可同时将此元数据另存为一个单独的元数据文件 (.metadata)。有关详细信息，请参阅 Adobe Access 文档。

2 创建 DRMContentData 实例。将此代码置于 try-catch 块中：

   new DRMContentData(***metadata_bytes***)

   其中 metadata_bytes 是步骤 1 中获取的 URLLoader 对象。

3 （仅限 Flash Player）运行时检查 Adobe Access 模块。如果未找到，将引发 IllegalOperationError，并显示 DRMErrorEvent 错误代码 3344 或 DRMErrorEvent 错误代码 3343。

   要处理此错误，请使用 SystemUpdater API 下载 Adobe Access 模块。下载此模块后， SystemUpdater 对象将调度 COMPLETE 事件。调度此事件时，包括将返回到步骤 2 的此事件的事件侦听器。下面的代码演示了这些步骤：

```
flash.system.SystemUpdater.addEventListener(Event.COMPLETE, updateCompleteHandler);
flash.system.SystemUpdater.update(flash.system.SystemUpdaterType.DRM)

private function updateCompleteHandler (event:Event):void {
    /*redo step 2*/
    drmContentData = new DRMContentData(metadata_bytes);
}
```

如果必须更新播放器本身，则会调度状态事件。有关处理此事件的详细信息，请参阅第 463 页的 " 侦听更新事件 "。

注：在 AIR 应用程序中， AIR 安装程序将处理更新 Adobe Access 模块和所需的运行时更新。

4 创建侦听器以侦听从 DRMManager 对象调度的 DRMStatusEvent 和 DRMErrorEvent：

```
DRMManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
DRMManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
```

在 DRMStatusEvent 侦听器中，检查凭证是否有效 （不是 null）。在 DRMErrorEvent 侦听器中，处理 DRMErrorEvent。请参阅第 456 页的 " 使用 DRMStatusEvent 类 " 和第 460 页的 " 使用 DRMErrorEvent 类 "。

5 加载播放内容所需的凭证 （许可证）。

首先，尝试加载本地存储的许可证以播放内容：

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.LOCAL_ONLY)
```

加载完成后， DRMManager 对象将调度 DRMStatusEvent.DRM_Status。

6 如果 DRMVoucher 对象不是 null，则凭证有效。跳到步骤 13.

7 如果 DRMVoucher 对象为 null，请检查此内容的策略所需的身份验证方法。使用 DRMContentData.authenticationMethod 属性。

8 如果身份验证方法是 ANONYMOUS，请转到步骤 13。

9 如果身份验证方法是 USERNAME_AND_PASSWORD，您的应用程序必须提供一个允许用户输入凭据的机制。将这些凭据传递给许可证服务器以对用户进行身份验证：

```
DRMManager.authenticate(metadata.serverURL, metadata.domain, username, password)
```

如果身份验证失败， DRMManager 调度 DRMAuthenticationErrorEvent，如果身份验证成功，则调度 DRMAuthenticationCompleteEvent。为这些事件创建侦听器。

10 如果身份验证方法为 UNKNOWN，则必须使用自定义身份验证方法。在此情况下，内容提供商已安排使用带外方式完成身份验证，而不使用 ActionScript 3.0 API。自定义身份验证过程必须生成一个能够传递给 DRMManager.setAuthenticationToken() 方法的身份验证令牌。

11 如果身份验证失败，您的应用程序必须返回步骤 9。确保您的应用程序具有处理和限制重复的身份验证失败次数的机制。例如，经过三次尝试后，向用户显示一则消息，指示身份验证失败，无法播放内容。

12 要使用存储的令牌，而不是提示用户输入凭据，请使用 DRMManager.setAuthenticationToken() 方法设置该令牌。然后，从许可证服务器下载许可证并按照步骤 8 中的说明播放内容。

13 （可选）如果身份验证成功，您可以捕获身份验证令牌，该令牌是在内存中缓存的字节数组。获取此令牌，其属性为 DRMAuthenticationCompleteEvent.token。您可以存储并使用该身份验证令牌，以便用户不必为此内容重复输入凭据。许可证服务器确定身份验证令牌的有效期。

14 如果身份验证成功，则从许可证服务器下载许可证：

```
DRMManager.loadvoucher(drmContentData, LoadVoucherSetting.FORCE_REFRESH)
```

加载完成后， DRMManager 对象调度 DRMStatusEvent.DRM_STATUS。侦听此事件，在调度该事件时，您可以播放内容。

15 创建 NetStream 对象，然后调用其 play() 方法以播放视频：

```
stream = new NetStream(connection);
stream.addEventListener(DRMStatusEvent.DRM _STATUS, drmStatusHandler);
stream.addEventListener(DRMErrorEvent.DRM_ERROR, drmErrorHandler);
stream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
stream.client = new CustomClient();
video.attachNetStream(stream);
stream.play(videoURL);
```

DRMContentData 和会话对象

在创建 DRMContentData 时，它将用作引用 Flash Player DRM 模块的会话对象。接收此 DRMContentData 的所有 DRMManager API 都将使用该特定的 DRM 模块。不过，有 2 个 DRMManager API 不使用 DRMContentData。这些规则是：

1 authenticate()

2 setAuthenticationToken()

由于没有关联的 DRMContentData，调用这些 DRMManager API 将使用磁盘中最新的 DRM 模块。如果在应用程序的 DRM 工作流程中间发生 DRM 模块的更新，这可能会出现问题。请考虑以下情况：

1 应用程序创建一个 DRMContentData 对象 contentData1，该对象使用 AdobeCP1 作为 DRM 模块。

2 应用程序调用 DRMManager.authenticate(contentData1.serverURL,...) 方法。

3 应用程序调用 DRMManager.loadVoucher(contentData1, ...) 方法。

如果应用程序在到达第 2 步之前 DRM 模块发生更新，则 DRMManager.authenticate() 方法将结束使用 AdobeCP2 作为 DRM 模块进行身份验证。第 3 步中的 loadVoucher() 方法将失败，因为它仍然使用 AdobeCP1 作为 DRM 模块。如果有另一应用程序调用 DRM 模块更新，则可能会发生此更新。通过在启动应用程序时调用 DRM 模块更新可以避免出现此情况。

与 DRM 相关的事件

应用程序尝试播放保护的内容时，运行时会调度许多事件：

- DRMDeviceGroupErrorEvent （仅限 AIR），由 DRMManager 调度
- DRMAuthenticateEvent （仅限 AIR），由 NetStream 调度
- DRMAuthenticationCompleteEvent，由 DRMManager 调度
- DRMAuthenticationErrorEvent，由 DRMManager 调度
- DRMErrorEvent，由 NetStream 和 DRMManager 调度
- DRMStatusEvent，由 NetStream 和 DRMManager 调度
- StatusEvent
- NetStatusEvent。请参阅第 463 页的 “ 侦听更新事件 ”

要支持受 Adobe Access 保护的内容，请添加处理 DRM 事件的事件侦听器。

预加载用于脱机播放的凭证

Adobe AIR 1.5 和更高版本

您可以预加载播放受 Adobe Access 保护的内容所需的凭证 （许可证）。通过预加载的凭证，无论是否具有有效的 Internet 连接，用户都可以查看内容。（预加载过程本身需要 Internet 连接。）您可以使用 NetStream 类 preloadEmbeddedMetadata() 方法和 DRMManager 类预加载凭证。在 AIR 2.0 和更高版本中，您可以使用 DRMContentData 对象直接预加载凭证。此技术更可取，因为它允许您更新与内容独立的 DRMContentData 对象。（preloadEmbeddedData() 方法从内容中获取 DRMContentData。）

使用 DRMContentData

Adobe AIR 2.0 和更高版本

下列步骤介绍了使用 DRMContentData 对象为保护的媒体文件预加载凭证的工作流程。

1 获取打包内容的二进制元数据。如果使用的是 Adobe Access Java Reference 打包程序，则会自动生成带有 .metadata 扩展名的元数据文件。例如，可以使用 URLLoader 类下载此元数据。

2 创建一个 DRMContentData 对象，将此元数据传递给构造函数：

```
var drmData:DRMContentData = new DRMContentData( metadata );
```

3 其余步骤与第 448 页的 " 了解受保护的内容工作流程 " 中介绍的工作流程相同。

使用 preloadEmbeddedMetadata()

Adobe AIR 1.5 和更高版本

下列步骤介绍了使用 preloadEmbeddedMetadata() 为 DRM 保护的媒体文件预加载凭证的工作流程：

1 下载并存储媒体文件。（只能从本地存储的文件中预加载 DRM 元数据。）

2 创建 NetConnection 和 NetStream 对象，为 NetStream 客户端对象的 onDRMContentData() 和 onPlayStatus() 回调函数提供实现。

3 创建 NetStreamPlayOptions 对象，并将 stream 属性设置为本地媒体文件的 URL。

4 调用 NetStream preloadEmbeddedMetadata()，并传入 NetStreamPlayOptions 对象，从而标识要分析的媒体文件。

5 如果媒体文件包含 DRM 元数据，则调用 onDRMContentData() 回调函数。将元数据作为 DRMContentData 对象传递到此函数。

6 使用 DRMContentData 对象通过 DRMManager loadVoucher() 方法获取凭证。

   如果 DRMContentData 对象的 authenticationMethod 属性的值为 flash.net.drm.AuthenticationMethod.USERNAME_AND_PASSWORD，则在加载凭证之前对媒体权限服务器上的用户进行身份验证。可以将 DRMContentData 对象的 serverURL 和 domain 属性传递给 DRMManager authenticate() 方法，附带用户的凭据。

7 文件分析完毕后将调用 onPlayStatus() 回调函数。如果尚未调用 onDRMContentData() 函数，则文件不包含获取凭证所需的元数据。未调用还可能意味着 Adobe Access 不保护此文件。

以下 AIR 代码示例说明如何为本地媒体文件预加载凭证：

```
package
{
import flash.display.Sprite;
import flash.events.DRMAuthenticationCompleteEvent;
import flash.events.DRMAuthenticationErrorEvent;
import flash.events.DRMErrorEvent;
import flash.ev ents.DRMStatusEvent;
import flash.events.NetStatusEvent;
import flash.net.NetConnection;
import flash.net.NetStream;
import flash.net.NetStreamPlayOptions;
import flash.net.drm.AuthenticationMethod;
import flash.net.drm.DRMContentData;
import flash.net.drm.DRMManager;
import flash.net.drm.LoadVoucherSetting;
public class DRMPreloader extends Sprite
{
    private var videoURL:String = "app-storage:/video.flv";
    private var userName:String = "user";
    private var password:String = "password";
    private var preloadConnection:NetConnection;
    private var preloadStream:NetStream;
    private var drmManager:DRMManager = DRMManager.getDRMManager();
    private var drmContentData:DRMContentData;
    public function DRMPreloader():void {
        drmManager.addEventListener(
            DRMAuthenticationCompleteEvent.AUTHENTICATION_COMPLETE,
            onAuthenticationComplete);
        drmManager.addEventListener(DRMAuthenticationErrorEvent.AUTHENTICATION_ERROR,
            onAuthenticationError);
        drmManager.addEventListener(DRMStatusEvent.DRM_STATUS, onDRMStatus);
        drmManager.addEventListener(DRMErrorEvent.DRM_ERROR, onDRMError);
        preloadConnection = new NetConnection();
        preloadConnection.addEventListener(NetStatusEvent.NET_STATUS, onConnect);
        preloadConnection.connect(null);
    }

    private function onConnect( event:NetStatusEvent ):void
    {
        preloadMetadata();
    }
    private function preloadMetadata():void
    {
        preloadStream = new NetStream( preloadConnection );
        preloadStream.client = this;
        var options:NetStreamPlayOptions = new NetStreamPlayOptions();
        options.streamName = videoURL;
        preloadStream.preloadEmbeddedData( options );
    }
    public function onDRMContentData( drmMetadata:DRMContentData ):void
    {
        drmContentData = drmMetadata;
        if ( drmMetadata.authenticationMethod == AuthenticationMethod.USERNAME_AND_PASSWORD )
        {
            authenticateUser();
        }
        else
            {
                getVoucher();
            }
    }
    private function getVoucher():void
```

```
    {
        drmManager.loadVoucher( drmContentData, LoadVoucherSetting.ALLOW_SERVER );
    }

    private function authenticateUser():void
    {
        drmManager.authenticate( drmContentData.serverURL, drmContentData.domain, userName, password );
    }
    private function onAuthenticationError( event:DRMAuthenticationErrorEvent ):void
    {
        trace( "Authentication error: " + event.errorID + ", " + event.subErrorID );
    }

    private function onAuthenticationComplete( event:DRMAuthenticationCompleteEvent ):void
    {
        trace( "Authenticated to: " + event.serverURL + ", domain: " + event.domain );
        getVoucher();
    }
    private function onDRMStatus( event:DRMStatusEvent ):void
    {
        trace( "DRM Status: " + event.detail);
        trace("--Voucher allows offline playback = " + event.isAvailableOffline );
        trace("--Voucher already cached          = " + event.isLocal );
        trace("--Voucher required authentication = " + !event.isAnonymous );
    }
    private function onDRMError( event:DRMErrorEvent ):void
    {
        trace( "DRM error event: " + event.errorID + ", " + event.subErrorID + ", " + event.text );
    }
    public function onPlayStatus( info:Object ):void
    {
        preloadStream.close();
    }
}
}
```

NetStream 类中与 DRM 相关的成员和事件

Flash Player 10.1 和更高版本， **Adobe AIR 1.0** 和更高版本

NetStream 类提供 Flash Player 或 AIR 应用程序与 Flash Media Server 或本地文件系统之间的单向流连接。（NetStream 类还支持渐进式下载。） NetStream 对象是 NetConnection 对象中的一个通道。 NetStream 类调度四个与 DRM 相关的事件：

事件	说明
drmAuthenticate （仅限 AIR）	在 DRMAuthenticateEvent 类中定义。 NetStream 对象尝试播放保护的内容 （播放前需要用户凭据以进行身份验证）时调度此事件。 此事件的属性包括 header、usernamePrompt、passwordPrompt 和 urlPrompt，这些属性可用于获取和设置用户凭据。此事件将重复发生，直到 NetStream 对象获得有效的用户凭据。
drmError	在 DRMErrorEvent 类中定义，在 NetStream 对象尝试播放保护的内容并遇到与 DRM 相关的错误时调度。例如，当用户授权失败时，将调度 DRM 错误事件对象。如果用户没有购买查看内容的权限，可能发生此错误。如果内容提供商不支持查看应用程序，也可能发生此错误。
drmStatus	在 DRMStatusEvent 类中定义。在开始播放保护的内容 （如果已对用户进行身份验证并授权播放该内容）时调度此事件。 DRMStatusEvent 对象包含与凭证有关的信息。凭证信息包括是否可以脱机使用该内容或凭证何时过期而无法再查看该内容。
状态	在 events.StatusEvent 中定义，仅当应用程序尝试通过调用 NetStream.play() 方法播放保护的内容时调度。 status 代码属性的值为 “DRM.encryptedFLV”。

NetStream 类包含下列特定于 DRM 的方法，这些方法仅限于在 AIR 中使用：

方法	说明
resetDRMVouchers()	删除所有本地缓存的数字权限管理 (DRM) 凭证数据。应用程序必须重新下载凭证，用户才能访问加密内容。 例如，以下代码将删除缓存中的所有凭证： NetStream.resetDRMVouchers();
setDRMAuthenticationCredentials()	将一组身份验证凭据 （即用户名、密码和身份验证类型）传递给 NetStream 对象以进行身份验证。有效身份验证类型包括 "drm" 和 "proxy"。使用 "drm" 身份验证类型时，将针对 Adobe Access 对提供的凭据进行身份验证。使用 "proxy" 身份验证类型时，凭据将针对代理服务器进行身份验证且必须与代理服务器所需的凭据相匹配。例如，企业可能需要应用程序先针对代理服务器对用户进行身份验证后，用户才可以访问 Internet。使用代理选项可实现此类型的身份验证。除非使用匿名身份验证，否则在代理身份验证之后，用户必须仍针对 Adobe Access 进行身份验证才能获取凭证并播放内容。您可以再次使用 setDRMAuthenticationCredentials() 与 “drm” 选项一起针对 Adobe Access 进行身份验证。
preloadEmbeddedMetadata()	分析本地媒体文件，从而得到嵌入的元数据。找到与 DRM 相关的元数据后， AIR 将调用 onDRMContentData() 回调函数。

此外，在 AIR 中，作为调用 preloadEmbeddedMetaData() 方法的结果， NetStream 对象调用 onDRMContentData() 和 onPlayStatus() 回调函数。在媒体文件中遇到 DRM 元数据后，将调用 onDRMContentData() 函数。解析文件后，调用 onPlayStatus() 函数。在分配给 NetStream 实例的 client 对象上必须定义 onDRMContentData() 和 onPlayStatus() 函数。如果使用同一个 NetStream 对象预加载凭证并播放内容，则必须在开始播放前等待由 preloadEmbeddedMetaData() 生成的 onPlayStatus() 调用。

在以下 AIR 代码中，将设置用户名 （“administrator”）、密码 （“password”）和 “drm” 身份验证类型以对用户进行身份验证。 setDRMAuthenticationCredentials() 方法提供的凭据必须与内容提供者已知并接受的凭据相匹配。这些凭据与用户查看内容所需的用户凭据相同。此处不包含用于播放视频和确保已成功连接到视频流的代码。

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

使用 DRMStatusEvent 类

Flash Player 10.1、 **Adobe AIR 1.0** 和更高版本

开始成功播放受 Adobe Access 保护的内容时， NetStream 对象调度 DRMStatusEvent 对象。（成功暗示已验证许可证且用户已经过身份验证并获得了查看内容的授权）。如果允许匿名用户访问，则还会为其调度 DRMStatusEvent。将检查许可证以验证是否允许不需要身份验证的匿名用户访问并播放内容。出于各种原因，可能会拒绝匿名用户进行访问。例如，匿名用户在许可证过期时无法访问内容。

DRMStatusEvent 对象包含与许可证有关的信息。此类信息包括是否可以脱机使用该许可证或凭证何时过期而无法再查看该内容。应用程序可使用此数据来传输用户的策略状态及其权限。

DRMStatusEvent 属性

Flash Player 10.1、 **Adobe AIR 1.0** 和更高版本

DRMStatusEvent 类包括下列属性。有些属性在 AIR 1.0 以后的更高版本中可用。有关完整版本的信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》。

对于 Flash Player 10.1 中不支持的属性， DRMVoucher 类针对 Flash Player 提供了类似属性。

属性	说明
contentData	包含内容中嵌入的 DRM 元数据的 DRMContentData 对象。
detail （仅限 AIR）	说明状态事件上下文的字符串。在 DRM 1.0 中，唯一的有效值为 DRM.voucherObtained。
isAnonymous （仅限 AIR）	指示受 Adobe Access 保护的内容是否在不需要用户提供身份验证凭据的情况下可用，如果是，则为 true，否则为 false。值为 false 表示用户提供的用户名和密码必须与内容提供者已知并需要的用户名和密码相匹配。
isAvailableOffline （仅限 AIR）	指示受 Adobe Access 保护的内容是否可以脱机使用，如果是，则为 true，否则为 false。为了使数字保护的内容可脱机使用，必须将其凭证缓存到用户的本地计算机中。
isLocal	指示是否本地缓存播放内容所需的凭证。
offlineLeasePeriod （仅限 AIR）	可以脱机查看内容的剩余天数。

属性	说明
policies （仅限 AIR）	可能包含自定义 DRM 属性的自定义对象。
凭证	DRMVoucher。
voucherEndDate （仅限 AIR）	凭证到期且内容不再可观看的绝对日期。

创建 DRMStatusEvent 处理函数

Flash Player 10.1、 **Adobe AIR 1.0** 和更高版本

以下示例将创建一个事件处理函数，该事件处理函数可为启动了事件的 NetStream 对象输出 DRM 内容状态信息。将此事件处理函数添加到指向保护的内容的 NetStream 对象。

```
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
function drmStatusEventHandler(event:DRMStatusEvent):void
{
    trace(event);
}
```

使用 DRMAuthenticateEvent 类

Adobe AIR 1.0 和更高版本

当 NetStream 对象尝试播放保护的内容 （播放前需要用户凭据以进行身份验证）时调度 DRMAuthenticateEvent 对象。

DRMAuthenticateEvent 处理函数负责收集所需凭据 （用户名、密码和类型）并将这些值传递给 NetStream.setDRMAuthenticationCredentials() 方法以进行验证。每个 AIR 应用程序都必须提供用于获取用户凭据的某种机制。例如，应用程序可以为用户提供一个输入用户名和密码值的简单用户界面。另外，还提供一种处理和限制重复的身份验证尝试次数的机制。

DRMAuthenticateEvent 属性

Adobe AIR 1.0 和更高版本

DRMAuthenticateEvent 类包含以下属性：

属性	说明
authenticationType	指示提供的凭据是针对 Adobe Access （“drm”）进行身份验证，还是针对代理服务器 （“proxy”）进行身份验证。例如，如果要求用户在针对代理服务器进行身份验证后才能访问 Internet，使用 “proxy” 选项可使应用程序针对代理服务器进行身份验证。除非使用匿名身份验证，否则在代理身份验证之后，用户必须仍针对 Adobe Access 进行身份验证才能获取凭证并播放内容。您可以再次使用 setDRMAuthenticationcredentials() 与 “drm” 选项一起针对 Adobe Access 进行身份验证。
header	服务器提供的加密内容文件标头。它包含有关加密内容的上下文的信息。 此标头字符串可以传递到 Flash 应用程序，以支持该应用程序构造一个 “ 用户名 - 密码 ” 对话框。标头字符串可用作该对话框的说明。例如，该标头可以是 “ 请键入您的用户名和密码 ”。
netstream	启动此事件的 NetStream 对象。
passwordPrompt	服务器提供的密码凭据提示。该字符串可以包括所需密码类型的说明。
urlPrompt	服务器提供的 URL 字符串提示。该字符串可以提供要将用户名和密码发送到的位置。
usernamePrompt	服务器提供的用户名凭据提示。该字符串可以包括所需用户名类型的说明。例如，内容提供者可能要求用电子邮件地址作为用户名。

上述字符串仅由 FMRMS 服务器提供。 Adobe Access Server 不使用这些字符串。

创建 DRMAuthenticateEvent 处理函数

Adobe AIR 1.0 和更高版本

以下示例将创建一个事件处理函数，该事件处理函数将一组硬编码身份验证凭据传递给启动了事件的 NetStream 对象。（此处不包含用于播放视频和确保已成功连接到视频流的代码。）

```
var connection:NetConnection = new NetConnection();
connection.connect(null);

var videoStream:NetStream = new NetStream(connection);

videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                             drmAuthenticateEventHandler)

private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
{
    videoStream.setDRMAuthenticationCredentials("administrator", "password", "drm");
}
```

创建用于获取用户凭据的界面

Adobe AIR 1.0 和更高版本

在保护的内容要求进行用户身份验证的情况下， AIR 应用程序通常必须通过用户界面检索用户的身份验证凭据。

以下是一个有关用于获取用户凭据的简单用户界面的 Flex 示例。它由一个包含两个 TextInput 对象 （分别针对用户名和密码凭据）的面板对象组成。该面板还包含一个启动 credentials() 方法的按钮。

```
<mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute" title="Login">
    <mx:TextInput x="110" y="46" id="uName"/>
    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
    <mx:Text x="35" y="48" text="Username:"/>
    <mx:Text x="35" y="78" text="Password:"/>
    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
</mx:Panel>
```

credentials() 方法是用户定义的方法，可将用户名和密码值传递给 setDRMAuthenticationCredentials() 方法。一旦传递这些值，credentials() 方法将重置 TextInput 对象的值。

```
<mx:Script>
    <![CDATA[
        public function credentials():void
        {
            videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
            uName.text = "";
            pWord.text = "";
        }
    ]]>
</mx:Script>
```

实现此类型的简单界面的一种方法是包含该窗格，将其作为新状态的一部分。新状态来自引发 DRMAuthenticateEvent 对象时的基本状态。以下示例包含其源属性指向保护的 FLV 文件的 VideoDisplay 对象。在此例中，对 credentials() 方法进行了修改，以便该方法也将应用程序返回到基本状态。此方法在传递用户凭据并重置 TextInput 对象值后将执行此操作。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    width="800"
    height="500"
    title="DRM FLV Player"
    creationComplete="initApp()" >

    <mx:states>
        <mx:State name="LOGIN">
            <mx:AddChild position="lastChild">
                    <mx:Panel x="236.5" y="113" width="325" height="204" layout="absolute"
                            title="Login">
                    <mx:TextInput x="110" y="46" id="uName"/>
                    <mx:TextInput x="110" y="76" id="pWord" displayAsPassword="true"/>
                    <mx:Text x="35" y="48" text="Username:"/>
                    <mx:Text x="35" y="78" text="Password:"/>
                    <mx:Button x="120" y="115" label="Login" click="credentials()"/>
                    <mx:Button x="193" y="115" label="Reset" click="uName.text='';
                            pWord.text='';"/>
                </mx:Panel>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:Script>
        <![CDATA[
                import flash.events.DRMAuthenticateEvent;
            private function initApp():void
            {
                videoStream.addEventListener(DRMAuthenticateEvent.DRM_AUTHENTICATE,
                                             drmAuthenticateEventHandler);
            }
```

```
            public function credentials():void
            {
                videoStream.setDRMAuthenticationCredentials(uName, pWord, "drm");
                uName.text = "";
                pWord.text = "";
                currentState='';
            }

            private function drmAuthenticateEventHandler(event:DRMAuthenticateEvent):void
            {
                currentState='LOGIN';
            }
        ]]>
    </mx:Script>

    <mx:VideoDisplay id="video" x="50" y="25" width="700" height="350"
        autoPlay="true"
        bufferTime="10.0"
        source="http://www.example.com/flv/Video.flv" />
</mx:WindowedApplication>
```

使用 DRMErrorEvent 类

Flash Player 10.1 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe Flash Player 和 Adobe AIR 在 NetStream 对象尝试播放保护的内容过程中遇到与 DRM 有关的错误时调度 DRMErrorEvent 对象。如果 AIR 应用程序中的用户凭据无效，则 DRMAuthenticateEvent 对象将重复调度，直到用户输入有效的凭据，或者应用程序拒绝进一步尝试。应用程序负责侦听任何其他 DRM 错误事件，以检测、识别和处理与 DRM 相关的错误。

即使使用有效的用户凭据，内容凭证的条款仍可能阻止用户查看加密内容。例如，当用户尝试查看未经授权的应用程序中的内容时，可能会被拒绝访问。未经授权的应用程序是加密内容的发布者未验证的应用程序。此时，将调度 DRMErrorEvent 对象。

如果内容已损坏或者应用程序的版本与凭证指定的版本不匹配，则也会引发错误事件。应用程序必须提供适当的机制来处理错误。

DRMErrorEvent 属性

Flash Player 10.1 和更高版本， **Adobe AIR 1.0** 和更高版本

有关错误的完整列表，请参阅《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的运行时错误代码。与 DRM 相关的错误从错误 3300 开始。

创建 DRMErrorEvent 处理函数

Flash Player 10.1 和更高版本， **Adobe AIR 1.0** 和更高版本

以下示例将针对启动了事件的 NetStream 对象创建一个事件处理函数。当 NetStream 在尝试播放保护的内容的过程中遇到错误时，调用该事件处理函数。通常，当应用程序遇到错误时，会执行任意次数的清理任务。应用程序随后会通知用户该错误，并提供用于解决该问题的选项。

```
private function drmErrorEventHandler(event:DRMErrorEvent):void
{
    trace(event.toString());
}
```

使用 DRMManager 类

Flash Player 10.1 和更高版本， **Adobe AIR 1.5** 和更高版本

使用 DRMManager 类管理应用程序中的凭证和媒体权限服务器会话。

凭证管理 （仅限 **AIR**）

只要用户播放保护的内容，运行时就会获取并缓存查看内容所需的许可证。如果应用程序在本地保存文件，并且许可证允许脱机播放，则用户可以查看 AIR 应用程序中的内容。即使与媒体权限服务器的连接不可用，仍可进行此类本地脱机播放。使用 DRMManager 和 NetStream preloadEmbeddedMetadata() 方法，您可以预缓存凭证。应用程序不必获取查看内容所需的许可证。例如，应用程序可以下载媒体文件，然后在用户仍联机时获取凭证。

若要预加载凭证，请使用 NetStream preloadEmbeddedMetadata() 方法获取 DRMContentData 对象。 DRMContentData 对象包含可以提供许可证的媒体权限服务器的 URL 和域，并介绍是否需要对用户进行身份验证。借助此信息，可以调用 DRMManager loadVoucher() 方法来获取和缓存凭证。第 451 页的 " 预加载用于脱机播放的凭证 " 中更详细地介绍了预加载凭证所采用的工作流程。

会话管理

还可以使用 DRMManager 验证用户对媒体权限服务器的身份以及管理持久性会话。

调用 DRMManager authenticate() 方法与媒体权限服务器建立会话。成功完成身份验证后， DRMManager 将调度 DRMAuthenticationCompleteEvent 对象。此对象包含会话令牌。可以保存此令牌供将来建立会话，以使用户不必提供其帐户凭据。将令牌传递给 setAuthenticationToken() 方法以建立通过身份验证的新会话。（生成令牌的服务器的设置决定令牌的有效期限和其他属性。 AIR 应用程序代码不应解释令牌数据结构，因为该结构在以后的 AIR 更新中可能会发生变化。）

可以将身份验证令牌转移到其他计算机。若要保护令牌，可以将其存储在 AIR 加密本地存储区中。有关详细信息，请参阅第 610 页的 " 加密的本地存储区 "。

DRMStatus 事件

Flash Player 10.1 和更高版本， **Adobe AIR 1.5** 和更高版本

成功完成调用 loadVoucher() 方法后， DRMManager 将调度 DRMStatusEvent 对象。

如果获取了凭证，则事件对象的 detail 属性 （仅限 AIR）的值将为 "DRM.voucherObtained"，并且 voucher 属性包含 DRMVoucher 对象。

如果未获取凭证，则 detail 属性 （仅限 AIR）的值仍为 "DRM.voucherObtained"，但是 voucher 属性为 null。例如，如果使用 localOnly 的 LoadVoucherSetting，并且没有本地缓存的凭证，则无法获取凭证。

如果 loadVoucher() 调用未成功完成 （可能由于身份验证或通信错误），则 DRMManager 改为调度 DRMErrorEvent 或 DRMAuthenticationErrorEvent 对象。

DRMAuthenticationComplete 事件

Flash Player 10.1 和更高版本， **Adobe AIR 1.5** 和更高版本

通过调用 authenticate() 方法成功验证用户的身份后， DRMManager 将调度 DRMAuthenticationCompleteEvent 对象。

DRMAuthenticationCompleteEvent 对象包含一个可重用的令牌，该令牌可用于在应用程序会话中保持用户的身份验证。将此标记传递给 DRMManager setAuthenticationToken() 方法以重新建立会话。（令牌创建者设置有效期限等令牌属性。Adobe 不提供用于检查标记属性的 API。）

DRMAuthenticationError 事件

Flash Player 10.1 和更高版本， **Adobe AIR 1.5** 和更高版本

用户无法通过调用 authenticate() 或 setAuthenticationToken() 方法成功验证用户的身份时， DRMManager 将调度 DRMAuthenticationErrorEvent 对象。

使用 DRMContentData 类

Flash Player 10.1 和更高版本， **Adobe AIR 1.5** 和更高版本

DRMContentData 对象包含受 Adobe Access 保护的内容的元数据属性。DRMContentData 属性包含获取用于查看内容的许可证凭证所需的信息。您可以使用 DRMContentData 类获取与内容关联的元数据文件，如第 449 页的 “ 详细的 API 工作流程 ”中所述。

有关详细信息，请参阅 《用于 Adobe Flash Platform 的 ActionScript 3.0 参考》中的 DRMContentData 类。

更新 Flash Player 以支持 Adobe Access

Flash Player 10.1 和更高版本

重要说明：Flash Player 11.5 和更高版本集成了 Adobe Access 模块，因此无需执行更新步骤 （调用 SystemUpdater.update(SystemUp aterType.DRM)）。其中包括下列浏览器和平台：

- Flash Player 11.5 ActiveX 控件，适用于除 Windows 8 上的 Internet Explorer 之外的所有平台
- Flash Player 11.5 插件，适用于所有浏览器
- Adobe AIR （桌面版和移动版）

这意味着在下列情况下，仍需要执行更新步骤：

- Windows 8 上的 Internet Explorer
- Flash Player 11.4 和更低版本，在 Google Chrome 22 和更高版本 （针对所有平台）或 21 和更高版本 （针对 Windows）上除外

注：您仍可在安装 Flash Player 11.5 或更高版本的系统上安全调用 SystemUpdater.update(SystemUpdaterType.DRM)，但不会下载任何文件。

要支持 Adobe Access，Flash Player 需要 Adobe Access 模块。Flash Player 尝试播放保护的内容时，运行时指示是否必须下载该模块或 Flash Player 的新版本。这样，如果需要， Flash Player 会允许 SWF 开发人员选择不进行更新。

大多数情况下， SWF 开发人员需要更新到所需的 Adobe Access 模块或播放器才可以播放受保护的内容。要进行更新，您可以使用 SystemUpdater API 获取最新版本的 Adobe Access 模块或 Flash Player。

SystemUpdater API 每次只允许进行一个更新。错误代码 2202 指示正在当前运行时实例或其他实例中进行更新。例如，如果正在 Internet Explorer 中对 Flash Player 实例进行更新，则无法对在 FireFox 中运行的 Flash Player 实例进行更新。

桌面平台仅支持 SystemUpdater API。

注：对于 Flash Player 10.1 以前的版本，请使用早期播放器版本中支持的更新机制 （从 www.adobe.com 或 ExpressInstall 手动下载并安装）。另外， AIR 安装程序处理 Adobe Access 的必要更新且不支持 SystemUpdater API。

侦听更新事件

Flash Player 10.1 和更高版本

需要对 Adobe Access 模块进行更新时， NetStream 对象调度代码值为 DRM.UpdateNeeded 的 NetStatusEvent。此值指示 NetStream 对象无法播放当前安装了任何 Adobe Access 模块的保护流。侦听此事件并调用以下代码：

```
SystemUpdater.update(flash.system.SystemUpdaterType.DRM)
```

此代码更新安装在播放器中的 Adobe Access 模块。不需要经过用户同意即可对此模块进行更新。

如果找不到 Adobe Access 模块，则会引发错误。请参阅第 449 页的 “ 详细的 API 工作流程 ” 中的步骤 3。

注：如果在 10.1 之前的播放器版本中的加密流上调用 play()，则会调度其代码值为 NetStream.Play.StreamNotFound 的 NetStatusEvent。对于早期版本的播放器，请使用这些播放器支持的更新机制 （从 www.adobe.com 或 ExpressInstall 手动下载并安装）。

需要对播放器本身进行更新时， SystemUpdater 对象将调度 StatusEvent，该事件的代码值 DRM.UpdateNeededButIncompatible 已调度。对于播放器更新，必须经过用户同意。在您的应用程序中，为用户提供一个用于表示同意并启动播放器更新的界面。侦听 StatusEvent 事件并调用以下代码：

```
SystemUpdater.update(flash.system.SystemUpdaterType.SYSTEM);
```

此代码启动播放器更新。

SystemUpdater 类的其他事件都记录在用于 Adobe Flash Platform 的 ActionScript 3.0 参考中。

播放器更新完成后，将用户重定向到开始进行更新的页面。已下载 Adobe Access 模块，可以开始播放该流。

带外许可证

Flash Player 11 和更高版本， **Adobe AIR 3.0** 和更高版本

通过使用 storeVoucher 方法将凭证 （许可证）存储在磁盘和内存中，还可以采用带外方式获取许可证 （无需联系 Adobe Access License Server）。

要在 Flash Player 和 AIR 中播放加密的视频，相应的运行时需要获取该视频的 DRM 凭证。 DRM 凭证包含该视频的解密密钥，而该凭证由客户部署的 Adobe Access License Server 生成。

Flash Player/AIR 运行时通常通过向视频的 DRM 元数据 （DRMContentData 类）中指示的 Adobe Access License Server 发送凭证请求来获取此凭证。 Flash/AIR 应用程序通过调用 DRMManager.loadVoucher() 方法可以触发此许可证请求。如果磁盘或内存中没有用于加密的视频的许可证， Flash Player/AIR 运行时在播放该内容时将自动请求许可证。在任一情况下， Flash/AIR 应用程序的性能都会受到与 Adobe Access License Server 通信的影响。

DRMManager.storeVoucher() 允许 Flash/AIR 应用程序将其从带外获取的 DRM 凭证发送给 Flash Player/AIR 运行时。随后，运行时可以跳过许可证请求过程，并使用转发的凭证播放加密的视频。 DRM 凭证仍需要由 Adobe Access License Server 生成，然后才可以采用带外方式获取。不过，您可以选择在任何 HTTP 服务器上托管凭证，而不是在面向公众的 Adobe Access License Server 上托管。

DRMManager.storeVoucher() 也用于支持在多个设备之间共享 DRM 凭证。在 Adobe Access 3.0 中，此功能称为 " 域支持 "。如果您的部署支持此使用案例，则可以使用 DRMManager.addToDeviceGroup() 方法向一个设备组注册多个计算机。如果某个计算机有一个用于给定内容的有效域绑定凭证，则 AIR 应用程序可以使用 DRMVoucher.toByteArray() 方法提取序列化 DRM 凭证，在其他计算机上，您可以使用 DRMManager.storeVoucher() 方法导入凭证。

设备注册

DRM 凭证绑定到最终用户的计算机。因此，Flash /AIR 应用程序需要一个对应于用户计算机的唯一 ID 来引用正确的序列化 DRM 凭证对象。以下情况描述了一个设备注册过程：

假定您已经执行了以下任务：

- 您已经设置了 Adobe Access Server SDK。
- 您已经设置了用于获取预生成许可证的 HTTP 服务器。
- 您已经创建了用于查看保护内容的 Flash 应用程序。

设备注册阶段涉及以下操作：

1. Flash 应用程序创建一个随机生成的 ID。
2. Flash 应用程序调用 DRMManager.authenticate() 方法。此应用程序必须在身份验证请求中包含随机生成的 ID。例如，在用户名字段中包含 ID。
3. 第 2 步中所述的操作将导致 Adobe Access 向客户的服务器发送身份验证请求。此请求包含设备证书。
   a. 服务器从请求中提取设备证书和生成的 ID 并将其存储。
   b. 客户子系统为此设备证书预生成许可证，将其存储，并通过将其与生成的 ID 关联来授予对它们的访问权限。
4. 服务器使用 " 成功 " 消息来响应该请求。
5. Flash 应用程序将生成的 ID 本地存储在本地共享对象 (LSO) 中。

在完成设备注册之后， Flash 应用程序采用与在上一方案中使用设备 ID 相同的方式使用生成的 ID：

1. Flash 应用程序将尝试在 LSO 中查找生成的 ID。
2. 如果找到生成的 ID， Flash 应用程序将在下载预生成的许可证时使用生成的 ID。 Flash 应用程序将使用 DRMManager.storeVoucher() 方法将许可证发送给 Adobe Access 客户端进行使用。
3. 如果未找到生成的 ID， Flash 应用程序将执行设备注册过程。

出厂重置

当设备的用户调用出厂重置选项时，将清除设备证书。要继续播放保护的内容， Flash 应用程序需要再次执行设备注册过程。如果 Flash 应用程序发送的预生成许可证已过期，Adobe Access 客户端将拒绝该许可证，因为该许可证已针对较旧的设备 ID 加密。

域支持

Flash Player 11 和更高版本， **Adobe AIR 3.0** 和更高版本

如果内容元数据指定需要域注册，则 AIR 应用程序可以通过调用 API 来加入设备组。此操作将触发域注册请求，并将该请求发送到域服务器。一旦将许可证颁发给设备组，即可导出该许可证并与加入设备组中的其他设备共享。

随后，设备组信息将用于 DRMContentData 的 VoucherAccessInfo 对象，进而使用该对象提供相关信息，以便成功检索和使用凭证。

使用域支持播放加密的内容

要使用 Adobe Access 播放加密的内容，请执行以下步骤：

1 使用 VoucherAccessInfo.deviceGroup 检查是否需要进行设备组注册。

2 是否需要进行身份验证：

   a 使用 DeviceGroupInfo.authenticationMethod 属性查明是否需要身份验证。

   b 如果需要身份验证，请通过执行下列步骤之一对用户进行身份验证：

      - 获取用户的用户名和密码。调用 DRMManager.authenticate(deviceGroup.serverURL, deviceGroup.domain, username, password)。
      - 获取缓存的 / 预生成的身份验证令牌并调用 DRMManager.setAuthenticationToken()。

   c 调用 DRMManager.addToDeviceGroup()。

3 通过执行下列任务之一获取内容的凭证：

   a 使用 DRMManager.loadVoucher() 方法。

   b 从在同一设备组中注册的其他设备获取凭证。通过 DRMManager.storeVoucher() 方法将凭证提供给 DRMManager。

4 使用 NetStream.play() 方法播放加密的内容。

要导出该内容的许可证，任何设备在从 Adobe Access License Server 获取许可证之后都可以使用 DRMVoucher.toByteArray() 方法提供许可证的原始字节。内容提供商通常会限制设备组中的设备数量。如果限制数量已满，在注册当前设备之前，您可能需要对未使用的设备调用 DRMManager.removeFromDeviceGroup() 方法。

许可证预览

Flash 应用程序可以发送许可证预览请求，即该应用程序可以在请求用户购买内容之前执行预览操作，以确定用户的计算机实际上是否满足播放所需的全部条件。许可证预览是指客户端能够预览许可证 （查看许可证允许哪些权限）而不是能够预览内容（在决定购买之前查看部分内容）。对每台计算机都是唯一的一些参数包括：可用输出及其保护状态、可用的运行时 /DRM 版本以及 DRM 客户端安全级别。许可证预览模式允许运行时 /DRM 客户端测试许可证服务器业务逻辑，并将信息返回用户，以便用户可以做出明智的决定。因此，客户端可以查看有效许可证是什么样的，但不会实际收到密钥来解密内容。对许可证预览的支持是可选的，并且仅在您实现使用此功能的自定义应用程序时才为必需项。

提交内容

Adobe Access 无法获知内容的交付机制，因为 Flash Player 是在网络层外部提取受保护的内容并仅向 Adobe Access 子系统提供该内容。因此，内容可以通过 HTTP、 HTTP 动态流、 RTMP 或 RTMPE 交付。

不过，由于 Adobe Access 需要受保护内容的元数据 （通常为 ".metadata" 形式的文件）才能获取许可证来解密内容，因此您可能会遇到一些问题。具体来说，使用 RTMP/RTMPE 协议，只有 FLV 和 F4V 数据可以通过 Flash Media Server (FMS) 交付给客户端。因此，客户端必须通过其他方法检索元数据 Blob。解决此问题的一种办法是将元数据托管在 HTTP Web 服务器上，并实施客户端视频播放器来检索适当的元数据 （具体取决于播放的内容）。

```
private function getMetadata():void{

    extrapolated-path-to-metadata = "http://metadatas.mywebserver.com/" + videoname;
    var urlRequest : URLRequest = new URLRequest(extrapolated-path-to-the-metadata + ".metadata");
    var urlStream : URLStream = new URLStream();
    urlStream.addEventListener(Event.COMPLETE, handleMetadata);
    urlStream.addEventListener(IOErrorEvent.NETWORK_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.IO_ERROR, handleIOError);
    urlStream.addEventListener(IOErrorEvent.VERIFY_ERROR, handleIOError);
    try{
        urlStream.load(urlRequest);
     }catch(se:SecurityError){
        videoLog.text += se.toString() + "\n";
     }catch(e:Error){
        videoLog.text += e.toString() + "\n";
     }
}
```

Open Source Media Framework

开源媒体框架 (Open Source Media Framework, OSMF) 是一个基于 ActionScript 的框架，可在您创建自己的富媒体体验时提供充分的灵活性和控制能力。有关 OSMF 的详细信息，请访问 OSMF 开发人员站点。

播放受保护内容的工作流程

1 创建 MediaPlayer 实例。

```
player = new MediaPlayer();
```

2 向播放器注册 MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE 事件。如果内容受 DRM 保护，将调度此事件。

```
player.addEventListener(MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE, onDRMCapabilityChange);
```

3 在事件处理函数中，获取 DRMTrait 实例。 DRMTrait 是一个接口，通过该接口可以调用与 DRM 相关的方法，如 authenticate()。在加载受 DRM 保护的内容时， OSMF 将执行 DRM 验证操作并调度状态事件。将 DRMEvent.DRM_STATE_CHANGE 事件处理函数添加到 DRMTrait。

```
private function onDRMCapabilityChange
    (event :MediaPlayerCapabilityChangeEvent) :void
        {
            if (event.type == MediaPlayerCapabilityChangeEvent.HAS_DRM_CHANGE
                && event.enabled)
            {
                drmTrait = player.media.getTrait(MediaTraitType.DRM) as DRMTrait;
                drmTrait.addEventListener
                    (DRMEvent.DRM_STATE_CHANGE, onDRMStateChange);
            }
        }
```

4 在 onDRMStateChange() 方法中处理 DRM 事件。

```
private function onDRMStateChange(event :DRMEvent) :void
{
    trace ( "DRMState: ",event.drmState);
    switch(event.drmState)
    {
        case DRMState.AUTHENTICATION_NEEDED:
            // Identity-based content
            var authPopup :AuthWindow = AuthWindow.create(_parentWin);
            authPopup.serverURL = event.serverURL;
            authPopup.addEventListener("dismiss", function () :void {
                trace ("Authentication dismissed");
                if(_drmTrait != null)
                {
                    //Ignore authentication. Just
                    //try to acquire a license.
                    _drmTrait.authenticate(null, null);
                }
            });
            authPopup.addEventListener("authenticate",
                            function (event :AuthWindowEvent) :void {
                if(_drmTrait != null)
                {
                    _drmTrait.authenticate(event.username, event.password);
                }
            });
            authPopup.show();
            break;
        case DRMState.AUTHENTICATING:
            //Display any authentication message.
            trace("Authenticating...");
            break;
        case DRMState.AUTHENTICATION_COMPLETE:
            // Start to retrieve voucher and playback.
            // You can display the voucher information at this point.
            if(event.token)
            // You just received the authentication token.
            {
                trace("Authentication success. Token: \n", event.token);
            }
            else
            // You have got the voucher.
            {
                trace("DRM License:");
                trace("Playback window period: ",
                    !isNaN(event.period) ? event.period == 0 ?
                    "<unlimited>" : event.period : "<none>");
```

```
                    trace("Playback window end date: ",
                        event.endDate != null ? event.endDate : "<none>");
                    trace("Playback window start date: ",
                        event.startDate != null ? event.startDate : "<none>");
                }
                break;
            case DRMState.AUTHENTICATION_ERROR:
                trace ("DRM Error:", event.mediaError.errorID +
                    "[" + DRMErrorEventRef.getDRMErrorMnemonic
                    (event.mediaError.errorID) + "]");
                //Stop everything.
                player.media = null;
                break;
            case DRMState.DRM_SYSTEM_UPDATING:
                Logger.log("Downloading DRM module...");
                break;
            case DRMState.UNINITIALIZED:
                break;
        }
    }
```

第 28 章：在 AIR 中添加 PDF 内容

Adobe AIR 1.0 和更高版本

Adobe® AIR® 中运行的应用程序不仅可以呈现 SWF 和 HTML 内容，而且还能呈现 PDF 内容。 AIR 应用程序使用 HTMLLoader 类、WebKit 引擎和 Adobe® Reader® 浏览器插件来呈现 PDF 内容。在 AIR 应用程序中，PDF 内容可以沿应用程序的全高和全宽进行拉伸，也可以作为界面的一部分。 Adobe Reader 浏览器插件控制 AIR 应用程序中的 PDF 文件显示。对 Reader 工具栏界面 （例如控件的位置、定位和可见性）的修改仍然存在于对 AIR 应用程序和浏览器中的 PDF 文件的后续查看中。

重要说明：要在 AIR 中呈现 PDF 内容，用户必须安装 Adobe Reader 或 Adobe® Acrobat® 版本 8.1 或更高版本。

检测 PDF 功能

Adobe AIR 1.0 和更高版本

如果用户没有 Adobe Reader 或 Adobe Acrobat 8.1 或更高版本，则无法在 AIR 应用程序中显示 PDF 内容。若要检测用户是否能够呈现 PDF 内容，请首先检查 HTMLLoader.pdfCapability 属性。此属性设置为 HTMLPDFCapability 类的以下常量之一：

常量	说明
HTMLPDFCapability.STATUS_OK	已检测到足够高的 Adobe Reader 版本（8.1 或更高版本），可以将 PDF 内容加载到 HTMLLoader 对象中。
HTMLPDFCapability.ERROR_INSTALLED_READER_NOT_FOUND	未检测到任何 Adobe Reader 版本。 HTMLLoader 对象无法显示 PDF 内容。
HTMLPDFCapability.ERROR_INSTALLED_READER_TOO_OLD	已检测到 Adobe Reader，但版本太旧。 HTMLLoader 对象无法显示 PDF 内容。
HTMLPDFCapability.ERROR_PREFERRED_READER_TOO_OLD	检测到的 Adobe Reader 版本足够高（8.1 或更高版本），但为处理 PDF 内容所安装的 Adobe Reader 版本早于 Reader 8.1。HTMLLoader 对象无法显示 PDF 内容。

在 Windows 上，如果用户的系统上运行的是 Adobe Acrobat 或 Adobe Reader 版本 7.x 或更高版本，即使安装了支持加载 PDF 的最新版本，也会使用当前运行的版本。在这种情况下，如果 pdfCapability 属性的值是 HTMLPDFCapability.STATUS_OK，则当 AIR 应用程序尝试加载 PDF 内容时，较旧版本的 Acrobat 或 Reader 会显示警报（并且 AIR 应用程序中不会引发异常）。如果您的最终用户有可能遇到这种情况，则可以考虑向他们提供说明，告知他们在运行应用程序的同时关闭 Acrobat。如果 PDF 内容在可以接受的时间范围内未加载，则您可能希望显示这些说明。

在 Linux 中， AIR 在由用户导出的 PATH （如果其包含 acroread 命令）中和 /opt/Adobe/Reader 目录中查找 Adobe Reader。

以下代码检测用户能否在 AIR 应用程序中显示 PDF 内容。如果用户无法显示 PDF，代码会跟踪对应于 HTMLPDFCapability 错误对象的错误代码：

```
if(HTMLLoader.pdfCapability == HTMLPDFCapability.STATUS_OK)
{
    trace("PDF content can be displayed");
}
else
{
    trace("PDF cannot be displayed. Error code:", HTMLLoader.pdfCapability);
}
```

加载 PDF 内容

Adobe AIR 1.0 和更高版本

可以通过创建 HTMLLoader 实例、设置其尺寸以及加载 PDF 的路径，将 PDF 添加到 AIR 应用程序。

以下示例从外部站点加载 PDF。将 URLRequest 替换为可用外部 PDF 的路径。

```
var request:URLRequest = new URLRequest("http://www.example.com/test.pdf");
pdf = new HTMLLoader();
pdf.height = 800;
pdf.width = 600;
pdf.load(request);
container.addChild(pdf);
```

还可以从文件 URL 和特定于 AIR 的 URL 方案（例如 app 和 app-storage）加载内容。例如，以下代码可加载应用程序目录的 PDFs 子目录中的 test.pdf 文件：

app:/js_api_reference.pdf

有关 AIR URL 方案的详细信息，请参阅第 698 页的 “URI 方案 ”。

编写 PDF 内容的脚本

Adobe AIR 1.0 和更高版本

可以像在浏览器的网页中那样，使用 JavaScript 控制 PDF 内容。

针对 Acrobat 的 JavaScript 扩展提供了以下功能（当然还包括其他功能）：

- 控制页面导航和缩放
- 处理文档中的表单
- 控制多媒体事件

有关 Adobe Acrobat 的 JavaScript 扩展的详细信息，请访问 Adobe Acrobat 开发人员联盟：http://www.adobe.com/devnet/acrobat/javascript.html。

HTML-PDF 通信基础知识

Adobe AIR 1.0 和更高版本

HTML 页中的 JavaScript 可以通过调用表示 PDF 内容的 DOM 对象的 postMessage() 方法来向 PDF 内容中的 JavaScript 发送消息。例如，请看以下嵌入的 PDF 内容：

```
<object id="PDFObj" data="test.pdf" type="application/pdf" width="100%" height="100%"/>
```

包含 HTML 内容中的以下 JavaScript 代码向 PDF 文件中的 JavaScript 发送消息：

```
pdfObject = document.getElementById("PDFObj");
pdfObject.postMessage(["testMsg", "hello"]);
```

PDF 文件可以包含 JavaScript 以便接收此消息。在某些上下文 （包括文档级、文件夹级、页级、字段级和批级上下文）中，可以向 PDF 文件添加 JavaScript 代码。此处仅讨论文档级上下文，这种上下文在打开 PDF 文档时会对计算出的脚本进行定义。

PDF 文件可以向 hostContainer 对象添加 messageHandler 属性。messageHandler 属性是用于定义处理函数以便响应消息的一种对象。例如，以下代码定义了一个函数，用于处理 PDF 文件从主机容器 （嵌入 PDF 文件的 HTML 内容）接收到的消息：

```
this.hostContainer.messageHandler = {onMessage: myOnMessage};

function myOnMessage(aMessage)
{
    if(aMessage[0] == "testMsg")
    {
        app.alert("Test message: " + aMessage[1]);
    }
    else
    {
        app.alert("Error");
    }
}
```

HTML 页中的 JavaScript 代码可以调用页面中包含的 PDF 对象的 postMessage() 方法。通过调用此方法，可向 PDF 文件中的文档级 JavaScript 发送消息 ("Hello from HTML")：

```
<html>
    <head>
    <title>PDF Test</title>
    <script>
        function init()
        {
            pdfObject = document.getElementById("PDFObj");
            try {
                 pdfObject.postMessage(["alert", "Hello from HTML"]);
            }
            catch (e)
            {
                alert( "Error: \n name = " + e.name + "\n message = " + e.message );
            }
        }
    </script>
    </head>
    <body onload='init()'>
        <object
            id="PDFObj"
            data="test.pdf"
            type="application/pdf"
            width="100%" height="100%"/>
    </body>
</html>
```

有关更高级示例，以及有关使用 Acrobat 8 向 PDF 文件添加 JavaScript 的信息，请参阅在 Adobe AIR 中跨脚本访问 PDF 内容。

通过 ActionScript 编写 PDF 内容的脚本

Adobe AIR 1.0 和更高版本

ActionScript 代码（在 SWF 内容中）不能与 PDF 内容中的 JavaScript 直接通信。但是，ActionScript 可与加载 PDF 内容的 HTMLLoader 对象所加载的 HTML 页中的 JavaScript 通信，并且该 JavaScript 代码可与加载的 PDF 文件中的 JavaScript 通信。有关详细信息，请参阅第 839 页的 " 在 AIR 中进行 HTML 和 JavaScript 编程 "。

对 AIR 中的 PDF 内容的已知限制

Adobe AIR 1.0 和更高版本

Adobe AIR 中的 PDF 内容具有以下限制：

- 在透明窗口 （NativeWindow 对象）中 （transparent 属性设置为 true），不显示 PDF 内容。
- PDF 文件的显示顺序与 AIR 应用程序中的其他显示对象不同。尽管根据 HTML 显示顺序对 PDF 内容进行了正确剪辑，但在 AIR 应用程序的显示顺序中，它通常位于内容的上方。
- 如果更改包含 PDF 文档的 HTMLLoader 对象的某些视觉属性，则 PDF 文档将不可见。这些属性包括 filters、 alpha、rotation 和 scaling 属性。更改这些属性会使 PDF 内容不可见，直到重置这些属性为止。如果更改包含 HTMLLoader 对象的显示对象容器的这些属性， PDF 内容也会不可见。
- 只有在将包含 PDF 内容的 NativeWindow 对象的舞台对象的 scaleMode 属性设置为 StageScaleMode.NO_SCALE 时，PDF 内容才可见。将该属性设置为任何其他值时， PDF 内容均不可见。
- 单击指向 PDF 文件中内容的链接会更新 PDF 内容的滚动位置。单击指向 PDF 文件外部内容的链接会重定向包含 PDF 的 HTMLLoader 对象 （即使链接的目标是新窗口）。
- 在 AIR 中， PDF 注释工作流不起作用。

第 29 章：用户交互的基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可使用 ActionScript 3.0 来创建交互性以响应用户活动。请注意，本节假定您已熟悉了 ActionScript 3.0 事件模型。有关详细信息，请参阅第 106 页的 " 处理事件 "。

捕获用户输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用户交互 （无论是通过键盘、鼠标、摄像头还是这些设备的组合）是交互性的基础。在 ActionScript 3.0 中，识别和响应用户交互主要涉及事件侦听。

InteractiveObject 类是 DisplayObject 类的一个子类，它提供了处理用户交互所需的事件和功能的通用结构。您无法直接创建 InteractiveObject 类的实例。而是由显示对象 （如 SimpleButton、 Sprite、 TextField 以及各种 Flash 创作工具和 Flex 组件）从此类中继承其用户交互模型，因而它们共享同一个通用结构。这意味着，您为处理从 InteractiveObject 派生的一个对象中的用户交互而编写的代码以及学会的方法适用于所有其他对象。

重要概念和术语

在继续阅读本章内容之前，一定要先熟悉以下重要的用户交互术语：

字符代码 表示当前字符集中的字符的数字代码 （与在键盘上按的键关联）。例如，尽管 "D" 和 "d" 是由美国英语键盘上的同一个键创建的，但它们具有不同的字符代码。

上下文菜单 当用户右键单击或使用特定的键盘鼠标组合时显示的菜单。上下文菜单命令通常直接应用于已单击的内容。例如，某个图像的上下文菜单可能包含用于在单独窗口中显示该图像以及下载该图像的命令。

焦点 指示所选的元素处于活动状态，而且该元素是键盘或鼠标交互的目标。

键代码 与键盘上的实际键对应的数字代码。

更多帮助主题

InteractiveObject

Keyboard

Mouse

ContextMenu

管理焦点

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

交互式对象可以按编程方式或通过用户动作来获得焦点。另外，如果将 tabEnabled 属性设置为 true，用户可通过按 Tab 将焦点从一个对象传递到另一个对象。请注意，默认情况下， tabEnabled 值为 false，但以下情况除外：

- 对于 SimpleButton 对象，值为 true。

- 对于输入文本字段，该值为 true。
- 对于 buttonMode 设置为 true 的 Sprite 或 MovieClip 对象，该值为 true。

在上述各种情况下，都可以为 FocusEvent.FOCUS_IN 或 FocusEvent.FOCUS_OUT 添加侦听器，以便在焦点更改时提供其他行为。这对文本字段和表单尤其有用，但也可以用于 sprite、影片剪辑或从 InteractiveObject 类进行继承的任何对象。下面的示例说明了如何使用 Tab 键启用焦点循环切换，以及如何响应后续的焦点事件。在本例中，每个正方形在收到焦点时将改变颜色。

注：Flash Professional 使用键盘快捷键来管理焦点；因此，若要正确模拟焦点管理，应在浏览器或 AIR 中测试 SWF 文件，而不是在 Flash 中进行测试。

```
var rows:uint = 10;
var cols:uint = 10;
var rowSpacing:uint = 25;
var colSpacing:uint = 25;
var i:uint;
var j:uint;
for (i = 0; i < rows; i++)
{
    for (j = 0; j < cols; j++)
    {
        createSquare(j * colSpacing, i * rowSpacing, (i * cols) + j);
    }
}

function createSquare(startX:Number, startY:Number, tabNumber:uint):void
{
    var square:Sprite = new Sprite();
    square.graphics.beginFill(0x000000);
    square.graphics.drawRect(0, 0, colSpacing, rowSpacing);
    square.graphics.endFill();
    square.x = startX;
    square.y = startY;
    square.tabEnabled = true;
    square.tabIndex = tabNumber;
    square.addEventListener(FocusEvent.FOCUS_IN, changeColor);
    addChild(square);
}
function changeColor(event:FocusEvent):void
{
    event.target.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    // Generate random values for the red, green, and blue color channels.
    var red:Number = (Math.random() * 512) - 255;
    var green:Number = (Math.random() * 512) - 255;
    var blue:Number = (Math.random() * 512) - 255;

    // Create and return a ColorTransform object with the random colors.
    return new ColorTransform(1, 1, 1, 1, red, green, blue, 0);
}
```

了解输入类型

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

Flash Player 10.1 和 Adobe AIR 2 版本引入了测试运行时环境是否支持特定输入类型的功能。您可以使用 ActionScript 测试设备当前是否部署了运行时：

- 支持笔针或手指输入 （或根本没有触摸屏输入）。
- 为用户提供虚拟或物理键盘 （或根本没有键盘）。
- 显示光标 （如果不显示，则与悬停在对象上方的光标有关的功能不起作用）。

输入发现 ActionScript API 包括：

- flash.system.Capabilities.touchscreenType 属性：运行时提供的一个值，指示当前环境支持哪种输入类型。
- flash.system.TouchscreenType 类：Capabilities.touchscreenType 属性的枚举值常量的类。
- flash.ui.Mouse.supportsCursor 属性：在运行时提供的值，指示永久光标是否可用。
- flash.ui.Keyboard.physicalKeyboardType 属性：在运行时提供的值，指示整个物理键盘可用、仅数字键盘可用还是键盘根本不可用。
- flash.ui.KeyboardType 类：flash.ui.Keyboard.physicalKeyboardType 属性的枚举值常量的类。
- flash.ui.Keyboard.hasVirtualKeyboard 属性：在运行时提供的值，指示是否为用户提供了虚拟键盘 （代替物理键盘，或除物理键盘外额外提供）。

通过输入发现 API，您可以利用用户的设备功能，或在这些功能不可用时提供备用功能。这些 API 对于开发移动应用程序和启用触摸的应用程序特别有用。例如，如果您的移动设备的界面带有用于笔针的小按钮，则可以为使用手指触摸进行输入的用户提供带有较大按钮的替代界面。以下代码用于如下应用程序：具有 createStylusUI() 函数，该函数可分配一组适合笔针交互的用户界面元素。另一个函数称为 createTouchUI()，可分配另一组适合手指交互的用户界面元素：

```
if(Capabilities.touchscreenType == TouchscreenType.STYLUS ){
    //Construct the user interface using small buttons for a stylus
    //and allow more screen space for other visual content
    createStylusUI();
} else if(Capabilities.touchscreenType = TouchscreenType.FINGER){
    //Construct the user interface using larger buttons
    //to capture a larger point of contact with the device
    createTouchUI();
}
```

为不同的输入环境开发应用程序时，考虑如下兼容性图表：

环境	supportsCursor	touchscreenType == FINGER	touchscreenType == STYLUS	touchscreenType == NONE
传统桌面	true	false	false	true

环境	supportsCursor	touchscreenType == FINGER	touchscreenType == STYLUS	touchscreenType == NONE
电容式触摸屏设备 （平板电脑、 PDA 和检测轻微触摸的电话，例如 Apple iPhone 或 Palm Pre）	false	true	false	false
电阻式触摸屏设备 （平板电脑、 PDA 和检测准确的高压接触的电话，例如 HTC Fuze）	false	false	true	false
非触摸屏设备 （运行应用程序但不带有检测接触的屏幕的功能电话和设备）	false	false	false	true

注： 不同的设备平台可以支持输入类型的多种组合。使用此图表作为常规指南。

第 30 章：键盘输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可捕获和响应键盘的输入并可操控 IME，使用户可以键入多字节语言的非 ASCII 文本字符。请注意，本节假定您已熟悉了 ActionScript 3.0 事件模型。有关详细信息，请参阅第 106 页的 " 处理事件 "。

有关确定在运行时支持何种类型的键盘 （例如物理键盘、虚拟键盘、字母数字键盘或者 12 个键的数字键盘）的信息，请参阅第 475 页的 " 了解输入类型 "。

通过输入法编辑器 (IME)，用户可以使用标准键盘键入复杂的字符和符号。您可以使用 IME 类使用户能够在您的应用程序中利用其系统 IME。

更多帮助主题

flash.events.KeyboardEvent

flash.system.IME

捕获键盘输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

从 InteractiveObject 类继承交互模型的显示对象可以使用事件侦听器来响应键盘事件。例如，您可以将事件侦听器放在舞台上以侦听并响应键盘输入。在以下代码中，事件侦听器捕获一个按键，并显示键名和键控代码属性：

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " + event.charCode +
")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

有些键 （如 Ctrl 键）虽然没有字型表示形式，但也能生成事件。

在上面的代码示例中，键盘事件侦听器捕获了整个舞台的键盘输入。也可以为舞台上的特定显示对象编写事件侦听器；当对象具有焦点时将触发该事件侦听器。

在以下示例中，仅当用户在 TextField 实例内键入内容时，才会在 " 输出 " 面板中反映键击。按下 Shift 键可暂时将 TextField 的边框颜色更改为红色。

此代码假定舞台上有一个名为 tf 的 TextField 实例。

```
tf.border = true;
tf.type = "input";
tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
tf.addEventListener(KeyboardEvent.KEY_UP,reportKeyUp);

function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (key code: " + event.keyCode + "
character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT) tf.borderColor = 0xFF0000;
}

function reportKeyUp(event:KeyboardEvent):void
{
    trace("Key Released: " + String.fromCharCode(event.charCode) + " (key code: " + event.keyCode + "
character code: " + event.charCode + ")");
    if (event.keyCode == Keyboard.SHIFT)
    {
        tf.borderColor = 0x000000;
    }
}
```

TextField 类还会报告 textInput 事件，当用户输入文本时，您可以侦听该事件。有关详细信息，请参阅第 317 页的 " 捕获文本输入 "。

注：在 AIR 运行时中，可以取消键盘事件。在 Flash Player 运行时中，无法取消键盘事件。

键控代码和字符代码

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以访问键盘事件的 keyCode 和 charCode 属性，以确定按下了哪个键，然后触发其他动作。keyCode 属性为数值，与键盘上的某个键的值相对应。 charCode 属性是该键在当前字符集中的数值。（默认字符集是 UTF-8，它支持 ASCII。）

键控代码值与字符值之间的主要区别是：键控代码值表示键盘上的特定键 （数字键盘上的 1 与最上面一排键中的 1 不同，但生成的 "1" 键与生成 "!" 的键是同一个键），字符值表示特定字符 （字符 R 与 r 是不同的）。

注：有关 ASCII 中的键和字符代码值之间的映射，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 flash.ui.Keyboard 类。

键与其键控代码之间的映射取决于设备和操作系统。因此，不应使用键映射来触发动作，而应使用 Keyboard 类提供的预定义常量值来引用相应的 keyCode 属性。例如，不要使用 Shift 的键映射，而应使用 Keyboard.SHIFT 常量 （如上面的代码范例中所示）。

KeyboardEvent 优先级

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

与其他事件一样，键盘事件序列是由显示对象的层次决定的，而不是由在代码中分配 addEventListener() 方法的顺序决定的。

例如，假定您将名为 tf 的文本字段放在名为 container 的影片剪辑内，并在这两个实例中添加键盘事件的事件侦听器，如下例所示：

```
container.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);
container.tf.border = true;
container.tf.type = "input";
container.tf.addEventListener(KeyboardEvent.KEY_DOWN,reportKeyDown);

function reportKeyDown(event:KeyboardEvent):void
{
    trace(event.currentTarget.name + " hears key press: " + String.fromCharCode(event.charCode) + " (key
code: " + event.keyCode + " character code: " + event.charCode + ")");
}
```

由于文本字段及其父容器上都有侦听器，因此，将为 TextField 内的每次键击调用两次 reportKeyDown() 函数。请注意，对于每次按键操作，文本字段在 container 影片剪辑调度事件之前调度事件。

操作系统和 Web 浏览器先于 Adobe Flash Player 或 AIR 处理键盘事件。例如，在 Microsoft Internet Explorer 中按 Ctrl+W 将先关闭浏览器窗口，然后再由包含的任何 SWF 文件调度键盘事件。

使用 IME 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过使用 IME 类，您可以在 Flash Player 或 Adobe AIR 中运用操作系统的 IME。

使用 ActionScript 可以确定以下内容：

- 用户的计算机上是否安装了 IME (Capabilities.hasIME)
- 用户计算机上是否启用了 IME (IME.enabled)
- 当前 IME 使用的转换模式 (IME.conversionMode)

可以使用特定 IME 上下文关联输入文本字段。在输入字段之间切换时，还可以在平假名 （日文）、全角数字、半角数字、直接输入等之间切换 IME。

利用 IME，用户可键入多字节语言 （例如中文、日文和韩文）的非 ASCII 文本字符。

有关使用 IME 的详细信息，请参阅要为其开发应用程序的操作系统的文档。要获取其他资源，请参阅以下 Web 站点：

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/library/mac/navigation/
- http://www.java.sun.com/

注：如果用户计算机上 IME 未处于活动状态，则对 IME 方法或属性 （除 Capabilities.hasIME 之外）的调用将会失败。一旦手动激活 IME，随后对 IME 方法和属性的 ActionScript 调用即会正常运行。例如，如果使用日文 IME，则必须在调用任何 IME 方法或属性之前将它激活。

查看是否已安装并启用了 IME

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

调用任何 IME 方法或属性之前，总应检查用户计算机上当前是否已安装并启用 IME。以下代码说明了在调用任何方法之前如何检查用户是否安装并启用了 IME：

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled. Please enable your IME and try again.");
    }
}
else
{
    trace("IME is not installed. Please install an IME and try again.");
}
```

上面的代码首先使用 Capabilities.hasIME 属性检查用户是否安装了 IME。如果此属性设置为 true，则代码使用 IME.enabled 属性检查当前是否启用了用户的 IME。

确定当前启用的是哪种 IME 转换模式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

构建多语言应用程序时，您可能需要确定用户当前启用的是哪种 IME 转换模式。以下代码说明了如何检查用户是否安装了 IME，以及在安装 IME 的情况下当前启用的是哪种 IME 转换模式：

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
            tf.text = "Current conversion mode is Japananese Hiragana.";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_FULL:
            tf.text = "Current conversion mode is Japanese Katakana (full-width).";
            break;
        case IMEConversionMode.JAPANESE_KATAKANA_HALF:
            tf.text = "Current conversion mode is Japanese Katakana (half-width).";
            break;
        case IMEConversionMode.KOREAN:
            tf.text = "Current conversion mode is Korean.";
            break;
        default:
            tf.text = "Current conversion mode is " + IME.conversionMode + ".";
            break;
    }
}
else
{
    tf.text = "Please install an IME and try again.";
}
```

上面的代码首先检查用户是否安装了 IME。接下来，该代码通过将 IME.conversionMode 属性与 IMEConversionMode 类中的每个常量进行比较，检查当前 IME 使用的是哪种转换模式。

设置 IME 转换模式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

更改用户的 IME 的转换模式时，您需要确保将代码封装在 try..catch 块中，因为使用 conversionMode 属性设置转换模式时，如果 IME 无法设置该转换模式，则可能会引发错误。下面的代码演示如何使用 try..catch 块 （在设置 IME.conversionMode 属性时）：

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
    try
    {
        IME.enabled = true;
        IME.conversionMode = IMEConversionMode.KOREAN;
        statusText.text = "Conversion mode is " + IME.conversionMode + ".";
    }
    catch (error:Error)
    {
        statusText.text = "Unable to set conversion mode.\n" + error.message;
    }
}
```

上面的代码先创建一个文本字段，该字段用于向用户显示状态消息。接下来，如果已安装 IME，该代码会启用 IME 并将转换模式设置为 “ 韩文 ”。如果用户计算机上未安装韩文 IME，则 Flash Player 或 AIR 将引发错误，该错误由 try..catch 块中来进行处理。 try..catch 块会在先前创建的文本字段中显示该错误消息。

为特定文本字段禁用 IME

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在某些情况下，最好在用户键入字符时禁用用户的 IME。例如，如果有一个文本字段只接受数字输入，您可能不想让 IME 出现并减缓数据输入的速度。

下面的示例演示如何侦听 FocusEvent.FOCUS_IN 和 FocusEvent.FOCUS_OUT 事件并相应地禁用用户的 IME：

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = false;
    }
}
function focusOutHandler(event:FocusEvent):void
{
    if (Capabilities.hasIME)
    {
        IME.enabled = true;
    }
}
```

此示例创建两个输入文本字段 phoneTxt 和 nameTxt，然后为 phoneTxt 文本字段添加两个事件侦听器。当用户将焦点设置为 phoneTxt 文本字段时，将调度 FocusEvent.FOCUS_IN 事件并禁用 IME。当 phoneTxt 文本字段失去焦点时，将调度 FocusEvent.FOCUS_OUT 事件以重新启用 IME。

侦听 IME 合成事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

设置合成字符串时会调度 IME 合成事件。例如，如果用户启用了 IME 并键入日文字符串，在用户选择合成字符串时，即会调度 IMEEvent.IME_COMPOSITION 事件。为了侦听 IMEEvent.IME_COMPOSITION 事件，您需要在 System 类的静态 ime 属性中添加一个事件侦听器 (flash.system.System.ime.addEventListener(...))，如下面的示例所示：

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
    IME.enabled = true;
    try
    {
        IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
    }
    catch (error:Error)
    {
        outputTxt.text = "Unable to change IME.";
    }
    System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

上面的代码创建两个文本字段，并将其添加到显示列表中。第一个文本字段 inputTxt 是一个输入文本字段，用户可以在其中输入日文文本。第二个文本字段 outputTxt 是一个动态文本字段，用于向用户显示错误消息，或回显用户在 inputTxt 文本字段中键入的日文字符串。

虚拟键盘

Flash Player 10.2 和更高版本， **AIR 2.6** 和更高版本

移动设备 （如手机和平板电脑）通常提供一种虚拟的软件键盘而不是物理键盘。使用 Flash API 中的类可以执行以下操作：

- 检测虚拟键盘何时开启和关闭。
- 阻止键盘开启。
- 确定被虚拟键盘覆盖的舞台区域。
- 创建获取焦点时开启键盘的交互对象。（iOS 上的 AIR 应用程序不支持此功能）
- （仅限 AIR）禁用自动平移行为，以便应用程序可以修改自己的显示以适应键盘。

控制虚拟键盘的行为

当用户在某个文本字段或特别配置的交互式对象内点击时，运行时会自动打开虚拟键盘。当该键盘打开时，运行时会遵从本机平台惯例来平移应用程序内容以及调整其大小，以便所有用户在键入文本的过程中可以看到这些文本。

当键盘打开时，具有焦点的对象会逐一调度以下事件：

softKeyboardActivating 事件 — 在键盘开始上浮到舞台上方前的一刹那调度该事件。如果您调用已调度事件对象的 preventDefault() 方法，则虚拟键盘不会打开。

softKeyboardActivate 事件 — 在完成 softKeyboardActivating 事件处理后调度该事件。当具有焦点的对象调度此事件时，Stage 对象的 softKeyboardRect 属性已更新，以反映被虚拟键盘遮住的舞台区域。无法取消此事件。

注：如果键盘更改大小，例如，当用户更改键盘类型时，具有焦点的对象会再次调度 softKeyboardActivate 事件。

softKeyboardDeactivate 事件 — 当虚拟键盘因任意原因关闭时调度该事件。无法取消此事件。

下面的示例在舞台上添加两个 TextField 对象。当您点击位于上方的 TextField 时，该字段会阻止键盘浮出；如果键盘已浮出，则关闭键盘。位于下方的 TextField 演示默认的行为。此示例报告由这两个文本字段同时调度的软键盘事件。

```
package
{
import flash.display.Sprite;
import flash.text.TextField;
import flash.text.TextFieldType;
import flash.events.SoftKeyboardEvent;
public class SoftKeyboardEventExample extends Sprite
{
    private var tf1:TextField = new TextField();
    private var tf2:TextField = new TextField();

    public function SoftKeyboardEventExample()
    {
        tf1.width = this.stage.stageWidth;
        tf1.type = TextFieldType.INPUT;
        tf1.border = true;
        this.addChild( tf1 );

        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, preventSoftKe    yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, preventSoftKe    yboard );
        tf1.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, preventSoftKeyboard );

        tf2.border = true;
        tf2.type = TextFieldType.INPUT;
        tf2.width = this.stage.stageWidth;
```

```
        tf2.y = tf1.y + tf1.height + 30;
        this.addChild( tf2 );

        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATING, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, allowSoftKeyboard );
        tf2.addEventListener( SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, allowSoftKeyboard);
    }

    private function preventSoftKeyboard( event:SoftKeyboardEvent ):void
    {
            event.preventDefault();
            this.stage.focus = null; //close the keyboard, if raised
            trace( "tf1 dispatched: " + event.type + " -- " + event.triggerType );
    }
    private function allowSoftKeyboard( event:SoftKeyboardEvent )    :void
    {
            trace( "tf2 dispatched: " + event.type + " -- " + event.triggerType );
    }
}
}
```

为交互式对象添加虚拟键盘支持

Flash Player 10.2 和更高版本， **AIR 2.6** 和更高版本 （但在 **iOS** 上不受支持）

通常情况下，仅在点击 TextField 对象时才会打开虚拟键盘。可以配置 InteractiveObject 类的一个实例，以便在其获得焦点时打开虚拟键盘。

若要配置 InteractiveObject 实例以打开软键盘，请将其 needsSoftKeyboard 属性设置为 true。每当对象分配到 stage focus 属性时，将自动打开软键盘。此外，还可以通过调用 InteractiveObject 的 requestSoftKeyboard() 方法来开启键盘。

下面的示例演示了如何对 InteractiveObject 编程，使之作为文本输入字段。示例中的 TextInput 类设置 needsSoftKeyboard 属性，以便在需要时开启键盘。然后，该对象侦听 keyDown 事件并将键入的字符插入字段。

该示例使用 Flash 文本引擎附加和显示键入的任何文本，并处理一些重要事件。为简单起见，该示例未实现完整功能的文本字段。

```
package  {
    import flash.geom.Rectangle;
    import flash.display.Sprite;
    import flash.text.engine.TextElement;
    import flash.text.engine.TextBlock;
    import flash.events.MouseEvent;
    import flash.events.FocusEvent;
    import flash.events.KeyboardEvent;
    import flash.text.engine.TextLine;
    import flash.text.engine.ElementFormat;
    import flash.events.Event;

    public class TextInput extends Sprite
    {

        public var text:String = " ";
        public  var textSize:Number = 24;
        public var textColor:uint = 0x000000;
        private var _bounds:Rectangle = new Rectangle( 0, 0, 100, textSize );
        private var textElement: TextElement;
        private var textBlock:TextBlock = new  TextBlock();

        public function TextInput( text:String = "" )
        {
```

```
            this.text = text;
            this.scrollRect = _bounds;
            this.focusRect= false;

            //Enable keyboard support
            this.needsSoftKeyboard = true;
            this.addEventListener(MouseEvent.MOUSE_DOWN, onSelect);
            this.addEventListener(FocusEvent.FOCUS_IN, onFocusIn);
            this.addEventListener(FocusEvent.FOCUS_OUT, onFocusOut);

            //Setup text engine
            textElement = new TextElement( text, new ElementFormat( null, textSize, textColor ) );
            textBlock.content = textElement;
            var firstLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8 );
            firstLine.x = 4;
            firstLine.y = 4 + firstLine.totalHeight;
            this.addChild( firstLine );

        }

        private function onSelect( event:MouseEvent ):void
        {
            stage.focus = this;
        }
        private function onFocusIn( event:FocusEvent ):void
        {
            this.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onFocusOut( event:FocusEvent ):void
        {
            this.removeEventListener( KeyboardEvent.KEY_UP, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            textElement.replaceText( textElement.text.length, textElement.text.length,
String.fromCharCode( event.charCode ) );
            updateText();
        }
        public function set bounds( newBounds:Rectangle ):void
        {
            _bounds = newBounds.clone();
            drawBackground();
            updateText();
            this.scrollRect = _bounds;

            //force update to focus rect, if needed
            if( this.stage!= null && this.focusRect && this.stage.focus == this )
                this.stage.focus = this;
        }

        private function updateText():void
        {
            //clear text lines
            while( this.numChildren > 0 ) this.removeChildAt( 0 );

            //and recreate them
            var textLine:TextLine = textBlock.createTextLine( null, _bounds.width - 8);
            while ( textLine)
            {
                textLine.x = 4;
```

```
                if( textLine.previousLine != null )
                {
                    textLine.y = textLine.previousLine.y +
                                 textLine.previousLine.totalHeight + 2;
                }
                            else
                {
                    textLine.y = 4 + textLine.totalHeight;
                }
                this.addChild(textLine);
                textLine = textBlock.createTextLine(textLine, _bounds.width - 8 );
            }
        }

        private function drawBackground():void
        {
            //draw background and border for the field
            this.graphics.clear();
            this.graphics.beginFill( 0xededed );
            this.graphics.lineStyle( 1, 0x000000 );
            this.graphics.drawRect( _bounds.x + 2, _bounds.y + 2, _bounds.width - 4, _bounds.height - 4);
            this.graphics.endFill();
        }
    }
}
```

下面的主应用程序类演示了当键盘开启或设备方向更改时如何使用 TextInput 类和管理应用程序布局。主类创建一个 TextInput 对象并设置其边界以填充舞台。当软键盘开启或舞台大小更改时，该类调整 TextInput 对象的大小。该类侦听 TextInput 对象的 soft keyboard 事件以及舞台的 resize 事件。无论事件的原因如何，应用程序都会确定舞台的可见区域，并调整输入控件的大小以填充该区域。当然，在实际应用程序中，需要更复杂的布局算法。

```
package  {

    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.geom.Rectangle;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;

    public class CustomTextField extends MovieClip {

        private var customField:TextInput = new TextInput("Input text: ");

        public function CustomTextField() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;
            this.addChild( customField );
            customField.bounds = new Rectangle( 0, 0, this.stage.stageWidth, this.stage.stageHeight );

            //track soft keyboard and stage resize events
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, onDisplayAreaChange );
            customField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, onDisplayAreaChange
);
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            //Fill the stage if possible, but avoid the area covered by a keyboard
            var desiredBounds = new Rectangle( 0, 0, this.stage.stageWidth, this.stage.stageHeight );
            if( this.stage.stageHeight - this.stage.softKeyboardRect.height < desiredBounds.height )
                desiredBounds.height = this.stage.stageHeight - this.stage.softKeyboardRect.height;

            customField.bounds = desiredBounds;
        }
    }
}
```

注：当 scaleMode 属性设置为 noScale 时，舞台仅调度 resize 事件以响应方向更改。在其他模式中，舞台尺寸不发生更改；而是通过缩放内容来补偿方向更改。

处理应用程序显示更改

AIR 2.6 和更高版本

在 AIR 中，可以通过将应用程序描述符中的 softKeyboardBehavior 元素设置为 none 来与关闭和开启软键盘相关的默认平移和调整大小行为。

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

在关闭自动行为后，由您的应用程序负责对应用程序显示的内容进行必要的调整。当打开键盘时将调度 softKeyboardActivate 事件。在调度 softKeyboardActivate 事件时，舞台的 softKeyboardRect 属性包含由打开的键盘覆盖的区域尺寸。使用这些尺寸来移动或调整内容的大小，以便在打开键盘和用户键入时正确显示内容。（当关闭键盘时，softKeyboardRect 矩形的尺寸全部为零。）

当关闭键盘时，会调度 softKeyboardDeactivate 事件，您可以将应用程序显示恢复成正常状态。

```
package  {
    import flash.display.MovieClip;
    import flash.events.SoftKeyboardEvent;
    import flash.events.Event;
    import flash.display.StageScaleMode;
    import flash.display.StageAlign;
    import flash.display.InteractiveObject;
    import flash.text.TextFieldType;
    import flash.text.TextField;

    public class PanningExample extends MovieClip {

        private var textField:TextField = new TextField();

        public function PanningExample() {
            this.stage.scaleMode = StageScaleMode.NO_SCALE;
            this.stage.align = StageAlign.TOP_LEFT;

            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
            textField.height = 200;
            textField.type = TextFieldType.INPUT;
            textField.border = true;
            textField.wordWrap = true;
            textField.multiline = true;

            this.addChild( textField );

            //track soft keyboard and stage resize events
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_ACTIVATE, onKeyboardChange );
            textField.addEventListener(SoftKeyboardEvent.SOFT_KEYBOARD_DEACTIVATE, onKeyboardChange );
            this.stage.addEventListener( Event.RESIZE, onDisplayAreaChange );
        }

        private function onDisplayAreaChange( event:Event ):void
        {
            textField.y = this.stage.stageHeight - 201;
            textField.width = this.stage.stageWidth;
        }

        private function onKeyboardChange( event:SoftKeyboardEvent ):void
        {
            var field:InteractiveObject = textField;
            var offset:int = 0;

            //if the softkeyboard is open and the field is at least partially covered
            if( (this.stage.softKeyboardRect.y != 0) && (field.y + field.height >
this.stage.softKeyboardRect.y) )
                offset = field.y + field.height - this.stage.softKeyboardRect.y;

            //but don't push the top of the field above the top of the screen
            if( field.y - offset < 0 ) offset += field.y - offset;

            this.y = -offset;
        }
    }
}
```

注：在 Android 上的某些情况 （包括全屏模式）下，无法从操作系统获取键盘的精确尺寸。在这些情况下，大小是估计值。另外，在横向方向中，对所有文本输入使用本机全屏 IME 键盘。此 IME 键盘具有一个内置的文本输入字段，并覆盖整个舞台。无法显示不能填充屏幕的横向键盘。

第 31 章：鼠标输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序可通过捕获并响应鼠标输入来创建交互性。请注意，本节假定您已熟悉了 ActionScript 3.0 事件模型。有关详细信息，请参阅第 106 页的 " 处理事件 "。

有关确定在运行时支持何种类型的鼠标 （例如永久性光标、笔针或者触摸屏输入）的信息，请参阅第 475 页的 " 了解输入类型 "。

更多帮助主题

flash.ui.Mouse

flash.events.MouseEvent

第 496 页的 " 触摸、多点触控和手势输入 "

捕获鼠标输入

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

鼠标单击将创建鼠标事件，这些事件可用来触发交互式功能。您可以将事件侦听器添加到舞台上以侦听在 SWF 文件中任何位置发生的鼠标事件。也可以将事件侦听器添加到舞台上从 InteractiveObject 进行继承的对象 （例如， Sprite 或 MovieClip）中；单击该对象时将触发这些侦听器。

与键盘事件一样，鼠标事件也会冒泡。在下面的示例中，由于 square 是 Stage 的子级，因此，单击正方形时，将从 Sprite square 和 Stage 对象中调度该事件：

```
var square:Sprite = new Sprite();
square.graphics.beginFill(0xFF0000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.addEventListener(MouseEvent.CLICK, reportClick);
square.x =
square.y = 50;
addChild(square);

stage.addEventListener(MouseEvent.CLICK, reportClick);

function reportClick(event:MouseEvent):void
{
    trace(event.currentTarget.toString() + " dispatches MouseEvent. Local coords [" + event.localX + "," +
event.localY + "] Stage coords [" + event.stageX + "," + event.stageY + "]");
}
```

请注意，在上面的示例中，鼠标事件包含有关单击的位置信息。 localX 和 localY 属性包含显示链中最低级别的子级上的单击位置。例如，单击 square 左上角时将报告本地坐标 [0,0]，因为它是 square 的注册点。或者， stageX 和 stageY 属性是指单击位置在舞台上的全局坐标。同一单击报告这些坐标为 [50,50]，因为 square 已移到这些坐标上。取决于响应用户交互的方式，这两种坐标对可能是非常有用的。

注：您可以在全屏模式下设置应用程序，以使用鼠标锁定。鼠标锁定将禁用光标，且启用没有限制的鼠标移动。有关详细信息，请参阅第 139 页的 " 使用全屏模式 "。

MouseEvent 对象还包含 altKey、 ctrlKey 和 shiftKey 布尔属性。可以使用这些属性来检查在鼠标单击时是否还按下了 Alt、Ctrl 或 Shift 键。

在舞台上拖曳 Sprite

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Sprite 类的 startDrag() 方法，可以允许用户在舞台上拖曳 Sprite 对象。下面的代码显示了一个这样的示例：

```
import flash.display.Sprite;
import flash.events.MouseEvent;

var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFFCC00);
circle.graphics.drawCircle(0, 0, 40);

var target1:Sprite = new Sprite();
target1.graphics.beginFill(0xCCFF00);
target1.graphics.drawRect(0, 0, 100, 100);
target1.name = "target1";

var target2:Sprite = new Sprite();
target2.graphics.beginFill(0xCCFF00);
target2.graphics.drawRect(0, 200, 100, 100);
target2.name = "target2";

addChild(target1);
addChild(target2);
addChild(circle);

circle.addEventListener(MouseEvent.MOUSE_DOWN, mouseDown)

function mouseDown(event:MouseEvent):void
{
    circle.startDrag();
}
circle.addEventListener(MouseEvent.MOUSE_UP, mouseReleased);

function mouseReleased(event:MouseEvent):void
{
    circle.stopDrag();
    trace(circle.dropTarget.name);
}
```

有关更多详细信息，请参阅第 144 页的 “ 改变位置 ” 中有关创建鼠标拖动交互的部分。

AIR 中的拖放

在 Adobe AIR 中，您可以启用拖放支持以允许用户将数据拖进及拖出您的应用程序。有关更多详细信息，请参阅 第 519 页的 “AIR 中的拖放 ”。

自定义鼠标光标

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以将鼠标光标 （鼠标指针）隐藏或交换为舞台上的任何显示对象。要隐藏鼠标光标，请调用 Mouse.hide() 方法。可通过以下方式来自定义光标：调用 Mouse.hide()，侦听舞台上是否发生 MouseEvent.MOUSE_MOVE 事件，以及将显示对象 （自定义光标）的坐标设置为事件的 stageX 和 stageY 属性。下面的示例说明了此任务的基本执行过程：

```
var cursor:Sprite = new Sprite();
cursor.graphics.beginFill(0x000000);
cursor.graphics.drawCircle(0,0,20);
cursor.graphics.endFill();
addChild(cursor);

stage.addEventListener(MouseEvent.MOUSE_MOVE,redrawCursor);
Mouse.hide();

function redrawCursor(event:MouseEvent):void
{
    cursor.x = event.stageX;
    cursor.y = event.stageY;
}
```

鼠标输入示例：WordSearch

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此示例通过处理鼠标事件来说明用户交互。用户在一个由随机字母组成的网格上构造尽可能多的词，拼写方法是：在网格中进行水平或垂直移动，但同一个字母只允许使用一次。此示例演示了 ActionScript 3.0 的下列功能：

- 动态构造组件网格
- 响应鼠标事件
- 根据用户交互维护分数

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/WordSearch 文件夹中找到 WordSearch 应用程序文件。该应用程序包含以下文件：

文件	说明
WordSearch.as	此类提供了应用程序的主要功能。
WordSearch.fla 或 WordSearch.mxml	Flex 或 Flash 的主应用程序文件 （分别为 MXML 和 FLA 格式）。
dictionary.txt	此文件用于确定拼写的词能否得分以及拼写是否正确。

加载字典

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要创建一个涉及查找词的游戏，您需要使用字典。本示例包含一个名为 dictionary.txt 的文本文件，该文件包含以回车符分隔的词列表。创建名为 words 的数组后， loadDictionary() 函数将请求加载此文件，成功加载后，此文件将变成一个很长的字符串。通过使用 split() 方法，在回车符 （字符代码 10）或换行符 （字符代码 13）的每个实例处断开，可以将该字符串分析为词的数组。这种分析是在 dictionaryLoaded() 函数中实现的：

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

创建用户界面

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在存储这些词后，您就可以设置用户界面了。请创建两个 Button 实例：一个用于提交词，另一个用于清除当前拼写的词。在每种情况下，您必须通过侦听该按钮所广播的 MouseEvent.CLICK 事件并随后调用一个函数来响应用户输入。在 setupUI() 函数中，此代码在两个按钮上创建侦听器：

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

生成游戏板

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

游戏板是由随机字母组成的网格。在 generateBoard() 函数中，二维网格是通过将一个循环嵌套到另一个循环中创建的。第一个循环增加行数，第二个循环增加每行的总列数。由这些行和列创建的每个单元格包含一个按钮，它表示游戏板上的一个字母。

```
private function generateBoard(startX:Number, startY:Number, totalRows:Number, totalCols:Number,
buttonSize:Number):void
{
    buttons = new Array();
    var colCounter:uint;
    var rowCounter:uint;
    for (rowCounter = 0; rowCounter < totalRows; rowCounter++)
    {
        for (colCounter = 0; colCounter < totalCols; colCounter++)
        {
            var b:Button = new Button();
            b.x = startX + (colCounter*buttonSize);
            b.y = startY + (rowCounter*buttonSize);
            b.addEventListener(MouseEvent.CLICK, letterClicked);
            b.label = getRandomLetter().toUpperCase();
            b.setSize(buttonSize,buttonSize);
            b.name = "buttonRow"+rowCounter+"Col"+colCounter;
            addChild(b);

            buttons.push(b);
        }
    }
}
```

虽然仅在一行中添加了 MouseEvent.CLICK 事件的侦听器，但由于该侦听器位于 for 循环中，因此会将其分配给每个 Button 实例。此外，还会为每个按钮分配一个从其行和列位置派生的名称，这为以后在代码中引用每个按钮的行和列提供了一种简便的方法。

通过用户输入来构造词

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

拼词规则是：选择垂直或水平相邻的字母，但每个字母只允许使用一次。每次单击时将生成一个鼠标事件，此时必须检查用户正在拼写的词，以确保该词正确地从以前单击的字母继续进行拼写。否则，将删除以前的词并开始拼写一个新词。此检查在 isLegalContinuation() 方法中执行。

```
private function isLegalContinuation(prevButton:Button, currButton:Button):Boolean
{
    var currButtonRow:Number = Number(currButton.name.charAt(currButton.name. indexOf("Row") + 3));
    var currButtonCol:Number = Number(currButton.name.charAt(currButton.name.indexOf("Col") + 3));
    var prevButtonRow:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Row") + 3));
    var prevButtonCol:Number = Number(prevButton.name.charAt(prevButton.name.indexOf("Col") + 3));

    return ((prevButtonCol == currButtonCol && Math.abs(prevButtonRow - currButtonRow) <= 1) ||
             (prevButtonRow == currButtonRow && Math.abs(prevButtonCol - currButtonCol) <= 1));
}
```

String 类的 charAt() 和 indexOf() 方法从当前单击的按钮和上次单击的按钮中检索相应的行和列。如果行或列未改变，或者行或列已改变但与上次单击位置的行或列之间相差不超过一行或一列，isLegalContinuation() 方法将返回 true。如果要更改游戏规则并允许斜向拼写，您可以删除对未改变的行或列的检查，最后一行将如下所示：

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol - currButtonCol) <= 1));
```

检查词的提交

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要完成游戏的代码，您需要提供检查词的提交和计算分数的机制。 searchForWord() 方法包含这两项功能：

```
private function searchForWord(str:String):Number
{
    if (words && str)
    {
        var i:uint = 0
        for (i = 0; i < words.length; i++)
        {
            var thisWord:String = words[i];
            if (str == words[i])
            {
                return i;
            }
        }
        return -1;
    }
    else
    {
        trace("WARNING: cannot find words, or string supplied is null");
    }
    return -1;
}
```

此函数循环访问字典中所有的词。如果用户的词与字典中的词匹配，则返回该词在字典中的位置。如果该位置有效，submitWord() 方法随后将检查响应并更新分数。

自定义

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

类的开头是几个常量。可通过修改这些变量来修改此游戏。例如，您可以通过增大 TOTAL_TIME 变量的值来更改可用于玩该游戏的时间。还可以稍微增大 PERCENT_VOWELS 变量的值来增大找到词的可能性。

第 32 章：触摸、多点触控和手势输入

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

Flash Platform 的触摸事件处理功能可以处理来自触摸感应设备上一个 / 多个接触点的输入。此外， Flash 运行时可以处理将多个触摸点与移动动作结合起来创建手势的事件。换句话说， Flash 运行时解释两种输入类型：

触摸 使用启用触摸的设备中的单点设备 （例如手指、笔针或其他工具）输入。某些设备支持多个同步接触点 （使用手指或笔针）。

多点触控 使用多个同步接触点输入。

手势 由设备或操作系统解释以响应一个或多个触摸事件的输入。例如，用户同时旋转两个手指，设备或操作系统会将触摸输入解释为旋转手势。有些手势使用一个手指或触摸点执行，而有些手势需要多个触摸点。设备或操作系统确定要分配给输入的手势类型。

触摸和手势输入都可以是多点触控输入，具体取决于用户的设备。 ActionScript 提供了相应 API，用于处理触摸事件、手势事件以及针对多点触控输入单独跟踪的触摸事件。

注：侦听触控和手势事件可能会占用大量的处理资源 （相当于每秒呈现若干帧），这取决于计算设备和操作系统。当您并不是真正需要触控或手势所提供的额外功能时，最好使用鼠标事件。当您使用触控或手势事件时，请考虑在平移、旋转或缩放操作期间减少可能发生图形更改的数量，尤其是当可以快速调度此类事件时。例如，当用户使用缩放手势调整某个组件的大小时，您可以停止该组件内的动画。

更多帮助主题

flash.ui.Multitouch

flash.events.TouchEvent

flash.events.GestureEvent

flash.events.TransformGestureEvent

flash.events.GesturePhase

flash.events.PressAndTapGestureEvent

Paul Trani：移动设备上的触控事件和手势

Mike Jones：虚拟游戏控制器

触摸输入的基础知识

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

当 Flash Platform 在支持触摸输入的环境中运行时， InteractiveObject 实例可以侦听触摸事件并调用处理函数。通常，可以像处理 ActionScript 中的其他事件一样处理触摸、多点触控和手势事件 （有关使用 ActionScript 进行事件处理的基本信息，请参阅第 106 页的 “ 处理事件 ”）。

但是，由于 Flash 运行时解释触摸或手势，所以运行时必须在支持触摸或多点触控输入的硬件和软件环境中运行。有关比较不同触摸屏类型的图表，请参阅第 475 页的 “ 了解输入类型 ”。此外，如果运行时在容器应用程序 （例如浏览器）内运行，则随后该容器会将输入传递到运行时。在某些情况下，虽然当前硬件和操作系统环境支持多点触控，但是包含 Flash 运行时的浏览器只解释输入却不将其传递到运行时。或者，完全忽略输入。

下图显示了从用户到运行时的输入流：

从用户到 Flash Platform 运行时的输入流

幸运的是，用于开发触摸应用程序的 ActionScript API 包括类、方法和属性，来确定运行时环境中是否支持触摸或多点触控输入。用于确定是否支持触摸输入的 API 是用于触摸事件处理的 " 发现 API"。

重要概念和术语

以下参考列表包含与编写触摸事件处理应用程序相关的重要术语：

发现 API 用于测试运行时环境是否支持触摸事件和其他输入模式的方法和属性。

触摸事件 使用单个接触点在启用触摸的设备上执行的输入动作。

触摸点 单个触摸事件的接触点。即使设备不支持手势输入，也可以支持多个同步触摸点。

触摸序列 表示单个触摸的生命期的一系列事件。这些事件包括一个开始事件、零个或多个移动事件和一个结束事件。

多点触控事件 使用多个接触点 （例如多个手指）在启用触摸的设备上执行的输入动作。

手势事件 在启用触摸的设备上执行的输入动作 （跟踪某些复杂的移动）。例如，某个手势使用两个手指触摸屏幕并同时沿抽象圆的周长移动以指示旋转。

阶段 事件流中不同的时间点 （例如开始和结束）。

笔针 与启用触摸的屏幕进行交互的工具。笔针比人类的手指更精确。某些设备仅识别来自特定类型的笔针的输入。识别笔针输入的设备可能不识别多个同步接触点或手指接触。

按住并点击 特定类型的多点触控输入手势，用户用一个手指按下启用触摸的设备，然后使用另一个手指或指针设备点击。此手势通常用于在多点触控应用程序中模拟鼠标右键单击。

触摸输入 API 结构

ActionScript 触摸输入 API 旨在面向触摸输入处理取决于 Flash 运行时的硬件和软件环境这一事实。触摸输入 API 主要面向三种触摸应用程序开发的需要：发现、事件和阶段。配合使用这些 API 可以为用户生成一个可预知和可以响应的体验；即使在您开发应用程序时目标设备是未知的也是如此。

发现

发现 API 提供了在运行时测试硬件和软件环境的功能。由运行时填充的值决定在当前上下文中，触摸输入是否可用于 Flash 运行时。此外，使用发现属性和方法的集合可以将应用程序设置为响应鼠标事件 （如果环境不支持某些触摸输入，则代替触摸事件）。有关详细信息，请参阅第 498 页的 " 触摸支持发现 "。

事件

ActionScript 使用事件侦听器和事件处理函数管理触摸屏输入事件，与它对其他事件的管理方式一样。但是，还必须考虑触摸屏输入事件处理：

- 设备或操作系统可以多种方式解释触摸，可以解释为一系列触摸，也可以笼统地解释为一个手势。
- 对启用触摸的设备的单个触摸 （通过手指、笔针或指针设备）也会始终调度鼠标事件。您可以处理具有 MouseEvent 类中事件类型的鼠标事件。或者，您可以将应用程序设计为只响应触摸事件。或者，您可以设计一个响应这两种类型事件的应用程序。

- 应用程序可以响应多个同步触摸事件并单独处理每个事件。

通常，使用发现 API 可以有条件地处理您的应用程序处理的事件以及处理方式。应用程序熟悉运行时环境后，在用户与应用程序交互时，它可以调用适当的处理函数或确定正确的事件对象。或者，应用程序可以指示在当前环境中无法处理特定输入，并为用户提供替代方法或信息。有关详细信息，请参阅第 499 页的 “Touch 事件处理 ” 和第 503 页的 “Gesture 事件处理 ”。

阶段

对于触摸和多点触控应用程序，触摸事件对象包含用于跟踪用户交互阶段的属性。编写 ActionScript 以处理阶段 （如用户输入的开始、更新或结束阶段），从而为用户提供反馈。响应事件阶段，以便可视对象随用户触摸和移动在屏幕上的触摸点而改变。或者，使用此阶段跟踪特定的手势属性，如手势演变。

对于触摸点事件，跟踪用户在特定的交互式对象上停留的时间。应用程序可分别跟踪多个同步触摸点阶段，并相应地对其进行处理。

对于手势，当动作发生时解释有关动作转换的特定信息。当接触点 （或多个接触点）沿屏幕移动时，跟踪其坐标。

触摸支持发现

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

使用多点触控类属性设置您的应用程序处理的触摸输入的范围。然后，测试此环境以确保支持您的 ActionScript 处理的事件。具体来说，首先确定应用程序的触摸输入的类型。选项是：触摸点、手势或无 （将所有触摸输入解释为鼠标单击且仅使用鼠标事件处理函数）然后，使用 Multitouch 类的属性和方法确保运行时位于支持您的应用程序需要的触摸输入的环境中。测试运行时环境是否支持这些类型的触摸输入 （例如，是否能够解释手势）并相应地做出响应。

注：Multitouch 类属性是静态属性，且不属于任何类的实例。将其与语法 Multitouch.property 一起使用，例如：

```
var touchSupport:Boolean = Multitouch.supportsTouchEvents;
```

设置输入类型

Flash 运行时必须知道要解释的触摸输入的类型，因为触摸事件可能包含多个元素或阶段。如果手指仅触摸启用触摸的屏幕，则运行时是否会调度触摸事件？或者，运行时是否会等待手势？或者，运行时是否将触摸作为鼠标按下事件进行跟踪？支持触摸输入的应用程序必须确定其针对 Flash 运行时处理的触摸事件的类型。使用 Multitouch.inputMode 属性为运行时确定触摸输入类型。输入模式可以是以下三个选项之一：

无 对触摸事件不提供特殊处理。设置：Multitouch.inputMode=MultitouchInputMode.NONE 并使用 MouseEvent 类处理输入。

单个触摸点 分别解释所有触摸输入，并跟踪和处理所有触摸点。设置：Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT 并使用 TouchEvent 类处理输入。

手势输入 设备或操作系统将输入解释为手指沿屏幕移动的一种复杂形式。设备或操作系统将移动集体分配给单个手势输入事件。设置：Multitouch.inputMode=MultitouchInputMode.GESTURE 并使用 TransformGestureEvent、PressAndTapGestureEvent 或 GestureEvent 类处理输入。

有关在处理触摸事件之前使用 Multitouch.inputMode 属性设置输入类型的示例，请参阅第 499 页的 “Touch 事件处理 ”。

测试是否支持触摸输入

Multitouch 类的其他属性提供了相应的值，用于微调您的应用程序以支持当前环境中的触摸。Flash 运行时填充允许同时进行的触摸点的数量值或可用手势的数量值。如果运行时位于不支持您的应用程序所需的触摸事件处理的环境中，则为用户提供其他处理。例如，提供鼠标事件处理或有关在当前环境中可用或不可用的功能的信息。

还可以将 API 用于键盘、触摸和鼠标支持，请参阅第 475 页的 " 了解输入类型 "。

有关兼容性测试的详细信息，请参阅第 506 页的 " 疑难解答 "。

Touch 事件处理

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

在 ActionScript 中，基本触摸事件与其他事件 （如鼠标事件）的处理方式相同。您可以侦听由 TouchEvent 类中的事件类型定义的一系列触摸事件。

注：对于多个触摸点输入 （例如，使用多个手指触摸设备），第一个接触点将调度鼠标事件和触摸事件。

处理基本触摸事件：

1 通过将 flash.ui.Multitouch.inputMode 属性设置为 MultitouchInputMode.TOUCH_POINT，可以将您的应用程序设置为处理触摸事件。

2 将事件侦听器附加到从 InteractiveObject 类继承属性的类实例，如 Sprite 或 TextField。

3 指定要处理的触摸事件的类型。

4 调用事件处理函数以执行某些操作，从而响应事件。

例如，当在启用触摸的屏幕上点击在 mySprite 上绘制的正方形时，以下代码显示一则消息：

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

Touch 事件属性

发生某个事件时，将创建一个事件对象。 TouchEvent 对象包含有关触摸事件的位置和条件的信息。您可以使用事件对象的属性检索该信息。

例如，以下代码创建 TouchEvent 对象 evt，然后在文本字段中显示事件对象的 stageX 属性（发生触摸的舞台空间中该点的 x 坐标）：

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taphandler);

function taphandler(evt:TouchEvent): void {
myTextField.text = evt.stageX.toString;
myTextField.y = 50;
addChild(myTextField);
}
```

请参阅通过事件对象提供的属性的 TouchEvent 类。

注：并非所有运行时环境都支持所有 TouchEvent 属性。例如，并非所有启用触摸的设备都能够检测用户应用到触摸屏的压力。因此，这些设备不支持 TouchEvent.pressure 属性。尝试测试是否支持特定属性以确保您的应用程序能够正常工作，有关详细信息，请参阅第 506 页的 “ 疑难解答 ”。

触摸事件阶段

跟踪 InteractiveObject 内外的各种舞台中的触摸事件，就像您跟踪鼠标事件一样。并且，跟踪触摸交互开头、中间和结尾中的触摸事件。 TouchEvent 类提供了用于处理 touchBegin、 touchMove 和 touchEnd 事件的值。

例如，您可以使用 touchBegin、 touchMove 和 touchEnd 事件在用户触摸和移动显示对象时为其提供可视反馈：

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
myTextField.width = 200;
myTextField.height = 20;
addChild(myTextField);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
function onTouchBegin(event:TouchEvent) {
    myTextField.text = "touch begin" + event.touchPointID;
}
function onTouchMove(event:TouchEvent) {
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    myTextField.text = "touch end" + event.touchPointID;
}
```

注：将初始触摸侦听器附加到 mySprite，但不要附加用于移动和结束触摸事件的侦听器。如果用户的手指或指针设备先于显示对象移动，则舞台将继续侦听触摸事件。

触摸点 ID

编写用于响应触摸输入的应用程序需要 TouchEvent.touchPointID 属性。 Flash 运行时为每个触摸点分配一个唯一的 touchPointID 值。当应用程序响应触摸输入阶段或触摸输入的移动时，请先检查 touchPointID，然后再处理该事件。 Sprite 类的触摸输入拖动方法将 touchPointID 属性用作参数，以便处理正确的输入实例。touchPointID 属性确保事件处理函数响应正确的触摸点。否则，事件处理函数将响应设备上触摸事件类型的任何实例 （例如，所有 touchMove 事件），从而产生不可预测的行为。此属性在用户拖动对象时特别重要。

使用 touchPointID 属性可以管理整个触摸序列。触摸序列包含一个 touchBegin 事件、 0 个或多个 touchMove 事件和一个 touchEnd 事件，所有这些事件都具有相同的 touchPointID 值。

以下示例建立了一个变量 touchMoveID，用于测试正确的 touchPointID 值，以便对触摸移动事件做出响应。此外，其他触摸输入也会触发事件处理函数。请注意，用于侦听移动和结束阶段的侦听器位于舞台上，而不是显示对象上。舞台将侦听移动或结束阶段，以防用户的触摸移动到显示对象边界以外。

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);
var myTextField:TextField = new TextField();
addChild(myTextField);
myTextField.width = 200;
myTextField.height = 20;
var touchMoveID:int = 0;

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
function onTouchBegin(event:TouchEvent) {
    if(touchMoveID != 0) {
        myTextField.text = "already moving. ignoring new touch";
        return;
    }
    touchMoveID = event.touchPointID;

    myTextField.text = "touch begin" + event.touchPointID;
    stage.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);
}
function onTouchMove(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch";
        return;
    }
    mySprite.x = event.stageX;
    mySprite.y = event.stageY;
    myTextField.text = "touch move" + event.touchPointID;
}
function onTouchEnd(event:TouchEvent) {
    if(event.touchPointID != touchMoveID) {
        myTextField.text = "ignoring unrelated touch end";
        return;
    }
    touchMoveID = 0;
    stage.removeEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
    stage.removeEventListener(TouchEvent.TOUCH_END, onTouchEnd);
    myTextField.text = "touch end" + event.touchPointID;
}
```

触摸和拖动

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

以下两个方法已添加到 Sprite 类，以便为支持触摸点输入且启用触摸的应用程序提供更多支持：Sprite.startTouchDrag() 和 Sprite.stopTouchDrag()。这些方法的作用与针对鼠标事件的 Sprite.startDrag() 和 Sprite.stopDrag() 的作用相同。然而，请注意，Sprite.startTouchDrag() 和 Sprite.stopTouchDrag() 方法都将 touchPointID 值用作参数。

运行时将 touchPointID 值分配给触摸事件的事件对象。当环境支持多个同步触摸点时 （即使不处理手势），使用此值响应特定的触摸点。有关 touchPointID 属性的详细信息，请参阅第 501 页的 " 触摸点 ID"。

以下代码显示触摸事件的简单开始拖动事件处理函数和停止拖动事件处理函数。变量 bg 是一个包含 mySprite 的显示对象：

```
mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(e:TouchEvent) {
    e.target.startTouchDrag(e.touchPointID, false, bg.getRect(this));
    trace("touch begin");
}

function onTouchEnd(e:TouchEvent) {
    e.target.stopTouchDrag(e.touchPointID);
    trace("touch end");
}
```

并且，以下代码显示更高级的示例，该示例将拖动与触摸事件阶段组合在一起：

```
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_BEGIN, onTouchBegin);
mySprite.addEventListener(TouchEvent.TOUCH_MOVE, onTouchMove);
mySprite.addEventListener(TouchEvent.TOUCH_END, onTouchEnd);

function onTouchBegin(evt:TouchEvent) {
    evt.target.startTouchDrag(evt.touchPointID);
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}

function onTouchMove(evt:TouchEvent) {
    evt.target.alpha = 0.5;
}

function onTouchEnd(evt:TouchEvent) {
    evt.target.stopTouchDrag(evt.touchPointID);
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
```

Gesture 事件处理

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

处理手势事件的方式与处理基本触摸事件相同。您可以侦听由 TransformGestureEvent 类、 GestureEvent 类和 PressAndTapGestureEvent 类中的事件类型常数定义的一系列手势事件。

要处理手势触摸事件，请执行下列操作：

1 通过将 flash.ui.Multitouch.inputMode 属性设置为 MultitouchInputMode.GESTURE 将您的应用程序设置为处理手势输入。

2 将事件侦听器附加到从 InteractiveObject 类继承属性的类实例，如 Sprite 或 TextField。

3 指定要处理的手势事件的类型。

4 调用事件处理函数以执行某些操作，从而响应事件。

例如，当在启用触摸的屏幕上点击在 mySprite 上绘制的正方形时，以下代码显示一则消息：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, swipehandler);

function swipehandler(evt:TransformGestureEvent): void {
myTextField.text = "I've been swiped";
myTextField.y = 50;
addChild(myTextField);
}
```

以相同的方式处理二指点击事件，但要使用 GestureEvent 类：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, taphandler);

function taphandler(evt:GestureEvent): void {
myTextField.text = "I've been two-finger tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

也以相同的方式处理按住轻敲事件，但要使用 PressAndTapGestureEvent 类：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField();

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(PressAndTapGestureEvent.GESTURE_PRESS_AND_TAP, taphandler);

function taphandler(evt:PressAndTapGestureEvent): void {
myTextField.text = "I've been press-and-tapped";
myTextField.y = 50;
addChild(myTextField);
}
```

注：所有运行时环境中并非支持所有 GestureEvent、 TransformGestureEvent 和 PressAndTapGestureEvent 事件类型。例如，并非所有启用触摸的设备都能够检测多个手指滑动。因此，这些设备不支持 InteractiveObject gestureSwipe 事件。尝试测试是否支持特定事件以确保您的应用程序能够正常工作，有关详细信息，请参阅第 506 页的 “ 疑难解答 ”。

Gesture 事件属性

手势事件的事件属性范围比基本触摸事件小。您可以通过事件处理函数中的事件对象以相同的方式访问这些属性。

例如，以下代码在用户对 mySprite 执行旋转手势时会旋转该对象。该文本字段显示自最后一个手势之后的旋转量 （测试此代码时，将其多旋转几次以查看值更改）：

```
Multitouch.inputMode=MultitouchInputMode.GESTURE;

var mySprite:Sprite = new Sprite();
var mySpriteCon:Sprite = new Sprite();
var myTextField:TextField = new TextField();
myTextField.y = 50;
addChild(myTextField);

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(-20,-20,40,40);
mySpriteCon.addChild(mySprite);
mySprite.x = 20;
mySprite.y = 20;
addChild(mySpriteCon);

mySprite.addEventListener(TransformGestureEvent.GESTURE_ROTATE, rothandler);

function rothandler(evt:TransformGestureEvent): void {
evt.target.parent.rotationZ += evt.target.rotation;
myTextField.text = evt.target.parent.rotation.toString();
}
```

注：所有运行时环境中并非支持所有 TransformGestureEvent 属性。例如，并非所有启用触摸的设备都能够检测屏幕上的手势旋转。因此，这些设备不支持 TransformGestureEvent.rotation 属性。尝试测试是否支持特定属性以确保您的应用程序能够正常工作，有关详细信息，请参阅第 506 页的 “ 疑难解答 ”。

Gesture 阶段

另外，可以跟踪阶段中的手势事件，因此您可以在发生手势时跟踪相应的属性。例如，您可以在使用滑动手势移动对象时跟踪 X 坐标。当滑动完成后，使用这些值绘制一条在其路径中经过所有点的直线。或者，在使用全景手势沿屏幕拖动显示对象时以可视方式更改该对象。当全景手势完成后，再次更改该对象。

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {

    evt.target.localX++;

if (evt.phase==GesturePhase.BEGIN) {
    myTextField.text = "Begin";
    evt.target.scaleX *= 1.5;
    evt.target.scaleY *= 1.5;
}
if (evt.phase==GesturePhase.UPDATE) {
    myTextField.text = "Update";
    evt.target.alpha = 0.5;
}
if (evt.phase==GesturePhase.END) {
    myTextField.text = "End";
    evt.target.width = 40;
    evt.target.height = 40;
    evt.target.alpha = 1;
}
}
```

注：更新阶段的频率取决于运行时的环境。某些操作系统和硬件组合根本不会传递更新。

手势阶段对于简单的动作事件为 “all”

某些手势事件对象不跟踪单个手势事件阶段，而是使用值 all 填充事件对象的 phase 属性。简单的动作滑动和二指点击不跟踪包含多个阶段的事件。侦听 gestureSwipe 或 gestureTwoFingerTap 事件的 InteractiveObject 的事件对象的 phase 属性在调度事件后始终为 all：

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;
var mySprite = new Sprite();
mySprite.addEventListener(TransformGestureEvent.GESTURE_SWIPE, onSwipe);
mySprite.addEventListener(GestureEvent.GESTURE_TWO_FINGER_TAP, onTwoTap);
mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0, 0, 40, 40);
var myTextField = new TextField();
myTextField.y = 200;
addChild(mySprite);
addChild(myTextField);

function onSwipe(swipeEvt:TransformGestureEvent):void {
    myTextField.text = swipeEvt.phase // Output is "all"
}
function onTwoTap(tapEvt:GestureEvent):void {
    myTextField.text = tapEvt.phase // Output is "all"
}
```

疑难解答

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

支持触摸输入的硬件和软件的变化速度很快。此参考不能维护支持多点触控的操作系统和软件组合上的每个设备的列表。但是，它提供了有关如何使用发现 API 来确定是否可以在支持多点触控的设备上部署您的应用程序的指导，还提供了用于疑难解答 ActionScript 代码的技巧。

Flash 运行时基于传递给运行时的设备、操作系统或包含的软件 （如浏览器）的信息来响应触摸事件。对软件环境的这种依赖关系使文档多点触控兼容变得很复杂。某些设备对手势或触摸动作的解释不同于其他设备。旋转是由两个手指同时旋转定义的吗？旋转是指使用一个手指在屏幕上绘制圆形吗？根据硬件和软件环境，旋转手势可能会完全不同。因此，当用户输入时，设备会告知操作系统，然后操作系统将该信息传递给运行时。如果运行时位于浏览器之内，则有时浏览器软件会解释手势或触摸事件，但不会将输入传递给运行时。此行为类似于 “ 热键 ” 行为：尝试使用特定的键组合使 Flash Player 在浏览器内部执行某些操作且浏览器始终打开一个菜单。

如果单独的 API 和类与特定的操作系统不兼容，会显示出来。您可以从以下位置了解各个 API 条目，首先了解 Multitouch 类：http://help.adobe.com/zh_CN/FlashPlatform/reference/actionscript/3/flash/ui/Multitouch.html。

以下是一些常用的手势和触摸说明：

平移 从左到右或从右到左移动手指。某些设备要求两个手指平移。

旋转 按下两个手指，然后绕一个圆移动 （就像它们同时在跟踪平面上一个虚构的圆一样）。将轴点设置为这两个手指触摸点之间的中点。

滑动 从左到右或从右到左、从上到下或从下到上快速移动三个手指。

缩放 按下两个手指，然后使其互相远离进行放大，互相靠近进行缩小。

按住并点击 移动或按下一个手指，然后使用另一个手指点击平面。

每个设备都有自己的文档，其中介绍有关该设备支持的手势以及如何在该设备上执行各个动作的信息。通常，在手势之间，用户必须移开与设备接触的所有手指，具体取决于操作系统。

如果您发现您的应用程序无法响应触摸事件或手势，请执行以下测试：

1 您是否将触摸或手势事件的事件侦听器附加到从 InteractiveObject 类继承的 Object 类？只有 InteractiveObject 实例可以侦听触摸和手势事件

2 是否正在 Flash Professional CS5 中测试您的应用程序？如果是，尝试发布和测试该应用程序，因为 Flash Professional 可截获交互。

3 首先启动简单的事件并查看哪些可以正常工作 （以下代码示例来自 Multitouch.inputMode 的 API 条目：

```
Multitouch.inputMode=MultitouchInputMode.TOUCH_POINT;
var mySprite:Sprite = new Sprite();
var myTextField:TextField = new TextField()

mySprite.graphics.beginFill(0x336699);
mySprite.graphics.drawRect(0,0,40,40);
addChild(mySprite);

mySprite.addEventListener(TouchEvent.TOUCH_TAP, taplistener);

function taplistener(e:TouchEvent): void {
    myTextField.text = "I've been tapped";
    myTextField.y = 50;
    addChild(myTextField);
}
```

点击此矩形。如果此示例可以正常运行，即可了解您的环境支持简单的轻敲。然后您可以尝试执行更复杂的处理。

测试手势支持更复杂。单个设备或操作系统支持任何动作输入组合或不支持任何手势输入。

以下是一个简单的缩放手势测试：

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_ZOOM , onZoom);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a zoom gesture";
addChild(myTextField);

function onZoom(evt:TransformGestureEvent):void {
    myTextField.text = "Zoom is supported";
}
```

对设备执行缩放手势，并查看该文本字段是否填充了 Zoom is supported 消息。事件侦听器已添加到舞台中，因此您可以对测试应用程序的任何部分执行手势。

以下是一个简单的全景手势测试：

```
Multitouch.inputMode = MultitouchInputMode.GESTURE;

stage.addEventListener(TransformGestureEvent.GESTURE_PAN , onPan);
var myTextField = new TextField();
myTextField.y = 200;
myTextField.text = "Perform a pan gesture";
addChild(myTextField);

function onPan(evt:TransformGestureEvent):void {
    myTextField.text = "Pan is supported";
}
```

对设备执行全景手势，并查看该文本字段是否填充了 Pan is supported 消息。事件侦听器已添加到舞台中，因此您可以对测试应用程序的任何部分执行手势。

某些操作系统和设备组合同时支持这两种手势，某些只支持一种，某些这两种都不支持。测试您的应用程序的部署环境以了解情况。

已知问题

以下是与触摸屏输入相关的已知问题：

1 Windows Mobile 操作系统上的 Mobile Internet Explorer 会自动缩放 SWF 文件内容：

通过将下列代码添加到承载 SWF 文件的 HTML 页面来覆盖此 Internet Explorer 缩放行为：

```
<head>
<meta name="viewport" content="width=device-width, height=device-height, initial-scale=1.0">
</head>
```

2 对于 Windows 7 （也可能是其他操作系统），用户必须在手势之间将指针设备 （或手指）从屏幕上移开。例如：旋转和缩放图像：

- 执行旋转手势。
- 将您的手指从屏幕上移开。
- 将您的手指放回到屏幕上并执行缩放手势。

3 对于 Windows 7 （可能是其他操作系统），如果用户非常快速地执行手势，则旋转和缩放手势并不始终生成 “update” 相位。

4 Windows 7 Starter Edition 不支持多点触控。有关详细信息，请参阅 AIR Labs 论坛：
http://forums.adobe.com/thread/579180?tstart=0

5 对于 Mac OS 10.5.3 及更高版本， Multitouch.supportsGestureEvents 值始终为 true，即使硬件不支持手势事件也是如此。

第 33 章：复制和粘贴

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

使用剪贴板 API 中的类将信息复制到系统剪贴板中或从中复制信息。与 Adobe® Flash® Player 或 Adobe® AIR® 中运行的应用程序进行相互数据传输时，可使用的数据格式包括：

- 文本
- HTML 格式的文本
- RTF 数据
- 序列化对象
- 对象引用 （仅在源应用程序内有效）
- 位图 （仅限 AIR）
- 文件 （仅限 AIR）
- URL 字符串 （仅限 AIR）

复制粘贴基础知识

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

复制和粘贴 API 包含以下类。

包	类
flash.desktop	• Clipboard • ClipboardFormats • ClipboardTransferMode

静态 Clipboard.generalClipboard 属性表示操作系统剪贴板。 Clipboard 类为从 Clipboard 对象读取数据或向其中写入数据提供了方法。

HTMLLoader 类 （在 AIR 中）和 TextField 类用于实现一般复制和粘贴快捷键的默认行为。若要实现自定义组件的复制和粘贴快捷键行为，您可以直接侦听这些键击。您也可以使用本机菜单命令及等效键来间接响应键击。

可以在一个 Clipboard 对象中包含同一信息的不同表示形式，以使其他应用程序更易于理解和使用其中的数据。例如，图像可以以图像数据形式、序列化的 Bitmap 对象形式和文件形式包含在其中。以某种格式呈现数据的操作可以延迟，以便直到读取此格式的数据时才真正创建此格式。

读取和写入系统剪贴板

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

若要读取操作系统剪贴板，请调用 Clipboard.generalClipboard 对象的 getData() 方法，并传递要读取的格式的名称：

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

注 在 Flash Player 中运行的内容或在 AIR 的非应用程序沙箱中运行的内容只能在 paste 事件的事件处理函数中调用 getData() 方法。换句话说，只有在 AIR 应用程序沙箱中运行的代码才能在 paste 事件处理函数的外部调用 getData() 方法。

若要写入剪贴板，请以一种或多种格式将数据添加到 Clipboard.generalClipboard 对象。任何同一格式的现有数据都将被自动覆盖。然而，建议在将新数据写入系统剪贴板之前清除系统剪贴板，这样可确保任何其他格式的无关数据也将删除。

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

var textToCopy:String = "Copy to clipboard.";
Clipboard.generalClipboard.clear();
Clipboard.generalClipboard.setData(ClipboardFormats.TEXT_FORMAT, textToCopy, false);
```

注：在 Flash Player 中运行的内容或在 AIR 的非应用程序沙箱中运行的内容只能在用户事件 （如键盘或鼠标事件，或者 copy 或 cut 事件）的事件处理函数中调用 setData() 方法。换句话说，只有在 AIR 应用程序沙箱中运行的代码才能在用户事件处理函数的外部调用 setData() 方法。

AIR 中的 HTML 复制和粘贴

Adobe AIR 1.0 和更高版本

Adobe AIR 中的 HTML 环境自身提供针对复制和粘贴的事件组和默认行为。只有在应用程序安全沙箱中运行的代码才能通过 AIR Clipboard.generalClipboard 对象直接访问系统剪贴板。在非应用程序安全沙箱中运行的 JavaScript 代码可以通过因响应 HTML 文档中某元素调度的某个复制或粘贴事件而调度的事件对象来访问剪贴板。

复制和粘贴事件包括：copy、 cut 和 paste。为这些事件调度的对象可通过 clipboardData 属性提供对剪贴板的访问。

默认行为

Adobe AIR 1.0 和更高版本

默认情况下，AIR 将复制选定的项目以响应复制命令，该命令可通过快捷键或上下文菜单生成。在可编辑区域内，AIR 将剪切文本以响应剪切命令，或将文本粘贴到光标位置或选定位置以响应粘贴命令。

若要阻止默认行为，事件处理函数可以调用被调度事件对象的 preventDefault() 方法。

使用事件对象的 clipboardData 属性

Adobe AIR 1.0 和更高版本

如果事件对象是因为某个复制或粘贴事件而被调度，则该事件对象的 clipboardData 属性允许您读取和写入剪贴板数据。

若要在处理复制或剪切事件时写入剪贴板，请使用 clipboardData 对象的 setData() 方法，并传递要复制的数据和 MIME 类型：

```
function customCopy(event){
    event.clipboardData.setData("text/plain", "A copied string.");
}
```

若要访问被粘贴的数据，您可以使用 clipboardData 对象的 getData() 方法，并传递数据格式的 MIME 类型。可用格式由 types 属性报告。

```
function customPaste(event){
    var pastedData = event.clipboardData("text/plain");
}
```

只能在 paste 事件调度的事件对象中访问 getData() 方法和 types 属性。

下面的示例说明如何覆盖 HTML 页中默认的复制和粘贴行为。 copy 事件处理函数将复制的文本设置为斜体并将其作为 HTML 文本复制到剪贴板。cut 事件处理函数将选定的数据复制到剪贴板并将其从文档中移除。paste 处理函数将剪贴板内容作为 HTML 插入并将插入内容的样式设置为粗体文本。

```
<html>
<head>
    <title>Copy and Paste</title>
    <script language="javascript" type="text/javascript">
        function onCopy(event){
            var selection = window.getSelection();
            event.clipboardData.setData("text/html","<i>" + selection + "</i>");
            event.preventDefault();
        }

        function onCut(event){
             var selection = window.getSelection();
             event.clipboardData.setData("text/html","<i>" + selection + "</i>");
             var range = selection.getRangeAt(0);
             range.extractContents();

            event.preventDefault();
        }

        function onPaste(event){
            var insertion = document.createElement("b");
            insertion.innerHTML = event.clipboardData.getData("text/html");
             var selection = window.getSelection();
             var range = selection.getRangeAt(0);
             range.insertNode(insertion);
            event.preventDefault();
        }
    </script>
</head>
<body onCopy="onCopy(event)"
     onPaste="onPaste(event)"
     onCut="onCut(event)">
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam
voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.</p>
</body>
</html>
```

剪贴板数据格式

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

剪贴板格式描述了 Clipboard 对象中放置的数据。Flash Player 或 AIR 可在 ActionScript 数据类型和系统剪贴板格式之间自动进行标准数据格式的转换。此外，使用应用程序定义的格式可以在基于 ActionScript 的应用程序内或这些应用程序之间传输应用程序对象。

Clipboard 对象可以包含采用不同格式的同一信息的多种表示形式。例如，表示 Sprite 的 Clipboard 对象可以包括在同一应用程序中使用的引用格式、由在 Flash Player 或 AIR 中运行的另一应用程序使用的序列化格式、由图像编辑器使用的位图格式以及文件列表格式，还可能具有延迟呈示功能以对 PNG 文件进行编码，以便将 Sprite 的表示形式复制或拖动到文件系统。

标准数据格式

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

ClipboardFormats 类中提供了用于定义标准格式名称的常量：

常量	说明
TEXT_FORMAT	文本格式数据与 ActionScript String 类之间的转换。
HTML_FORMAT	带有 HTML 标记的文本。
RICH_TEXT_FORMAT	RTF 格式数据与 ActionScript ByteArray 类之间的转换。不会以任何方式对 RTF 标记进行解释或转换。
BITMAP_FORMAT	（仅限 AIR）位图格式数据与 ActionScript BitmapData 类之间的转换。
FILE_LIST_FORMAT	（仅限 AIR）文件列表格式数据与 ActionScript File 对象数组之间的转换。
URL_FORMAT	（仅限 AIR） URL 格式数据与 ActionScript String 类之间的转换。

因响应 HTML 内容 （在 AIR 应用程序中承载的）中的 copy、 cut 或 paste 事件而复制和粘贴数据时，必须使用 MIME 类型而不是 ClipboardFormat 字符串。有效的数据 MIME 类型为：

MIME 类型	说明
文本	"text/plain"
URL	"text/uri-list"
位图	"image/x-vnd.adobe.air.bitmap"
文件列表	"application/x-vnd.adobe.air.file-list"

注：如果事件对象是因为 HTML 内容中的 paste 事件而被调度，则无法从该事件对象的 clipboardData 属性中获得 RTF 格式数据。

自定义数据格式

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以使用应用程序定义的自定义格式将对象作为引用或序列化副本传输。参考只在同一应用程序中有效。序列化对象可以在应用程序之间传输，但只能用于序列化和取消序列化后仍然有效的对象。如果对象的属性为简单类型或可序列化对象，则通常可以将该对象序列化。

若要将序列化对象添加到 Clipboard 对象，请在调用 Clipboard.setData() 方法时将可序列化 参数设置为 true。格式名称可以是标准格式中的某一种或由应用程序定义的任意字符串。

传输模式

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

使用自定义数据格式将对象写入剪贴板后，可以从剪贴板中将对象数据作为引用读取，也可以将其作为原始对象的序列化副本读取。共有四种传输模式，这些模式确定了对象是以引用还是以序列化副本的形式传输：

传输模式	说明
ClipboardTransferModes.ORIGINAL_ONLY	仅返回引用。如果没有引用，则返回 null 值。
ClipboardTransferModes.ORIGINAL_PREFFERED	如果存在引用，将返回引用。否则返回序列化副本。
ClipboardTransferModes.CLONE_ONLY	仅返回序列化副本。如果没有可用的序列化副本，则返回 null 值。
ClipboardTransferModes.CLONE_PREFFERED	如果存在序列化副本，将返回序列化副本。否则返回引用。

读取和写入自定义数据格式

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

将对象写入剪贴板时，可将任何不以保留前缀 air: 或 flash: 开头的字符串用于格式 参数。请使用相同字符串作为读取对象的格式。下面的示例说明了如何从剪贴板读取对象和向其中写入对象：

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

若要从 Clipboard 对象中提取序列化对象 （放置或粘贴操作之后），请使用相同格式名称和 CLONE_ONLY 或 CLONE_PREFFERED 传输模式。

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

此时将始终向 Clipboard 对象添加一个引用。若要从 Clipboard 对象中提取引用 （放置或粘贴操作之后）而不是提取序列化副本，请使用 ORIGINAL_ONLY 或 ORIGINAL_PREFFERED 传输模式：

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

仅当 Clipboard 对象是源自当前应用程序中时，引用才有效。当引用可用时，可使用 ORIGINAL_PREFFERED 传输模式访问该引用；当引用不可用时，则可使用该模式访问序列化副本。

延迟呈现

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

如果创建数据格式的计算成本高昂，则可以通过提供按需提供数据的函数来使用延迟呈现。仅当放置或粘贴操作的接收方请求延迟格式的数据时才会调用此函数。

呈现函数是通过使用 setDataHandler() 方法添加到 Clipboard 对象中的。该函数必须返回相应格式的数据。例如，如果您调用 setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)，则 writeText() 函数必须返回字符串。

如果使用 setData() 方法将同一类型的数据格式添加到 Clipboard 对象，则该数据的优先级高于其延迟版本 （不会调用呈示函数）。如果再次访问该同一剪贴板数据，则可能会再次调用或不调用该呈示函数。

注：在 Mac OS X 上，延迟呈示仅适用于自定义数据格式。使用标准数据格式时，会直接调用呈示函数。

使用延迟呈现函数粘贴文本

Flash Player 10 和更高版本， **Adobe AIR 1.0** 和更高版本

下面的示例说明了如何实现延迟呈现函数。

当用户按下 “Copy” 按钮时，应用程序将清除系统剪贴板以确保不会留下上次剪贴板操作的数据。然后， setDataHandler() 方法将 renderData() 函数设置为剪贴板渲染器。

当用户从目标文本字段的上下文菜单中选择 “ 粘贴 ” 命令时，应用程序将访问剪贴板并设置目标文本。由于已使用函数而不是字符串设置了剪贴板上的文本数据格式，剪贴板将调用 renderData() 函数。 renderData() 函数将返回源文本中的文本，然后将其分配给目标文本。

请注意，如果按下 “Paste” 按钮前编辑源文本，则此编辑操作将反映到粘贴的文本中，即使按下 “Paste” 按钮后再进行编辑也是如此。这是因为呈现函数直到按下 “Paste” 按钮后才会复制源文本。（当在实际的应用程序中使用延迟呈现时，最好以某种方式存储或保护源数据以防止出现此问题。）

Flash 示例

```
package {
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;
    import flash.desktop.ClipboardTransferMode;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldType;
    import flash.events.MouseEvent;
    import flash.events.Event;
    public class DeferredRenderingExample extends Sprite
    {
        private var sourceTextField:TextField;
        private var destination:TextField;
        private var copyText:TextField;
        public function DeferredRenderingExample():void
        {
            sourceTextField = createTextField(10, 10, 380, 90);
            sourceTextField.text = "Neque porro quisquam est qui dolorem "
                + "ipsum quia dolor sit amet, consectetur, adipisci velit.";

            copyText = createTextField(10, 110, 35, 20);
            copyText.htmlText = "<a href='#'>Copy</a>";
            copyText.addEventListener(MouseEvent.CLICK, onCopy);

            destination = createTextField(10, 145, 380, 90);
            destination.addEventListener(Event.PASTE, onPaste);
        }
        private function createTextField(x:Number, y:Number, width:Number,
                        height:Number):TextField
        {
            var newTxt:TextField = new TextField();
            newTxt.x = x;
            newTxt.y = y;
            newTxt.height = height;
            newTxt.width = width;
            newTxt.border = true;
            newTxt.multiline = true;
            newTxt.wordWrap = true;
            newTxt.type = TextFieldType.INPUT;
```

```
            addChild(newTxt);
            return newTxt;
        }
        public function onCopy(event:MouseEvent):void
        {
            Clipboard.generalClipboard.clear();
            Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT,
                            renderData);
        }
        public function onPaste(event:Event):void
        {
            sourceTextField.text =
            Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
        }
        public function renderData():String
        {
            trace("Rendering data");
            var sourceStr:String = sourceTextField.text;
            if (sourceTextField.selectionEndIndex >
                     sourceTextField.selectionBeginIndex)
            {
                return sourceStr.substring(sourceTextField.selectionBeginIndex,
                                    sourceTextField.selectionEndIndex);
            }
            else
            {
                return sourceStr;
            }
        }
    }
}
```

Flex 示例

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="326" height="330"
applicationComplete="init()">
    <mx:Script>
    <![CDATA[
    import flash.desktop.Clipboard;
    import flash.desktop.ClipboardFormats;

    public function init():void
    {
        destination.addEventListener("paste", doPaste);
    }

    public function doCopy():void
    {
        Clipboard.generalClipboard.clear();
        Clipboard.generalClipboard.setDataHandler(ClipboardFormats.TEXT_FORMAT, renderData);
    }
    public function doPaste(event:Event):void
    {
        destination.text = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT).toString;
    }

    public function renderData():String{
        trace("Rendering data");
        return source.text;
    }
    ]]>
    </mx:Script>
    <mx:Label x="10" y="10" text="Source"/>
    <mx:TextArea id="source" x="10" y="36" width="300" height="100">
        <mx:text>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit.</mx:text>
    </mx:TextArea>
    <mx:Label x="10" y="181" text="Destination"/>
    <mx:TextArea id="destination"  x="12" y="207" width="300" height="100"/>
    <mx:Button click="doCopy();" x="91" y="156" label="Copy"/>
</mx:Application>
```

第 34 章：加速计输入

Flash Player 10.1 和更高版本， **Adobe AIR 2** 和更高版本

Accelerometer 类根据由设备的运动传感器检测的活动调度事件。此数据表示设备的位置或沿三维轴的移动。当设备移动时，传感器检测此移动并返回设备的加速坐标。 Accelerometer 类提供了查询是否支持加速计以及设置调度加速事件的速率的方法。

加速计轴将标准化为显示方向，而不是设备的物理方向。当设备重定向显示方向时，加速计轴也将随之重定向。因此，当用户以正常的垂直查看位置握持手机时，无论手机向哪个方向旋转， y 轴始终保持大致的垂直。如果禁用了自动方向，例如，以全屏模式在浏览器中显示 SWF 内容，则当设备旋转时，加速计轴不会随之重定向。

更多帮助主题

flash.sensors.Accelerometer

flash.events.AccelerometerEvent

检查加速计支持

使用 Accelerometer.isSupported 属性测试运行时环境是否能够使用此功能：

```
if (Accelerometer.isSupported)
{
    // Set up Accelerometer event listeners and code.
}
```

为每个 API 入口列出的运行时版本都可以访问 Accelerometer 类及其成员。但是，当前环境在运行时确定此功能的可用性。例如，您可以使用 Flash Player 10.1 的 Accelerometer 类属性编译代码，但需要使用 Accelerometer.isSupported 属性测试用户设备上 Accelerometer 功能的可用性。如果 Accelerometer.isSupported 在运行时为 true，则当前存在加速计支持。

检测加速计更改

要使用加速计传感器，请实例化 Accelerometer 对象并注册调度的 update 事件。update 事件是 Accelerometer 事件对象。此事件包含四个属性，每个都是数字：

- accelerationX — 沿 X 轴的加速，以 g’s 为度量单位。当设备位于垂直位置时，X 轴的方向是从设备左侧到右侧。（设备位于垂直位置指设备顶部朝上。）如果设备向右移动，则加速度为正。
- accelerationY — 沿 Y 轴的加速，以 g’s 为度量单位。当设备位于垂直位置时，Y 轴的方向是从设备底部到顶部。（设备位于垂直位置指设备顶部朝上。）如果设备相对于此轴上移，加速为正。
- accelerationZ — 沿 Z 轴的加速，以 g’s 为度量单位。 Z 轴取向与设备表面垂直。如果您移动设备使其面朝上，则加速度为正。如果设备面朝下，则加速度为负。
- timestamp — 自初始化运行时后事件的毫秒数。

1 g 是标准的重力加速度，大约为 9.8 米 / 秒 ^{2}。

以下是在文本字段中显示加速计数据的基本示例：

```
var accl:Accelerometer;
if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(evt:AccelerometerEvent):void
{
    accTextField.text = "acceleration X: " + evt.accelerationX.toString() + "\n"
            + "acceleration Y: " + evt.accelerationY.toString() + "\n"
            + "acceleration Z: " + evt.accelerationZ.toString()
}
```

要使用此示例，请确保创建 accTextField 文本字段并在使用此代码之前将其添加到显示列表。

您可以通过调用 Accelerometer 对象的 setRequestedUpdateInterval() 方法为加速计事件调整所需的时间间隔。此方法使用一个参数 interval，该参数是请求的更新时间间隔 （以毫秒为单位）：

```
var accl:Accelerometer;
accl = new Accelerometer();
accl.setRequestedUpdateInterval(1000);
```

加速计更新之间的实际时间可能大于或小于此值。更新间隔的任何更改都会影响所有注册侦听器。如果您不调用 setRequestedUpdateInterval() 方法，应用程序将根据设备的默认时间间隔接收更新。

加速计数据具有某种程度的不准确性。您可以使用最近数据的移动平均数来尽量消除数据误差。例如，以下示例列出了带当前读数的最近加速计读数以获得舍入结果：

```
var accl:Accelerometer;
var rollingX:Number = 0;
var rollingY:Number = 0;
var rollingZ:Number = 0;
const FACTOR:Number = 0.25;

if (Accelerometer.isSupported)
{
    accl = new Accelerometer();
    accl.setRequestedUpdateInterval(200);
    accl.addEventListener(AccelerometerEvent.UPDATE, updateHandler);
}
else
{
    accTextField.text = "Accelerometer feature not supported";
}
function updateHandler(event:AccelerometerEvent):void
{
    accelRollingAvg(event);
    accTextField.text = rollingX + "\n" +  rollingY + "\n" + rollingZ + "\n";
}

function accelRollingAvg(event:AccelerometerEvent):void
{
    rollingX = (event.accelerationX * FACTOR) + (rollingX * (1 - FACTOR));
    rollingY = (event.accelerationY * FACTOR) + (rollingY * (1 - FACTOR));
    rollingZ = (event.accelerationZ * FACTOR) + (rollingZ * (1 - FACTOR));
}
```

但是，如果加速计更新时间间隔很小，此移动平均数只是理想值。

第 35 章 : AIR 中的拖放

Adobe AIR 1.0 和更高版本

使用 Adobe® AIR™ 拖放 API 中的类可以支持在用户界面上执行拖放手势。此处所说的手势 是指由用户通过操作系统和应用程序执行的一种操作，表示要复制、移动或链接信息。当用户将对象拖出组件或应用程序时，执行的就是拖出 手势。当用户从组件或应用程序外部拖入对象时，执行的就是拖入 手势。

利用拖放 API，可以允许用户在应用程序之间以及在应用程序内的组件之间拖动数据。支持的传输格式包括：

- 位图
- 文件
- HTML 格式的文本
- 文本
- RTF 数据
- URL
- 文件释放
- 序列化对象
- 对象引用 （仅在源应用程序内有效）

AIR 中拖放的基础知识

Adobe AIR 1.0 和更高版本

有关在 AIR 应用程序中使用拖放操作的快速介绍和代码示例，请参阅 Adobe Developer Connection 中的以下快速入门文章：

- 支持拖放及复制和粘贴 (Flex)
- 支持拖放及复制和粘贴 (Flash)

拖放 API 包含以下类。

包	类
flash.desktop	• NativeDragManager • NativeDragOptions • Clipboard • URLFilePromise • IFilePromise 在以下类中定义了与拖放 API 搭配使用的常量： • NativeDragActions • ClipboardFormat • ClipboardTransferModes
flash.events	NativeDragEvent

拖放手势包含的阶段
拖放手势分成以下三个阶段：

启动 用户通过按住鼠标按键从组件或组件中的项目进行拖动，启动拖放操作。作为被拖动项目的来源的组件通常称为拖动启动器，它会调度 nativeDragStart 和 nativeDragComplete 事件。Adobe AIR 应用程序通过调用 NativeDragManager.doDrag() 方法响应 mouseDown 或 mouseMove 事件，来启动拖动操作。

如果从 AIR 应用程序之外启动拖动操作，则没有启动器对象会调度 nativeDragStart 或 nativeDragComplete 事件。

拖动 用户在按住鼠标按键的同时，将鼠标光标移至其他组件、应用程序，或移至桌面。只要拖动操作仍在进行，启动器对象就会调度 nativeDragUpdate 事件。（但是，对于 Linux，则不会在 AIR 中调度此事件。）当用户将鼠标移至 AIR 应用程序中可能的放置目标上方时，此放置目标会调度 nativeDragEnter 事件。事件处理函数会检查此事件对象，以确定所拖动的数据是否是以此目标接受的格式提供的，如果是，则会通过调用 NativeDragManager.acceptDragDrop() 方法允许用户将数据放在此目标上。

只要拖动手势仍在交互式对象上方，此对象就会调度 nativeDragOver 事件。当拖动手势离开此交互式对象时，此对象会调度 nativeDragExit 事件。

放置 用户在符合条件的放置目标上释放鼠标。如果目标是 AIR 应用程序或组件，则目标对象会调度 nativeDragDrop 事件。事件处理函数可以从事件对象访问所传输的数据。如果此目标在 AIR 外部，则由操作系统或其他应用程序来处理放置操作。在上述两种情况下，启动对象都会调度 nativeDragComplete 事件 （如果拖动是从 AIR 内部启动的）。

NativeDragManager 类既控制拖入手势，也控制拖出手势。 NativeDragManager 类的所有成员均为静态成员，所以请勿创建此类的实例。

Clipboard 对象

拖入或拖出应用程序或组件的数据包含在 Clipboard 对象中。单个 Clipboard 对象可以提供相同信息的不同表示形式，以便增加其他应用程序能够理解和使用这些数据的可能性。例如，图像可以以图像数据形式、序列化的 Bitmap 对象形式和文件形式包含在其中。以某种格式呈现这些数据的操作可以由到读取这些数据时才调用的呈现函数来处理。

启动拖动手势后，只能从 nativeDragEnter、 nativeDragOver 和 nativeDragDrop 事件的事件处理函数内访问 Clipboard 对象。拖动手势终止后，便无法读取和重用 Clipboard 对象。

应用程序对象可以作为引用和序列化对象进行传输。引用只在源应用程序中有效。在 AIR 应用程序之间是可以进行序列化对象传输的，但是，只能对在进行序列化和反序列化后仍然有效的对象进行这种传输。经过序列化的对象将转换成 ActionScript 3 的 Action Message Format (AMF3)，这是一种基于字符串的数据传输格式。

使用 **Flex** 框架

大多数情况下，在构建 Flex 应用程序时最好使用 Adobe® Flex™ 拖放 API。在 AIR 中运行 Flex 应用程序时，Flex 框架提供同等的功能集（它在内部使用 AIR NativeDragManager）。当应用程序或组件在限制性更强的浏览器环境中运行时，Flex 也会保留一个更有限的功能集。在 AIR 运行时环境外部运行的组件或应用程序中，不能使用 AIR 类。

支持拖出手势

Adobe AIR 1.0 和更高版本

若要支持拖出手势，必须创建一个 Clipboard 对象来响应 mouseDown 事件，并将此对象发送给 NativeDragManager.doDrag() 方法。这样，应用程序就可以侦听启动对象上发生的 nativeDragComplete 事件，以决定在用户完成或放弃此手势时要采取何种动作。

准备要传输的数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要准备要拖动的数据或对象，请创建一个 Clipboard 对象，然后以一种或多种格式添加要传输的信息。可以使用标准数据格式来传递可自动转换成本机剪贴板格式的数据，使用应用程序定义的格式来传递对象。

如果将要传输的信息转换为某种特定格式需要占用很大的计算开销，则可以提供用于执行这种转换的处理函数的名称。当且仅当接收组件或应用程序读取关联的格式时，才会调用此函数。

有关剪贴板格式的详细信息，请参阅第 512 页的 “ 剪贴板数据格式 ”。

下面的示例说明如何创建包含一个采用多种格式 （Bitmap 对象、本机位图格式，以及包含最初从中加载位图的文件的文件列表格式）的位图的 Clipboard 对象：

```
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
public function createClipboard(image:Bitmap, sourceFile:File):Clipboard{
    var transfer:Clipboard = new Clipboard();
    transfer.setData("CUSTOM_BITMAP", image, true); //Flash object by value and by reference
    transfer.setData(ClipboardFormats.BITMAP_FORMAT, image.bitmapData, false);
    transfer.setData(ClipboardFormats.FILE_LIST_FORMAT, new Array(sourceFile), false);
    return transfer;
}
```

启动拖出操作

Adobe AIR 1.0 和更高版本

若要启动拖动操作，请调用 NativeDragManager.doDrag() 方法来响应鼠标按下事件。doDrag() 方法是一个静态方法，它带有以下参数：

参数	说明
initiator	从中发起拖动操作且调度 dragStart 和 dragComplete 事件的对象。 initiator 必须为交互式对象。
clipboard	包含要传输的数据的 Clipboard 对象。此 Clipboard 对象在执行拖放序列期间调度的 NativeDragEvent 对象中得到引用。
dragImage	（可选）要在拖动期间显示的 BitmapData 对象。此图像可以指定 alpha 值。（注意：Microsoft Windows 始终对拖动图像应用固定的 alpha 淡化）。
offset	（可选）指定拖动图像相对于鼠标热点的偏移量的 Point 对象。使用负坐标可相对于鼠标光标向左上方移动拖动图像。如果未提供偏移量，则拖动图像的左上角将放在鼠标热点位置。
actionsAllowed	（可选）指定哪些动作 （复制、移动或链接）对拖动操作有效的 NativeDragOptions 对象。如果未提供参数，则允许所有动作。在 NativeDragEvent 对象中引用 DragOptions 对象，以使潜在的拖动目标能够检查允许的动作是否与目标组件的用途一致。例如，“垃圾桶”组件可能只接受允许移动动作的拖动手势。

下面的示例说明如何对从文件加载的位图对象启动拖动操作。此示例加载一个图像，并在发生 mouseDown 事件时启动拖动操作。

```
package
{
import flash.desktop.NativeDragManager;
import mx.core.UIComponent;
import flash.display.Sprite;
import flash.display.Loader;
import flash.system.LoaderContext;
import flash.net.URLRequest;
import flash.geom.Point;
import flash.desktop.Clipboard;
import flash.display.Bitmap;
import flash.filesystem.File;
import flash.events.Event;
import flash.events.MouseEvent;

public class DragOutExample extends UIComponent Sprite {
    protected var fileURL:String = "app:/image.jpg";
    protected var display:Bitmap;

    private function init():void {
        loadImage();
    }
    private function onMouseDown(event:MouseEvent):void {
        var bitmapFile:File = new File(fileURL);
        var transferObject:Clipboard = createClipboard(display, bitmapFile);
        NativeDragManager.doDrag(this,
                            transferObject,
                            display.bitmapData,
                            new Point(-mouseX,-mouseY));
    }
    public function createClipboard(image:Bitmap, sourceFile:File):Clipboard {
        var transfer:Clipboard = new Clipboard();
        transfer.setData("bitmap",
                            image,
                            true);
                            // ActionScript 3 Bitmap object by value and by reference
        transfer.setData(ClipboardFormats.BITMAP_FORMAT,
                            image.bitmapData,
                            false);
                            // Standard BitmapData format
        transfer.setData(ClipboardFormats.FILE_LIST_FORMAT,
```

```
                            new Array(sourceFile),
                            false);
                            // Standard file list format
        return transfer;
    }
    private function loadImage():void {
        var url:URLRequest = new URLRequest(fileURL);
        var loader:Loader = new Loader();
        loader.load(url,new LoaderContext());
        loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onLoadComplete);
    }
    private function onLoadComplete(event:Event):void {
        display = event.target.loader.content;
        var flexWrapper:UIComponent = new UIComponent();
        flexWrapper.addChild(event.target.loader.content);
        addChild(flexWrapper);
        flexWrapper.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
    }
}
}
```

完成拖出传输

Adobe AIR 1.0 和更高版本

当用户通过释放鼠标放置拖动的项目时，启动器对象会调度 nativeDragComplete 事件。您可以检查事件对象的 dropAction 属性，然后采取适当的动作。例如，如果动作是 NativeDragAction.MOVE，您可以将源项目从其原始位置删除。用户可以放弃拖动手势，方法是：当光标位于符合条件的放置目标外部时，释放鼠标按键。拖动管理器将已放弃的手势的 dropAction 属性设置为 NativeDragAction.NONE。

支持拖入手势

Adobe AIR 1.0 和更高版本

若要支持拖入手势，应用程序 （更多情况下是应用程序的可视组件）必须响应 nativeDragEnter 或 nativeDragOver 事件。

典型放置操作包含的步骤

Adobe AIR 1.0 和更高版本

放置操作通常会涉及以下事件序列：

1. 用户将一个 Clipboard 对象拖到一个组件上方。
2. 此组件调度 nativeDragEnter 事件。
3. nativeDragEnter 事件处理函数检查事件对象，以查看可用的数据格式和允许的动作。如果此组件能够处理放置操作，它会调用 NativeDragManager.acceptDragDrop()。
4. NativeDragManager 更改鼠标光标，以指示可以放置此对象。
5. 用户将此对象放在此组件上。
6. 接收组件调度 nativeDragDrop 事件。
7. 接收组件从事件对象内的 Clipboard 对象中读取所需格式的数据。

8 如果拖动手势是在 AIR 应用程序内发起的，则启动此动作的交互式对象会调度 nativeDragComplete 事件。如果此手势是在 AIR 外部发起的，则不发送回馈。

确认拖入手势

Adobe AIR 1.0 和更高版本

当用户将剪贴板项目拖入一个可视组件的范围内时，此组件会调度 nativeDragEnter 和 nativeDragOver 事件。为了确定此组件是否可以接受此剪贴板项目，这些事件的处理函数会检查此事件对象的 clipboard 和 allowedActions 属性。为了发出信号指示此组件可以接受放置操作，事件处理函数必须调用 NativeDragManager.acceptDragDrop() 方法，并在调用时传递对此接收组件的引用。如果有多个已注册的事件侦听器调用 acceptDragDrop() 方法，则列表中的最后一个处理函数优先。在鼠标离开接受对象的范围而触发 nativeDragExit 事件之前， acceptDragDrop() 调用将一直有效。

如果在传递给 doDrag() 的 allowedActions 参数中允许多个动作，则用户可以通过按住功能键指示在这些允许的动作中他们要执行何种动作。拖动管理器更改光标图像，以此方式告知用户在他们完成拖动后将执行何种动作。要执行的动作由 NativeDragEvent 对象的 dropAction 属性予以报告。为拖动手势设置的动作仅作为建议。传输过程中涉及的组件必须实现适当的行为。例如，若要完成移动动作，拖动启动器可能会删除拖动的项目，而放置目标则可能会添加此项目。

拖动目标可以通过设置 NativeDragManager 类的 dropAction 属性，将放置动作限定为三种可能的动作之一。如果用户试图使用键盘选择其他动作， NativeDragManager 便会显示不可用 光标。请同时在 nativeDragEnter 和 nativeDragOver 事件的处理函数中设置 dropAction 属性。

下面的示例说明了 nativeDragEnter 或 nativeDragOver 事件的事件处理函数。如果所拖动的剪贴板包含文本格式的数据，则此处理函数仅接受拖入手势。

```
import flash.desktop.NativeDragManager;
import flash.events.NativeDragEvent;

public function onDragIn(event:NativeDragEvent):void{
    NativeDragManager.dropAction = NativeDragActions.MOVE;
    if(event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
        NativeDragManager.acceptDragDrop(this); //'this' is the receiving component
    }
}
```

完成放置

Adobe AIR 1.0 和更高版本

当用户将拖动的项目放置到已接受相应手势的交互式对象上时，此交互式对象便会调度 nativeDragDrop 事件。此事件的处理函数可以从事件对象的 clipboard 属性中提取数据。

当剪贴板包含应用程序定义的格式时，传递给 Clipboard 对象的 getData() 方法的 transferMode 参数决定拖动管理器是返回对象的引用还是序列化版本。

下面的示例展示了 nativeDragDrop 事件的事件处理函数：

```
 import flash.desktop.Clipboard;
import flash.events.NativeDragEvent;

public function onDrop(event:NativeDragEvent):void {
    if (event.clipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)) {
    var text:String =
        String(event.clipboard.getData(ClipboardFormats.TEXT_FORMAT,
                                       ClipboardTransferMode.ORIGINAL_PREFERRED));
}
```

事件处理函数退出后， Clipboard 对象便不再有效。如有任何试图访问此对象或其数据的行为，就会产生错误。

更新组件的可视外观

Adobe AIR 1.0 和更高版本

组件可以根据 NativeDragEvent 事件更新其可视外观。下表说明了典型组件为响应不同事件而做出的更改的类型：

事件	说明
nativeDragStart	启动操作的交互式对象可以使用 nativeDragStart 事件来提供可视回馈，以说明拖动手势已从该交互式对象发起。
nativeDragUpdate	启动操作的交互式对象可以在手势执行过程中使用 nativeDragUpdate 事件更新其状态。（对于 Linux， AIR 中不存在此事件。）
nativeDragEnter	潜在的接收交互式对象可以使用此事件获取焦点，或者以可视方式指示其可以或不可以接受放置。
nativeDragOver	潜在的接收交互式对象可以使用此事件来响应鼠标在此交互式对象内的移动，例如，当鼠标进入诸如路线图显示这样的复杂组件的 “ 热 ” 区域时。
nativeDragExit	潜在的接收交互式对象可以在拖动手势移出其范围时，使用此事件来还原其状态。
nativeDragComplete	发起操作的交互式对象可以使用此事件更新其关联的数据模型 （例如通过从列表中删除项目）和还原其可视状态。

在执行拖入手势期间跟踪鼠标位置

Adobe AIR 1.0 和更高版本

当拖动手势仍然在组件上方时，此组件便会调度 nativeDragOver 事件。每隔几毫秒就会调度这些事件；此外，只要鼠标移动，就会调度这些事件。 nativeDragOver 事件对象可用于确定鼠标在组件上方的位置。如果接收组件比较复杂，但不是由若干子组件构成的，在这种情况下，如果能访问鼠标位置，则会有帮助作用。例如，如果应用程序显示了一个包含路线图的位图，并且您希望在用户将信息拖入此图中的区域时加亮显示这些区域，则可以使用在 nativeDragOver 事件中报告的鼠标坐标来跟踪鼠标在此图中的位置。

HTML 中的拖放

Adobe AIR 1.0 和更高版本

若要将数据拖入和拖出基于 HTML 的应用程序（或者拖入和拖出 HTMLLoader 中显示的 HTML），则可以使用 HTML 拖放事件。使用 HTML 拖放 API，可以拖到和拖出 HTML 内容中的 DOM 元素。

注：还可以通过侦听包含 HTML 内容的 HTMLLoader 对象上发生的事件，来使用 AIR NativeDragEvent 和 NativeDragManager API。但是， HTML API 更好地与 HTML DOM 集成到了一起，因而您可以控制默认行为。

默认的拖放行为

Adobe AIR 1.0 和更高版本

HTML 环境为涉及文本、图像和 URL 的拖放手势提供了默认行为。利用默认行为，始终都可以将这些类型的数据拖出元素。但是，只能将文本拖入元素，并且只能拖到页面的可编辑区域内的元素。在页面的可编辑区域之间或可编辑区域内部拖动文本时，默认行为会执行移动动作。从不可编辑的区域或应用程序外部向可编辑区域拖动文本时，默认行为则会执行复制动作。

可以通过自行处理拖放事件来覆盖默认行为。若要取消默认行为，必须调用为拖放事件调度的对象的 preventDefault() 方法。这样，您就可以根据需要将数据插入放置目标以及从拖动源中删除数据，以便执行所选的动作。

默认情况下，用户可以选择和拖动任何文本，并可以拖动图像和链接。可以使用 WebKit CSS 属性 -webkit-user-select 来控制任何 HTML 元素的选择方式。例如，如果将 -webkit-user-select 设置为 none，则元素内容是不可选择的，因而也无法拖动。还可以使用 -webkit-user-drag CSS 属性控制元素作为一个整体是否可以拖动。不过，元素的内容则单独处理。用户仍可以拖动文本的选定部分。有关详细信息，请参阅 第 834 页的 “AIR 中的 CSS”。

HTML 中的拖放事件

Adobe AIR 1.0 和更高版本

从中发起拖动操作的启动器元素调度的事件有：

事件	说明
dragstart	在用户启动拖动手势时调度。如有必要，此事件的处理函数可以通过调用事件对象的 preventDefault() 方法来阻止拖动。若要控制是否可以复制、链接或移动所拖动的数据，请设置 effectAllowed 属性。所选的文本、图像和链接由默认行为放置到剪贴板上，但您可以使用事件对象的 dataTransfer 属性为拖动手势设置其他数据。
drag	在执行拖动手势期间持续调度
dragend	在用户释放鼠标按键终止拖动手势时调度。

由拖动目标调度的事件有：

事件	说明
dragover	当拖动手势仍然在元素边界内时持续调度。此事件的处理函数应设置 dataTransfer.dropEffect 属性，以指示在用户释放鼠标时放置操作是否会导致复制、移动或链接动作。
dragenter	在拖动手势进入元素的边界内时调度。 如果在 dragenter 事件处理函数中更改 dataTransfer 对象的任何属性，则这些更改将很快被下一个 dragover 事件覆盖。另一方面，在 dragenter 与第一个 dragover 事件之间有一个短暂的延迟，如果设置了不同的属性，此延迟可导致光标闪烁。在很多情况下，都可以对这两个事件使用相同的事件处理函数。
dragleave	在拖动手势离开元素边界时调度。
drop	在用户将数据放到元素上时调度。只能在此事件的处理函数内访问所拖动的数据。

为响应这些事件而调度的事件对象与鼠标事件类似。可以使用诸如 (clientX, clientY) 和 (screenX, screenY) 等鼠标事件属性来确定鼠标位置。

拖动事件对象最重要的属性是 dataTransfer，此属性包含被拖动的数据。 dataTransfer 对象本身具有以下属性和方法：

属性或方法	说明
effectAllowed	拖动源允许的效果。通常情况下， dragstart 事件的处理函数设置此值。请参阅 第 527 页的 “HTML 中的拖动效果 ”。
dropEffect	由目标或用户选择的效果。如果在 dragover 或 dragenter 事件处理函数中设置 dropEffect，则 AIR 会更新鼠标光标，以指示在用户释放鼠标时出现的效果。如果允许的效果中没有一种效果与 dropEffect 设置匹配，则不允许放置，并显示不可用 光标。如果尚未设置 dropEffect 来响应最新的 dragover 或 dragenter 事件，则用户可以使用标准的操作系统功能键从允许的效果中进行选择。 最终的效果由为 dragend 调度的对象的 dropEffect 属性报告。如果用户通过在符合条件的目标外部释放鼠标放弃放置操作，则 dropEffect 将设置为 none。
类型	一个数组，其中包含了 dataTransfer 对象中存在的每种数据格式的 MIME 类型字符串。
getData(mimeType)	以 mimeType 参数指定的格式获取数据。 只能为响应 drop 事件调用 getData() 方法。

属性或方法	说明
setData(mimeType)	以 mimeType 参数指定的格式向 dataTransfer 添加数据。可以通过对每种 MIME 类型调用 setData()，以多种格式添加数据。将清除由默认拖动行为放入 dataTransfer 对象的所有数据。 只能为响应 dragstart 事件调用 setData() 方法。
clearData(mimeType)	清除采用 mimeType 参数指定的格式的所有数据。
setDragImage(image, offsetX, offsetY)	设置自定义拖动图像。只能为响应 dragstart 事件，且只有当拖动整个 HTML 元素 （通过将 -webkit-user-drag CSS 样式设置为 element）时，才可以调用 setDragImage() 方法。 image 参数可以是 JavaScript 元素或 Image 对象。

用于 HTML 拖放的 MIME 类型

Adobe AIR 1.0 和更高版本

与 HTML 拖放事件的 dataTransfer 对象一起使用的 MIME 类型包括：

数据格式	MIME 类型
文本	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
位图	"image/x-vnd.adobe.air.bitmap"
文件列表	"application/x-vnd.adobe.air.file-list"

还可以使用其他 MIME 字符串，包括应用程序定义的字符串。但是，其他应用程序可能无法识别或使用所传输的数据。以所需格式向 dataTransfer 对象添加数据的工作由您负责完成。

重要说明：只有在应用程序沙箱中运行的代码才能访问放置的文件。如果试图在非应用程序沙箱内读取或设置 File 对象的任何属性，则会产生安全错误。有关详细信息，请参阅第 531 页的 “ 在非应用程序 HTML 沙箱中处理文件放置 ”。

HTML 中的拖动效果

Adobe AIR 1.0 和更高版本

拖动手势的启动器可以通过在 dragstart 事件的处理函数中设置 dataTransfer.effectAllowed 属性，来限制允许的拖动效果。可以使用以下字符串值：

字符串值	说明
"none"	不允许任何拖动操作。
"copy"	将数据复制到目标位置，并保留原始数据的位置不变。
"link"	使用指回原始数据的链接与放置目标共享数据。
"move"	将数据复制到目标，并将其从原始位置删除。
"copyLink"	可以复制数据，也可链接数据。
"copyMove"	可以复制数据，也可移动数据。
"linkMove"	可以链接数据，也可移动数据。
"all"	可以复制数据、移动数据，也可链接数据。当您阻止默认行为时， All 是默认效果。

拖动手势的目标可以设置 dataTransfer.dropEffect 属性，以指示在用户完成放置后采取的动作。如果放置效果是允许的动作之一，则系统将显示相应的复制、移动或链接光标。如果不是，则系统将显示不可用 光标。如果目标未设置放置效果，则用户可以使用功能键从允许的动作中进行选择。

同时在 dragover 和 dragenter 事件的处理函数中设置 dropEffect 值：

```
function doDragStart(event) {
    event.dataTransfer.setData("text/plain","Text to drag");
    event.dataTransfer.effectAllowed = "copyMove";
}

function doDragOver(event) {
    event.dataTransfer.dropEffect = "copy";
}

function doDragEnter(event) {
    event.dataTransfer.dropEffect = "copy";
}
```

注：尽管您始终都应在 dragenter 的处理函数中设置 dropEffect 属性，但要注意，下一个 dragover 事件会将此属性重置为其默认值。设置 dropEffect 以响应这两个事件。

将数据拖出 HTML 元素

Adobe AIR 1.0 和更高版本

默认行为允许以拖动方式复制 HTML 页面中的大部分内容。可以使用 CSS 属性 -webkit-user-select 和 -webkit-user-drag 来控制允许拖动的内容。

在 dragstart 事件的处理函数中覆盖默认的拖出行为。调用事件对象的 dataTransfer 属性的 setData() 方法，以便将您自己的数据放入拖动手势。

若要指示在不依赖默认行为时源对象支持的拖动效果，请设置为 dragstart 事件调度的事件对象的 dataTransfer.effectAllowed 属性。您可以选择任意效果组合。例如，如果源元素既支持复制 效果，也支持链接 效果，则请将此属性设置为 "copyLink"。

设置拖动的数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 dataTransfer 属性在 dragstart 事件的处理函数中为拖动手势添加数据。使用 dataTransfer.setData() 方法将数据放到剪贴板上，同时传入 MIME 类型和要传输的数据。

例如，如果应用程序中有一个 ID 为 imageOfGeorge 的图像元素，则可以使用下面的 dragstart 事件处理函数。此示例以多种数据格式添加 George 照片的表示形式，从而增加了其他应用程序能够使用拖动的数据的可能性。

```
function dragStartHandler(event){
    event.dataTransfer.effectAllowed = "copy";

    var dragImage = document.getElementById("imageOfGeorge");
    var dragFile = new air.File(dragImage.src);
    event.dataTransfer.setData("text/plain","A picture of George");
    event.dataTransfer.setData("image/x-vnd.adobe.air.bitmap", dragImage);
    event.dataTransfer.setData("application/x-vnd.adobe.air.file-list",
                                new Array(dragFile));
}
```

注：调用 dataTransfer 对象的 setData() 方法时，默认拖放行为不会添加任何数据。

将数据拖入 HTML 元素

Adobe AIR 1.0 和更高版本

默认行为只允许将文本拖入页面的可编辑区域中。可以通过在元素的开始标签中包含 contenteditable 属性，指定可以使元素及其子级可编辑。还可以通过将文档对象的 designMode 属性设置为 "on"，使整个文档可编辑。

可以通过处理可接受所拖动数据的任何元素的 dragenter、 dragover 和 drop 事件，支持页面上的替代拖入行为。

允许拖入

Adobe AIR 1.0 和更高版本

若要处理拖入手势，必须先取消默认行为。侦听您要用作放置目标的任何 HTML 元素上发生的 dragenter 和 dragover 事件。在这些事件的处理函数中，调用调度的事件对象的 preventDefault() 方法。如果取消默认行为，则会允许不可编辑的区域接收放置。

获取放置的数据

Adobe AIR 1.0 和更高版本

可以在 ondrop 事件的处理函数中访问放置的数据：

```
function doDrop(event){
    droppedText = event.dataTransfer.getData("text/plain");
}
```

使用 dataTransfer.getData() 方法将数据读到剪贴板上，同时传入要读取的数据格式的 MIME 类型。可以使用 dataTransfer 对象的 types 属性查明哪些数据格式可用。 types 数组包含每种可用格式的 MIME 类型字符串。

取消 dragenter 或 dragover 事件中的默认行为后，须由您将放置的任何数据插入到其在文档中的正确位置。不存在可将鼠标位置转换成元素内的插入点的 API。由于存在这一限制，因而可能很难实现插入类型的拖动手势。

示例：覆盖默认的 HTML 拖入行为

Adobe AIR 1.0 和更高版本

此示例实现一个放置目标，此目标显示了一个表，表中显示放置的项目中可用的每种数据格式。

默认行为用于允许在应用程序内拖动文本、链接和图像。此示例覆盖用作放置目标的 div 元素的默认拖入行为。使不可编辑的内容能接受拖入手势的关键步骤，就是调用为 dragenter 和 dragover 事件调度的事件对象的 preventDefault() 方法。为响应 drop 事件，处理函数将传输的数据转换成 HTML 行元素，然后将该行插入表中以进行显示。

```
 <html>
<head>
<title>Drag-and-drop</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    function init(){
        var target = document.getElementById('target');
        target.addEventListener("dragenter", dragEnterOverHandler);
        target.addEventListener("dragover", dragEnterOverHandler);
        target.addEventListener("drop", dropHandler);

        var source = document.getElementById('source');
        source.addEventListener("dragstart", dragStartHandler);
        source.addEventListener("dragend", dragEndHandler);

        emptyRow = document.getElementById("emptyTargetRow");
    }

    function dragStartHandler(event){
        event.dataTransfer.effectAllowed = "copy";
    }

    function dragEndHandler(event){
        air.trace(event.type + ": " + event.dataTransfer.dropEffect);
    }

    function dragEnterOverHandler(event){
        event.preventDefault();
    }

    var emptyRow;
    function dropHandler(event){
        for(var prop in event){
            air.trace(prop + " = " + event[prop]);
        }
        var row = document.createElement('tr');
        row.innerHTML = "<td>" + event.dataTransfer.getData("text/plain") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/html") + "</td>" +
            "<td>" + event.dataTransfer.getData("text/uri-list") + "</td>" +
            "<td>" + event.dataTransfer.getData("application/x-vnd.adobe.air.file-list") +
            "</td>";

        var imageCell = document.createElement('td');
        if((event.dataTransfer.types.toString()).search("image/x-vnd.adobe.air.bitmap") > -1){
            imageCell.appendChild(event.dataTransfer.getData("image/x-vnd.adobe.air.bitmap"));
        }
        row.appendChild(imageCell);
        var parent = emptyRow.parentNode;
        parent.insertBefore(row, emptyRow);
    }
</script>
</head>
<body onLoad="init()" style="padding:5px">
<div>
    <h1>Source</h1>
       <p>Items to drag:</p>
    <ul id="source">
        <li>Plain text.</li>
           <li>HTML <b>formatted</b> text.</li>
           <li>A <a href="http://www.adobe.com">URL.</a></li>
           <li><img src="icons/AIRApp_16.png" alt="An image"/></li>
           <li style="-webkit-user-drag:none;">
```

```
        Uses "-webkit-user-drag:none" style.
           </li>
           <li style="-webkit-user-select:none;">
        Uses "-webkit-user-select:none" style.
           </li>

       </ul>
</div>
<div id="target" style="border-style:dashed;">
    <h1 >Target</h1>
       <p>Drag items from the source list (or elsewhere).</p>
       <table id="displayTable" border="1">
        <tr><th>Plain text</th><th>Html text</th><th>URL</th><th>File list</th><th>Bitmap Data</th></tr>
           <tr
id="emptyTargetRow"><td> </td><td> </td><td> </td><td> </td><td> </td></tr>
       </table>
       </div>
</div>
</body>
</html>
```

在非应用程序 HTML 沙箱中处理文件放置

Adobe AIR 1.0 和更高版本

非应用程序内容无法访问在将文件拖入 AIR 应用程序时产生的 File 对象。也无法将这些 File 对象中的一个对象通过沙箱桥传递给应用程序内容。（必须在序列化期间访问对象属性。）但是，您仍可以通过侦听 HTMLLoader 对象上发生的 AIR nativeDragDrop 事件，在应用程序中放置文件。

通常，如果用户将文件放入承载非应用程序内容的框架中，则放置事件不会从子级传播到父级。但是，由于 HTMLLoader（AIR 应用程序中所有 HTML 内容的容器）调度的事件不是 HTML 事件流的一部分，因此您仍可以在应用程序内容中接收放置事件。

为了接收文件放置事件，父级文档会使用 window.htmlLoader 提供的引用向 HTMLLoader 对象添加一个事件侦听器：

```
window.htmlLoader.addEventListener("nativeDragDrop",function(event){
    var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
    air.trace(filelist[0].url);
});
```

下面的示例使用将子级页面加载到远程沙箱 (http://localhost/) 中的父级文档：父级侦听 HTMLLoader 对象上发生的 nativeDragDrop 事件，并输出文件 URL。

```
<html>
<head>
<title>Drag-and-drop in a remote sandbox</title>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>
<script language="javascript">
    window.htmlLoader.addEventListener("nativeDragDrop",function(event){
        var filelist = event.clipboard.getData(air.ClipboardFormats.FILE_LIST_FORMAT);
        air.trace(filelist[0].url);
    });
</script>
</head>
<body>
       <iframe src="child.html"
               sandboxRoot="http://localhost/"
               documentRoot="app:/"
               frameBorder="0"  width="100%" height="100%">
       </iframe>
</body>
</html>
```

子级文档必须通过在 HTML dragenter 和 dragover 事件处理函数中调用 Event 对象的 preventDefault() 方法来提供有效的放置目标。否则，放置事件绝对不会发生。

```
<html>
<head>
    <title>Drag and drop target</title>
    <script language="javascript" type="text/javascript">
        function preventDefault(event){
            event.preventDefault();
        }
    </script>
</head>
<body ondragenter="preventDefault(event)" ondragover="preventDefault(event)">
<div>
<h1>Drop Files Here</h1>
</div>
</body>
</html>
```

放置文件释放

Adobe AIR 2 和更高版本

文件释放是一种拖放剪贴板格式，这种格式允许用户将尚不存在的文件拖出 AIR 应用程序外。例如，使用文件释放，您的应用程序使用户可将代理图标拖动到桌面文件夹中。代理图标表示 URL 上已知并可用的文件或一些数据。在用户放置图标后，运行时将下载数据并将文件写入放置位置。

可使用 AIR 应用程序中的 URLFilePromise 类来到拖放 URL 上可访问的文件。在 aircore 库中， URLFilePromise 实现作为 AIR 2 SDK 的一部分提供。使用包含在 SDK frameworks/libs/air 目录中的 aircore.swc 或 aircore.swf 文件。

或者，您可以使用 IFilePromise 接口 （在运行时 flash.desktop 包中定义）实现自己的文件承诺逻辑。

文件释放在概念上类似于在剪贴板上使用数据处理函数的延迟呈现。在拖放文件时使用文件释放而不使用延迟呈现。当生成或下载数据时，使用延迟呈现技术可能会导致拖动手势出现不需要的暂停。使用延迟呈现执行复制粘贴操作 （文件释放不支持此操作）。

使用文件释放时的限制

与您可以放在拖放剪贴板中的其他数据格式相比，文件释放有以下限制：

- 文件释放只能拖出 AIR 应用程序；而不能拖入 AIR 应用程序中。
- 并非所有操作系统都不支持文件释放。使用 Clipboard.supportsFilePromise 属性测试主机系统是否支持文件释放。在不支持文件释放的系统中，您应提供替代机制，以便下载或生成文件数据。
- 文件释放不能与复制粘贴剪贴板 (Clipboard.generalClipboard) 一起使用。

更多帮助主题

flash.desktop.IFilePromise

air.desktop.URLFilePromise

放置远程文件

Adobe AIR 2 和更高版本

使用 URLFilePromise 类创建表示 URL 上可用文件或数据的文件释放对象。使用 FILE_PROMISE_LIST 剪贴板格式将一个或多个文件释放对象添加到剪贴板。在以下示例中，从 http://www.example.com/foo.txt 下载一个单独的文件，并作为 bar.txt 保存到放置位置。（远程和本地文件名不必匹配。）

```
if( Clipboard.supportsFilePromise )
{
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/foo.txt");
    filePromise.relativePath = "bar.txt";

    var fileList:Array = new Array( filePromise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

通过将多个文件释放对象添加到分配给剪贴板的数组，您可以允许用户一次拖动多个文件。您还可以在 relativePath 属性中指定子目录，以便将操作中包括的部分文件或所有文件放置到子文件夹 （相对于放置位置）中。

以下示例演示如何启动包括多个文件释放的拖动操作。在此示例中，HTML 页面 article.html 作为文件释放与两个链接的图像文件一起放置在剪贴板上。这些图像被复制到 images 子文件夹中，以便保持相对链接。

```
if( Clipboard.supportsFilePromise )
{   //Create the promise objects
    var filePromise:URLFilePromise = new URLFilePromise();
    filePromise.request = new URLRequest("http://example.com/article.html");
    filePromise.relativePath = "article.html";

    var image1Promise:URLFilePromise = new URLFilePromise();
    image1Promise.request = new URLRequest("http://example.com/images/img_1.jpg");
    image1Promise.relativePath = "images/img_1.html";
    var image2Promise:URLFilePromise = new URLFilePromise();
    image2Promise.request = new URLRequest("http://example.com/images/img_2.jpg");
    image2Promise.relativePath = "images/img_2.jpg";


    //Put the promise objects onto the clipboard inside an array
    var fileList:Array = new Array( filePromise, image1Promise, image2Promise );
    var clipboard:Clipboard = new Clipboard();
    clipboard.setData( ClipboardFormats.FILE_PROMISE_LIST_FORMAT, fileList );
    //Start the drag operation
    NativeDragManager.doDrag( dragSource, clipboard );
}
```

实现 IFilePromise 接口

Adobe AIR 2 和更高版本

若要为不能使用 URLFilePromise 对象访问的资源提供文件释放，可以在自定义类中实现 IFilePromise 接口。一旦放置文件释放后， IFilePromise 接口就会定义由 AIR 运行时访问要写入到文件中的数据所用的方法和属性。

IFilePromise 实现将为该文件释放提供数据的另一个对象传递到 AIR 运行时。此对象必须实现 AIR 运行时用于读取数据的 IDataInput 接口。例如，可实现 IFilePromise 的 URLFilePromise 类使用 URLStream 对象作为数据提供程序。

AIR 可以同步读取数据，也可以异步读取数据。 IFilePromise 实现通过返回 isAsync 属性中的相应值来报告支持哪种访问模式。如果提供异步数据访问，则数据提供程序对象必须实现 IEventDispatcher 接口并调度必要的事件，例如 open、 progress 和 complete。

可使用自定义类或以下内置类之一作为文件释放的数据提供程序：

- ByteArray （同步）
- FileStream （同步或异步）
- Socket （异步）
- URLStream （异步）

要实现 IFilePromise 接口，必须为下列函数和属性提供代码：

- open():IDataInput — 从读取的已释放文件的数据返回数据提供程序对象。该对象必须实现 IDataInput 接口。如果异步提供数据，对象还必须实现 IEventDispatcher 接口并调度必要的事件 （请参阅第 536 页的 " 在文件释放中使用异步数据提供程序 "）。
- get relativePath():String — 提供已创建文件的路径，包括文件名。此路径被解析为相对于由拖放操作中用户选择的放置位置。要确保此路径使用适用于主机操作系统的适当分隔符，请在指定包含目录的路径时使用 File.separator 常量。您可以添加 setter 函数或使用构造函数参数来允许在运行时设置路径。
- get isAsync():Boolean — 通知 AIR 运行时数据提供程序对象是异步提供数据还是同步提供数据。
- close():void — 当完全读取数据时由运行时调用 （否则将出现阻止进一步读取的错误）。可使用此函数清除资源。
- reportError(e:ErrorEvent):void — 在读取数据出错时由运行时调用。

在涉及文件释放的拖放操作期间，所有 IFilePromise 方法都由运行时调用。通常，应用程序逻辑不应直接调用上述任何方法。

在文件释放中使用同步数据提供程序

Adobe AIR 2 和更高版本

实现 IFilePromise 接口最简单的方法是使用同步数据提供程序对象，例如 ByteArray 或同步 FileStream。在下面的示例中，创建一个 ByteArray 对象，用数据填充它，并在调用 open() 方法时返回。

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class SynchronousFilePromise implements IFilePromise
    {
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "SynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return false;
        }

        public function open():IDataInput
        {
            var fileContents:ByteArray = new ByteArray();

            //Create some arbitrary data for the file
            for( var i:int = 0; i < fileSize; i++ )
            {
                fileContents.writeUTFBytes( 'S' );
            }

            //Important: the ByteArray is read from the current position
            fileContents.position = 0;
            return fileContents;
        }

        public function close():void
        {
            //Nothing needs to be closed in this case.
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

实际上，同步文件释放的效用很受限。如果数据量很小，您可以在临时目录中轻松创建一个文件，将普通文件列表数组添加到拖放剪贴板。另一方面，如果数据量很大或生成数据的计算成本很高，则长的同步进程是必需的。长的同步进程会在明显较长的时间内阻止 UI 更新，使应用程序看起来像无响应一样。若要避免此问题，您可以创建一个由计时器驱动的异步数据提供程序。

在文件释放中使用异步数据提供程序

Adobe AIR 2 和更高版本

使用异步数据提供程序对象时， IFilePromise isAsync 属性必须为 true，并且由 open() 方法返回的对象必须实现 IEventDispatcher 接口。运行时侦听多个替代事件，以便可以使用不同的内置对象作为数据提供程序。例如，progress 事件由 FileStream 和 URLStream 对象调度，而 socketData 事件由 Socket 对象调度。运行时从所有这些对象侦听适当的事件。

下列事件推动从数据提供程序对象读取数据的进程：

- Event.OPEN — 通知运行时数据源已准备好。
- ProgressEvent.PROGRESS — 通知运行时数据可用。运行时将从数据提供程序对象读取可用的数据量。
- ProgressEvent.SOCKET_DATA — 通知运行时数据可用。 socketData 事件由基于 Socket 的对象调度。对于其他对象类型，您应调度 progress 事件。（运行时同时侦听这两个事件以检测何时可以读取数据。）
- Event.COMPLETE — 通知运行时已读取所有数据。
- Event.CLOSE — 通知运行时已读取所有数据。（运行时同时侦听 close 和 complete 来实现此目的。）
- IOErrorEvent.IOERROR — 通知运行时读取数据时出错。运行时终止文件创建并调用 IFilePromise close() 方法。
- SecurityErrorEvent.SECURITY_ERROR — 通知运行时出现安全性错误。运行时终止文件创建并调用 IFilePromise close() 方法。
- HTTPStatusEvent.HTTP_STATUS — 运行时同时使用它和 httpResponseStatus，以确保可用数据表示所需内容，而不会显示错误消息 （例如 404 页面）。基于 HTTP 协议的对象应调度此事件。
- HTTPStatusEvent.HTTP_RESPONSE_STATUS — 运行时同时使用它和 httpStatus，以确保可用数据表示所需内容。基于 HTTP 协议的对象应调度此事件。

数据提供程序应按照以下顺序调度这些事件：

1. open 事件
2. progress 或 socketData 事件
3. complete 或 close 事件

注：内建对象 （FileStream、 Socket 和 URLStream）自动调度相应的事件。

以下示例使用自定义异步数据提供程序创建一个文件释放。数据提供程序类扩展了 ByteArray （以实现 IDataInput 支持）并实现 IEventDispatcher 接口。在每个 timer 事件发生时，对象会生成大量数据，并调度 progress 事件以通知运行时这些数据可用。当生成的数据足够多时，对象调度 complete 事件。

```
package
{
import flash.events.Event;
import flash.events.EventDispatcher;
import flash.events.IEventDispatcher;
import flash.events.ProgressEvent;
import flash.events.TimerEvent;
import flash.utils.ByteArray;
import flash.utils.Timer;

[Event(name="open", type="flash.events.Event.OPEN")]
[Event(name="complete",  type="flash.events.Event.COMPLETE")]
[Event(name="progress", type="flash.events.ProgressEvent")]
[Event(name="ioError", type="flash.events.IOErrorEvent")]
[Event(name="securityError", type="flash.events.SecurityErrorEvent")]
public class AsyncDataProvider extends ByteArray implements IEventDispatcher
{
    private var dispatcher:EventDispatcher = new EventDispatcher();
    public var fileSize:int = 0; //The number of characters in the file
    private const chunkSize:int = 1000; //Amount of data written per event
    private var dispatchDataTimer:Timer = new Timer( 100 );
    private var opened:Boolean = false;

    public function AsyncDataProvider()
    {
        super();
        dispatchDataTimer.addEventListener( TimerEvent.TIMER, generateData );
    }

    public function begin():void{
        dispatchDataTimer.start();
    }

    public function end():void
    {
        dispatchDataTimer.stop();
    }
    private function generateData( event:Event ):void
    {
        if( !opened )
        {
            var open:Event = new Event( Event.OPEN );
            dispatchEvent( open );
            opened = true;
        }
        else if( position + chunkSize < fileSize )
        {
            for( var i:int = 0; i <= chunkSize; i++ )
            {
                writeUTFBytes( 'A' );
            }
            //Set position back to the start of the new data
            this.position -= chunkSize;
            var progress:ProgressEvent =
                new ProgressEvent( ProgressEvent.PROGRESS, false, false, bytesAvailable, bytesAvailable +
chunkSize);
            dispatchEvent( progress )
        }
        else
        {
            var complete:Event = new Event( Event.COMPLETE );
            dispatchEvent( complete );
```

```
        }
    }
     //IEventDispatcher implementation
    public function addEventListener(type:String, listener:Function, useCapture:Boolean=false,
priority:int=0, useWeakReference:Boolean=false):void
    {
        dispatcher.addEventListener( type, listener, useCapture, priority, useWeakReference );
    }

    public function removeEventListener(type:String, listener:Function, useCapture:Boolean=false):void
    {
        dispatcher.removeEventListener( type, listener, useCapture );
    }

    public function dispatchEvent(event:Event):Boolean
    {
        return dispatcher.dispatchEvent( event );
    }

    public function hasEventListener(type:String):Boolean
    {
        return dispatcher.hasEventListener( type );
    }

    public function willTrigger(type:String):Boolean
    {
        return dispatcher.willTrigger( type );
    }
}
}
```

注：由于示例中的 AsyncDataProvider 类扩展了 ByteArray，所以它无法同时扩展 EventDispatcher。为实现 IEventDispatcher 接口，该类使用内部 EventDispatcher 对象并将 IEventDispatcher 方法调用转发给该内部对象。您也可以扩展 EventDispatcher 并实现 IDataInput （或同时实现两个接口）。

异步 IFilePromise 实现与同步实现几乎是相同的。主要区别是 isAsync 返回 true，而 open() 方法返回异步数据对象：

```
package
{
    import flash.desktop.IFilePromise;
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.utils.IDataInput;

    public class AsynchronousFilePromise extends EventDispatcher implements IFilePromise
    {
        private var fileGenerator:AsyncDataProvider;
        private const fileSize:int = 5000; //size of file data
        private var filePath:String = "AsynchronousFile.txt";

        public function get relativePath():String
        {
            return filePath;
        }

        public function get isAsync():Boolean
        {
            return true;
        }
```

```
        public function open():IDataInput
        {
            fileGenerator = new AsyncDataProvider();
            fileGenerator.fileSize = fileSize;
            fileGenerator.begin();
            return fileGenerator;
        }

        public function close():void
        {
            fileGenerator.end();
        }

        public function reportError(e:ErrorEvent):void
        {
            trace("Something went wrong: " + e.errorID + " - " + e.type + ", " + e.text );
        }
    }
}
```

第 36 章：使用菜单

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在基于 Web 的 Flex 和 Flash Player 应用程序中，使用上下文菜单 API 中的类修改上下文菜单。

在 Adobe® AIR® 中，使用本机菜单 API 中的类定义应用程序、窗口、上下文和弹出菜单。

菜单基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有关在 AIR 应用程序中创建本机菜单的快速介绍和代码示例，请参阅 Adobe Developer Connection 上的以下快速入门文章：

- 向 AIR 应用程序添加本机菜单 (Flex)
- 向 AIR 应用程序添加本机菜单 (Flash)

通过本机菜单类，可以访问运行您的应用程序的操作系统的本机菜单功能。NativeMenu 对象可用于应用程序菜单（Mac OS X 中提供）、窗口菜单 （Windows 和 Linux 中提供）、上下文菜单和弹出菜单。

在 AIR 外，您可以使用上下文菜单类修改上下文菜单，当用户右键单击或按住 Cmd 键单击应用程序的中的对象时， Flash Player 会自动显示上下文菜单。（AIR 应用程序不会自动显示上下文菜单。）

菜单类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

菜单类包括：

包	类
flash.display	• NativeMenu • NativeMenuItem
flash.ui	• ContextMenu • ContextMenuItem
flash.events	• Event • ContextMenuEvent

菜单类型

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

AIR 支持以下类型的菜单：

上下文菜单 右键单击或按住 Command 键单击 SWF 内容中的交互式对象或 HTML 内容中的文档元素时，作为响应，上下文菜单会打开。

在 Flash Player 运行时中，上下文菜单自动显示。您可以使用 ContextMenu 类和 ContextMenuItem 类向此菜单中添加您自己的命令。您也可以删除一些 （非全部）内置命令。

在 AIR 运行时中，可以使用 NativeMenu 类或 ContextMenu 类创建上下文菜单。在 AIR 中的 HTML 内容中，您可以使用 Webkit HTML 和 JavaScript API 向 HTML 元素添加上下文菜单。

应用程序菜单 （仅限 AIR） 应用程序菜单是应用于整个应用程序的全局菜单。 Mac OS X 支持应用程序菜单，但 Windows 或 Linux 不支持。在 Mac OS X 上，操作系统自动创建应用程序菜单。可以使用 AIR 菜单 API 将项目和子菜单添加到标准菜单中。可以添加用于处理现有菜单命令的侦听器。还可以删除现有项目。

窗口菜单 （仅限 AIR） 窗口菜单与单个窗口关联，并显示在标题栏的下方。通过创建 NativeMenu 对象并将它分配给 NativeWindow 对象的 menu 属性，可以向窗口添加菜单。 Windows 和 Linux 操作系统支持窗口菜单，但 Mac OS X 不支持。本机窗口菜单只能与有系统镶边的窗口一起使用。

停靠栏和系统任务栏图标菜单 （仅限 AIR） 这些图标菜单类似于上下文菜单，分配给 Mac OS X 停靠栏或 Windows 和 Linux 任务栏上通知区域中的应用程序图标。停靠栏图标菜单和系统任务栏图标菜单使用 NativeMenu 类。在 Mac OS X 上，菜单中的项目添加到标准操作系统项目的上方。 Windows 或 Linux 中没有标准菜单。

弹出菜单 （仅限 AIR） AIR 弹出菜单类似于上下文菜单，但不一定与特定应用程序对象或组件关联。通过调用任何 NativeMenu 对象的 display() 方法，可以使弹出菜单在窗口中的任何位置显示。

自定义菜单 本机菜单完全由操作系统调出，因此存在于 Flash 和 HTML 呈现模型以外。您可以不使用本机菜单，而总是使用 MXML、 ActionScript 或 JavaScript 创建自己的自定义非本机菜单 （仅 AIR）。这些菜单必须在应用程序内容中完全呈现。

Flex 菜单 Adobe® Flex™ 框架提供了一组 Flex 菜单组件。 Flex 菜单都是由运行时调出，而不是操作系统来调出，而且 Flex 菜单不是本机菜单。 Flex 菜单组件可用于没有系统镶边的 Flex 窗口。使用 Flex 菜单组件的另一个好处是能够以声明方式指定 MXML 格式的菜单。如果您使用的是 Flex 框架，请对窗口菜单使用 Flex 菜单类，非不要使用本机类。

默认菜单 （仅 **AIR**）
以下默认菜单由操作系统或内置 AIR 类提供：

- Mac OS X 上的应用程序菜单
- Mac OS X 上的停靠栏图标菜单
- HTML 内容中的所选文本和图像的上下文菜单
- TextField 对象 （或扩展 TextField 的对象）中的所选文本的上下文菜单

关于上下文菜单

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 SWF 内容中，对于从 InteractiveObject 继承的任何对象，通过将菜单对象分配给该对象的 contextMenu 属性，可以为该对象指定上下文菜单。默认情况下包含的几个命令有：“ 前进 ”、“ 后退 ”、“ 打印 ”、“ 品质 ”和 “ 缩放 ”。在 AIR 运行时中，指定给 contextMenu 的菜单对象的类型可以是 NativeMenu 或 ContextMenu。在 Flash Player 运行时中，仅 ContextMenu 类可用。

可以在使用 ContextMenu 和 ContextMenuItem 类时侦听本机菜单事件或上下文菜单事件；两者都已调度。ContextMenuEvent 对象属性所具有的一个优势为：contextMenuOwner 可识别与菜单关联的对象，而 mouseTarget 可识别为打开菜单曾单击的对象。 NativeMenuEvent 对象中无此信息。

以下示例创建一个 Sprite，并添加一个简单的编辑上下文菜单：

```
 var sprite:Sprite = new Sprite();
sprite.contextMenu = createContextMenu()
private function createContextMenu():ContextMenu{
    var editContextMenu:ContextMenu = new ContextMenu();
    var cutItem:ContextMenuItem = new ContextMenuItem("Cut")
    cutItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCutCommand);
    editContextMenu.customItems.push(cutItem);

    var copyItem:ContextMenuItem = new ContextMenuItem("Copy")
    copyItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doCopyCommand);
    editContextMenu.customItems.push(copyItem);

    var pasteItem:ContextMenuItem = new ContextMenuItem("Paste")
    pasteItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT, doPasteCommand);
    editContextMenu.customItems.push(pasteItem);

    return editContextMenu
}
private function doCutCommand(event:ContextMenuEvent):void{trace("cut");}
private function doCopyCommand(event:ContextMenuEvent):void{trace("copy");}
private function doPasteCommand(event:ContextMenuEvent):void{trace("paste");}
```

注：与在浏览器环境中显示的 SWF 内容相比， AIR 中的上下文菜单没有任何内置命令。

自定义 **Flash Player** 上下文菜单

在浏览器或放映文件中，SWF 内容中的上下文菜单总包含内置项。您可以删除菜单中 “ 设置 ” 和 “ 关于 ” 命令外的所有这些默认命令。如果将 Stage 属性 showDefaultContextMenu 设置为 false，则会从上下文菜单中删除这些命令。

要为特定显示对象创建自定义的上下文菜单，请创建 ContextMenu 类的一个新实例，调用 hideBuiltInItems() 方法，并将该实例分配给该 DisplayObject 实例的 contextMenu 属性。下面的示例为一个动态绘制的正方形提供了一个上下文菜单命令，用于将其更改为随机颜色：

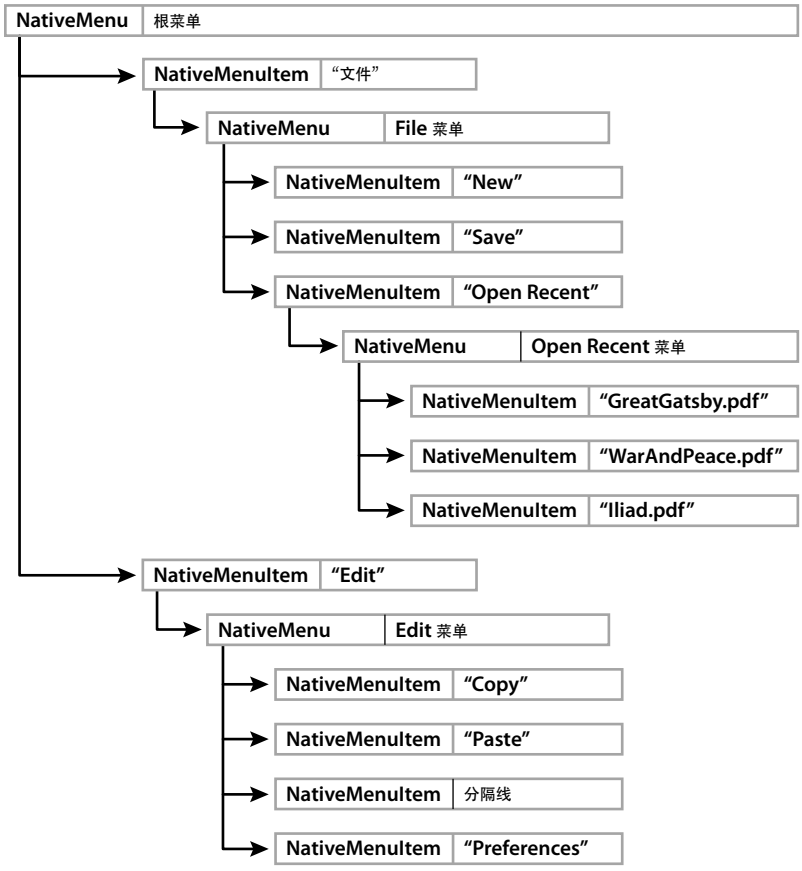
```
var square:Sprite = new Sprite();
square.graphics.beginFill(0x000000);
square.graphics.drawRect(0,0,100,100);
square.graphics.endFill();
square.x =
square.y = 10;
addChild(square);

var menuItem:ContextMenuItem = new ContextMenuItem("Change Color");
menuItem.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,changeColor);
var customContextMenu:ContextMenu = new ContextMenu();
customContextMenu.hideBuiltInItems();
customContextMenu.customItems.push(menuItem);
square.contextMenu = customContextMenu;

function changeColor(event:ContextMenuEvent):void
{
    square.transform.colorTransform = getRandomColor();
}
function getRandomColor():ColorTransform
{
    return new ColorTransform(Math.random(), Math.random(), Math.random(),1,(Math.random() * 512) - 255,
(Math.random() * 512) -255, (Math.random() * 512) - 255, 0);
}
```

本机菜单结构 (AIR)

Adobe AIR 1.0 和更高版本

本机菜单本质上是分层的。 NativeMenu 对象包含子级 NativeMenuItem 对象。表示子菜单的 NativeMenuItem 对象又可以包含 NativeMenu 对象。结构中的顶级或根级的菜单对象表示应用程序菜单和窗口菜单的菜单栏。（上下文、图标和弹出菜单没有菜单栏）。

下图说明了一个典型菜单的结构。根菜单表示菜单栏，它包含的两个菜单项引用“File”（文件）子菜单和“Edit”（编辑）子菜单。在此结构中，“File”（文件）子菜单包含两个命令项和一个引用项，该引用项引用“Open Recent”（打开最近的项目）子菜单，而该子菜单本身又包含三个项目。“Edit”（编辑）子菜单包含三个命令和一个分隔符。



定义子菜单同时需要 NativeMenu 和 NativeMenuItem 对象。 NativeMenuItem 对象定义在父菜单中显示的标签，并允许用户打开子菜单。NativeMenu 对象充当子菜单中的项目的容器。NativeMenuItem 对象通过 NativeMenuItem 的 submenu 属性引用 NativeMenu 对象。

若要查看创建此菜单的代码示例，请参阅第 551 页的“本机菜单示例：窗口和应用程序菜单 (AIR)”。

菜单的事件

Adobe AIR 1.0 和更高版本

NativeMenu 和 NativeMenuItem 对象均调度 preparing、 displaying 和 select 事件：

Preparing：每当对象即将开始用户交互时，该菜单及其菜单项会将 preparing 事件调度到任何注册的侦听器。交互包括打开菜单或使用键盘快捷键选择项目。

注：preparing 事件仅适用于 Adobe AIR 2.6 和更高版本。

Displaying： 在显示菜单的前一刻，菜单及其菜单项将 displaying 事件调度到任何注册的侦听器。

preparing 和 displaying 事件允许您在向用户显示菜单内容或项目外观之前对其进行更新。例如，在“打开最近的文件”菜单的 displaying 事件的侦听器中，可以更改菜单项以反映最近查看过的文档的当前列表。

如果您删除了一个其键盘快捷键会触发 preparing 事件的菜单项，则将实际取消菜单交互，并且将不会调度 select 事件。

该事件的 target 和 currentTarget 属性均为在其上注册了侦听器的对象：菜单本身或其中的一个项目。

preparing 事件将在 displaying 事件之前调度。通常，您只会侦听这两个事件中的一个，而不会同时侦听二者。

Select： 当用户选择命令项时，项目会将 select 事件调度到任何注册的侦听器。不能选择子菜单和分隔符项，因此永远不会调度 select 事件。

select 事件从菜单项上升到它的包含菜单，然后上升到根菜单。可以直接在项目上侦听 select 事件，也可以在菜单结构的更高位置侦听。在菜单上侦听 select 事件时，可以使用该事件的 target 属性识别所选项。当事件沿菜单层次结构上升时，该事件对象的 currentTarget 属性可识别当前菜单对象。

注 ContextMenu 和 ContextMenuItem 对象调度 menuItemSelect 和 menuSelect 事件以及 select、preparing 和 displaying 事件。

本机菜单命令的等效键 (AIR)

Adobe AIR 1.0 和更高版本

可以为菜单命令分配等效键 （有时称为快捷键）。当按下键或组合键时，菜单项会将 select 事件调度到任何注册的侦听器。包含该项目的菜单必须是应用程序菜单的一部分，或者是要调用的命令的活动窗口菜单的一部分。

等效键有两部分，一部分是表示主键的字符串，另一部分是一组必须同时按下的功能键。若要分配主键，请将菜单项 keyEquivalent 属性设置为该键的单字符字符串。如果使用大写字母，则 Shift 键会自动添加到功能键组中。

在 Mac OS X 上，默认功能键是 Command 键 (Keyboard.COMMAND)。在 Windows 和 Linux 中，是 Control 键 (Keyboard.CONTROL)。这些默认键会自动添加到功能键组中。若要分配其他功能键，请将包含所需键代码的新键组分配给 keyEquivalentModifiers 属性。默认键组将被覆盖。无论使用默认功能键还是分配自己的功能键组，如果分配给 keyEquivalent 属性的字符串是大写字母，则会添加 Shift 键。用于功能键的键代码的常量在 Keyboard 类中定义。

所分配的等效键字符串将自动显示在菜单项名称的旁边。格式取决于用户的操作系统和系统首选项。

注：在 Windows 操作系统上，如果将 Keyboard.COMMAND 值分配给功能键组，则菜单中不显示等效键。但是，必须使用 Ctrl 键才能激活菜单命令。

以下示例分配 Ctrl+Shift+G 作为菜单项的等效键：

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
```

此示例通过直接设置功能键组将 Ctrl+Shift+G 分配为等效键：

```
var item:NativeMenuItem = new NativeMenuItem("Ungroup");
item.keyEquivalent = "G";
item.keyEquivalentModifiers = [Keyboard.CONTROL];
```

注：等效键仅为应用程序菜单和窗口菜单触发。如果将等效键添加到上下文或弹出菜单，则等效键将显示在菜单标签中，但永远不会调用关联的菜单命令。

助记键 (AIR)

Adobe AIR 1.0 和更高版本

助记键是菜单的操作系统键盘接口的一部分。 Linux、 Mac OS X 和 Windows 都允许用户通过键盘打开菜单并选择命令，但有一些细微的区别。

在 Mac OS X 中，用户键入菜单或命令的第一个或前两个字母，然后按 Return 键。mnemonicIndex 属性将被忽略。

在 Windows 上，仅一个字母是有效的。默认情况下，有效字母是标签中的第一个字符，但是，如果将助记键分配给菜单项，则有效字符将成为所指定的字母。如果一个菜单中的两个项有相同的有效字符 （无论是否分配了助记键），则用户的键盘与菜单的交互稍有改变。用户必须将该字母按下所需的次数来突出显示所需项，然后按 Enter 完成选择，而不是仅按单个字母就能选择菜单或命令。为了保持一致的行为，应向窗口菜单中的每个菜单项都分配一个唯一的助记键。

在 Linux 中不提供默认的助记键。必须指定菜单项的 mnemonicIndex 属性的值才能提供助记键。

指定助记键字符作为标签字符串中的索引。标签中第一个字符的索引是 0。因此，若要使用 “r” 作为带 “Format” 标签的菜单项的助记键，可以将 mnemonicIndex 属性设置为等于 2。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.mnemonicIndex = 2;
```

菜单项状态

Adobe AIR 1.0 和更高版本

菜单项有两个状态属性：checked 和 enabled。

checked 设置为 true 将在项目标签旁边显示选中标记。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.checked = true;
```

enabled 在 true 和 false 之间切换值可以控制是否启用命令。禁用的项目将在视觉上 “ 灰显 ”，并且不调度 select 事件。

```
var item:NativeMenuItem = new NativeMenuItem("Format");
item.enabled = false;
```

将对象附加到菜单项

Adobe AIR 1.0 和更高版本

使用 NativeMenuItem 类的 data 属性可以引用每个项目中的任意对象。例如，在 “Open Recent” （打开最近的项目）菜单中，可以将每个文档的 File 对象分配给每个菜单项。

```
var file:File = File.applicationStorageDirectory.resolvePath("GreatGatsby.pdf")
var menuItem:NativeMenuItem = docMenu.addItem(new NativeMenuItem(file.name));
menuItem.data = file;
```

创建本机菜单 (AIR)

Adobe AIR 1.0 和更高版本

本主题描述如何创建 AIR 所支持的各种类型的本机菜单。

创建根菜单对象

Adobe AIR 1.0 和更高版本

若要创建 NativeMenu 对象来充当菜单的根，请使用 NativeMenu 构造函数：

```
var root:NativeMenu = new NativeMenu();
```

对于应用程序菜单和窗口菜单，根菜单表示菜单栏，并且应当只包含打开子菜单的项目。上下文菜单和弹出菜单没有菜单栏，因此根菜单可以包含命令和分隔线以及子菜单。

在创建菜单之后，可以添加菜单项。除非使用菜单对象的 addItemAt() 方法在给定索引处添加项目，否则项目以添加顺序出现在菜单中。

将菜单分配为应用程序、窗口、图标或上下文菜单，或将其显示为弹出菜单，如以下几节所示：

设置应用程序菜单或窗口菜单
您的代码中应包含应用程序菜单 （受 Mac OS 支持）和窗口菜单 （受其他操作系统支持），这很重要

```
var root:NativeMenu = new NativeMenu();
if (NativeApplication.supportsMenu)
{
    NativeApplication.nativeApplication.menu = root;
}
else if (NativeWindow.supportsMenu)
{
    nativeWindow.menu = root;
}
```

注：Mac OS 定义了一个菜单，其中包含可用于每个应用程序的标准项目。将新 NativeMenu 对象分配给 NativeApplication 对象的 menu 属性可以替换标准菜单。还可以使用标准菜单，而不是替换它。

Adobe Flex 提供了 FlexNativeMenu 类，用于方便地创建跨平台工作的菜单。如果您使用的是 Flex 框架，请使用 FlexNativeMenu 类，而不要使用 NativeMenu 类。

在交互式对象上设置上下文菜单

```
interactiveObject.contextMenu = root;
```

设置停靠栏图标菜单或系统托盘图标菜单
您的代码中应包含应用程序菜单 （受 Mac OS 支持）和窗口菜单 （受其他操作系统支持），这很重要

```
if (NativeApplication.supportsSystemTrayIcon)
{
    SystemTrayIcon(NativeApplication.nativeApplication.icon).menu = root;
}
else if (NativeApplication.supportsDockIcon)
{
    DockIcon(NativeApplication.nativeApplication.icon).menu = root;
}
```

注：Mac OS X 为应用程序停靠栏图标定义了标准菜单。在将新 NativeMenu 分配给 DockIcon 对象的 menu 属性时，该菜单中的项目将显示在标准项目之上。不能删除、访问或修改标准菜单项。

以弹出方式显示菜单

```
root.display(stage, x, y);
```

更多帮助主题

开发跨平台 AIR 应用程序

创建子菜单

Adobe AIR 1.0 和更高版本

若要创建子菜单，请将 NativeMenuItem 对象添加到父菜单，然后将定义子菜单的 NativeMenu 对象分配给该项目的 submenu 属性。 AIR 提供了两种方式来创建子菜单项及其关联的菜单对象：

可以使用 addSubmenu() 方法在一个步骤中创建菜单项及其相关的菜单对象：

```
var editMenuItem:NativeMenuItem = root.addSubmenu(new NativeMenu(), "Edit");
```

也可以创建菜单项，然后单独将菜单对象分配给其 submenu 属性：

```
var editMenuItem:NativeMenuItem = root.addItem("Edit", false);
editMenuItem.submenu = new NativeMenu();
```

创建菜单命令

Adobe AIR 1.0 和更高版本

若要创建菜单命令，请将 NativeMenuItem 对象添加到菜单，然后添加一个事件侦听器来引用实现菜单命令的函数：

```
var copy:NativeMenuItem = new NativeMenuItem("Copy", false);
copy.addEventListener(Event.SELECT, onCopyCommand);
editMenu.addItem(copy);
```

可以在命令项本身上侦听 select 事件 （如本例中所示），也可以在父菜单对象上侦听 select 事件。

注：表示子菜单和分隔线的菜单项不调度 select 事件，因此不能用作命令。

创建菜单分隔线

Adobe AIR 1.0 和更高版本

若要创建分隔线，请创建 NativeMenuItem，并在构造函数中将 isSeparator 参数设置为 true。然后，将分隔符项目添加到菜单中的正确位置：

```
var separatorA:NativeMenuItem = new NativeMenuItem("A", true);
editMenu.addItem(separatorA);
```

不显示为分隔符指定的标签 （如果有）。

关于 HTML 中的上下文菜单 (AIR)

Adobe AIR 1.0 和更高版本

在使用 HTMLLoader 对象显示的 HTML 内容中， contextmenu 事件可用于显示上下文菜单。默认情况下，当用户调用所选文本上的上下文菜单事件时 （通过右键单击或命令单击文本），将自动显示上下文菜单。若要防止打开默认菜单，可以侦听 contextmenu 事件并调用事件对象的 preventDefault() 方法：

```
function showContextMenu(event){
    event.preventDefault();
}
```

然后可以使用 DHTML 技术或通过显示 AIR 本机上下文菜单来显示自定义上下文菜单。以下示例通过调用菜单的 display() 方法响应 HTML contextmenu 事件，来显示本机上下文菜单：

```
<html>
<head>
<script src="AIRAliases.js" language="JavaScript" type="text/javascript"></script>
<script language="javascript" type="text/javascript">

function showContextMenu(event){
    event.preventDefault();
    contextMenu.display(window.nativeWindow.stage, event.clientX, event.clientY);
}

function createContextMenu(){
    var menu = new air.NativeMenu();
    var command = menu.addItem(new air.NativeMenuItem("Custom command"));
    command.addEventListener(air.Event.SELECT, onCommand);
    return menu;
}

function onCommand(){
    air.trace("Context command invoked.");
}

var contextMenu = createContextMenu();
</script>
</head>
<body>
<p oncontextmenu="showContextMenu(event)" style="-khtml-user-select:auto;">Custom context menu.</p>
</body>
</html>
```

显示弹出本机菜单 (AIR)

Adobe AIR 1.0 和更高版本

通过调用菜单的 display() 方法，可以随时随地在窗口上方显示任何 NativeMenu 对象。此方法需要对舞台的引用；因此，只有应用程序沙箱中的内容可以将菜单显示为弹出菜单。

以下方法显示由名为 popupMenu 的 NativeMenu 对象定义的菜单来响应鼠标单击：

```
private function onMouseClick(event:MouseEvent):void {
    popupMenu.display(event.target.stage, event.stageX, event.stageY);
}
```

注：不需要显示此菜单直接响应事件。任何方法都可以调用 display() 函数。

处理菜单事件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当用户选择菜单时，或用户选择菜单项时，菜单将调度事件。

菜单类的事件摘要

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

将事件侦听器添加到菜单或个别项目来处理菜单事件。

Object	调度的事件
NativeMenu (AIR)	Event.PREPARING （Adobe AIR 2.6 和更高版本） Event.DISPLAYING Event.SELECT （从子项目和子菜单传播）
NativeMenuItem (AIR)	Event.PREPARING （Adobe AIR 2.6 和更高版本） Event.SELECT Event.DISPLAYING （从父菜单传播）
ContextMenu	ContextMenuEvent.MENU_SELECT
ContextMenuItem	ContextMenuEvent.MENU_ITEM_SELECT Event.SELECT (AIR)

选择菜单事件

Adobe AIR 1.0 和更高版本

若要处理菜单项上的单击，请将 select 事件的事件侦听器添加到 NativeMenuItem 对象：

```
var menuCommandX:NativeMenuItem = new NativeMenuItem("Command X");
menuCommandX.addEventListener(Event.SELECT, doCommandX)
```

因为 select 事件会上升到包含菜单，所以还可以在父菜单上侦听 select 事件。在菜单级别上侦听时，可以使用事件对象的 target 属性来确定选择了哪个菜单命令。以下示例跟踪所选命令的标签：

```
var colorMenuItem:NativeMenuItem = new NativeMenuItem("Choose a color");
var colorMenu:NativeMenu = new NativeMenu();
colorMenuItem.submenu = colorMenu;

var red:NativeMenuItem = new NativeMenuItem("Red");
var green:NativeMenuItem = new NativeMenuItem("Green");
var blue:NativeMenuItem = new NativeMenuItem("Blue");
colorMenu.addItem(red);
colorMenu.addItem(green);
colorMenu.addItem(blue);

if(NativeApplication.supportsMenu){
    NativeApplication.nativeApplication.menu.addItem(colorMenuItem);
    NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT, colorChoice);
} else if (NativeWindow.supportsMenu){
    var windowMenu:NativeMenu = new NativeMenu();
    this.stage.nativeWindow.menu = windowMenu;
    windowMenu.addItem(colorMenuItem);
    windowMenu.addEventListener(Event.SELECT, colorChoice);
}

function colorChoice(event:Event):void {
    var menuItem:NativeMenuItem = event.target as NativeMenuItem;
    trace(menuItem.label + " has been selected");
}
```

如果正在使用 ContextMenuItem 类，则可以侦听 select 事件或 menuItemSelect 事件。menuItemSelect 事件可以提供有关拥有此上下文菜单的对象的其他信息，但不能上升到包含菜单。

显示菜单事件

Adobe AIR 1.0 和更高版本

若要处理菜单的打开，可以为 displaying 事件添加一个侦听器，在显示菜单之前将调度该侦听器。可以使用 displaying 事件更新菜单，例如，通过添加或删除项目，或通过更新个别项目的启用或选中状态。还可以从 ContextMenu 对象侦听 menuSelect 事件。

在 AIR 2.6 和更高版本中，可以使用 preparing 事件来更新菜单，以响应显示菜单或使用键盘快捷键选择项目。

本机菜单示例：窗口和应用程序菜单 (AIR)

Adobe AIR 1.0 和更高版本

以下示例创建在第 543 页的 " 本机菜单结构 (AIR)" 中显示的菜单。

此菜单在设计上可同时用于 Windows （仅支持窗口菜单）和 Mac OS X （仅支持应用程序菜单）。为进行区分，MenuExample 类构造函数将检查 NativeWindow 和 NativeApplication 类的静态 supportsMenu 属性。如果 NativeWindow.supportsMenu 为 true，则该构造函数将为窗口创建 NativeMenu 对象，然后创建和添加 "File" （文件）和 "Edit" （编辑）子菜单。如果 NativeApplication.supportsMenu 为 true，则该构造函数将创建 "File" （文件）和 "Edit" （编辑）菜单，并将它们添加到 Mac OS X 操作系统所提供的现有菜单中。

本示例还将说明菜单事件的处理过程。select 事件在项目级别以及菜单级别进行处理。从包含所选项的菜单到根菜单的菜单链中的每个菜单都将响应 select 事件。 displaying 事件与 "Open Recent" （最近打开的项目）菜单一起使用。在打开菜单之前，将以最新的 Documents 数组 （在此示例中实际上不发生更改）刷新菜单中的项目。尽管此示例中不显示，但还可以在个别项目上侦听 displaying 事件。

```
package {
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.filesystem.File;
    import flash.desktop.NativeApplication;

    public class MenuExample extends Sprite
    {
        private var recentDocuments:Array =
            new Array(new File("app-storage:/GreatGatsby.pdf"),
                      new File("app-storage:/WarAndPeace.pdf"),
                      new File("app-storage:/Iliad.pdf"));

        public function MenuExample()
        {
            var fileMenu:NativeMenuItem;
            var editMenu:NativeMenuItem;

            if (NativeWindow.supportsMenu){
                stage.nativeWindow.menu = new NativeMenu();
                stage.nativeWindow.menu.addEventListener(Event.SELECT, selectCommandMenu);
                fileMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("File"));
```

```
                fileMenu.submenu = createFileMenu();
                editMenu = stage.nativeWindow.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }

            if (NativeApplication.supportsMenu){
                NativeApplication.nativeApplication.menu.addEventListener(Event.SELECT,
selectCommandMenu);
                fileMenu = NativeApplication.nativeApplication.menu.addItem(new NativeMenuItem("File"));
                fileMenu.submenu = createFileMenu();
                editMenu = NativeApplication.nativeApplication.menu.addItem(new NativeMenuItem("Edit"));
                editMenu.submenu = createEditMenu();
            }
        }

        public function createFileMenu():NativeMenu {
            var fileMenu:NativeMenu = new NativeMenu();
            fileMenu.addEventListener(Event.SELECT, selectCommandMenu);

            var newCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("New"));
            newCommand.addEventListener(Event.SELECT, selectCommand);
            var saveCommand:NativeMenuItem = fileMenu.addItem(new NativeMenuItem("Save"));
            saveCommand.addEventListener(Event.SELECT, selectCommand);
            var openRecentMenu:NativeMenuItem =
                    fileMenu.addItem(new NativeMenuItem("Open Recent"));
            openRecentMenu.submenu = new NativeMenu();
            openRecentMenu.submenu.addEventListener(Event.DISPLAYING,
                                            updateRecentDocumentMenu);
            openRecentMenu.submenu.addEventListener(Event.SELECT, selectCommandMenu);

            return fileMenu;
        }

        public function createEditMenu():NativeMenu {
            var editMenu:NativeMenu = new NativeMenu();
            editMenu.addEventListener(Event.SELECT, selectCommandMenu);

            var copyCommand:NativeMenuItem = editMenu.addItem(new NativeMenuItem("Copy"));
            copyCommand.addEventListener(Event.SELECT, selectCommand);
            copyCommand.keyEquivalent = "c";
            var pasteCommand:NativeMenuItem =
                    editMenu.addItem(new NativeMenuItem("Paste"));
            pasteCommand.addEventListener(Event.SELECT, selectCommand);
            pasteCommand.keyEquivalent = "v";
            editMenu.addItem(new NativeMenuItem("", true));
            var preferencesCommand:NativeMenuItem =
                    editMenu.addItem(new NativeMenuItem("Preferences"));
            preferencesCommand.addEventListener(Event.SELECT, selectCommand);

            return editMenu;
        }

        private function updateRecentDocumentMenu(event:Event):void {
            trace("Updating recent document menu.");
            var docMenu:NativeMenu = NativeMenu(event.target);

            for each (var item:NativeMenuItem in docMenu.items) {
                docMenu.removeItem(item);
            }

            for each (var file:File in recentDocuments) {
                var menuItem:NativeMenuItem =
```

```
                    docMenu.addItem(new NativeMenuItem(file.name));
            menuItem.data = file;
            menuItem.addEventListener(Event.SELECT, selectRecentDocument);
        }
    }

    private function selectRecentDocument(event:Event):void {
        trace("Selected recent document: " + event.target.data.name);
    }

    private function selectCommand(event:Event):void {
        trace("Selected command: " + event.target.label);
    }

    private function selectCommandMenu(event:Event):void {
        if (event.currentTarget.parent != null) {
            var menuItem:NativeMenuItem =
                    findItemForMenu(NativeMenu(event.currentTarget));
            if (menuItem != null) {
                trace("Select event for \"" +
                        event.target.label +
                        "\" command handled by menu: " +
                        menuItem.label);
            }
        } else {
            trace("Select event for \"" +
                    event.target.label +
                    "\" command handled by root menu.");
        }
    }

    private function findItemForMenu(menu:NativeMenu):NativeMenuItem {
        for each (var item:NativeMenuItem in menu.parent.items) {
            if (item != null) {
                if (item.submenu == menu) {
                    return item;
                }
            }
        }
        return null;
    }
  }
}
```

第 37 章 : AIR 中的任务栏图标

Adobe AIR 1.0 和更高版本

很多操作系统都提供有任务栏 （例如 Mac OS X 停靠栏），任务栏中可包含表示应用程序的图标。 Adobe® AIR® 提供了一个接口，可以通过 NativeApplication.nativeApplication.icon 属性与应用程序任务栏图标进行交互。

- 使用系统任务栏和停靠栏图标 (Flex)
- 使用系统任务栏和停靠栏图标 (Flash)

更多帮助主题

flash.desktop.NativeApplication

flash.desktop.DockIcon

flash.desktop.SystemTrayIcon

关于任务栏图标

Adobe AIR 1.0 和更高版本

AIR 会自动创建 NativeApplication.nativeApplication.icon 对象。对象类型可以为 DockIcon 或 SystemTrayIcon，具体取决于操作系统。可以使用 NativeApplication.supportsDockIcon 和 NativeApplication.supportsSystemTrayIcon 属性来确定 AIR 在当前操作系统上支持哪些 InteractiveIcon 子类。InteractiveIcon 基类提供了 width、height 和 bitmaps 属性，可以使用这些属性来更改图标所使用的图像。但是，在错误操作系统上访问特定于 DockIcon 或 SystemTrayIcon 的属性会生成运行时错误。

若要设置或更改图标所使用的图像，请创建一个包含一个或多个图像的数组，然后将该数组分配给 NativeApplication.nativeApplication.icon.bitmaps 属性。在不同操作系统上，任务栏图标的大小会有所不同。为了避免因缩放而导致图像质量降级，可以向 bitmaps 数组中添加多个不同大小的图像。如果提供多个图像， AIR 会选择大小与任务栏图标的当前显示大小最接近的图像，仅在需要时进行缩放。以下示例使用两个图像来设置任务栏图标的图像：

```
NativeApplication.nativeApplication.icon.bitmaps =
            [bmp16x16.bitmapData, bmp128x128.bitmapData];
```

若要更改图标图像，请将包含新图像的数组分配给 bitmaps 属性。通过响应 enterFrame 或 timer 事件来更改图像，可以为图标添加动画效果。

若要从 Windows 和 Linux 的通知区域中删除图标，或者恢复 Mac OS X 中的默认图标外观，请将 bitmaps 设置为空数组：

```
NativeApplication.nativeApplication.icon.bitmaps = [];
```

停靠栏图标

Adobe AIR 1.0 和更高版本

AIR 在 NativeApplication.supportsDockIcon 为 true 时支持停靠栏图标。NativeApplication.nativeApplication.icon 属性表示停靠栏上的应用程序图标 （而不是窗口的停靠栏图标）。

注：AIR 不支持更改 Mac OS X 停靠栏上的窗口图标。此外，对应用程序的停靠栏图标所做的更改只有当应用程序运行时才会应用，应用程序终止时图标将还原为其正常外观。

停靠栏图标菜单

Adobe AIR 1.0 和更高版本

通过创建包含命令的 NativeMenu 对象并将其分配给 NativeApplication.nativeApplication.icon.menu 属性，可以向标准停靠栏菜单中添加命令。菜单中的项目显示在标准停靠栏图标菜单项的上方。

回弹停靠栏

Adobe AIR 1.0 和更高版本

通过调用 NativeApplication.nativeApplication.icon.bounce() 方法，可以回弹停靠栏图标。如果将 bounce() priority 参数设置为 informational，则图标将回弹一次。如果将该参数设置为 critical，则图标将始终保持回弹状态，直到用户激活该应用程序。用作 priority 参数的常量由 NotificationType 类定义。

注：如果应用程序已经处于活动状态，则不会回弹图标。

停靠栏图标事件

Adobe AIR 1.0 和更高版本

单击停靠栏图标后， NativeApplication 对象将调度 invoke 事件。如果应用程序尚未运行，则系统将启动该应用程序。否则，invoke 事件将传送到正在运行的应用程序实例。

系统任务栏图标

Adobe AIR 1.0 和更高版本

当 NativeApplication.supportsSystemTrayIcon 为 true 时， AIR 支持系统任务栏图标 （目前只有 Windows 和大多数 Linux 发行版是这种情况）。在 Windows 和 Linux 中，系统任务栏图标显示在任务栏的通知区域中。默认情况下不显示任何图标。要显示图标，请将包含 BitmapData 对象的数组分配给图标的 bitmaps 属性。若要更改图标图像，请将包含新图像的数组分配给 bitmaps。若要删除图标，请将 bitmaps 设置为 null。

系统任务栏图标菜单

Adobe AIR 1.0 和更高版本

通过创建 NativeMenu 对象并将其分配给 NativeApplication.nativeApplication.icon.menu 属性，可以向系统任务栏图标中添加菜单 （操作系统不会提供任何默认菜单）。右键单击图标即可访问系统任务栏图标菜单。

系统任务栏图标工具提示

Adobe AIR 1.0 和更高版本

通过设置 tooltip 属性可以向图标添加工具提示：

```
NativeApplication.nativeApplication.icon.tooltip = "Application name";
```

系统任务栏图标事件

Adobe AIR 1.0 和更高版本

NativeApplication.nativeApplication.icon 属性引用的 SystemTrayIcon 对象可以为 click、 mouseDown、 mouseUp、 rightClick、 rightMouseDown 和 rightMouseUp 事件调度 ScreenMouseEvent。可以组合使用这些事件和图标菜单，以便用户能够在您的应用程序没有可见窗口时与应用程序进行交互。

示例：创建不带任何窗口的应用程序

Adobe AIR 1.0 和更高版本

以下示例创建了一个具有系统任务栏图标但没有可见窗口的 AIR 应用程序。（在应用程序描述符中，不得将应用程序的 visible 属性设置为 true，否则窗口在应用程序启动时可见。）

```
package
{
    import flash.display.Loader;
    import flash.display.NativeMenu;
    import flash.display.NativeMenuItem;
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.desktop.DockIcon;
    import flash.desktop.SystemTrayIcon;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.desktop.NativeApplication;

    public class SysTrayApp extends Sprite
    {
        public function SysTrayApp():void{
            NativeApplication.nativeApplication.autoExit = false;
            var icon:Loader = new Loader();
            var iconMenu:NativeMenu = new NativeMenu();
            var exitCommand:NativeMenuItem = iconMenu.addItem(new NativeMenuItem("Exit"));
                exitCommand.addEventListener(Event.SELECT, function(event:Event):void {
                    NativeApplication.nativeApplication.icon.bitmaps = [];
                    NativeApplication.nativeApplication.exit();
                });

            if (NativeApplication.supportsSystemTrayIcon) {
                NativeApplication.nativeApplication.autoExit = false;
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE, iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_16.png"));

                var systray:SystemTrayIcon =
```

```
                    NativeApplication.nativeApplication.icon as SystemTrayIcon;
                systray.tooltip = "AIR application";
                systray.menu = iconMenu;
            }

            if (NativeApplication.supportsDockIcon){
                icon.contentLoaderInfo.addEventListener(Event.COMPLETE,iconLoadComplete);
                icon.load(new URLRequest("icons/AIRApp_128.png"));
                var dock:DockIcon = NativeApplication.nativeApplication.icon as DockIcon;
                dock.menu = iconMenu;
            }
        }

        private function iconLoadComplete(event:Event):void
        {
            NativeApplication.nativeApplication.icon.bitmaps =
                [event.target.content.bitmapData];
        }
    }
}
```

注：使用 Flex WindowedApplication 组件时，必须将 WindowedApplication 标记的 visible 属性设置为 false。该属性取代了应用程序描述符中的设置。

注：该示例假设应用程序的 icons 子目录中存在名为 AIRApp_16.png 和 AIRApp_128.png 的图像文件。（示例图标文件包含在 AIR SDK 中，您可以将这些文件复制到项目文件夹中。）

Window 任务栏图标和按钮

Adobe AIR 1.0 和更高版本

窗口的图标化表示形式通常显示在任务栏或停靠栏的窗口区域中，以便用户能够轻松访问后台窗口或最小化的窗口。 Mac OS X 停靠栏会为您的应用程序显示一个图标，并为每个最小化的窗口分别显示一个图标。 Microsoft Windows 和 Linux 任务栏显示一个按钮，其中包含您的应用程序中每个普通类型窗口的程序图标和标题。

加亮显示任务栏窗口按钮

Adobe AIR 1.0 和更高版本

如果窗口位于后台，则可以通知用户发生了与该窗口相关的需要关注的事件。在 Mac OS X 中，可以通过回弹应用程序的停靠栏图标来通知用户 （如第 555 页的 “ 回弹停靠栏 ” 中所述）。在 Windows 和 Linux 中，可以通过调用 NativeWindow 实例的 notifyUser() 方法来加亮显示窗口的任务栏按钮。传递给该方法的 type 参数用于确定通知的紧急程度：

- NotificationType.CRITICAL：在用户将窗口置于前台之前，窗口图标一直闪烁。
- NotificationType.INFORMATIONAL：通过更改颜色来加亮显示窗口图标。

  注：在 Linux 中，仅支持信息性类型的通知。向 notifyUser() 函数传递任何一种类型值都会产生相同的效果。

  以下语句加亮显示窗口的任务栏按钮：

  ```
  stage.nativeWindow.notifyUser(NotificationType.CRITICAL);
  ```

  在不支持窗口级别通知的操作系统中，调用 NativeWindow.notifyUser() 方法将不起作用。使用 NativeWindow.supportsNotification 属性可确定是否支持窗口通知。

创建不带任务栏按钮或图标的窗口

Adobe AIR 1.0 和更高版本

在 Windows 操作系统中，使用 utility 或 lightweight 类型创建的窗口不会显示在任务栏中。不可见窗口也不会显示在任务栏中。

由于初始窗口必定为 normal 类型，因此要创建不会在任务栏中显示任何窗口的应用程序，必须关闭初始窗口或保持初始窗口不可见。若要关闭应用程序中的所有窗口而不终止应用程序，请在关闭最后一个窗口之前，将 NativeApplication 对象的 autoExit 属性设置为 false。如果只是需要使初始窗口变得不可见，请向应用程序描述符文件的 <initalWindow> 元素添加 <visible>false</visible> （且不要将 visible 属性设置为 true 或调用窗口的 activate() 方法）。

在应用程序打开的新窗口中，将传递给窗口构造函数的 NativeWindowInitOption 对象的 type 属性设置为 NativeWindowType.UTILITY 或 NativeWindowType.LIGHTWEIGHT。

在 Mac OS X 中，最小化的窗口显示在停靠任务栏中。通过隐藏窗口而不是最小化窗口可以避免显示最小化的图标。以下示例侦听 nativeWindowDisplayState 更改事件，并在窗口最小化时取消该事件。处理函数会改为将窗口的 visible 属性设置为 false：

```
private function preventMinimize(event:NativeWindowDisplayStateEvent):void{
    if(event.afterDisplayState == NativeWindowDisplayState.MINIMIZED){
        event.preventDefault();
        event.target.visible = false;
    }
}
```

如果将 visible 属性设置为 false 后窗口最小化到 Mac OS X 停靠栏中，则无法删除该停靠栏图标。用户仍可单击图标来重新显示窗口。

第 38 章：使用文件系统

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash® Player 通过 FileReference 类提供了基本的文件阅读和编写功能。出于安全考虑，在 Flash Player 中运行时，必须始终先为用户授予权限，然后才能阅读或编写文件。

与 Flash Player 相比，Adobe® AIR® 提供了对主机的文件系统更全面的访问权限。使用 AIR 文件系统 API，您可以访问和管理目录和文件、创建目录和文件、向文件中写入数据等。

更多帮助主题

flash.net.FileReference

flash.net.FileReferenceList

flash.filesystem.File

flash.filesystem.FileStream

使用 FileReference 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

FileReference 对象表示客户端或服务器计算机上的数据文件。通过 FileReference 类的方法可使应用程序本地加载和保存数据文件，以及与远程服务器之间传输文件数据。

FileReference 类为加载、传输和保存数据文件提供了两种不同的方法。自从引入 FileReference 类，该类就包括 browse() 方法、upload() 方法和 download() 方法。使用 browse() 方法使用户可以选择文件。使用 upload() 方法可以将文件数据传输到远程服务器。使用 download() 方法可以从服务器检索数据并将其保存在本地文件中。从 Flash Player 10 和 Adobe AIR 1.5 开始，FileReference 类包括 load() 和 save() 方法。load() 和 save() 这两种方法允许您直接访问和保存本地文件。这些方法的用法类似于 URLLoader 和 Loader 类中的同名方法。

注：File 类 （用于扩展 FileReference 类）和 FileStream 类提供了其他的函数用以使用文件和本地文件系统。仅 AIR 支持 File 和 FileStream 类，而 Flash Player 不支持这两个类。

FileReference 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

每个 FileReference 对象都代表本地计算机上的一个数据文件。 FileReference 类的属性包含有关文件大小、类型、名称、文件扩展名、创建者、创建日期和修改日期的信息。

注：仅 Mac OS 支持 creator 属性。所有其他平台都会返回 null。

注：仅 Adobe AIR 支持 extension 属性。

可以通过以下两种方式之一创建 FileReference 类的实例：

- 使用 new 运算符，如下面的代码所示：

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- 调用 FileReferenceList.browse() 方法，该方法将打开一个对话框，提示用户选择一个或多个要上载的文件。如果用户成功选择了一个或多个文件，则创建一个 FileReference 对象数组。

在创建完 FileReference 对象后，您便可以进行以下操作：

- 调用 FileReference.browse() 方法，该方法将打开一个对话框，提示用户从本地文件系统中选择一个文件。这种情况通常在后续调用 FileReference.upload() 方法或 FileReference.load() 方法之前执行。调用 FileReference.upload() 方法可将文件上载到远程服务器。调用 FileReference.load() 方法可以打开本地文件。
- 调用 FileReference.download() 方法。 download() 方法将打开一个对话框，让用户选择用于保存新文件的位置。然后从服务器下载数据，并将数据存储在新文件中。
- 调用 FileReference.load() 方法。此方法使用 browse() 方法开始从之前所选的文件中加载数据。直到 browse() 操作完成 （用户选择了文件）时，才能调用 load() 方法。
- 调用 FileReference.save() 方法。此方法将打开一个对话框，提示用户在本地文件系统上选择一个文件位置。然后此方法将数据保存到该指定位置。

注：一次只能执行一个 browse()、 download() 或 save() 操作，因为在任何时刻都只能打开一个对话框。

除非发生以下情况之一，否则不会定义 FileReference 对象属性 （如 name、 size 或 modificationDate）：

- 已调用 FileReference.browse() 方法或 FileReferenceList.browse() 方法，并且用户已经使用对话框选择了文件。
- 已调用 FileReference.download() 方法，并且用户已经使用对话框指定了新的文件位置。

注：当执行下载时，在下载完成前只填充 FileReference.name 属性。下载完文件后，所有属性都将可用。

在执行对 FileReference.browse()、 FileReferenceList.browse()、 FileReference.download()、 FileReference.load() 或 FileReference.save() 方法的调用时，大多数播放器将继续播放 SWF 文件，其中包括调度事件和执行代码。

对于上载和下载操作， SWF 文件只能访问自己的域 （包括由策略文件指定的任何域）内的文件。如果包含文件的服务器与启动上载或下载的 SWF 文件不在同一个域中，则需要在该服务器上放置策略文件。

请参阅 FileReference。

从文件加载数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 FileReference.load() 方法可以将数据从本地文件加载到内存中。

注：代码必须先调用 FileReference.browse() 方法，让用户选择要加载的文件。此限制不适用于在应用程序安全沙箱的 Adobe AIR 中运行的内容

调用 FileReference.load() 方法后，该方法立即就会返回，但并不能立即得到所加载的数据。 FileReference 对象在加载过程的每个步骤中都会调度事件以调用侦听器方法。

FileReference 对象在加载过程中将调度以下事件。

- open 事件 (Event.OPEN)：在加载操作开始时调度。
- progress 事件 (ProgressEvent.PROGRESS)：以字节为单位从文件读取数据时定期调度。
- complete 事件 (Event.COMPLETE)：加载操作成功完成时调度。
- ioError 事件 (IOErrorEvent.IO_ERROR)：如果由于在打开文件或从文件读取数据时发生输入 / 输出错误而使加载过程失败，则调度此事件。

FileReference 对象调度 complete 事件后，即可将所加载的数据作为 FileReference 对象的 data 属性中的 ByteArray 进行访问。

下面的示例演示如何提示用户选择一个文件，然后从该文件将数据加载到内存中：

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample1 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample1()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            trace(fileRef.data);
        }

        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

示例代码首先创建名为 fileRef 的 FileReference 对象，然后调用其 browse() 方法。browse() 方法将打开一个对话框，提示用户选择文件。选择文件后，代码会调用 onFileSelected() 方法。此方法添加针对 progress 和 complete 事件的侦听器，然后调用 FileReference 对象的 load() 方法。示例中的其他处理函数方法只是输出消息以报告加载操作的进度。加载完成后，应用程序会使用 trace() 方法显示所加载文件的内容。

在 Adobe AIR 中， FileStream 类提供了读取本地文件中的数据的附加功能。请参阅第 591 页的 “ 读取和写入文件 ”。

将数据保存到本地文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 FileReference.save() 方法可以将数据保存到本地文件。该方法在开始时会打开一个对话框，让用户输入新文件名以及用于保存文件的位置。在用户选择文件名和位置后，数据会写入到新文件中。在成功保存文件之后，将使用本地文件的属性填充 FileReference 对象的属性。

注：只有在响应用户启动的事件 （如鼠标单击或按键事件）时，您的代码才会调用 FileReference.save() 方法。否则将引发错误。此限制不适用于在应用程序安全沙箱的 Adobe AIR 中运行的内容。

调用 FileReference.save() 方法之后，该方法会立即返回。 FileReference 对象随后在文件保存过程的每个步骤中调度事件以调用侦听器方法。

FileReference 对象在文件保存过程中可以调度以下事件：

- select 事件 (Event.SELECT)：在用户为要保存的新文件指定位置和文件名时调度。
- cancel 事件 (Event.CANCEL)：在用户单击对话框中的 “ 取消 ” 按钮时调度。
- open 事件 (Event.OPEN)：在保存操作开始时调度。
- progress 事件 (ProgressEvent.PROGRESS)：以字节为单位向文件保存数据时定期调度。
- complete 事件 (Event.COMPLETE)：保存操作成功完成时调度。
- ioError 事件 (IOErrorEvent.IO_ERROR)：如果由于尝试向文件保存数据时发生输入 / 输出错误而使保存过程失败，则调度此事件。

FileReference.save() 方法的 data 参数中传递的对象类型决定着向文件写入数据的方式：

- 如果是字符串值，则使用 UTF-8 编码将其另存为文本文件。
- 如果是 XML 对象，则以 XML 格式将其写入到文件，并保留所有格式设置。
- 如果是 ByteArray 对象，则将其内容直接写入文件，不经过转换。
- 如果是其他某些对象，则 FileReference.save() 方法调用该对象的 toString() 方法，然后将生成的字符串值保存到 UTF-8 文本文件中。如果无法调用该对象的 toString() 方法，则会引发错误。

如果 data 参数的值为 null，则会引发错误。

下面的代码扩展了前面针对 FileReference.load() 方法的示例。从文件中读取数据后，此示例将提示用户提供文件名，然后将数据保存到新文件中：

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.utils.ByteArray;

    public class FileReferenceExample2 extends Sprite
    {
        private var fileRef:FileReference;
        public function FileReferenceExample2()
        {
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, onFileSelected);
            fileRef.addEventListener(Event.CANCEL, onCancel);
            fileRef.addEventListener(IOErrorEvent.IO_ERROR, onIOError);
            fileRef.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
                        onSecurityError);
            var textTypeFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)",
                        "*.txt;*.rtf");
            fileRef.browse([textTypeFilter]);
        }
        public function onFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.addEventListener(Event.COMPLETE, onComplete);
            fileRef.load();
        }

        public function onProgress(evt:ProgressEvent):void
        {
            trace("Loaded " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }
        public function onCancel(evt:Event):void
        {
            trace("The browse request was canceled by the user.");
        }
        public function onComplete(evt:Event):void
        {
            trace("File was successfully loaded.");
            fileRef.removeEventListener(Event.SELECT, onFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onProgress);
            fileRef.removeEventListener(Event.COMPLETE, onComplete);
            fileRef.removeEventListener(Event.CANCEL, onCancel);
            saveFile();
        }
        public function saveFile():void
        {
            fileRef.addEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.save(fileRef.data,"NewFileName.txt");
        }

        public function onSaveFileSelected(evt:Event):void
        {
            fileRef.addEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.addEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.addEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveProgress(evt:ProgressEvent):void
```

```
        {
            trace("Saved " + evt.bytesLoaded + " of " + evt.bytesTotal + " bytes.");
        }

        public function onSaveComplete(evt:Event):void
        {
            trace("File saved.");
            fileRef.removeEventListener(Event.SELECT, onSaveFileSelected);
            fileRef.removeEventListener(ProgressEvent.PROGRESS, onSaveProgress);
            fileRef.removeEventListener(Event.COMPLETE, onSaveComplete);
            fileRef.removeEventListener(Event.CANCEL, onSaveCancel);
        }

        public function onSaveCancel(evt:Event):void
        {
            trace("The save request was canceled by the user.");
        }

        public function onIOError(evt:IOErrorEvent):void
        {
            trace("There was an IO Error.");
        }
        public function onSecurityError(evt:Event):void
        {
            trace("There was a security error.");
        }
    }
}
```

从文件加载所有数据后，代码将调用 onComplete() 方法。 onComplete() 方法删除针对加载事件的侦听器，然后调用 saveFile() 方法。 saveFile() 方法调用 FileReference.save() 方法。 FileReference.save() 方法会打开一个新对话框，让用户输入新文件名和用于保存文件的位置。其余事件侦听器方法在文件保存过程完成之前跟踪该过程的进度。

在 Adobe AIR 中， FileStream 类提供了将数据写入本地文件的附加功能。请参阅第 591 页的 “ 读取和写入文件 ”。

将文件上载至服务器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要将文件上载到服务器，需要首先调用 browse() 方法，以允许用户选择一个或多个文件。接下来，当调用 FileReference.upload() 方法时，将所选文件传输到服务器。如果用户使用 FileReferenceList.browse() 方法选择了多个文件，则 Flash Player 会创建所选文件的数组，称为 FileReferenceList.fileList。随后便可使用 FileReference.upload() 方法分别上载每个文件。

注：使用 FileReference.browse() 方法时，您只能上载单个文件。要允许用户上载多个文件，请使用 FileReferenceList.browse() 方法。

默认情况下，系统文件选取器对话框允许用户从本地计算机选取任何文件类型。开发人员可以通过使用 FileFilter 类并将文件过滤器实例数组传递给 browse() 方法来指定一个或多个自定义文件类型过滤器：

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg; *.jpeg; *.gif;
*.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

用户在系统文件选取器中选择文件并单击 “ 打开 ” 按钮后，会调度 Event.SELECT 事件。如果使用 FileReference.browse() 方法选择要上载的文件，下列代码会将文件发送到 Web 服务器：

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
    var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm")
    try
    {
        fileRef.upload(request);
    }
    catch (error:Error)
    {
        trace("Unable to upload file.");
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

通过使用 URLRequest.method 和 URLRequest.data 属性以 POST 或 GET 方法发送变量，可以用 FileReference.upload() 方法将数据发送到服务器。

尝试使用 FileReference.upload() 方法上载文件时，将调度以下事件：

- open 事件 (Event.OPEN)：在上载操作开始时调度。
- progress 事件 (ProgressEvent.PROGRESS)：以字节为单位上载文件中的数据时定期调度。
- complete 事件 (Event.COMPLETE)：上载操作成功完成时调度。
- httpStatus 事件 (HTTPStatusEvent.HTTP_STATUS)：上载过程因 HTTP 错误而失败时调度。
- httpResponseStatus 事件 (HTTPStatusEvent.HTTP_RESPONSE_STATUS): 如果调用 upload() 或 uploadUnencoded() 方法时尝试通过 HTTP 访问数据，并且 Adobe AIR 可以检测并返回请求的状态代码，则调度此事件。
- securityError 事件 (SecurityErrorEvent.SECURITY_ERROR)：上载操作因违反安全规则而失败时调度。
- uploadCompleteData 事件 (DataEvent.UPLOAD_COMPLETE_DATA)：成功上载并从服务器接收数据之后调度。
- ioError 事件 (IOErrorEvent.IO_ERROR)：由于下列任何原因导致上载过程失败时调度：
  - 当 Flash Player 正在读取、写入或传输文件时发生输入 / 输出错误。
  - SWF 尝试将文件上载到要求身份验证 （如用户名和密码）的服务器。在上载期间， Flash Player 不提供用户用于输入密码的方法。
  - url 参数包含无效协议。 FileReference.upload() 方法必须使用 HTTP 或 HTTPS。

Flash Player 不对需要身份验证的服务器提供完全支持。只有使用浏览器插件或 Microsoft ActiveX® 控件在浏览器中运行的 SWF 文件才可以提供一个对话框，来提示用户输入用户名和密码以进行身份验证，并且仅用于下载操作。对于使用插件或 ActiveX 控件进行的上载操作，或者使用独立或外部播放器进行的上载 / 下载操作，文件传输会失败。

若要以 ColdFusion 创建服务器脚本以接受来自 Flash Player 的文件上载，可以使用类似于以下内容的代码：

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#" nameconflict="OVERWRITE" />
```

此 ColdFusion 代码上载 Flash Player 发送的文件，并将其保存到 ColdFusion 模板所在的目录中以覆盖具有相同名称的任何文件。上面的代码显示接受文件上载所需的最低代码量；不应在生产环境中使用此脚本。理想情况下，添加数据验证可以确保用户仅上载接受的文件类型 （例如图像），而不是上载可能危险的服务器端脚本。

下面的代码使用 PHP 说明文件上载，并且包含数据验证。该脚本将上载目录中的上载文件数限制为 10，确保文件小于 200 KB，并且只允许 JPEG、 GIF 或 PNG 文件上载和保存到文件系统。

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // keep maximum 10 files on server
echo exif_imagetype($_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;
    }
}
?>
```

可以使用 POST 或 GET 请求方法将附加变量传递到上载脚本。要将附加 POST 变量发送到上载脚本，可以使用下面的代码：

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
    var params:URLVariables = new URLVariables();
    params.date = new Date();
    params.ssid = "94103-1394-2345";
    var request:URLRequest = new
URLRequest("http://www.yourdomain.com/FileReferenceUpload/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    request.data = params;
    fileRef.upload(request, "Custom1");
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

上面的示例创建一个将传递到远程服务器端脚本的 URLVariables 对象。在早期版本的 ActionScript 中，可以通过在查询字符串中传递值来将变量传递到服务器上载脚本。在 ActionScript 3.0 中，可以使用 URLRequest 对象将变量传递到远程脚本，该对象允许您使用 POST 或 GET 方法传递数据；因此，可以更轻松和更清晰地传递较大数据集。为了指定是使用 GET 还是使用 POST 请求方法来传递变量，可以将 URLRequest.method 属性相应设置为 URLRequestMethod.GET 或 URLRequestMethod.POST。

在 ActionScript 3.0 中，还可以通过向 upload() 方法提供第二个参数来覆盖默认 Filedata 上载文件字段名称，如上面的示例所示 （该示例使用 Custom1 替换默认值 Filedata）。

默认情况下，Flash Player 不尝试发送测试上载，虽然您可以通过将值 true 作为第三个参数传递给 upload() 方法来覆盖此默认行为。测试上载的目的是检查实际文件上载是否会成功，如果需要服务器身份，还会检查服务器身份验证是否会成功。

注：目前，只在基于 Windows 的 Flash Player 上进行测试上载。

处理文件上载的服务器脚本应收到包含下列元素的 HTTP POST 请求：

- Content-Type，其值为 multipart/form-data。
- Content-Disposition，其 name 属性设置为“Filedata”， filename 属性设置为原始文件的名称。您可以通过在 FileReference.upload() 方法中传递 uploadDataFieldName 参数的值来指定自定义 name 属性。
- 文件的二进制内容。

下面是一个 HTTP POST 请求范例：

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

Test File
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

下面的 HTTP POST 请求范例发送三个 POST 变量：api_sig、 api_key 和 auth_token，并使用自定义上载数据字段名的值 "photo"：

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream

(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

从服务器下载文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以让用户使用 FileReference.download() 方法从服务器下载文件，该方法使用两个参数：request 和 defaultFileName。第一个参数是 URLRequest 对象，该对象包含要下载的文件的 URL。第二个参数是可选的，使用该参数可以指定出现在下载文件对话框中的默认文件名。如果省略第二个参数 defaultFileName，则使用指定 URL 中的文件名。

下面的代码从 SWF 文件所在的目录下载名为 index.xml 的文件：

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

若要将默认名称设置为 currentnews.xml 而非 index.xml，请指定 defaultFileName 参数，如下面的片断所示：

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

如果服务器文件名不直观或是由服务器生成的，重命名文件会很有帮助。另外，在使用服务器端脚本下载文件 （而不是直接下载文件）时，最好明确指定 defaultFileName 参数。例如，如果有基于传递给它的 URL 变量下载特定文件的服务器端脚本，则需要指定 defaultFileName 参数。否则，下载文件的默认名称即是服务器端脚本的名称。

可以使用 download() 方法将数据发送至服务器，方法是将参数追加到要分析的服务器脚本的 URL。下面的 ActionScript 3.0 片断基于传递给 ColdFusion 脚本的参数下载文档：

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

下面的代码显示了 ColdFusion 脚本 download.cfm，该脚本根据 URL 变量的值从服务器下载两个文件之一：

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No" />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No" />
    </cfdefaultcase>
</cfswitch>
```

FileReferenceList 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 FileReferenceList 类，用户可以选择一个或多个要上载到服务器端脚本的文件。文件上载是由 FileReference.upload() 方法处理的，必须对用户选择的每个文件调用此方法。

下面的代码创建两个 FileFilter 对象 （imageFilter 和 textFilter），并在一个数组中将它们传递到 FileReferenceList.browse() 方法。这导致操作系统文件对话框显示两个可能的文件类型过滤器。

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif, *.png)", "*.jpg; *.jpeg;
*.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt; *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

允许用户使用 FileReferenceList 类选择并上载一个或多个文件与使用 FileReference.browse() 选择文件是相同的，但 FileReferenceList 允许选择多个文件。上载多个文件时，要求您使用 FileReference.upload() 上载每个所选的文件，如以下代码所示：

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();

function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

由于 Event.COMPLETE 事件会添加到数组中每个单独的 FileReference 对象中，因此，Flash Player 在每个文件完成上载时都会调用 completeHandler() 方法。

使用 AIR 文件系统 API

Adobe AIR 1.0 和更高版本

Adobe AIR 文件系统 API 包括以下类：

- 文件
- FileMode
- FileStream

借助该文件系统 API，您可以执行以下操作 （以及其他更多操作）：

- 复制、创建、删除和移动文件和目录
- 获取有关文件和目录的信息
- 读写文件

AIR 文件基础知识

Adobe AIR 1.0 和更高版本

有关在 AIR 中使用文件系统的快速介绍和代码示例，请参阅 Adobe Developer Connection 中的以下快速入门文章：

- 构建文本文件编辑器) (Flash)
- 构建文本文件编辑器) (Flex)
- 构建目录搜索应用程序) (Flex)
- 从 XML 首选参数文件中读取和写入) (Flex)
- 压缩文件和数据) (Flex)

您可以使用 Adobe AIR 提供的类访问、创建和管理文件和文件夹。这些类包含在 flash.filesystem 包中，用法如下所示：

File 类	说明
File	File 对象表示文件或目录的路径。您可以使用 file 对象创建指向文件或文件夹的指针，启动与文件或文件夹的交互。
FileMode	FileMode 类定义 FileStream 类的 open() 和 openAsync() 方法的 fileMode 参数中使用的字符串常量。这些方法的 fileMode 参数确定文件打开后 FileStream 对象可用的功能，包括写入、读取、追加和更新功能。
FileStream	FileStream 对象用于打开文件以进行读取和写入。创建了指向新文件或现有文件的 File 对象后，可以将该指针传递到 FileStream 对象，以便您可以打开该文件并读取或写入数据。

File 类中的一些方法具有同步版本和异步版本：

- File.copyTo() 和 File.copyToAsync()
- File.deleteDirectory() 和 File.deleteDirectoryAsync()
- File.deleteFile() 和 File.deleteFileAsync()
- File.getDirectoryListing() 和 File.getDirectoryListingAsync()
- File.moveTo() 和 File.moveToAsync()
- File.moveToTrash() 和 File.moveToTrashAsync()

另外，FileStream 操作是以同步方式运行还是以异步方式运行取决于 FileStream 对象打开文件的方式：是通过调用 open() 方法还是通过调用 openAsync() 方法。

使用异步版本可以启动在后台运行的进程，然后在完成时 （或出现错误事件时）调度事件。在运行异步后台进程的同时可以执行其他代码。使用操作的异步版本时，必须使用调用该函数的 File 或 FileStream 对象的 addEventListener() 方法设置事件侦听器函数。

使用同步版本可以编写较简单的代码，而无需设置事件侦听器。不过，由于在执行同步方法的同时无法执行其他代码，可能会延迟重要的进程 （如显示对象呈现和动画）。

使用 AIR 中的 File 对象

Adobe AIR 1.0 和更高版本

File 对象是指向文件系统中文件或目录的指针。

File 类扩展了 FileReference 类。 Adobe® Flash® Player 和 AIR 中提供的 FileReference 类表示指向文件的指针。 File 类添加了一些属性和方法，出于安全方面的考虑，在 Flash Player 中 （在浏览器中运行的 SWF 文件中）未公开这些属性和方法。

关于 File 类

Adobe AIR 1.0 和更高版本

您可以使用 File 类执行以下操作：

- 获取特殊目录的路径，包括用户目录、用户的文档目录、应用程序的启动目录以及应用程序目录
- 复制文件和目录
- 移动文件和目录
- 删除文件和目录 （或将它们移到垃圾桶）
- 列出目录中包含的文件和目录
- 创建临时文件和文件夹

当 File 对象指向文件路径后，您可以通过 FileStream 类使用该 File 对象读取和写入文件数据。

File 对象可以指向尚不存在的文件或目录的路径。创建文件或目录时可以使用这种 File 对象。

File 对象的路径

Adobe AIR 1.0 和更高版本

每个 File 对象具有两个属性，各属性可分别定义该对象的路径：

属性	说明
nativePath	指定文件在特定平台上的路径。例如，在 Windows 中，路径可能是 “c:\Sample directory\test.txt”，而在 Mac OS 中，路径可能是 “/Sample directory/test.txt”。 nativePath 属性在 Windows 中使用反斜杠 (\) 字符作为目录分隔符，在 Mac OS 和 Linux 中使用正斜杠 (/) 字符。
url	此属性可以使用 file URL 方案指向文件。例如，在 Windows 中，路径可能是 “file:///c:/Sample%20directory/test.txt”，而在 Mac OS 中，路径可能是 “file:///Sample%20directory/test.txt”。除 file 之外，运行时还包括其他特殊 URL 方案，在第 581 页的 “支持的 AIR URL 方案 ” 中将予以介绍

File 类包括用于指向 Mac OS、 Windows 和 Linux 中的标准目录的静态属性。这些属性包括：

- File.applicationStorageDirectory — 每个已安装的 AIR 应用程序独有的存储目录。此目录适用于存储动态应用程序资源和用户首选项。考虑在其他位置存储大量数据。

  在 Android 和 iOS 上，当卸载应用程序或用户选择清除应用程序数据时，会删除应用程序存储目录，但这不适用于其他平台。

- File.applicationDirectory — 安装应用程序的目录 （还存储任何安装的资源）。在有些操作系统上，应用程序存储在一个软件包文件中而不是物理目录中。在这种情况下，可能无法使用本机路径访问内容。应用程序目录是只读的。
- File.desktopDirectory— 用户的桌面目录。如果平台不定义桌面目录，则使用文件系统上的另一个位置。
- File.documentsDirectory— 用户的文档目录。如果平台不定义文档目录，则使用文件系统上的另一个位置。

- File.userDirectory — 用户目录。如果平台不定义用户目录，则使用文件系统上的另一个位置。

注：当平台不定义桌面、文档或用户目录的标准位置时， File.documentsDirectory、 File.desktopDirectory 和 File.userDirectory 可以引用同一个目录。

这些属性在不同的操作系统上有不同的值。例如，对于用户的桌面目录， Mac 和 Windows 有各自不同的本机路径。然而，File.desktopDirectory 属性在每个平台上指向适当的目录路径。要编写可以跨平台正常工作的应用程序，在需要引用应用程序使用的其他目录和文件时，请以这些属性为基础，然后使用 resolvePath() 方法来完善路径。例如，此代码会指向应用程序存储目录中的 preferences.xml 文件：

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("preferences.xml");
```

尽管可以使用 File 类来指向特定文件路径，但这种做法会指向无法跨平台工作的应用程序。例如，路径 C:\Documents and Settings\joe\ 仅适用于 Windows。出于以上原因，最好使用 File 类的静态属性，如 File.documentsDirectory。

公用目录位置

平台	目录类型	典型的文件系统位置
Android	应用程序	/data/data/
	应用程序存储	/data/data/air.applicationID/filename/Local Store
	缓存	/data/data/applicationID/cache
	桌面	/mnt/sdcard
	文档	/mnt/sdcard
	临时	/data/data/applicationID/cache/FlashTmp.randomString
	用户	/mnt/sdcard
iOS	应用程序	/var/mobile/Applications/uid/filename.app
	应用程序存储	/var/mobile/Applications/uid/Library/Application Support/applicationID/Local Store
	缓存	/var/mobile/Applications/uid/Library/Caches
	桌面	不可访问
	文档	/var/mobile/Applications/uid/Documents
	临时	/private/var/mobile/Applications/uid/tmp/FlashTmpNNN
	用户	不可访问
Linux	应用程序	/opt/filename/share
	应用程序存储	/home/userName/.appdata/applicationID/Local Store
	桌面	/home/userName/Desktop
	文档	/home/userName/Documents
	临时	/tmp/FlashTmp.randomString
	用户	/home/userName
Mac	应用程序	/Applications/filename.app/Contents/Resources
	应用程序存储	/Users/*userName*/Library/Preferences/*applicationid*/Local Store （AIR 3.2 和早期版本） *path*/Library/Application Support/*applicationid*/Local Store （AIR 3.3 和更高版本），其中 path 为 /Users/*userName*/Library/Containers/*bundle-id*/Data （沙箱环境）或 /Users/userName （在沙箱环境外运行时）
	缓存	/Users/userName/Library/Caches
	桌面	/Users/*userName*/Desktop
	文档	/Users/*userName*/Documents
	临时	/private/var/folders/JY/randomString/TemporaryItems/FlashTmp
	用户	/Users/*userName*

平台	目录类型	典型的文件系统位置
Windows	应用程序	C:\Program Files\filename
	应用程序存储	C:\Documents and settings\userName\ApplicationData\applicationID\Local Store
	缓存	C:\Documents and settings\userName\Local Settings\Temp
	桌面	C:\Documents and settings\userName\Desktop
	文档	C:\Documents and settings\userName\My Documents
	临时	C:\Documents and settings\userName\Local Settings\Temp\randomString.tmp
	用户	C:\Documents and settings\userName

根据具体的操作系统和计算机配置，这些目录的实际本机路径会有所不同。此表中显示的路径是典型示例。您应该始终使用适当的静态 File 类属性引用这些目录，以便您的应用程序在任何平台上都能正常工作。在一个实际的 AIR 应用程序中，表中显示的 applicationID 和 filename 的值取自应用程序描述符。如果您在应用程序描述符中指定发布者 ID，则发布者 ID 在这些路径中会追加到应用程序 ID。 userName 的值是安装用户的帐户名称。

将 File 对象指向目录

Adobe AIR 1.0 和更高版本

可以采用多种不同方式设置 File 对象以使其指向某目录。

指向用户的主目录

Adobe AIR 1.0 和更高版本

您可以将 File 对象指向用户的主目录。以下代码将设置 File 对象以使其指向主目录中的 AIR Test 子目录：

```
var file:File = File.userDirectory.resolvePath("AIR Test");
```

指向用户的文档目录

Adobe AIR 1.0 和更高版本

您可以将 File 对象指向用户的文档目录。以下代码设置 File 对象以指向文档目录中的 AIR Test 子目录：

```
var file:File = File.documentsDirectory.resolvePath("AIR Test");
```

指向桌面目录

Adobe AIR 1.0 和更高版本

您可以使 File 对象指向桌面。以下代码设置 File 对象以使其指向桌面的 AIR Test 子目录：

```
var file:File = File.desktopDirectory.resolvePath("AIR Test");
```

指向应用程序存储目录

Adobe AIR 1.0 和更高版本

您可以使 File 对象指向应用程序存储目录。对于每个 AIR 应用程序，有一个唯一的关联路径用于定义应用程序存储目录。此目录对每个应用程序和用户是唯一的。您可以使用此目录存储特定于用户、特定于应用程序的数据 （如用户数据或首选参数文件）。例如，以下代码将使 File 对象指向应用程序存储目录中包含的首选参数文件 prefs.xml：

```
var file:File = File.applicationStorageDirectory;
file = file.resolvePath("prefs.xml");
```

应用程序存储目录位置通常基于用户名称和应用程序 ID。此处提供了下列文件系统位置以帮助您调试应用程序。您应该始终使用 File.applicationStorage 属性或 app-storage: URI 方案解析此目录中的文件：

- 在 Mac 操作系统上 — 因 AIR 版本而异：

  AIR 3.2 和早期版本：/Users/user name/Library/Preferences/***applicationID***/Local Store/

  AIR 3.3 和更高版本：***path***/Library/Application Support/***applicationID***/Local Store，其中 ***path*** 为 /Users/***username***/Library/Containers/***bundle-id***/Data （沙箱环境）或 /Users/***username*** （在沙箱环境外运行时）

  例如 (AIR 3.2)：

  ```
  /Users/babbage/Library/Preferences/com.example.TestApp/Local Store
  ```

- 在 Windows 中，位于 Documents and Settings 目录下的以下位置：

  C:\Documents and Settings\user name\Application Data\applicationID\Local Store\

  例如：

  ```
  C:\Documents and Settings\babbage\Application Data\com.example.TestApp\Local Store
  ```

- 在 Linux 中位于：

  /home/user name/.appdata/applicationID/Local Store/

  例如：

  ```
  /home/babbage/.appdata/com.example.TestApp/Local Store
  ```

- 在 Android 上，在以下位置：

  /data/data/androidPackageID/applicationID/Local Store

  例如：

  ```
  /data/data/air.com.example.TestApp/com.example.TestApp/Local Store
  ```

注：如果应用程序具有发行商 ID，则还可将该 ID 用作应用程序存储目录路径的一部分。

通过 File.applicationStorageDirectory 创建的 File 对象的 URL （和 url 属性）将使用 app-storage URL 方案 （请参阅第 581 页的 “ 支持的 AIR URL 方案 ”），如下所示：

```
var dir:File = File.applicationStorageDirectory;
dir = dir.resolvePath("preferences");
trace(dir.url); // app-storage:/preferences
```

指向应用程序目录

Adobe AIR 1.0 和更高版本

您可以使 File 对象指向应用程序的安装目录，即应用程序目录。您可以使用 File.applicationDirectory 属性引用此目录。您可以使用此目录检查应用程序描述符文件或与应用程序一起安装的其他资源。例如，以下代码将使 File 对象指向应用程序目录中名为 images 的目录：

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
```

通过 File.applicationDirectory 创建的 File 对象的 URL （和 url 属性）将使用 app URL 方案 （请参阅第 581 页的 “ 支持的 AIR URL 方案 ”），如下所示：

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("images");
trace(dir.url); // app:/images
```

注：在 Android 上，通过 nativePath 无法访问应用程序软件包中的文件。 nativePath 属性是空字符串。始终使用 URL （而不是本机路径）访问应用程序目录中的文件。

指向缓存目录

Adobe AIR 3.6 和更高版本

您可以使用 File.cacheDirectory 属性使 File 对象指向操作系统的临时目录或缓存目录。该目录包含的是对应用程序运行并非必需的临时文件，因此删除这些文件时不会产生问题或对用户造成数据损失。

在大多数操作系统中，缓存目录都是临时目录。在 iOS 上，缓存目录对应于应用程序库的 Caches 目录。该目录中的文件不会备份到在线存储中，如果设备的可用存储空间过少，操作系统可能会将这些文件删除。有关详细信息，请参阅第 581 页的 “ 控制文件的备份和缓存 ”。

指向文件系统根目录

Adobe AIR 1.0 和更高版本

File.getRootDirectories() 方法列出所有根卷，如 Windows 计算机中的 C: 和已装好的卷。在 Mac OS 和 Linux 中，此方法始终返回计算机的唯一根目录 （“/” 目录）。 StorageVolumeInfo.getStorageVolumes() 方法提供有已关装的存储卷的更多详细信息（请参阅第 590 页的 “ 使用存储卷 ”）。

注：在 Android 上，文件系统的根目录是不可读的。返回引用具有本机路径 “/” 的目录的 File 对象，但是该对象的属性没有准确值。例如， spaceAvailable 始终为 0。

指向明确的目录

Adobe AIR 1.0 和更高版本

通过设置 File 对象的 nativePath 属性，可以使 File 对象指向某个明确的目录，如以下示例中所示 （在 Windows 中）：

```
var file:File = new File();
file.nativePath = "C:\\AIR Test";
```

重要说明：通过这种方式指向明确的路径会导致代码无法跨平台使用。例如，上面的示例仅适用于 Windows。您可以使用 File 类的静态属性 （如 File.applicationStorageDirectory）来定位跨平台工作的目录。然后使用 resolvePath() 方法 （请参阅下一节）导航到相对路径。

导航到相对路径

Adobe AIR 1.0 和更高版本

您可以使用 resolvePath() 方法获取相对于其他给定路径的路径。例如，以下代码将设置 File 对象以使其指向用户主目录中的 “AIR Test” 子目录：

```
var file:File = File.userDirectory;
file = file.resolvePath("AIR Test");
```

您还可以使用 File 对象的 url 属性以使该对象指向基于 URL 字符串的目录，如下所示：

```
var urlStr:String = "file:///C:/AIR Test/";
var file:File = new File()
file.url = urlStr;
```

有关详细信息，请参阅第 581 页的 “ 修改文件路径 ”。

让用户浏览以选择目录

Adobe AIR 1.0 和更高版本

File 类包括 browseForDirectory() 方法，它表示系统对话框，在该对话框中用户可以选择要分配给对象的目录。browseForDirectory() 方法为异步方法。如果用户选择一个目录并单击 " 打开 " 按钮， File 对象将调度一个 select 事件；如果用户单击 " 取消 " 按钮，它将调度一个 cancel 事件。

例如，以下代码能使用户选择一个目录，并在选择后输出目录路径：

```
var file:File = new File();
file.addEventListener(Event.SELECT, dirSelected);
file.browseForDirectory("Select a directory");
function dirSelected(e:Event):void {
    trace(file.nativePath);
}
```

注：在 Android 上，不支持 browseForDirectory() 方法。调用此方法没有效果；会立即调度 cancel 事件。相反，要允许用户选择目录，则应改用应用程序定义的自定义对话框。

指向从中调用应用程序的目录

Adobe AIR 1.0 和更高版本

通过检查调用应用程序时所调度的 InvokeEvent 对象的 currentDirectory 属性，可以获取从中调用应用程序的目录位置。有关详细信息，请参阅第 752 页的 " 捕获命令行参数 "。

将 File 对象指向文件

Adobe AIR 1.0 和更高版本

可采用多种不同方式设置 File 对象所指向的文件。

指向明确的文件路径

Adobe AIR 1.0 和更高版本

重要说明：指向明确的路径会导致代码无法跨平台工作。例如，路径 C:/foo.txt 仅适用于 Windows。您可以使用 File 类的静态属性 （如 File.applicationStorageDirectory）来定位跨平台工作的目录。然后使用 resolvePath() 方法 （请参阅第 581 页的 " 修改文件路径 "）导航到相对路径。

您可以使用 File 对象的 url 属性以使该对象指向基于 URL 字符串的文件或目录，如下所示：

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File()
file.url = urlStr;
```

您还可以将 URL 传递到 File() 构造函数，如下所示：

```
var urlStr:String = "file:///C:/AIR Test/test.txt";
var file:File = new File(urlStr);
```

url 属性始终返回 URL 的 URI 编码版本 （例如，空格替换为 %20）：

```
file.url = "file:///c:/AIR Test";
trace(file.url); // file:///c:/AIR%20Test
```

您还可以使用 File 对象的 nativePath 属性设置明确的路径。例如，在 Windows 计算机中运行以下代码，可以设置 File 对象以使其指向 C: 驱动器的 AIR Test 子目录中的 test.txt 文件：

```
var file:File = new File();
file.nativePath = "C:/AIR Test/test.txt";
```

您还可以将此路径传递到 File() 构造函数，如下所示：

```
var file:File = new File("C:/AIR Test/test.txt");
```

请使用正斜杠 (/) 字符作为 nativePath 属性的路径分隔符。在 Windows 上，还可以使用反斜杠 (\) 字符，但这会导致应用程序无法跨平台工作。

有关详细信息，请参阅第 581 页的 " 修改文件路径 "。

枚举目录中的文件

Adobe AIR 1.0 和更高版本

您可以使用 File 对象的 getDirectoryListing() 方法获取指向位于某目录根级的文件和子目录的 File 对象数组。有关详细信息，请参阅第 586 页的 " 枚举目录 "。

让用户浏览以选择文件

Adobe AIR 1.0 和更高版本

File 类包括以下方法，它们表示系统对话框，在该对话框中用户可以选择要分配给对象的文件：

- browseForOpen()
- browseForSave()
- browseForOpenMultiple()

这些方法均为异步方法。当用户选择一个文件时（或者，对于 browseForSave() 选择一个目标路径时），browseForOpen() 和 browseForSave() 方法将调度 select 事件。对 browseForOpen() 和 browseForSave() 方法，在进行选择后目标 File 对象将指向所选的文件。当用户选择多个文件时，browseForOpenMultiple() 方法调度一个 selectMultiple 事件。selectMultiple 事件的类型是 FileListEvent，它具有一个 files 属性，该属性是一个 File 对象数组（指向所选的文件）。

例如，以下代码向用户显示 "Open" 对话框，在该对话框中用户可以选择文件：

```
var fileToOpen:File = File.documentsDirectory;
selectTextFile(fileToOpen);

function selectTextFile(root:File):void
{
    var txtFilter:FileFilter = new FileFilter("Text", "*.as;*.css;*.html;*.txt;*.xml");
    root.browseForOpen("Open", [txtFilter]);
    root.addEventListener(Event.SELECT, fileSelected);
}

function fileSelected(event:Event):void
{
    trace(fileToOpen.nativePath);
}
```

当您调用浏览方法时，如果应用程序已打开了其他浏览器对话框，则运行时会引发一个错误异常。

注：在 Android 上，只能使用 browseForOpen() 和 browseForOpenMultiple() 方法选择图像、视频和音频文件。尽管用户可以输入任意文件名，但 browseForSave() 对话框仍仅显示媒体文件。为了打开和保存非媒体文件，您应该考虑使用自定义对话框而不是这些方法。

修改文件路径

Adobe AIR 1.0 和更高版本

通过调用 resolvePath() 方法或通过修改对象的 nativePath 或 url 属性，您还可以修改现有 File 对象的路径，如以下示例中所示（在 Windows 中）：

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
trace(file1.nativePath); // C:\Documents and Settings\userName\My Documents\AIR Test
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("..");
trace(file2.nativePath); // C:\Documents and Settings\userName
var file3:File = File.documentsDirectory;
file3.nativePath += "/subdirectory";
trace(file3.nativePath); // C:\Documents and Settings\userName\My Documents\subdirectory
var file4:File = new File();
file4.url = "file:///c:/AIR Test/test.txt";
trace(file4.nativePath); // C:\AIR Test\test.txt
```

使用 nativePath 属性时，请使用正斜杠 (/) 字符作为目录分隔符。在 Windows 上，还可以使用反斜杠 (\) 字符，但不应这样做，因为这会导致代码无法跨平台工作。

支持的 AIR URL 方案

Adobe AIR 1.0 和更高版本

在 AIR 中定义 File 对象的 url 属性时，可以使用以下任一 URL 方案：

URL 方案	说明
file	用于指定相对于文件系统根目录的路径。例如： `file:///c:/AIR Test/test.txt` URL 标准规定 file URL 采用 file://<host>/<path> 形式。作为一个特例， <host> 可以是空字符串，它被解释为 “ 解释该 URL 的计算机 ”。因此， file URL 通常具有三个斜杠 (///)。
app	用于指定相对于所安装应用程序的根目录 （该目录包含所安装应用程序的 application.xml 文件）的路径。例如，以下路径指向所安装应用程序的目录的 images 子目录： `app:/images`
app-storage	用于指定相对于应用程序存储目录的路径。对于每个安装的应用程序， AIR 定义了一个唯一的应用程序存储目录，此目录对于存储特定于该应用程序的数据很有用。例如，以下路径指向应用程序存储目录的 settings 子目录中的 prefs.xml 文件： `app-storage:/settings/prefs.xml`

控制文件的备份和缓存

Adobe AIR 3.6 和更高版本，仅针对 **iOS** 和 **OS X**

某些操作系统 （特别是 iOS 和 Mac OS X）允许用户将应用程序文件自动备份到远程存储。此外，对于文件是否可以备份以及不同用途的文件可以存储在哪里，在 iOS 上会有相关限制。

以下摘要说明了如何遵循 Apple 关于文件备份和存储的准则。有关详细信息，请参阅接下来的部分。

- 若要指定某文件不需要备份且在设备存储空间过小时可由操作系统删除 （仅针对 iOS），可将该文件保存在缓存目录 (File.cacheDirectory) 中。这是 iOS 上的首选存储位置，对于大多数可以重新生成或重新下载的文件都应使用此位置。

- 若要指定某文件不需要备份但不应由操作系统删除，可将该文件保存在其中一个应用程序库目录中，如应用程序存储目录 (File.applicationStorageDirectory) 或文档目录 (File.documentsDirectory)。将 File 对象的 preventBackup 属性设置为 true。这是 Apple 对于可以重新生成或重新下载内容的必需要求，但不是您的应用程序在脱机使用期间正常工作的必需要求。

指定用于备份的文件

为节省备份空间并降低网络带宽的使用， Apple 的 iOS 和 Mac 应用程序准则指明，只应将包含用户输入数据的文件或包含无法重新生成或重新下载数据的文件指定为予以备份。

默认情况下，应用程序库文件夹中的所有文件都将进行备份。在 Mac OS X 上，这是应用程序存储目录。在 iOS 上，则包括应用程序存储目录、应用程序目录、桌面目录、文档目录及用户目录 （因为在 iOS 上这些目录将映射到应用程序库文件夹）。因此，默认情况下这些目录中的任何文件都将备份到服务器存储。

如果要将文件保存在其中一个可以由您的应用程序重新创建的位置，您应对该文件进行标识以便操作系统知道不用对它进行备份。要指明不应备份某个文件，可将 File 对象的 preventBackup 属性设为 true。

请注意，在 iOS 上，对于任何应用程序库文件夹中的文件，即便该文件的 preventBackup 属性设为 true，也应将该文件标记为操作系统不应删除的永久性文件。

控制文件的缓存和删除

Apple 的 iOS 应用程序准则指明，对于可以重新生成的内容，应尽可能让操作系统可以将其删除，以防设备存储空间过少。

在 iOS 上，应用程序库文件夹 （如应用程序存储目录或文档目录）中的文件应标识为永久性文件，不让操作系统删除。

对于可以由应用程序重新生成且可以安全删除以防应用程序缓存目录中的存储空间过少这样的文件，应当予以保存。可使用 File.cacheDirectory 静态属性来访问缓存目录。

在 iOS 上，缓存目录对应于应用程序的缓存目录 （< 应用程序主目录 >/Library/Caches）。在其他操作系统上，该目录映射到一个类似的目录。例如，在 Mac OS X 上，它也映射到应用程序库中的 Caches 目录。在 Android 上，缓存目录映射到应用程序的缓存目录。在 Windows 上，缓存目录映射到操作系统临时目录。在 Android 和 Windows 上， File 类的 createTempDirectory() 和 createTempFile() 方法调用访问的就是这个目录。

查找两个文件之间的相对路径

Adobe AIR 1.0 和更高版本

您可以使用 getRelativePath() 方法查找两个文件之间的相对路径：

```
var file1:File = File.documentsDirectory.resolvePath("AIR Test");
var file2:File = File.documentsDirectory
file2 = file2.resolvePath("AIR Test/bob/test.txt");

trace(file1.getRelativePath(file2)); // bob/test.txt
```

getRelativePath() 方法的第二个参数 useDotDot 允许在结果中返回 .. 语法，以指示父目录：

```
var file1:File = File.documentsDirectory;
file1 = file1.resolvePath("AIR Test");
var file2:File = File.documentsDirectory;
file2 = file2.resolvePath("AIR Test/bob/test.txt");
var file3:File = File.documentsDirectory;
file3 = file3.resolvePath("AIR Test/susan/test.txt");

trace(file2.getRelativePath(file1, true)); // ../..
trace(file3.getRelativePath(file2, true)); // ../../bob/test.txt
```

获取文件名的规范版本

Adobe AIR 1.0 和更高版本

文件名和路径名在 Windows 和 Mac OS 中不区分大小写。在以下示例中，两个 File 对象指向同一个文件：

```
File.documentsDirectory.resolvePath("test.txt");
File.documentsDirectory.resolvePath("TeSt.TxT");
```

不过，文档和目录名确实包括大小写。例如，以下代码假定在文档目录中有一个名为 AIR Test 的文件夹，如以下示例中所示：

```
var file:File = File.documentsDirectory.resolvePath("AIR test");
trace(file.nativePath); // ... AIR test
file.canonicalize();
trace(file.nativePath); // ... AIR Test
```

canonicalize() 方法可以转换 nativePath 对象，以使用文件名或目录名的正确大写形式。在区分大小写的文件系统 （如 Linux）上，当多个文件的名称只有大小写不同时， canonicalize() 方法将调整路径以匹配最先找到的文件 （以文件系统确定的顺序）。

在 Windows 中，您还可以使用 canonicalize() 方法将短文件名 （“8.3” 名称）转换为长文件名，如以下示例中所示：

```
var path:File = new File();
path.nativePath = "C:\\AIR~1";
path.canonicalize();
trace(path.nativePath); // C:\AIR Test
```

使用包和符号链接

Adobe AIR 1.0 和更高版本

多种操作系统支持包文件和符号链接文件：

包 — 在 Mac OS 中，可以指定目录作为包，并且目录可以作为单个文件而非目录出现在 Mac OS Finder 中。

符号链接 — Mac OS、 Linux 和 Windows Vista 支持符号链接。通过符号链接，文件可以指向磁盘上的另一个文件或目录。尽管符号链接与别名类似，不过它们并不相同。别名始终报告为文件 （而不是目录），读取或写入别名或快捷方式从不影响它指向的原始文件或目录。另一方面，符号链接的行为则完全与它指向的文件或目录类似。可以将符号链接报告为文件或目录，并且读写符号链接影响的是符号链接所指向的文件或目录，而不影响其本身。此外，在 Windows 中，引用交接点 （用于 NTFS 文件系统中）的 File 对象的 isSymbolicLink 属性设置为 true。

File 类包括 isPackage 和 isSymbolicLink 属性，用于检查 File 对象是否引用包或符号链接。

以下代码将遍历用户的桌面目录，列出不是 包的子目录：

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isDirectory && !desktopNodes[i].isPackage)
    {
        trace(desktopNodes[i].name);
    }
}
```

以下代码将遍历户的桌面目录，列出不是 符号链接的文件和目录：

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (!desktopNodes[i].isSymbolicLink)
    {
        trace(desktopNodes[i].name);
    }
}
```

canonicalize() 方法可更改符号链接的路径，以指向该链接所引用的文件或目录。以下代码将遍历用户的桌面目录，报告由是符号链接的文件引用的路径：

```
var desktopNodes:Array = File.desktopDirectory.getDirectoryListing();
for (var i:uint = 0; i < desktopNodes.length; i++)
{
    if (desktopNodes[i].isSymbolicLink)
    {
        var linkNode:File = desktopNodes[i] as File;
        linkNode.canonicalize();
        trace(linkNode.nativePath);
    }
}
```

确定卷上的可用空间

Adobe AIR 1.0 和更高版本

File 对象的 spaceAvailable 属性是 File 位置的可用空间 （以字节为单位）。例如，以下代码检查应用程序存储目录中的可用空间：

```
trace(File.applicationStorageDirectory.spaceAvailable);
```

如果 File 对象引用一个目录，则 spaceAvailable 属性将指示可供文件使用的目录空间。如果 File 对象引用一个文件，则 spaceAvailable 属性将指示可供该文件使用的空间。如果 File 位置不存在，则 spaceAvailable 属性将设置为 0。如果 File 对象引用一个符号链接，则 spaceAvailable 属性将设置为符号链接指向的位置的可用空间。

通常，目录或文件的可用空间与包含目录或文件的卷上的可用空间相同。不过，可用空间与磁盘配额及每个目录的空间限制有关。

将文件或目录添加到卷中通常需要比文件的实际大小或目录中内容的实际大小更多的空间。例如，操作系统可能需要更多空间来存储索引信息。或者，所需的磁盘扇区可能会使用额外的空间。此外，可用空间是动态变化的。因此，您不能期望为文件存储分配报告的全部空间。有关写入文件系统的信息，请参阅第 591 页的 “ 读取和写入文件 ”。

StorageVolumeInfo.getStorageVolumes() 方法提供有已关装的存储卷的更多详细信息 （请参阅第 590 页的 “ 使用存储卷 ”）。

使用默认系统应用程序打开文件

Adobe AIR 2 和更高版本

在 AIR 2 中，您可以使用操作系统注册的用来打开某文件的应用程序打开该文件。例如，AIR 应用程序可以使用注册的用来打开文档文件的应用程序打开一个文档文件。使用 File 对象的 openWithDefaultApplication() 方法打开该文件。例如，以下代码打开用户桌面上名为 test.doc 的文件，并且打开该文件所用的是与文档文件相对应的默认应用程序：

```
var file:File = File.deskopDirectory;
file = file.resolvePath("test.doc");
file.openWithDefaultApplication();
```

注：在 Linux 中，文件的 MIME 类型 （而不是文件扩展名）确定文件的默认应用程序。

以下代码使用户可以导航到一个 mp3 文件，并在用于播放 mp3 文件的默认应用程序中打开它：

```
var file:File = File.documentsDirectory;
var mp3Filter:FileFilter = new FileFilter("MP3 Files", "*.mp3");
file.browseForOpen("Open", [mp3Filter]);
file.addEventListener(Event.SELECT, fileSelected);

function fileSelected(e:Event):void
{
    file.openWithDefaultApplication();
}
```

无法对位于应用程序目录中的文件使用 openWithDefaultApplication() 方法。

AIR 会阻止您使用 openWithDefaultApplication() 方法打开某些文件。在 Windows 中， AIR 会阻止您打开某些文件类型的文件，例如 EXE 或 BAT。在 Mac OS 和 Linux 中，AIR 阻止您打开将在某些应用程序中启动的文件。(其中包括 Mac OS 中的 Terminal 和 AppletLauncher 以及 Linux 中的 csh、bash 或 ruby。) 尝试使用 openWithDefaultApplication() 方法打开其中一个文件将导致异常。有关阻止打开的文件类型的完整列表，请参阅 File.openWithDefaultApplication() 方法的语言参考条目。

注：对于使用本机安装程序 （一种扩展桌面应用程序）安装的 AIR 应用程序不存在这种限制。

获取文件系统信息

Adobe AIR 1.0 和更高版本

File 类包括以下可提供有关文件系统的一些有用信息的静态属性：

属性	说明
File.lineEnding	主机操作系统使用的行结束字符序列。在 Mac OS 和 Linux 中，这是换行符。在 Windows 中，它是回车符后跟换行符。
File.separator	主机操作系统的路径组件分隔符。在 Mac OS 和 Linux 中，这是正斜杠 (/) 字符。在 Windows 中，它是反斜杠 (\) 字符。
File.systemCharset	主机操作系统为文件使用的默认编码。此属性与操作系统使用的字符集有关，与操作系统语言相对应。

Capabilities 类还包括有用的系统信息，在使用文件时这些信息可能很有用：

属性	说明
Capabilities.hasIME	指定播放器是在安装有 (true) 输入法编辑器 (IME) 的系统上运行，还是在未安装 (false) IME 的系统上运行。
Capabilities.language	指定运行播放器的系统的语言代码。
Capabilities.os	指定当前的操作系统。

注：使用 Capabilities.os 确定系统特性时，应务必小心。如果有更加具体的属性可用来确定系统特性，请使用该属性。否则，您可能面临所写代码无法在所有平台上正常工作的风险。例如，请看以下代码：

```
var separator:String;
if (Capablities.os.indexOf("Mac") > -1)
{
    separator = "/";
}
else
{
    separator = "\\";
}
```

此代码会导致 Linux 上出现问题。最好只使用 File.separator 属性。

使用目录

Adobe AIR 1.0 和更高版本

通过运行时提供的功能，可以使用本地文件系统上的目录。

有关创建指向目录的 File 对象的详细信息，请参阅第 576 页的 " 将 File 对象指向目录 "。

创建目录

Adobe AIR 1.0 和更高版本

使用 File.createDirectory() 方法可以创建目录。例如，以下代码创建名为 AIR Test 的目录以作为用户主目录的子目录：

```
var dir:File = File.userDirectory.resolvePath("AIR Test");
dir.createDirectory();
```

如果该目录存在， createDirectory() 方法不执行任何操作。

另外，在某些模式中， FileStream 对象在打开文件时会创建目录。如果 FileStream() 构造函数的 fileMode 参数设置为 FileMode.APPEND 或 FileMode.WRITE，则在实例化 FileStream 实例时将创建缺少的目录。有关详细信息，请参阅第 592 页的 " 读取和写入文件的工作流程 "。

创建临时目录

Adobe AIR 1.0 和更高版本

File 类包括一个 createTempDirectory() 方法，该方法可在系统的临时目录文件夹中创建一个目录，如以下示例中所示：

```
var temp:File = File.createTempDirectory();
```

createTempDirectory() 方法会自动创建一个唯一的临时目录 （您无需确定新的唯一位置）。

您可以使用临时目录暂时存储应用程序会话中使用的临时文件。请注意，有一个 createTempFile() 方法可以在系统临时目录中创建新的、唯一的临时文件。

您可能需要在关闭应用程序前删除临时目录，因为不会 在所有设备上自动删除临时目录。

枚举目录

Adobe AIR 1.0 和更高版本

您可以使用 File 对象的 getDirectoryListing() 方法或 getDirectoryListingAsync() 方法获取指向目录中的文件和子文件夹的 File 对象数组。

例如，以下代码将列出用户的文档目录的内容 （无需检查子目录）：

```
var directory:File = File.documentsDirectory;
var contents:Array = directory.getDirectoryListing();
for (var i:uint = 0; i < contents.length; i++)
{
    trace(contents[i].name, contents[i].size);
}
```

当使用该方法的异步版本时， directoryListing 事件对象具有一个 files 属性，该属性是与目录有关的 File 对象数组：

```
var directory:File = File.documentsDirectory;
directory.getDirectoryListingAsync();
directory.addEventListener(FileListEvent.DIRECTORY_LISTING, dirListHandler);

function dirListHandler(event:FileListEvent):void
{
    var contents:Array = event.files;
    for (var i:uint = 0; i < contents.length; i++)
    {
        trace(contents[i].name, contents[i].size);
    }
}
```

复制和移动目录

Adobe AIR 1.0 和更高版本

您可以使用与复制或移动文件相同的方法复制或移动目录。例如，以下代码将以同步方式复制目录：

```
var sourceDir:File = File.documentsDirectory.resolvePath("AIR Test");
var resultDir:File = File.documentsDirectory.resolvePath("AIR Test Copy");
sourceDir.copyTo(resultDir);
```

当您将 copyTo() 方法的 overwrite 参数指定为 true 时，现有目标目录中的所有文件和文件夹都将删除，并替换为源目录中的文件和文件夹 （即使在源目录中目标文件不存在）。

指定为 copyTo() 方法的 newLocation 参数的目录将指定所生成的目录的路径，它不 指定将包含所生成的目录的父 目录。

有关详细信息，请参阅第 589 页的 “ 复制和移动文件 ”。

删除目录内容

Adobe AIR 1.0 和更高版本

File 类包括一个 deleteDirectory() 方法和一个 deleteDirectoryAsync() 方法。这些方法删除目录，第一个方法以同步方式运行，第二个方法以异步方式运行 （请参阅 第 572 页的 “AIR 文件基础知识 ”）。两个方法都包括一个 deleteDirectoryContents 参数（该参数取布尔值）；当此参数设置为 true 时 （默认值为 false），调用该方法将删除非空目录；否则，只删除空目录。

例如，以下代码以同步方式删除用户的文档目录中的 AIR Test 子目录：

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.deleteDirectory(true);
```

以下代码以异步方式删除用户的文档目录中的 AIR Test 子目录：

```
var directory:File = File.documentsDirectory.resolvePath("AIR Test");
directory.addEventListener(Event.COMPLETE, completeHandler)
directory.deleteDirectoryAsync(true);

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

此外，还包括 moveToTrash() 和 moveToTrashAsync() 方法，您可以使用这些方法将目录移到系统垃圾桶。有关详细信息，请参阅第 590 页的 “ 将文件移到垃圾桶 ”。

使用文件

Adobe AIR 1.0 和更高版本

使用 AIR 文件 API，您可以向应用程序中添加基本的文件交互功能。例如，您可以读取和写入文件、复制和删除文件等。由于您的应用程序可以访问本地文件系统，因此请参阅 第 922 页的 “AIR 安全性 ” （如果您还没有阅读该章节）。

注：您可以将文件类型与 AIR 应用程序相关联 （以便双击时可以打开应用程序）。有关详细信息，请参阅第 759 页的 “ 管理文件关联 ”。

获取文件信息

Adobe AIR 1.0 和更高版本

File 类包括以下属性，这些属性提供有关 File 对象指向的文件或目录的信息：

File 属性	说明
creationDate	本地磁盘上文件的创建日期。
creator	已废弃。请使用 extension 属性。（此属性报告文件的 Macintosh 创建者类型，此属性仅用于 Mac OS X 之前的 Mac OS 版本中。）
downloaded	（AIR 2 和更高版本）指示是否已 （从 Internet）下载引用的文件或目录。属性仅在文件可以标记为已下载的操作系统上有意义： • Windows XP Service Pack 2 和更高版本，在 Windows Vista 上 • Mac OS 10.5 和更高版本
exists	引用的文件或目录是否存在。
extension	文件扩展名，它是最后一个句点 （“.”）后面的名称部分 （不包括句点）。如果文件名中没有句点，则 extension 为 null。
icon	包含为文件定义的图标的 Icon 对象。
isDirectory	File 对象引用是否为对目录的引用。
modificationDate	本地磁盘上文件或目录的上一次修改日期。
name	本地磁盘上文件或目录的名称 （如果存在文件扩展名，则包括文件扩展名）。
nativePath	采用主机操作系统表示形式的完整路径。请参阅 第 573 页的 “File 对象的路径 ”。
parent	包含由 File 对象表示的文件夹或文件的文件夹。如果 File 对象引用的是文件系统根目录中的文件或目录，此属性为 null。
size	本地磁盘上文件的大小 （以字节为单位）。
type	已废弃。请使用 extension 属性。（在 Macintosh 中，此属性是四个字符的文件类型，它仅用于 Mac OS X 之前的 Mac OS 版本中。）
url	文件或目录的 URL。请参阅 第 573 页的 “File 对象的路径 ”。

有关这些属性的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 File 类条目。

复制和移动文件

Adobe AIR 1.0 和更高版本

File 类包括两个用于复制文件或目录的方法：copyTo() 和 copyToAsync()。 File 类包括两个用于移动文件或目录的方法：moveTo() 和 moveToAsync()。copyTo() 和 moveTo() 方法以同步方式运行，copyToAsync() 和 moveToAsync() 方法以异步方式运行 （请参阅 第 572 页的 “AIR 文件基础知识 ”）。

若要复制或移动文件，请设置两个 File 对象。一个对象指向要复制或移动的文件，它是调用复制或移动方法的对象；另一个对象指向目标 （结果）路径。

以下代码将 test.txt 文件从用户的文档目录的 AIR Test 子目录复制到同一目录中名为 copy.txt 的文件：

```
var original:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var newFile:File = File.resolvePath("AIR Test/copy.txt");
original.copyTo(newFile, true);
```

在此例中， copyTo() 方法的 overwrite 参数 （第二个参数）的值设置为 true。通过将 overwrite 设置为 true，可以覆盖现有目标文件。此参数是可选的。如果您将它设置为 false （默认值），则当目标文件存在时该操作调度一个 IOErrorEvent 事件 （文件没有复制）。

复制和移动方法的 “ 异步 ” 版本以异步方式运行。使用 addEventListener() 方法可以监视任务是否完成或错误条件，如以下代码中所示：

```
var original = File.documentsDirectory;
original = original.resolvePath("AIR Test/test.txt");

var destination:File = File.documentsDirectory;
destination =  destination.resolvePath("AIR Test 2/copy.txt");

original.addEventListener(Event.COMPLETE, fileMoveCompleteHandler);
original.addEventListener(IOErrorEvent.IO_ERROR, fileMoveIOErrorEventHandler);
original.moveToAsync(destination);

function fileMoveCompleteHandler(event:Event):void {
    trace(event.target); // [object File]
}
function fileMoveIOErrorEventHandler(event:IOErrorEvent):void {
    trace("I/O Error.");
}
```

File 类还包括 File.moveToTrash() 和 File.moveToTrashAsync() 方法，它们将文件或目录移到系统垃圾桶。

删除文件

Adobe AIR 1.0 和更高版本

File 类包括一个 deleteFile() 方法和一个 deleteFileAsync() 方法。这些方法删除文件，第一个方法以同步方式运行，第二个方法以异步方式运行 （请参阅 第 572 页的 “AIR 文件基础知识 ”）。

例如，以下代码以同步方式删除用户的文档目录中的 test.txt 文件：

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.deleteFile();
```

以下代码以异步方式删除用户的文档目录中的 test.txt 文件：

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, completeHandler)
file.deleteFileAsync();

function completeHandler(event:Event):void {
    trace("Deleted.")
}
```

此外，还包括 moveToTrash() 和 moveToTrashAsync 方法，您可以使用这些方法将文件或目录移到系统垃圾桶。有关详细信息，请参阅第 590 页的 “ 将文件移到垃圾桶 ”。

将文件移到垃圾桶

Adobe AIR 1.0 和更高版本

File 类包括一个 moveToTrash() 方法和一个 moveToTrashAsync() 方法。这些方法将文件或目录发送到系统垃圾桶，第一个方法以同步方式运行，第二个方法以异步方式运行 （请参阅 第 572 页的 “AIR 文件基础知识 ”）。

例如，以下代码以同步方式将用户的文档目录中的 test.txt 文件移到系统垃圾桶：

```
var file:File = File.documentsDirectory.resolvePath("test.txt");
file.moveToTrash();
```

注：在不支持可恢复垃圾桶文件夹概念的操作系统上，会立即删除文件。

创建临时文件

Adobe AIR 1.0 和更高版本

File 类包括一个 createTempFile() 方法，该方法在系统的临时目录文件夹中创建一个文件，如以下示例中所示：

```
var temp:File = File.createTempFile();
```

createTempFile() 方法会自动创建一个唯一的临时文件 （您无需确定新的唯一位置）。

您可以使用临时文件暂时存储应用程序会话中使用的信息。请注意，还有一个 createTempDirectory() 方法可以在系统临时目录中创建唯一的临时目录。

您可能需要在关闭应用程序前删除临时文件，因为不会 在所有设备上自动删除临时文件。

使用存储卷

Adobe AIR 2 和更高版本

在 AIR 2 中，安装或卸载大容量存储卷时，您可以进行检测。 StorageVolumeInfo 类定义单个 storageVolumeInfo 对象。 StorageVolumeInfo.storageVolumeInfo 对象在存储卷安装之后调度 storageVolumeMount 事件。它在某个卷被卸载时调度 storageVolumeUnmount 事件。 StorageVolumeChangeEvent 类定义这些事件。

注：在现今的 Linux 发行版中， StorageVolumeInfo 对象仅对在特定位置装载的物理设备和网络驱动器调度 storageVolumeMount 和 storageVolumeUnmount 事件。

StorageVolumeChangeEvent 类的 storageVolume 属性是一个 StorageVolume 对象。 StorageVolume 类定义存储卷的基本属性：

- drive — Windows 上的卷驱动器号 （在其他操作系统上为 null）
- fileSystemType — 存储卷上的文件系统的类型 （如 “FAT”、 “NTFS”、 “HFS” 或 “UFS”）
- isRemoveable — 卷是可删除 （为 true）还是不可删除 （为 false）

- isWritable — 卷可写入 (true) 还是不可写入 (false)
- name — 卷的名称
- rootDirectory — 与卷的根目录对应的 File 对象

StorageVolumeChangeEvent 类还包括一个 rootDirectory 属性。rootDirectory 属性是一个引用已装载或已卸载的存储卷的根目录的 File 对象。

对于卸载的卷，未定义 StorageVolumeChangeEvent 对象的 storageVolume 属性 (null)。然而，您可以访问此事件的 rootDirectory 的属性。

以下代码在安装完存储卷之后输出该存储卷的名称和文件路径：

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME_MOUNT,
onVolumeMount);
function onVolumeMount(event:StorageVolumeChangeEvent):void
{
    trace(event.storageVolume.name, event.rootDirectory.nativePath);
}
```

以下代码在卸载完存储卷之后输出该存储卷的文件路径：

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(StorageVolumeChangeEvent.STORAGE_VOLUME_UNMOUNT,
onVolumeUnmount);
function onVolumeUnmount(event:StorageVolumeChangeEvent):void
{
    trace(event.rootDirectory.nativePath);
}
```

StorageVolumeInfo.storageVolumeInfo 对象包含 getStorageVolumes() 方法。此方法返回与当前安装的存储卷对应的 StorageVolume 对象的矢量。以下代码显示如何列出所有安装的存储卷的名称和根目录：

```
var volumes:Vector.<StorageVolume> = new Vector.<StorageVolume>;
volumes = StorageVolumeInfo.storageVolumeInfo.getStorageVolumes();
for (var i:int = 0; i < volumes.length; i++)
{
    trace(volumes[i].name, volumes[i].rootDirectory.nativePath);
}
```

注：在现今的 Linux 发行版中， getStorageVolumes() 方法返回与在特定位置安装的物理设备和网络驱动器相对应的对象。

File.getRootDirectories() 方法列出根目录 （请参阅第 578 页的 “ 指向文件系统根目录 ”）。然而， StorageVolume 对象 （由 StorageVolumeInfo.getStorageVolumes() 方法枚举）提供有关存储卷的详细信息。

可使用 StorageVolume 对象的 rootDirectory 属性的 spaceAvailable 属性查看存储卷上的可用空间。（请参阅第 584 页的 “ 确定卷上的可用空间 ”。）

更多帮助主题

StorageVolume

StorageVolumeInfo

读取和写入文件

Adobe AIR 1.0 和更高版本

FileStream 类允许 AIR 应用程序读取和写入文件系统。

读取和写入文件的工作流程

Adobe AIR 1.0 和更高版本

读取和写入文件的工作流程如下所示。

初始化指向路径的 **File** 对象。
File 对象表示您要使用的文件 （或您以后将创建的文件）的路径。

```
var file:File = File.documentsDirectory;
file = file.resolvePath("AIR Test/testFile.txt");
```

此例使用 File 对象的 File.documentsDirectory 属性和 resolvePath() 方法来初始化 File 对象。不过，有许多其他方式可以将 File 对象指向文件。有关详细信息，请参阅第 579 页的 " 将 File 对象指向文件 "。

初始化 **FileStream** 对象。

调用 **FileStream** 对象的 **open()** 方法或 **openAsync()** 方法。
具体调用哪个方法取决于您希望是以同步还是异步方式打开文件。使用 File 对象作为打开方法的 file 参数。对于 fileMode 参数，请指定 FileMode 类中的一个常量，以指定使用文件的方式。

例如，以下代码将初始化一个 FileStream 对象，该对象用于创建一个文件并覆盖所有现有数据：

```
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);
```

有关详细信息，请参阅第 593 页的 " 初始化 FileStream 对象以及打开和关闭文件 " 和 第 593 页的 "FileStream 打开模式 "。

如果您以异步方式打开了文件 （使用 **openAsync()** 方法），请为 **FileStream** 对象添加并设置事件侦听器。
这些事件侦听器方法响应在各种情形下由 FileStream 对象调度的事件。这些情形包括：从文件读取数据时、遇到 I/O 错误时、要写入的全部数据量已写入时。

有关详细信息，请参阅第 596 页的 " 异步编程和以异步方式打开的 FileStream 对象所生成的事件 "。

根据需要包含用于读取和写入数据的代码。
FileStream 类中有许多与读取和写入相关的方法。（它们都以 "read" 或 "write" 开头。）具体选择使用哪个方法读取或写入数据取决于目标文件中数据的格式。

例如，如果目标文件中的数据为 UTF 编码的文本，您可以使用 readUTFBytes() 和 writeUTFBytes() 方法。如果您希望将数据作为字节数组处理，您可以使用 readByte()、 readBytes()、 writeByte() 和 writeBytes() 方法。有关详细信息，请参阅第 597 页的 " 数据格式以及选择要使用的读取和写入方法 "。

如果以异步方式打开了文件，请确保在调用读取方法前有足够的可用数据。有关详细信息，请参阅第 595 页的 " 读取缓冲区和 FileStream 对象的 bytesAvailable 属性 "。

写入文件之前，如果要检查可用磁盘空间量，您可以检查 File 对象的 spaceAvailable 属性。有关详细信息，请参阅第 584 页的 " 确定卷上的可用空间 "。

当您处理完文件后，请调用 **FileStream** 对象的 **close()** 方法。
调用 close() 方法可使文件对其他应用程序可用。

有关详细信息，请参阅第 593 页的 " 初始化 FileStream 对象以及打开和关闭文件 "。

若要查看使用 FileStream 类读取和写入文件的范例应用程序，请参阅 Adobe AIR 开发人员中心上的以下文章：

- 构建文本文件编辑器

- 构建文本文件编辑器
- 从 XML 首选参数文件中读取和写入

使用 FileStream 对象

Adobe AIR 1.0 和更高版本

FileStream 类定义打开、读取和写入文件的方法。

FileStream 打开模式

Adobe AIR 1.0 和更高版本

FileStream 对象的 open() 和 openAsync() 方法都包括一个 fileMode 参数，该参数定义文件流的一些属性，其中包括以下属性:

- 从文件读取的能力
- 写入文件的能力
- 数据是否始终追加到文件的结尾 （当写入时）
- 当文件不存在时 （以及当文件的父目录不存在时）执行哪些操作

以下是各种文件模式 （您可以将这些模式指定为 open() 和 openAsync() 方法的 fileMode 参数）：

文件模式	说明
FileMode.READ	指定只能打开文件进行读取。
FileMode.WRITE	指定打开文件进行写入。如果文件不存在，则在打开 FileStream 对象时创建它。如果文件存在，则删除所有现有数据。
FileMode.APPEND	指定打开文件进行追加。如果文件不存在，则创建它。如果文件存在，不覆盖现有数据，所有写入操作都从文件结尾开始。
FileMode.UPDATE	指定打开文件进行读取和写入。如果文件不存在，则创建它。如果想对文件进行随机读取 / 写入访问，可以指定此模式。您可以从文件中的任何位置读取。当写入文件时，只有写入的字节会覆盖现有字节 （所有其他字节保持不变）。

初始化 FileStream 对象以及打开和关闭文件

Adobe AIR 1.0 和更高版本

当您打开 FileStream 对象后，即可使用该对象从文件读取数据和向文件写入数据。通过将 File 对象传递到 FileStream 对象的 open() 或 openAsync() 方法，可以打开 FileStream 对象：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
```

fileMode 参数 （open() 和 openAsync() 方法的第二个参数）指定打开文件的模式：read、 write、 append 或 update。有关详细信息，请参阅上一部分 第 593 页的 “FileStream 打开模式 ”。

如果您使用 openAsync() 方法打开文件进行异步文件操作，请设置事件侦听器以处理异步事件：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(IOErrorEvent.IO_Error, errorHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function completeHandler(event:Event):void {
    // ...
}

function progressHandler(event:ProgressEvent):void {
    // ...
}

function errorHandler(event:IOErrorEvent):void {
    // ...
}
```

打开文件进行同步操作还是异步操作取决于您使用 open() 方法还是 openAsync() 方法。有关详细信息，请参阅 第 572 页的 “AIR 文件基础知识 ”。

如果您在 FileStream 对象的打开方法中将 fileMode 参数设置为 FileMode.READ 或 FileMode.UPDATE，则当打开 FileStream 对象后数据将立即读入读取缓冲区中。有关详细信息，请参阅第 595 页的 “ 读取缓冲区和 FileStream 对象的 bytesAvailable 属性 ”。

您可以调用 FileStream 对象的 close() 方法关闭关联文件，使其他应用程序可以使用该文件。

FileStream 对象的 position 属性

Adobe AIR 1.0 和更高版本

FileStream 对象的 position 属性确定下一个读取或写入方法读取或写入数据的位置。

执行读取或写入操作之前，请将 position 属性设置为文件中的任何有效位置。

例如，以下代码在文件的位置 8 处写入字符串 "hello" （采用 UTF 编码）：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 8;
myFileStream.writeUTFBytes("hello");
```

当您首次打开 FileStream 对象时， position 属性设置为 0。

执行读取操作之前， position 的值必须至少为 0 并且小于文件中的字节数 （即文件中的现有位置）。

只有在以下情况下才修改 position 属性的值：

- 当您明确设置 position 属性时。
- 当您调用读取方法时。
- 当您调用写入方法时。

当您调用 FileStream 对象的读取或写入方法时， position 属性值立即增加您读取或写入的字节数。根据您使用的读取方法，position 属性可以增加您指定读取的字节数，也可以增加可用的字节数。当您随后调用读取或写入方法时，它从新的位置开始读取或写入。

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
trace(myFileStream.position); // 4200
```

不过，有一个例外：对于以 append 模式打开的 FileStream，调用写入方法后 position 属性不变。（在 append 模式中，数据始终写入文件的结尾，而与 position 属性的值无关。）

对于打开进行异步操作的文件，在下一行代码执行之前不会完成写入操作。不过，您可以连续调用多个异步方法，运行时会按顺序执行它们：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.WRITE);
myFileStream.writeUTFBytes("hello");
myFileStream.writeUTFBytes("world");
myFileStream.addEventListener(Event.CLOSE, closeHandler);
myFileStream.close();
trace("started.");

closeHandler(event:Event):void
{
    trace("finished.");
}
```

此代码的跟踪输出如下所示：

```
started.
finished.
```

您可以 在调用读取或写入方法后立即 （或在任何时间）指定 position 值，下一个读取或写入操作将从该位置开始执行。例如，请注意以下代码在调用 writeBytes() 操作后立即设置 position 属性，即使写入操作完成后 position 仍设置为该值 (300)：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.openAsync(myFile, FileMode.UPDATE);
myFileStream.position = 4000;
trace(myFileStream.position); // 4000
myFileStream.writeBytes(myByteArray, 0, 200);
myFileStream.position = 300;
trace(myFileStream.position); // 300
```

读取缓冲区和 FileStream 对象的 bytesAvailable 属性

Adobe AIR 1.0 和更高版本

当打开具有读取功能的 FileStream 对象时 （在该对象中， open() 或 openAsync() 方法的 fileMode 参数设置为 READ 或 UPDATE），运行时将数据存储在内部缓冲区中。当您打开文件后， FileStream 对象立即开始将数据读取到缓冲区中 （通过调用 FileStream 对象的 open() 或 openAsync() 方法）。

对于打开执行同步操作的文件 （使用 open() 方法），您始终可以设置 position 指针指向任何有效位置 （在文件的范围内），并开始读取任何数量的数据 （在文件的范围内），如以下代码中所示 （假定该文件包含至少 100 个字节）：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.READ);
myFileStream.position = 10;
myFileStream.readBytes(myByteArray, 0, 20);
myFileStream.position = 89;
myFileStream.readBytes(myByteArray, 0, 10);
```

无论打开文件进行同步操作还是异步操作，读取方法始终从由 bytesAvalable 属性表示的 " 可用 " 字节读取。当以同步方式读取时，任何时间文件的所有字节都是可用的。当以异步方式读取时，在由 progress 事件指示的一系列异步缓冲区填充数据中，从 position 属性指定的位置开始的字节是可用的。

对于打开进行同步 操作的文件， bytesAvailable 属性始终设置为表示从 position 属性到文件结尾的字节数 （文件中所有字节始终是可以读取的）。

对于打开进行异步 操作的文件，您需要确保在调用读取方法之前读取缓冲区已包含了足够的数据。对于以异步方式打开的文件，随着读取操作的进行，文件中从读取操作开始时指定的 position 开始的数据将添加到缓冲区，每读取一个字节 bytesAvailable 属性增加一。 bytesAvailable 属性指示从 position 属性指定的位置的字节开始到缓冲区结尾的可用字节数。 FileStream 对象定期发送一个 progress 事件。

对于以异步方式打开的文件，随着数据在读取缓冲区中可用， FileStream 对象定期调度 progress 事件。例如，当数据读取到缓冲区时，以下代码将把该数据读取到 ByteArray 对象 bytes 中：

```
var bytes:ByteArray = new ByteArray();
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);

function progressHandler(event:ProgressEvent):void
{
    myFileStream.readBytes(bytes, myFileStream.position, myFileStream.bytesAvailable);
}
```

对于以异步方式打开的文件，只能读取读取缓冲区中的数据。而且，当您读取数据后，它即从读取缓冲区中删除。对于读取操作，您需要确保在调用读取操作之前数据在读取缓冲区中存在。例如，以下代码读取文件中从位置 4000 开始的 8000 个字节：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
myFileStream.position = 4000;

var str:String = "";

function progressHandler(event:Event):void
{
    if (myFileStream.bytesAvailable > 8000 )
    {
        str += myFileStream.readMultiByte(8000, "iso-8859-1");
    }
}
```

在写入操作过程中， FileStream 对象不会将数据读入读取缓冲区中。当写入操作完成后 （写入缓冲区中所有数据都已写入文件），FileStream 对象启动一个新的读取缓冲区（假定关联的 FileStream 对象已打开并具有读取功能），并开始将从 position 属性指定的位置开始的数据读入读取缓冲区中。 position 属性可以是写入的最后一个字节的位置，也可以是其他位置 （如果在写入操作后用户为 position 对象指定了其他值）。

异步编程和以异步方式打开的 FileStream 对象所生成的事件

Adobe AIR 1.0 和更高版本

当以异步方式打开文件时 （使用 openAsync() 方法），读取和写入文件是以异步方式执行的。在将数据读入读取缓冲区以及写入输出数据时，可以执行其他 ActionScript 代码。

这表示您需要注册由以异步方式打开的 FileStream 对象所生成的事件。

通过注册 progress 事件，当有新数据可供使用时您可以收到通知，如以下代码中所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(ProgressEvent.PROGRESS, progressHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function progressHandler(event:ProgressEvent):void
{
    str += myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

通过注册 complete 事件，您可以读取全部数据，如以下代码中所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";
function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

就像输入数据将存储到缓冲区中以便可以执行异步操作一样，您在异步流上写入的数据也将存储到缓冲区中，然后以异步方式写入文件。随着数据写入文件，FileStream 对象将定期调度一个 OutputProgressEvent 对象。OutputProgressEvent 对象包括一个 bytesPending 属性，该属性设置为剩余的要写入的字节数。您可以注册 outputProgress 事件，以便当此缓冲区实际写入文件时收到通知，或者为了显示进度对话框。不过，通常情况下不需要这样做。具体而言，您可以调用 close() 方法，而不考虑未写入的字节。 FileStream 对象将继续写入数据，当最后一个字节写入文件并且基础文件关闭后将传递 close 事件。

数据格式以及选择要使用的读取和写入方法

Adobe AIR 1.0 和更高版本

每个文件是磁盘上的一组字节。在 ActionScript 中，文件中的数据始终可以表示为 ByteArray。例如，以下代码将文件中的数据读入到名为 bytes 的 ByteArray 对象中：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completeHandler);
myFileStream.openAsync(myFile, FileMode.READ);
var bytes:ByteArray = new ByteArray();

function completeHandler(event:Event):void
{
    myFileStream.readBytes(bytes, 0, myFileStream.bytesAvailable);
}
```

同样，以下代码将名为 bytes 的 ByteArray 中的数据写入文件：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.open(myFile, FileMode.WRITE);
myFileStream.writeBytes(bytes, 0, bytes.length);
```

不过，通常您不希望在 ActionScript ByteArray 对象中存储数据，并且数据文件通常是指定的文件格式。

例如，文件中的数据可能是文本文件格式，您可能希望在 String 对象中表示这种数据。

因此， FileStream 类包括读取和写入方法，用于读取和写入除 ByteArray 对象以外的类型的数据。例如，使用 readMultiByte() 方法可以从文件中读取数据，然后将它存储到字符串，如以下代码中所示：

```
var myFile:File = File.documentsDirectory.resolvePath("AIR Test/test.txt");
var myFileStream:FileStream = new FileStream();
myFileStream.addEventListener(Event.COMPLETE, completed);
myFileStream.openAsync(myFile, FileMode.READ);
var str:String = "";

function completeHandler(event:Event):void
{
    str = myFileStream.readMultiByte(myFileStream.bytesAvailable, "iso-8859-1");
}
```

readMultiByte() 方法的第二个参数指定 ActionScript 用来解释数据的文本格式（在示例中是“iso-8859-1”）。Adobe AIR 支持常见的字符集编码 （请参阅支持的字符集）。

FileStream 类还包括 readUTFBytes() 方法，该方法使用 UTF-8 字符集将读取缓冲区中的数据读入一个字符串中。由于 UTF-8 字符集中字符的长度是可变的，请不要在响应 progress 事件的方法中使用 readUTFBytes()，这是因为读取缓冲区结尾的数据可能表示不完整的字符。（将 readMultiByte() 方法用于可变长度字符编码时，需同样遵循上述要求。）因此，当 FileStream 对象调度 complete 事件时，应读取整个数据集。

还有类似的写入方法 writeMultiByte() 和 writeUTFBytes() 可用于 String 对象和文本文件。

readUTF() 和 writeUTF() 方法 （请不要与 readUTFBytes() 和 writeUTFBytes() 相混淆）也可以从文件读取文本数据和将文本数据写入文件，不过它们假定文本数据前面有指定文本数据长度的数据 （此方式在标准文本文件中不常见）。

有些 UTF 编码的文本文件以“UTF-BOM” （字节顺序标记）字符开头，该字符定义字节顺序以及编码格式 （如 UTF-16 或 UTF-32）。

有关读取和写入文本文件的示例，请参阅第 599 页的“示例：将 XML 文件读取到 XML 对象中”。

使用 readObject() 和 writeObject() 可以很方便地存储和检索复杂 ActionScript 对象的数据。数据采用 AMF (ActionScript Message Format) 编码。 Adobe AIR、 Flash Player、 Flash Media Server 和 Flex Data Services 包括用于此格式数据的 API。

还有一些其他读取和写入方法 （如 readDouble() 和 writeDouble()）。不过，如果您使用这些方法，请确保文件格式与这些方法定义的数据的格式相匹配。

文件格式通常比简单文本格式复杂。例如， MP3 文件包括压缩的数据，这些数据只能使用特定于 MP3 文件的解压缩和解码算法解释。 MP3 文件还可能包括 ID3 标签，这些标签包含有关文件的元标签信息 （如歌曲的标题和艺术家）。 ID3 格式有多种版本，不过最简单的版本 （ID3 第 1 版）在第 600 页的“示例：使用随机访问读取和写入数据”部分中进行了介绍。

其他文件格式 （用于图像、数据库、应用程序文档等）具有不同的结构，若要在 ActionScript 中使用这些格式的数据，必须了解数据的构造方式。

使用 load() 和 save() 方法

Flash Player 10 和更高版本， **Adobe AIR 1.5** 和更高版本

Flash Player 10 向 FileReference 类添加了 load() 和 save() 方法。AIR 1.5 中也有这些方法，并且 File 类从 FileReference 类继承这些方法。这些方法旨在为用户提供一种在 Flash Player 中加载和保存文件数据的安全方法。但是， AIR 应用程序还可以使用这些方法作为一种异步加载和保存文件的简便方式。

例如，以下代码将字符串保存到文本文件：

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
var str:String = "Hello.";
file.addEventListener(Event.COMPLETE, fileSaved);
file.save(str);
function fileSaved(event:Event):void
{
    trace("Done.");
}
```

save() 方法的 data 参数可以采用 String、XML 或 ByteArray 值。当参数为 String 或 XML 值时，该方法将文件保存为 UTF-8 编码的文本文件。

执行此代码示例时，应用程序将显示一个对话框，用户在该对话框中选择所保存文件的目标。

以下代码从 UTF-8 编码的文本文件加载字符串：

```
var file:File = File.applicationStorageDirectory.resolvePath("test.txt");
file.addEventListener(Event.COMPLETE, loaded);
file.load();
var str:String;
function loaded(event:Event):void
{
    var bytes:ByteArray = file.data;
    str = bytes.readUTFBytes(bytes.length);
    trace(str);
}
```

FileStream 类所提供的功能要多于 load() 和 save() 方法：

- 借助 FileStream 类，既可以同步读写数据，也可以异步读写数据。
- 使用 FileStream 类可以用增量方式写入文件。
- 使用 FileStream 类可以打开文件进行随机访问 （读写文件的任意部分）。
- 使用 FileStream 类可以指定对文件具有的访问权限的类型，具体途径是设置 open() 或 openAsync() 方法的 fileMode 参数。
- 通过 FileStream，不用向用户显示 “ 打开 ” 或 “ 保存 ” 对话框即可将数据保存到文件。
- 用 FileStream 类读取数据时可以直接使用字节数组之外的类型。

示例：将 XML 文件读取到 XML 对象中

Adobe AIR 1.0 和更高版本

以下示例演示如何读取和写入包含 XML 数据的文本文件。

若要从文件读取，请初始化 File 和 FileStream 对象，调用 FileStream 的 readUTFBytes() 方法，然后将字符串转换为 XML 对象：

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.open(file, FileMode.READ);
var prefsXML:XML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
fileStream.close();
```

同样，将数据写入文件也很容易，比如设置适当的 File 和 FileStream 对象，然后调用 FileStream 对象的写入方法。将 XML 数据的字符串版本传递到写入方法，如以下代码中所示：

```
var prefsXML:XML = <prefs><autoSave>true</autoSave></prefs>;
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
fileStream = new FileStream();
fileStream.open(file, FileMode.WRITE);

var outputString:String = '<?xml version="1.0" encoding="utf-8"?>\n';
outputString += prefsXML.toXMLString();

fileStream.writeUTFBytes(outputString);
fileStream.close();
```

这些示例使用 readUTFBytes() 和 writeUTFBytes() 方法，这是因为它们假定文件采用 UTF-8 格式。如果不是此格式，您可能需要使用其他方法 （请参阅第 597 页的 “ 数据格式以及选择要使用的读取和写入方法 ”）。

前面的示例使用为进行同步操作而打开的 FileStream 对象。您还可以打开文件进行异步操作 （这依赖于事件侦听器函数以响应事件）。例如，以下代码演示如何以异步方式读取 XML 文件：

```
var file:File = File.documentsDirectory.resolvePath("AIR Test/preferences.xml");
var fileStream:FileStream = new FileStream();
fileStream.addEventListener(Event.COMPLETE, processXMLData);
fileStream.openAsync(file, FileMode.READ);
var prefsXML:XML;

function processXMLData(event:Event):void
{
    prefsXML = XML(fileStream.readUTFBytes(fileStream.bytesAvailable));
    fileStream.close();
}
```

在将整个文件读入到读取缓冲区时 （当 FileStream 对象调度 complete 事件时），将调用 processXMLData() 方法。它调用 readUTFBytes() 方法以获取所读数据的字符串版本，然后它基于该字符串创建一个 XML 对象 prefsXML。

要查看演示这些功能的示例应用程序，请参阅从 XML 首选参数文件中读取和写入。

示例：使用随机访问读取和写入数据

Adobe AIR 1.0 和更高版本

MP3 文件可以包括 ID3 标签，这种标签是位于文件开头或结尾、包含用于标识录制情况的元数据的部分。 ID3 标签格式本身具有不同的修订版本。此例描述如何使用 “ 随机访问文件数据 ” 从包含最简单的 ID3 格式 （ID3 1.0 版）的 MP3 文件读取和写入，“ 随机访问文件数据 ” 表示它在文件中任意位置进行读取和写入。

包含 ID3 第 1 版标签的 MP3 文件在文件结尾最后 128 个字节中包括 ID3 数据。

当访问文件以进行随机读取 / 写入访问时，将 FileMode.UPDATE 指定为 open() 或 openAsync() 方法的 fileMode 参数很重要。

```
var file:File = File.documentsDirectory.resolvePath("My Music/Sample ID3 v1.mp3");
var fileStr:FileStream = new FileStream();
fileStr.open(file, FileMode.UPDATE);
```

这样可以读取和写入文件。

打开文件时，您可以设置 position 指针以指向文件结尾向前 128 个字节的位置：

```
fileStr.position = file.size - 128;
```

此代码将 position 属性设置为指向文件中的此位置，这是因为 ID3 1.0 版格式指定 ID3 标签数据存储在文件的最后 128 个字节中。该规范还包含以下内容：

- 标签的前 3 个字节包含字符串 "TAG"。
- 接下来的 30 个字符包含 MP3 曲目的标题，为字符串。
- 接下来的 30 个字符包含艺术家的姓名，为字符串。
- 接下来的 30 个字符包含唱片的名称，为字符串。
- 接下来的 4 个字符包含年份，为字符串。
- 接下来的 30 个字符包含注释，为字符串。
- 接下来的 1 个字节包含代码，指示曲目的流派。
- 所有文本数据都采用 ISO 8859-1 格式。

当读取数据后 （在调度 complete 事件时）， id3TagRead() 方法将检查数据：

```
 function id3TagRead():void
{
    if (fileStr.readMultiByte(3, "iso-8859-1").match(/tag/i))
    {
        var id3Title:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Artist:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Album:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3Year:String = fileStr.readMultiByte(4, "iso-8859-1");
        var id3Comment:String = fileStr.readMultiByte(30, "iso-8859-1");
        var id3GenreCode:String =  fileStr.readByte().toString(10);
    }
}
```

您还可以对文件执行随机访问写入。例如，您可以解析 id3Title 变量以确保它的大小写正确 （使用 String 类的方法），然后将修改后的名为 newTitle 的字符串写入文件，如下所示：

```
fileStr.position = file.length - 125;    // 128 - 3
fileStr.writeMultiByte(newTitle, "iso-8859-1");
```

为了遵守 ID3 第 1 版标准， newTitle 字符串的长度应为 30 个字符，结尾以字符代码 0 (String.fromCharCode(0)) 填充。

第 39 章：存储本地数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 SharedObject 类可以将少量数据存储到客户端计算机上。在 Adobe AIR 中，您还可以使用 EncryptedLocalStore 类在本地计算机上将少量隐私敏感性用户数据存储在 AIR 应用程序中。

您还可以在文件系统中读写文件，并 （在 Adobe AIR 中）访问本地数据库文件。有关详细信息，请参阅第 559 页的 " 使用文件系统 " 和第 613 页的 " 在 AIR 中使用本地 SQL 数据库 "。

有几个与共享对象相关的安全因素。有关详细信息，请参阅 第 894 页的 " 安全性 " 中的第 920 页的 " 共享对象 "。

共享对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

共享对象 （有时称为 "Flash cookie"）是一个数据文件，您访问的站点可能会在您的计算机上创建该文件。共享对象通常用于增强您的 Web 浏览体验 — 例如，使用这种对象可以个性化经常访问的网站的外观。

关于共享对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

共享对象与浏览器 Cookie 的功能类似。使用 SharedObject 可以将数据存储到用户的本地硬盘上，然后在同一会话期间或以后的会话中调用这些数据。各应用程序仅能访问它们自己的 SharedObject 数据，而且仅当它们在同一域中运行时才能访问。不会将这些数据发送到服务器，并且在其他域中运行的其他应用程序不能访问这些数据，但同一域中的应用程序可以访问这些数据。

共享对象与 Cookie 的比较

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Cookie 和共享对象非常相似。由于大多数 Web 程序员都熟悉 Cookie 的工作原理，因此将 Cookie 与本地共享对象进行比较可能非常有用。

遵循 RFC 2109 标准的 Cookie 通常有以下属性：

- 可以到期，并且默认情况下通常在会话结束时到期。
- 可以由客户端在特定站点上禁用。
- Cookie 总数最多为 300 个，每个站点最多只能有 20 个 Cookie。
- 每个 Cookie 的大小通常限制为 4 KB。
- 有时被认为是安全威胁，因此有时在客户端上被禁用。
- 存储在客户端浏览器指定的位置。
- 通过 HTTP 从客户端传输到服务器。

相比而言，共享对象有以下属性：

- 默认情况下不会到期。
- 默认情况下，每个共享对象的大小限制为 100 KB。
- 可以存储简单的数据类型 （例如字符串、数组和日期等）。
- 存储在应用程序指定的位置 （位于用户的主目录中）。
- 永远不会在客户端和服务器之间传输。

关于 SharedObject 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 SharedObject 类，可以创建和删除共享对象，并且可以检测正在使用的 SharedObject 对象的当前大小。

创建共享对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要创建 SharedObject 对象，请使用 SharedObject.getLocal() 方法，该方法的语法如下：

```
SharedObject.getLocal("objectName" [, pathname]): SharedObject
```

以下示例将创建一个名为 mySO 的共享对象：

```
public var mySO:SharedObject;
mySO = SharedObject.getLocal("preferences");
```

这样会在客户端计算机上创建一个名为 preferences.sol 的文件。

术语本地是指共享对象的位置。在本例中， Adobe® Flash® Player 将 SharedObject 文件本地存储到客户端的主目录中。

当您创建共享对象时， Flash Player 会为其沙盒中的应用程序和域创建一个新目录，还会创建一个用于存储 SharedObject 数据的 *.sol 文件。此文件的默认位置为用户主目录的子目录。下表显示了此目录的默认位置：

操作系统	位置
Windows 95/98/ME/2000/XP	c:/Documents and Settings/username/Application Data/Macromedia/Flash Player/#SharedObjects
Windows Vista/Windows 7	c:/Users/username/AppData/Roaming/Macromedia/Flash Player/#SharedObjects
Macintosh OS X	/Users/username/Library/Preferences/Macromedia/Flash Player/#SharedObjects/web_domain/path_to_application/application_name/object_name.sol
Linux/Unix	/home/username/.macromedia/Flash_Player/#SharedObjects/web_domain/path_to_application/application_name/object_name.sol

#SharedObjects 目录下是一个随机命名的目录。随机命名目录下是一个与主机名匹配的目录，再下一级是与应用程序路径匹配的目录，最后一级是 *.sol 文件。

例如，如果您在本地主机上的 /sos 子目录中请求名为 MyApp.swf 的应用程序，则 Flash Player 会将 *.sol 文件存储在 Windows XP 的以下位置：

```
c:/Documents and Settings/fred/Application Data/Macromedia/Flash
Player/#SharedObjects/KROKWXRK/#localhost/sos/MyApp.swf/data.sol
```

注：如果未在 SharedObject.getLocal() 方法中提供名称， Flash Player 会将该文件命名为 undefined.sol。

默认情况下，在使用 Flash 时，每个域可在本地保存的持久性 SharedObject 对象的大小不能超过 100 KB。用户可以配置此值。如果应用程序尝试将数据保存到共享对象中，但假如保存成功的话共享对象的大小将超过 100 KB，则 Flash Player 将显示 " 本地存储区 " 对话框，用户可在该对话框中允许或拒绝为请求访问的域增加本地存储区。

指定路径

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用可选的 pathname 参数指定 SharedObject 文件的位置。此文件必须是该域的 SharedObject 目录的子目录。例如，如果请求本地主机上的应用程序并指定以下内容：

```
mySO = SharedObject.getLocal("myObjectFile","/");
```

Flash Player 会将 SharedObject 文件写入 /#localhost 目录 （如果应用程序处于脱机状态，则写入 /localhost 目录）。如果希望客户端上的多个应用程序能够访问同一共享对象，这将非常有用。这种情况下，客户端可以运行两个 Flex 应用程序，这两个应用程序都指定了作为域的根目录的共享对象路径；此后，客户端可以从这两个应用程序访问同一共享对象。要在多个非持久性应用程序之间共享数据，可以使用 LocalConnection 对象。

如果指定的目录不存在， Flash Player 将不会创建 SharedObject 文件。

将数据添加到共享对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 SharedObject 对象的 data 属性将数据添加至 SharedObject 的 * .sol 文件中。要将新数据添加到共享对象，请使用下面的语法：

```
sharedObject_name.data.variable = value;
```

以下示例将 userName、 itemNumbers 和 adminPrivileges 属性及其值添加到 SharedObject 中：

```
public var currentUserName:String = "Reiner";
public var itemsArray:Array = new Array(101,346,483);
public var currentUserIsAdmin:Boolean = true;
mySO.data.userName = currentUserName;
mySO.data.itemNumbers = itemsArray;
mySO.data.adminPrivileges = currentUserIsAdmin;
```

为 data 属性赋值后，必须强制 Flash Player 将这些值写入 SharedObject 文件。要强制 Flash Player 将这些值写入 SharedObject 文件，请使用 SharedObject.flush() 方法，如下所示：

```
mySO.flush();
```

如果未调用 SharedObject.flush() 方法， Flash Player 会在应用程序退出时将值写入该文件。但是，如果该数据超出默认设置，用户将失去增加 Flash Player 可用空间来存储该数据的机会。因此，较好的做法是调用 SharedObject.flush()。

使用 flush() 方法将共享对象写入用户的硬盘驱动器时，应仔细检查用户是否已使用 Flash Player 设置管理器 (www.macromedia.com/support/documentation/cn/flashplayer/help/settings_manager07.html) 明确禁用了本地存储，如下面的示例所示：

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

在共享对象中存储对象

可以将简单对象 （如数组或字符串）存储到 SharedObject 的 data 属性中。

在以下示例中，一个 ActionScript 类定义了一些控制与共享对象进行交互的方法。用户可以使用这些方法将对象添加到共享对象中或从共享对象中删除对象。此类将存储一个包含简单对象的 ArrayCollection。

```
package {
    import mx.collections.ArrayCollection;
    import flash.net.SharedObject;

    public class LSOHandler {

        private var mySO:SharedObject;
        private var ac:ArrayCollection;
        private var lsoType:String;

        // The parameter is "feeds" or "sites".
        public function LSOHandler(s:String) {
            init(s);
        }

        private function init(s:String):void {
            ac = new ArrayCollection();
            lsoType = s;
            mySO = SharedObject.getLocal(lsoType);
            if (getObjects()) {
                ac = getObjects();
            }
        }

        public function getObjects():ArrayCollection {
            return mySO.data[lsoType];
        }

        public function addObject(o:Object):void {
            ac.addItem(o);
            updateSharedObjects();
        }

        private function updateSharedObjects():void {
            mySO.data[lsoType] = ac;
            mySO.flush();
        }
    }

}
```

下面的 Flex 应用程序为所需的每个共享对象类型创建一个 ActionScript 类的实例，然后在用户添加或删除博客或站点 URL 时调用该类的方法。

```
<?xml version="1.0"?>
<!-- lsos/BlogAggregator.mxml -->
<mx:Application
    xmlns:local="*"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    creationComplete="initApp()"
    backgroundColor="#ffffff"
>
    <mx:Script>
        <![CDATA[
        import mx.collections.ArrayCollection;
        import mx.utils.ObjectUtil;
        import flash.net.SharedObject;

        [Bindable]
        public var welcomeMessage:String;

        [Bindable]
        public var localFeeds:ArrayCollection = new ArrayCollection();

        [Bindable]
        public var localSites:ArrayCollection = new ArrayCollection();

        public var lsofeeds:LSOHandler;
        public var lsosites:LSOHandler;

        private function initApp():void {
            lsofeeds = new LSOHandler("feeds");
            lsosites = new LSOHandler("sites");

            if (lsofeeds.getObjects()) {
                localFeeds = lsofeeds.getObjects();
            }
            if (lsosites.getObjects()) {
                localSites = lsosites.getObjects();
            }
        }

        // Adds a new feed to the feeds DataGrid.
        private function addFeed():void {
            // Construct an object you want to store in the
            // LSO. This object can contain any number of fields.
            var o:Object = {name:ti1.text, url:ti2.text, date:new Date()};
            lsofeeds.addObject(o);

            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method.
            localFeeds = lsofeeds.getObjects();

            // Clear the text fields.
            ti1.text = '';
            ti2.text = '';
        }

        // Removes feeds from the feeds DataGrid.
        private function removeFeed():void {
            // Use a method of ArrayCollection to remove a feed.
            // Because the DataGrid's dataProvider property is
            // bound to the ArrayCollection, Flex updates the
            // DataGrid when you call this method. You do not need
            // to update it manually.
```

```
            if (myFeedsGrid.selectedIndex > -1) {

localFeeds.removeItemAt(myFeedsGrid.selectedIndex);
             }
        }

        private function addSite():void {
            var o:Object = {name:ti3.text, date:new Date()};
            lsosites.addObject(o);
            localSites = lsosites.getObjects();
            ti3.text = '';
        }

        private function removeSite():void {
            if (mySitesGrid.selectedIndex > -1) {

localSites.removeItemAt(mySitesGrid.selectedIndex);
            }
        }

        ]]>
    </mx:Script>

    <mx:Label text="Blog aggregator" fontSize="28"/>

    <mx:Panel title="Blogs">
        <mx:Form id="blogForm">
            <mx:HBox>
                <mx:FormItem label="Name:">
                    <mx:TextInput id="ti1" width="100"/>
                </mx:FormItem>
                <mx:FormItem label="Location:">
                    <mx:TextInput id="ti2" width="300"/>
                </mx:FormItem>
                <mx:Button id="b1" label="Add Feed" click="addFeed()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Feeds:">
                <mx:DataGrid
                    id="myFeedsGrid"
                    dataProvider="{localFeeds}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b2" label="Remove Feed" click="removeFeed()"/>
        </mx:Form>
    </mx:Panel>

    <mx:Panel title="Sites">
```

```
        <mx:Form id="siteForm">
            <mx:HBox>
                <mx:FormItem label="Site:">
                    <mx:TextInput id="ti3" width="400"/>
                </mx:FormItem>
                <mx:Button id="b3" label="Add Site" click="addSite()"/>
            </mx:HBox>

            <mx:FormItem label="Existing Sites:">
                <mx:DataGrid
                    id="mySitesGrid"
                    dataProvider="{localSites}"
                    width="400"
                />
            </mx:FormItem>
            <mx:Button id="b4" label="Remove Site" click="removeSite()"/>
        </mx:Form>
    </mx:Panel>

</mx:Application>
```

在共享对象中存储指定了类型的对象

可以在共享对象中存储指定了类型的 ActionScript 实例。通过调用 flash.net.registerClassAlias() 方法来注册类可以实现此目的。如果创建了类的实例并将该实例存储在共享对象的数据成员中，而以后要读出该对象，则将得到指定了类型的实例。默认情况下， SharedObject objectEncoding 属性支持 AMF3 编码，并从 SharedObject 对象中解包存储的实例；存储的实例将保持您调用 registerClassAlias() 方法时指定的同一类型。

（仅限 iOS）阻止本地共享目标的云备份

Adobe AIR 3.7 和更高版本，仅针对 **iOS**

您可以设置 SharedObject.preventBackup 属性，控制是否将在 iOS 云备份服务上备份本地共享对象。这是 Apple 对于可以重新生成或重新下载内容的必需要求，但不是您的应用程序在脱机使用期间正常工作的必需要求。

创建多个共享对象

可以为同一 Flex 应用程序创建多个共享对象。为此，需要为每个共享对象指定不同的实例名，如以下示例所示：

```
public var mySO:SharedObject = SharedObject.getLocal("preferences");
public var mySO2:SharedObject = SharedObject.getLocal("history");
```

这样可以在 Flex 应用程序的本地目录中创建一个 preferences.sol 文件和一个 history.sol 文件。

创建安全 SharedObject

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当使用 getLocal() 或 getRemote() 创建本地或远程 SharedObject 时，有一个名为 secure 的可选参数，该参数确定对此共享对象的访问是否限于通过 HTTPS 连接传递的 SWF 文件。如果此参数设置为 true 且 SWF 文件是通过 HTTPS 传递的， Flash Player 将新建一个安全共享对象，或者获取对现有安全共享对象的引用。只可由通过 HTTPS 传递的 SWF 文件对此安全共享对象进行读取或写入，而该 HTTPS 调用 SharedObject.getLocal() 并将 secure 参数设置为 true。如果此参数设置为 false 且 SWF 文件是通过 HTTPS 传递的， Flash Player 将新建一个共享对象，或者获取对现有共享对象的引用。

仅可由通过非 HTTPS 连接传递的 SWF 文件对此共享对象进行读取或写入。如果 SWF 文件是通过非 HTTPS 连接传递的，并且您尝试将此参数设置为 true，将无法创建新的共享对象 （或访问以前创建的安全共享对象），并会引发错误，并且共享对象设置为 null。如果尝试通过非 HTTPS 连接运行以下代码片断， SharedObject.getLocal() 方法将引发错误：

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

无论此参数为何值，创建的共享对象的数量都接近域所允许的磁盘空间的总量。

显示共享对象的内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

值存储在共享对象中的 data 属性中。可以使用 for..in 循环来循环访问共享对象中的每个值，如下面的示例所示：

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
    trace(i + ":\t" + so.data[i]);
}
```

销毁共享对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要破坏客户端上的 SharedObject，请使用 SharedObject.clear() 方法。这样做不会销毁应用程序共享对象的默认路径中的目录。

以下示例将从客户端中删除 SharedObject 文件：

```
public function destroySharedObject():void {
    mySO.clear();
}
```

SharedObject 示例

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下示例显示了可在 SharedObject 对象中存储简单对象 （如 Date 对象），而不必对这些对象手动进行序列化和反序列化。

以下示例从欢迎您第一次访问开始。在您单击 “ 注销 ” 后，应用程序会将当前日期存储在共享对象中。下次您启动此应用程序或者刷新该页时，应用程序将欢迎您回来，并显示一则提醒，指出您上次注销的时间。

要查看应用程序的运行情况，请启动该应用程序，单击 “ 注销 ”，然后刷新该页。应用程序将显示您上次访问时单击 “ 注销 ” 按钮的日期和时间。您随时都可以通过单击 “ 删除 LSO” 按钮来删除存储的信息。

```
<?xml version="1.0"?>
<!-- lsos/WelcomeMessage.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="initApp()">
  <mx:Script><![CDATA[
  public var mySO:SharedObject;
  [Bindable]
  public var welcomeMessage:String;

  public function initApp():void {
     mySO = SharedObject.getLocal("mydata");
     if (mySO.data.visitDate==null) {
        welcomeMessage = "Hello first-timer!"
     } else {
        welcomeMessage = "Welcome back. You last visited on " +
           getVisitDate();
     }
  }

  private function getVisitDate():Date {
     return mySO.data.visitDate;
  }

  private function storeDate():void {
     mySO.data.visitDate = new Date();
     mySO.flush();
  }

  private function deleteLSO():void {
     // Deletes the SharedObject from the client machine.
     // Next time they log in, they will be a 'first-timer'.
     mySO.clear();
  }

  ]]></mx:Script>
  <mx:Label id="label1" text="{welcomeMessage}"/>
  <mx:Button label="Log Out" click="storeDate()"/>
  <mx:Button label="Delete LSO" click="deleteLSO()"/>
</mx:Application>
```

加密的本地存储区

EncryptedLocalStore 类 (ELS) 提供的加密本地存储机制可用作存储应用程序隐私数据的小缓存。 ELS 数据不能在应用程序之间实现共享。使用 ELS 的目的在于，使应用程序可轻松存储重新创建的项目，如登录凭据及其他隐私信息。如下面 “ 加密本地存储区的限制 ” 和 “ 最佳做法 ” 中列出的内容， ELS 数据不应被视为永久数据。

注：除了加密本地存储区之外， AIR 还可以对 SQL 数据库中存储的内容进行加密。有关详细信息，请参阅第 649 页的 “ 对 SQL 数据库使用加密 ”。

您可能想使用加密本地存储区来缓存必须保护的信息，如用于获取 Web 服务的登录凭据。 ELS 适合存储不得向其他用户公开的信息。但是，使用同一用户帐户运行的其他进程仍可访问它存储的数据。因此，它不适用于合保护秘密应用程序数据，例如 DRM 或加密密钥。

在桌面平台上，通过在 Windows 中使用 DPAPI，在 Mac OS 和 iOS 中使用 KeyChain，以及在 Linux 中使用 KeyRing 或 KWallet， AIR 将加密本地存储区与每个应用程序和用户相关联。加密的本地存储区使用 AES-CBC 128 位加密。

在 Android 上， EncryptedLocalStorage 类存储的数据未加密。而该数据由操作系统提供的用户级别的安全性进行保护。Android 操作系统为每个应用程序分配一个单独的用户 ID。应用程序只能访问自己的文件和在公共位置创建的文件 （如移动存储卡）。注意，在 Android 的 " 根 " 设备上，使用根权限运行的应用程序可以访问其他应用程序的文件。因此，在根设备上，加密的本地存储不提供与非根设备上级别一样高的数据保护。

加密的本地存储区中的信息仅可用于应用程序安全沙箱中的 AIR 应用程序内容。

如果更新 AIR 应用程序，则更新后的版本仍能够访问加密本地存储区中的任何现有数据，以下情况除外：

- 使用 stronglyBound 参数添加的项目设置为 true
- 现有和更新版本发布的时间都早于 AIR 1.5.3，并且更新使用迁移签名进行签名。

加密本地存储区的限制

加密本地存储区中的数据由用户操作系统帐户凭据进行保护。除非可以用该用户的身份进行登录，否则其他实体无法访问存储区中的数据。但是，已通过身份验证的用户运行的其他应用程序仍可访问这些数据。

由于用户必须经过身份验证才能使这些攻击生效，所以用户的隐私数据仍然受到保护 （除非用户的帐户本身已被泄漏）。但是，应用程序希望对用户保密的数据 （如用于授权或数字版权管理的密钥）是不安全的。因此， ELS 不是存储此类信息的适当位置。它只适合存储用户的隐私数据，如密码。

ELS 中的数据可能由于各种原因而丢失。例如，用户可能会卸载应用程序并删除加密的文件。或者，发行商 ID 可能由于更新而发生更改。因此，应将 ELS 用作私有缓存，而不是永久性数据存储。

stronglyBound 参数已弃用，不应将其设置为 true。此参数设置为 true 后，不会对数据进行任何额外保护。同时，即使发行商 ID 保持不变，应用程序每次更新后都会丢失对数据的访问。

如果存储的数据超过 10MB，则加密的本地存储区的运行速度可能变慢。

当卸载 AIR 应用程序时，卸载程序不会删除存储在加密的本地存储区中的数据。

最佳做法

使用 ELS 的最佳做法包括：

- 使用 ELS 存储例如密码等敏感用户数据 （将 stronglyBound 设置为 false）
- 不使用 ELS 存储应用程序机密 （如 DRM 密钥或授权令牌）。
- 为应用程序提供在 ELS 数据丢失的情况下重新创建 ELS 中存储的数据的方法。例如，在必要时，通过提示用户重新输入帐户凭据来实现此操作。
- 不要使用 stronglyBound 参数。
- 如果 stronglyBound 确实设置为 true，则在更新期间不要迁移存储的项目。而应在更新后重新创建数据。
- 仅存储较少数量的数据。对于大量数据，请使用加密的 AIR SQL 数据库。

更多帮助主题

flash.data.EncryptedLocalStore

将数据添加到加密本地存储区

使用 EncryptedLocalStore 类的 setItem() 静态方法将数据存储在本地存储区中。数据存储在哈希表中 （使用字符串作为键，以字节数组的形式存储数据）。

例如，下面的代码将一个字符串存储在加密的本地存储区中：

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes);
```

setItem() 方法的第三个参数 （即 stronglyBound 参数）是可选参数。如果此参数设置为 true，则加密本地存储区会将存储的项目绑定到存储 AIR 应用程序的数字签名和位：

```
var str:String = "Bob";
var bytes:ByteArray = new ByteArray();
bytes.writeUTFBytes(str);
EncryptedLocalStore.setItem("firstName", bytes, false);
```

对于将 stronglyBound 设置为 true 后存储的项目，在以后调用 getItem() 时，仅当调用方 AIR 应用程序与存储方应用程序相同时才会成功 （前提是应用程序目录中的文件未发生数据更改）。如果执行调用的 AIR 应用程序与执行存储的应用程序不同，则当您对强绑定项目调用 getItem() 时，该应用程序将引发 Error 异常。如果您更新应用程序，则该应用程序将无法读取先前写入到加密的本地存储区中的强绑定数据。忽略在移动设备上将 stronglyBound 设置为 true ；始终将该参数视为 false。

如果将 stronglyBound 参数设置为 false （默认值），则只有发行商 ID 需要保持不变，供应用程序读取数据。应用程序的位可以更改 （需由同一发行商对这些位进行签名），但不需要与存储数据的应用程序中的位完全相同。如果更新后的应用程序与原始应用程序的发行商 ID 相同，则可继续访问这些数据。

注: 实际上，将 stronglyBound 设置为 true 不会增加任何额外数据保护。“ 恶意 ” 用户仍可更改应用程序，从而访问存储在 ELS 中的项目。而且，无论将 stronglyBound 设置为 true 还是 false，保护数据免受外部非用户威胁的强度都是一样的。出于以上原因，建议不要将 stronglyBound 设置为 true。

访问加密的本地存储区中的数据

Adobe AIR 1.0 和更高版本

您可以使用 EncryptedLocalStore.getItem() 方法从加密的本地存储区中检索值，如下例所示：

```
var storedValue:ByteArray = EncryptedLocalStore.getItem("firstName");
trace(storedValue.readUTFBytes(storedValue.length)); // "Bob"
```

从加密的本地存储区中删除数据

Adobe AIR 1.0 和更高版本

您可以使用 EncryptedLocalStore.removeItem() 方法删除加密的本地存储区中的值，如下例所示：

```
EncryptedLocalStore.removeItem("firstName");
```

您可以通过调用 EncryptedLocalStore.reset() 方法清除加密的本地存储区中的所有数据，如下例所示：

```
EncryptedLocalStore.reset();
```

第 40 章：在 AIR 中使用本地 SQL 数据库

Adobe AIR 1.0 和更高版本

Adobe® AIR® 包括创建和使用本地 SQL 数据库的功能。运行时包括一个 SQL 数据库引擎，该引擎使用开放源代码 SQLite 数据库系统，支持许多标准 SQL 功能。本地 SQL 数据库可用于存储本地永久性数据。例如，它可用于应用程序数据、应用程序用户设置、文档或希望应用程序在本地保存的任何其他类型的数据。

关于本地 SQL 数据库

Adobe AIR 1.0 和更高版本

有关使用 SQL 数据库的快速介绍和代码示例，请参阅 Adobe Developer Connection 中的以下快速入门文章：

- 异步处理本地 SQL 数据库 (Flex)
- 同步处理本地 SQL 数据库 (Flex)
- 使用加密数据库 (Flex)
- 异步处理 SQL 数据库 (Flash)
- 同步处理本地 SQL 数据库 (Flash)
- 使用加密数据库 (Flash)

Adobe AIR 包括一个基于 SQL 的关系数据库引擎，该引擎在运行时中运行，数据以本地方式存储在运行 AIR 应用程序的计算机上的数据库文件中 （例如，在计算机的硬盘驱动器上）。由于数据库的运行和数据文件的存储都在本地进行，因此，不管网络连接是否可用， AIR 应用程序都可以使用数据库。这样，运行时的本地 SQL 数据库引擎为存储永久的本地应用程序数据提供了一种便利机制，特别是您具有 SQL 和关系数据库经验时。

本地 SQL 数据库的用途

Adobe AIR 1.0 和更高版本

AIR 本地 SQL 数据库功能可以用于将应用程序数据存储在用户的本地计算机上的任何目的。Adobe AIR 包括在本地存储数据的几种机制，各机制具有不同的优点。以下是本地 SQL 数据库在 AIR 应用程序中的一些可能用途：

- 对于面向数据的应用程序 （例如通讯簿），数据库可以用于存储主应用程序数据。
- 对于面向文档的应用程序 （用户创建要保存并可能共享的文档），可以在用户指定的位置将每个文档另存为数据库文件。（不过请注意，除非加密了数据库，否则任何 AIR 应用程序都可以打开该数据库文件。对于可能存在敏感信息的文档，建议使用加密。）
- 对于支持网络的应用程序，数据库可以用于存储应用程序数据的本地缓存，或者在网络连接不可用时暂时存储数据。可以创建一种将本地数据库与网络数据存储同步的机制。
- 对于任何应用程序，数据库都可以用于存储单个用户的应用程序设置，例如用户选项或应用程序信息 （如窗口大小和位置）。

更多帮助主题

Christophe Coenraets：AIR for Android 上的 Employee 目录

Raymond Camden：jQuery 和 AIR – 从网页到应用程序

关于 AIR 数据库和数据库文件

Adobe AIR 1.0 和更高版本

单个 Adobe AIR 本地 SQL 数据库作为单个文件存储在计算机的文件系统中。运行时包括 SQL 数据库引擎，该引擎管理数据库文件的创建和结构化以及操作和检索数据库文件中的数据。运行时不指定在文件系统上存储数据库数据的方式或位置；相反，每个数据库完全存储在单个文件中。您指定在文件系统中存储数据库文件的位置。单个 AIR 应用程序可以访问一个或多个单独的数据库 （即单独的数据库文件）。由于运行时将每个数据库作为单个文件存储在文件系统上，因此可以在需要时按照应用程序的设计和操作系统的文件访问约束查找您的数据库。每个用户都可以具有其特定数据的单独数据库文件，或者数据库文件可以由在单个计算机上共享数据的所有应用程序用户访问。由于数据对单个计算机是本地的，因此在不同计算机上的用户之间并不自动共享数据。本地 SQL 数据库引擎未提供对远程数据库或基于服务器的数据库执行 SQL 语句的任何功能。

关于关系数据库

Adobe AIR 1.0 和更高版本

关系数据库是一种在计算机上存储 （和检索）数据的机制。数据被组织到表中：行表示记录或项目，而列 （有时称为 “ 字段 ”）将每个记录分到各个值中。例如，通讯簿应用程序可能包含 “ 朋友 ” 表。表中的每个行都表示存储在数据库中的单个朋友。表的列表示名字、姓氏、出生日期等数据。对于表中的每个朋友行，数据库为每个列存储一个单独的值。

关系数据库设计用于存储复杂数据，其中一个项目与其他类型的项目关联或相关。在关系数据库中，应该将具有一对多关系 （其中单个记录可以与不同类型的多个记录相关）的任何数据分到不同的表中。例如，假定您希望通讯簿应用程序为每个朋友存储多个电话号码；这就是一对多关系。 “ 朋友 ” 表包含每个朋友的所有个人信息。单独的 “ 电话号码 ” 表包含所有朋友的所有电话号码。

除了存储有关朋友和电话号码的数据外，每个表都需要一段数据来跟踪这两个表之间的关系，以便使单个朋友记录与其电话号码匹配。该数据称为主键 — 将一个表中的每个行与该表中的其他行区分开的唯一标识符。主键可以是 “ 自然键 ”，这意味着它是自然区分表中每个记录的数据项目之一。在 “ 朋友 ” 表中，如果您知道朋友的出生日期都是不同的，则可以将出生日期列用作 “ 朋友 ” 表的主键 （自然键）。如果没有自然键，则应单独创建一个主键列，如 “ 朋友 id” （应用程序用于区分各行的人工值）。

使用主键，可以设置多个表之间的关系。例如，假定 “ 朋友 ” 表有一个 “ 朋友 id” 列，其中包含每行 （每个朋友）的唯一编号。可以用以下两列来构建相关的 “ 电话号码 ” 表：一个列包含电话号码所属的朋友的 “ 朋友 id”，另一列包含实际的电话号码。这样，不管单个朋友具有多少个电话号码，都可以将它们全部存储在 “ 电话号码 ” 表中，并可以使用 “ 朋友 id” 主键将其链接到相关的朋友。在相关表中使用一个表的主键指定记录之间的联系时，相关表中的值称为外键。与许多数据库不同， AIR 本地数据库引擎不允许您创建外键约束 （即自动检查插入的或更新的外键值在主键表中是否具有对应行的约束）。然而，外键关系是关系数据库结构的重要部分，而且在数据库的表之间创建关系时应该使用外键。

关于 SQL

Adobe AIR 1.0 和更高版本

结构化查询语言 (SQL) 用于关系数据库以操作和检索数据。 SQL 是一种描述性语言，而不是一种过程语言。 SQL 语句描述您所需的一组数据，而不是提供有关它应该如何检索数据的计算机指令。数据库引擎确定如何检索该数据。

SQL 语言已由美国国家标准协会 (ANSI) 进行了标准化。 Adobe AIR 本地 SQL 数据库支持 SQL-92 标准的大部分内容。

有关 Adobe AIR 中支持的 SQL 语言的特定说明，请参阅第 946 页的 “ 本地数据库中的 SQL 支持 ”。

关于 SQL 数据库类

Adobe AIR 1.0 和更高版本

要在 ActionScript 3.0 中使用本地 SQL 数据库，请使用 flash.data 包中这些类的实例：

类	说明
flash.data.SQLConnection	提供创建和打开数据库 （数据库文件）的方式，以及执行数据库级操作和控制数据库事务的方法。
flash.data.SQLStatement	表示对数据库执行的单个 SQL 语句 （单个查询或命令），包括定义语句文本和设置参数值。
flash.data.SQLResult	提供一种通过执行语句获取信息或结果的方法，如执行 SELECT 语句后的结果行、受 UPDATE 或 DELETE 语句影响的行数等。

若要获取描述数据库结构的架构信息，请使用 flash.data 包中的以下类：

类	说明
flash.data.SQLSchemaResult	充当通过调用 SQLConnection.loadSchema() 方法生成的数据库架构结果的容器。
flash.data.SQLTableSchema	提供描述数据库中单个表的信息。
flash.data.SQLViewSchema	提供描述数据库中单个视图的信息。
flash.data.SQLIndexSchema	提供描述数据库中表或视图的单个列的信息。
flash.data.SQLTriggerSchema	提供描述数据库中单个触发器的信息。

flash.data 包中的其他类提供用于 SQLConnection 类和 SQLColumnSchema 类的常数：

类	说明
flash.data.SQLMode	定义一组常量，它们表示 SQLConnection.open() 和 SQLConnection.openAsync() 方法的 openMode 参数的可能值。
flash.data.SQLColumnNameStyle	定义一组常量，它们表示 SQLConnection.columnNameStyle 属性的可能值。
flash.data.SQLTransactionLockType	定义一组常量，它们表示 SQLConnection.begin() 方法的 option 参数的可能值。
flash.data.SQLCollationType	定义一组常数，它们表示 SQLColumnSchema() 构造函数的 SQLColumnSchema.defaultCollationType 属性和 defaultCollationType 参数的可能值。

此外， flash.events 包中的以下类表示所用的事件 （和支持常数）：

类	说明
flash.events.SQLEvent	定义其任何操作成功执行时 SQLConnection 或 SQLStatement 实例调度的事件。每个操作都具有一个在 SQLEvent 类中定义的关联事件类型常数。
flash.events.SQLErrorEvent	定义 SQLConnection 或 SQLStatement 实例在其任何操作导致错误时调度的事件。
flash.events.SQLUpdateEvent	定义因执行 INSERT、 UPDATE 或 DELETE SQL 语句而导致其连接数据库之一中的表数据更改时 SQLConnection 实例调度的事件。

最后， flash.errors 包中的以下类提供有关数据库操作错误的信息：

类	说明
flash.errors.SQLError	提供有关数据库操作错误的信息，包括尝试的操作和出错原因。
flash.errors.SQLErrorOperation	定义一组常数，它们表示 SQLError 类的 operation 属性 （它指示导致错误的数据库操作）的可能值。

关于同步和异步执行模式

Adobe AIR 1.0 和更高版本

编写代码以处理本地 SQL 数据库时，会指定以两种执行模式之一执行数据库操作：异步或同步执行模式。通常，代码示例说明如何以这两种方式执行每个操作，以便您可以使用最适合您需求的示例。

在异步执行模式中，为运行时提供一个指令，运行时将在请求的操作完成或失败时调度事件。首先，通知数据库引擎执行操作。在应用程序继续运行的同时，数据库引擎在后台工作。最后，完成操作时 （或者它失败时），数据库引擎调度事件。由事件触发的代码执行后续操作。此方法具有一个重要的优点：运行时在后台执行数据库操作，同时主应用程序代码继续执行。如果数据库操作花费大量的时间，则应用程序继续运行。最重要的是，用户可以继续与其交互，而屏幕不会冻结。但是，与其他代码相比，编写异步操作代码可能更加复杂。在必须将多个相关的操作分配给各个事件侦听器方法的情况下，通常会出现此复杂性。

从概念上说，将操作作为单个步骤序列 （一组同步操作，而不是分到几个事件侦听器方法中的一组操作）进行编码更为简单。除了异步数据库操作外， Adobe AIR 还允许您同步执行数据库操作。在同步执行模式中，操作不在后台运行。相反，它们以与所有其他应用程序代码相同的执行序列运行。通知数据库引擎执行操作。然后，代码在数据库引擎工作时暂停。完成操作后，继续执行下一行代码。

异步还是同步执行操作是在 SQLConnection 级别上设置的。使用单个数据库连接，无法同步执行某些操作或语句，同时异步执行其他操作或语句。通过调用 SQLConnection 方法打开数据库，可以指定 SQLConnection 是在同步还是异步执行模式下操作。如果调用 SQLConnection.open()，则连接在同步执行模式下操作；如果调用 SQLConnection.openAsync()，则连接在异步执行模式下操作。使用 open() 或 openAsync() 将 SQLConnection 实例连接到数据库后，除非先关闭再重新打开到数据库的连接，否则该实例将固定为同步或异步执行模式。

每种执行模式各有其优点。虽然每种模式的大多数方面是类似的，但是在每种模式下工作时要牢记一些差异。有关这些主题的详细信息以及在每种模式下工作的建议，请参阅第 645 页的 “ 使用同步和异步数据库操作 ”。

创建和修改数据库

Adobe AIR 1.0 和更高版本

数据库中必须定义了应用程序可以访问的表，应用程序才可以添加或检索数据。下面说明了创建数据库和在数据库中创建数据结构的任务。虽然这些任务的使用频率低于数据插入和数据检索，但大多数应用程序都必须使用这些任务。

更多帮助主题

使用 Flex：更新现有 AIR 数据库

创建数据库

Adobe AIR 1.0 和更高版本

要创建数据库文件，需要首先创建 SQLConnection 实例。调用其 open() 方法在同步执行模式下打开它，或者调用其 openAsync() 方法在异步执行模式下打开它。 open() 和 openAsync() 方法用于打开到数据库的连接。如果传递的 File 实例引用 reference 参数 （第一个参数）的不存在的文件位置，则 open() 或 openAsync() 方法将在该文件位置创建一个数据库文件，并打开到新创建的数据库的连接。

无论创建数据库时调用的是 open() 方法还是 openAsync() 方法，数据库文件的名称都可以是采用任何文件扩展名的任何有效文件名。如果调用 reference 参数为 null 的 open() 或 openAsync() 方法，则将创建新的内存中数据库，而不是在磁盘上创建数据库文件。

以下代码清单说明使用异步执行模式创建数据库文件 （新数据库）的过程。在本例中，数据库文件保存在第 576 页的 “ 指向应用程序存储目录 ” 中，文件名为 “DBSample.db”:

```
import flash.data.SQLConnection;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile);

function openHandler(event:SQLEvent):void
{
    trace("the database was created successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database was created successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

注：尽管 File 类用于指向特定本机文件路径，但这会导致应用程序无法跨平台工作。例如，路径 C:\Documents and Settings\joe\test.db 仅适用于 Windows。出于以上原因，最好使用 File 类的静态属性（如 File.applicationStorageDirectory）和 resolvePath() 方法（如上一示例所示）。有关详细信息，请参阅第 573 页的 “File 对象的路径 ”。

要同步执行操作，请在使用 SQLConnection 实例打开数据库连接时，调用 open() 方法。以下示例说明如何创建和打开同步执行其操作的 SQLConnection 实例：

```
import flash.data.SQLConnection;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile);
                    trace("the database was created successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

创建数据库表

Adobe AIR 1.0 和更高版本

在数据库中创建表包括使用与执行 SELECT、 INSERT 等 SQL 语句相同的过程对该数据库执行 SQL 语句。要创建表，请使用 CREATE TABLE 语句，该语句包括新表的列和约束的定义。有关执行 SQL 语句的详细信息，请参阅第 625 页的 “ 使用 SQL 语句 ”。

以下示例演示如何使用异步执行模式在现有数据库文件中创建一个名为 “employees” 的表。请注意，此代码假定存在一个名为 conn 的 SQLConnection 实例，并且该实例已经实例化并连接到数据库。

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

createStmt.addEventListener(SQLEvent.RESULT, createResult);
createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

createStmt.execute();

function createResult(event:SQLEvent):void
{
    trace("Table created");
}

function createError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;
```

```
                createStmt.addEventListener(SQLEvent.RESULT, createResult);
                createStmt.addEventListener(SQLErrorEvent.ERROR, createError);

                createStmt.execute();
            }

            private function createResult(event:SQLEvent):void
            {
                trace("Table created");
            }

            private function createError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

以下示例演示如何使用同步执行模式在现有数据库文件中创建一个名为 “employees” 的表。请注意，此代码假定存在一个名为 conn 的 SQLConnection 实例，并且该实例已经实例化并连接到数据库。

```
import flash.data.SQLConnection;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

var createStmt:SQLStatement = new SQLStatement();
createStmt.sqlConnection = conn;

var sql:String =
    "CREATE TABLE IF NOT EXISTS employees (" +
    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "   firstName TEXT, " +
    "   lastName TEXT, " +
    "   salary NUMERIC CHECK (salary > 0)" +
    ")";
createStmt.text = sql;

try
{
    createStmt.execute();
    trace("Table created");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                var createStmt:SQLStatement = new SQLStatement();
                createStmt.sqlConnection = conn;

                var sql:String =
                    "CREATE TABLE IF NOT EXISTS employees (" +
                    "   empId INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "   firstName TEXT, " +
                    "   lastName TEXT, " +
                    "   salary NUMERIC CHECK (salary > 0)" +
                    ")";
                createStmt.text = sql;

                try
                {
                    createStmt.execute();
                    trace("Table created");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

操作 SQL 数据库数据

Adobe AIR 1.0 和更高版本

使用本地 SQL 数据库时，会执行一些常见任务。这些任务包括连接到数据库、向数据库的表中添加数据以及从数据库的表中检索数据。执行这些任务时，还要牢记几个问题，例如使用数据类型和处理错误。

请注意，还有几个数据库任务的执行频率较低，但经常需要在执行这些更常见的任务之前执行。例如，需要创建数据库和在数据库中创建表结构，才可以连接到数据库和从表中检索数据。第 616 页的 “ 创建和修改数据库 ” 中讨论了这些执行频率较低的初始设置任务。

可以选择异步执行数据库操作，这意味着数据库引擎在后台运行并在操作成功或失败时通过调度事件来通知您。还可以同步执行这些操作。在这种情况下，数据库操作依次执行，整个应用程序 （包括对屏幕的更新）等待操作完成后再执行其他代码。有关使用异步执行模式或同步执行模式的详细信息，请参阅第 645 页的 “ 使用同步和异步数据库操作 ”。

连接到数据库

Adobe AIR 1.0 和更高版本

在执行任何数据库操作之前，请首先打开到数据库文件的连接。 SQLConnection 实例用于表示到一个或多个数据库的连接。使用 SQLConnection 实例连接的第一个数据库称为 " 主 " 数据库。此数据库是使用 open() 方法 （对于同步执行模式）或 openAsync() 方法 （对于异步执行模式）连接的。

如果使用异步 openAsync() 操作打开数据库，则注册 SQLConnection 实例的 open 事件，以便知道 openAsync() 操作何时完成。注册 SQLConnection 实例的 error 事件，以确定操作是否失败。

以下示例说明如何为异步执行打开现有的数据库文件。数据库文件名为 "DBSample.db"，位于用户的第 576 页的 " 指向应用程序存储目录 " 下。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

conn.openAsync(dbFile, SQLMode.UPDATE);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                conn.openAsync(dbFile, SQLMode.UPDATE);
            }

            private function openHandler(event:SQLEvent):void
            {
                trace("the database opened successfully");
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

以下示例说明如何为同步执行打开现有的数据库文件。数据库文件名为 “DBSample.db”，位于用户的第 576 页的 “ 指向应用程序存储目录 ” 下。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.errors.SQLError;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

try
{
    conn.open(dbFile, SQLMode.UPDATE);
    trace("the database opened successfully");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLMode;
            import flash.errors.SQLError;
            import flash.filesystem.File;

            private function init():void
            {
                var conn:SQLConnection = new SQLConnection();

                // The database file is in the application storage directory
                var folder:File = File.applicationStorageDirectory;
                var dbFile:File = folder.resolvePath("DBSample.db");

                try
                {
                    conn.open(dbFile, SQLMode.UPDATE);
                    trace("the database opened successfully");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

请注意，在异步示例的 openAsync() 方法调用中以及同步示例的 open() 方法调用中，第二个参数是常数 SQLMode.UPDATE。如果指定的文件不存在，则为第二个参数 (openMode) 指定 SQLMode.UPDATE 会导致运行时调度一个错误。如果为 openMode 参数传递 SQLMode.CREATE （或者如果使 openMode 参数处于关闭状态），则在指定的文件不存在时，运行时将尝试创建数据库文件。但是如果文件存在，则会打开该文件，这与使用 SQLMode.Update 相同。也可以为 openMode 参数指定 SQLMode.READ，以便在只读模式下打开现有的数据库。在这种情况下，可以从数据库检索数据，但是不能添加、删除或更改数据。

使用 SQL 语句

Adobe AIR 1.0 和更高版本

单个 SQL 语句 （查询或命令）在运行时中表示为 SQLStatement 对象。按照以下步骤创建和执行 SQL 语句：

创建 **SQLStatement** 实例。
在您的应用程序中， SQLStatement 对象表示 SQL 语句。

```
var selectData:SQLStatement = new SQLStatement();
```

指定对其运行查询的数据库。
为此，请将 SQLStatement 对象的 sqlConnection 属性设置为与所需数据库连接的 SQLConnection 实例。

```
// A SQLConnection named "conn" has been created previously
selectData.sqlConnection = conn;
```

指定实际的 **SQL** 语句。
将语句文本创建为字符串，并将其分配给 SQLStatement 实例的 text 属性。

```
selectData.text = "SELECT col1, col2 FROM my_table WHERE col1 = :param1";
```

定义函数以处理执行操作的结果 （仅限异步执行模式）。
使用 addEventListener() 方法将函数注册为 SQLStatement 实例的 result 和 error 事件的侦听器。

```
// using listener methods and addEventListener()

selectData.addEventListener(SQLEvent.RESULT, resultHandler);
selectData.addEventListener(SQLErrorEvent.ERROR, errorHandler);

function resultHandler(event:SQLEvent):void
{
    // do something after the statement execution succeeds
}

function errorHandler(event:SQLErrorEvent):void
{
    // do something after the statement execution fails
}
```

或者，可以使用 Responder 对象指定侦听器方法。在这种情况下，创建 Responder 实例并将侦听器方法链接到该实例。

```
// using a Responder (flash.net.Responder)

var selectResponder = new Responder(onResult, onError);

function onResult(result:SQLResult):void
{
    // do something after the statement execution succeeds
}

function onError(error:SQLError):void
{
    // do something after the statement execution fails
}
```

如果语句文本包括参数定义，则分配这些参数的值。
若要分配参数值，请使用 SQLStatement 实例的 parameters 关联数组属性。

```
selectData.parameters[":param1"] = 25;
```

执行 **SQL** 语句。
调用 SQLStatement 实例的 execute() 方法。

```
// using synchronous execution mode
// or listener methods in asynchronous execution mode
selectData.execute();
```

此外，如果在异步执行模式下使用 Responder 而不是事件侦听器，则将 Responder 实例传递到 execute() 方法。

```
// using a Responder in asynchronous execution mode
selectData.execute(-1, selectResponder);
```

有关演示这些步骤的特定示例，请参阅以下主题：

第 629 页的 “ 从数据库检索数据 ”

第 637 页的 “ 插入数据 ”

第 642 页的 “ 更改或删除数据 ”

在语句中使用参数

Adobe AIR 1.0 和更高版本

使用 SQL 语句参数，您可以创建可重用的 SQL 语句。使用语句参数时，语句中的值可以更改 （如在 INSERT 语句中添加的值），但是基本的语句文本保持不变。所以，使用参数可提供性能优势，并且可以更轻松地进行应用程序编码。

了解语句参数

Adobe AIR 1.0 和更高版本

应用程序经常在自身中多次使用单个 SQL 语句，只是稍有不同。以一个库存跟踪应用程序为例，用户可以在其中向数据库添加新的库存项目。向数据库添加库存项目的应用程序代码执行 SQL INSERT 语句，该语句实际上向数据库添加数据。但是，每次执行该语句时都稍有不同。具体来说，在表中插入的实际值是不同的，因为它们特定于所添加的库存项目。

在多次使用一个 SQL 语句但该语句中的值不同的情况下，最佳方法是使用包括参数的 SQL 语句而不是在 SQL 文本中包括字面值。参数是语句文本中的一个占位符，每次执行语句时都将它替换为实际的值。要在 SQL 语句中使用参数，请像通常一样创建 SQLStatement 实例。对于分配给 text 属性的实际 SQL 语句，使用参数占位符而不是字面值。然后通过在 SQLStatement 实例的 parameters 属性中设置元素值来定义每个参数的值。 parameters 属性是一个关联数组，所以需要使用以下语法设置特殊值：

```
statement.parameters[parameter_identifier] = value;
```

如果使用命名参数，则 parameter_identifier 是字符串；如果使用未命名参数，则它是整数索引。

使用命名参数

Adobe AIR 1.0 和更高版本

参数可以是命名参数。命名参数具有一个特定的名称，数据库使用该名称将参数值与语句文本中其占位符位置相匹配。参数名称由 “:” 或 “@” 字符后跟一个名称组成，如以下示例所示：

```
:itemName
@firstName
```

以下代码清单演示命名参数的用法：

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (:name, :productCode)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[":name"] = "Item name";
addItemStmt.parameters[":productCode"] = "12345";

addItemStmt.execute();
```

使用未命名参数

Adobe AIR 1.0 和更高版本

作为使用命名参数的一种替代方式，也可以使用未命名参数。若要使用未命名参数，请使用 “?” 字符表示 SQL 语句中的参数。按照参数在语句中的顺序，每个参数都分配有一个数字索引，数字索引从索引 0 （表示第一个参数）开始。以下示例使用未命名参数演示上述示例的一个版本：

```
var sql:String =
    "INSERT INTO inventoryItems (name, productCode)" +
    "VALUES (?, ?)";

var addItemStmt:SQLStatement = new SQLStatement();
addItemStmt.sqlConnection = conn;
addItemStmt.text = sql;

// set parameter values
addItemStmt.parameters[0] = "Item name";
addItemStmt.parameters[1] = "12345";

addItemStmt.execute();
```

使用参数的优点

Adobe AIR 1.0 和更高版本

在 SQL 语句中使用参数有以下几个优点：

性能更佳 与每次执行时动态创建 SQL 文本的 SQLStatement 实例相比，使用参数的 SQLStatement 实例可以更高效地执行。性能之所以得到提高，是因为只准备一次语句，然后可以使用不同的参数值多次执行它，而无需重新编译 SQL 语句。

显式数据类型指定 参数用于允许对构造 SQL 语句时未知的值进行类型替代。使用参数是保证将值的存储类传递到数据库的唯一方式。不使用参数时，运行时会尝试根据相关联列的类型关联将所有值从其文本表示形式转换为存储类。

有关存储类和列关联的详细信息，请参阅第 963 页的 " 数据类型支持 "。

安全性更高 使用参数有助于防止恶意技术攻击（称为 SQL 注入攻击）。在 SQL 注入攻击中，用户在用户可访问的位置（例如数据输入字段）输入 SQL 代码。如果应用程序代码通过将用户输入直接连接到 SQL 文本来构造 SQL 语句，则将对数据库执行用户输入的 SQL 代码。下面的列表显示将用户输入连接到 SQL 文本的示例。不要使用此技术：

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = '" + username + "' " +
    "    AND password = '" + password + "'";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;
```

使用语句参数而不是将用户输入的值连接到语句的文本中，可防止 SQL 注入攻击。 SQL 注入无法发生的原因是，系统将参数值明确视为替代值，而不是作为字面语句文本的一部分。下面是对前面列表的建议替代方法：

```
// assume the variables "username" and "password"
// contain user-entered data

var sql:String =
    "SELECT userId " +
    "FROM users " +
    "WHERE username = :username " +
    "    AND password = :password";

var statement:SQLStatement = new SQLStatement();
statement.text = sql;

// set parameter values
statement.parameters[":username"] = username;
statement.parameters[":password"] = password;
```

从数据库检索数据

Adobe AIR 1.0 和更高版本

从数据库检索数据分为以下两步。首先，执行 SQL SELECT 语句 （描述要从数据库检索的一组数据）。然后，访问已检索的数据，并根据需要由应用程序显示或操作它。

执行 SELECT 语句

Adobe AIR 1.0 和更高版本

要从数据库检索现有数据，请使用 SQLStatement 实例。将相应的 SQL SELECT 语句分配给实例的 text 属性，然后调用其 execute() 方法。

有关 SELECT 语句的语法的详细信息，请参阅第 946 页的 " 本地数据库中的 SQL 支持 "。

以下示例演示如何使用异步执行模式执行 SELECT 语句从名为 "products" 的表中检索数据：

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

selectStmt.execute();

function resultHandler(event:SQLEvent):void
{
    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}

function errorHandler(event:SQLErrorEvent):void
{
    // Information about the error is available in the
    // event.error property, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                selectStmt.addEventListener(SQLEvent.RESULT, resultHandler);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, errorHandler);

                selectStmt.execute();
            }

            private function resultHandler(event:SQLEvent):void
            {
                var result:SQLResult = selectStmt.getResult();

                var numResults:int = result.data.length;
                for (var i:int = 0; i < numResults; i++)
                {
                    var row:Object = result.data[i];
                    var output:String = "itemId: " + row.itemId;
                    output += "; itemName: " + row.itemName;
                    output += "; price: " + row.price;
                    trace(output);
                }
            }

            private function errorHandler(event:SQLErrorEvent):void
            {
                // Information about the error is available in the
                // event.error property, which is an instance of
                // the SQLError class.
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

以下示例演示如何使用同步执行模式执行 SELECT 语句从名为 “ 产品 ” 的表检索数据:

```
var selectStmt:SQLStatement = new SQLStatement();

// A SQLConnection named "conn" has been created previously
selectStmt.sqlConnection = conn;

selectStmt.text = "SELECT itemId, itemName, price FROM products";

try
{
    selectStmt.execute();

    var result:SQLResult = selectStmt.getResult();

    var numResults:int = result.data.length;
    for (var i:int = 0; i < numResults; i++)
    {
        var row:Object = result.data[i];
        var output:String = "itemId: " + row.itemId;
        output += "; itemName: " + row.itemName;
        output += "; price: " + row.price;
        trace(output);
    }
}
catch (error:SQLError)
{
    // Information about the error is available in the
    // error variable, which is an instance of
    // the SQLError class.
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                var selectStmt:SQLStatement = new SQLStatement();

                // A SQLConnection named "conn" has been created previously
                selectStmt.sqlConnection = conn;

                selectStmt.text = "SELECT itemId, itemName, price FROM products";

                try
                {
                    selectStmt.execute();

                    var result:SQLResult = selectStmt.getResult();
```

```
                    var numResults:int = result.data.length;
                    for (var i:int = 0; i < numResults; i++)
                    {
                        var row:Object = result.data[i];
                        var output:String = "itemId: " + row.itemId;
                        output += "; itemName: " + row.itemName;
                        output += "; price: " + row.price;
                        trace(output);
                    }
                }
                catch (error:SQLError)
                {
                    // Information about the error is available in the
                    // error variable, which is an instance of
                    // the SQLError class.
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

在异步执行模式下，语句完成执行时， SQLStatement 实例调度 result 事件 (SQLEvent.RESULT)，指示该语句已成功运行。或者，如果 Responder 对象作为参数传递给 execute() 方法，则调用 Responder 对象的结果处理函数。在同步执行模式下，执行暂停，直到 execute() 操作完成，然后继续执行下一行代码。

访问 SELECT 语句结果数据

Adobe AIR 1.0 和更高版本

SELECT 语句完成执行后，下一步是访问已检索的数据。通过调用 SQLStatement 对象的 getResult() 方法，从执行 SELECT 语句中检索结果数据：

```
var result:SQLResult = selectStatement.getResult();
```

getResult() 方法返回 SQLResult 对象。 SQLResult 对象的 data 属性是一个包含 SELECT 语句的结果的数组：

```
var numResults:int = result.data.length;
for (var i:int = 0; i < numResults; i++)
{
    // row is an Object representing one row of result data
    var row:Object = result.data[i];
}
```

SELECT 结果集中的每行数据成为包含在 data 数组中的 Object 实例。该对象具有其名称与结果集的列名称匹配的属性。该属性包含结果集的列值。例如，假定 SELECT 语句指定一个结果集，该结果集具有名为 “itemId”、 “itemName” 和 “price” 的三个列。对于结果集中的每一行，使用名为 itemId、 itemName 和 price 的属性创建 Object 实例。这些属性包含来自其相应列的值。

以下代码清单定义其文本为 SELECT 语句的 SQLStatement 实例。该语句从名为 employees 的表的所有行中检索包含 firstName 和 lastName 列值的行。此示例使用异步执行模式。执行完成时，调用 selectResult() 方法，使用 SQLStatement.getResult() 访问生成的数据行，使用 trace() 方法显示它们。请注意，此列表假定存在一个名为 conn、已进行实例化并连接到数据库的 SQLConnection 实例。它还假定已创建 “employees” 表并为其填充了数据。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

// register listeners for the result and error events
selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

// execute the statement
selectStmt.execute();

function selectResult(event:SQLEvent):void
{
    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}

function selectError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
                selectStmt.text = sql;

                // register listeners for the result and error events
                selectStmt.addEventListener(SQLEvent.RESULT, selectResult);
                selectStmt.addEventListener(SQLErrorEvent.ERROR, selectError);

                // execute the statement
                selectStmt.execute();
            }

            private function selectResult(event:SQLEvent):void
            {
                // access the result data
                var result:SQLResult = selectStmt.getResult();

                var numRows:int = result.data.length;
                for (var i:int = 0; i < numRows; i++)
                {
                    var output:String = "";
                    for (var columnName:String in result.data[i])
                    {
                        output += columnName + ": " + result.data[i][columnName] + "; ";
                    }
                    trace("row[" + i.toString() + "]\t", output);
                }
            }

            private function selectError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

以下代码清单演示与前面的代码清单相同的技术，但使用的是同步执行模式。该示例定义其文本为 SELECT 语句的 SQLStatement 实例。该语句从名为 employees 的表的所有行中检索包含 firstName 和 lastName 列值的行。使用 SQLStatement.getResult() 访问生成的数据行，并使用 trace() 方法显示它们。请注意，此列表假定存在一个名为 conn、已进行实例化并连接到数据库的 SQLConnection 实例。它还假定已创建 “employees” 表并为其填充了数据。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var selectStmt:SQLStatement = new SQLStatement();
selectStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "SELECT firstName, lastName " +
    "FROM employees";
selectStmt.text = sql;

try
{
    // execute the statement
    selectStmt.execute();

    // access the result data
    var result:SQLResult = selectStmt.getResult();

    var numRows:int = result.data.length;
    for (var i:int = 0; i < numRows; i++)
    {
        var output:String = "";
        for (var columnName:String in result.data[i])
        {
            output += columnName + ": " + result.data[i][columnName] + "; ";
        }
        trace("row[" + i.toString() + "]\t", output);
    }
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var selectStmt:SQLStatement = new SQLStatement();
                selectStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "SELECT firstName, lastName " +
                    "FROM employees";
```

```
                selectStmt.text = sql;

                try
                {
                    // execute the statement
                    selectStmt.execute();

                    // access the result data
                    var result:SQLResult = selectStmt.getResult();

                    var numRows:int = result.data.length;
                    for (var i:int = 0; i < numRows; i++)
                    {
                        var output:String = "";
                        for (var columnName:String in result.data[i])
                        {
                            output += columnName + ": ";
                            output += result.data[i][columnName] + "; ";
                        }
                        trace("row[" + i.toString() + "]\t", output);
                    }
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

定义 SELECT 结果数据的数据类型

Adobe AIR 1.0 和更高版本

默认情况下，由 SELECT 语句返回的每行都创建为 Object 实例，以结果集的列名作为属性名，每列的值作为其关联属性的值。但是，在执行 SQL SELECT 语句之前，可以将 SQLStatement 实例的 itemClass 属性设置为类。通过设置 itemClass 属性，由 SELECT 语句返回的每个行将创建为指定类的实例。通过将 SELECT 结果集中的列名与 itemClass 类中的属性名相匹配，运行时将结果列值分配给属性值。

作为 itemClass 属性值分配的任何类都必须具有不需要任何参数的构造函数。另外，对于由 SELECT 语句返回的每个列，该类必须具有一个单一的属性。如果 SELECT 列表中的列在 itemClass 类中没有相匹配的属性名称，系统会将其视为错误。

检索部分 SELECT 结果

Adobe AIR 1.0 和更高版本

默认情况下，执行 SELECT 语句会一次检索结果集的所有行。语句完成后，通常以某种方式处理检索的数据，如创建对象或在屏幕上显示数据。如果语句返回了大量的行，则同时处理所有数据可能对计算机要求过高，这又会导致用户界面无法自行重绘。

通过指示运行时一次返回特定数量的结果行，可以提高应用程序的感知性能。这样做会使初始结果数据更快地返回。它还允许您将结果行分到各组中，以便在处理每组行后更新用户界面。请注意，只有在异步执行模式下使用此技术才是可行的。

要检索部分 SELECT 结果，请为 SQLStatement.execute() 方法的第一个参数（prefetch 参数）指定一个值。prefetch 参数指示首次执行语句时检索的行数。调用 SQLStatement 实例的 execute() 方法时，请指定 prefetch 参数值，并只检索由此值指定的行数：

```
var stmt:SQLStatement = new SQLStatement();
stmt.sqlConnection = conn;

stmt.text = "SELECT ...";

stmt.addEventListener(SQLEvent.RESULT, selectResult);

stmt.execute(20); // only the first 20 rows (or fewer) are returned
```

该语句调度 result 事件，指示第一组结果行是可用的。得到的 SQLResult 实例的 data 属性包含数据行，并且其 complete 属性指示是否存在要检索的其他结果行。要检索其他结果行，请调用 SQLStatement 实例的 next() 方法。与 execute() 方法一样，使用 next() 方法的第一个参数指示下次调度 result 事件时要检索的行数。

```
function selectResult(event:SQLEvent):void
{
    var result:SQLResult = stmt.getResult();
    if (result.data != null)
    {
        // ... loop through the rows or perform other processing ...

        if (!result.complete)
        {
            stmt.next(20); // retrieve the next 20 rows
        }
        else
        {
            stmt.removeEventListener(SQLEvent.RESULT, selectResult);
        }
    }
}
```

每次 next() 方法返回后续的一组结果行时， SQLStatement 都会调度 result 事件。因此，可以使用同一侦听器函数继续处理结果 （通过 next() 调用），直到检索了所有行。

有关详细信息，请参阅 SQLStatement.execute() 方法 （prefetch 参数描述）和 SQLStatement.next() 方法的描述。

插入数据

Adobe AIR 1.0 和更高版本

向数据库添加数据包括执行 SQL INSERT 语句。语句完成执行后，如果数据库生成了键，则可以访问新插入的行的主键。

执行 INSERT 语句

Adobe AIR 1.0 和更高版本

要向数据库中的表添加数据，请创建并执行其文本为 SQL INSERT 语句的 SQLStatement 实例。

以下示例使用 SQLStatement 实例向已存在的 employees 表添加数据行。此示例演示如何使用异步执行模式插入数据。请注意，此列表假定存在一个名为 conn 的 SQLConnection 实例，并且该实例已经实例化并连接到数据库。它还假定已创建 “employees” 表。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

// register listeners for the result and failure (status) events
insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

// execute the statement
insertStmt.execute();

function insertResult(event:SQLEvent):void
{
    trace("INSERT statement succeeded");
}

function insertError(event:SQLErrorEvent):void
{
    trace("Error message:", event.error.message);
    trace("Details:", event.error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.events.SQLErrorEvent;
            import flash.events.SQLEvent;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                // register listeners for the result and failure (status) events
```

```
                insertStmt.addEventListener(SQLEvent.RESULT, insertResult);
                insertStmt.addEventListener(SQLErrorEvent.ERROR, insertError);

                // execute the statement
                insertStmt.execute();
            }

            private function insertResult(event:SQLEvent):void
            {
                trace("INSERT statement succeeded");
            }

            private function insertError(event:SQLErrorEvent):void
            {
                trace("Error message:", event.error.message);
                trace("Details:", event.error.details);
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

以下示例使用同步执行模式向已存在的 employees 表添加数据行。请注意，此列表假定存在一个名为 conn 的 SQLConnection 实例，并且该实例已经实例化并连接到数据库。它还假定已创建 “employees” 表。

```
import flash.data.SQLConnection;
import flash.data.SQLResult;
import flash.data.SQLStatement;
import flash.errors.SQLError;

// ... create and open the SQLConnection instance named conn ...

// create the SQL statement
var insertStmt:SQLStatement = new SQLStatement();
insertStmt.sqlConnection = conn;

// define the SQL text
var sql:String =
    "INSERT INTO employees (firstName, lastName, salary) " +
    "VALUES ('Bob', 'Smith', 8000)";
insertStmt.text = sql;

try
{
    // execute the statement
    insertStmt.execute();

    trace("INSERT statement succeeded");
}
catch (error:SQLError)
{
    trace("Error message:", error.message);
    trace("Details:", error.details);
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.data.SQLConnection;
            import flash.data.SQLResult;
            import flash.data.SQLStatement;
            import flash.errors.SQLError;

            private function init():void
            {
                // ... create and open the SQLConnection instance named conn ...

                // create the SQL statement
                var insertStmt:SQLStatement = new SQLStatement();
                insertStmt.sqlConnection = conn;

                // define the SQL text
                var sql:String =
                    "INSERT INTO employees (firstName, lastName, salary) " +
                    "VALUES ('Bob', 'Smith', 8000)";
                insertStmt.text = sql;

                try
                {
                    // execute the statement
                    insertStmt.execute();
                    trace("INSERT statement succeeded");
                }
                catch (error:SQLError)
                {
                    trace("Error message:", error.message);
                    trace("Details:", error.details);
                }
            }
        ]]>
    </mx:Script>
</mx:WindowedApplication>
```

检索已插入行的数据库生成的主键

Adobe AIR 1.0 和更高版本

通常，向表中插入数据行后，代码需要知道新插入行的数据库生成的主键或行标识符值。例如，在一个表中插入行后，您可能希望在相关表中添加行。在这种情况下，希望将主键值作为外键插入到相关表中。新插入的行的主键可使用与语句执行相关联的 SQLResult 对象进行检索。这是在执行 SELECT 语句后用来访问结果数据的同一对象。与任何 SQL 语句一样， INSERT 语句执行完成时，运行时将创建 SQLResult 实例。如果您使用的是事件侦听器或同步执行模式，则可通过调用 SQLStatement 对象的 getResult() 方法来访问 SQLResult 实例。或者，如果您使用的是异步执行模式并将 Responder 实例传递给 execute() 调用，则 SQLResult 实例将作为参数传递给结果处理函数。在任一情况下， SQLResult 实例都具有属性 lastInsertRowID ；如果执行的 SQL 语句是 INSERT 语句，则该属性包含最近插入的行的行标识符。

以下示例演示如何在异步执行模式下访问已插入行的主键：

```
insertStmt.text = "INSERT INTO ...";

insertStmt.addEventListener(SQLEvent.RESULT, resultHandler);

insertStmt.execute();

function resultHandler(event:SQLEvent):void
{
    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
```

以下示例演示如何在同步执行模式下访问已插入行的主键：

```
insertStmt.text = "INSERT INTO ...";

try
{
    insertStmt.execute();

    // get the primary key
    var result:SQLResult = insertStmt.getResult();

    var primaryKey:Number = result.lastInsertRowID;
    // do something with the primary key
}
catch (error:SQLError)
{
    // respond to the error
}
```

请注意，根据以下规则，行标识符可能是也可能不是在表定义中指定为主键列的列值：

- 如果表是用其关联 （列数据类型）为 INTEGER 的主键列定义的，则 lastInsertRowID 属性包含插入到该行中的值 （如果它是 AUTOINCREMENT 列，则为由运行时生成的值）。
- 如果表是用多个主键列 （组合键）或其关联不是 INTEGER 的单个主键列定义的，则数据库将在后台为行生成整数行标识符值。该生成的值是 lastInsertRowID 属性的值。
- 该值始终是最近插入的行的行标识符。如果 INSERT 语句导致一个触发器激发 （这将插入一行），则 lastInsertRowID 属性包含由触发器插入的最后一行的行标识符，而不是由 INSERT 语句创建的行的行标识符。

由于存在这些规则，因此，如果您希望获得明确定义的主键列，并且其值通过 SQLResult.lastInsertRowID 属性调用 INSERT 命令后获得，则该列必须定义为 INTEGER PRIMARY KEY 列。即使表不包括显式 INTEGER PRIMARY KEY 列，但就定义与相关表的关系而言，将数据库生成的行标识符用作表的主键也是同样可接受的。通过使用特殊的列名 ROWID、 _ROWID_ 或 OID 之一，行标识符列值在任何 SQL 语句中都是可用的。可以在相关表中创建外键列，并将行标识符值用作外键列值，就像明确声明的 INTEGER PRIMARY KEY 列一样。在该意义上，如果使用的是任意主键而不是自然键，并且只要您不在乎生成主键值的运行时，则将 INTEGER PRIMARY KEY 列还是系统生成的行标识符用作表的主键以定义两个表之间的外键关系几乎没有差别。

有关主键和生成的行标识符的详细信息，请参阅第 946 页的 “ 本地数据库中的 SQL 支持 ”。

更改或删除数据

Adobe AIR 1.0 和更高版本

执行其他数据处理操作的过程和用于执行 SQL SELECT 或 INSERT 语句的过程相同，具体内容在第 625 页的 " 使用 SQL 语句 " 中进行了介绍。仅需使用 SQLStatement 实例的 text 属性中的不同 SQL 语句进行替换即可：

- 若要更改表中的现有数据，请使用 UPDATE 语句。
- 要从表中删除一行或多行数据，请使用 DELETE 语句。

有关这些语句的说明，请参阅第 946 页的 " 本地数据库中的 SQL 支持 "。

使用多个数据库

Adobe AIR 1.0 和更高版本

使用 SQLConnection.attach() 方法，在已具有打开的数据库的 SQLConnection 实例上打开到其他数据库的连接。在 attach() 方法调用中，使用 name 参数为附加的数据库提供一个名称。在编写语句操作该数据库时，可以在前缀中使用该名称 （使用格式 database-name.table-name）在 SQL 语句中限定任何表名，指示运行时可以在指定的数据库中找到该表。

可以执行包括多个数据库中的表的单个 SQL 语句，这些数据库连接到同一 SQLConnection 实例。如果事务是在 SQLConnection 实例上创建的，则该事务适用于使用 SQLConnection 实例执行的所有 SQL 语句。不管语句运行在哪个附加的数据库上，这一点都适用。

或者，也可以在一个应用程序中创建多个 SQLConnection 实例，其中每个实例都连接到一个或多个数据库。但是，如果确实使用到同一数据库的多个连接，请牢记数据库事务不是跨 SQLConnection 实例共享的。因此，如果使用多个 SQLConnection 实例连接到同一数据库文件，则不能指望以预期方式应用这两个连接的数据更改。例如，如果通过不同的 SQLConnection 实例对同一数据库运行两个 UPDATE 或 DELETE 语句，并且在一个操作发生后出现应用程序错误，则数据库数据可能处于不可逆的中间状态，而且可能影响数据库的完整性 （进而影响应用程序）。

处理数据库错误

Adobe AIR 1.0 和更高版本

通常，数据库错误处理与其他运行时错误处理类似。应该编写代码以备可能出现的错误，并对错误作出响应，而不是直到运行时才这样做。通常认为，可以将可能的数据库错误分为以下三类：连接错误、 SQL 语法错误和约束错误。

连接错误

Adobe AIR 1.0 和更高版本

大多数的数据库错误是连接错误，它们可能出现在任何操作过程中。尽管存在防止连接错误的策略，但是，如果数据库是应用程序的关键部分，则几乎没有从连接错误中正常恢复的简单方法。

大多数连接错误与运行时和操作系统、文件系统及数据库文件交互的方式有关。例如，如果用户没有在文件系统上的特定位置创建数据库文件的权限，则会出现连接错误。以下策略有助于防止连接错误：

使用特定于用户的数据库文件 为每个用户提供其自己的数据库文件，而不是将单个数据库文件用于在单个计算机上使用应用程序的所有用户。该文件应该位于与用户帐户关联的目录中。例如，它可能在以下位置：应用程序的存储目录、用户的文档文件夹、用户的桌面等。

考虑不同的用户类型 在不同的操作系统上，用不同类型的用户帐户测试应用程序。请勿假定用户具有计算机上的管理员权限。此外，请勿假定安装了某应用程序的个人是运行该应用程序的用户。

考虑各个文件位置 如果允许用户指定保存数据库文件的位置或者选择要打开的文件，请考虑用户可能使用的文件位置。此外，请考虑定义对用户可以存储 （或他们可以从中打开）数据库文件的位置的限制。例如，可以仅允许用户打开位于其用户帐户存储位置中的文件。

如果出现连接错误，则很可能出现在创建或打开数据库的首次尝试中。这意味着用户无法在应用程序中执行与数据库相关的任何操作。对于某些类型的错误，如只读或权限错误，一种可能的恢复技术是将数据库文件复制到其他位置。应用程序可以将数据库文件复制到用户有权创建和写入文件的其他位置，然后改用该位置。

语法错误

Adobe AIR 1.0 和更高版本

在 SQL 语句格式不正确，而应用程序尝试执行该语句时，会出现语法错误。由于本地数据库 SQL 语句是作为字符串创建的，因此无法进行编译时 SQL 语法检查。必须执行所有 SQL 语句才能检查其语法。使用以下策略可防止 SQL 语法错误：

全面地测试所有 SQL 语句 如有可能，在开发应用程序的过程中，将 SQL 语句编码为应用程序代码中的语句文本之前，单独对其进行测试。此外，使用代码测试方法 （如单元测试）创建一组测试，在代码中运用每个可能的选项和变体。

使用语句参数和避免连接的 （动态生成的） SQL 使用参数和避免动态生成的 SQL 语句，意味着每次执行语句时都使用相同的 SQL 语句文本。因此，测试语句和限制可能的变体更为容易。如果必须动态生成 SQL 语句，请将语句的动态部分保持在最小限度内。此外，仔细验证任何用户输入，以确保它不会导致语法错误。

若要从语法错误中恢复，应用程序将需要复杂的逻辑才能检查 SQL 语句和更正其语法。通过遵循用于防止语法错误的上述准则，您的代码可以识别 SQL 语法错误的任何潜在运行时根源 （如语句中使用的用户输入）。若要从语法错误中恢复，请为用户提供指导。指出要更正哪些内容才能使语句正确执行。

约束错误

Adobe AIR 1.0 和更高版本

在 INSERT 或 UPDATE 语句尝试向列添加数据时，会出现约束错误。如果新数据违反表或列的已定义约束之一，则会发生该错误。一组可能的约束包括：

唯一约束 指示对于表中的所有行，在一个列中不能有重复值。或者，将多个列组合在唯一约束中时，这些列中值的组合不得重复。换句话说，对于指定的具有唯一性的一列或多列，每个行必须是不同的。

主键约束 对于约束允许和不允许的数据，主键约束与唯一约束完全相同。

非 null 约束 指定单个列不能存储 NULL 值，因此在每个行中，该列必须具有一个值。

检查约束 允许您在一个或多个表上指定任意约束。常见的检查约束是一个规则，它定义列的值必须在某些界限内 （例如，数字列的值必须大于 0）。另一种常见的检查约束类型指定列值之间的关系 （例如，一个列的值必须与同一行中其他列的值不同）。

数据类型 （列关联）约束 运行时强制实施列值的数据类型，尝试将类型不正确的值存储在列中时会出现错误。但是，在许多情况下，会转换值以匹配列的已声明数据类型。有关详细信息，请参阅第 644 页的 “ 使用数据库数据类型 ”。

运行时不对外键值强制实施约束。换句话说，匹配现有的主键值不需要外键值。

除了预定义的约束类型外，运行时 SQL 引擎还支持使用触发器。触发器类似于事件处理函数。它是发生某个操作时执行的一组预定义指令。例如，可以定义一个触发器，它在向特定表插入数据或从中删除数据时运行。触发器的一个可能用途是检查数据更改，并在不满足指定的条件时导致出现错误。因此，触发器可以具有与约束相同的用途，防止约束错误和从中恢复的策略也适用于触发器生成的错误。但是，触发器生成的错误的错误 id 与约束错误的错误 id 不同。

在设计应用程序时，就确定了适用于特定表的一组约束。通过有意识地设计约束，可以更轻松地设计应用程序，以防止约束错误和从中恢复。但是，约束错误很难系统地预测和预防。很难预测的原因是，约束错误在添加应用程序数据后才出现。数据在创建后被添加到数据库时会出现约束错误。这些错误通常是由新数据和已经存在于数据库中的数据之间的关系导致的。以下策略可以帮助您避免许多约束错误：

仔细计划数据库结构和约束 约束的用途是强制实施应用程序规则和帮助保护数据库数据的完整性。在计划应用程序时，请考虑如何构建数据库来支持您的应用程序。作为该过程的一部分，确定数据的规则，如某些值是否必需、某值是否具有默认值、是否允许重复值等。这些规则可以指导您定义数据库约束。

明确指定列名 可以编写 INSERT 语句而不明确指定要在其中插入值的列，但是这样做会产生不必要的风险。通过明确命名要在其中插入值的列，可以允许自动生成的值、具有默认值的列和允许 NULL 值的列。此外，这样做可确保所有的 NOT NULL 列都插入了显式值。

使用默认值 每当为列指定 NOT NULL 约束时，尽可能在列定义中指定默认值。应用程序代码也可以提供默认值。例如，代码可以检查 String 变量是否为 null，并在使用它设置语句参数值之前为它分配一个值。

验证用户输入的数据 提前检查用户输入的数据，以确保它符合约束指定的限制，尤其是对于 NOT NULL 和 CHECK 约束。当然， UNIQUE 约束更难检查，因为这样做会要求执行 SELECT 查询来确定数据是否唯一。

使用触发器 可以编写一个触发器，用于验证 （并可能替换）插入的数据或执行其他操作以更正无效的数据。此验证和更正可防止出现约束错误。

在许多方面，约束错误比其他类型的错误更难防范。幸运的是，有几个从约束错误恢复的策略，这样就不会使应用程序变得不稳定或不可用：

使用冲突算法 在列上定义约束时，以及创建 INSERT 或 UPDATE 语句时，您可以选择指定冲突算法。冲突算法定义在出现约束违规时数据库执行的操作。数据库引擎可以执行几种可能的操作。数据库引擎可以结束单个语句或整个事务。它可以忽略错误。它甚至可以删除旧数据，并将它替换为代码尝试存储的数据。

有关详细信息，请参阅第 946 页的 “ 本地数据库中的 SQL 支持 ” 中的 “ON CONFLICT （冲突算法） ” 部分。

提供纠正反馈 可以提前识别可能影响特定 SQL 命令的一组约束。因此，可以预期语句可能导致的约束错误。知道这一点，就可以生成应用程序逻辑来响应约束错误。例如，假定应用程序包括用于输入新产品的数据条目表单。如果数据库中的产品名称列是用 UNIQUE 约束定义的，则在数据库中插入新产品行的操作可能会导致约束错误。因此，应用程序设计用于预期约束错误。在错误发生时，应用程序将提醒用户，指出指定的产品名称已在使用中，并要求用户选择其他名称。另一种可能的响应是，允许用户查看有关同名的其他产品的信息。

使用数据库数据类型

Adobe AIR 1.0 和更高版本

在数据库中创建表时，用于创建表的 SQL 语句将为表中的每个列定义关联或数据类型。尽管可以省略关联声明，但是最好在 CREATE TABLE SQL 语句中明确声明列关联。

通常，在执行 SELECT 语句时，使用 INSERT 语句存储在数据库中的任何对象都将作为相同数据类型的实例返回。但是，已检索值的数据类型可能随存储该值的数据库列的关联的不同而不同。当值存储在列中时，如果其数据类型与列的关联不匹配，则数据库会尝试转换该值以便与列的关联匹配。例如，如果数据库列是用 NUMERIC 关联声明的，则在存储数据之前，数据库会尝试将插入的数据转换为数字存储类 （INTEGER 或 REAL）。如果无法转换数据，则数据库将引发错误。按照此规则，如果将字符串 “12345” 插入到 NUMERIC 列中，则在将它存储在数据库中之前，数据库会自动将它转换为整数值 12345。使用 SELECT 语句检索该值时，它将作为数字数据类型 （如 Number）的实例而不是 String 实例返回。

避免不需要的数据类型转换的最佳方式是遵循以下两个规则。首先，使用与其要存储的数据类型匹配的关联定义每个列。其次，仅插入其数据类型与定义的关联匹配的值。遵循这些规则有两个优点。插入数据时，不会意外转换它 （结果是可能丢失其预期含义）。此外，检索数据时，它会按其原始数据类型返回。

有关可用列关联类型以及如何在 SQL 语句中使用数据类型的详细信息，请参阅第 963 页的 “ 数据类型支持 ”。

使用同步和异步数据库操作

Adobe AIR 1.0 和更高版本

前面几节已描述常见的数据库操作，如检索、插入、更新和删除数据，以及在数据库中创建数据库文件和表以及其他对象。示例已演示如何以异步和同步方式执行这些操作。

需要提醒的是，在异步执行模式下，您指示数据库引擎执行操作。然后，在应用程序保持运行的同时，数据库引擎在后台工作。当操作完成时，数据库引擎调度事件以提醒您该情况。异步执行的主要优点是，在主应用程序代码继续执行的同时，运行时在后台执行数据库操作。当操作运行所用时间非常长时，这尤其有价值。

另一方面，在同步执行模式下，操作不在后台运行。通知数据库引擎执行操作。代码在数据库引擎工作时暂停。完成操作后，继续执行下一行代码。

使用单个数据库连接，无法同步执行某些操作或语句，同时异步执行其他操作或语句。指定当您打开到数据库的连接时，是异步还是同步操作 SQLConnection。如果调用 SQLConnection.open()，则连接在同步执行模式下操作；如果调用 SQLConnection.openAsync()，则连接在异步执行模式下操作。使用 open() 或 openAsync() 将 SQLConnection 实例连接到数据库后，该实例将固定为同步或异步执行。

使用同步数据库操作

Adobe AIR 1.0 和更高版本

与异步执行模式的代码相比，使用同步执行时用于执行和响应操作的实际代码几乎没有差异。两种方法之间的主要差异体现在以下两个方面。首先，执行一个依赖于另一个操作 （如 SELECT 结果行或由 INSERT 语句添加的行的主键）的操作。第二方面的差异体现在处理错误上。

为同步操作编写代码

Adobe AIR 1.0 和更高版本

同步执行和异步执行的主要差异在于：在同步模式下，以单个步骤系列的形式编写代码。相反，在异步代码中，注册事件侦听器，并经常在侦听器方法之间分配操作。当在同步执行模式下连接数据库时，可以在单个代码块中连续执行一系列数据库操作。以下示例对此技术进行了演示：

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, OpenMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

如您所看到的，不管使用的是同步执行还是异步执行，都调用相同的方法来执行数据库操作。两种方法的主要差异在于：执行一个依赖于另一个操作的操作和处理错误。

执行一个依赖于另一个操作的操作

Adobe AIR 1.0 和更高版本

使用同步执行模式时，无需编写侦听事件的代码来确定操作完成的时间。相反，可以假定如果一个代码行中的操作成功完成，则继续执行下一代码行。因此，要执行一个依赖于另一个操作成功的操作，只需在紧随它所依赖的操作之后编写相关代码即可。例如，要为应用程序编码以开始事务，可执行 INSERT 语句，检索已插入行的主键，将该主键插入到不同表的其他行中，最后提交事务，可以将代码全部编写为一系列语句。以下示例对这些操作进行了演示：

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

// add the customer record to the database
var insertCustomer:SQLStatement = new SQLStatement();
insertCustomer.sqlConnection = conn;
insertCustomer.text =
    "INSERT INTO customers (firstName, lastName) " +
    "VALUES ('Bob', 'Jones')";
insertCustomer.execute();

var customerId:Number = insertCustomer.getResult().lastInsertRowID;

// add a related phone number record for the customer
var insertPhoneNumber:SQLStatement = new SQLStatement();
insertPhoneNumber.sqlConnection = conn;
insertPhoneNumber.text =
    "INSERT INTO customerPhoneNumbers (customerId, number) " +
    "VALUES (:customerId, '800-555-1234')";
insertPhoneNumber.parameters[":customerId"] = customerId;
insertPhoneNumber.execute();

// commit the transaction
conn.commit();
```

在同步执行时处理错误

Adobe AIR 1.0 和更高版本

在同步执行模式下，不侦听错误事件来确定操作是否已失败。相反，将可能触发错误的任何代码括在一组 try..catch..finally 代码块中。将引发错误的代码包装在 try 块中。在单独的 catch 块中，编写响应每种类型的错误时要执行的操作。在 finally 块中放置不管成功还是失败（例如，关闭不再需要的数据库连接）都希望始终执行的任何代码。以下示例演示如何使用 try..catch..finally 块进行错误处理。它建立在前面示例的基础之上，添加了错误处理代码：

```
var conn:SQLConnection = new SQLConnection();

// The database file is in the application storage directory
var folder:File = File.applicationStorageDirectory;
var dbFile:File = folder.resolvePath("DBSample.db");

// open the database
conn.open(dbFile, SQLMode.UPDATE);

// start a transaction
conn.begin();

try
{
    // add the customer record to the database
    var insertCustomer:SQLStatement = new SQLStatement();
    insertCustomer.sqlConnection = conn;
    insertCustomer.text =
        "INSERT INTO customers (firstName, lastName)" +
        "VALUES ('Bob', 'Jones')";

    insertCustomer.execute();

    var customerId:Number = insertCustomer.getResult().lastInsertRowID;

    // add a related phone number record for the customer
    var insertPhoneNumber:SQLStatement = new SQLStatement();
    insertPhoneNumber.sqlConnection = conn;
    insertPhoneNumber.text =
        "INSERT INTO customerPhoneNumbers (customerId, number)" +
        "VALUES (:customerId, '800-555-1234')";
    insertPhoneNumber.parameters[":customerId"] = customerId;

    insertPhoneNumber.execute();

    // if we've gotten to this point without errors, commit the transaction
    conn.commit();
}
catch (error:SQLError)
{
    // rollback the transaction
    conn.rollback();
}
```

了解异步执行模式

Adobe AIR 1.0 和更高版本

使用异步执行模式的一个常见问题就是，假设您当前正在对同一个数据库连接执行某个 SQLStatement 实例，则无法开始执行另一个 SQLStatement 实例。事实上，此假定是不正确的。在 SQLStatement 实例执行的同时，无法更改语句的 text 属性。但是，如果对要执行的每个不同 SQL 语句使用单独的 SQLStatement 实例，则可以在其他 SQLStatement 实例仍执行的同时调用 SQLStatement 的 execute() 方法，且不会导致错误。

在内部，当您使用异步执行模式执行数据库操作时，每个数据库连接（每个 SQLConnection 实例）都具有自己的队列或指示它执行的操作列表。运行时依次执行每个操作（按照将它们添加到队列的顺序）。创建 SQLStatement 实例并调用其 execute() 方法时，会将该语句执行操作添加到连接队列。如果在该 SQLConnection 实例上当前未执行操作，则语句将在后台开始执行。假定在同一代码块中，创建另一个 SQLStatement 实例，并且也调用该方法的 execute() 方法。该第二个语句执行操作将添加到队列中的第一个语句之后。在第一个语句完成执行后，运行时立即移动到队列中的下一个操作。队列中后续操作的处理发生在后台，即使在主应用程序代码中调度第一个操作的 result 事件也如此。以下代码对此技术进行了演示：

```
// Using asynchronous execution mode
var stmt1:SQLStatement = new SQLStatement();
stmt1.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt1.execute();

// At this point stmt1's execute() operation is added to conn's execution queue.

var stmt2:SQLStatement = new SQLStatement();
stmt2.sqlConnection = conn;

// ... Set statement text and parameters, and register event listeners ...

stmt2.execute();

// At this point stmt2's execute() operation is added to conn's execution queue.
// When stmt1 finishes executing, stmt2 will immediately begin executing
// in the background.
```

数据库自动执行排队的后续语句会产生另一个重要效果。如果一个语句依赖于另一个操作的结果，则在第一个操作完成之前，无法将该语句添加到队列中 （换句话说，无法调用其 execute() 方法）。这是因为调用第二个语句的 execute() 方法后，无法更改该语句的 text 或 parameters 属性。在此情况下，在开始下一个操作之前，必须等待指示第一个操作已完成的事件。例如，如果要在事务的上下文中执行语句，则该语句的执行将取决于打开事务的操作。调用 SQLConnection.begin() 方法打开事务后，需要等待 SQLConnection 实例调度其 begin 事件。只有这时才能调用 SQLStatement 实例的 execute() 方法。在此示例中，组织应用程序以确保正确执行操作的最简单方法是，创建一个方法，并将其注册为 begin 事件的侦听器。将调用 SQLStatement.execute() 方法的代码放置在该侦听器方法中。

对 SQL 数据库使用加密

Adobe AIR 1.5 和更高版本

所有 Adobe AIR 应用程序都共享同一个本地数据库引擎。因此，任何 AIR 应用程序都可以连接到、读取和写入未加密的数据库文件。从 Adobe AIR 1.5 起，AIR 中加入了创建和连接到加密数据库文件的功能。使用加密数据库时，应用程序必须提供正确的加密密钥才能连接到数据库。如果提供的加密密钥有误 （或不提供密钥），则应用程序无法连接到数据库。因此，应用程序无法从数据库中读取数据，也无法写入数据库或更改数据库中的数据。

若要使用加密数据库，创建数据库时必须将其创建为加密数据库。有了加密数据库，即可打开到数据库的连接。还可以更改加密数据库的加密密钥。除了创建和连接到加密数据库之外，处理加密数据库的方法与处理未加密数据库的方法相同。尤其是无论数据库是否加密，执行 SQL 语句的方式都相同。

加密数据库的用途

Adobe AIR 1.5 和更高版本

希望限制对数据库中所存储信息的访问时，加密很有帮助。 Adobe AIR 的数据库加密功能可以用于多种用途。下面是需要使用加密数据库的一些示例情况：

- 从服务器下载的专用应用程序数据的只读缓存
- 与服务器进行同步 （向服务器发送数据以及从服务器加载数据）的专用数据的本地应用程序存储区
- 用作由应用程序创建和编辑的文档的文件格式的加密文件。可以专用于一个用户、或可以在应用程序的所有用户中共享的文件。

- 本地数据存储区的任何其他用途 (如第 613 页的 " 本地 SQL 数据库的用途 " 中所述) ，在这些用途中不能向有权访问计算机或数据库文件的人员公开数据。

了解需要使用加密数据库的原因有助于您确定构建应用程序的方式。特别是，它可影响您的应用程序为数据库创建、获取及存储加密密钥的方式。有关这些注意事项的详细信息，请参阅第 653 页的 " 对数据库使用加密的注意事项 "。

除了加密数据库，加密本地存储区是用于保存私有敏感数据的另一种机制。使用加密本地存储区，可使用字符串密钥存储单一 ByteArray 值。只有存储该值的 AIR 应用程序可访问它，而且只能在存储该值的计算机上进行访问。在采用加密本地存储区的情况下，不需要创建自己的加密密钥。出于这些原因，加密本地存储区最适合于方便地存储易于在 ByteArray 中编码的一个值或一组值。加密数据库最适合于需要结构化数据存储和查询的大型数据集。有关使用加密本地存储的详细信息，请参阅第 610 页的 " 加密的本地存储区 "。

创建加密数据库

Adobe AIR 1.5 和更高版本

若要使用加密数据库，则创建数据库文件时必须将其加密。一旦在不加密的情况下创建数据库，以后就无法再对其进行加密。同样，以后也无法对加密数据库进行解密。如有必要，可以更改加密数据库的加密密钥。有关详细信息，请参阅第 652 页的 " 更改数据库的加密密钥 "。如果现有的数据库未加密，而您又希望使用数据库加密，则可以新建一个加密数据库，然后将现有的表结构和数据复制到新数据库中。

创建加密数据库与创建未加密数据库几乎完全相同，如第 617 页的 " 创建数据库 " 中所述。首先创建一个表示数据库连接的 SQLConnection 实例。通过调用 SQLConnection 对象的 open() 方法或 openAsync() 方法创建数据库，并为数据库位置指定一个尚未存在的文件。创建加密数据库时的唯一区别在于要为 encryptionKey 参数 （open() 方法的第五个参数和 openAsync() 方法的第六个参数）提供值。

有效的 encryptionKey 参数值为正好包含 16 个字节的 ByteArray 对象。

下列示例演示创建加密数据库。为简洁起见，在这些示例中，加密密钥在应用程序代码中采用硬编码形式。但是，由于此方法不安全，强烈建议不要使用此方法。

```
var conn:SQLConnection = new SQLConnection();

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

// Create an encrypted database in asynchronous mode
conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);

// Create an encrypted database in synchronous mode
conn.open(dbFile, SQLMode.CREATE, false, 1024, encryptionKey);
```

有关介绍生成加密密钥的建议方式的示例，请参阅第 654 页的 " 示例：生成和使用加密密钥 "。

连接到加密数据库

Adobe AIR 1.5 和更高版本

与创建加密数据库类似的是，打开到加密数据库的连接所采用的步骤类似于连接到未加密数据库。该步骤在第 623 页的 " 连接到数据库 " 中有更详细的说明。使用 open() 方法在同步执行模式下打开连接，或者使用 openAsync() 方法在异步执行模式下打开连接。唯一的区别在于，若要打开加密数据库，要为 encryptionKey 参数 （open() 方法的第五个参数和 openAsync() 方法的第六个参数）指定正确的值。

如果所提供的加密密钥有误，则会出错。对于 open() 方法，将引发 SQLError 异常。对于 openAsync() 方法，SQLConnection 对象将调度 SQLErrorEvent，其 error 属性包含 SQLError 对象。在任一情况下，由异常生成的 SQLError 对象的 errorID 属性值都为 3138。该错误 ID 对应于错误消息 " 所打开的文件不是数据库文件 "。

以下示例介绍以异步执行模式打开加密数据库。为了简单起见，此示例中的加密密钥在应用程序代码中采用硬编码形式。但是，由于此方法不安全，强烈建议不要使用此方法。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.events.SQLErrorEvent;
import flash.events.SQLEvent;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, errorHandler);
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

conn.openAsync(dbFile, SQLMode.UPDATE, null, false, 1024, encryptionKey);

function openHandler(event:SQLEvent):void
{
    trace("the database opened successfully");
}

function errorHandler(event:SQLErrorEvent):void
{
    if (event.error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", event.error.message);
        trace("Details:", event.error.details);
    }
}
```

以下示例介绍以同步执行模式打开加密数据库。为了简单起见，此示例中的加密密钥在应用程序代码中采用硬编码形式。但是，由于此方法不安全，强烈建议不要使用此方法。

```
import flash.data.SQLConnection;
import flash.data.SQLMode;
import flash.filesystem.File;

var conn:SQLConnection = new SQLConnection();
var dbFile:File = File.applicationStorageDirectory.resolvePath("DBSample.db");

var encryptionKey:ByteArray = new ByteArray();
encryptionKey.writeUTFBytes("Some16ByteString"); // This technique is not secure!

try
{
    conn.open(dbFile, SQLMode.UPDATE, false, 1024, encryptionKey);
    trace("the database was created successfully");
}
catch (error:SQLError)
{
    if (error.errorID == 3138)
    {
        trace("Incorrect encryption key");
    }
    else
    {
        trace("Error message:", error.message);
        trace("Details:", error.details);
    }
}
```

有关介绍生成加密密钥的建议方式的示例，请参阅第 654 页的 “ 示例：生成和使用加密密钥 ”。

更改数据库的加密密钥

Adobe AIR 1.5 和更高版本

数据库加密后，可以在以后更改数据库的加密密钥。要更改数据库的加密密钥，请首先创建一个 SQLConnection 实例并调用其 open() 或 openAsync() 方法，从而打开到数据库的连接。连接数据库后，调用 reencrypt() 方法，并传递新的加密密钥作为参数。

与大多数数据库操作类似的是， reencrypt() 方法的行为根据数据库连接使用同步还是异步执行模式而有所不同。如果使用 open() 方法连接到数据库，则 reencrypt() 操作会同步运行。操作完成后，继续执行下一行代码：

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey
conn.reencrypt(newKey);
```

另一方面，如果使用 openAsync() 方法打开数据库连接，则 reencrypt() 操作为异步方式。调用 reencrypt() 将开始重新加密的过程。操作完成后， SQLConnection 对象调度一个 reencrypt 事件。使用事件侦听器确定重新加密完成的时间：

```
var newKey:ByteArray = new ByteArray();
// ... generate the new key and store it in newKey

conn.addEventListener(SQLEvent.REENCRYPT, reencryptHandler);

conn.reencrypt(newKey);

function reencryptHandler(event:SQLEvent):void
{
    // save the fact that the key changed
}
```

reencrypt() 操作在其自身的事务中运行。如果操作中断或失败 （例如，如果在操作完成之前关闭应用程序），则事务将回滚。在这种情况下，原始的加密密钥仍为数据库的加密密钥。

reencrypt() 方法不能用于解除对数据库的加密。向 reencrypt() 方法传递 null 值或非 16 字节 ByteArray 的加密密钥会导致错误。

对数据库使用加密的注意事项

Adobe AIR 1.5 和更高版本

第 649 页的 " 加密数据库的用途 " 部分介绍了需要使用加密数据库的几种情况。显然，不同应用程序的使用情况 （包括以上这些情况和其他情况）具有不同的隐私要求。如何安排加密在应用程序中的用途，对于控制数据库的数据私密程度起着重要作用。例如，如果使用加密数据库来保持个人数据的私密性 （甚至针对同一计算机的其他用户），则每个用户的数据库都需要有自己的加密密钥。为尽可能的安全起见，应用程序可以根据用户输入的密码生成密钥。加密密钥以密码为基础可以确保，即使其他人可以在计算机上模拟用户的帐户，也无法访问数据。换个角度来看隐私问题，假设您希望数据库文件可以由您的应用程序的任何用户读取，但不能由其他应用程序读取。在这种情况下，每个已安装的应用程序副本都需要具有访问共享加密密钥的权限。

可以根据希望应用程序数据所达到的隐私等级来设计应用程序，尤其是用于生成加密密钥的方法。以下列表为各种级别的数据隐私提供了设计建议：

- 若要使任何计算机上有权访问应用程序的任何用户都可以访问数据库，请使用一个密钥，该密钥对于应用程序的所有实例都可用。例如，应用程序首次运行时，可以使用一个安全协议 （如 SSL）从服务器下载共享的加密密钥。然后可以将密钥保存在加密本地存储区中，以供将来使用。作为替代方法，可以按计算机上的每个用户对数据进行加密，然后将数据与远程数据存储区 （如服务器）同步，以使数据可移植。
- 若要使任何计算机上的单个用户都可以访问数据库，请根据用户的保密事项 （如密码）生成加密密钥。尤其是不要使用任何与特定计算机关联的值 （如存储在加密本地存储区中的值）生成密钥。作为替代方法，可以按计算机上的每个用户对数据进行加密，然后将数据与远程数据存储区 （如服务器）同步，以使数据可移植。
- 若要使单个计算机上的单个人可以访问数据库，请根据密码和所生成的 salt 来生成密钥。有关此方法的示例，请参阅第 654 页的 " 示例：生成和使用加密密钥 "。

设计应用程序使用加密数据库时，还有一些安全注意事项务必要牢记，具体如下所示：

- 系统的安全性取决于其最薄弱环节的安全性。如果使用用户输入的密码生成加密密钥，则请考虑对密码施加最小长度和复杂性的限制。只使用基本字符的短密码很快就会被猜中。
- AIR 应用程序的源代码以纯文本形式 （对于 HTML 内容）或易于反编译的二进制格式 （对于 SWF 内容）存储在用户的计算机上。由于源代码可访问，因此有两点要牢记：
  - 切勿在源代码中对加密密钥进行硬编码
  - 始终假设用于生成加密密钥的方法 （如随机字符生成器或特定的哈希算法）很容易就会被攻击者破解
- AIR 数据库加密使用高级加密标准 (AES) 及 Counter with CBC-MAC (CCM) 模式。这种加密密码需要将用户输入的密钥与 salt 值组合在一起才安全。有关此方法的示例，请参阅第 654 页的 " 示例：生成和使用加密密钥 "。
- 如要加密数据库，则数据库引擎所使用的所有磁盘文件与该数据库一起都要进行加密。但是，在事务过程中，数据库引擎会在内存中的缓存中临时保留一些数据，以提高读写性能。驻留在内存中的任何数据都不加密。如果攻击者可以访问 AIR 应用程序所使用的内存 （例如通过使用调试器），则数据库中当前公开且未加密的数据即可供其使用。

示例：生成和使用加密密钥

Adobe AIR 1.5 和更高版本

此示例应用程序介绍生成加密密钥的一种方法。此应用程序旨在为用户的数据提供最高级别的隐私和安全。保障私有数据安全的一个重要原则是：在应用程序每次连接到数据库时都要求用户输入密码。因此，正如此例所示，需要此种保密级别的应用程序在任何时候都不应直接存储数据库加密密钥。

应用程序由两部分组成：生成加密密钥的 ActionScript 类 （EncryptionKeyGenerator 类），以及介绍如何使用该类的基础用户界面。有关完整的源代码，请参阅第 655 页的 " 用于生成和使用加密密钥的完整示例代码 "。

使用 EncryptionKeyGenerator 类获得安全加密密钥

Adobe AIR 1.5 和更高版本

不需要了解 EncryptionKeyGenerator 类的工作方式的详情即可在您的应用程序中使用它。如果您有兴趣详细了解此类是如何为数据库生成加密密钥的，请参阅第 659 页的 " 了解 EncryptionKeyGenerator 类 "。

请按照以下步骤在应用程序中使用 EncryptionKeyGenerator 类：

1. 将 EncryptionKeyGenerator 类作为源代码或编译的 SWC 进行下载。 EncryptionKeyGenerator 类包括在开源 ActionScript 3.0 核心库 (as3corelib) 项目中。可以下载包括源代码和文档的 as3corelib 包。还可以从项目页面下载 SWC 或源代码文件。
2. 将 EncryptionKeyGenerator 类的源代码 （即 as3corelib SWC）放置在应用程序源代码可以找到的地方。
3. 在应用程序源代码中，为 EncryptionKeyGenerator 类添加一条 import 语句。

   ```
   import com.adobe.air.crypto.EncryptionKeyGenerator;
   ```

4. 在创建数据库或打开数据库连接的代码位置之前，添加一段代码，以通过调用 EncryptionKeyGenerator() 构造函数来创建 EncryptionKeyGenerator 实例。

   ```
   var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
   ```

5. 从用户处获取密码：

   ```
   var password:String = passwordInput.text;

   if (!keyGenerator.validateStrongPassword(password))
   {
       // display an error message
       return;
   }
   ```

   EncryptionKeyGenerator 实例使用此密码作为加密密钥的基础（下一步中介绍）。EncryptionKeyGenerator 实例对照特定的强密码验证要求测试该密码。如果验证失败，则发生错误。如示例代码所示，可以通过调用 EncryptionKeyGenerator 对象的 validateStrongPassword() 方法提前检查密码。这样可以确定密码是否符合强密码的最低要求，从而避免出错。

6. 根据密码生成加密密钥：

   ```
   var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
   ```

   getEncryptionKey() 方法生成并返回加密密钥 （16 字节的 ByteArray）。然后即可使用该加密密钥创建新的加密数据库，或打开现有的加密数据库。

   getEncryptionKey() 方法有一个必需的参数，即第 5 步中获取的密码。

   注：为使数据获得最大程度的安全性和保密性，应用程序必须在每次连接到数据库时都要求用户输入密码。请勿直接存储用户的密码或数据库加密密钥。这样做将面临安全风险。正如本例所示，应用程序在创建加密数据库时和以后连接到该数据库时，应该使用相同的技术根据密码派生加密密钥。

getEncryptionKey() 方法还接受第二个（可选）参数，即 overrideSaltELSKey。EncryptionKeyGenerator 创建一个随机值（称为 salt），将其用作加密密钥的一部分。为了能够重新创建加密密钥，salt 值存储在 AIR 应用程序的加密本地存储区 (ELS) 中。默认情况下，EncryptionKeyGenerator 类使用特定的字符串作为 ELS 密钥。虽然可能性不大，但该密钥有可能与应用程序使用的另一个密钥发生冲突。您可能希望指定自己的 ELS 密钥，而不使用默认密钥。在这种情况下，请指定自定义密钥，方法是将它作为第二个 getEncryptionKey() 参数进行传递，如下所示：

```
var customKey:String = "My custom ELS salt key";
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password, customKey);
```

7 创建或打开数据库

通过由 getEncryptionKey() 方法返回的加密密钥，代码可以创建新的加密数据库，或尝试打开现有的加密数据库。在这两种情况下，都使用 SQLConnection 类的 open() 或 openAsync() 方法，如第 650 页的 “ 创建加密数据库 ” 和第 650 页的 “ 连接到加密数据库 ”中所述。

在此示例中，应用程序设计为以异步执行模式打开数据库。代码设置适当的事件侦听器，调用 openAsync() 方法，并传递加密密钥作为最终参数：

```
conn.addEventListener(SQLEvent.OPEN, openHandler);
conn.addEventListener(SQLErrorEvent.ERROR, openError);

conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
```

在侦听器方法中，代码取消了事件侦听器的注册。然后显示状态消息，表明是创建、打开了数据库，还是发生了错误。这些事件处理函数中，最值得注意的部分是 openError() 方法。在该方法中，有一个 if 语句检查数据库是否存在（意味着代码正在尝试连接到现有的数据库），以及错误 ID 是否与常量 EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID 匹配。如果这两个条件都符合，则可能表示用户输入的密码有误。（也有可能表示指定的文件根本不是数据库文件。）以下是用于检查错误 ID 的代码：

```
if (!createNewDB && event.error.errorID == EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
{
    statusMsg.text = "Incorrect password!";
}
else
{
    statusMsg.text = "Error creating or opening database.";
}
```

有关示例事件侦听器的完整代码，请参阅第 655 页的 “ 用于生成和使用加密密钥的完整示例代码 ”。

用于生成和使用加密密钥的完整示例代码

Adobe AIR 1.5 和更高版本

以下是示例应用程序 “ 生成和使用加密密钥 ” 的完整代码。代码由两部分组成。

该示例使用 EncryptionKeyGenerator 类根据密码创建加密密钥。EncryptionKeyGenerator 类包括在开源 ActionScript 3.0 核心库 (as3corelib) 项目中。可以下载包括源代码和文档的 as3corelib 包。还可以从项目页面下载 SWC 或源代码文件。

Flex 示例

应用程序 MXML 文件包含一个简单应用程序的源代码，该应用程序创建加密数据库或打开到加密数据库的连接：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            import com.adobe.air.crypto.EncryptionKeyGenerator;

            private const dbFileName:String = "encryptedDatabase.db";

            private var dbFile:File;
            private var createNewDB:Boolean = true;
            private var conn:SQLConnection;

            // ------- Event handling -------

            private function init():void
            {
                conn = new SQLConnection();
                dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);
                if (dbFile.exists)
                {
                    createNewDB = false;
                    instructions.text = "Enter your database password to open the encrypted database.";
                    openButton.label = "Open Database";
                }
            }

            private function openConnection():void
            {
                var password:String = passwordInput.text;

                var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

                if (password == null || password.length <= 0)
                {
                    statusMsg.text = "Please specify a password.";
                    return;
                }

                if (!keyGenerator.validateStrongPassword(password))
                {
                    statusMsg.text = "The password must be 8-32 characters long. It must contain at least
one lowercase letter, at least one uppercase letter, and at least one number or symbol.";
                    return;
                }

                passwordInput.text = "";
                passwordInput.enabled = false;
                openButton.enabled = false;

                var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

                conn.addEventListener(SQLEvent.OPEN, openHandler);
                conn.addEventListener(SQLErrorEvent.ERROR, openError);

                conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
            }

            private function openHandler(event:SQLEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);
```

```
                statusMsg.setStyle("color", 0x009900);
                if (createNewDB)
                {
                    statusMsg.text = "The encrypted database was created successfully.";
                }
                else
                {
                    statusMsg.text = "The encrypted database was opened successfully.";
                }
            }

            private function openError(event:SQLErrorEvent):void
            {
                conn.removeEventListener(SQLEvent.OPEN, openHandler);
                conn.removeEventListener(SQLErrorEvent.ERROR, openError);

                if (!createNewDB && event.error.errorID ==
EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
                {
                    statusMsg.text = "Incorrect password!";
                }
                else
                {
                    statusMsg.text = "Error creating or opening database.";
                }
            }
        ]]>
    </mx:Script>
    <mx:Text id="instructions" text="Enter a password to create an encrypted database. The next time you
open the application, you will need to re-enter the password to open the database again." width="75%"
height="65"/>
    <mx:HBox>
        <mx:TextInput id="passwordInput" displayAsPassword="true"/>
        <mx:Button id="openButton" label="Create Database" click="openConnection();"/>
    </mx:HBox>
    <mx:Text id="statusMsg" color="#990000" width="75%"/>
</mx:WindowedApplication>
```

Flash Professional 示例

应用程序 FLA 文件包含一个简单应用程序的源代码，该应用程序创建加密数据库或打开到加密数据库的连接。FLA 文件有以下四个放置在舞台上的组件：

实例名称	组件类型	说明
instructions	标签	包含提供给用户的说明
passwordInput	文本输入	用户从中输入密码的输入字段
openButton	按钮	输入密码后用户单击的按钮
statusMsg	标签	显示状态 （成功或失败）消息

在主时间轴的第 1 帧上的关键帧上定义此应用程序的代码。以下是应用程序的代码：

```
import com.adobe.air.crypto.EncryptionKeyGenerator;

const dbFileName:String = "encryptedDatabase.db";

var dbFile:File;
var createNewDB:Boolean = true;
var conn:SQLConnection;

init();

// ------- Event handling -------

function init():void
{
    passwordInput.displayAsPassword = true;
    openButton.addEventListener(MouseEvent.CLICK, openConnection);
    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x990000));

    conn = new SQLConnection();
    dbFile = File.applicationStorageDirectory.resolvePath(dbFileName);

    if (dbFile.exists)
    {
        createNewDB = false;
        instructions.text = "Enter your database password to open the encrypted database.";
        openButton.label = "Open Database";
    }
    else
    {
        instructions.text = "Enter a password to create an encrypted database. The next time you open the
application, you will need to re-enter the password to open the database again.";
        openButton.label = "Create Database";
    }
}

function openConnection(event:MouseEvent):void
{
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();

    var password:String = passwordInput.text;

    if (password == null || password.length <= 0)
    {
        statusMsg.text = "Please specify a password.";
        return;
    }

    if (!keyGenerator.validateStrongPassword(password))
    {
        statusMsg.text = "The password must be 8-32 characters long. It must contain at least one lowercase
letter, at least one uppercase letter, and at least one number or symbol.";
        return;
    }

    passwordInput.text = "";
    passwordInput.enabled = false;
    openButton.enabled = false;

    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);

    conn.addEventListener(SQLEvent.OPEN, openHandler);
    conn.addEventListener(SQLErrorEvent.ERROR, openError);
```

```
    conn.openAsync(dbFile, SQLMode.CREATE, null, false, 1024, encryptionKey);
}

function openHandler(event:SQLEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    statusMsg.setStyle("textFormat", new TextFormat(null, null, 0x009900));
    if (createNewDB)
    {
        statusMsg.text = "The encrypted database was created successfully.";
    }
    else
    {
        statusMsg.text = "The encrypted database was opened successfully.";
    }
}

function openError(event:SQLErrorEvent):void
{
    conn.removeEventListener(SQLEvent.OPEN, openHandler);
    conn.removeEventListener(SQLErrorEvent.ERROR, openError);

    if (!createNewDB && event.error.errorID == EncryptionKeyGenerator.ENCRYPTED_DB_PASSWORD_ERROR_ID)
    {
        statusMsg.text = "Incorrect password!";
    }
    else
    {
        statusMsg.text = "Error creating or opening database.";
    }
}
```

了解 EncryptionKeyGenerator 类

Adobe AIR 1.5 和更高版本

不必了解 EncryptionKeyGenerator 类的内部工作机制，也可以使用它为应用程序数据库创建安全加密密钥。第 654 页的 “使用 EncryptionKeyGenerator 类获得安全加密密钥 ” 中介绍了使用该类的过程。但是，您会发现非常值得了解该类使用的技术。例如，对于需要不同数据保密级别的场合，可能要改编该类或加入它的某些技术。

EncryptionKeyGenerator 类包括在开源 ActionScript 3.0 核心库 (as3corelib) 项目中。可以下载 包括源代码和文档的 as3corelib 包。还可以在项目站点查看源代码，或进行下载以按照介绍进行操作。

代码创建 EncryptionKeyGenerator 实例并调用其 getEncryptionKey() 方法时，将采取若干步骤来确保只有正当的用户才能访问数据。此过程与创建数据库之前，根据用户输入的密码生成加密密钥的过程相同，亦与重新创建加密密钥，以打开数据库的过程相同。

获取并验证强密码

Adobe AIR 1.5 和更高版本

代码调用 getEncryptionKey() 方法时，将传入密码作为参数。密码用作加密密钥的基础。此设计使用只有用户知道的一条信息，从而确保只有知道密码的用户才能访问数据库中的数据。即使攻击者可以访问用户在计算机上的帐户，在不知道密码的情况下也无法进入数据库。为尽可能安全起见，应用程序从不存储密码。

应用程序的代码创建 EncryptionKeyGenerator 实例并调用其 getEncryptionKey() 方法，将用户输入的密码作为参数 （此示例中为 password 变量）传递：

```
var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
var encryptionKey:ByteArray = keyGenerator.getEncryptionKey(password);
```

调用 getEncryptionKey() 方法后， EncryptionKeyGenerator 类所采取的第一步是检查用户输入的密码，以确保其符合密码强度的要求。EncryptionKeyGenerator 类要求密码长度为 8 - 32 个字符。密码必须包含大小写字母的混合形式，而且至少包括一个数字或符号字符。

检查此模式的正则表达式被定义为一个常量，名为 STRONG_PASSWORD_PATTERN：

```
private static const STRONG_PASSWORD_PATTERN:RegExp = /(?=^.{8,32}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/;
```

检查密码的代码位于 EncryptionKeyGenerator 类的 validateStrongPassword() 方法中。代码如下所示：

```
public function vaidateStrongPassword(password:String):Boolean
{
    if (password == null || password.length <= 0)
    {
        return false;
    }

    return STRONG_PASSWORD_PATTERN.test(password))
}
```

在内部，getEncryptionKey() 方法调用 EncryptionKeyGenerator 类的 validateStrongPassword() 方法，并且密码无效时会引发异常。 validateStrongPassword() 方法是一个公共方法，这样应用程序代码无需调用 getEncryptionKey() 方法即可进行密码检查，从而避免导致错误。

将密码扩展到 256 位

Adobe AIR 1.5 和更高版本

在过程的后期，密码的长度必须为 256 位。代码并不要求每个用户输入长度刚好为 256 位 （32 个字符）的密码，而是通过重复密码字符来创建更长的密码。

getEncryptionKey() 方法调用 concatenatePassword() 方法以执行创建长密码的任务。

```
var concatenatedPassword:String = concatenatePassword(password);
```

以下是 concatenatePassword() 方法的代码：

```
private function concatenatePassword(pwd:String):String
{
    var len:int = pwd.length;
    var targetLength:int = 32;

    if (len == targetLength)
    {
        return pwd;
    }

    var repetitions:int = Math.floor(targetLength / len);
    var excess:int = targetLength % len;

    var result:String = "";

    for (var i:uint = 0; i < repetitions; i++)
    {
        result += pwd;
    }

    result += pwd.substr(0, excess);

    return result;
}
```

如果密码长度小于 256 位，则代码将密码与密码自身连接在一起，使其达到 256 位。如果不能刚好达到这一长度，则缩短最后一次重复的内容，以便刚好得到 256 位。

生成或检索 256 位 salt 值

Adobe AIR 1.5 和更高版本

下一步是获得 256 位 salt 值，后面的一个步骤中会将此值与密码组合在一起。 salt 是向用户输入的值添加或与之组合在一起而形成密码的一个随机值。结合使用 salt 和密码确保了即使用户选择真实单词或常用词汇作为密码，系统所使用的 " 密码加 salt" 组合也是一个随机值。这有助于防御字典攻击，攻击者在字典攻击中使用单词列表尝试猜出密码。此外，通过生成 salt 值并将其存储在加密本地存储区中，该值与数据库文件所在计算机上的用户帐户相关联。

如果应用程序是第一次调用 getEncryptionKey() 方法，则代码将创建一个随机的 256 位 salt 值。否则，代码从加密本地存储区加载 salt 值。

salt 存储在名为 salt 的变量中。此代码确定它是否已通过从加密本地存储区加载 salt 而创建了 salt：

```
var salt:ByteArray = EncryptedLocalStore.getItem(saltKey);
if (salt == null)
{
    salt = makeSalt();
    EncryptedLocalStore.setItem(saltKey, salt);
}
```

如果代码要创建新的 salt 值， makeSalt() 方法会生成一个 256 位的随机值。由于该值最终存储在加密本地存储区中，因此生成该值时将其作为 ByteArray 对象。makeSalt() 方法使用 Math.random() 方法随机生成该值。Math.random() 方法无法一次生成 256 位。作为替代，代码使用循环，调用 Math.random() 八次。每次都生成一个 0 到 4294967295 （最大的 uint 值）之间的 uint 值。使用 uint 值是出于方便的目的，因为 uint 刚好为 32 位。通过向 ByteArray 中写入八个 uint 值，即生成一个 256 位值。以下是 makeSalt() 方法的代码：

```
private function makeSalt():ByteArray
{
    var result:ByteArray = new ByteArray;

    for (var i:uint = 0; i < 8; i++)
    {
        result.writeUnsignedInt(Math.round(Math.random() * uint.MAX_VALUE));
    }

    return result;
}
```

代码将 salt 保存到加密本地存储区 (ELS) 或从 ELS 检索 salt 时，代码需要将 salt 保存在其下的 String 密钥。不知道密钥即无法检索 salt 值。在这种情况下，无法在每次重新打开数据库时重新创建加密密钥。默认情况下，EncryptionKeyGenerator 使用预定义的 ELS 密钥，该密钥在常量 SALT_ELS_KEY 中定义。如果不使用默认密钥的话，应用程序代码还可以指定一个 ELS 密钥，供调用 getEncryptionKey() 方法的过程中使用。默认的或由应用程序指定的 salt ELS 密钥存储在名为 saltKey 的变量中。该变量用于调用 EncryptedLocalStore.setItem() 和 EncryptedLocalStore.getItem()，如上所示。

使用 XOR 运算符组合 256 位密码和 salt

Adobe AIR 1.5 和更高版本

代码现在拥有一个 256 位密码和一个 256 位 salt 值。然后，代码使用按位 XOR 运算将 salt 和连接而成的密码组合为一个值。实际上，这种方法创建的 256 位密码由整个可用字符范围内的字符组成。即使实际输入的密码主要由字母数字字符组成也是如此。如此提高随机性的优点在于：无须用户输入长而复杂的密码，即可使可能密码的集合变得非常大。

XOR 运算的结果存储在变量 unhashedKey 中。对这两个值执行按位 XOR 的实际过程发生在 xorBytes() 方法中：

```
var unhashedKey:ByteArray = xorBytes(concatenatedPassword, salt);
```

按位 XOR 运算符 (^) 采用两个 uint 值，并返回一个 uint 值。（uint 值包含 32 位。）作为参数传递给 xorBytes() 方法的输入值为 String（密码）和 ByteArray (salt)。因此，代码使用循环，一次从每个输入提取 32 位，以使用 XOR 运算符进行组合。

```
private function xorBytes(passwordString:String, salt:ByteArray):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 4)
    {
        // ...
    }

    return result;
}
```

在循环中，首先从 passwordString 参数提取 32 位（4 个字节）。提取这些位之后，将在一个包含两部分的过程中将这些位转换为 uint (o1)。首先，charCodeAt() 方法获取每个字符的数值。接下来，使用按位左移运算符 (<<) 将该值移动到 uint 中的适当位置，并将所移动的值添加到 o1。例如，通过使用按位左移运算符 (<<) 将第一个字符 (i) 左移 24 位，并将该值赋给 o1，使第一个字符成为第一个 8 位。通过将第二个字符 (i + 1) 的值左移 16 位，并将结果添加到 o1，使第二个字符成为第二个 8 位。如法炮制，添加第三个和第四个字符的值。

```
        // ...

        // Extract 4 bytes from the password string and convert to a uint
        var o1:uint = passwordString.charCodeAt(i) << 24;
        o1 += passwordString.charCodeAt(i + 1) << 16;
        o1 += passwordString.charCodeAt(i + 2) << 8;
        o1 += passwordString.charCodeAt(i + 3);

        // ...
```

变量 o1 现在包含 passwordString 参数中的 32 位。接下来，通过调用其 readUnsignedInt() 方法，从 salt 参数提取 32 位。这 32 位存储在 uint 变量 o2 中。

```
        // ...

        salt.position = i;
        var o2:uint = salt.readUnsignedInt();

        // ...
```

最后，使用 XOR 运算符将这两个 32 位 (uint) 值组合在一起，并将结果写入名为 result 的 ByteArray 中。

```
        // ...

        var xor:uint = o1 ^ o2;
        result.writeUnsignedInt(xor);
        // ...
```

循环完成后，即返回包含 XOR 结果的 ByteArray。

```
        // ...
    }

    return result;
}
```

对密钥进行哈希处理

Adobe AIR 1.5 和更高版本

将连接而成的密码与 salt 组合在一起后，下一步就是进一步加强对此值的保护，具体而言就是使用 SHA-256 哈希算法对此值进行哈希处理。对值进行哈希处理使攻击者更难以对其进行反向工程。

此时，代码拥有一个名为 unhashedKey 的 ByteArray，其中包含与 salt 组合在一起的连接而成的密码。ActionScript 3.0 核心库 (as3corelib) 项目在 com.adobe.crypto 包中含有一个 SHA256 类。SHA256.hashBytes() 方法对 ByteArray 执行 SHA-256 哈希处理，并返回一个包含 256 位哈希值结果的 String （以十六进制数表示）。 EncryptionKeyGenerator 类使用 SHA256 类对密钥进行哈希处理：

```
var hashedKey:String = SHA256.hashBytes(unhashedKey);
```

从哈希值提取加密密钥

Adobe AIR 1.5 和更高版本

加密密钥必须为刚好 16 个字节 （128 位）长的 ByteArray。 SHA-256 哈希算法的结果的长度始终为 256 位。因此，最后一步是从哈希处理的结果中选择 128 位作为实际的加密密钥。

在 EncryptionKeyGenerator 类中，代码通过调用 generateEncryptionKey() 方法，将密钥缩短为 128 位。代码随后返回该方法的结果作为 getEncryptionKey() 方法的结果：

```
var encryptionKey:ByteArray = generateEncryptionKey(hashedKey);
return encryptionKey;
```

并不一定要使用前 128 位作为加密密钥。可以选择从某任意点开始的一系列位，可以每隔一位选择一位，或使用某些其他方式来选择位。重要的是代码选择 128 个不同的位，并且每次都使用相同的 128 位。

本例中， generateEncryptionKey() 方法使用从第 18 个字节开始的一些位作为加密密钥。如前所述， SHA256 类返回一个包含 256 位哈希值的 String （以十六进制数表示）。单块 128 位的字节数过多，无法一次添加到 ByteArray。因此，代码使用 for 循环，从十六进制的 String 提取字符，将这些字符转换为实际的数字值，并将其添加到 ByteArray。SHA-256 结果 String 的长度为 64 个字符。128 位的范围等于 String 中的 32 个字符，因此每个字符代表 4 位。可以添加到 ByteArray 的最小数据增量为一个字节 （8 位），等于 hash String 中的两个字符。因此，循环的计数范围是 0 到 31 （32 个字符），增量为 2 个字符。

在循环内，代码首先确定当前字符对的起始位置。由于所需的范围从索引位置 17 （第 18 个字节）处的字符开始，因此将当前迭代器的值 (i) 加上 17 后赋给 position 变量。代码使用 String 对象的 substr() 方法提取当前位置的两个字符。将这些字符存储在变量 hex 中。接下来，代码使用 parseInt() 方法将 hex String 转换为十进制整数值。将该值存储在 int 变量 byte 中。最后，代码使用其 writeByte() 方法将 byte 中的值添加到 result ByteArray 中。循环完成时， result ByteArray 包含 16 个字节，并已准备就绪，可以用作数据库加密密钥。

```
private function generateEncryptionKey(hash:String):ByteArray
{
    var result:ByteArray = new ByteArray();

    for (var i:uint = 0; i < 32; i += 2)
    {
        var position:uint = i + 17;
        var hex:String = hash.substr(position, 2);
        var byte:int = parseInt(hex, 16);
        result.writeByte(byte);
    }

    return result;
}
```

使用 SQL 数据库的策略

Adobe AIR 1.0 和更高版本

应用程序可以通过各种方式来访问和使用本地 SQL 数据库。应用程序设计可能随应用程序代码的组织方式、操作执行方式的序列和计时等的不同而不同。所选技术可能影响开发应用程序的难易程度。它们可能影响在将来的更新中修改应用程序的难易程度。它们还可能影响从用户的角度看应用程序性能的高低。

分发预填充的数据库

Adobe AIR 1.0 和更高版本

在应用程序中使用 AIR 本地 SQL 数据库时，应用程序期望一个特定结构的表、列等的数据库。某些应用程序还期望在数据库文件中预填充特定数据。确保数据库具有正确结构的一种方法是，在应用程序代码中创建数据库。应用程序在加载时将检查在特定位置中是否存在其数据库文件。如果该文件不存在，则应用程序将执行一组命令来创建数据库文件，创建数据库结构并用初始数据填充表。

创建数据库及其表的代码常常很复杂。在应用程序的安装生存期内，它通常仅使用一次，但是仍增加了应用程序的大小和复杂性。作为以编程方式创建数据库、结构和数据的一种替代方法，可以随应用程序分发预填充的数据库。要分发预定义的数据库，请将数据库文件包括在应用程序的 AIR 包中。

与 AIR 包中包括的所有文件一样，捆绑的数据库文件安装在应用程序目录 （由 File.applicationDirectory 属性表示的目录）中。但是，该目录中的文件是只读的。将 AIR 包中的文件用作 “ 模板 ” 数据库。用户第一次运行该应用程序时，会将原始数据库文件复制到用户的第 576 页的 “ 指向应用程序存储目录 ” （或其他位置），并在应用程序中使用该数据库。

使用本地 SQL 数据库的最佳做法

Adobe AIR 1.0 和更高版本

下面列出了一组建议的方法，在使用本地 SQL 数据库时，可以通过这些方法提高应用程序的性能、安全性和易维护性。

预创建数据库连接

Adobe AIR 1.0 和更高版本

尽管应用程序在首次加载时不执行任何语句，但是在运行语句时，提前 （如在应用程序初始启动之后）实例化 SQLConnection 对象并调用其 open() 或 openAsync() 方法可以避免延迟。请参阅第 623 页的 " 连接到数据库 "。

重用数据库连接

Adobe AIR 1.0 和更高版本

如果在应用程序的整个执行时间内访问某个数据库，请保存对 SQLConnection 实例的引用，并在整个应用程序中重用它，而不是先关闭再重新打开连接。请参阅第 623 页的 " 连接到数据库 "。

推荐使用异步执行模式

Adobe AIR 1.0 和更高版本

编写数据访问代码时，可能很想同步执行操作而不是异步执行，因为使用同步操作通常需要更短的代码，并且代码的复杂性更低。但是，如第 645 页的 " 使用同步和异步数据库操作 " 中所述，同步操作可产生对用户而言很明显的性能影响，并损害用户对应用程序的体验。单个操作所用的时间随操作、尤其是它所涉及的数据量的不同而不同。例如，仅向数据库中添加一行的 SQL INSERT 语句所用的时间要比检索成千上万行数据的 SELECT 语句所用的时间少。但是，使用同步执行来执行多个操作时，这些操作通常串在一起。即使每个操作所用的时间非常短，但是在所有同步操作完成之前会冻结应用程序。因此，串在一起的多个操作的累积时间可能足以停止应用程序。

将异步操作用作一种标准方法，尤其是对于涉及大量行的操作。有一种技术可拆分大型 SELECT 语句结果集的处理，如第 636 页的 " 检索部分 SELECT 结果 " 所述。但是，此技术只能在异步执行模式下使用。只有在使用异步编程无法实现某些功能时，已考虑到应用程序的用户所要面临的性能折衷时，以及已测试应用程序以便了解对应用程序性能的影响程度时，才应使用同步操作。使用异步执行可能涉及更复杂的编码。但是，请记住只需编写一次代码，但是应用程序用户必须重复使用它 （或快或慢）。

在许多情况下，通过对要执行的每个 SQL 语句使用单独的 SQLStatement 实例，可以同时将多个 SQL 操作排队，这将使异步代码在代码编写方式上与同步代码类似。有关详细信息，请参阅第 648 页的 " 了解异步执行模式 "。

使用单独的 SQL 语句，且不更改 SQLStatement 的 text 属性

Adobe AIR 1.0 和更高版本

对于在应用程序中多次执行的任何 SQL 语句，为每个 SQL 语句创建单独的 SQLStatement 实例。 SQL 命令在每次执行时都使用该 SQLStatement 实例。例如，假定您要生成一个应用程序，它包括四个多次执行的不同 SQL 操作。在此情况下，创建四个单独的 SQLStatement 实例，并调用每个语句的 execute() 方法来运行它。避免对所有 SQL 语句使用单个 SQLStatement 实例，每次执行语句之前都重新定义其 text 属性。

使用语句参数

Adobe AIR 1.0 和更高版本

使用 SQLStatement 参数 — 从不将用户输入连接到语句文本中。使用参数会使应用程序更安全，因为这样可消除 SQL 注入攻击的可能性。这样就有可能在查询中使用对象 （而不是仅使用 SQL 字面值）。这样还会提高语句的运行效率，因为可以重用语句，每次执行它们时无需将其重新编译。有关详细信息，请参阅第 627 页的 " 在语句中使用参数 "。

第 41 章：使用字节数组

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ByteArray 类允许读取和写入二进制数据流，该数据流本质上是字节数组。该类提供了一种访问最基本的数据的方法。因为计算机数据由字节 （即包含 8 位的组）组成，因此能够以字节为单位读取数据意味着您可以访问那些不存在类和访问方法的数据。 ByteArray 类允许您在字节级分析任何数据流，从位图到通过网络的数据流。

利用 writeObject() 方法可以将具有序列化 Action Message Format (AMF) 格式的对象写入 ByteArray，而利用 readObject() 方法则可以从 ByteArray 中将序列化对象读取到原始数据类型的变量中。可以将显示对象以外的任何对象序列化，显示对象是可以放在显示列表中的那些对象。如果自定义类可用于运行时，也可以将序列化对象重新指定给自定义类实例。将一个对象转换为 AMF 格式之后，就可以通过网络连接有效传输该对象或将该对象保存到文件中。

此处描述的 Adobe® AIR® 应用程序示例将读取一个 .zip 文件，以该文件为例说明处理字节流、提取 .zip 文件包含的文件列表并将其写入桌面的过程。

更多帮助主题

flash.utils.ByteArray

flash.utils.IExternalizable

Action Message Format 规范

读取并写入 ByteArray

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ByteArray 类在 flash.utils 包中。若要使用 ActionScript 3.0 创建一个 ByteArray 对象，请导入 ByteArray 类并调用构造函数，如下例所示：

```
import flash.utils.ByteArray;
var stream:ByteArray = new ByteArray();
```

ByteArray 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

任何有意义的数据流均将以某种格式组织起来，以便于您分析并找到所需信息。例如，简单的员工文件中的记录可能包括 ID 号、姓名、地址、电话号码等等。 MP3 音频文件包含用于标识所下载文件的标题、作者、专辑、发行日期和流派的 ID3 标签。通过格式您可以知道数据在数据流中的预期顺序。这样，您就可以采用智能方式读取字节流。

ByteArray 类包括几种方法，可以使数据流的读写更加容易。其中几种方法包括 readBytes() 和 writeBytes()、 readInt() 和 writeInt()、readFloat() 和 writeFloat()、readObject() 和 writeObject()、readUTFBytes() 和 writeUTFBytes()。利用以上方法可以将数据从数据流读入特定数据类型的变量，也可以从特定数据类型的变量直接写入二进制数据流。

例如，以下代码读取一个简单的由字符串和浮点数组成的数组并将每个元素写入 ByteArray。此数组结构允许代码调用相应的 ByteArray 方法 （writeUTFBytes() 和 writeFloat()）来写入数据。重复的数据模式使循环读取该数组成为可能。

```
 // The following example reads a simple Array (groceries), made up of strings
// and floating-point numbers, and writes it to a ByteArray.

import flash.utils.ByteArray;

// define the grocery list Array
var groceries:Array = ["milk", 4.50, "soup", 1.79, "eggs", 3.19, "bread" , 2.35]
// define the ByteArray
var bytes:ByteArray = new ByteArray();
// for each item in the array
for (var i:int = 0; i < groceries.length; i++) {
        bytes.writeUTFBytes(groceries[i++]); //write the string and position to the next item
        bytes.writeFloat(groceries[i]);// write the float
        trace("bytes.position is: " + bytes.position);//display the position in ByteArray
}
trace("bytes length is: " +  bytes.length);// display the length
```

position 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

position 属性存储指针的当前位置，该指针在读写过程中指向 ByteArray。position 属性的初始值为 0，如以下代码中所示：

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
```

当您读取或写入 ByteArray 时，您使用的方法将更新 position 属性以指向紧随上次所读取或写入字节后的位置。例如，以下代码将一个字符串写入 ByteArray，随后 position 属性将指向 ByteArray 中该字符串后面紧邻的字节：

```
var bytes:ByteArray = new ByteArray();
trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
```

同样，读取操作会使 position 属性值按照读取的字节数而相应增加。

```
var bytes:ByteArray = new ByteArray();

trace("bytes.position is initially: " + bytes.position); // 0
bytes.writeUTFBytes("Hello World!");
trace("bytes.position is now: " + bytes.position);// 12
bytes.position = 0;
trace("The first 6 bytes are: " + (bytes.readUTFBytes(6)));//Hello
trace("And the next 6 bytes are: " + (bytes.readUTFBytes(6)));// World!
```

请注意，您可以将 position 属性设置为 ByteArray 中的一个特定位置从而基于该偏移量进行读取或写入。

bytesAvailable 和 length 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

length 和 bytesAvailable 属性分别指示 ByteArray 的长度为多少以及从当前位置到结尾处还剩多少字节。以下示例说明了您可以通过什么方式使用这些属性。该示例将一个文本字符串写入 ByteArray 然后一次从 ByteArray 中读取一个字节，直到遇到字符 “a” 或到达结尾 (bytesAvailable <= 0)。

```
var bytes:ByteArray = new ByteArray();
var text:String = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus etc.";

bytes.writeUTFBytes(text); // write the text to the ByteArray
trace("The length of the ByteArray is: " + bytes.length);// 70
bytes.position = 0; // reset position
while (bytes.bytesAvailable > 0 && (bytes.readUTFBytes(1) != 'a')) {
    //read to letter a or end of bytes
}
if (bytes.position < bytes.bytesAvailable) {
    trace("Found the letter a; position is: " + bytes.position); // 23
    trace("and the number of bytes available is: " + bytes.bytesAvailable);// 47
}
```

endian 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在存储多字节数字 （即需要超过 1 个字节的内存来存储的数字），各种计算机可能彼此有所差异。例如，一个整数可能需要占用 4 个字节，即 32 位内存。某些计算机首先将数字中的最高有效字节存储在最低的内存地址，而其他计算机则首先存储最低有效字节。计算机的这一属性 （即字节顺序属性）被称为 big endian （最高有效字节位于最前）或 little endian （最低有效字节位于最前）。例如，数字 0x31323334 对于 big endian 和 little endian 字节顺序将分别存储为以下形式，其中 a0 代表 4 个字节的最低内存地址而 a3 代表最高内存地址：

Big Endian	Big Endian	Big Endian	Big Endian
a0	a1	a2	a3
31	32	33	34

Little Endian	Little Endian	Little Endian	Little Endian
a0	a1	a2	a3
34	33	32	31

利用 ByteArray 类的 endian 属性可以为要处理的多字节数字表示此字节顺序。该属性可接受的值为 "bigEndian" 或 "littleEndian"，并且 Endian 类定义了常量 BIG_ENDIAN 和 LITTLE_ENDIAN，从而通过这些字符串设置 endian 属性。

compress() 和 uncompress() 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

利用 compress() 方法可以根据指定为参数的压缩算法压缩 ByteArray。利用 uncompress() 方法可以根据压缩算法对压缩的 ByteArray 进行解压缩。调用 compress() 和 uncompress() 之后，字节数组的长度将设置为新的长度并且 position 属性将设置为结尾。

CompressionAlgorithm 类定义了可用来指定压缩算法的常量。 ByteArray 类支持 deflate （仅限 AIR）、 zlib 和 lzma 算法。http://www.ietf.org/rfc/rfc1950.txt 中介绍了 zlib 压缩的数据格式。lzma 算法是为 Flash Player 11.4 和 AIR 3.4 添加的。 http://www.7-zip.org/7z.html 中介绍了该算法。

这种 deflate 压缩算法用于多种压缩格式，如 zlib、 gzip 及一些 zip 实现等。 http://www.ietf.org/rfc/rfc1951.txt 中介绍了 deflate 压缩算法。

以下示例使用 lzma 算法对称为 bytes 的 ByteArray 进行压缩：

```
bytes.compress(CompressionAlgorithm.LZMA);
```

以下示例使用 deflate 算法对压缩的 ByteArray 进行解压缩：

```
bytes.uncompress(CompressionAlgorithm.LZMA);
```

读取和写入对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

readObject() 和 writeObject() 方法可从 ByteArray 中读取并向其写入以序列化 Action Message Format (AMF) 格式编码的对象。 AMF 是 Adobe 创建并由各种 ActionScript 3.0 类使用的专有消息协议，这些类包括 Netstream、 NetConnection、NetStream、 LocalConnection 和 SharedObject。

单字节类型标记说明了编码数据遵循的类型。 AMF 使用以下 13 种数据类型：

```
value-type = undefined-marker | null-marker | false-marker | true-marker | integer-type |
    double-type | string-type | xml-doc-type | date-type | array-type | object-type |
    xml-type | byte-array-type
```

编码数据将遵循类型标记，除非标记表示一个可能的值（例如 null、true 或 false），在这种情况下将不对任何数据进行编码。

存在两种版本的 AMF: AMF0 和 AMF3。AMF 0 通过引用支持发送复杂对象并允许端点还原对象关系。AMF 3 通过以下方式对 AMF 0 进行了改进：通过引用发送对象 traits 和字符串，除对象引用外，还支持 ActionScript 3.0 中引入的新的数据类型。ByteArray.objectEcoding 属性指定了用于对对象数据进行编码的 AMF 版本。flash.net.ObjectEncoding 类定义了用于指定 AMF 版本的常量：ObjectEncoding.AMF0 和 ObjectEncoding.AMF3。

以下示例调用 writeObject() 将 XML 对象写入 ByteArray 中，随后使用 Deflate 算法压缩 ByteArray 并将其写入桌面上的 order 文件中。该示例完成时将使用标签在 AIR 窗口中显示消息 “Wrote order file to desktop!”

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;
public class WriteObjectExample extends Sprite
{
    public function WriteObjectExample()
    {
        var bytes:ByteArray = new ByteArray();
        var myLabel:TextField = new TextField();
        myLabel.x = 150;
        myLabel.y = 150;
        myLabel.width = 200;
        addChild(myLabel);

        var myXML:XML =
            <order>
                <item id='1'>
                    <menuName>burger</menuName>
                    <price>3.95</price>
                </item>
                <item id='2'>
                    <menuName>fries</menuName>
                    <price>1.45</price>
                </item>
            </order>;

        // Write XML object to ByteArray
```

```
        bytes.writeObject(myXML);
        bytes.position = 0;//reset position to beginning
        bytes.compress(CompressionAlgorithm.DEFLATE);// compress ByteArray
        writeBytesToFile("order.xml", bytes);
        myLabel.text = "Wrote order file to desktop!";
    }

    private function writeBytesToFile(fileName:String, data:ByteArray):void
    {
        var outFile:File = File.desktopDirectory; // dest folder is desktop
        outFile = outFile.resolvePath(fileName);  // name of file to write
        var outStream:FileStream = new FileStream();
        // open output file stream in WRITE mode
        outStream.open(outFile, FileMode.WRITE);
        // write out the file
        outStream.writeBytes(data, 0, data.length);
        // close it
        outStream.close();
    }
}
```

readObject() 方法从 ByteArray 中读取序列化 AMF 格式的对象并将其存储到指定类型的对象中。以下示例将桌面中的 order 文件读入 ByteArray (inBytes)，将其解压缩并调用 readObject() 将其存储在 XML 对象 orderXML 中。该示例使用 for each() 循环构造将每个节点添加到文本区域以便显示。该示例还显示了 objectEncoding 属性的值以及 order 文件内容的标头。

```
import flash.filesystem.*;
import flash.display.Sprite;
import flash.display.TextField;
import flash.utils.ByteArray;

public class ReadObjectExample extends Sprite
{
    public function ReadObjectExample()
    {
        var inBytes:ByteArray = new ByteArray();
        // define text area for displaying XML content
        var myTxt:TextField = new TextField();
        myTxt.width = 550;
        myTxt.height = 400;
        addChild(myTxt);
        //display objectEncoding and file heading
        myTxt.text = "Object encoding is: " + inBytes.objectEncoding + "\n\n" + "order file: \n\n";
        readFileIntoByteArray("order", inBytes);

        inBytes.position = 0; // reset position to beginning
        inBytes.uncompress(CompressionAlgorithm.DEFLATE);
        inBytes.position = 0;//reset position to beginning
        // read XML Object
        var orderXML:XML = inBytes.readObject();
```

```
        // for each node in orderXML
        for each (var child:XML in orderXML)
        {
            // append child node to text area
            myTxt.text += child + "\n";
        }
    }

    // read specified file into byte array
    private function readFileIntoByteArray(fileName:String, data:ByteArray):void
    {
        var inFile:File = File.desktopDirectory; // source folder is desktop
        inFile = inFile.resolvePath(fileName);  // name of file to read
        var inStream:FileStream = new FileStream();
        inStream.open(inFile, FileMode.READ);
        inStream.readBytes(data);
        inStream.close();
    }
}
```

ByteArray 示例：读取 .zip 文件

Adobe AIR 1.0 和更高版本

该示例演示了如何读取包含若干不同类型文件的简单 .zip 文件。读取过程如下：从每个文件的元数据中提取相关数据，将每个文件解压缩至 ByteArray 并将该文件写入桌面。

.zip 文件的一般结构基于 PKWARE Inc. 的规范 （此规范位于 http://www.pkware.com/documents/casestudies/APPNOTE.TXT）。首先是 .zip 归档文件中第一个文件的文件标头和文件数据，接下来依次是其余各文件的文件标头及文件数据。（文件标头的结构将在后面介绍。）接下来， .zip 文件有可能包括数据描述符记录 （通常是在内存中创建输出 zip 文件而不是将其保存到磁盘的情况下包括该记录）。接下来是若干其他可选元素：归档解密标头、归档额外数据记录、中央目录结构、中央目录记录的 Zip64 结尾、中央目录定位器的 Zip64 结尾和中央目录记录的结尾。

本示例中所编写的代码仅用于分析不包含文件夹且不需要数据描述符记录的 zip 文件。它将忽略最后一个文件的数据后面的所有信息。

每个文件的文件标头的格式如下：

文件标头签名	4 字节
所需版本	2 字节
一般用途位标记	2 字节
压缩方法	2 字节 (8=DEFLATE; 0=UNCOMPRESSED)
文件的最后修改时间	2 字节
文件的最后修改日期	2 字节
crc-32	4 字节
压缩后的大小	4 字节
解压缩后的大小	4 字节

文件名长度	2 字节
额外字段长度	2 字节
文件名	变量
额外字段	变量

文件标头的后面是实际的文件数据，既可以是压缩后的也可以是解压缩后的文件数据，具体取决于压缩方法标志。如果文件数据为解压缩后的数据，则此标志为 0；如果该数据为使用 DEFLATE 算法压缩的数据，则为 8；如果采用的是其他压缩算法，则为其他值。

该示例的用户界面由一个标签和一个文本区域 (taFiles) 组成。该应用程序将它在 zip 文件中遇到的各文件的以下信息写入文本区域：文件名、压缩后的大小和解压缩后的大小。以下 MXML 文档为应用程序的 Flex 版本定义用户界面：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
creationComplete="init();">
    <mx:Script>
        <![CDATA[
            // The application code goes here
        ]]>
    </mx:Script>
    <mx:Form>
        <mx:FormItem label="Output">
            <mx:TextArea id="taFiles" width="320" height="150"/>
        </mx:FormItem>
    </mx:Form>
</mx:WindowedApplication>
```

程序开头将执行以下任务：

- 导入所需的类

```
import flash.filesystem.*;
import flash.utils.ByteArray;
import flash.events.Event;
```

- 定义用户界面对于 Flash

```
import fl.controls.*;

//requires TextArea and Label components in the Library
var taFiles = new TextArea();
var output = new Label();
taFiles.setSize(320, 150);
taFiles.move(10, 30);
output.move(10, 10);
output.width = 150;
output.text = "Contents of HelloAir.zip";
addChild(taFiles);
addChild(output);
```

- 定义 bytes ByteArray

```
var bytes:ByteArray = new ByteArray();
```

- 定义用来存储文件标头中元数据的变量

```
// variables for reading fixed portion of file header
var fileName:String = new String();
var flNameLength:uint;
var xfldLength:uint;
var offset:uint;
var compSize:uint;
var uncompSize:uint;
var compMethod:int;
var signature:int;
```

- 定义用来表示 .zip 文件的 File (zfile) 和 FileStream (zStream) 对象，并指定将从中提取文件的 .zip 文件的位置（桌面目录下名为 “HelloAIR.zip” 的文件）。

```
// File variables for accessing .zip file
var zfile:File = File.desktopDirectory.resolvePath("HelloAIR.zip");
var zStream:FileStream = new FileStream();
```

在 Flex 中，程序代码从 init() 方法开始，该方法将作为根 mx:WindowedApplication 标签的 creationComplete 处理函数调用。

```
// for Flex
private function init():void
{
```

该程序将首先以 READ 模式打开 .zip 文件。

```
zStream.open(zfile, FileMode.READ);
```

然后将 bytes 的 endian 属性设置为 LITTLE_ENDIAN，以指示数字字段的字节顺序为最低有效字节位于最前。

```
bytes.endian = Endian.LITTLE_ENDIAN;
```

接下来， while() 语句开始了一个循环，直到文件流中的当前位置大于或等于文件大小时才停止循环。

```
while (zStream.position < zfile.size)
{
```

循环中的第一个语句将文件流的前 30 个字节读入 ByteArray bytes 中。前 30 个字节组成了第一个文件标头的固定大小部分。

```
    // read fixed metadata portion of local file header
    zStream.readBytes(bytes, 0, 30);
```

接下来，代码将从这 30 字节标头最前面的字节中读取一个整数 (signature)。 ZIP 格式定义指定每个文件标头的签名为十六进制值 0x04034b50 ；如果签名不同，则表明代码已移出 zip 文件的文件部分且再没有可提取的文件。在此情况下，代码将立即退出 while 循环而不是等待到达字节数组的结尾。

```
    bytes.position = 0;
    signature = bytes.readInt();
    // if no longer reading data files, quit
    if (signature != 0x04034b50)
    {
        break;
    }
```

代码的下一部分将在偏移量为 8 处读取标头字节并将值存储在变量 compMethod 中。该字节包含指示压缩此文件时所用压缩方法的值。允许使用多种压缩方法，但实际上几乎所有 .zip 文件均使用 DEFLATE 压缩算法。如果当前文件以 DEFLATE 压缩方式进行压缩，则 compMethod 为 8 ；如果文件未压缩，则 compMethod 为 0。

```
    bytes.position = 8;
    compMethod = bytes.readByte();  // store compression method (8 == Deflate)
```

位于前 30 个字节之后的部分是标头的可变长度部分，包含了文件名并可能包含额外字段。变量 offset 用于存储此部分的大小。该大小的计算方式为将文件名长度和额外字段长度相加，这两个长度可分别从标头中偏移量为 26 和 28 的位置读取。

```
        offset = 0;// stores length of variable portion of metadata
        bytes.position = 26;  // offset to file name length
        flNameLength = bytes.readShort();// store file name
        offset += flNameLength; // add length of file name
        bytes.position = 28;// offset to extra field length
        xfldLength = bytes.readShort();
        offset += xfldLength;// add length of extra field
```

接下来程序将读取文件标头的可变长度部分，以将该部分的字节数存储在 offset 变量中。

```
        // read variable length bytes between fixed-length header and compressed file data
        zStream.readBytes(bytes, 30, offset);
```

程序将从标头的可变长度部分中读取文件名并在文本区域中显示它，同时显示文件压缩后 （已压缩）及解压缩后 （原始）大小。

```
// Flash version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.appendText(fileName + "\n"); // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.appendText("\tCompressed size is: " + compSize + '\n');
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.appendText("\tUncompressed size is: " + uncompSize + '\n');


// Flex version
        bytes.position = 30;
        fileName = bytes.readUTFBytes(flNameLength); // read file name
        taFiles.text += fileName + "\n"; // write file name to text area
        bytes.position = 18;
        compSize = bytes.readUnsignedInt();  // store size of compressed portion
        taFiles.text += "\tCompressed size is: " + compSize + '\n';
        bytes.position = 22;  // offset to uncompressed size
        uncompSize = bytes.readUnsignedInt();  // store uncompressed size
        taFiles.text += "\tUncompressed size is: " + uncompSize + '\n';
```

该示例从文件流中将文件的其余部分按照压缩后大小所指定的长度读入到 bytes 中，同时覆盖前 30 字节中的文件标头。即使文件并未压缩，压缩后的大小也是精确的，因为在此情况下，压缩后的大小将等于文件未压缩时的大小。

```
    // read compressed file to offset 0 of bytes; for uncompressed files
    // the compressed and uncompressed size is the same
    if (compSize == 0) continue;
    zStream.readBytes(bytes, 0, compSize);
```

接下来，示例将对压缩的文件进行解压缩并调用 outfile() 函数将文件写入输出文件流。它将向 outfile() 传递文件名和包含文件数据的字节数组。

```
        if (compMethod == 8) // if file is compressed, uncompress
        {
            bytes.uncompress(CompressionAlgorithm.DEFLATE);
        }
        outFile(fileName, bytes);   // call outFile() to write out the file
```

在之前提到的范例中， bytes.uncompress(CompressionAlgorithm.DEFLATE) 仅在 AIR 应用程序中有效。要为 AIR 和 Flash Player 将压缩的数据解压，可调用 ByteArray 的 inflate() 函数。

右括号指示 while 循环、init() 方法以及 Flex 应用程序代码结束，不过 outFile() 方法除外。执行过程再次返回至 while 循环的开始处并继续处理 .zip 文件中其余的字节：提取另一个文件，如果最后一个文件已处理完毕，则终止该 .zip 文件处理。

```
    } // end of while loop
} // for Flex version, end of init() method and application
```

outfile() 函数将以 WRITE 模式打开桌面上的输出文件，为其指定 filename 参数提供的名称。然后该函数将把文件数据从 data 参数中写入输出文件流 (outStream) 并关闭该文件。

```
// Flash version
function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // destination folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

```
private function outFile(fileName:String, data:ByteArray):void
{
    var outFile:File = File.desktopDirectory; // dest folder is desktop
    outFile = outFile.resolvePath(fileName);  // name of file to write
    var outStream:FileStream = new FileStream();
    // open output file stream in WRITE mode
    outStream.open(outFile, FileMode.WRITE);
    // write out the file
    outStream.writeBytes(data, 0, data.length);
    // close it
    outStream.close();
}
```

第 42 章：网络和通信基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 Flash Player 或 AIR 中构建应用程序时，经常需要访问应用程序之外的资源。例如，您可能会向 Internet Web 服务器发送图像请求并因而获得图像数据。或者，您可能使用应用程序服务器的套接字连接反复发送序列化对象。 Flash Player 和 AIR API 提供了几个类，使您的应用程序可以参与这种交换。这些 API 支持基于 IP 的网络使用 UDP、TCP、HTTP、RTMP 和 RTMFP 等协议。

下列类可以用于通过网络发送和接收数据：

类	支持的数据格式	协议	说明
Loader	SWF、 PNG、 JPEG、 GIF	HTTP、HTTPS	加载支持的数据类型并将数据转换为显示对象。 请参阅第 164 页的 " 动态加载显示内容 "。
URLLoader	任意格式 （文本、 XML、 二进制等）	HTTP、HTTPS	加载任意格式的数据。您的应用程序负责解释数据。 请参阅第 699 页的 " 使用 URLLoader 类 "
FileReference	Any	HTTP	上载和下载文件。 请参阅第 559 页的 " 使用 FileReference 类 "
NetConnection	视频、音频、ActionScript Message Format (AMF)	HTTP、HTTPS、RTMP、RTMFP	连接到视频、音频和远程对象流。 请参阅第 401 页的 " 使用视频 "。
Sound	音频	HTTP	加载并播放支持的音频格式。 请参阅第 375 页的 " 加载外部声音文件 "。
XMLSocket	XML	TCP	与 XMLSocket 服务器交换 XML 消息。 请参阅 第 688 页的 "XML 套接字 "。
Socket	Any	TCP	连接到 TCP 套接字服务器。 请参阅第 683 页的 " 二进制客户端套接字 "。
SecureSocket (AIR)	Any	带有 SSLv3 或 TLSv1 的 TCP	连接到要求 SSL 或 TLS 安全的 TCP 套接字服务器。 请参阅第 684 页的 " 安全客户端套接字 (AIR)"。
ServerSocket (AIR)	Any	TCP	用作传入 TCP 套接字连接的服务器。 请参阅第 691 页的 " 服务器套接字 "。
DatagramSocket (AIR)	Any	UDP	发送和接收 UDP 包。 请参阅 第 693 页的 "UDP 套接字 (AIR)"

通常，在创建 Web 应用程序时，存储有关用户的应用程序状态的永久信息很有帮助。 HTML 页和应用程序通常使用 Cookie 来实现此目的。在 Flash Player 中，您可以将 SharedObject 类用于相同的目的。请参阅第 602 页的 " 共享对象 "。（SharedObject 类可以在 AIR 应用程序中使用，但在仅将数据保存到常规文件时的限制更少。）

如果 Flash Player 或 AIR 应用程序需要与相同计算机上的其他 Flash Player 或 AIR 应用程序进行通信，则可以使用 LocalConnection 类。例如，相同网页上的两个或更多 SWF 可以彼此通信。同样，在网页上运行的 SWF 可以与 AIR 应用程序通信。请参阅第 712 页的 " 与其他 Flash Player 和 AIR 实例通信 "。

如果需要与本地计算机上其他非 SWF 进程通信，则可以使用 AIR 2 中添加的 NativeProcess 类。 NativeProcess 类可以使 AIR 应用程序启动并与其他应用程序通信。请参阅第 717 页的 " 与 AIR 中的本机进程通信 "。

如果需要有关运行 AIR 应用程序的计算机的网络环境的信息，可以使用下列类：

- NetworkInfo — 提供有关可用网络接口的信息，例如计算机的 IP 地址。请参阅第 677 页的 " 网络接口 "。

- DNSResolver — 使您可以查找 DNS 记录。请参阅第 681 页的 " 域名系统 (DNS) 记录 "。
- ServiceMonitor — 使您可以监控服务器的可用性。请参阅第 679 页的 " 服务监控 "。
- URLMonitor — 使您可以监控特定 URL 中的资源的可用性。请参阅 第 680 页的 "HTTP 监控 "。
- SocketMonitor 和 SecureSocketMonitor — 使您可以监控套接字中的资源的可用性。请参阅第 680 页的 " 套接字监控 "。

重要概念和术语

以下参考列表包含进行网络和通信代码编程时会遇到的重要术语：

外部数据 在应用程序外部以某些形式存储的数据，需要时可加载到应用程序中。可以将此数据存储在直接加载的文件中，也可存储在数据库中，还可采用可通过调用服务器上所运行的脚本或程序进行检索的其他形式存储。

URL 编码变量 URL 编码格式提供了一种在单个文本字符串中表示多个变量 （变量名和值对）的方法。各变量采用 name=value 格式书写。各个变量 （即各个名称 - 值对）之间用 & 符隔开，如下所示：variable1=value1&variable2=value2。这样，便可以将不限数量的变量作为一条消息进行发送。

MIME 类型 用于在 Internet 通信中标识给定文件类型的标准代码。任何给定文件类型都具有用于对其进行标识的特定代码。发送文件或消息时，计算机 （如 Web 服务器或用户的 Flash Player 或 AIR 实例）将指定要发送的文件类型。

HTTP 超文本传输协议，这是一种标准格式，用于传送通过 Internet 发送的网页和其他各种类型的内容。

请求方法 应用程序 （例如 AIR 应用程序或 Web 浏览器）将消息 （称为 HTTP 请求）发送到 Web 服务器时，发送的任何数据都可以使用以下两种方法之一嵌入到请求中 — 这两种请求方法是 GET 和 POST。在服务器端，接收请求的程序需要查看相应的请求部分以查找数据，因此用于从您的应用程序发送数据的请求方法应与用于在服务器上读取该数据的请求方法匹配。

套接字连接 用于在两台计算机之间进行通信的永久连接。

上载 将文件发送到另一台计算机。

下载 从另一台计算机检索文件。

网络接口

Adobe AIR 2 和更高版本

可以使用 NetworkInfo 对象了解您的应用程序可以使用的硬件和软件网络接口。 NetworkInfo 对象是 singleton 对象，无需您创建。您可以使用静态类属性 networkInfo 访问单个 NetworkInfo 对象。当某一个可用接口更改时， NetworkInfo 对象还将调度 networkChange 事件。

调用 findInterfaces() 方法可以获取 NetworkInterface 对象列表。列表中的每个 NetworkInterface 对象描述一个可用接口。NetworkInterface 对象提供了 IP 地址、硬件地址、最大传输单位以及此接口是否处于活动状态等信息。

以下代码示例跟踪客户端计算机上每个接口的 NetworkInterface 属性：

```
package {
import flash.display.Sprite;
import flash.net.InterfaceAddress;
import flash.net.NetworkInfo;
import flash.net.NetworkInterface;

public class NetworkInformationExample extends Sprite
{
    public function NetworkInformationExample()
    {
        var networkInfo:NetworkInfo = NetworkInfo.networkInfo;
        var interfaces:Vector.<NetworkInterface> = networkInfo.findInterfaces();

        if( interfaces != null )
        {
            trace( "Interface count: " + interfaces.length );
            for each ( var interfaceObj:NetworkInterface in interfaces )
            {
                trace( "\nname: " + interfaceObj.name );
                trace( "display name: " + interfaceObj.displayName );
                trace( "mtu: "   + interfaceObj.mtu );
                trace( "active?: " + interfaceObj.active );
                trace( "parent interface: " + interfaceObj.parent );
                trace( "hardware address: " + interfaceObj.hardwareAddress );
                if( interfaceObj.subInterfaces != null )
                {
                    trace( "# subinterfaces: " + interfaceObj.subInterfaces.length );
                }
                trace("# addresses: " + interfaceObj.addresses.length );
                for each ( var address:InterfaceAddress in interfaceObj.addresses )
                {
                    trace( "  type: "            + address.ipVersion );
                    trace( "  address: " + address.address );
                    trace( "  broadcast: " + address.broadcast );
                    trace( "  prefix length: " + address.prefixLength );
                }
            }
        }
    }
}
}
```

有关详细信息，请参阅：

- NetworkInfo
- NetworkInterface
- InterfaceAddress
- Flexpert：使用 Flex 4.5 检测网络连接类型

网络连接更改

Adobe AIR 1.0 和更高版本

AIR 应用程序可以在具有不确定且不断更改的网络连接的环境中运行。为了有助于应用程序管理到在线资源的连接，每当网络连接变为可用或不可用时 Adobe AIR 都会发送一个网络更改事件。NetworkInfo 对象和应用程序的 NativeApplication 对象均会调度 networkChange 事件。为了响应该事件，可添加一个侦听器：

```
NetworkInfo.networkInfo.addEventListener(Event.NETWORK_CHANGE, onNetworkChange);
```

并定义一个事件处理函数：

```
function onNetworkChange(event:Event)
{
    //Check resource availability
}
```

networkChange 事件不指示所有网络活动中的更改，仅指示个别网络连接已更改。 AIR 不尝试解释网络更改的含义。网络计算机可以具有许多真实连接和虚拟连接，因此中断连接不一定意味着丢失资源。而另一方面，新建连接也无法保证改善资源的可用性。有时，新建连接甚至可能阻止对之前可用资源的访问 （例如，连接到 VPN 时）。

通常，应用程序确定其是否可连接到远程资源的唯一方法是，尝试连接该远程资源。服务监控框架提供了一种基于事件的方式来响应对指定主机的网络连接的更改。

注：服务监控框架会检测服务器是否对请求进行接受响应。成功地进行检查并不能保证完全连接。通常，可扩展的 Web 服务使用缓存和负载平衡设备将流量重定向到 Web 服务器群集。在这种情况下，服务提供商仅提供对网络连接的局部诊断。

服务监控

Adobe AIR 1.0 和更高版本

服务监视器框架独立于 AIR 框架并位于 aircore.swc 文件中。要使用该框架， aircore.swc 文件必须包含在构建过程中。

Adobe® Flash® Builder 自动包括此库。

ServiceMonitor 类实现用于监视网络服务的框架并为服务监视器提供基本功能。默认情况下，ServiceMonitor 类的实例会调度有关网络连接的事件。在创建实例或运行时检测到网络更改时， ServiceMonitor 对象会调度这些事件。此外，可以设置 ServiceMonitor 实例的 pollInterval 属性进而以指定的间隔 （以毫秒为单位）检查连接，而不考虑一般的网络连接事件。在调用 start() 方法之前， ServiceMonitor 对象不检查网络连接。

URLMonitor 类 （ServiceMonitor 类的子类）可检测针对指定的 URLRequest 的 HTTP 连接的更改。

SocketMonitor 类 （也是 ServiceMonitor 类的子类）可在指定的端口检测到指定主机的连接的更改。

注：对于 AIR 2 之前的 AIR 版本，服务监视器框架是在 servicemonitor.swc 库中发布的。此库目前已弃用。请改用 aircore.swc 库。

Flash CS4 和 **CS5 Professional**

在 Adobe® Flash® CS4 或 CS5 Professional 中使用这些类：

1. 选择 “ 文件 ”>“ 发布设置 ” 命令。
2. 单击 ActionScript 3.0 的 “ 设置 ” 按钮。选择 “ 库路径 ”。
3. 单击 “ 浏览到 SWC” 按钮并浏览到 Flash Professional 安装文件夹中的 AIK 文件夹。
4. 在此文件夹中，找到 /frameworks/libs/air/aircore.swc （对于 AIR 2）或 /frameworks/libs/air/servicemonitor.swc （对于 AIR 1.5）。
5. 单击 “ 确定 ” 按钮。
6. 将下面的 import 语句添加到 ActionScript 3.0 代码中：

```
import air.net.*;
```

Flash CS3 Professional

要在 Adobe® Flash® CS3 Professional 中使用这些类，请将 ServiceMonitorShim 组件从 “ 组件 ” 面板拖动到 “ 库 ” 中。然后，将以下 import 语句添加到 ActionScript 3.0 代码中：

```
import air.net.*;
```

HTTP 监控

Adobe AIR 1.0 和更高版本

URLMonitor 类确定是否可从端口 80（HTTP 通信的标准端口）向指定地址发送 HTTP 请求。以下代码使用 URLMonitor 类的实例来检测到 Adobe 网站的连接更改：

```
import air.net.URLMonitor;
import flash.net.URLRequest;
import flash.events.StatusEvent;
var monitor:URLMonitor;
monitor = new URLMonitor(new URLRequest('http://www.example.com'));
monitor.addEventListener(StatusEvent.STATUS, announceStatus);
monitor.start();
function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + monitor.available);
}
```

套接字监控

Adobe AIR 1.0 和更高版本

AIR 应用程序也可将套接字连接用于推模式连接。防火墙和网络路由器通常会因某些安全原因而对未授权端口上的网络通信进行限制。为此，开发人员必须考虑到用户并非始终能够进行套接字连接。

下面的代码使用 SocketMonitor 类的实例检测对套接字连接的连接更改。已监控的端口是 6667，这是一个适用于 IRC 的通用端口：

```
import air.net.ServiceMonitor;
import flash.events.StatusEvent;

socketMonitor = new SocketMonitor('www.example.com',6667);
socketMonitor.addEventListener(StatusEvent.STATUS, socketStatusChange);
socketMonitor.start();

function announceStatus(e:StatusEvent):void {
    trace("Status change. Current status: " + socketMonitor.available);
}
```

如果套接字服务器需要安全连接，则可以使用 SecureSocketMonitor 类而不是 SocketMonitor。

域名系统 (DNS) 记录

Adobe AIR 2.0 和更高版本

使用 DNSResolver 类，您可以查找 DNS 资源记录。DNS 资源记录提供了域名的 IP 地址和 IP 地址的域名等信息。可以查找以下类型的 DNS 资源记录：

- ARecord — 主机的 IPv4 地址。
- AAAARecord — 主机的 IPv6 地址。
- MXRecord — 主机的邮件交换记录。
- PTRRecord — IP 地址的主机名。
- SRVRecord — 服务的服务记录。。

要查找记录，您可将查询字符串和表示记录类型的类对象传递给 DNSResolver 对象的 lookup() 方法。要使用的查询字符串取决于记录类型：

记录类	查询字符串	查询字符串示例
ARecord	主机名	“example.com”
AAAARecord	主机名	“example.com”
MXRecord	主机名	“example.com”
PTRRecord	IP 地址	“208.77.188.166”
SRVRecord	服务标识符：_service._protocol.host	“_sip._tcp.example.com”

以下代码示例查找主机 “example.com” 的 IP 地址。

```
package
{
    import flash.display.Sprite;
    import flash.events.DNSResolverEvent;
    import flash.events.ErrorEvent;
    import flash.net.dns.ARecord;
    import flash.net.dns.DNSResolver;

    public class DNSResolverExample extends Sprite
    {

        public function DNSResolverExample()
        {
            var resolver:DNSResolver = new DNSResolver();
            resolver.addEventListener( DNSResolverEvent.LOOKUP, lookupComplete );
            resolver.addEventListener( ErrorEvent.ERROR, lookupError );

            resolver.lookup( "example.com.", ARecord );
        }

        private function lookupComplete( event:DNSResolverEvent ):void
        {
            trace( "Query string: " + event.host );
            trace( "Record count: " + event.resourceRecords.length );
            for each( var record:* in event.resourceRecords )
            {
                if( record is ARecord ) trace( record.address );
            }

        }

        private function lookupError( error:ErrorEvent ):void
        {
            trace("Error: " + error.text );
        }
    }
}
```

有关详细信息，请参阅：

- DNSResolver
- DNSResolverEvent
- ARecord
- AAAARecord
- MXRecord
- PTRRecord
- SRVRecord

第 43 章：套接字

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

套接字是在两个计算机进程之间建立的一种网络连接类型。通常，这些进程在两台连接到相同 Internet 协议 (IP) 网络的不同计算机上运行。然而，连接的进程可以在使用特定 " 本地主机 "IP 地址的同一台计算机上运行。

Adobe Flash Player 支持客户端传输控制协议 (TCP) 套接字。 Flash Player 应用程序可以作为套接字服务器连接到其他进程，但是不能接受来自其他进程的传入连接请求。换句话说， Flash Player 应用程序可以连接到 TCP 服务器，但不能用作 TCP 服务器。

Flash Player API 也包含 XMLSocket 类。XMLSocket 类使用 Flash Player 特定的协议，通过该协议，您可以与识别该协议的服务器交换 XML 消息。 ActionScript 1 中引入了 XMLSocket 类，该类现在仍然受支持以提供向后兼容性。通常，除非连接到特别创建以与 Flash XMLSocket 进行通信的服务器，否则 Socket 类应该用于新应用程序。

Adobe AIR 添加了多个用于基于套接字的网络编程的附加类。借助 ServerSocket 类， AIR 应用程序可以用作 TCP 套接字服务器，并可以连接到要求 SSL 或 TLS 安全的套接字服务器。AIR 应用程序还可以使用 DatagramSocket 类发送和接收通用数据报协议 (UDP) 消息。

更多帮助主题

flash.net 包

第 916 页的 " 连接到套接字 "

TCP 套接字

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过传输控制协议 (TCP)，可通过永久网络连接交换消息。 TCP 可确保发送的任何消息都以正确的顺序到达，出现重大网络问题时除外。TCP 连接要求具有 " 客户端 " 和 " 服务器 "。Flash Player 可以创建客户端套接字。此外，Adobe AIR 可以创建服务器套接字。

下列 ActionScript API 提供了 TCP 连接：

- 套接字 — 允许客户端应用程序连接到服务器。 Socket 类无法侦听传入连接。
- SecureSocket (AIR) — 允许客户端应用程序连接到受信任服务器并进行加密通信。
- ServerSocket (AIR) — 允许应用程序侦听传入连接并用作服务器。
- XMLSocket — 允许客户端应用程序连接到 XMLSocket 服务器。

二进制客户端套接字

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

二进制套接字连接与 XML 套接字类似，不同的是客户端和服务器不局限于交换 XML 消息。该连接可以将数据作为二进制信息传输。因此，您可以连接到更加丰富的服务，包括邮件服务器 （POP3、 SMTP 和 IMAP）和新闻服务器 (NNTP)。

Socket 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Socket 类，您可以建立套接字连接以及读取和编写原始二进制数据。在与使用二进制协议的服务器进行交互操作时，Socket 类非常有用。使用二进制套接字连接，您可以编写与多个不同的 Internet 协议 （例如 POP3、 SMTP、 IMAP 和 NNTP）进行交互的代码。这种交互进而又使您的应用程序可以连接到邮件服务器和新闻服务器。

Flash Player 可通过使用服务器的二进制协议直接与该服务器连接。某些服务器使用 big-endian 字节顺序，某些服务器则使用 little-endian 字节顺序。 Internet 上的大多数服务器使用 big-endian 字节顺序，因为 " 网络字节顺序 " 为 big-endian。little-endian 字节顺序很常用，因为 Intel® x86 体系结构使用该字节顺序。您应使用与收发数据的服务器的字节顺序相匹配的 endian 字节顺序。默认情况下， IDataInput 和 IDataOutput 接口执行的所有操作和实现这些接口的类 （ByteArray、Socket 和 URLStream）都以 big-endian 格式编码；即，最高有效字节位于前面。选择的这种默认字节顺序与 Java 和正式网络字节顺序匹配。要更改是使用 big-endian 还是使用 little-endian 字节顺序，可以将 endian 属性设置为 Endian.BIG_ENDIAN 或 Endian.LITTLE_ENDIAN。

Socket 类继承了由 IDataInput 和 IDataOutput 接口定义的所有方法（位于 flash.utils 包中）。必须使用这些方法写入和读取套接字。

有关详细信息，请参阅：

- Socket
- IDataInput
- IDataOutput
- socketData 事件

安全客户端套接字 (AIR)

Adobe AIR 2 和更高版本

您可以使用 SecureSocket 类连接到使用第 4 版安全套接字层 (SSLv4) 或第 1 版传输层安全性 (TLSv1) 的套接字服务器。安全套接字提供了三大优势：服务器身份验证、数据完整性和消息机密性。运行时使用服务器证书及其在用户信任存储区中授权机构颁发的根证书或中级证书的相关性对服务器进行身份验证。运行时依靠 SSL 和 TLS 协议实现所使用的加密算法提供数据完整性和消息机密性。

当您使用 SecureSocket 对象连接到服务器时，运行时将使用证书信任存储区验证服务器证书。在 Windows 和 Mac 上，操作系统提供信任存储区。在 Linux 中，运行时自行提供信任存储区。

如果服务器证书无效或不可信，运行时将调度 ioError 事件。您可以检查 SecureSocket 对象的 serverCertificateStatus 属性来确定验证失败的原因。没有为与不具备有效和可信证书的服务器进行通信提供任何设置。

CertificateStatus 类用于定义表示可能的验证结果的字符串常量：

- 过期 — 已超过证书过期日期。
- 无效 — 多种原因可导致证书无效。例如，证书可能已更改、损坏，或者是错误类型的证书。
- 无效链 — 证书的服务器链中有一个或多个证书无效。
- 主体不匹配 — 服务器的主机名与证书公用名不匹配。换句话说，服务器使用的是错误的证书。
- 吊销 — 证书颁发机构已吊销该证书。
- 受信任 — 证书有效且受信任。 SecureSocket 对象仅可以连接到使用有效的受信任证书的服务器。
- 未知 — SecureSocket 对象尚未验证证书。serverCertificateStatus 属性在您调用 connect() 和调度 connect 或 ioError 事件之前具有此状态值。
- 不受信任的签名者 — 证书不能 " 链接 " 到客户端计算机的信任存储区中的受信任根证书。

与 SecureSocket 对象通信要求服务器使用安全协议，并包含有效的受信任证书。在其他方面，使用 SecureSocket 对象与使用 Socket 对象是相同的。

并不是所有平台都支持 SecureSocket 对象。使用 SecureSocket 类 isSupported 属性测试运行时是否支持在当前客户端计算机上使用 SecureSocket 对象。

有关详细信息，请参阅：

- SecureSocket
- CertificateStatus
- IDataInput
- IDataOutput
- socketData 事件

TCP 套接字示例：构建 Telnet 客户端

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Telnet 示例说明了使用 Socket 类连接远程服务器和传输数据的方法。该示例演示下列方法：

- 使用 Socket 类创建自定义 telnet 客户端
- 使用 ByteArray 对象将文本发送到远程服务器
- 处理从远程服务器收到的数据

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/Telnet 文件夹中找到 Telnet 应用程序文件。该应用程序包含以下文件：

文件	说明
TelnetSocket.fla 或 TelnetSocket.mxml	由用户界面组成的 Flex 或 Flash 的主应用程序文件 （分别为 MXML 和 FLA 格式）。
TelnetSocket.as	用于提供用户界面逻辑的文档类 （仅限 Flash）。
com/example/programmingas3/Telnet/Telnet.as	为应用程序提供 Telnet 客户端功能，例如连接到远程服务器以及发送、接收和显示数据。

Telnet 套接字应用程序概述

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

主 TelnetSocket.mxml 文件负责为整个应用程序创建用户界面 (UI)。

除了 UI，此文件还定义两个方法 login() 和 sendCommand()，以便将用户连接到指定的服务器。

下面的代码列出了主应用程序文件中的 ActionScript：

```
import com.example.programmingas3.socket.Telnet;

private var telnetClient:Telnet;
private function connect():void
{
    telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
    console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
    console.enabled = true;
}
private function sendCommand():void
{
    var ba:ByteArray = new ByteArray();
    ba.writeMultiByte(command.text + "\n", "UTF-8");
    telnetClient.writeBytesToSocket(ba);
    command.text = "";
}
```

第一行代码从自定义 com.example.programmingas.socket 包中导入 Telnet 类。第二行代码声明 Telnet 类的实例 telnetClient，稍后由 connect() 方法初始化该实例。接下来，声明 connect() 方法，该方法初始化先前声明的 telnetClient 变量。此方法传递用户指定的 telnet 服务器名称、 telnet 服务器端口和对显示列表中 TextArea 组件的引用，其中，显示列表用于显示来自套接字服务器的文本响应。connect() 方法的最后两行设置 Panel 的 title 属性并启用 Panel 组件，该组件允许用户将数据发送到远程服务器。主应用程序文件中的最后一个方法 sendCommand() 用于将用户的命令作为 ByteArray 对象发送到远程服务器。

Telnet 类概述

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Telnet 类负责连接到远程 Telnet 服务器和发送 / 接收数据。

Telnet 类声明下列私有变量:

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

第一个变量 serverURL 包含要连接到的用户指定的服务器地址。

第二个变量 portNumber 是 Telnet 服务器当前在其上运行的端口号。默认情况下， Telnet 服务在端口 23 上运行。

第三个变量 socket 是一个 Socket 实例，该实例尝试连接到 serverURL 和 portNumber 变量定义的服务器。

第四个变量 ta 是对舞台上的 TextArea 组件实例的引用。此组件用于显示来自远程 Telnet 服务器的响应或者任何可能的错误消息。

最后一个变量 state 是数值，用于确定 Telnet 客户端支持哪些选项。

正如您之前所见， Telnet 类的构造函数由主应用程序文件中的 connect() 方法调用。

Telnet 构造函数采用三个参数: server、port 和 output。server 和 port 参数指定 Telnet 服务器在其上运行的服务器名称和端口号。最后一个参数 output 是对舞台上的 TextArea 组件实例的引用，在该舞台上将为用户显示服务器输出。

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

向套接字写入数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要将数据写入套接字连接，可以调用 Socket 类中的任何写入方法。这些写入方法包括 writeBoolean()、 writeByte()、 writeBytes()、 writeDouble() 及其他方法。然后，使用 flush() 方法刷新输出缓冲区中的数据。在 Telnet 服务器中，使用 writeBytes() 方法向套接字连接中写入数据，该方法将字节数组作为参数，并将其发送到输出缓冲区。 writeBytesToSocket() 方法如下：

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

此方法是由主应用程序文件中的 sendCommand() 方法调用的。

显示来自套接字服务器的消息

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当从套接字服务器接收消息，或者发生事件时，将调用自定义 msg() 方法。此方法将字符串追加到舞台上的 TextArea 并调用自定义 setScroll() 方法，该方法使 TextArea 组件滚动到底部。 msg() 方法如下：

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

如果您未将 TextArea 组件的内容设置为自动滚动，则用户需要手动拖动文本区域中的滚动条才能看到来自服务器的最新响应。

滚动 TextArea 组件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

setScroll() 方法包含一行 ActionScript，用于垂直滚动 TextArea 组件内容，以便用户可以查看返回文本的最后一行。下面的片断显示了 setScroll() 方法：

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

此方法设置 verticalScrollPosition 属性 （该属性是当前显示的最上面一行字符的行号），并将其设置为 maxVerticalScrollPosition 属性的值。

XML 套接字

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过 XML 套接字，可以创建与远程服务器的连接，且该服务器在明确关闭之前始终保持打开状态。您可以在服务器和客户端之间交换字符串数据，如 XML。使用 XML 套接字服务器的优点之一是客户端不需要明确请求数据。服务器无需等待请求即可发送数据，并且可以将数据发送到每个已连接的客户端。

在应用程序沙箱外的 Flash Player 和 Adobe AIR 内容中， XML 套接字连接要求在目标服务器上提供套接字策略文件。有关详细信息，请参阅第 901 页的 “ 网站控制 （策略文件） ” 和第 916 页的 “ 连接到套接字 ”。

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 Media Server （支持 RTMP）。

对于应用程序安全沙箱外的 Flash Player 或 AIR 应用程序中的内容使用 XMLSocket 对象连接到服务器的方式及位置，规定了下列限制：

- 对于应用程序安全沙箱外部的内容，XMLSocket.connect() 方法只能连接到端口号大于或等于 1024 的 TCP 端口。这种限制所带来的后果之一是，向与 XMLSocket 对象通信的服务器守护程序分配的端口号也必须大于等于 1024。端口号小于 1024 的端口通常用于系统服务，例如 FTP (21)、Telnet (23)、SMTP (25)、HTTP (80) 和 POP3 (110)，因此，出于安全方面的考虑，禁止 XMLSocket 对象使用这些端口。这种端口号方面的限制可以减少不恰当地访问和滥用这些资源的可能性。
- 对于应用程序安全沙箱外部的内容， XMLSocket.connect() 方法只能连接到该内容所在的同一域中的计算机。（此限制与 URLLoader.load() 的安全规则相同。）若要连接到在内容所在域之外的其他域中运行的服务器守护程序，可以在该服务器上创建一个允许从特定域进行访问的跨域策略文件。有关跨域策略文件的详细信息，请参阅第 922 页的 “AIR 安全性 ”。

注：将服务器设置为与 XMLSocket 对象进行通信可能会遇到一些困难。如果您的应用程序不需要进行实时交互，请使用 URLLoader 类，而不要使用 XMLSocket 类。

可以使用 XMLSocket 类的 XMLSocket.connect() 和 XMLSocket.send() 方法，通过套接字连接与服务器之间传输 XML。XMLSocket.connect() 方法与 Web 服务器端口建立套接字连接。XMLSocket.send() 方法将 XML 对象传递给套接字连接中指定的服务器。

当调用 XMLSocket.connect() 方法时，应用程序会打开到服务器的 TCP/IP 连接并使该连接保持打开状态，直到出现下列情况之一：

- XMLSocket 类的 XMLSocket.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）。

使用 XMLSocket 类连接到服务器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要创建套接字连接，必须创建服务器端应用程序以等待套接字连接请求，并将响应发送到 Flash Player 或 AIR 应用程序。此类服务器端应用程序可使用 AIR 或其他编程语言 （如 Java、 Python 或 Perl）编写。要使用 XMLSocket 类，服务器计算机必须运行可识别 XMLSocket 类使用的简单协议的守护程序：

- XML 消息通过全双工 TCP/IP 流套接字连接发送。
- 每个 XML 消息都是一个完整的 XML 文档，以一个零 (0) 字节结束。
- 通过 XMLSocket 连接发送和接收的 XML 消息的数量没有限制。

创建并连接到 Java XML 套接字服务器

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

下面的代码演示了一个用 Java 编写的简单 XMLSocket 服务器，该服务器接受传入连接并在命令提示窗口中显示接收到的消息。虽然从命令行启动服务器时可以指定其他端口号，但默认情况下，在本地计算机上的 8080 端口创建新服务器。

新建一个文本文档并添加下面的代码：

```
import java.io.*;
import java.net.*;

class SimpleServer
{
    private static SimpleServer server;
    ServerSocket socket;
    Socket incoming;
    BufferedReader readerIn;
    PrintStream printOut;

    public static void main(String[] args)
    {
        int port = 8080;

        try
        {
            port = Integer.parseInt(args[0]);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            // Catch exception and keep going.
        }

        server = new SimpleServer(port);
    }

    private SimpleServer(int port)
    {
        System.out.println(">> Starting SimpleServer");
        try
        {
            socket = new ServerSocket(port);
            incoming = socket.accept();
            readerIn = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
            printOut = new PrintStream(incoming.getOutputStream());
            printOut.println("Enter EXIT to exit.\r");
            out("Enter EXIT to exit.\r");
            boolean done = false;
```

```
            while (!done)
            {
                String str = readerIn.readLine();
                if (str == null)
                {
                    done = true;
                }
                else
                {
                    out("Echo: " + str + "\r");
                    if(str.trim().equals("EXIT"))
                    {
                        done = true;
                    }
                }
                incoming.close();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

    private void out(String str)
    {
        printOut.println(str);
        System.out.println(str);
    }
}
```

将文档保存到硬盘，命名为 SimpleServer.java 并使用 Java 编译器对其进行编译，这会创建一个名为 SimpleServer.class 的 Java 类文件。

您可以通过打开命令提示并键入 java SimpleServer 来启动 XMLSocket 服务器。 SimpleServer.class 文件可以位于本地计算机或网络上的任何位置，不需要放置在 Web 服务器的根目录中。

如果由于文件没有位于 Java 类路径中而无法启动服务器，请尝试使用 java -classpath .SimpleServer 启动服务器。

要从应用程序连接到 XMLSocket，需要新建一个 XMLSocket 类实例，并在传递主机名和端口号时调用 XMLSocket.connect() 方法，如下所示：

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

只要从服务器接收数据，就会调度 data 事件 (flash.events.DataEvent.DATA)：

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

若要将数据发送到 XMLSocket 服务器，可以使用 XMLSocket.send() 方法并传递 XML 对象或字符串。Flash Player 将提供的参数转换为 String 对象，并将内容发送到 XMLSocket 服务器 （后跟零 (0) 字节）：

```
xmlsock.send(xmlFormattedData);
```

XMLSocket.send() 方法不返回指示数据是否成功传输的值。如果尝试发送数据时发生错误，将引发 IOError 错误。

发送到 XML 套接字服务器的每条消息必须以换行符 (\n) 结束。

有关详细信息，请参阅 XMLSocket。

服务器套接字

Adobe AIR 2 和更高版本

使用 ServerSocket 类可以允许其他进程使用传输控制协议 (TCP) 套接字连接到您的应用程序。可以在本地计算机或另一台网络连接的计算机上运行连接进程。当 ServerSocket 对象收到连接请求时，会调度 connect 事件。该事件调度的 ServerSocketConnectEvent 对象包含 Socket 对象。您可以使用此 Socket 对象与其他进程进行后续通信。

要侦听传入的套接字连接，请执行以下操作：

1 创建一个 ServerSocket 对象并将其绑定到本地端口

2 为 connect 事件添加事件侦听器

3 调用 listen() 方法

4 响应 connect 事件，它为每个传入连接提供 Socket 对象

ServerSocket 对象在您调用 close() 方法之前会继续侦听新连接。

以下代码示例说明了如何创建套接字服务器应用程序。该示例侦听端口 8087 上的传入连接。收到连接时，此示例会将消息（字符串 “Connected”）发送给客户端套接字。此后，服务器会将任何收到的消息回显给客户端。

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;
    import flash.events.ServerSocketConnectEvent;
    import flash.net.ServerSocket;
    import flash.net.Socket;

    public class ServerSocketExample extends Sprite
    {
        private var serverSocket:ServerSocket;
        private var clientSockets:Array = new Array();

        public function ServerSocketExample()
        {
            try
            {
                // Create the server socket
                serverSocket = new ServerSocket();

                // Add the event listener
                serverSocket.addEventListener( Event.CONNECT, connectHandler );
                serverSocket.addEventListener( Event.CLOSE, onClose );

                // Bind to local port 8087
                serverSocket.bind( 8087, "127.0.0.1" );

                // Listen for connections
                serverSocket.listen();
                trace( "Listening on " + serverSocket.localPort );

            }
            catch(e:SecurityError)
            {
                trace(e);
```

```
            }
        }

        public function connectHandler(event:ServerSocketConnectEvent):void
        {
            //The socket is provided by the event object
            var socket:Socket = event.socket as Socket;
            clientSockets.push( socket );

            socket.addEventListener( ProgressEvent.SOCKET_DATA, socketDataHandler);
            socket.addEventListener( Event.CLOSE, onClientClose );
            socket.addEventListener( IOErrorEvent.IO_ERROR, onIOError );

            //Send a connect message
            socket.writeUTFBytes("Connected.");
            socket.flush();

            trace( "Sending connect message" );
        }

        public function socketDataHandler(event:ProgressEvent):void
        {
            var socket:Socket = event.target as Socket

            //Read the message from the socket
            var message:String = socket.readUTFBytes( socket.bytesAvailable );
            trace( "Received: " + message);

            // Echo the received message back to the sender
            message = "Echo -- " + message;
            socket.writeUTFBytes( message );
            socket.flush();
            trace( "Sending: " + message );
        }

        private function onClientClose( event:Event ):void
        {
            trace( "Connection to client closed." );
            //Should also remove from clientSockets array...
        }

        private function onIOError( errorEvent:IOErrorEvent ):void
        {
            trace( "IOError: " + errorEvent.text );
        }

        private function onClose( event:Event ):void
        {
            trace( "Server socket closed by OS." );
        }
}}
```

有关详细信息，请参阅：

- ServerSocket
- ServerSocketConnectEvent
- Socket

UDP 套接字 (AIR)

Adobe AIR 2 和更高版本

通用数据报协议 (UDP) 提供了一种通过无状态网络连接交换消息的方法。UDP 无法确保消息按顺序传送，甚至无法确保消息的传送。使用 UDP，操作系统的网络代码通常在封送、跟踪和确认消息上将花费更少的时间。因此，通常 UDP 消息到达目标应用程序的延迟比 TCP 消息到达目标应用程序的延迟要短。

在必须发送实时信息 （例如游戏中的位置更新或音频聊天应用程序中的声音数据包）时， UDP 套接字通信很有用。在此类应用程序中，丢失一些数据是可以接受的，并且低传输延迟比保证及时到达更重要。对于几乎所有其他目的， TCP 套接字是更好的选择。

AIR 应用程序可以使用 DatagramSocket 和 DatagramSocketDataEvent 类发送和接收 UDP 消息。要发送或接收 UDP 消息，请执行以下操作：

1 创建一个 DatagramSocket 对象
2 为 data 事件添加事件侦听器
3 使用 bind() 方法将套接字绑定到本地 IP 地址和端口
4 通过调用 send() 方法发送消息，传递目标计算机的 IP 地址和端口
5 通过响应 data 事件接收消息。为此事件调度的 DatagramSocketDataEvent 对象包含一个 ByteArray 对象，该对象中包含消息数据。

以下代码示例说明应用程序如何发送和接收 UDP 消息。此示例将包含字符串 “Hello” 的单一消息发送到目标计算机。它还跟踪接收的任何消息内容。

```
package
{
import flash.display.Sprite;
import flash.events.DatagramSocketDataEvent;
import flash.events.Event;
import flash.net.DatagramSocket;
import flash.utils.ByteArray;

public class DatagramSocketExample extends Sprite
{
    private var datagramSocket:DatagramSocket;

    //The IP and port for this computer
    private var localIP:String = "192.168.0.1";
    private var localPort:int = 55555;

    //The IP and port for the target computer
    private var targetIP:String = "192.168.0.2";
    private var targetPort:int = 55555;

    public function DatagramSocketExample()
    {
        //Create the socket
        datagramSocket = new DatagramSocket();
        datagramSocket.addEventListener( DatagramSocketDataEvent.DATA, dataReceived );

        //Bind the socket to the local network interface and port
```

```
        datagramSocket.bind( localPort, localIP );

        //Listen for incoming datagrams
        datagramSocket.receive();

        //Create a message in a ByteArray
        var data:ByteArray = new ByteArray();
        data.writeUTFBytes("Hello.");

        //Send the datagram message
        datagramSocket.send( data, 0, 0, targetIP, targetPort);
    }

    private function dataReceived( event:DatagramSocketDataEvent ):void
    {
        //Read the data from the datagram
        trace("Received from " + event.srcAddress + ":" + event.srcPort + "> " +
            event.data.readUTFBytes( event.data.bytesAvailable ) );
    }
}}
```

使用 UDP 套接字时，请记住下列注意事项：

- 单一数据包不能大于网络接口或发送方和接收方之间任何网络节点的最大传输单位 (MTU) 中最小的那一个。传递到 send() 方法的 ByteArray 对象中的所有数据都将作为单一数据包发送。（在 TCP 中，较大的消息被分为几个单独的包。）
- 发送方和目标之间不存在信号交换。如果目标不存在或指定端口上没有活动侦听器，则会丢弃消息且不会报错。
- 使用 connect() 方法时，将忽略从其他源发送的消息。 UDP 连接仅提供方便的数据包过滤。这并不意味着目标地址和端口上必须存在有效的侦听进程。
- UDP 流量可以造成网络拥塞。如果发生网络拥塞，网络管理员可能需要实现服务质量控制。（TCP 有内置的流量控制来减少网络拥塞的影响。）

有关详细信息，请参阅：

- DatagramSocket
- DatagramSocketDataEvent
- ByteArray

IPv6 地址

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 9.0.115.0 及更高版本支持 IPv6（Internet 协议版本 6）。IPv6 是支持 128 位地址的 Internet 协议版本（它是支持 32 位地址的早期 IPv4 协议的改进版本）。您可能需要在网络接口中激活 IPv6。有关详细信息，请参阅承载数据的操作系统的帮助。

如果承载系统中支持 IPv6，您可以在用中括号 ([]) 括起的 URL 中指定数字 IPv6 文本地址，如下所示：

```
[2001:db8:ccc3:ffff:0:444d:555e:666f]
```

Flash Player 根据以下规则返回 IPv6 字面值：

- Flash Player 返回长形式的 IPv6 地址字符串。
- IP 值没有双冒号缩写词。
- 十六进制数字全小写。

- IPv6 地址包含在中括号 ([]) 中。
- 每个四重地址都输出为 0 到 4 个十六进制数字 （省略前导零）。
- 内容全为零的四重地址输出为单个零 （而不是双冒号），下表所列例外情况除外。

Flash Player 返回的 IPv6 值具有以下例外：

- 未指定的 IPv6 地址 （内容全为零）输出为 [::]。
- 环回或本地主机 IPv6 地址输出为 [::1]。
- IPv4 映射 （转换为 IPv6）地址输出为 [::ffff:a.b.c.d]，其中 a.b.c.d 为典型的 IPv4 点分十进制值。
- IPv4 兼容地址输出为 [::a.b.c.d]，其中 a.b.c.d 为典型的 IPv4 点分十进制值。

第 44 章：HTTP 通信

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe® AIR® 和 Adobe® Flash® Player 应用程序可以与基于 HTTP 的服务器通信，以便加载数据、图像、视频和交换消息。

更多帮助主题

flash.net.URLLoader

flash.net.URLStream

flash.net.URLRequest

flash.net.URLRequestDefaults

flash.net.URLRequestHeader

flash.net.URLRequestMethod

flash.net.URLVariables

加载外部数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 3.0 包含用于从外部源加载数据的机制。这些源可以提供静态内容 （例如文本文件）或 Web 脚本生成的动态内容。可以使用多种方法来设置数据的格式，并且 ActionScript 提供了用于解码和访问数据的功能。也可以在检索数据的过程中将数据发送到外部服务器。

使用 URLRequest 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

加载外部数据的许多 API 使用 URLRequest 类来定义所需网络请求的属性。

URLRequest 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以在任何安全沙箱中设置 URLRequest 对象的下列属性：

属性	说明
contentType	使用 URL 请求发送的任何数据的 MIME 内容类型。如果未设置 contentType，则值将作为 application/x-www-form-urlencoded 发送。
data	一个对象，它包含将随 URL 请求一起传输的数据。
digest	唯一标识要存储为 （或从其检索） Adobe® Flash® Player 缓存的已签名 Adobe 平台组件的字符串。

属性	说明
method	HTTP 请求方法，例如 GET 或 POST。（AIR 应用程序安全域中运行的内容可将除 "GET" 或 "POST" 之外的字符串指定为 method 属性。允许任何 HTTP 动词，"GET" 为默认方法。请参阅第 922 页的 "AIR 安全性 "。）
requestHeaders	要追加到 HTTP 请求的 HTTP 请求标头的数组。请注意，在 Flash Player 以及在应用程序安全沙箱外运行的 AIR 内容中，设置一些标题的权限时会受到限制。
url	指定要请求的 URL。

在 AIR 中，您可以设置 URLRequest 类的其他属性，这些属性仅适用于在应用程序安全沙箱中运行的 AIR 内容。应用程序沙箱中的内容还可以使用新 URL 方案 （除 file 和 http 等标准方案之外）定义 URL。

属性	说明
followRedirects	指定是否要遵循重定向 （默认值为 true ；如果不遵循，则为 false）。仅 AIR 应用程序沙箱支持此属性。
manageCookies	指定 HTTP 协议堆栈是否应管理此请求的 cookie （默认值为 true ；如果不管理，则为 false）。仅 AIR 应用程序沙箱支持设置此属性。
authenticate	指定是否应为此请求处理身份验证请求 （如果是，则为 true）。仅 AIR 应用程序沙箱支持设置此属性。默认为对请求进行身份验证 — 如果服务器要求提供凭据，则可能会显示身份验证对话框。您还可以使用 URLRequestDefaults 类设置用户名和密码 — 请参阅第 697 页的 " 设置 URLRequest 默认值 （仅 AIR)"。
cacheResponse	指定是否为此请求缓存响应数据。仅 AIR 应用程序沙箱支持设置此属性。默认值为缓存响应 (true)。
useCache	指定在此 URLRequest 获取数据之前是否应查询本地缓存。仅 AIR 应用程序沙箱支持设置此属性。默认值 (true) 为使用本地缓存版本 （如果可用）。
userAgent	指定要在 HTTP 请求中使用的用户代理字符串。

注：HTMLLoader 类具有相关属性，用于设置与 HTMLLoader 对象加载的内容有关的设置。有关详细信息，请参阅第 839 页的 " 关于 HTMLLoader 类 "。

设置 URLRequest 默认值 （仅 AIR)

Adobe AIR 1.0 和更高版本

借助 URLRequestDefaults 类，您可以定义 URLRequest 对象的应用程序特定默认设置例如，以下代码设置 manageCookies 和 useCache 属性的默认值。所有新 URLRequest 对象将使用这些属性的指定值，而不是常规默认值：

```
URLRequestDefaults.manageCookies = false;
URLRequestDefaults.useCache = false;
```

注：URLRequestDefaults 类仅针对 Adobe AIR 中运行的内容进行定义。 Flash Player 中不支持该类。

URLRequestDefaults 类包含 setLoginCredentialsForHost() 方法，使用该方法可指定要为特定主机使用的默认用户名和密码。该方法的 hostname 参数中定义的主机可以为域 （例如 "www.example.com"）或域加端口号 （例如 "www.example.com:80"）。请注意， "example.com"、 "www.example.com" 和 "sales.example.com" 均视为唯一的主机。

只有在服务器要求提供凭据时才会使用这些凭据。如果用户已通过身份验证 （例如，使用身份验证对话框），则调用 setLoginCredentialsForHost() 方法不会更改通过身份验证的用户。

以下代码设置用于发送到 www.example.com 的请求的默认用户名和密码：

```
URLRequestDefaults.setLoginCredentialsForHost("www.example.com", "Ada", "love1816$X");
```

URLRequestDefaults 设置仅适用于当前应用程序域，只有一个例外。传递到 setLoginCredentialsForHost() 方法的凭据用于 AIR 应用程序中任何应用程序域所发出的请求。

有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的 URLRequestDefaults 类。

URI 方案

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

标准 URI 方案 （例如下列方案）可以用于任何安全沙箱所发出的请求：

http: 和 **https:**
将这些用于标准 Internet URL （与在 Web 浏览器的使用方式相同）。

文件：
使用 file: 指定位于本地文件系统中的文件的 URL。例如：

```
file:///c:/AIR Test/test.txt
```

在 AIR 中，您还可以在定义在应用程序安全性沙箱中运行的内容的 URL 时使用下列方案：

app:
使用 app: 指定相对于安装的应用程序的根目录的路径。例如，以下路径指向应用程序安装目录的 resources 子目录：

```
app:/resources
```

使用 AIR Debug Launcher (ADL) 启动 AIR 应用程序时，应用程序目录是包含应用程序描述符文件的目录。

使用 File.applicationDirectory 创建的 File 对象的 URL （和 url 属性）使用 app URI 方案，如下所示：

```
var dir:File = File.applicationDirectory;
dir = dir.resolvePath("assets");
trace(dir.url); // app:/assets
```

app-storage:
使用 app-storage: 指定相对于应用程序的数据存储目录的路径。 AIR 为安装的每个应用程序 （和用户）都创建了唯一的应用程序存储目录，这些目录对于存储特定于各个应用程序的数据非常有用。例如，以下路径指向应用程序存储目录的 settings 子目录中的 prefs.xml 文件：

```
app-storage:/settings/prefs.xml
```

使用 File.applicationStorageDirectory 创建的 File 对象的 URL （和 url 属性）使用 app-storage URI 方案，如下所示：

```
var prefsFile:File = File.applicationStorageDirectory;
prefsFile = prefsFile.resolvePath("prefs.xml");
trace(dir.prefsFile); // app-storage:/prefs.xml
```

mailto:
在传递给 navigateToURL() 函数的 URLRequest 对象中可以使用 mailto 方案。请参阅第 710 页的 “ 在其他应用程序中打开 URL”。

您可以使用 URLRequest 对象 （使用这些 URI 方案中的任何一个方案）定义许多不同对象 （例如 FileStream 或 Sound 对象）的 URL 请求。还可以在 AIR 中运行的 HTML 内容中使用这些方案；例如，可以在 img 标签的 src 属性中使用它们。

但是，您仅可以使用应用程序安全沙箱中的内容中的 AIR 特定 URI 方案 （app: 和 app-storage:）。有关详细信息，请参阅第 922 页的 “AIR 安全性 ”。

设置 URL 变量

当可以直接向 URL 字符串添加变量时，可以更轻松地使用 URLVariables 类来定义请求所需要的任何变量。

可以使用三个方法向 URLVariables 对象添加参数：

- 在 URLVariables 构造函数中

- 使用 URLVariables.decode() 方法
- 作为 URLVariables 对象自身的动态属性

以下示例演示了全部三个方法以及如何将变量分配到 URLRequest 对象：

```
var urlVar:URLVariables = new URLVariables( "one=1&two=2" );
urlVar.decode("amp=" + encodeURIComponent( "&" ) );
urlVar.three = 3;
urlVar.amp2 = "&&";
trace(urlVar.toString()); //amp=%26&amp2=%26%26&one=1&two=2&three=3

var urlRequest:URLRequest = new URLRequest( "http://www.example.com/test.cfm" );
urlRequest.data = urlVar;
```

当在 URLVariables 构造函数或在 URLVariables.decode() 方法中定义变量时，请确保 URL 编码在 URI 字符串中包含特定含义的字符。例如，当在参数名称或值中使用 & 符时，必须通过将其从 & 更改为 %26 来编码该 & 符，因为 & 符用作分隔符。顶级 encodeURIComponent() 函数可以用于此目的。

使用 URLLoader 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

借助 URLLoader 类，您可以向服务器发送请求并访问返回的信息。您也可以使用 URLLoader 类来访问允许访问本地文件的上下文中 （例如 Flash Player 只能与本地文件系统内容交互的沙箱和 AIR 应用程序沙箱）的本地文件系统上的文件。URLLoader 类以文本、二进制数据或 URL 编码变量形式从 URL 下载数据。 URLLoader 类调度的事件包括 complete、 httpStatus、 ioError、 open、 progress 和 securityError 等。

ActionScript 3.0 事件处理模型与 ActionScript 2.0 模型有很大区别，后者使用 LoadVars.onData、 LoadVars.onHTTPStatus 和 LoadVars.onLoad 事件处理程序。有关在 ActionScript 3.0 中处理事件的详细信息，请参阅第 106 页的 “ 处理事件 ”

下载完成之前，下载的数据不可用。可通过侦听要调度的 progress 事件监视下载进度 （已加载字节数和总字节数）。但是，如果文件加载足够快，可能不调度 progress 事件。成功下载文件后，将调度 complete 事件。通过设置 URLLoader dataFormat 属性，您可以以文本、原始二进制数据或 URLVariables 对象格式接收数据。

URLLoader.load() 方法 （并且可能 URLLoader 类的构造函数）使用单个参数， request，该参数是 URLRequest 对象。URLRequest 对象包含所有用于单一 HTTP 请求的信息，例如目标 URL、请求方法 （GET 或 POST）、其他标题信息和 MIME 类型。

例如，若要将 XML 数据包上载到服务器端脚本，可以使用以下代码：

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <clock>
        <time>{secondsUTC}</time>
    </clock>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/time.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
loader.load(request);
```

上一个代码片断创建包含要发送到服务器的 XML 包的名为 dataXML 的 XML 文档。此示例将 URLRequest contentType 属性设置为 "text/xml" 并将 XML 文档分配给 URLRequest data 属性。最后，示例创建 URLLoader 对象并通过使用 load() 方法将请求发送到远程脚本。

使用 URLStream 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

数据到达时， URLStream 类将提供对下载的数据的访问。并且 URLStream 类还允许在完成下载前关闭流。下载的数据以原始二进制数据的形式提供。

当从 URLStream 对象读取数据时，使用 bytesAvailable 属性确定在读取数据之前是否提供了足够的数据。当您尝试阅读的数据比可用数据多时，将引发 EOFError 异常。

httpResponseStatus 事件 **(AIR)**

在 Adobe AIR 中，除调度 httpStatus 事件之外， URLStream 类还调度 httpResponseStatus 事件。 httpResponseStatus 事件在任何响应数据之前传送。 httpResponseStatus 事件 （由 HTTPStatusEvent 类表示）包括 responseURL 属性 （该属性是返回响应的 URL）和 responseHeaders 属性 （该属性是表示响应返回的响应标题的 URLRequestHeader 对象的数组）。

从外部文档加载数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

构建动态应用程序时，从外部文件或从服务器端脚本加载数据会很有用。这样，您不必编辑或重新编译应用程序，即可生成动态应用程序。例如，如果您要构建一个 " 每日提示 " 应用程序，就可以编写一个服务器端脚本，该脚本每天从数据库检索一次随机提示并将其保存到文本文件。然后，应用程序可以加载静态文本文件的内容，而不必每次查询数据库。

下面的片断创建 URLRequest 和 URLLoader 对象，用于加载外部文本文件 params.txt 的内容：

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

默认情况下，如果您未定义请求方法，则 Flash Player 和 Adobe AIR 会使用 HTTP GET 方法加载内容。要使用 POST 方法发送请求，可使用静态常量 URLRequestMethod.POST 将 request.method 属性设置为 POST，如以下代码所示：

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

在运行时加载的外部文档 params.txt 包含以下数据：

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,December&dayNames
=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

该文件包含两个参数，即 monthNames 和 dayNames。每个参数包含一个逗号分隔列表，该列表被分析为字符串。可以使用 String.split() 方法将此列表拆分为数组。

不要将保留字或语言构造作为外部数据文件中的变量名称，因为这样做会使代码的读取和调试变得更困难。

加载数据后，将调度 complete 事件，随后就可以在 URLLoader 的 data 属性中使用外部文档的内容，如以下代码所示：

```
function completeHandler(event)
{
    var loader2 = event.target;
    air.trace(loader2.data);
}
```

如果远程文档包含名称 - 值对，则可以通过传入加载文件的内容，使用 URLVariables 类来分析数据，如下所示：

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

外部文件中的各个名称 - 值对都创建为 URLVariables 对象中的一个属性。在上面的代码范例中，变量对象中的各个属性都被视为字符串。如果名称 - 值对的值是一个项目列表，则可以通过调用 String.split() 方法将字符串转换为数组，如下所示：

```
var dayNameArray:Array = variables.dayNames.split(",");
```

如果从外部文本文件加载数值数据，可使用顶级函数 （如 int()、 uint() 或 Number()）将这些值转换为数值。

无需将远程文件的内容作为字符串加载和新建 URLVariables 对象，您可以将 URLLoader.dataFormat 属性设置为在 URLLoaderDataFormat 类中找到的静态属性之一。 URLLoader.dataFormat 属性有以下三个可能的值：

- URLLoaderDataFormat.BINARY — URLLoader.data 属性包含 ByteArray 对象中存储的二进制数据。
- URLLoaderDataFormat.TEXT — URLLoader.data 属性包含 String 对象中的文本。
- URLLoaderDataFormat.VARIABLES — URLLoader.data 属性包含 URLVariables 对象中存储的 URL 编码的变量。

下面的代码演示了如何通过将 URLLoader.dataFormat 属性设置为 URLLoaderDataFormat.VARIABLES，从而自动将加载的数据分析为 URLVariables 对象：

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
        var loader:URLLoader = URLLoader(event.target);
        trace(loader.data.dayNames);
        }
    }
}
```

注：URLLoader.dataFormat 的默认值为 URLLoaderDataFormat.TEXT。

如以下示例所示，从外部文件加载 XML 与加载 URLVariables 相同。可以创建 URLRequest 实例和 URLLoader 实例，然后使用它们下载远程 XML 文档。文件完全下载后，调度 Event.COMPLETE 事件，并将外部文件的内容转换为可使用 XML 方法和属性分析的 XML 实例。

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

与外部脚本进行通信

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

除了加载外部数据文件，还可以使用 URLVariables 类将变量发送到服务器端脚本并处理服务器的响应。这是非常有用的，例如，如果您正在编写游戏，想要将用户的得分发送到服务器以计算是否应添加到高分列表中，甚至想要将用户的登录信息发送到服务器以进行验证。服务器端脚本可以处理用户名和密码，向数据库验证用户名和密码，然后返回用户提供的凭据是否有效的确认。

下面的片断创建一个名为 variables 的 URLVariables 对象，该对象创建称为 name 的新变量。接下来，创建一个 URLRequest 对象，该对象指定变量要发送到的服务器端脚本的 URL。然后，设置 URLRequest 对象的 method 属性，以便将变量作为 HTTP POST 请求发送。为了将 URLVariables 对象添加到 URL 请求，需要将 URLRequest 对象的 data 属性设置为早先创建的 URLVariables 对象。最后，创建 URLLoader 实例并调用 URLLoader.load() 方法，此方法用于启动该请求。

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
    loader.load(request);
}
catch (error:Error)
{
    trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
    trace(event.target.data.welcomeMessage);
}
```

下面的代码包含上面的示例中使用的 Adobe ColdFusion® greeting.cfm 文档的内容：

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Web 服务请求

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

存在各种基于 HTTP 的 Web 服务。主要类型包括：

- REST
- XML-RPC
- SOAP

要在 ActionScript 3 中使用 Web 服务，应创建一个 URLRequest 对象，使用 URL 变量或 XML 文档构建 Web 服务调用，然后使用 URLLoader 对象将调用发送到服务。 Flex 框架包含诸多类，其中几个类使借助 Web 服务更加轻松，尤其非常适用于访问复杂的 SOAP 服务。从 Flash Professional CS3 开始，您就可以将 Flex 类用于在 Flash Professional 和 Flash Builder 中开发的应用程序。

在基于 HTML 的 AIR 应用程序中，您可以使用 URLRequest 和 URLLoader 类或 JavaScript XMLHttpRequest 类。如果需要，您还可以创建 SWF 库，将 Flex 框架的 Web 服务组件公开到您的 JavaScript 代码。

当应用程序在浏览器中运行时，您使用的 Web 服务只能位于调用 SWF 所在的 Internet 域，除非承载 Web 服务的服务器也承载允许从其他域访问的跨域策略文件。当跨域策略文件不可用时，通常使用通过自己的服务器代理请求这项技术。 Adobe Blaze DS 和 Adobe LiveCycle 支持 Web 服务代理。

在 AIR 应用程序中，当应用程序安全沙箱调用 Web 服务时不需要跨域策略文件。远程域从不为 AIR 应用程序内容提供服务，因此该域不能加入跨域策略阻止的攻击类型。在基于 HTML 的 AIR 应用程序中，应用程序安全沙箱中的内容可以生成跨域 XMLHttpRequest。您可以允许其他安全沙箱中的内容生成跨域 XMLHttpRequest，只要该内容加载到 iframe 中。

更多帮助主题

第 901 页的 " 网站控制 （策略文件） "

Adobe BlazeDS

Adobe LiveCycle ES2

REST 体系结构

XML-RPC

SOAP 协议

REST 样式 Web 服务请求

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

REST 样式 Web 服务使用 HTTP 方法动词指定基本动作，并使用 URL 变量指定动作详细信息。例如，请求获得某个项目的数据时可以使用 GET 动词和 URL 变量指定方法名称和项目 ID。生成的 URL 字符串可能是：

```
http://service.example.com/?method=getItem&id=d3452
```

要使用 ActionScript 访问 REST 样式 Web 服务，您可以使用 URLRequest、URLVariables 和 URLLoader 类。在 AIR 应用程序内的 JavaScript 代码中，您还可以使用 XMLHttpRequest。

在 ActionScript 中编程 REST 样式 Web 服务调用，通常包括下列步骤：

1. 创建 URLRequest 对象。
2. 针对请求对象设置服务 URL 和 HTTP 方法动词。
3. 创建 URLVariables 对象。
4. 将服务调用参数设置为变量对象的动态属性。
5. 将变量对象分配给请求对象的数据属性。
6. 使用 URLLoader 对象将调用发送到服务。
7. 处理由 URLLoader 调度的 complete 事件，指示服务调用已完成。还应该侦听可由 URLLoader 对象调度的多个错误事件。

例如，请考虑一种用于公开测试方法的 Web 服务，该方法将调用参数回显给请求者。可使用以下 ActionScript 代码调用此服务：

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;

private var requestor:URLLoader = new URLLoader();
public function restServiceCall():void
{
    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/" );
    request.method = URLRequestMethod.GET;

    //Add the URL variables
    var variables:URLVariables = new URLVariables();
    variables.method = "test.echo";
    variables.api_key = "123456ABC";
    variables.message = "Able was I, ere I saw Elba.";
    request.data = variables;

    //Initiate the transaction
    requestor = new URLLoader();
    requestor.addEventListener( Event.COMPLETE, httpRequestComplete );
    requestor.addEventListener( IOErrorEvent.IOERROR, httpRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, httpRequestError );
    requestor.load( request );
}
private function httpRequestComplete( event:Event ):void
{
    trace( event.target.data );
}

private function httpRequestError( error:ErrorEvent ):void{
    trace( "An error occured: " + error.message );
}
```

在 AIR 应用程序内的 JavaScript 中，您可以使用 XMLHttpRequest 对象提出相同请求：

```
<html>
<head><title>RESTful web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    //Create a conveninece object to hold the call properties
    var request = {};
    request.URL = "http://service.example.com/";
    request.method = "test.echo";
    request.HTTPmethod = "GET";
    request.parameters = {};
    request.parameters.api_key = "ABCDEF123";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestURL = makeURL( request );
    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, requestURL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
```

```
            resultDisplay.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);

    requestDisplay.innerHTML = requestURL;
}
//Convert the request object into a properly formatted URL
function makeURL( request )
{
    var url = request.URL + "?method=" + escape( request.method );
    for( var property in request.parameters )
    {
        url += "&" + property + "=" + escape( request.parameters[property] );
    }

    return url;
}
</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<div id="request"></div>
<h1>Result:</h1>
<div id="result"></div>
</body>
</html>
```

XML-RPC Web 服务请求

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

XML-RPC Web 服务将其调用参数看作 XML 文档而不是一组 URL 变量。要使用 XML-RPC Web 服务执行事务，请创建格式正确的 XML 消息并使用 HTTP POST 方法将其发送到 Web 服务。此外，您应该为请求设置 Content-Type 标题，以便服务器将请求数据看作 XML。

下面的示例演示如何使用与 REST 示例中显示的相同的 Web 服务调用，但这次是 XML-RPC 服务：

```
import flash.events.Event;
import flash.events.ErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.SecurityErrorEvent;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.net.URLRequestMethod;
import flash.net.URLVariables;
public function xmlRPCRequest():void
{
    //Create the XML-RPC document
    var xmlRPC:XML = <methodCall>
                                    <methodName></methodName>
                                    <params>
                                        <param>
                                            <value>
                                                <struct/>
                                            </value>
                                        </param>
                                    </params>
                                </methodCall>;

    xmlRPC.methodName = "test.echo";
```

```
    //Add the method parameters
    var parameters:Object = new Object();
    parameters.api_key = "123456ABC";
    parameters.message = "Able was I, ere I saw Elba.";

    for( var propertyName:String in parameters )
    {
        xmlRPC..struct.member[xmlRPC..struct.member.length + 1] =
            <member>
                <name>{propertyName}</name>
                <value>
                    <string>{parameters[propertyName]}</string>
                </value>
            </member>;
    }

    //Create the HTTP request object
    var request:URLRequest = new URLRequest( "http://service.example.com/xml-rpc/" );
    request.method = URLRequestMethod.POST;
    request.cacheResponse = false;
    request.requestHeaders.push(new URLRequestHeader("Content-Type", "application/xml"));
    request.data = xmlRPC;

    //Initiate the request
    requestor = new URLLoader();
    requestor.dataFormat = URLLoaderDataFormat.TEXT;
    requestor.addEventListener( Event.COMPLETE, xmlRPCRequestComplete );
    requestor.addEventListener( IOErrorEvent.IO_ERROR, xmlRPCRequestError );
    requestor.addEventListener( SecurityErrorEvent.SECURITY_ERROR, xmlRPCRequestError );
    requestor.load( request );
}

private function xmlRPCRequestComplete( event:Event ):void
{
    trace( XML(event.target.data).toXMLString() );
}

private function xmlRPCRequestError( error:ErrorEvent ):void
{
    trace( "An error occurred: " + error );
}
```

AIR 中的 WebKit 不支持 E4X 语法，因此在上一示例中用于创建 XML 文档的方法在 JavaScript 代码中不可用。您必须使用 DOM 方法创建 XML 文档或将文档创建为字符串，并使用 JavaScript DOMParser 类将字符串转换为 XML。

在以下示例中，使用 DOM 方法创建 XML-RPC 消息和 XMLHttpRequest 来执行 Web 服务事务：

```
<html>
<head>
<title>XML-RPC web service request</title>
<script type="text/javascript">

function makeRequest()
{
    var requestDisplay = document.getElementById( "request" );
    var resultDisplay  = document.getElementById( "result" );

    var request = {};
    request.URL = "http://services.example.com/xmlrpc/";
    request.method = "test.echo";
    request.HTTPmethod = "POST";
    request.parameters = {};
    request.parameters.api_key = "123456ABC";
    request.parameters.message = "Able was I ere I saw Elba.";
    var requestMessage = formatXMLRPC( request );

    xmlhttp = new XMLHttpRequest();
    xmlhttp.open( request.HTTPmethod, request.URL, true);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            resultDisplay.innerText = xmlhttp.responseText;
        }
    }
    xmlhttp.send( requestMessage );

    requestDisplay.innerText = xmlToString( requestMessage.documentElement );
}

//Formats a request as XML-RPC document
function formatXMLRPC( request )
{
    var xmldoc = document.implementation.createDocument( "", "", null );
    var root = xmldoc.createElement( "methodCall" );
    xmldoc.appendChild( root );
    var methodName = xmldoc.createElement( "methodName" );
    var methodString = xmldoc.createTextNode( request.method );
    methodName.appendChild( methodString );

    root.appendChild( methodName );

        var params = xmldoc.createElement( "params" );
        root.appendChild( params );

        var param = xmldoc.createElement( "param" );
        params.appendChild( param );
        var value = xmldoc.createElement( "value" );
        param.appendChild( value );
        var struct = xmldoc.createElement( "struct" );
        value.appendChild( struct );

        for( var property in request.parameters )
        {
            var member = xmldoc.createElement( "member" );
            struct.appendChild( member );

            var name = xmldoc.createElement( "name" );
            var paramName = xmldoc.createTextNode( property );
            name.appendChild( paramName )
            member.appendChild( name );
```

```
            var value = xmldoc.createElement( "value" );
            var type = xmldoc.createElement( "string" );
            value.appendChild( type );
            var paramValue = xmldoc.createTextNode( request.parameters[property] );
            type.appendChild( paramValue )
            member.appendChild( value );
        }
    return xmldoc;
}

//Returns a string representation of an XML node
function xmlToString( rootNode, indent )
{
    if( indent == null ) indent = "";
    var result = indent + "<" + rootNode.tagName + ">\n";
    for( var i = 0; i < rootNode.childNodes.length; i++)
    {
        if(rootNode.childNodes.item( i ).nodeType == Node.TEXT_NODE )
        {
            result += indent + "    " + rootNode.childNodes.item( i ).textContent + "\n";
        }
    }
    if( rootNode.childElementCount > 0 )
    {
        result += xmlToString( rootNode.firstElementChild, indent + "    " );
    }
    if( rootNode.nextElementSibling )
    {
        result += indent + "</" + rootNode.tagName + ">\n";
        result += xmlToString( rootNode.nextElementSibling, indent );
    }
    else
    {
        result += indent +"</" + rootNode.tagName + ">\n";
    }
    return result;
}

</script>
</head>
<body onload="makeRequest()">
<h1>Request:</h1>
<pre id="request"></pre>
<h1>Result:</h1>
<pre id="result"></pre>
</body>
</html>
```

SOAP Web 服务请求

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

SOAP 是在通用 XML-RPC Web 服务概念上构建的，提供了用于传输类型化数据的方法，这些方法虽然比较复杂但很丰富。通常， SOAP Web 服务提供 Web 服务描述语言文件 (WSDL)，其中可指定 Web 服务调用、数据类型和服务 URL。当 ActionScript 3 不为 SOAP 提供直接支持时，您可以 " 手动 " 构造一个 SOAP XML 消息，将其发送给服务器，然后分析结果。但是，对于最简单的 SOAP Web 服务以外的任何服务，您也许可以使用现有的 SOAP 库节省大量开发时间。

Flex 框架包含用于访问 SOAP Web 服务的库。在 Flash Builder 中， rpc.swc 库自动包含在 Flex 项目中，因为它是 Flex 框架的一部分。在 Flash Professional 中，您可以将 Flex framework.swc 和 rpc.swc 添加到项目的库路径，然后使用 ActionScript 访问 Flex 类。

更多帮助主题

在 Flash Professional 中使用 Flex Web 服务组件

Cristophe Coenraets：Android 的实时交易桌面程序

在其他应用程序中打开 URL

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以使用 navigateToURL() 函数在 Web 浏览器或其他应用程序中打开 URL。对于在 AIR 中运行的内容，navigateToURL() 函数在默认系统 Web 浏览器中打开页面。

对于作为此函数的 request 参数传递的 URLRequest 对象，仅使用 url 属性。

navigateToURL() 函数的第一个参数 （即 navigate 参数）是一个 URLRequest 对象 （请参阅第 696 页的 " 使用 URLRequest 类 "）。第二个参数是可选的 window 参数，您可以使用该参数指定窗口名称。例如，下面的代码打开 www.adobe.com 网页：

```
var url:String = "http://www.adobe.com";
var urlReq:URLRequest = new URLRequest(url);
navigateToURL(urlReq);
```

注：使用 navigateToURL() 函数时，运行时将使用 POST 方法的 URLRequest 对象 （其 method 属性设置为 URLRequestMethod.POST）视为使用 GET 方法。

使用 navigateToURL() 函数时，根据调用 navigateToURL() 函数的代码的安全沙箱，决定是否允许 URI 方案。

某些 API 允许在 Web 浏览器中启动内容。出于安全方面的考虑，当在 AIR 中使用这些 API 时禁止使用某些 URI 方案。禁止的方案列表取决于使用 API 的代码所在的安全沙箱。（有关安全沙箱的详细信息，请参阅 第 922 页的 "AIR 安全性 "。）

应用程序沙箱 （仅限 **AIR**）

任何 URI 方案均可用于 AIR 应用程序沙箱中运行的内容所启动的 URL。应用程序必须经过注册才能处理 URI 方案，否则该请求不起任何作用。许多计算机和设备上支持以下方案：

- http:
- https:
- file:
- mailto: — AIR 将这些请求指向注册的系统邮件应用程序
- sms: — AIR 将 sms: 请求定向到默认的短信应用程序。 URL 格式必须符合运行应用程序的系统约定。例如，在 Android 上， URI 方案必须小写。

  ```
  navigateToURL( new URLRequest( "sms:+15555550101") );
  ```

- tel: — AIR 将 tel: 请求定向到默认的电话拨号应用程序。URL 格式必须符合运行应用程序的系统约定。例如，在 Android 上， URI 方案必须小写。

  ```
  navigateToURL( new URLRequest( "tel:5555555555") );
  ```

- market: — AIR 将 market: 请求定向到通常在 Android 设备上支持的 Market 应用程序。

  ```
  navigateToURL( new URLRequest( "market://search?q=Adobe Flash") );
  navigateToURL( new URLRequest( "market://search?q=pname:com.adobe.flashplayer") );
  ```

如果操作系统允许，应用程序可以定义和注册自定义 URI 方案。您可以使用该方案创建 URL，以便从 AIR 启动该应用程序。

远程沙箱

允许以下方案。使用这些方案的方法与在 Web 浏览器中的用法相同。

- http:
- https:
- mailto: — AIR 将这些请求指向注册的系统邮件应用程序

所有其他 URL 方案已禁止。

只能与本地文件系统内容交互的沙箱

允许以下方案。使用这些方案的方法与在 Web 浏览器中的用法相同。

- file:
- mailto: — AIR 将这些请求指向注册的系统邮件应用程序

所有其他 URL 方案已禁止。

只能与远程内容交互的沙箱

允许以下方案。使用这些方案的方法与在 Web 浏览器中的用法相同。

- http:
- https:
- mailto: — AIR 将这些请求指向注册的系统邮件应用程序

所有其他 URL 方案已禁止。

受信任的本地沙箱

允许以下方案。使用这些方案的方法与在 Web 浏览器中的用法相同。

- file:
- http:
- https:
- mailto: — AIR 将这些请求指向注册的系统邮件应用程序

所有其他 URL 方案已禁止。

第 45 章：与其他 Flash Player 和 AIR 实例通信

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

LocalConnection 类支持在 Adobe® AIR® 应用程序之间以及在浏览器中运行的 SWF 内容之间进行通信。您也可以使用 LocalConnection 类在 AIR 应用程序与在浏览器中运行的 SWF 内容之间进行通信。使用 LocalConnection 类，您可以构建能在 Flash Player 和 AIR 实例之间共享数据的通用应用程序。

关于 LocalConnection 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

LocalConnection 类用于开发 SWF 文件，这些文件无需使用 fscommand() 方法或 JavaScript 即可向其他 SWF 文件发送指令。 LocalConnection 对象只能在同一客户端计算机上运行的 SWF 文件间进行通信，但是它们可以在不同的应用程序中运行。例如，虽然放映文件维护本地信息，而基于浏览器的 SWF 文件进行的是远程连接，但在浏览器中运行的 SWF 文件和在放映文件中运行的 SWF 文件可以共享信息。（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 — 即，放映文件嵌入在可执行文件中，因此不需要安装 Flash Player。）

可以使用 LocalConnection 对象在使用不同 ActionScript 版本的 SWF 之间进行通信：

- ActionScript 3.0 LocalConnection 对象可以与使用 ActionScript 1.0 或 2.0 创建的 LocalConnection 对象进行通信。
- ActionScript 1.0 或 2.0 LocalConnection 对象可以与使用 ActionScript 3.0 创建的 LocalConnection 对象进行通信。

Flash Player 可自动处理不同版本 LocalConnection 对象间的通信。

最简便的 LocalConnection 对象使用方法是只允许位于同一个域或同一 AIR 应用程序中的 LocalConnection 对象之间进行通信。这样，您就不必担心安全问题了。但如果您需要在不同域之间进行通信，则可采用多种方法来实施安全措施。有关详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中列出的 send() 方法的 connectionName 参数和 LocalConnection 类中的 allowDomain() 和 domain 条目的介绍。

可以使用 LocalConnection 对象在一个 SWF 文件中收发数据，但是 Adobe 不建议这样做。而推荐使用共享对象。

可以使用三种方式将回调方法添加到 LocalConnection 对象中：

- 使 LocalConnection 类成为子类，并添加方法。
- 将 LocalConnection.client 属性设置为实现方法的对象。
- 创建扩展 LocalConnection 的动态类，并动态附加方法。

添加回调方法的第一种方式是扩展 LocalConnection 类。您在自定义类中定义方法，而不是将它们动态添加到 LocalConnection 实例中。下面的代码说明了此方式：

```
package
{
    import flash.net.LocalConnection;
    public class CustomLocalConnection extends LocalConnection
    {
        public function CustomLocalConnection(connectionName:String)
        {
            try
            {
                connect(connectionName);
            }
            catch (error:ArgumentError)
            {
                // server already created/connected
            }
        }
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

要创建 CustomLocalConnection 类的新实例，您可以使用以下代码：

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

添加回调方法的第二种方式是使用 LocalConnection.client 属性。这包括创建自定义类和将新实例分配给 client 属性，如下面的代码所示：

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

LocalConnection.client 属性指示应调用的对象回调方法。在上面的代码中， client 属性设置为自定义类 CustomClient 的新实例。 client 属性的默认值是当前 LocalConnection 实例。如果有两个具有同一方法集但是行为不同的数据处理函数，则可以使用 client 属性 — 例如，在某个应用程序中，一个窗口中的按钮切换另一个窗口中的视图。

要创建 CustomClient 类，可以使用下面的代码：

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

添加回调方法的第三种方式是创建动态类并动态附加该方法，这与在早期版本的 ActionScript 中使用 LocalConnection 类非常相似，如下面的代码所示：

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

通过使用下面的代码，可以将回调方法动态添加到此类中：

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

不建议使用上面这种添加回调方法的方式，因为该代码不是非常易于移植。另外，使用此方法创建本地连接会导致性能问题，因为访问动态属性比访问密封属性慢得多。

isPerUser 属性

将 isPerUser 属性添加到 Flash Player (10.0.32) 和 AIR (1.5.2) 中，旨在解决多个用户登录到 Mac 计算机时发生的冲突。在其他操作系统上，将忽略此属性，因为本地连接始终限制为单个用户。在新代码中应该将 isPerUser 属性设置为 true。但是，目前的默认值为 false 以实现向后兼容。在以后的运行时版本中此默认值可能会更改。

在两个应用程序之间发送消息

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可使用 LocalConnection 类在不同的 AIR 应用程序之间以及在浏览器中运行的不同 Adobe® Flash® Player (SWF) 应用程序之间进行通信。还可使用此 LocalConnection 类在 AIR 应用程序和在浏览器中运行的 SWF 应用程序之间进行通信。

例如，可以在网页中包含多个 Flash Player 实例，或者让 Flash Player 实例从弹出窗口中的 Flash Player 实例检索数据。

以下代码定义了一个用作服务器的 LocalConnection 对象，负责接受从其他应用程序传入的 LocalConnection 调用：

```
package
{
    import flash.net.LocalConnection;
    import flash.display.Sprite;
    public class ServerLC extends Sprite
    {
        public function ServerLC()
        {
            var lc:LocalConnection = new LocalConnection();
            lc.client = new CustomClient1();
            try
            {
                lc.connect("conn1");
            }
            catch (error:Error)
            {
                trace("error:: already connected");
            }
        }
    }
}
```

此代码首先创建一个名为 lc 的 LocalConnection 对象，然后将 client 属性设置为对象 clientObject。当其他应用程序调用此 LocalConnection 实例中的方法时，运行时会在 clientObject 对象中查找此方法。

如果已存在具有指定名称的连接，则会引发 Argument Error 异常，指出由于已经连接了该对象，连接尝试失败。

当 Flash Player 实例连接到此 SWF 文件并尝试调用指定本地连接的任何方法时，系统会将请求发送到 client 属性指定的类（该属性被设置为 CustomClient1 类）：

```
package
{
    import flash.events.*;
    import flash.system.fscommand;
    import flash.utils.Timer;
    public class CustomClient1 extends Object
    {
        public function doMessage(value:String = ""):void
        {
            trace(value);
        }
        public function doQuit():void
        {
            trace("quitting in 5 seconds");
            this.close();
            var quitTimer:Timer = new Timer(5000, 1);
            quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

若要创建 LocalConnection 服务器，请调用 LocalConnection.connect() 方法并提供唯一的连接名称。如果已存在具有指定名称的连接，则会生成 ArgumentError 错误，指出由于已经连接了该对象，连接尝试失败。

以下代码段说明如何创建名为 conn1 的 LocalConnection：

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

从辅助应用程序连接到主应用程序要求您首先在发送 LocalConnection 对象中创建一个 LocalConnection 对象；然后使用连接名称和要执行的方法名称来调用 LocalConnection.send() 方法。例如，要向您早期创建的 LocalConnection 对象发送 doQuit 方法，可使用以下代码：

```
sendingConnection.send("conn1", "doQuit");
```

此代码使用连接名称 conn1 连接到现有 LocalConnection 对象，并调用远程应用程序中的 doMessage() 方法。如果想要将参数发送到远程应用程序，可以在 send() 方法中的方法名称后指定附加参数，如以下代码段所示：

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

连接到不同域中的内容和 AIR 应用程序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要只允许从特定域进行通信，可以调用 LocalConnection 类的 allowDomain() 或 allowInsecureDomain() 方法，并传递包含允许访问此 LocalConnection 对象的一个或多个域的列表，以便传递允许的一个或多个域名。

在早期版本的 ActionScript 中， LocalConnection.allowDomain() 和 LocalConnection.allowInsecureDomain() 是必须由开发人员实现的、且必须返回布尔值的回调方法。在 ActionScript 3.0 中， LocalConnection.allowDomain() 和 LocalConnection.allowInsecureDomain() 都是内置方法，开发人员可以像调用 Security.allowDomain() 和 Security.allowInsecureDomain() 那样调用这两个内置方法，传递要允许的一个或多个域的名称。

Flash Player 8 对本地 SWF 文件引入了安全限制。可以访问 Internet 的 SWF 文件还不能访问本地文件系统。如果指定 localhost，则任何本地 SWF 文件都可以访问 SWF 文件。如果 LocalConnection.send() 方法试图从调用代码没有访问权限的安全沙箱与 SWF 文件进行通信，则会调度 securityError 事件 (SecurityErrorEvent.SECURITY_ERROR)。若要解决此错误，可以在接收方的 LocalConnection.allowDomain() 方法中指定调用方的域。

可以向 LocalConnection.allowDomain() 和 LocalConnection.allowInsecureDomain() 方法传递两个特殊值：* 和 localhost。星号值 (*) 表示允许从所有域访问。字符串 localhost 允许将从应用程序资源目录之外的本地安装内容调用应用程序。

如果 LocalConnection.send() 方法尝试从调用代码没有访问权限的安全沙箱与应用程序进行通信，则会调度 securityError 事件 (SecurityErrorEvent.SECURITY_ERROR)。若要解决此错误，可以在接收方的 LocalConnection.allowDomain() 方法中指定调用方的域。

如果仅在同一个域中的内容之间实现通信，可以指定一个不以下划线 (_) 开头且不指定域名的 connectionName 参数 （例如 myDomain:connectionName）。在 LocalConnection.connect(connectionName) 命令中使用相同的字符串。

如果要实现不同域中的内容之间的通信，可以指定一个以下划线开头的 connectionName 参数。指定下划线使具有接收方 LocalConnection 对象的内容更易于在域之间移植。下面是两种可能的情形：

- 如果 connectionName 字符串不以下划线开头，则运行时会添加一个包含超级域名和冒号的前缀 （例如 myDomain:connectionName）。虽然这可以确保您的连接不会与其他域中具有同一名称的连接冲突，但任何发送方 LocalConnection 对象都必须指定此超级域 （例如 myDomain:connectionName）。如果将具有接收方 LocalConnection 对象的 HTML 或 SWF 文件移动到另一个域中，则运行时会更改前缀，以反映新的超级域 （例如 anotherDomain:connectionName）。必须手动编辑所有发送方 LocalConnection 对象，以指向新超级域。
- 如果 connectionName 字符串以下划线开头 （例如 _connectionName），则运行时不会向该字符串添加前缀。这意味着接收方和发送方 LocalConnection 对象都将使用相同的 connectionName 字符串。如果接收方对象使用 LocalConnection.allowDomain() 来指定可以接受来自任何域的连接，则可以将具有接收方 LocalConnection 对象的 HTML 或 SWF 文件移动到另一个域，而无需更改任何发送方 LocalConnection 对象。

  在 connectionName 中使用下划线名称的缺点是存在潜在冲突，例如当两个应用程序使用同一 connectionName 同时尝试连接时。第二个相关缺点是安全方面的。使用下划线语法的连接名称不会标识侦听应用程序的域。出于这些原因，应优先选择使用域限定名称。

Adobe AIR

要与在 AIR 应用程序安全沙箱中运行的内容（随 AIR 应用程序一起安装的内容）通信，必须将可标识 AIR 应用程序的超级域用作连接名称的前缀。超级域字符串以 app# 开头，然后依次是应用程序 ID、点 (.) 字符和发行商 ID （如果已定义）。例如，在 ID 为 com.example.air.MyApp 且没有发行商 ID 的应用程序中， connectionName 参数中使用的正确超级域为 "app#com.example.air.MyApp"。因此，如果基础连接名称为 “appConnection”，则应在 connectionName 参数中使用的整个字符串为 "app#com.example.air.MyApp:appConnection"。如果应用程序包含发行商 ID，则必须将该 ID 也包含在超级域字符串中，即 "app#com.example.air.MyApp.B146A943FBD637B68C334022D304CEA226D129B4.1"。

当您允许其他 AIR 应用程序通过本地连接与您的应用程序通信时，必须调用 LocalConnection 对象的 allowDomain() 来传入本地连接域名。对于 AIR 应用程序，此域名的形式与连接字符串相同，都包含了应用程序 ID 和发行商 ID。例如，如果发送方 AIR 应用程序的应用程序 ID 为 com.example.air.FriendlyApp 且发行商 ID 为 214649436BD677B62C33D02233043EA236D13934.1，则用于允许此应用程序进行连接的域字符串为：app#com.example.air.FriendlyApp.214649436BD677B62C33D02233043EA236D13934.1。（自 AIR 1.5.3 起，并不是所有 AIR 应用程序都包含发行商 ID。）

第 46 章：与 AIR 中的本机进程通信

Adobe AIR 2 和更高版本

自 Adobe AIR 2 起，AIR 应用程序可以通过命令行运行并与其他本机进程通信。例如，AIR 应用程序可以运行进程，并通过标准的输入和输出流与之通信。

要与本机进程通信，请将 AIR 应用程序打包为通过本机安装程序安装。本机安装程序的文件类型取决于目标操作系统：

- 在 Mac OS 中为 DMG 文件。
- 在 Windows 中为 EXE 文件。
- 在 Linux 中为 RPM 或 DEB 包。

这些应用程序称为扩展的桌面配置文件应用程序。可以通过在使用 ADT 调用 -package 命令时指定 -target native 选项来创建本机安装程序文件。

更多帮助主题

flash.filesystem.File.openWithDefaultApplication()

flash.desktop.NativeProcess

本机进程通信概述

Adobe AIR 2 和更高版本

扩展的桌面配置文件中的 AIR 应用程序可以执行文件，执行方式与命令行调用该文件的方式相同。它可以与本机进程的标准流通信。标准流包括标准输入流 (stdin)、输出流 (stdout) 和标准错误流 (stderr)。

注：扩展的桌面配置文件中的应用程序还可以使用 File.openWithDefaultApplication() 方法启动文件和应用程序。然而，使用此方法不能为 AIR 应用程序提供访问标准流的权限。有关详细信息，请参阅第 584 页的 “ 使用默认系统应用程序打开文件 ”

以下代码示例显示如何启动应用程序目录中的 test.exe 应用程序。应用程序将参数 "hello" 作为命令行参数传递，并将事件侦听器添加到进程的标准输出流中：

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs.push("hello");
nativeProcessStartupInfo.arguments = processArgs;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, onOutputData);
process.start(nativeProcessStartupInfo);
public function onOutputData(event:ProgressEvent):void
{
    var stdOut:ByteArray = process.standardOutput;
    var data:String = stdOut.readUTFBytes(process.standardOutput.bytesAvailable);
    trace("Got: ", data);
}
```

启动和关闭本机进程

Adobe AIR 2 和更高版本

要启动本机进程，请设置 NativeProcessInfo 对象以执行下列操作：

- 指向您要启动的文件
- 存储命令行参数以在启动时传递到进程 (optional)
- 设置进程的工作目录 （可选）

若要开始本机进程，请将 NativeProcessInfo 对象作为 NativeProcess 对象的 start() 方法的参数进行传递。

例如，以下代码显示如何启动应用程序目录中的 test.exe 应用程序。应用程序传递参数 "hello"，并将用户的文档目录设置为工作目录：

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
var processArgs:Vector.<String> = new Vector.<String>();
processArgs[0] = "hello";
nativeProcessStartupInfo.arguments = processArgs;
nativeProcessStartupInfo.workingDirectory = File.documentsDirectory;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
```

若要终止进程，可调用 NativeProcess 对象的 exit() 方法。

如果希望可以在已安装应用程序中执行某个文件，请确保在您打包应用程序时，该文件在文件系统中是可执行文件。（在 Mac 和 Linux 上，如果需要，可以使用 chmod 来设置可执行标记。）

与本机进程通信

Adobe AIR 2 和更高版本

一旦 AIR 应用程序启动本机进程后，它就可以与该进程的标准输入、标准输出和标准错误流通信。

使用 NativeProcess 对象的下列属性流读取数据以及将数据写入流：

- standardInput — 包含访问标准输入流数据的权限。
- standardOutput — 包含对标准输出流数据的访问。
- standardError — 包含访问标准错误流数据的权限。

写入标准输入流

您可以使用 NativeProcess 对象的 standardInput 属性的写入方法将数据写入标准输入流。 AIR 应用程序将数据写入进程时，NativeProcess 对象会调度多个 standardInputProgress 事件。

如果写入标准输入流时出错， NativeProcess 对象会调度 ioErrorStandardInput 事件。

可以通过调用 NativeProcess 对象的 closeInput() 方法关闭输入流。关闭输入流后， NativeProcess 对象会调度一个 standardInputClose 事件。

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.start(nativeProcessStartupInfo);
process.standardInput.writeUTF("foo");
if(process.running)
{
    process.closeInput();
}
```

从标准输出流中读取

可使用此属性的读取方法从标准输出流中读取数据。AIR 应用程序从进程获取输出流数据时，NativeProcess 对象会调度多个 standardOutputData 事件。

如果写入标准输出流时出错， NativeProcess 对象会调度一个 standardOutputError 事件。

当进程关闭输出流时， NativeProcess 对象会调度 standardOutputClose 事件。

从标准输入流读取数据时，请确保在生成数据时读取数据。换句话说，为 standardOutputData 事件添加事件侦听器。在 standardOutputData 事件侦听器中，从 NativeProcess 对象的 standardOutput 属性读取数据。不要只等待 standardOutputClose 事件或 exit 事件读取所有数据。如果在本机进程生成数据时不读取数据，可能导致缓冲区填满或数据丢失。缓冲区填满可能会导致本机进程在尝试写入更多数据时终止。但是，如果您不为 standardOutputData 事件注册事件侦听器，则不会填充缓冲区，也不会终止进程。在这种情况下，您将无权访问数据。

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA, dataHandler);
process.start(nativeProcessStartupInfo);
var bytes:ByteArray = new ByteArray();
function dataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardOutput.readBytes(process.standardOutput.bytesAvailable);
}
```

从标准错误流中读取

可使用此属性的读取方法从标准错误流中读取数据。AIR 应用程序从进程中读取错误流数据时，NativeProcess 对象会调度多个 standardErrorData 事件。

如果写入标准错误流时出错， NativeProcess 对象会调度一个 standardErrorIoError 事件。

当进程关闭错误流时， NativeProcess 对象会调度 standardErrorClose 事件。

从标准错误流中读取数据时，请确保在生成数据时读取数据。换句话说，为 standardErrorData 事件添加事件侦听器。在 standardErrorData 事件侦听器中，从 NativeProcess 对象的 standardError 属性中读取数据。不要只等待 standardErrorClose 事件或 exit 事件读取所有数据。如果在本机进程生成数据时不读取数据，可能导致缓冲区填满或数据丢失。缓冲区填满可能会导致本机进程在尝试写入更多数据时终止。但是，如果您不为 standardErrorData 事件注册事件侦听器，则不会填充缓冲区，也不会终止进程。在这种情况下，您将无权访问数据。

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new NativeProcessStartupInfo();
var file:File = File.applicationDirectory.resolvePath("test.exe");
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
process.addEventListener(ProgressEvent.STANDARD_ERROR_DATA, errorDataHandler);
process.start(nativeProcessStartupInfo);
var errorBytes:ByteArray = new ByteArray();
function errorDataHandler(event:ProgressEvent):void
{
    bytes.writeBytes(process.standardError.readBytes(process.standardError.bytesAvailable);
}
```

本机进程通信的安全性注意事项

Adobe AIR 2 和更高版本

本机进程 API 可以在用户系统上运行任何可执行文件。构造和执行命令时要格外小心。如果要执行的命令的任何部分来自外部源，务必仔细验证该命令可以安全执行。同样， AIR 应用程序应对传递给正在运行的进程的任何数据进行验证。

但是，对输入进行验证可能十分困难。为了避免这种困难，最好编写具有特定 API 的本机应用程序 （如 Windows 上的 EXE 文件）。这些 API 应该只处理由应用程序定义的那些命令。例如，应用程序可能只通过标准输入流接受有限的一组指令。

Windows 上的 AIR 不允许直接运行 .bat 文件。命令解释器应用程序 (cmd.exe) 执行 Windows .bat 文件。调用 .bat 文件时，此命令应用程序可以将传递给该命令的参数解释为要启动的其他应用程序。在参数字符串中恶意注入额外字符可能导致 cmd.exe 执行有害的或不安全的应用程序。例如，在没有经过正确的数据验证的情况下， AIR 应用程序可能会调用 myBat.bat myArguments c:/evil.exe。该命令应用程序除了运行您的批处理文件外，还将启动 evil.exe 应用程序。

第 47 章：使用外部 API

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

借助 ActionScript 3.0 外部 API (flash.external.ExternalInterface)，可在 ActionScript 与在其中运行 Adobe Flash Player 的容器应用程序之间实现直接通信。使用 ExternalInterface API 可在 HTML 页面中的 SWF 文档和 JavaScript 之间创建交互。

您可以使用外部 API 与容器应用程序进行交互，在 HTML 页面中的 ActionScript 和 JavaScript 之间传递数据。

一些常见的外部 API 任务是：

- 获取有关容器应用程序的信息
- 使用 ActionScript 在浏览器中显示的网页或 AIR 桌面应用程序中调用代码
- 从网页中调用 ActionScript 代码
- 创建代理以简化从网页中调用 ActionScript 代码

注：有关外部接口的讨论仅涉及 SWF 文件中的 ActionScript 和容器应用程序之间的通信，该应用程序包含对 Flash Player 或已加载 SWF 文件的实例的引用。应用程序中其他任何使用 Flash Player 的情况都不在本文档的讨论范围之内。 Flash Player 用作浏览器插件或放映文件 （独立应用程序）。可能还在一定程度上支持其他使用方案。

在 **AIR** 中使用外部 **API**

由于 AIR 应用程序没有外部容器，因此通常不使用也不需要此外部接口。AIR 应用程序直接加载 SWF 文件时，应用程序代码可直接与 SWF 中的 ActionScript 代码通信 （受安全沙箱限制约束）。

但是，当 AIR 应用程序使用 HTMLLoader 对象中的 HTML 页面 （或 Flex 中的 HTML 组件）加载 SWF 文件时，该 HTMLLoader 对象将用作外部容器。因此，您可以使用外部接口在加载的 SWF 中的 ActionScript 代码和在 HTMLLoader 中加载的 HTML 页面中的 JavaScript 代码之间通信。

使用外部 API 的基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

虽然在某些情况下可以独立运行 SWF 文件 （例如，如果您使用 Adobe® Flash® Professional 来创建 SWF 放映文件），但在大多数情况下， SWF 应用程序作为另一个应用程序中的元素来运行。通常，包含 SWF 的容器是 HTML 文件；在少数情况下， SWF 文件用于台式机应用程序的全部或部分用户界面。

当处理更高级的应用程序时，您可能会发现，需要在 SWF 文件和容器应用程序之间建立通信。例如，网页通常使用 HTML 格式来显示文本或其他信息，并包含 SWF 文件以显示动态可视内容，如图表或视频。在这种情况下，您可能需要建立此类通信，以便当用户单击网页上的按钮时，它将更改 SWF 文件中的某些内容。ActionScript 包含一种称为外部 API 的机制，它有助于在 SWF 文件中的 ActionScript 与容器应用程序中的其他代码之间建立这种类型的通信。

重要概念和术语

以下参考列表包含与此功能相关的重要术语：

容器应用程序 Flash Player 在其中运行 SWF 文件的应用程序，如 Web 浏览器和包括 Flash Player 内容的 HTML 页面或在网页中加载 SWF 的 AIR 应用程序。

放映文件 一个包含 SWF 内容和 Flash Player 嵌入版本的可执行文件。您可以使用 Flash Professional 或独立的 Flash Player 来创建放映文件。放映文件通常用于以 CD-ROM 形式分发 SWF 文件，或在以下类似情况下进行 SWF 文件分发：下载文件大小不是问题，并且 SWF 作者希望确保用户能够运行 SWF 文件，而无论用户计算机上是否安装了 Flash Player。

代理 中介应用程序或代码，它代表一个应用程序 （“ 调用应用程序 ”）调用另一应用程序 （“ 外部应用程序 ”）中的代码，并将值返回到调用应用程序。可以出于各种不同的原因来使用代理，其中包括：

- 通过将调用应用程序中的本机函数调用转换为外部应用程序所理解的格式，简化进行外部函数调用的过程。
- 解决禁止调用方直接与外部应用程序进行通信的安全问题或其他限制问题。

串行化 将对象或数据值转换为某种格式，这种格式可用于通过消息在两个编程系统之间传递值，如通过 Internet 或在一台计算机上运行的两个不同应用程序之间进行传递。

完成示例

提供的许多代码示例是用于演示的较小的代码清单，而不是完整的工作示例或用于检查值的完整代码。由于使用外部 API 需要（根据定义）编写 ActionScript 代码以及容器应用程序中的代码，因此测试示例涉及创建一个容器 （例如，包含 SWF 文件的网页）和使用代码清单与该容器交互。

要测试 **ActionScript** 与 **JavaScript** 之间的通信的示例，请执行以下操作：

1. 使用 Flash Professional 创建一个新文档并将该文档保存到您的计算机上。
2. 从主菜单中，选择 “ 文件 ”>“ 发布设置 ”。
3. 在 “ 发布设置 ” 对话框中，在 “ 格式 ” 选项卡上确认选中了 “Flash” 和 “HTML” 复选框。
4. 单击 “ 发布 ” 按钮。这将在同一个文件夹中生成一个 SWF 文件和一个 HTML 文件，且两个文件的名称与您用于保存 文档的名称相同。单击 “ 确定 ” 以关闭 “ 发布设置 ” 对话框。
5. 取消选择 “HTML” 复选框。现在 HTML 页即生成，您将修改该页来添加适当的 JavaScript 代码。取消选择 “HTML” 复选框可确保在您修改 HTML 页之后， Flash 在发布 SWF 文件时不会使用新的 HTML 页覆盖您所做的更改。
6. 单击 “ 确定 ” 以关闭 “ 发布设置 ” 对话框。
7. 使用 HTML 或文本编辑器应用程序打开由 Flash 在您发布 SWF 文件时创建的 HTML 文件。在 HTML 源代码中，添加开始和结束 script 标签，并将示例代码清单中的 JavaScript 代码复制到其中：

```
<script>
// add the sample JavaScript code here
</script>
```

8. 保存该 HTML 文件并返回到 Flash。
9. 选择时间轴的第 1 帧中的关键帧，并打开 “ 动作 ” 面板。
10. 将 ActionScript 代码清单复制到 “ 脚本 ” 窗格中。
11. 从主菜单中，选择 “ 文件 ”>“ 发布 ” 以使用您所做的更改更新该 SWF 文件。
12. 使用 Web 浏览器打开您编辑过的 HTML 页，查看该页并测试 ActionScript 与 HTML 页之间的通信。

要测试 **ActionScript** 与 **ActiveX** 容器之间的通信的示例，请执行以下操作：

1. 使用 Flash Professional 创建一个新文档并将该文档保存到您的计算机上。您可能想要将它保存在容器应用程序预计能够在其中找到 SWF 文件的文件夹中。
2. 从主菜单中，选择 “ 文件 ”>“ 发布设置 ”。
3. 在 “ 发布设置 ” 对话框中，在 “ 格式 ” 选项卡上确认仅选中了 “Flash” 复选框。
4. 在“Flash”复选框旁边的“文件”字段中，单击文件夹图标以选择 SWF 文件将发布到的文件夹。通过设置 SWF 文件的位置，您可以 （举例而言）将文档放在一个文件夹中，而将发布的 SWF 文件放在另一个文件夹 （例如包含容器应用程序的源代码的文件夹）中。

5 选择时间轴的第 1 帧中的关键帧，并打开 " 动作 " 面板。

6 将示例的 ActionScript 代码复制到 " 脚本 " 窗格中。

7 从主菜单中，选择 " 文件 ">" 发布 " 以重新发布该 SWF 文件。

8 创建并运行您的容器应用程序，以测试 ActionScript 与容器应用程序之间的通信。

有关使用外部 API 与 HTML 页面通信的完整示例，请参阅以下主题：

- 第 727 页的 " 外部 API 示例：在 ActionScript 和 Web 浏览器中的 JavaScript 之间进行通信 "

这些示例包含完整代码，其中包括 ActionScript 和容器错误检查代码，您在使用外部 API 编写代码时将用到这些代码。有关使用外部 API 的另一个完整示例，请参阅 《ActionScript 3.0 参考》中 ExternalInterface 类的类示例。

外部 API 要求和优点

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

外部 API 是 ActionScript 中的一部分，它为在 ActionScript 与充当 Flash Player 容器的 " 外部应用程序 "（通常是 Web 浏览器或独立放映文件应用程序）中所运行的代码之间进行通信提供了一种机制。在 ActionScript 3.0 中，外部 API 的功能是由 ExternalInterface 类提供的。在 Flash Player 8 之前的 Flash Player 版本中，使用 fscommand() 动作与容器应用程序进行通信。 ExternalInterface 类取代了 fscommand()。

注：如需使用旧的 fscommand() 函数 （例如，为了与较早的应用程序保持兼容或与第三方 SWF 容器应用程序或独立的 Flash Player 进行交互），仍可将其作为 flash.system 包中的包级函数来使用。

ExternalInterface 类是一个子系统，通过该子系统可在 HTML 页面中的 ActionScript 和 Flash Player 与 JavaScript 之间进行轻松通信。

ExternalInterface 类只在以下情况下可用：

- 在所有受支持的 Internet Explorer for Windows 版本 （5.0 和更高版本）中
- 在支持 NPRuntime 接口的任何浏览器中（当前包括 Firefox 1.0 和更高版本、Mozilla 1.7.5 和更高版本、Netscape 8.0 和更高版本以及 Safari 1.3 和更高版本）。
- 在 AIR 应用程序中 （当 SWF 嵌入 HTMLLoader 控件显示的 HTML 页中时）。

在其他所有情况下 （例如，在独立的播放器中运行）， ExternalInterface.available 属性均返回 false。

从 ActionScript 中，可以在 HTML 页上调用 JavaScript 函数。与 fscommand() 相比，外部 API 可提供以下改进功能：

- 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；而不是仅限于传递一个命令和一个字符串参数。这为外部 API 提供了比 fscommand() 大得多的灵活性。
- 可以传递各种数据类型 （例如 Boolean、 Number 和 String）；不再仅限于 String 参数。
- 可以接收调用值，该值将立即返回到 ActionScript （作为进行的调用的返回值）。

重要说明：如果为 HTML 页中的 Flash Player 实例指定的名称 （object 标签的 id 属性）包含有连字符 (-) 或在 JavaScript 中定义为运算符的其他字符 （如 +、 *、 /、 \、 . 等等），在 Internet Explorer 中查看容器网页时，来自 ActionScript 的 ExternalInterface 调用将不起作用。另外，如果定义 Flash Player 实例 （object 和 embed 标签）的 HTML 标签在 HTML form 标签中嵌套，来自 ActionScript 的 ExternalInterface 调用将不起作用。

使用 ExternalInterface 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 与容器应用程序之间的通信方式有两种：ActionScript 可以调用容器中定义的代码 （如 JavaScript 函数），或者容器中的代码可以调用被指定为可调用函数的 ActionScript 函数。在这两种情况下，都可以将信息发送给被调用的代码，而将结果返回给执行调用的代码。

为了便于这种通信， ExternalInterface 类包含了两个静态属性和两个静态方法。这些属性和方法可用于获取有关外部接口连接的信息，从 ActionScript 执行容器中的代码，以及使 ActionScript 函数可供容器调用。

获取有关外部容器的信息

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ExternalInterface.available 属性指示当前的 Flash Player 是否位于提供外部接口的容器中。如果外部接口可用，则此属性为 true ；否则，为 false。在使用 ExternalInterface 类中的任何其他功能之前，应始终进行检查以确保当前容器支持外部接口通信，如下所示：

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

注：ExternalInterface.available 属性报告当前容器是否为支持 ExternalInterface 连接的容器类型。它不会报告当前浏览器中是否启用了 JavaScript。

通过使用 ExternalInterface.objectID 属性，您可以确定 Flash Player 实例的唯一标识符 （具体来说，是指 Internet Explorer 中 object 标签的 id 属性，或者是指使用 NPRuntime 接口的浏览器中 embed 标签的 name 属性）。这个唯一的 ID 代表浏览器中的当前 SWF 文档，并可用于对 SWF 文档进行引用 — 例如，在容器 HTML 页中调用 JavaScript 函数时进行引用。当 Flash Player 容器不是 Web 浏览器时，此属性为 null。

从 ActionScript 中调用外部代码

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ExternalInterface.call() 方法执行容器应用程序中的代码。它至少需要一个参数，即包含容器应用程序中要调用函数的名称的字符串。传递给 ExternalInterface.call() 方法的其他任何参数均作为函数调用的参数传递给容器。

```
// calls the external function "addNumbers"
// passing two parameters, and assigning that function's result
// to the variable "result"
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

如果容器为 HTML 页，此方法将调用具有指定名称的 JavaScript 函数，必须在包含 HTML 页中的 script 元素中定义该函数。JavaScript 函数的返回值被传递回 ActionScript。

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

如果容器为其他的 ActiveX 容器，此方法将导致 Flash Player ActiveX 控件调度它的 FlashCall 事件。Flash Player 将指定的函数名及所有参数序列化为一个 XML 字符串。容器可以在事件对象的 request 属性中访问该信息，并用它来确定如何执行它自己的代码。为了将值返回 ActionScript，容器代码会调用 ActiveX 对象的 SetReturnValue() 方法，并将结果（序列化为一个 XML 字符串）作为该方法的参数进行传递。有关用于此通信的 XML 格式的详细信息，请参阅第 726 页的 " 外部 API 的 XML 格式 "。

无论容器为 Web 浏览器还是为其他 ActiveX 容器，只要调用失败或容器方法没有指定返回值，都将返回 null。如果包含环境属于调用代码无权访问的安全沙箱， ExternalInterface.call() 方法将引发 SecurityError 异常。可以通过在包含环境中为 allowScriptAccess 设置合适的值来解决此问题。例如，要在 HTML 页中更改 allowScriptAccess 的值，请编辑 object 和 embed 标签中的相应属性。

从容器中调用 ActionScript 代码

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

容器只能调用函数中的 ActionScript 代码，不能调用任何其他 ActionScript 代码。若要从容器应用程序调用 ActionScript 函数，必须执行两项操作：向 ExternalInterface 类注册该函数，然后从容器的代码调用该函数。

首先，必须注册 ActionScript 函数，指示其应能够为容器所用。使用 ExternalInterface.addCallback() 方法，如下所示：

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

addCallback() 方法有两个参数。第一个参数为 String 类型的函数名，容器将籍此名称得知要调用的函数。第二个参数为容器调用定义的函数名时要执行的实际 ActionScript 函数。由于这些名称是截然不同的，因此可以指定将由容器使用的函数名，即使实际的 ActionScript 函数具有不同的名称。这在函数名未知的情况下特别有用 — 例如，指定了匿名函数或需要在运行时确定要调用的函数。

一旦向 ExternalInterface 类注册了 ActionScript 函数，容器就可以实际调用该函数。完成该操作的具体方法依容器的类型而定。例如，在 Web 浏览器的 JavaScript 代码中，使用已注册的函数名调用 ActionScript 函数，就像它是 Flash Player 浏览器对象的方法（即表示 object 或 embed 标签的 JavaScript 对象的方法）。也就是说，将传递参数并返回结果，就如同调用本地函数一样。

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

或者，在运行于台式机应用程序中的 SWF 文件中调用 ActionScript 函数时，必须将已注册的函数名及所有参数序列化为一个 XML 格式的字符串。然后，将该 XML 字符串作为一个参数来调用 ActiveX 控件的 CallFunction() 方法，以实际执行该调用。有关用于此通信的 XML 格式的详细信息，请参阅第 726 页的 " 外部 API 的 XML 格式 "。

不管是哪种情况， ActionScript 函数的返回值都被传递回容器代码，当调用方为浏览器中的 JavaScript 代码时直接作为值返回，而当调用方为 ActiveX 容器时则会序列化为 XML 格式字符串。

外部 API 的 XML 格式

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

ActionScript 与承载 Shockwave Flash ActiveX 控件的应用程序间的通信使用特定的 XML 格式对函数调用和值进行编码。外部 API 使用的 XML 格式分为两种。一种格式用于表示函数调用。另一种格式用于表示各个值；此格式用于函数中的参数及函数返回值。函数调用的 XML 格式用于对 ActionScript 的调用和来自 ActionScript 的调用。对于来自 ActionScript 的函数调用，Flash Player 将 XML 传递给容器；而对于来自容器的调用，Flash Player 需要容器应用程序将向其传递一个此格式的 XML 字符串。下面的 XML 片断说明了一个 XML 格式的函数调用示例：

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

根节点为 invoke 节点。它具有两个属性：name，指示要调用的函数的名称；以及 returntype，总是为 xml。如果函数调用包括参数，则 invoke 节点具有一个 arguments 子节点，该节点的子节点是使用单个值格式 （下面将予以说明）进行了格式设置的参数值。

每个值 （包括函数参数和函数返回值）均使用一个格式设置方案，除了实际值之外，该方案还包括数据类型信息。下表列出了 ActionScript 类以及用于对该数据类型的值进行编码的 XML 格式：

ActionScript 类 / 值	C# 类 / 值	格式	注释
null	null	<null/>	
Boolean true	bool true	<true/>	
Boolean false	bool false	<false/>	
String	string	<string> 字符串值 </string>	
Number、int、uint	single、double、int、uint	<number>27.5</number> <number>-12</number>	
Array （元素可以是混合类型）	允许混合类型元素的集合，如 ArrayList 或 object[]	<array> <property id="0"> <number>27.5</number> </property> <property id="1"> <string>Hello there!</string> </property> ... </array>	property 节点定义各个元素，而 id 属性为从零开始的数值索引。
Object	含有字符串键和对象值的字典，如具有字符串键的 HashTable	<object> <property id="name"> <string>John Doe</string> </property> <property id="age"> <string>33</string> </property> ... </object>	property 节点定义各个属性，而 id 属性为属性名称 （字符串）。
其他内置或自定义的类		<null/> or <object></object>	ActionScript 将其他对象编码为 null 或空对象。不管是哪种情况，所有属性值都会丢失。

注：该表举例说明了 ActionScript 类，并且还列出了等效 C# 类；但是，外部 API 可用来与支持 ActiveX 控件的任何编程语言或运行时进行通信，而不仅限于 C# 应用程序。

通过将外部 API 用于 ActiveX 容器应用程序来构建您自己的应用程序时，您可能会发现，编写代理以执行将本机函数调用转换为序列化 XML 格式的任务是非常方便的。有关使用 C# 编写的代理类的示例，请参阅深入 ExternalInterfaceProxy 类内部。

外部 API 示例：在 ActionScript 和 Web 浏览器中的 JavaScript 之间进行通信

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

本范例应用程序演示实现 ActionScript 与 Web 浏览器中的 JavaScript 间的通信的正确方法，范例的环境为一个允许用户相互对话的即时消息应用程序 （因此该应用程序的名称为：Introvert IM）。它使用外部 API 在网页中的 HTML 表单与 SWF 接口之间发送消息。本示例说明的方法包括：

- 通过验证浏览器在建立通信之前已准备就绪，正确地启动通信
- 检查容器是否支持外部 API
- 从 ActionScript 调用 JavaScript 函数，传递参数，并接收响应中的值
- 使 ActionScript 方法可由 JavaScript 调用，并执行这些调用

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。Introvert IM 应用程序文件位于 Samples/IntrovertIM_HTML 文件夹中。该应用程序包含以下文件：

文件	说明
IntrovertIMApp.fla 或 IntrovertIMApp.mxml	Flash 或 Flex 的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/introvertIM/IMManager.as	建立和管理 ActionScript 与容器间的通信的类。
com/example/programmingas3/introvertIM/IMMessageEvent.as	自定义事件类型，从容器接收到消息时由 IMManager 类调度。
com/example/programmingas3/introvertIM/IMStatus.as	枚举，其值表示可在该应用程序中选择的不同 “ 可用性 ” 状态值。
html-flash/IntrovertIMApp.html 或 html-template/index.template.html	Flash 应用程序的 HTML 页 (html-flash/IntrovertIMApp.html)，或用于为 Adobe Flex 应用程序创建容器 HTML 页的模板 (html-template/index.template.html)。此文件包含组成该应用程序的容器部分的所有 JavaScript 函数。

准备 ActionScript 与浏览器间的通信

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

外部 API 最常见的用途之一就是允许 ActionScript 应用程序与 Web 浏览器进行通信。使用外部 API 时，ActionScript 方法可以调用使用 JavaScript 编写的代码，反之亦然。由于浏览器的复杂性及其内部呈示页的方式，因此根本无法保证 SWF 文档能够在 HTML 页中的第一个 JavaScript 运行之前注册它的回调。出于这个原因，在从 JavaScript 调用 SWF 文档中的函数之前， SWF 文档应总是调用 HTML 页，以通知它 SWF 文档已准备好接受连接。

例如，通过 IMManager 类执行的一系列步骤， Introvert IM 可确定浏览器是否做好了通信的准备，并为通信准备 SWF 文件。第一个步骤 （确定浏览器是否已做好通信准备）是在 IMManager 构造函数中执行的，如下所示：

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // Check if the container is able to use the external API.
    if (ExternalInterface.available)
    {
        try
        {
            // This calls the isContainerReady() method, which in turn calls
            // the container to see if Flash Player has loaded and the container
            // is ready to receive calls from the SWF.
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // If the container is ready, register the SWF's functions.
                setupCallbacks();
            }
            else
            {
                // If the container is not ready, set up a Timer to call the
                // container at 100ms intervals. Once the container responds that
                // it's ready, the timer will be stopped.
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
    }
}
```

首先，代码使用 ExternalInterface.available 属性检查外部 API 是否还可在当前容器中使用。如果可用，它将开始建立通信的过程。由于尝试与外部应用程序通信会产生安全异常及其他错误，因此将代码包括在 try 块中 （为了简便起见，列表中省略了相应的 catch 块）。

然后，代码调用 isContainerReady() 方法，如下所示：

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

isContainerReady() 方法又使用 ExternalInterface.call() 方法来调用 JavaScript 函数 isReady()，如下所示：

```
<script language="JavaScript">
<!--
// ------- Private vars -------
var jsReady = false;
...
// ------- functions called by ActionScript -------
// called to check if the page has initialized and JavaScript is available
function isReady()
{
    return jsReady;
}
...
// called by the onload event of the <body> tag
function pageInit()
{
    // Record that JavaScript is ready to go.
    jsReady = true;
}
...
//-->
</script>
```

isReady() 函数仅返回 jsReady 变量的值。该变量最初为 false；一旦触发了网页的 onload 事件，该变量的值即更改为 true。也就是说，如果 ActionScript 在加载页之前调用 isReady() 函数，则 JavaScript 对 ExternalInterface.call("isReady") 返回 false，因此使得 ActionScript isContainerReady() 方法返回 false。加载页之后， JavaScript isReady() 函数将返回 true，从而使得 ActionScript isContainerReady() 方法也返回 true。

回到 IMManager 构造函数中，根据容器的准备情况，将发生两种情况之一。如果 isContainerReady() 返回 true，则代码调用 setupCallbacks() 方法，该方法将完成与 JavaScript 建立通信的过程。另一方面，如果 isContainerReady() 返回 false，实际上会使该过程陷入停顿状态。创建 Timer 对象，并指示其每隔 100 毫秒调用 timerHandler() 方法一次，如下所示：

```
private function timerHandler(event:TimerEvent):void
{
    // Check if the container is now ready.
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // If the container has become ready, we don't need to check anymore,
        // so stop the timer.
        Timer(event.target).stop();
        // Set up the ActionScript methods that will be available to be
        // called by the container.
        setupCallbacks();
    }
}
```

每次调用 timerHandler() 方法时，它都会再次检查 isContainerReady() 方法的结果。初始化容器后，该方法返回 true。然后，代码停止 Timer，并调用 setupCallbacks() 方法来完成与浏览器建立通信的过程。

向 JavaScript 公开 ActionScript 方法

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如上例所示，一旦代码确定浏览器已就绪，就将调用 setupCallbacks() 方法。此方法准备 ActionScript 以接收来自 JavaScript 的调用，如下所示：

```
private function setupCallbacks():void
{
    // Register the SWF client functions with the container
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // Notify the container that the SWF is ready to be called.
    ExternalInterface.call("setSWFIsReady");
}
```

setCallBacks() 方法通过调用 ExternalInterface.addCallback() 来注册两个可从 JavaScript 调用的方法，从而完成与容器进行通信的准备任务。在此代码中，第一个参数与 ActionScript 中方法的名称是相同的，该参数就是 JavaScript 所知道的方法的名称（"newMessage" 和 "getStatus"）。（在本例中，使用不同的名称毫无益处，因此为了简便起见，重复使用相同的名称。）最后，使用 ExternalInterface.call() 方法来调用 JavaScript 函数 setSWFIsReady()，该函数通知容器 ActionScript 函数已注册。

从 ActionScript 到浏览器的通信

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Introvert IM 应用程序说明一系列在容器页中调用 JavaScript 函数的示例。在最简单的情况下 （setupCallbacks() 方法的示例），调用 JavaScript 函数 setSWFIsReady() 时不传递任何参数也不接收返回值：

```
ExternalInterface.call("setSWFIsReady");
```

在 isContainerReady() 方法的另一个示例中， ActionScript 调用 isReady() 函数，并接受响应中的一个布尔值：

```
var result:Boolean = ExternalInterface.call("isReady");
```

还可以使用外部 API 将参数传递给 JavaScript 函数。以 IMManager 类的 sendMessage() 方法为例，该方法在用户向他 （或她）的 " 对话伙伴 " 发送新消息时调用：

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

再次使用 ExternalInterface.call() 调用指定的 JavaScript 函数，以通知浏览器有新的消息。另外，消息本身也作为另一个参数传递给 ExternalInterface.call()，从而作为参数传递给 JavaScript 函数 newMessage()。

从 JavaScript 调用 ActionScript 代码

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通信必须是双向的， Introvert IM 应用程序也不例外。不仅 Flash Player IM 客户端调用 JavaScript 来发送消息，而且 HTML 表单也调用 JavaScript 代码来向 SWF 文件发送消息和请求信息。例如，当 SWF 文件通知容器它已建立了联系并做好了通信的准备时，浏览器执行的第一个操作就是调用 IMManager 类的 getStatus() 方法，以检索 SWF IM 客户端的初始用户可用性状态。此操作是在网页中 （在 updateStatus() 函数中）执行的，如下所示：

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

代码检查 swfReady 变量的值，该变量跟踪 SWF 文件是否已经通知浏览器，它已在 ExternalInterface 类中注册了其方法。如果 SWF 文件已准备好接收通信，则下一行代码 (var currentStatus = ...) 实际调用 IMManager 类中的 getStatus() 方法。该行代码中执行三个操作：

- 调用 getSWF() JavaScript 函数，返回对表示 SWF 文件的 JavaScript 对象的引用。当 HTML 页中存在多个 SWF 文件时，传递给 getSWF() 的参数决定了返回哪一个浏览器对象。传递给该参数的值必须与 object 标签的 id 属性以及用于包括 SWF 文件的 embed 标签的 name 属性相匹配。
- 在使用对 SWF 文件的引用时， getStatus() 方法就像是 SWF 对象的方法一样被调用。在这种情况下，使用函数名 “getStatus”，因为它是使用 ExternalInterface.addCallback() 注册的 ActionScript 函数名。
- getStatus() ActionScript 方法返回一个值，该值被赋给 currentStatus 变量，该变量作为 status 文本字段的内容 （value 属性）被赋值。

注：如果您仔细查看这段代码，可能会注意到在 updateStatus() 函数的源代码中，调用 getSWF() 函数的代码行实际编写如下：var currentStatus = getSWF("${application}").getStatus()； ${application} 文本是 HTML 页面模板中的占位符；在 Adobe Flash Builder 为应用程序生成实际的 HTML 页面时，此占位符文本会由用作 object 标签的 id 属性和 embed 标签的 name 属性的相同文本所替换 （在本示例中为 IntrovertIMApp）。这是 getSWF() 函数所需的值。

sendMessage() JavaScript 函数说明如何将参数传递给 ActionScript 函数。（sendMessage() 是用户按 HTML 页上的 “ 发送 ” 按钮时调用的函数。）

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

newMessage() ActionScript 方法需要使用一个参数，因此将 JavaScript message 变量作为 JavaScript 代码中 newMessage() 方法调用中的参数传递给 ActionScript。

检测浏览器类型

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

由于各种浏览器在访问内容的方式上存在差异，所以必须总是使用 JavaScript 来检测用户正在运行哪种浏览器，并根据特定于浏览器的语法使用窗口对象或文本对象访问影片，如本例中的 getSWF() JavaScript 函数所示：

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

如果脚本不检测用户的浏览器类型，则在 HTML 容器中播放 SWF 文件时，用户可能会遇到意外情况。

第 48 章：AIR 中的 XML 签名验证

Adobe AIR 1.5 和更高版本

使用 Adobe® AIR® XMLSignatureValidator API 中的类验证数字签名是否符合针对 XML 签名语法和处理的 W3C 建议的子集 (http://www.w3.org/TR/xmldsig-core/)。 XML 签名可用于帮助验证数据或信息的完整性和签名者身份。

XML 签名可用于验证应用程序所下载的消息或资源。例如，如果应用程序按订阅提供服务，则可以将订阅条款封装在已签名的 XML 文档中。如果有人尝试更改订阅文档，验证就会失败。

还可以使用 XML 签名帮助验证应用程序所使用的下载的资源，具体方法是加入一个包含这些资源的摘要的已签名清单。应用程序通过将已签名文件中的摘要与根据所加载字节计算得出的摘要进行比较，可以验证资源是否尚未被更改。当所下载的资源是 SWF 文件或其他要在应用程序安全沙箱中运行的活动内容时，这种做法很有用。

XML 签名验证的基础知识

Adobe AIR 1.5 和更高版本

有关验证 XML 签名的快速介绍和代码示例，请参阅 Adobe Developer Connection 中的以下快速入门文章：

- 创建和验证 XML 签名 (Flex)
- 创建和验证 XML 签名 (Flash)

Adobe® AIR® 提供了用于验证 XML 签名的 XMLSignatureValidator 类和 IURIDereferencer 接口。XMLSignatureValidator 类接受的 XML 语法是针对 XML 签名语法和处理的 W3C 建议的子集。（由于仅支持该建议的子集，因此并不是可以验证所有合法签名。） AIR 未提供用于创建 XML 签名的 API。

XML 签名验证类

Adobe AIR 1.5 和更高版本

XML 签名验证 API 包含以下类：

包	类
flash.security	• XMLSignatureValidator • IURIDereferencer （接口） XMLSignatureValidator 字符串常量在以下类中定义： • ReferencesValidationSetting • RevocationCheckSettings • SignatureStatus • SignerTrustSettings
flash.events	• Event • ErrorEvent

使用 XML 签名验证类

Adobe AIR 1.5 和更高版本

若要使用 XMLSignatureValidator 类验证 XML 签名，您必须：

- 创建 XMLSignatureValidator 对象
- 提供 IURIDereferencer 接口的实现。XMLSignatureValidator 对象调用 IURIDereferencer dereference() 方法，将签名中的每个引用传入 URI。dereference() 方法必须解析 URI 并返回引用的数据（该数据可能与签名位于同一文档中，也可能位于外部资源中）。
- 设置与您的应用程序相适应的 XMLSignatureValidator 对象的证书信任、吊销检查和引用验证设置。
- 为 complete 和 error 事件添加事件侦听器。
- 调用 verify() 方法，传入要验证的签名。
- 处理 complete 和 error 事件并解释结果。

以下示例实现用于验证 XML 签名有效性的 validate() 函数。设置 XMLSignatureValidator 属性，以便签名证书必须位于系统信任存储区中，或者链接到信任存储区中的证书。该示例还假定存在一个合适的名为 XMLDereferencer 的 IURIDereferencer 类。

```
private function validate( xmlSignature:XML ):void
{
    var verifier:XMLSignatureValidator = new XMLSignatureValidator();
    verifier.addEventListener(Event.COMPLETE, verificationComplete);
    verifier.addEventListener(ErrorEvent.ERROR, verificationError);
    try
    {
        verifier.uriDereferencer = new XMLDereferencer();

        verifier.referencesValidationSetting =
            ReferencesValidationSetting.VALID_IDENTITY;
        verifier.revocationCheckSetting = RevocationCheckSettings.BEST_EFFORT;
        verifier.useSystemTrustStore = true;

        //Verify the signature
        verifier.verify( xmlSignature );
    }
    catch (e:Error)
        {
            trace("Verification error.\n" + e);
        }
}

//Trace verification results
private function verificationComplete(event:Event):void

    var signature:XMLSignatureValidator = event.target as XMLSignatureValidator;
    trace("Signature status: " + signature.validityStatus + "\n");
    trace("  Digest status: " + signature.digestStatus + "\n");
    trace("  Identity status: " + signature.identityStatus + "\n");
    trace("  Reference status: " + signature.referencesStatus + "\n");
}

private function verificationError(event:ErrorEvent):void
{
    trace("Verification error.\n" + event.text);
}
```

XML 签名验证过程

Adobe AIR 1.5 和更高版本

当调用 XMLSignatureValidator verify() 方法时， AIR 将执行以下步骤：

- 运行时使用签名证书的公钥来验证签名的加密完整性。
- 运行时根据 XMLSignatureValidator 对象的当前设置建立证书的加密完整性、标识和信任度。

  放置在签名证书中的信任关系是验证过程完整性的关键。签名验证是使用良好定义的加密过程执行的，但签名证书的信任度是一种判断，无法通过算法计算。

  通常，可以使用三种方法确定证书是否可信：

  - 通过依靠证书颁发机构和操作系统信任存储区。
  - 直接从签名者获取证书副本 （该证书副本是充当证书信任锚的另一证书）或者可靠地标识该证书的足够信息 （如公钥）。
  - 如果最终用户信任该证书，则向他们询问您的应用程序。此类查询对自签名证书无效，因为证书中的标识信息本身并不可靠。
- 运行时验证已签名数据的加密完整性。

  签名的数据是借助 IURIDereferencer 实现来验证的。对于签名文档中的每个引用，调用 IURIDereferencer 实现的 dereference() 方法。 dereference() 方法返回的数据用于计算引用摘要。此摘要值与签名文档中记录的摘要进行比较。如果摘要匹配，则该数据自签名以来没有被更改。

  在依赖验证 XML 签名的结果时，需要重点考虑的一点是只有对什么签名才是安全的。例如，考虑使用列出包中文件的签名清单。当 XMLSignatureValidator 验证签名时，它仅检查清单本身是否未更改。而文件中的数据未被签名，因此当清单中引用的文件被更改或删除时，仍将验证签名。

  注：若要验证此类清单中的文件，可以计算文件数据的摘要 （使用与清单相同的哈希算法），并将结果与签名清单中存储的摘要进行比较。在某些情况下，还应检查其他文件的存在。

解释验证结果

Adobe AIR 1.5 和更高版本

验证结果由 XMLSignatureValidator 对象的状态属性报告。在验证程序对象调度 complete 事件后，可以读取这些属性。这四个状态属性是：validityStatus、 digestStatus、 identityStatus 和 referencesStatus。

validityStatus 属性

Adobe AIR 1.5 和更高版本

validityStatus 属性报告签名的总体有效性。 validityStatus 取决于其他三种状态属性的状态，并且可以是下列值之一：

- valid — 如果 digestStatus、 identityStatus 和 referencesStatus 都是 valid。
- invalid — 如果一个或多个状态属性为 invalid。
- unknown — 如果一个或多个单独的状态属性为 unknown，且没有单独的状态为 invalid。

digestStatus 属性

Adobe AIR 1.5 和更高版本

digestStatus 属性报告消息摘要的加密验证的结果。 digestStatus 属性可以是下列值之一：

- valid — 如果签名文档本身自签名以来未被更改。

- invalid — 如果签名文档已更改或格式不正确。
- unknown — 如果 verify() 方法未完成，且未出现错误。

identityStatus 属性

Adobe AIR 1.5 和更高版本

identityStatus 属性报告签名证书的状态。此属性的值取决于多个因素，其中包括：

- 证书的加密完整性
- 证书是否过期或被吊销
- 证书在当前计算机上是否受信任
- XMLSignatureValidator 对象的状态 （如，是否为构建信任链添加了其他证书，这些证书是否受信任以及 useSystemTrustStore 和 revocationCheckSettings 属性的值）

identityStatus 属性可以是下列值：

- valid — 若要使签名证书被认为是有效的，必须符合下列条件：
  - 签名证书必须未被更改。
  - 签名证书必须未到期或吊销 （除非签名中存在有效的时间戳）。如果签名没有时间戳，则只要对文档进行签名时证书有效，就将该证书视为有效。 （时间戳服务对时间戳进行签名所使用的证书必须与用户计算机上的受信任根证书有联系。）
  - 签名证书是受信任证书。如果证书位于系统信任存储区或者链接到系统信任存储区中的其他证书，并且将 useSystemTrustStore 属性设置为 true，该证书即是受信任证书。还可以使用 XMLSignatureValidator 对象的 addCertificate() 方法将证书指定为受信任证书。
  - 该证书实际上是签名证书。
- invalid — 该证书已过期或被吊销，并且在签名时不存在任何证明签名有效性的时间戳，或者证书已更改。
- unknown — 如果该证书没有失效，但也不被信任。例如，自签名证书将被报告为 unknown （除非明确被信任）。如果 verify() 方法未完成且未出现错误，或者由于签名摘要无效而未检查标识， identityStatus 也会被报告为 unknown。

referencesStatus 属性

Adobe AIR 1.5 和更高版本

referencesStatus 属性报告签名的 SignedData 元素中的引用的加密完整性。

- valid — 如果签名中每个引用的计算的摘要与 XML 签名中记录的相应摘要匹配。 valid 状态指示签名的数据未被更改。
- invalid — 如果计算的任何摘要与签名中相应的摘要都不匹配。
- unknown — 如果未检查引用摘要。如果整个签名摘要为 invalid 或者签名证书无效，则不检查引用。如果 identityStatus 为 unknown，则仅在 referencesValidationSetting 为 validOrUnknown 时才检查引用。

关于 XML 签名

Adobe AIR 1.5 和更高版本

XML 签名是使用 XML 语法表示的数字签名。XML 签名中的数据可用于验证签名的信息自签名以来是否未更改过。此外，当签名证书由受信任证书颁发机构颁发之后，可以通过公钥基础结构验证签名者的身份。

XML 签名可应用于任何类型的数字数据 （以二进制或 XML 格式）。 XML 签名一般用于如下目的：

- 检查外部或下载的资源是否被修改
- 验证消息是否来自已知来源
- 验证应用程序许可证或订阅权限

支持的 XML 签名语法

Adobe AIR 1.5 和更高版本

AIR 支持以下来自针对 XML 签名语法和处理的 W3C 建议的元素：

- 所有核心签名语法元素 （W3C 建议文档的第 4 节） — 但不完全支持 KeyInfo 元素
- KeyInfo 元素必须仅包含 X509Data 元素
- X509Data 元素必须仅包含 X509Certificate 元素
- SHA256 摘要方法
- RSA-SHA1 (PKCS1) 签名算法
- “ 不带注释的规范化 XML” 规范化方法和转换
- 封装的签名转换
- 时间戳

以下文档阐释典型的 XML 签名 （为简化本例，已删除大多数加密数据）：

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <Reference URI="URI_to_signed_data">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>uoo...vY=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>Ked...w==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>i7d...w==</X509Certificate>
        </X509Data>
    </KeyInfo>
</Signature>
```

签名的重要元素有：

- SignedInfo — 包含对签名的数据和在签名时计算的摘要值的引用。签名的数据本身可能包含在与 XML 签名相同的文档中，也可能在外部。
- SignatureValue — 包含使用签名者的私钥加密的 SignedInfo 元素的摘要。
- KeyInfo — 包含签名证书，以及建立信任链所需的任何其他证书。注意，尽管从技术角度来看 KeyInfo 元素是可选的，但如果不包括该元素， AIR 就无法验证签名。

有三种常规类型的 XML 签名：

- Enveloped — 此签名插入要签名的 XML 数据的内部。

- Enveloping — 签名的 XML 数据包含在 Signature 元素的 Object 元素中。
- Detached — 签名的数据位于 XML 签名的外部。签名的数据可能在外部文件中。另一种情况是，该数据可能位于与签名相同的 XML 文档中，只不过不是 Signature 元素的父元素或子元素。

XML 签名使用 URI 引用签名的数据。签名和验证应用程序必须使用相同的约定解析这些 URI。当使用 XMLSignatureValidator 类时，必须提供 IURIDereferencer 接口的实现。此实现负责解析 URI 并作为 ByteArray 对象返回签名的数据。使用在签名中生成摘要的同一算法对返回的 ByteArray 对象生成摘要。

证书和信任

Adobe AIR 1.5 和更高版本

证书由公钥、标识信息以及属于颁发证书的认证机构的一个或多个证书构成。

共有两种方法在证书中建立信任。可以通过直接从签名者获取证书的副本来建立信任，例如在物理媒体上，或者通过安全数字传输，如 SSL 事务。还可以依赖证书颁发机构来确定签名证书是否可信。

若要依赖证书颁发机构，签名证书必须由验证签名的计算机上信任的颁发机构颁发。大多数操作系统制造商都将许多证书颁发机构的根证书放置在操作系统的信任存储区中。用户还可以向存储区添加或从中删除证书。

即使证书是由受信任证书颁发机构颁发的，仍须确定该证书是否属于您信任的人。在许多使用案例中，此决定传递到最终用户。例如，当安装 AIR 应用程序时，AIR 安装程序在要求用户确认是否要安装应用程序时，将显示发行者证书的标识信息。在其他情况下，可能需要将公钥或其他证书信息与可接受的密钥列表进行比较。（此列表必须是安全的，它或者由自己签名，或者存储在 AIR 加密的本地存储区中，以便列表本身不会被篡改。）

注：虽然可以选择信任没有独立验证的签名证书 （如 " 自签名 " 证书），但这样将不会通过验证签名获得任何保证。不知道谁创建了签名，保证签名不被篡改几乎没有意义。签名可能是对有效签名的伪造。

证书过期和吊销

Adobe AIR 1.5 和更高版本

所有证书过期。证书还可以由颁发证书的认证机构吊销，例如，与证书关联的私钥泄露或被盗。如果使用过期或吊销的证书进行了签名，则会将该签名报告为无效签名，除非将时间戳包括在签名中。如果存在时间戳，则只要证书在签名时是有效的，XMLSignatureValidator 类将验证该签名。

时间戳是来自时间戳服务的签名数字消息，用于证明该数据在特定的时间和日期经过签名。时间戳由时间戳颁发机构颁发，并由时间戳颁发机构自己的证书进行签名。嵌入时间戳的时间戳颁发机构证书必须在当前计算机上受信任，时间戳才会被认为是有效的。 XMLSignatureValidator 不提供用于指定在验证时间戳时使用其他证书的 API。

实现 IURIDereferencer 接口

Adobe AIR 1.5 和更高版本

要验证 XML 签名，必须提供 IURIDereferencer 接口的实现。此实现负责解析 XML 签名文档的 Reference 元素中的 URI 并返回该数据，以便可以计算摘要。计算的摘要与签名中的摘要进行比较，以确定自创建签名以来引用的数据是否被更改。

注：基于 HTML 的 AIR 应用程序必须导入包含 ActionScript 实现的 SWF 库才能验证 XML 签名。在 JavaScript 中无法实现 IURIDereferencer 接口。

IURIDerefencer 接口有一个必须实现的单一方法 dereference(uri:String)。 XMLSignatureValidator 对象对签名中的每个引用调用此方法。此方法必须将数据返回 ByteArray 对象。

在大多数情况下，还需要添加允许取消引用程序对象定位被引用数据的属性或方法。例如，如果签名数据与签名位于同一文档中，则可以添加成员变量来提供对 XML 文档的引用。然后， dereference() 方法可以使用此变量和 URI 定位被引用的数据。同样，如果签名数据位于本地文件系统的目录中，则 dereference() 方法可能需要一个提供该目录路径的属性才能解析被引用的文件。

XMLSignatureValidator 完全依赖取消引用程序来解释 URI 字符串。“W3C Recommendation for XML Signature Syntax and Processing”（W3C 推荐的 XML 签名语法和处理标准）的第 4.3.3 节中给出了取消引用 URI 的标准规则。

在封装的签名中取消引用 URI

Adobe AIR 1.5 和更高版本

当生成封装的 XML 签名时，签名元素将被插入签名的数据中。例如，如果使用封装的签名结构对以下消息进行了签名，

```
<message>
    <data>...</data>
</message>
```

结果签名的文档将如下所示：

```
<message>
    <data>...</data>
     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <DigestValue>yv6...Z0Y=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>cCY...LQ==</SignatureValue>
        <KeyInfo>
            <X509Data>
                <X509Certificate>MII...4e</X509Certificate>
            </X509Data>
        </KeyInfo>
    </Signature>
</message>
```

注意，该签名包含单一的 Reference 元素，并使用空字符串作为其 URI。此上下文中的空字符串引用文档的根。

还应注意，转换算法指定应用了封装的签名转换。当应用了封装的签名转换后， XMLSignatureValidator 在计算简要之前将自动从文档中删除签名。这意味着在返回数据时取消引用程序不需要删除 Signature 元素。

以下示例阐释了用于封装的签名的取消引用程序：

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class EnvelopedDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var signedMessage:XML;

        public function EnvelopedDereferencer( signedMessage:XML )
        {
            this.signedMessage = signedMessage;
        }

        public function dereference( uri:String ):IDataInput
        {
            try
            {
                if( uri.length != 0 )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var data:ByteArray = new ByteArray();
                data.writeUTFBytes( signedMessage.toXMLString() );
                data.position = 0;
            }
            catch (e:Error)
                {
                var error:ErrorEvent =
                    new ErrorEvent("Ref error " + uri + " ", false, false, e.message);
                this.dispatchEvent(error);
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }
        }
    }
}
```

此取消引用程序类使用构造函数和参数 signedMessage 将封装的签名文档提供给 dereference() 方法。由于封装的签名中的引用始终引用签名的数据的根，因此 dereferencer() 方法可以将文档写入字节数组并将其返回。

在封装签名和断开的签名中取消引用 URI

Adobe AIR 1.5 和更高版本

当签名的数据与签名本身位于同一文档时，引用中的 URI 通常使用 XPath 或 XPointer 语法对签名的元素寻址。针对 XML 签名语法和处理的 W3C 建议仅建议使用此语法，因此应将您的实现基于预期会遇到的签名 （并添加足够的错误检查来妥善地处理不支持的语法）。

AIR 应用程序的签名是封装签名的一个示例。该应用程序中的文件列在 Manifest 元素中。 Manifest 元素使用字符串 “#PackageContents” 存入 Reference URI 属性中，该字符串引用 Manifest 元素的 Id：

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" Id="PackageSignature">
    <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/TR/xmldsig-core#rsa-sha1"/>
        <Reference URI="#PackageContents">
            <Transforms>
                <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <DigestValue>ZMGqQdaRKQc1HirIRsDpeBDlaElS+pPotdziIAyAYDk=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue Id="PackageSignatureValue">cQK...7Zg==</SignatureValue>
    <KeyInfo>
        <X509Data>
            <X509Certificate>MII...T4e</X509Certificate>
        </X509Data>
    </KeyInfo>
    <Object>
    <Manifest Id="PackageContents">
        <Reference URI="mimetype">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>0/oCb84THKMagtI0Dy0KogEu92TegdesqRr/clXct1c=</DigestValue>
        </Reference>
        <Reference URI="META-INF/AIR/application.xml">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>P9MqtqSdqcqnFgeoHCJysLQu4PmbUW2JdAnc1WLq8h4=</DigestValue>
        </Reference>
        <Reference URI="XMLSignatureValidation.swf">
            <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256">
            </DigestMethod>
            <DigestValue>OliRHRAgc9qt3Dk0m0Bi53Ur5ur3fAweIFwju74rFgE=</DigestValue>
        </Reference>
    </Manifest>
</Object>
</Signature>
```


用于验证此签名的取消引用程序必须从 Reference 元素获取包含 "#PackageContents" 的 URI 字符串，并将 Manifest 元素返回到 ByteArray 对象中。“#” 符号引用元素 ID 属性的值。

以下示例实现用于验证 AIR 应用程序签名的取消引用程序。此实现通过依赖于 AIR 签名的已知结构而保持了简单的结构。通用的取消引用程序要复杂得多。

```
package
{
    import flash.events.ErrorEvent;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;

    public class AIRSignatureDereferencer implements IURIDereferencer {
        private const XML_SIG_NS:Namespace =
            new Namespace( "http://www.w3.org/2000/09/xmldsig#" );
        private var airSignature:XML;

        public function AIRSignatureDereferencer( airSignature:XML ) {
            this.airSignature = airSignature;
        }

        public function dereference( uri:String ):IDataInput {
            var data:ByteArray = null;
            try
            {
                if( uri != "#PackageContents" )
                {
                    throw( new Error("Unsupported signature type.") );
                }
                var manifest:XMLList =
                    airSignature.XML_SIG_NS::Object.XML_SIG_NS::Manifest;
                data = new ByteArray();
                data.writeUTFBytes( manifest.toXMLString());
                data.position = 0;
            }
            catch (e:Error)
            {
                data = null;
                throw new Error("Reference not resolvable: " + uri + ", " + e.message);
            }
            finally
            {
                return data;
            }

        }
    }
}
```

当验证此类型的签名时，仅验证 Manifest 元素中的数据。根本未检查包中的实际文件。要检查包文件是否被篡改，必须读取此文件，计算 SHA256 摘要，并将结果与清单中记录的摘要进行比较。 XMLSignatureValidator 不自动检查此类二级引用。

注：提供本例仅为了阐释签名验证过程。在验证自签名的 AIR 应用程序中很少使用。如果应用程序已被篡改，则篡改代理只需删除该验证检查。

计算外部资源的摘要值

Adobe AIR 1.5 和更高版本

AIR 不包含用于计算 SHA256 摘要的内置函数，但 Flex SDK 包含一个 SHA256 实用程序类。SDK 还包含 Base64 编辑器实用程序类，该类在比较计算的摘要与签名中存储的摘要时非常有用。

以下示例函数读取并验证 AIR 包清单中的文件。

```
import mx.utils.Base64Encoder;
import mx.utils.SHA256;

private function verifyManifest( sigFile:File, manifest:XML ):Boolean
{
    var result:Boolean = true;
    var message:String = '';
    var nameSpace:Namespace = manifest.namespace();

    if( manifest.nameSpace::Reference.length() <= 0 )
    {
        result = false;
        message = "Nothing to validate.";
    }
    for each (var reference:XML in manifest.nameSpace::Reference)
    {
        var file:File = sigFile.parent.parent.resolvePath( reference.@URI );
        var stream:FileStream = new FileStream();
        stream.open(file, FileMode.READ);
        var fileData:ByteArray = new ByteArray();
        stream.readBytes( fileData, 0, stream.bytesAvailable );

        var digestHex:String = SHA256.computeDigest( fileData );
        //Convert hexidecimal string to byte array
        var digest:ByteArray = new ByteArray();
        for( var c:int = 0; c < digestHex.length; c += 2 ){
            var byteChar:String = digestHex.charAt(c) + digestHex.charAt(c+1);
            digest.writeByte( parseInt( byteChar, 16 ));
        }
        digest.position = 0;

        var base64Encoder:Base64Encoder = new Base64Encoder();
        base64Encoder.insertNewLines = false;
        base64Encoder.encodeBytes( digest, 0, digest.bytesAvailable );
        var digestBase64:String = base64Encoder.toString();
        if( digestBase64 == reference.nameSpace::DigestValue )
        {
            result = result && true;
            message += "   " + reference.@URI + " verified.\n";
        }
        else
        {
            result = false;
            message += " ---- " + reference.@URI + " has been modified!\n";
        }
        base64Encoder.reset();
    }
    trace( message );
    return result;
}
```

该函数循环访问 Manifest 元素中的所有引用。对于每个引用，将计算 SHA256 摘要，用 base64 格式对其编码，并与清单中的摘要进行比较。 AIR 包中的 URI 引用相对于应用程序目录的路径。基于签名文件的位置解析该路径，签名文件始终位于应用程序目录的 META-INF 子目录中。注意， Flex SHA256 类返回以十六进制字符表示的摘要字符串。此字符串必须转换为 ByteArray，其中包含由十六进制字符串表示的字节。

若要在 Flash CS4 中使用 mx.utils.SHA256 和 Base64Encoder 类，可以找到这些类并将其复制到您的应用程序开发目录中，或者使用 Flex SDK 编译包含这些类的库 SWF。

在引用外部数据的断开签名中取消引用 URI

Adobe AIR 1.5 和更高版本

当 URI 引用外部资源时，必须访问该数据并将其加载到 ByteArray 对象中。如果 URI 包含绝对 URL，则只需读取一个文件或请求一个 URL。如果 URI 包含到相对路径中，则 IURIDereferencer 实现必须包含一种方法，能够将该路径解析到签名文件中，这可能是较常见的情形。

以下示例使用构建取消引用程序实例时初始化的 File 对象作为解析签名文件的基础。

```
package
{
    import flash.events.ErrorEvent;
    import flash.events.EventDispatcher;
    import flash.filesystem.File;
    import flash.filesystem.FileMode;
    import flash.filesystem.FileStream;
    import flash.security.IURIDereferencer;
    import flash.utils.ByteArray;
    import flash.utils.IDataInput;
    public class RelativeFileDereferencer
        extends EventDispatcher implements IURIDereferencer
    {
        private var base:File;

        public function RelativeFileDereferencer( base:File )
        {
            this.base = base;
        }

        public function dereference( uri:String ):IDataInput
        {
            var data:ByteArray = null;
            try{
                var referent:File = this.base.resolvePath( uri );
                var refStream:FileStream = new FileStream();
                data = new ByteArray();
                refStream.open( referent, FileMode.READ );

                refStream.readBytes( data, 0, data.bytesAvailable );

            } catch ( e:Error ) {
                data = null;
                throw new Error("Reference not resolvable: " + referent.nativePath + ", " + e.message );
            } finally {
                return data;
            }
        }
    }
}
```

dereference() 函数找出由引用 URI 寻址的文件，将文件内容加载到字节数组中，并返回 ByteArray 对象。

注：验证远程外部引用之前，应该考虑您的应用程序是否容易遭受恶意制造的签名文档所发起的 “ 打电话回家 ”(phone home) 攻击或类似攻击。

第 49 章：客户端系统环境

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

此讨论阐述了如何与用户的系统进行交互。具体介绍了如何确定支持哪些功能，以及如何通过用户安装的输入法编辑器 (IME)（如果有）构建多语言应用程序。还将介绍应用程序域的典型用法。

更多帮助主题

flash.system.System

flash.system.Capabilities

客户端系统环境基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在构建更高级的应用程序时，您可能需要了解有关用户操作系统的详细信息并访问用户操作系统的功能。 flash.system 包包含一个类集合，使您可以访问系统级功能，例如：

- 确定应用程序和安全域代码的执行位置
- 确定用户的 Flash 运行时 （如 Flash® Player 或 Adobe® AIR™）实例的功能，如屏幕大小 （分辨率），以及某种功能 （如 mp3 音频）是否可用
- 使用 IME 建立多语言站点
- 与 Flash 运行时的容器 （可能是 HTML 页面或容器应用程序）交互。
- 将信息保存到用户的剪贴板中

flash.system 包还包括 IMEConversionMode 和 SecurityPanel 类。这两个类分别包含与 IME 和 Security 类一起使用的静态常量。

重要概念和术语

以下参考列表包含重要术语：

操作系统 计算机上运行的主要程序 （所有其他应用程序均在其中运行），如 Microsoft Windows、 Mac OS X 或 Linux®。

剪贴板 用于保存复制或剪切的文本或项目的操作系统容器，可从中将项目粘贴到应用程序中。

应用程序域 用于将不同 SWF 文件中使用的类分开的机制，以便在 SWF 文件包含具有相同名称的不同类时，这些类不会彼此覆盖。

IME （输入方法编辑器） 用于使用标准键盘输入复杂字符或符号的程序 （或操作系统工具）。

客户端系统 在编程术语中，客户端是在个人计算机上运行且由单个用户使用的应用程序部分 （或整个应用程序）。“ 客户端系统 ” 是指用户计算机上的基础操作系统。

使用 System 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以通过 System 类包含的一些方法和属性与用户的操作系统进行交互，并检索运行时当前的内存使用情况。还可以使用 System 类的方法和属性来侦听 imeComposition 事件、指示运行时使用用户的当前代码页加载外部文本文件或作为 Unicode 进行加载或设置用户剪贴板的内容。

在运行时获取有关用户系统的数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过检查 System.totalMemory 属性，可以确定运行时当前使用的内存量 （以字节为单位）。该属性可让您监视内存使用情况，并根据内存级别的更改方式优化应用程序。例如，如果特定视觉效果导致内存使用量大幅增加，您可能需要考虑修改此效果或将其完全消除。

System.ime 属性是对当前安装的输入法编辑器 (IME) 的引用。该属性允许使用 addEventListener() 方法来侦听 imeComposition 事件 (flash.events.IMEEvent.IME_COMPOSITION)。

System 类中的第三个属性是 useCodePage。当 useCodePage 设置为 true 时，运行时使用操作系统的传统代码页加载外部文本文件。如果将此属性设置为 false，就是告诉运行时按 Unicode 解释外部文件。

如果将 System.useCodePage 设置为 true，请记住，操作系统的传统代码页必须包括在外部文本文件中使用的字符，这样才能显示文本。例如，如果您加载了一个包含中文字符的外部文本文件，则这些字符不能显示在使用英文 Windows 代码页的系统上，因为该代码页不包括中文字符。

要确保所有平台上的用户都能查看您的应用程序中使用的外部文本文件，应使所有外部文本文件采用 Unicode 编码，并将 System.useCodePage 设置保留为默认设置 false。这样，运行时将按 Unicode 解释文本。

将文本保存到剪贴板

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

System 类包括一个名为 setClipboard() 的方法，该方法允许 Flash 运行时使用指定的字符串设置用户剪贴板的内容。出于安全方面的考虑，不存在 Security.getClipboard() 方法，因为这样的方法可能允许恶意站点访问最近复制到用户剪贴板中的数据。

下面的代码演示出现安全错误时如何将错误消息复制到用户剪贴板。如果用户要报告应用程序的潜在错误，则错误消息会很有用。

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Flash Player 10 和 **AIR 1.0**

您可使用 Clipboard 类读写剪贴板数据以响应用户事件。在 AIR 中，在应用程序沙箱中运行的代码访问剪贴板时不需要用户事件。

使用 Capabilities 类

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

开发人员可通过 Capabilities 类来确定正在运行应用程序的环境。使用 Capabilities 类的多种属性，您可以了解用户的系统分辨率，用户的系统是否支持辅助软件、用户操作系统的语言，以及当前安装的 Flash 运行时版本。

通过检查 Capabilities 类的属性，可以自定义应用程序，使其以最佳方式与特定用户环境配合工作。例如，通过检查 Capabilities.screenResolutionX 和 Capabilities.screenResolutionY 属性，可以确定用户系统所使用的显示分辨率，并确定最合适的视频大小。或者，在尝试加载外部 mp3 文件之前，可以检查 Capabilities.hasMP3 属性以查看用户系统是否支持 mp3 播放。

以下代码使用正则表达式分析客户端正在使用的 Flash 运行时版本：

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
    trace("input: " + result.input);
    trace("platform: " + result[1]);
    trace("majorVersion: " + result[2]);
    trace("minorVersion: " + result[3]);
    trace("buildNumber: " + result[4]);
    trace("internalBuildNumber: " + result[5]);
}
else
{
    trace("Unable to match RegExp.");
}
```

如果要向某个服务器端脚本发送用户的系统功能，以便将信息存储在数据库中，可以使用下面的 ActionScript 代码：

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

功能示例：检测系统功能

Flash Player 9 和更高版本

CapabilitiesExplorer 示例演示了如何使用 flash.system.Capabilities 类确定用户的 Flash 运行时版本支持的功能。该示例讲授以下方法：

- 使用 Capabilities 类检测用户的 Flash 运行时版本支持的功能
- 使用 ExternalInterface 类检测用户的浏览器支持哪些浏览器设置

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/CapabilitiesExplorer 文件夹中找到 CapabilitiesExplorer 应用程序文件。该应用程序包含下列文件：

文件	说明
CapabilitiesExplorer.fla 或 CapabilitiesExplorer.mxml	Flash 或 Flex 中的主应用程序文件 （分别为 FLA 和 MXML）。
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	用于提供应用程序的主要功能的类，这些主要功能包括在数组中添加系统功能、对项目进行排序以及使用 ExternalInterface 类检索浏览器功能。
capabilities.html	包含与外部 API 进行通信所必需的 JavaScript 的 HTML 容器。

CapabilitiesExplorer 概述

Flash Player 9 和更高版本

CapabilitiesExplorer.mxml 文件负责设置 CapabilitiesExplorer 应用程序的用户界面。用户的 Flash 运行时版本的功能将在舞台上的 DataGrid 组件实例内显示。如果用户是从 HTML 容器运行应用程序且外部 API 可用，也将显示其浏览器功能。

调度主应用程序文件的 creationComplete 事件时，将调用 initApp() 方法。 initApp() 方法从 com.example.programmingas3.capabilities.CapabilitiesGrabber 类中调用 getCapabilities() 方法。initApp() 方法的代码如下所示：

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

CapabilitiesGrabber.getCapabilities() 方法返回 Flash 运行时和浏览器功能的排序的数组，然后将其设置为舞台上的 capabilitiesGrid DataGrid 组件实例的 dataProvider 属性。

CapabilitiesGrabber 类概述

Flash Player 9 和更高版本

CapabilitiesGrabber 类的静态 getCapabilities() 方法将 flash.system.Capabilities 类中的每个属性添加到数组 (capDP)。然后调用 CapabilitiesGrabber 类中的静态 getBrowserObjects() 方法。getBrowserObjects() 方法使用外部 API 循环访问浏览器的导航器对象 （包含浏览器的功能）。 getCapabilities() 方法如下所示：

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable", value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility", value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

getBrowserObjects() 方法返回一个数组，该数组包含浏览器的导航器对象中的每个属性。如果该数组的长度为一个或多个项目，则将浏览器功能的数组 (navArr) 追加到 Flash Player 功能的数组 (capDP) 末尾，并按字母顺序对整个数组进行排序。最后，排序后的数组将返回到主应用程序文件，随后填充数据网格。 getBrowserObjects() 方法的代码如下所示：

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // ignore
        }
    }
    return itemArr;
}
```

如果当前用户环境中有外部 API，Flash 运行时将调用 JavaScript JS_getBrowserObjects() 方法，该方法循环访问浏览器的导航器对象并向 ActionScript 返回 URL 编码的值的字符串。该字符串随后将转换为 URLVariables 对象 (itemVars) 并添加到 itemArr 数组中，将向调用脚本返回该数组。

与 JavaScript 进行通信

Flash Player 9 和更高版本

作为构建 CapabilitiesExplorer 应用程序的最后一部分工作，要编写循环访问浏览器导航器对象中的每个项目所必需的 JavaScript，并将一个名称 / 值对追加到一个临时数组的末尾。container.html 文件中的 JavaScript JS_getBrowserObjects() 方法的代码如下所示：

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // Create an array to hold each of the browser's items.
        var tempArr = new Array();

        // Loop over each item in the browser's navigator object.
        for (var name in navigator)
        {
            var value = navigator[name];

            // If the current value is a string or Boolean object, add it to the
            // array, otherwise ignore the item.
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // Create a temporary string which will be added to the array.
                    // Make sure that we URL-encode the values using JavaScript's
                    // escape() function.
```

```
                    var tempStr = "navigator." + name + "=" + escape(value);
                    // Push the URL-encoded name/value pair onto the array.
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Loop over each item in the browser's screen object.
        for (var name in screen)
        {
            var value = screen[name];

            // If the current value is a number, add it to the array, otherwise
            // ignore the item.
            switch (typeof(value))
            {
                case "number":
                    var tempStr = "screen." + name + "=" + escape(value);
                    tempArr.push(tempStr);
                    break;
            }
        }
        // Return the array as a URL-encoded string of name-value pairs.
        return tempArr.join("&");
    }
</script>
```

代码首先创建一个临时数组，该数组用于保存导航器对象中的所有名称 / 值对。接下来，使用 for..in 循环对导航器对象进行循环访问，并计算出当前值的数据类型以过滤掉不需要的值。在此应用程序中，我们只需要 String 或 Boolean 值，其他数据类型 （例如函数或数组）将被忽略。导航器对象中的每个 String 或 Boolean 值将追加到 tempArr 数组末尾。然后，使用 for..in 循环对浏览器的屏幕对象进行循环访问，并将每个数值添加到 tempArr 数组中。最后，使用 Array.join() 方法将临时数组转换为字符串。该数组使用 (&) 符号作为分隔符，这使得 ActionScript 可以使用 URLVariables 类轻松分析数据。

第 50 章：AIR 应用程序的调用和终止

Adobe AIR 1.0 和更高版本

本节讨论几种对已安装的 Adobe® AIR® 应用程序进行调用的方法，以及关闭运行中的应用程序的选项和注意事项。

注：NativeApplication、InvokeEvent 和 BrowserInvokeEvent 对象只可用于 AIR 应用程序沙箱中运行的 SWF 内容。在 Flash Player 运行时、浏览器、独立播放器（放映文件）或应用程序沙箱之外的 AIR 应用程序中运行的 SWF 内容无法访问这些类。

有关调用和终止 AIR 应用程序的快速介绍和代码示例，请参阅 Adobe Developer Connection 上的以下快速入门文章：

- 启动选项

更多帮助主题

flash.desktop.NativeApplication

flash.events.InvokeEvent

flash.events.BrowserInvokeEvent

应用程序调用

Adobe AIR 1.0 和更高版本

在用户（或操作系统）执行以下操作时，将调用 AIR 应用程序：

- 从桌面解释程序启动该应用程序。
- 使用该应用程序作为命令行解释程序中的命令。
- 打开某类型文件而该应用程序是此类型文件的默认打开程序。
- (Mac OS X) 单击停靠任务栏中的该应用程序图标（无论应用程序当前是否正在运行）。
- 选择从安装程序启动该应用程序（在新安装过程结束时，或者在双击已安装应用程序的 AIR 文件之后）。
- 当已安装版本指示其自身正在处理应用程序更新（通过在应用程序描述符文件中加入 <customUpdateUI>true</customUpdateUI> 声明）时，开始更新 AIR 应用程序。
- (iOS) 从 Apple 推送通知服务 (APNs) 接收通知。
- 通过 URL 调用应用程序。
- 访问承载了将调用 com.adobe.air.AIR launchApplication() 方法（该方法可为 AIR 应用程序指定识别信息）的 Flash 标志或应用程序的网页。（要使浏览器调用成功，应用程序描述符还必须包含 <allowBrowserInvocation>true</allowBrowserInvocation> 声明。）

每当调用 AIR 应用程序时，AIR 都会通过单一 NativeApplication 对象调度类型为 invoke 的 InvokeEvent 对象。若要给应用程序留出时间来初始化自身并注册事件侦听器，将对 invoke 事件进行排队而非将其丢弃。一旦侦听器已注册，就会传送所有排队的事件。

注：使用浏览器调用功能来调用某个应用程序时，如果该应用程序尚未运行，则 NativeApplication 对象将仅调度一个 invoke 事件。

若要接收 invoke 事件，请调用 NativeApplication 对象 (NativeApplication.nativeApplication) 的 addEventListener() 方法。当某个事件侦听器为 invoke 事件进行注册后，该事件侦听器还会接收到在注册前发生的所有 invoke 事件。在对 addEventListener() 的调用返回后不久，将以短时间间隔一次调度一个排队的 invoke 事件。如果在此过程中发生了新的 invoke 事件，则可能会在一个或多个排队的事件之前调度该事件。通过该事件队列可处理在初始化代码执行之前发生的任何 invoke 事件。请记住，如果在执行后期 （在应用程序初始化之后）添加一个事件侦听器，该事件侦听器仍将会接收自应用程序启动以来发生的所有 invoke 事件。

仅启动 AIR 应用程序的一个实例。当再次调用某个已经运行的应用程序时， AIR 将向该正在运行的实例调度一个新的 invoke 事件。 AIR 应用程序负责响应 invoke 事件并采取适当的动作 （例如，打开一个新的文档窗口）。

InvokeEvent 对象包含任何传递给该应用程序的参数，以及已从中调用该应用程序的目录。如果该应用程序是由于文件类型关联而被调用，则文件的完整路径将包含在命令行参数中。同样，如果该应用程序是由于某个应用程序升级而被调用，则会提供升级 AIR 文件的完整路径。

当在一次操作中打开多个文件时，在 Mac OS X 中将调度一个 InvokeEvent 对象。每个文件都包括在 arguments 数组中。在 Windows 和 Linux 中将为每个文件调度一个单独的 InvokeEvent 对象。

应用程序可通过以下方法处理 invoke 事件：即向其 NativeApplication 对象注册侦听器，

```
NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvokeEvent);
```

然后定义事件侦听器：

```
 var arguments:Array;
var currentDir:File;
public function onInvokeEvent(invocation:InvokeEvent):void {
    arguments = invocation.arguments;
    currentDir = invocation.currentDirectory;
}
```

捕获命令行参数

Adobe AIR 1.0 和更高版本

与 AIR 应用程序调用关联的命令行参数是在由 NativeApplication 对象调度的 InvokeEvent 对象中进行传送的。InvokeEvent arguments 属性包含在调用 AIR 应用程序时由操作系统传递的参数数组。如果这些参数包含相对文件路径，则通常可以使用 currentDirectory 属性来解析路径。

除非用双引号引起来，否则传递给 AIR 程序的参数会视为以空白分隔的字符串：

参数	Array
tick tock	{tick,tock}
tick "tick tock"	{tick,tick tock}
"tick" “tock”	{tick,tock}
\"tick\" \"tock\"	{"tick","tock"}

InvokeEvent 对象的 currentDirectory 属性包含一个表示应用程序启动时所在目录的 File 对象。

如果调用某应用程序是由于要打开该应用程序所注册类型的文件，则该文件的本机路径将以字符串的形式包含在命令行参数中。（您的应用程序负责打开该文件或对其执行预定操作。）同样，如果已对应用程序进行编程使其能实现自我更新 （而非依赖于标准 AIR 更新用户界面），则当用户双击某个 AIR 文件（该文件包含具有匹配应用程序 ID 的应用程序）时，将会包含该 AIR 文件的本机路径。

使用 currentDirectory File 对象的 resolve() 方法可访问该文件：

```
 if((invokeEvent.currentDirectory != null)&&(invokeEvent.arguments.length > 0)){
    dir = invokeEvent.currentDirectory;
    fileToOpen = dir.resolvePath(invokeEvent.arguments[0]);
}
```

此外，还应验证参数是否的确是文件的路径。

示例：调用事件日志

Adobe AIR 1.0 和更高版本

下面的示例演示如何为 invoke 事件注册侦听器以及如何处理该事件。该示例会记录所有接收到的调用事件并显示当前目录和命令行参数。

ActionScript 示例

```
package
{
    import flash.display.Sprite;
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;
    import flash.text.TextField;

    public class InvokeEventLogExample extends Sprite
    {
        public var log:TextField;

        public function InvokeEventLogExample()
        {
            log = new TextField();
            log.x = 15;
            log.y = 15;
            log.width = 520;
            log.height = 370;
            log.background = true;

            addChild(log);

            NativeApplication.nativeApplication.addEventListener(InvokeEvent.INVOKE, onInvoke);
        }

        public function onInvoke(invokeEvent:InvokeEvent):void
        {
            var now:String = new Date().toTimeString();
            logEvent("Invoke event received: " + now);

            if (invokeEvent.currentDirectory != null)
            {
                logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
            }
            else
            {
```

```
                logEvent("--no directory information available--");
            }

            if (invokeEvent.arguments.length > 0)
            {
                logEvent("Arguments: " + invokeEvent.arguments.toString());
            }
            else
            {
                logEvent("--no arguments--");
            }
        }

        public function logEvent(entry:String):void
        {
            log.appendText(entry + "\n");
            trace(entry);
        }
    }
}
```

Flex 示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
    invoke="onInvoke(event)" title="Invocation Event Log">
    <mx:Script>
    <![CDATA[
    import flash.events.InvokeEvent;
    import flash.desktop.NativeApplication;

    public function onInvoke(invokeEvent:InvokeEvent):void {
        var now:String = new Date().toTimeString();
        logEvent("Invoke event received: " + now);

        if (invokeEvent.currentDirectory != null){
            logEvent("Current directory=" + invokeEvent.currentDirectory.nativePath);
        } else {
            logEvent("--no directory information available--");
        }

        if (invokeEvent.arguments.length > 0){
            logEvent("Arguments: " + invokeEvent.arguments.toString());
        } else {
            logEvent("--no arguments--");
        }
    }

    public function logEvent(entry:String):void {
        log.text += entry + "\n";
        trace(entry);
    }
    ]]>
    </mx:Script>
    <mx:TextArea id="log" width="100%" height="100%" editable="false"
        valueCommit="log.verticalScrollPosition=log.textHeight;"/>
</mx:WindowedApplication>
```

用户登录时调用 AIR 应用程序

Adobe AIR 1.0 和更高版本

通过将 NativeApplication startAtLogin 属性设置为 true，可以将 AIR 应用程序设置为在当前用户登录时自动启动。设置后，每当该用户登录时该应用程序都将自动启动。除非该设置更改为 false、用户通过操作系统手动更改该设置或者卸载了该应用程序，否则该应用程序将始终在登录时启动。登录时启动是一种运行时设置。该设置仅适用于当前用户。必须安装该应用程序以将 startAtLogin 属性成功设置为 true。如果该属性是在未安装应用程序时设置的，则会引发错误 （例如，通过 ADL 启动该应用程序时）。

注：该应用程序在计算机系统启动时不会启动。它会在用户登录时启动。

若要确定应用程序是自动启动还是用户操作的结果，可以检查 InvokeEvent 对象的 reason 属性。如果该属性的值等于 InvokeEventReason.LOGIN，则应用程序为自动启动。对于其他调用路径， reason 属性设置如下：

- InvokeEventReason.NOTIFICATION （仅限 iOS） — 已通过 APNs 调用应用程序。有关 APNs 的详细信息，请参阅使用推送通知。
- InvokeEventReason.OPEN_URL — 该应用程序已被其他应用程序或系统调用。
- InvokeEventReason.Standard — 所有其他情况。

要访问 reason 属性，您的应用程序必须面向 AIR 1.5.1 或更高版本 （通过在应用程序描述符文件中设置正确命名空间值）。

以下简化的应用程序利用 InvokeEvent reason 属性来确定在发生 invoke 事件时如何响应。如果 reason 属性为 “login”，则应用程序仍在后台运行。否则，它将显示主要应用程序。采用此方式的应用程序通常在登录时启动 （从而可以执行后台处理或事件监控），并打开窗口以响应用户触发的 invoke 事件。

```
package {
    import flash.desktop.InvokeEventReason;
    import flash.desktop.NativeApplication;
    import flash.display.Sprite;
    import flash.events.InvokeEvent;

    public class StartAtLogin extends Sprite
    {
        public function StartAtLogin()
        {
            try
            {
                NativeApplication.nativeApplication.startAtLogin = true;
            }
            catch ( e:Error )
            {
                trace( "Cannot set startAtLogin:" + e.message );
            }

            NativeApplication.nativeApplication.addEventListener( InvokeEvent.INVOKE, onInvoke );
        }

        private function onInvoke( event:InvokeEvent ):void
        {
            if( event.reason == InvokeEventReason.LOGIN )
            {
                //do background processing...
                trace( "Running in background..." );
            }
            else
            {
                this.stage.nativeWindow.activate();
            }
        }
    }
}
```

注：若要查看行为中的差异，请打包并安装该应用程序。只能为已安装的应用程序设置 startAtLogin 属性。

从浏览器调用 AIR 应用程序

Adobe AIR 1.0 和更高版本

使用浏览器调用功能，网站可启动要从浏览器启动的已安装 AIR 应用程序。仅在应用程序描述符文件将 allowBrowserInvocation 设置为 true 时，才允许浏览器调用：

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

当该应用程序通过浏览器调用时，该应用程序的 NativeApplication 对象将会调度 BrowserInvokeEvent 对象。

若要接收 BrowserInvokeEvent 事件，请在 AIR 应用程序中调用 NativeApplication 对象 (NativeApplication.nativeApplication) 的 addEventListener() 方法。当某个事件侦听器针对 BrowserInvokeEvent 事件进行注册时，该事件侦听器还将接收在注册前发生的所有 BrowserInvokeEvent 事件。在对 addEventListener() 的调用返回后将调度这些事件，但并不一定会在注册后可能接收到的其他 BrowserInvokeEvent 事件之前调度。这样就可处理在初始化代码执行之前 （如从浏览器首次调用应用程序时）已发生的 BrowserInvokeEvent 事件。请记住，如果在执行后期 （在应用程序初始化之后）添加一个事件侦听器，它仍然会接收到自应用程序启动以来发生的所有 BrowserInvokeEvent 事件。

BrowserInvokeEvent 对象包含以下属性：

属性	说明
arguments	要传递给应用程序的参数 （字符串）数组。
isHTTPS	浏览器中的内容是否使用 https URL 方案。如果是，则为 (true，否则为 (false)。
isUserEvent	浏览器调用是否生成用户事件 （如鼠标单击）。在 AIR 1.0 中，它始终设置为 true； AIR 需要与浏览器调用功能有关的用户事件。
sandboxType	浏览器中内容的沙箱类型。定义的有效值与可用于 Security.sandboxType 属性的值相同，可以是以下值之一： • Security.APPLICATION — 内容位于应用程序安全沙箱中。 • Security.LOCAL_TRUSTED — 内容位于只能与本地文件系统内容交互的安全沙箱中。 • Security.LOCAL_WITH_FILE — 内容位于只能与本地文件系统内容交互的安全沙箱中。 • Security.LOCAL_WITH_NETWORK — 内容位于只能与远程内容交互的安全沙箱中。 • Security.REMOTE — 内容位于远程 （网络）域中。
securityDomain	浏览器中的内容的安全域，如 "www.adobe.com" 或 "www.example.org"。仅对于远程安全沙箱中的内容 （来自网络域的内容）设置此属性，而不对位于本地或应用程序安全沙箱中的内容设置此属性。

如果使用浏览器调用功能，请务必考虑安全性问题。网站启动 AIR 应用程序时，它可通过 BrowserInvokeEvent 对象的 arguments 属性发送数据。请在任何敏感操作 （例如文件或代码加载 API）中谨慎使用此数据。风险级别取决于应用程序处理数据的方式。如果只希望某个特定网站调用该应用程序，则该应用程序应检查 BrowserInvokeEvent 对象的 securityDomain 属性。也可以要求调用该应用程序的网站使用 HTTPS，这样就可通过检查 BrowserInvokeEvent 对象的 isHTTPS 属性进行验证。

应用程序应验证传入的数据。例如，如果某个应用程序要求传入指向某个特定域的 URL，则该应用程序应验证 URL 确实指向该域。这样就能够阻止攻击者欺骗该应用程序向其发送敏感数据。

任何应用程序都不应使用可能指向本地资源的 BrowserInvokeEvent 参数。例如，应用程序不应基于从浏览器传递的路径创建 File 对象。如果从浏览器传递的将是远程路径，则该应用程序应确保路径不使用 file:// 协议替代远程协议。

应用程序终止

Adobe AIR 1.0 和更高版本

终止应用程序最快的方法是调用 NativeApplication exit() 方法。在应用程序没有要保存的数据或没有要清理的外部资源时，此方法非常适用。调用 exit() 将关闭所有窗口，然后终止该应用程序。但是，若要允许窗口或应用程序的其他组件中断该终止进程 （也许要保存重要数据），请在调用 exit() 之前调度适当的警告事件。

另一种妥善关闭应用程序的方法是提供单一执行路径，而不考虑关闭进程的启动方式。用户 （或操作系统）可以通过以下方式触发应用程序终止：

- 在 NativeApplication.nativeApplication.autoExit 为 true 的情况下关闭最后一个应用程序窗口。
- 从操作系统中选择应用程序退出命令；例如，用户从默认菜单中选择退出应用程序命令。（这种情况仅适用于 Mac OS； Windows 和 Linux 并不通过系统镶边提供应用程序退出命令。）
- 关闭计算机。

当采用上述方式之一通过操作系统执行退出命令时， NativeApplication 将调度 exiting 事件。如果没有侦听器取消 exiting 事件，则任何打开的窗口都将关闭。每个窗口都会先调度一个 closing 事件，然后调度一个 close 事件。如果任何窗口取消 closing 事件，则关闭进程将会停止。

如果需要考虑应用程序的窗口关闭顺序，则可侦听来自 NativeApplication 的 exiting 事件并自行决定以适当的顺序关闭窗口。例如，如果您拥有带有工具调色板的文档窗口，则可能需要运用此方法。如果系统关闭了调色板而用户却决定取消退出命令以保存某些数据，这可能会带来不便，甚至更糟糕的情形。在 Windows 中，收到 exiting 事件的唯一时间是在关闭最后一个窗口之后 （当 NativeApplication 对象的 autoExit 属性设置为 true 时）。

若要在所有平台上提供一致的行为，而不考虑退出顺序是通过操作系统镶边、菜单命令还是通过应用程序逻辑启动的，请遵守用于退出应用程序的以下良好做法：

1 始终先通过 NativeApplication 对象调度 exiting 事件，然后再调用应用程序代码中的 exit() 并检查应用程序的其他组件是否没有取消该事件。

```
public function applicationExit():void {
    var exitingEvent:Event = new Event(Event.EXITING, false, true);
    NativeApplication.nativeApplication.dispatchEvent(exitingEvent);
    if (!exitingEvent.isDefaultPrevented()) {
        NativeApplication.nativeApplication.exit();
    }
}
```

2 侦听来自 NativeApplication.nativeApplication 对象的应用程序 exiting 事件，并在处理函数中关闭任何窗口 （首先调度 closing 事件）。在所有窗口都已关闭后执行任何所需的清理任务，例如保存应用程序数据或删除临时文件。在清理期间仅使用同步方法以确保在应用程序退出之前能完成清理任务。

   如果窗口关闭的顺序无关紧要，则可以遍历 NativeApplication.nativeApplication.openedWindows 数组并依次关闭每个窗口。如果顺序的确 很重要，则需提供一种以正确顺序关闭窗口的方法。

```
private function onExiting(exitingEvent:Event):void {
    var winClosingEvent:Event;
    for each (var win:NativeWindow in NativeApplication.nativeApplication.openedWindows) {
        winClosingEvent = new Event(Event.CLOSING,false,true);
        win.dispatchEvent(winClosingEvent);
        if (!winClosingEvent.isDefaultPrevented()) {
            win.close();
        } else {
            exitingEvent.preventDefault();
        }
    }

    if (!exitingEvent.isDefaultPrevented()) {
        //perform cleanup
    }
}
```

3 Windows 应始终通过侦听自己的 closing 事件来处理自己的清理任务。

4 在应用程序中仅使用一个 exiting 侦听器，因为调用时间较早的处理函数无法知道随后的处理函数是否将取消 exiting 事件（依赖于执行顺序将是不明智的做法）。

第 51 章：处理 AIR 运行时和操作系统信息

Adobe AIR 1.0 和更高版本

本部分讨论 AIR 应用程序如何管理操作系统文件关联，如何检测用户活动以及如何获取 Adobe® AIR® 运行时的有关信息。

更多帮助主题
flash.desktop.NativeApplication

管理文件关联

Adobe AIR 1.0 和更高版本

必须在应用程序描述符中声明应用程序与文件类型之间的关联。在安装过程中， AIR 应用程序安装程序会将 AIR 应用程序关联为声明的各个文件类型的默认打开应用程序，但当文件类型已有与之关联的默认打开应用程序时除外。 AIR 应用程序安装过程不会覆盖现有的文件类型关联。要接管与其他应用程序的关联，请在运行时调用 NativeApplication.setAsDefaultApplication() 方法。

在启动应用程序时验证所需的文件关联是否就绪是一种很好的做法。这是因为 AIR 应用程序安装程序不会覆盖现有的文件关联，并且用户系统上的文件关联随时都可能发生更改。如果文件类型当前已关联到其他应用程序，则在接管现有关联之前询问用户是否更改文件关联也是一种很有礼貌的做法。

应用程序可以通过 NativeApplication 类的以下方法管理文件关联。各个方法都将文件类型扩展名视为一个参数：

方法	说明
isSetAsDefaultApplication()	如果 AIR 应用程序当前与指定的文件类型关联，则返回 true。
setAsDefaultApplication()	在 AIR 应用程序与文件类型的打开操作之间建立关联。
removeAsDefaultApplication()	删除 AIR 应用程序与文件类型之间的关联。
getDefaultApplication()	报告当前与文件类型关联的应用程序的路径。

AIR 只能管理最初在应用程序描述符中声明的文件类型的关联。即使用户在文件类型与您的应用程序之间手动建立了关联，您也无法获取有关未声明文件类型的关联信息。使用未在应用程序描述符中声明的文件类型的扩展名调用任何文件关联管理方法，将导致应用程序引发运行时异常。

获取运行时版本和修补级别

Adobe AIR 1.0 和更高版本

NativeApplication 对象包含一个 runtimeVersion 属性，该属性表示在其中运行应用程序的运行时的版本 （一个字符串，例如 "1.0.5"）。 NativeApplication 对象还包含一个 runtimePatchLevel 属性，该属性表示运行时的修补级别 （一个数字，例如 2960）。以下代码使用了这些属性：

```
trace(NativeApplication.nativeApplication.runtimeVersion);
trace(NativeApplication.nativeApplication.runtimePatchLevel);
```

检测 AIR 功能

Adobe AIR 1.0 和更高版本

对于与 Adobe AIR 应用程序捆绑的文件，Security.sandboxType 属性设置为由 Security.APPLICATION 常量定义的值。可以根据文件是否位于 Adobe AIR 安全沙箱中来加载内容 （可以包含也可以不包含特定于 AIR 的 API），如以下代码所示：

```
if (Security.sandboxType == Security.APPLICATION)
{
    // Load SWF that contains AIR APIs
}
else
{
    // Load SWF that does not contain AIR APIs
}
```

对于未随 AIR 应用程序一起安装的所有资源，其所分配的安全沙箱与 Adobe® Flash® Player 在 Web 浏览器中为其分配的安全沙箱是相同的。远程资源根据其源域放在相应的沙箱中，本地资源放在只能与远程内容交互的沙箱、只能与本地文件系统内容交互的沙箱或受信任的本地沙箱中。

可以检查 Capabilities.playerType 静态属性是否设置为 "Desktop"，从而查看内容是否在运行时中执行 （不是在浏览器中运行的 Flash Player 中运行）。

有关详细信息，请参阅 第 922 页的 “AIR 安全性 ”。

跟踪用户当前状态

Adobe AIR 1.0 和更高版本

NativeApplication 对象调度两个事件，可帮助检测用户是否正在使用计算机。如果在 NativeApplication.idleThreshold 属性指定的间隔内未检测到任何鼠标或键盘活动，则 NativeApplication 将调度 userIdle 事件。当发生下一次键盘或鼠标输入时，NativeApplication 对象将调度 userPresent 事件。 idleThreshold 间隔以秒为单位，默认值为 300 （5 分钟）。还可以通过 NativeApplication.nativeApplication.lastUserInput 属性获取自上一个用户输入以来经过的秒数。

以下代码行将空闲阈值设置为 2 分钟，并同时侦听 userIdle 和 userPresent 事件：

```
NativeApplication.nativeApplication.idleThreshold = 120;
NativeApplication.nativeApplication.addEventListener(Event.USER_IDLE, function(event:Event) {
    trace("Idle");
});
NativeApplication.nativeApplication.addEventListener(Event.USER_PRESENT, function(event:Event) {
    trace("Present");
});
```

注：在任意两个 userPresent 事件之间，只调度一个 userIdle 事件。

第 52 章：使用 AIR 本机窗口

Adobe AIR 1.0 和更高版本

使用 Adobe® AIR® 本机窗口 API 提供的类可创建和管理桌面窗口。

AIR 中的本机窗口的基础知识

Adobe AIR 1.0 和更高版本

有关在 AIR 中使用本机窗口的快速介绍和代码示例，请参阅 Adobe Developer Connection 中的以下快速入门文章：

- 创建透明窗口应用程序 (Flex)
- 与窗口进行交互 (Flex)
- 自定义本机窗口的外观 (Flex)
- 启动窗口 (Flex)
- 创建弹出式窗口 (Flex)
- 控制窗口的显示顺序 (Flex)
- 创建可调整大小的非矩形窗口 (Flex)
- 与窗口进行交互 (Flash)
- 自定义本机窗口的外观 (Flash)
- 创建弹出式窗口 (Flash)
- 控制窗口的显示顺序 (Flash)
- 创建可调整大小的非矩形窗口 (Flash)

AIR 提供易于使用的跨平台窗口 API，以便使用 Flash®、 Flex™ 和 HTML 编程技术创建本机操作系统窗口。

使用 AIR 可使您在开发应用程序的外观时具有广泛的自由度。您创建的窗口可以类似于标准的桌面应用程序，也就是在 Mac 上运行时与 Apple 风格相媲美、在 Windows 上运行时符合 Microsoft 惯例以及在 Linux 上与窗口管理器协调一致，所有这些的实现都不需要撰写平台专用的代码。此外，无论应用程序运行于何处，都可以使用 Flex 框架提供的可设置外观、可扩展的镶边树立您自己的风格。由于完全支持针对桌面进行透明度和 Alpha 混合，因此甚至可以用矢量图和位图绘制您自己的窗口镶边。是否厌倦了矩形窗口？现在可以绘制圆形窗口。

AIR 中的窗口

Adobe AIR 1.0 和更高版本

AIR 支持三个不同的 API 来处理窗口：

- 面向 ActionScript 的 NativeWindow 类提供最底层的窗口 API。在使用 ActionScript 和 Flash Professional 创作的应用程序中使用 NativeWindow。考虑扩展 NativeWindow 类，以使应用程序中使用的窗口专用化。
- 在 HTML 环境中，可以使用 JavaScript Window 类，就像在基于浏览器的 Web 应用程序中的那样。对 JavaScript Window 方法的调用将转移到基础本机窗口对象。

- Flex 框架 mx:WindowedApplication 和 mx:Window 类为 NativeWindow 类提供 Flex“ 包装 ”。用 Flex 创建 AIR 应用程序时，WindowedApplication 组件将代替 Application 组件，并且必须始终使用前者作为您的 Flex 应用程序的初始窗口。

ActionScript 窗口

使用 NativeWindow 类创建窗口时，会直接使用 Flash Player 舞台并显示列表。若要向 NativeWindow 添加视觉对象，请将该对象添加到窗口舞台的显示列表或添加到舞台上的另一个显示对象容器。

HTML 窗口

在创建 HTML 窗口时，可使用 HTML、 CSS 和 JavaScript 来显示内容。若要向 HTML 窗口添加可视对象，请将该内容添加到 HTML DOM。HTML 窗口是一种特殊类别的 NativeWindow。AIR 主机定义 HTML 窗口中的 nativeWindow 属性，该属性提供对基础 NativeWindow 实例的访问。使用此属性可以访问此处所述的 NativeWindow 属性、方法和事件。

注：JavaScript Window 对象还具有用于脚本访问包含窗口的方法，例如 moveTo() 和 close()。如果多个方法均可用，您可以使用其中简便易用的方法。

Flex Framework 窗口

使用 Flex 框架创建窗口时，通常使用 MXML 组件来填充该窗口。若要向窗口添加 Flex 组件，请将该组件元素添加到窗口 MXML 定义。还可以使用 ActionScript 来动态添加内容。mx:WindowedApplication 和 mx:Window 组件被设计为 Flex 容器，因此可以直接接受 Flex 组件，而 NativeWindow 对象无法直接接受 Flex 组件。在必要时，可以使用 nativeWindow 属性通过 WindowedApplication 和 Window 对象来访问 NativeWindow 属性和方法。

初始应用程序窗口

应用程序的第一个窗口是由 AIR 自动为您创建的。AIR 使用应用程序描述符文件的 initialWindow 元素中指定的参数设置该窗口的属性和内容。

如果根内容是 SWF 文件，则 AIR 将创建 NativeWindow 实例，加载 SWF 文件并将其添加到窗口舞台。如果根内容是 HTML 文件，则 AIR 将创建 HTML 窗口并加载 HTML。

本机窗口类

Adobe AIR 1.0 和更高版本

本机窗口 API 包含以下类：

包	类
flash.display	• NativeWindow • NativeWindowInitOptions • NativeWindowDisplayState • NativeWindowResize • NativeWindowSystemChrome • NativeWindowType
flash.events	• NativeWindowBoundsEvent • NativeWindowDisplayStateEvent

本机窗口事件流

Adobe AIR 1.0 和更高版本

本机窗口调度事件，以便通知感兴趣的组件将要发生或已发生重要更改。对于许多与窗口相关的事件的调度是成对进行的。第一个事件警告即将发生更改。第二个事件通知已完成更改。可以取消警告事件，但不能取消通知事件。以下序列说明在用户单击窗口的最大化按钮后发生的事件流：

1. NativeWindow 对象调度 displayStateChanging 事件。
2. 如果已注册的侦听器均未取消该事件，则窗口将最大化。
3. NativeWindow 对象调度 displayStateChange 事件。

   此外， NativeWindow 对象还调度对窗口大小和位置进行相关更改的事件。窗口不调度这些相关更改的警告事件。相关事件包括：

   a. move 事件，如果窗口的左上角由于最大化操作而发生移动，则调度该事件。
   b. resize 事件，如果窗口大小由于最大化操作而发生更改，则调度该事件。

   在最小化、还原、关闭、移动窗口和调整窗口大小时， NativeWindow 对象调度相似序列的事件。

   在通过窗口镶边或其他操作系统控制的机制启动更改时，仅调度警告事件。在调用窗口方法以更改窗口大小、位置或显示状态时，窗口仅调度通知更改的事件。如果需要，可以使用窗口 dispatchEvent() 方法调度警告事件，然后检查在继续进行更改之前是否取消了警告事件。

   有关窗口 API 类、方法、属性和事件的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

控制本机窗口样式和行为的属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下属性控制窗口的基本外观和行为：

- type
- systemChrome
- transparent
- owner

创建窗口时，在传递到 window 构造函数的 NativeWindowInitOptions 对象上设置这些属性。 AIR 从应用程序描述符中读取初始应用程序窗口的属性（不包括 type 属性，该属性无法在应用程序描述符中设置且始终设置为 normal）。窗口创建后将无法更改这些属性。

这些属性的一些设置互不兼容：在 transparent 为 true 或 type 为 lightweight 时， systemChrome 无法设置为 standard。

窗口类型

Adobe AIR 1.0 和更高版本

AIR 窗口类型组合本机操作系统的镶边属性和可见性属性来创建三种功能类型的窗口。使用 NativeWindowType 类中定义的常量可引用代码中的类型名称。 AIR 提供以下窗口类型：

类型	说明
Normal	典型窗口。普通窗口使用标准尺寸样式的镶边，并显示在 Windows 的任务栏中和 Mac OS X 的窗口菜单中。
Utility	工具面板。实用程序窗口使用较细的系统镶边，而且不显示在 Windows 的任务栏中和 Mac OS X 的窗口菜单中。
Lightweight	简单窗口没有镶边，而且不显示在 Windows 的任务栏中和 Mac OS X 的窗口菜单中。此外， Windows 中的简单窗口没有 " 系统 "(Alt+Space) 菜单。简单窗口适用于通知气泡和控件，例如用于打开短期显示区域的组合框。在使用的 type 为简单时， systemChrome 必须设置为 none。

窗口镶边

Adobe AIR 1.0 和更高版本

窗口镶边是一组使用户可以在桌面环境中操作窗口的控件。镶边元素包括标题栏、标题栏按钮、边框和调整大小手柄。

系统镶边

可以将 systemChrome 属性设置为 standard 或 none。选择 standard 系统镶边可为窗口提供一组由用户的操作系统创建和设置样式的标准控件。选择 none 可为窗口提供您自己的镶边。使用 NativeWindowSystemChrome 类中定义的常量可引用代码中的系统镶边设置。

系统镶边由系统管理。应用程序无法直接访问控件本身，但在使用控件时可以响应调度的事件。对窗口使用标准镶边时，transparent 属性必须设置为 false， type 属性必须设置为 normal 或 utility。

Flex 镶边

在使用 Flex WindowedApplication 或 Window 组件时，窗口可以使用系统镶边或 Flex 框架提供的镶边。若要使用 Flex 镶边，请将用于创建该窗口的 systemChrome 属性设置为 none。当使用 Flex 4 spark 组件而不是 mx 组件时，必须指定外观类才能使用 Flex 镶边。您可以使用内置外观或提供您自己的外观。以下示例演示了如何使用内置 spark WindowedApplication 外观类来提供窗口镶边：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
xmlns:mx="library://ns.adobe.com/flex/mx">
<fx:Style>
@namespace "library://ns.adobe.com/flex/spark";
WindowedApplication
{
    skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
}
</fx:Style>
</s:WindowedApplication>
```

有关详细信息，请参阅使用 Flex 4：关于 AIR 窗口容器：控制窗口镶边

自定义镶边

创建不带系统镶边的窗口时，您必须添加自己的镶边控件才能处理用户和该窗口之间的交互。还可以根据需要随意创建透明的非矩形窗口。

若要与 mx:WindowedApplication 或 mx:Window 组件一起使用自定义镶边，必须将 showFlexChrome 样式设置为 false。否则， Flex 会将自己的镶边添加到您的窗口。

窗口透明度

Adobe AIR 1.0 和更高版本

若要允许窗口与桌面或其他窗口进行 Alpha 混合，请将该窗口的 transparent 属性设置为 true。必须在创建窗口之前设置 transparent 属性，否则将无法更改该属性。

透明窗口没有默认背景。不包含应用程序所绘制对象的任何窗口区域都不可见。如果所显示对象的 Alpha 设置小于 1，则该对象下方的任何内容都会显示出来，包括同一窗口中的其他显示对象、其他窗口和桌面。

在希望创建具有不规则形状边框、“淡出”边框或显示为不可见的边框的应用程序时，透明窗口非常有用。然而，呈现经过 Alpha 混合的较大区域可能会很慢，因此应谨慎使用该效果。

重要说明：在 Linux 中，不能穿过完全透明的像素传递鼠标事件。应避免用完全透明的大型区域创建窗口，因为可能会在无法察觉的情况下阻止用户访问其他窗口或其桌面上的项目。在 Mac OS X 和 Windows 中，可以穿过完全透明的像素传递鼠标事件。

不能对具有系统镶边的窗口使用透明度。此外，透明窗口中不能显示 HTML 中的 SWF 和 PDF 内容。有关详细信息，请参阅第 860 页的“在 HTML 页中加载 SWF 或 PDF 内容时的注意事项”。

静态 NativeWindow.supportsTransparency 属性可报告窗口透明度是否可用。在不支持透明度时，应用程序将与黑色背景合成。在这些情况下，应用程序的任何透明区域都显示为不透明的黑色。这种做法可以很好地应对万一此属性测试失败而需要回退的情况。例如，您可以向用户显示警告对话框，或显示矩形非透明用户界面。

请注意， Mac 和 Windows 操作系统始终支持透明度。支持 Linux 操作系统需要使用合成窗口管理器，但即使有合成窗口管理器处于活动状态，透明度也可能因用户显示选项或硬件配置而不可用。

MXML 应用程序窗口中的透明度

Adobe AIR 1.0 和更高版本

默认情况下，即使将窗口创建为 transparent， MXML 窗口的背景也是不透明的。（请注意窗口各个角的透明度效果。）若要显示窗口的透明背景，请设置样式表中的背景颜色和 Alpha 值或应用程序 MXML 文件中包含的 <mx:Style> 元素。例如，以下样式声明使背景具有略微透明的绿色阴影：

```
WindowedApplication
{
    background-alpha:".8";
    background-color:"0x448234";
}
```

HTML 应用程序窗口中的透明度

Adobe AIR 1.0 和更高版本

默认情况下，即使包含窗口是透明的， HTML 窗口和 HTMLLoader 对象中所显示的 HTML 内容的背景也是不透明的。若要关闭为 HTML 内容显示的默认背景，请将 paintsDefaultBackground 属性设置为 false。以下示例创建 HTMLLoader 并关闭默认背景：

```
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.paintsDefaultBackground = false;
```

此示例使用 JavaScript 来关闭 HTML 窗口的默认背景：

```
window.htmlLoader.paintsDefaultBackground = false;
```

如果 HTML 文档中的元素设置背景颜色，则该元素的背景不透明。不支持设置局部透明度 （或不透明度）值。但是，可以使用透明 PNG 格式的图形作为页面或页面元素的背景以实现相似的视觉效果。

窗口所有权

一个窗口可以拥有 一个或多个其他窗口。这些拥有的窗口始终显示在主窗口的前面，随主窗口一起最小化和还原，并在关闭主窗口时关闭。窗口所有权无法转让给其他窗口，也无法删除窗口所有权。一个窗口只能归一个主窗口所有，但该窗口可拥有任意数量的其他窗口。

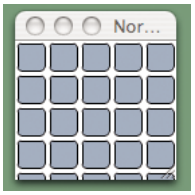
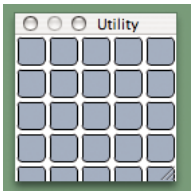
使用窗口所有权，可以更加轻松地管理工具调色板和对话框所使用的窗口。例如，如果在文档窗口中显示与之关联的“保存”对话框，使文档窗口拥有该对话框可使该对话框自动显示在文档窗口前面。

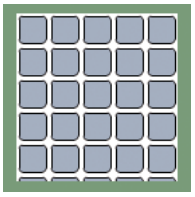
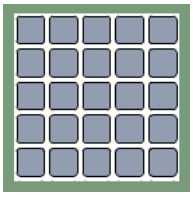
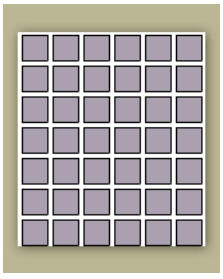
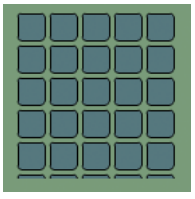
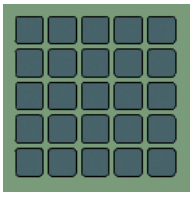
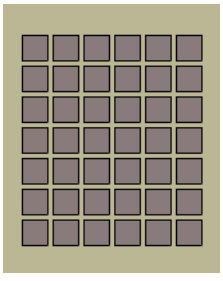
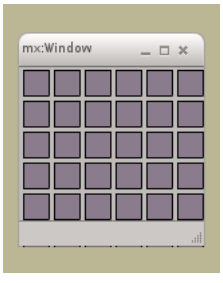
- NativeWindow.owner
- Christian Cantrell：AIR 2.6 中的专属窗口

可视窗口目录

Adobe AIR 1.0 和更高版本

下表说明了窗口属性设置的不同组合在 Mac OS X、 Windows 和 Linux 操作系统中的视觉效果：

窗口设置	Mac OS X	Microsoft Windows	Linux*
Type：normal SystemChrome：standard Transparent：false			
Type：utility SystemChrome：standard Transparent：false			

窗口设置	Mac OS X	Microsoft Windows	Linux*
Type：Any SystemChrome：none Transparent：false			
Type：Any SystemChrome：none Transparent：true			
mxWindowedApplication 或 mx:Window Type：Any SystemChrome：none Transparent：true			

*Ubuntu （带有 Compiz 窗口管理器）

注：AIR 不支持以下系统镶边元素：Mac OS X 工具栏、 Mac OS X 代理图标、 Windows 标题栏图标以及替代系统镶边。

创建窗口

Adobe AIR 1.0 和更高版本

AIR 自动创建应用程序的第一个窗口，但您可以创建所需的任何其他窗口。若要创建本机窗口，请使用 NativeWindow 构造函数方法。

若要创建 HTML 窗口，请使用 HTMLLoader createRootWindow() 方法或者从 HTML 文档中调用 JavaScript window.open() 方法。创建的窗口是一个 NativeWindow 对象，其显示列表中包含 HTMLLoader 对象。 HTMLLoader 对象解释并显示窗口的 HTML 内容和 JavaScript 内容。您可以使用 window.nativeWindow 属性从 JavaScript 访问基础 NativeWindow 对象的属性。（只有在 AIR 应用程序沙箱中运行的代码可以访问此属性。）

初始化窗口时 （包括初始应用程序窗口），您应考虑在不可见状态下创建窗口、加载内容或执行任何图形更新，然后使该窗口可见。此过程可防止用户看到任何不和谐的可视更改。您可以指定应用程序的初始窗口应在不可见状态下创建，方法是在应用程序描述符中指定 <visible>false</visible> 标签 （或通过完全忽略此标签，因为 false 是默认值）。默认情况下，新 NativeWindow 不可见。使用 HTMLLoader createRootWindow() 方法创建 HTML 窗口时，可以将 visible 参数设置为 false。调用 NativeWindow activate() 方法或将 visible 属性设置为 true 以使窗口可见。

指定窗口初始化属性

Adobe AIR 1.0 和更高版本

创建桌面窗口后，无法更改本机窗口的初始化属性。这些不可改变的属性及其默认值包括：

属性	默认值
systemChrome	standard
type	normal
transparent	false
owner	null
maximizable	true
minimizable	true
resizable	true

在应用程序描述符文件中设置 AIR 所创建的初始窗口的属性。 AIR 应用程序主窗口的 type 值始终是 normal。（可以在描述符文件中指定其他窗口属性，例如 visible、 width 和 height，但可以随时更改这些属性。）

使用 NativeWindowInitOptions 类可设置应用程序创建的其他本机窗口和 HTML 窗口的属性。在创建窗口时，必须将指定窗口属性的 NativeWindowInitOptions 对象传递到 NativeWindow 构造函数或 HTMLLoader createRootWindow() 方法。

以下代码为实用程序窗口创建 NativeWindowInitOptions 对象：

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.type = NativeWindowType.UTILITY
options.transparent = false;
options.resizable = false;
options.maximizable = false;
```

当 transparent 为 true 或 type 为 lightweight 时，不支持 将 systemChrome 设置为 standard。

注：无法为使用 JavaScript window.open() 函数创建的窗口设置初始化属性。但是，您可以通过实现自己的 HTMLHost 类来覆盖这些窗口的创建方式。有关详细信息，请参阅第 869 页的 “ 处理对 window.open() 的 JavaScript 调用 ”。

使用 Flex mx:Window 类创建窗口时，在 window 对象自身的窗口 MXML 声明中或创建窗口的代码中可指定该窗口的初始化属性。在调用 open() 方法之前，不会创建基础 NativeWindow 对象。打开窗口后，无法更改这些初始化属性。

创建初始应用程序窗口

Adobe AIR 1.0 和更高版本

AIR 根据应用程序描述符中指定的属性来创建初始应用程序窗口并加载内容元素中引用的文件。内容元素必须引用 SWF 文件或 HTML 文件。

初始窗口可以是应用程序的主窗口，也可以仅用于启动一个或多个其他窗口。该窗口不必一定可见。

使用 ActionScript 创建初始窗口

Adobe AIR 1.0 和更高版本

使用 ActionScript 创建 AIR 应用程序时，应用程序的主类必须扩展 Sprite 类（或 Sprite 的子类）。此类用作应用程序的主入口点。

在应用程序启动时， AIR 创建窗口，创建主类的实例并将该实例添加到窗口舞台。为访问窗口，可以侦听 addedToStage 事件，然后使用 Stage 对象的 nativeWindow 属性来获取 NativeWindow 对象的引用。

以下示例说明用 ActionScript 所构建 AIR 应用程序的主类的基本框架：

```
package {
    import flash.display.NativeWindow;
    import flash.display.Sprite;
    import flash.events.Event;

    public class MainClass extends Sprite
    {
        private var mainWindow:NativeWindow;
        public function MainClass(){
            this.addEventListener(Event.ADDED_TO_STAGE, initialize);
        }

        private function initialize(event:Event):void{
            mainWindow = this.stage.nativeWindow;
            //perform initialization...
            mainWindow.activate(); //show the window
        }
    }
}
```

注：从技术角度讲，您可以访问主类的构造函数中的 nativeWindow 属性。然而，这是一种仅适用于初始应用程序窗口的特殊情况。

在 Flash Professional 中创建应用程序时，如果您不在单独的 ActionScript 文件中创建自己的主文档类，会自动创建主文档类。可使用舞台 nativeWindow 属性为初始窗口访问 NativeWindow 对象。例如，以下代码激活最大化状态的主窗口 （从时间轴）：

```
import flash.display.NativeWindow;

var mainWindow:NativeWindow = this.stage.nativeWindow;
mainWindow.maximize();
mainWindow.activate();
```

使用 Flex 创建初始窗口

Adobe AIR 1.0 和更高版本

在用 Flex 框架创建 AIR 应用程序时，使用 mx:WindowedApplication 作为主 MXML 文件的根元素。（可以使用 mx:Application 组件，但它仅支持 AIR 中部分可用的功能。） WindowedApplication 组件可用作应用程序的初始入口点。

在启动应用程序时， AIR 将创建本机窗口、初始化 Flex 框架并将 WindowedApplication 对象添加到窗口舞台。启动序列完成后， WindowedApplication 调度 applicationComplete 事件。使用 WindowedApplication 实例的 nativeWindow 属性访问桌面 window 对象。

以下示例创建设置了其 x 坐标和 y 坐标的简单 WindowedApplication 组件：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    applicationComplete="placeWindow()">
    <mx:Script>
        <![CDATA[
            private function placeWindow():void{
                this.nativeWindow.x = 300;
                this.nativeWindow.y = 300;
            }
        ]]>
    </mx:Script>
    <mx:Label text="Hello World" horizontalCenter="0" verticalCenter="0"/>
</mx:WindowedApplication>
```

创建 NativeWindow

Adobe AIR 1.0 和更高版本

若要创建 NativeWindow，请将 NativeWindowInitOptions 对象传递到 NativeWindow 构造函数：

```
var options:NativeWindowInitOptions = new NativeWindowInitOptions();
options.systemChrome = NativeWindowSystemChrome.STANDARD;
options.transparent = false;
var newWindow:NativeWindow = new NativeWindow(options);
```

在将 visible 属性设置为 true 或调用 activate() 方法后，才显示该窗口。

创建窗口后，可以使用舞台属性和 Flash 显示列表技术来初始化其属性和将内容加载到该窗口中。

几乎在所有情况下，都应将新本机窗口的舞台 scaleMode 属性设置为 noScale（使用 StageScaleMode.NO_SCALE 常量）。Flash 缩放模式旨在用于应用程序作者事先不知道应用程序显示区高宽比的情况。使用这些缩放模式，作者可以选择最少的内容损失：剪辑内容、拉伸或压缩内容或者用空白空间进行填充。由于您控制 AIR （窗口帧）中的显示区，因此可以在不损失内容的情况下使窗口大小适合内容或者使内容大小适合窗口。

Flex 和 HTML 窗口的缩放模式自动设置为 noScale。

注：若要确定当前操作系统中允许的最大窗口大小和最小窗口大小，请使用以下静态 NativeWindow 属性：

```
var maxOSSize:Point = NativeWindow.systemMaxSize;
var minOSSize:Point = NativeWindow.systemMinSize;
```

创建 HTML 窗口

Adobe AIR 1.0 和更高版本

若要创建 HTML 窗口，可以调用 JavaScript Window.open() 方法，也可以调用 AIR HTMLLoader 类的 createRootWindow() 方法。

任何安全沙箱中的 HTML 内容都可以使用标准 JavaScript Window.open() 方法。如果内容在应用程序沙箱外部运行，则只能调用 open() 方法来响应用户交互，例如鼠标单击或按键。在调用 open() 时，将创建具有系统镶边的窗口以在指定 URL 处显示内容。例如：

```
newWindow = window.open("xmpl.html", "logWindow", "height=600, width=400, top=10, left=10");
```

注：可以在 ActionScript 中扩展 HTMLHost 类以自定义用 JavaScript window.open() 函数创建的窗口。请参阅第 864 页的 “ 关于扩展 HTMLHost 类 ”。

应用程序安全沙箱中的内容可以访问更强大的窗口创建方法 HTMLLoader.createRootWindow()。使用此方法，可以指定新窗口的所有创建选项。例如，以下 JavaScript 代码创建不具有大小为 300x400 像素的系统镶边的简单类型窗口：

```
var options = new air.NativeWindowInitOptions();
options.systemChrome = "none";
options.type = "lightweight";

var windowBounds = new air.Rectangle(200,250,300,400);
newHTMLLoader = air.HTMLLoader.createRootWindow(true, options, true, windowBounds);
newHTMLLoader.load(new air.URLRequest("xmpl.html"));
```

注：如果新窗口加载的内容位于应用程序安全沙箱外部，则 window 对象没有以下 AIR 属性：runtime、 nativeWindow 或 htmlLoader。

如果创建透明窗口，则不一定总能显示加载到该窗口的 HTML 中嵌入的 SWF 内容。对于用于引用 SWF 文件的 object 或 embed 标签，必须将它们的 wmode 参数设置为 opaque 或 transparent。 wmode 的默认值为 window，因此默认情况下，透明窗口中不显示 SWF 内容。无论设置哪种 wmode 值，透明窗口中都无法显示 PDF 内容。（在早于 AIR 1.5.2 的版本中，透明窗口中也无法显示 SWF 内容。）

使用 createRootWindow() 方法创建的窗口与打开窗口相互独立。JavaScript Window 对象的 parent 和 opener 属性为 null。打开窗口可以使用 createRootWindow() 函数返回的 HTMLLoader 引用来访问新窗口的 Window 对象。在前一个示例的上下文中，语句 newHTMLLoader.window 引用所创建窗口的 JavaScript Window 对象。

注：可以从 JavaScript 和 ActionScript 中调用 createRootWindow() 函数。

创建 mx:Window

Adobe AIR 1.0 和更高版本

若要创建 mx:Window，可以将 mx:Window 用作根标签创建 MXML 文件，也可以直接调用 Window 类构造函数。

下例通过调用 Window 构造函数来创建和显示 mx:Window：

```
var newWindow:Window = new Window();
newWindow.systemChrome = NativeWindowSystemChrome.NONE;
newWindow.transparent = true;
newWindow.title = "New Window";
newWindow.width = 200;
newWindow.height = 200;
newWindow.open(true);
```

向窗口添加内容

Adobe AIR 1.0 和更高版本

向 AIR 窗口添加内容的方式取决于窗口类型。例如，使用 MXML 和 HTML，您能够以声明方式定义窗口的基本内容。可以在应用程序 SWF 文件中嵌入资源，也可以从单独的应用程序文件中加载这些资源。Flex、Flash 和 HTML 内容都可以动态创建和动态添加到窗口。

在加载包含 JavaScript 的 SWF 内容或 HTML 内容时，必须考虑 AIR 安全模型。应用程序安全沙箱中的任何内容（即与应用程序一起安装且可用 app: URL 方案加载的内容）具有对所有 AIR API 的完全访问权限。从此沙箱外部加载的任何内容均无法访问 AIR API。应用程序沙箱外部的 JavaScript 内容无法使用 JavaScript Window 对象的 runtime、 nativeWindow 或 htmlLoader 属性。

为允许安全的跨脚本访问，可以使用沙箱桥在应用程序内容和非应用程序内容之间提供有限的接口。在 HTML 内容中，还可以将应用程序的页面映射到非应用程序沙箱中以允许该页面上的代码跨脚本访问外部内容。请参阅 第 922 页的 “AIR 安全性”。

加载 SWF 文件或图像

可以使用 flash.display.Loader 类将 Flash SWF 文件或图像加载到本机窗口的显示列表中：

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.URLRequest;
    import flash.display.Loader;

    public class LoadedSWF extends Sprite
    {
        public function LoadedSWF(){
            var loader:Loader = new Loader();
            loader.load(new URLRequest("visual.swf"));
            loader.contentLoaderInfo.addEventListener(Event.COMPLETE,loadFlash);
        }

        private function loadFlash(event:Event):void{
            addChild(event.target.loader);
        }
    }
}
```

注：如果使用 ActionScript 1 或 2 创建的早期 SWF 文件加载到同一窗口中，则它们共享全局状态，例如定义、单一实例和全局变量。如果此类 SWF 文件依赖于未做改动的全局状态才能正确工作，则无法将其多次加载到同一窗口，也无法使用重叠的类定义和变量将其作为另一个 SWF 文件加载到同一窗口。此内容可以加载到单独的窗口中。

将 **HTML** 内容加载到 **NativeWindow** 中

若要将 HTML 内容加载到 NativeWindow 中，可以将 HTMLLoader 对象添加到窗口舞台并将 HTML 内容加载到 HTMLLoader 中，也可以使用 HTMLLoader.createRootWindow() 方法创建已包含 HTMLLoader 对象的窗口。以下示例在本机窗口舞台上的 300 x 500 像素的显示区域内显示 HTML 内容：

```
//newWindow is a NativeWindow instance
var htmlView:HTMLLoader = new HTMLLoader();
htmlView.width = 300;
htmlView.height = 500;

//set the stage so display objects are added to the top-left and not scaled
newWindow.stage.align = "TL";
newWindow.stage.scaleMode = "noScale";
newWindow.stage.addChild( htmlView );

//urlString is the URL of the HTML page to load
htmlView.load( new URLRequest(urlString) );
```

若要将 HTML 页加载到 Flex 应用程序中，可以使用 Flex HTML 组件。

如果窗口使用透明度 （即窗口的 transparent 属性为 true），则不会显示 HTML 文件中的 SWF 内容，除非将用于引用 SWF 文件的 object 或 embed 标签的 wmode 参数设置为 opaque 或 transparent。wmode 的默认值为 window，因此默认情况下，透明窗口中不显示 SWF 内容。无论使用哪种 wmode 值，透明窗口中都不显示 PDF 内容。

另外，如果缩放、旋转 HTMLLoader 控件或将 HTMLLoader alpha 属性设置为 1.0 之外的任何值，则不会显示 SWF 内容和 PDF 内容。

将 **SWF** 内容添加为 **HTML** 窗口上的覆盖图

由于 HTML 窗口包含在 NativeWindow 实例中，因此可以将 Flash 显示对象添加到显示列表中 HTML 层的上方或下方。

若要将显示对象添加到 HTML 层的上方，请使用 window.nativeWindow.stage 属性的 addChild() 方法。addChild() 方法将分层的内容添加到窗口中任何现有内容的上方。

若要将显示对象添加到 HTML 层的下方，请使用 window.nativeWindow.stage 属性的 addChildAt() 方法，为 index 参数传入值零。将对象放在零索引处会将现有内容 （包括 HTML 显示）上移一层并在底部插入新内容。为使在 HTML 页下方分层的内容可见，必须将 HTMLlLoader 对象的 paintsDefaultBackground 属性设置为 false。此外，该页中设置背景颜色的任何元素都将不是透明的。例如，如果设置页面 body 元素的背景颜色，则该页的所有内容都将不是透明的。

以下示例说明如何将 Flash 显示对象作为覆盖图或衬垫层添加到 HTML 页。该示例创建两个简单的 shape 对象，在 HTML 内容下方和上方各添加一个 shape 对象。该示例还基于 enterFrame 事件更新形状位置。

```
 <html>
<head>
<title>Bouncers</title>
<script src="AIRAliases.js" type="text/javascript"></script>
<script language="JavaScript" type="text/javascript">
air.Shape = window.runtime.flash.display.Shape;

function Bouncer(radius, color){
    this.radius = radius;
    this.color = color;

    //velocity
    this.vX = -1.3;
    this.vY = -1;

    //Create a Shape object and draw a circle with its graphics property
    this.shape = new air.Shape();
    this.shape.graphics.lineStyle(1,0);
    this.shape.graphics.beginFill(this.color,.9);
    this.shape.graphics.drawCircle(0,0,this.radius);
    this.shape.graphics.endFill();

    //Set the starting position
    this.shape.x = 100;
    this.shape.y = 100;


    //Moves the sprite by adding (vX,vY) to the current position
    this.update = function(){
        this.shape.x += this.vX;
        this.shape.y += this.vY;

        //Keep the sprite within the window
        if( this.shape.x - this.radius < 0){
            this.vX = -this.vX;
        }
        if( this.shape.y - this.radius < 0){
            this.vY = -this.vY;
        }
        if( this.shape.x  + this.radius > window.nativeWindow.stage.stageWidth){
            this.vX = -this.vX;
        }
        if( this.shape.y  + this.radius > window.nativeWindow.stage.stageHeight){
            this.vY = -this.vY;
        }

    };
}

function init(){
    //turn off the default HTML background
    window.htmlLoader.paintsDefaultBackground = false;
    var bottom = new Bouncer(60,0xff2233);
    var top = new Bouncer(30,0x2441ff);
```

```
    //listen for the enterFrame event
    window.htmlLoader.addEventListener("enterFrame",function(evt){
        bottom.update();
        top.update();
    });

    //add the bouncing shapes to the window stage
    window.nativeWindow.stage.addChildAt(bottom.shape,0);
    window.nativeWindow.stage.addChild(top.shape);
}
</script>
<body onload="init();">
<h1>de Finibus Bonorum et Malorum</h1>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.</p>
<p style="background-color:#FFFF00; color:#660000;">This paragraph has a background color.</p>
<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis
praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias
excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui
officia deserunt mollitia animi, id est laborum et dolorum fuga.</p>
</body>
</html>
```

此示例简要介绍了 AIR 中跨越 JavaScript 和 ActionScript 之间边界的一些高级技术。如果您不熟悉如何使用 ActionScript 显示对象，请参阅 《Adobe ActionScript 3.0 开发人员指南》中的第 126 页的 “ 显示编程 ”。

示例：创建本机窗口

Adobe AIR 1.0 和更高版本

以下示例说明如何创建本机窗口：

```
public function createNativeWindow():void {
    //create the init options
    var options:NativeWindowInitOptions = new NativeWindowInitOptions();
    options.transparent = false;
    options.systemChrome = NativeWindowSystemChrome.STANDARD;
    options.type = NativeWindowType.NORMAL;

    //create the window
    var newWindow:NativeWindow = new NativeWindow(options);
    newWindow.title = "A title";
    newWindow.width = 600;
    newWindow.height = 400;

    newWindow.stage.align = StageAlign.TOP_LEFT;
    newWindow.stage.scaleMode = StageScaleMode.NO_SCALE;

    //activate and show the new window
    newWindow.activate();
}
```

管理窗口

Adobe AIR 1.0 和更高版本

使用 NativeWindow 类的属性和方法可管理桌面窗口的外观、行为和生命周期。

注：当使用 Flex 框架时，通常最好使用框架类来管理窗口行为。通过 mx:WindowedApplication 类和 mx:Window 类可访问大多数 NativeWindow 属性和方法。

获取 NativeWindow 实例

Adobe AIR 1.0 和更高版本

若要操作窗口，必须首先获取窗口实例。可以从以下任一位置获取窗口实例：

- 用于创建窗口的本机窗口构造函数：

```
var win:NativeWindow = new NativeWindow(initOptions);
```

- 窗口舞台的 nativeWindow 属性：

```
var win:NativeWindow = stage.nativeWindow;
```

- 窗口中显示对象的 stage 属性：

```
var win:NativeWindow = displayObject.stage.nativeWindow;
```

- 由窗口调度的本机窗口事件的 target 属性：

```
private function onNativeWindowEvent(event:NativeWindowBoundsEvent):void
{
    var win:NativeWindow = event.target as NativeWindow;
}
```

- 窗口中显示的 HTML 页的 nativeWindow 属性：

```
var win:NativeWindow = htmlLoader.window.nativeWindow;
```

- NativeApplication 对象的 activeWindow 和 openedWindows 属性：

```
var nativeWin:NativeWindow = NativeApplication.nativeApplication.activeWindow;
var firstWindow:NativeWindow = NativeApplication.nativeApplication.openedWindows[0];
```

  NativeApplication.nativeApplication.activeWindow 引用应用程序的活动窗口 （但如果该活动窗口不是此 AIR 应用程序的窗口，则返回 null）。 NativeApplication.nativeApplication.openedWindows 数组包含 AIR 应用程序中尚未关闭的所有窗口。

由于 Flex mx:WindowedApplication 和 mx:Window 对象都是显示对象，因此使用 stage 属性即可轻松地在 MXML 文件中引用应用程序窗口，如下所示：

```
 <?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" applicationComplete="init();">
    <mx:Script>
        <![CDATA[
            import flash.display.NativeWindow;

            public function init():void{
                var appWindow:NativeWindow = this.stage.nativeWindow;
                //set window properties
                appWindow.visible = true;
            }
        ]]>
    </mx:Script>
</WindowedApplication
```

注：在 Flex 框架将 WindowedApplication 或 Window 组件添加到窗口舞台后，该组件的 stage 属性为 null。此行为与 Flex Application 组件的行为一致，但并不意味着不能访问舞台或 WindowedApplication 和 Window 组件初始化周期中较早出现事件 （例如 creationComplete）的侦听器中的 NativeWindow 实例。在调度 applicationComplete 事件后，访问舞台和 NativeWindow 实例是安全的。

激活、显示和隐藏窗口

Adobe AIR 1.0 和更高版本

若要激活窗口，请调用 NativeWindow activate() 方法。激活窗口会将该窗口置于前面，为其提供键盘和鼠标焦点，并在必要时通过还原窗口或将 visible 属性设置为 true 来使其可见。激活窗口不会更改应用程序中其他窗口的顺序。调用 activate() 方法会导致窗口调度 activate 事件。

若要在不激活的情况下显示隐藏窗口，请将 visible 属性设置为 true。此操作使该窗口置于前面，但不会向它分配焦点。

若要从视图中隐藏窗口，请将其 visible 属性设置为 false。隐藏窗口会禁止窗口和任何相关任务栏图标的显示，并且在 Mac OS X 中还会禁止窗口菜单中条目的显示。

更改窗口的可见性时，该窗口所拥有的所有窗口的可见性也将发生更改。例如，如果隐藏窗口，则该窗口所拥有的所有窗口也将隐藏。

注：在 Mac OS X 中，无法完全隐藏在停靠栏的窗口部分拥有图标的最小化窗口。如果最小化窗口上的 visible 属性设置为 false，则仍然显示该窗口的停靠栏图标。如果用户单击该图标，则会将该窗口还原为可见状态并显示该窗口。

更改窗口显示顺序

Adobe AIR 1.0 和更高版本

AIR 提供几种方法来直接更改窗口的显示顺序。可以将窗口的显示顺序向前或向后移动；可以将窗口移动到其他窗口的前面或后面。同时，用户可以通过激活窗口来对窗口进行重新排序。

可以通过将窗口的 alwaysInFront 属性设置为 true 来使该窗口位于其他窗口的前面。如果多个窗口都具有此设置，则这些窗口的显示顺序是它们相互之间的排序顺序，而且它们始终排序在 alwaysInFront 设置为 false 的窗口前面。

即使 AIR 应用程序未处于活动状态，最上面组中的窗口也显示在其他应用程序中窗口的前面。由于此行为会阻碍用户查看其他窗口，因此应仅在必要和适当时才能将 alwaysInFront 设置为 true。经调整的使用示例包括：

- 工具提示、弹出列表、自定义菜单或组合框等控件的临时弹出窗口。由于这些窗口在失去焦点后将关闭，因此可以避免阻碍用户查看其他窗口的不便。
- 极其紧急的错误消息和警告。在可能发生不可撤销的更改时，如果用户未及时响应，则可能调整为将警告窗口置于最前面。但是，可以按正常的窗口显示顺序处理大多数错误消息和警告。
- 短期弹出式窗口。

注：AIR 不强制要求 alwaysInFront 属性的正确使用。但是，如果应用程序打断了用户的工作流，则可能将其传递到同一用户的垃圾桶。

如果窗口拥有其他窗口，则这些窗口始终按顺序显示在该窗口前面。如果对拥有其他窗口的某个窗口调用 orderToFront()，或者在该窗口上将 alwaysInFront 设置为 true，则所有者窗口将显示在其他窗口前面，该窗口所拥有的窗口随该窗口一起重新排序，且所拥有的窗口仍显示在所有者的前面。

在同一窗口所拥有的窗口中，对拥有的窗口调用窗口顺序方法可正常工作，但同时会更改整个拥有的窗口组相对于该组之外的其他窗口的排序顺序。例如，如果对某个拥有的窗口调用 orderToFront()，则该窗口、其所有者以及同一所有者所拥有的所有其他窗口都将移动到窗口显示列表的前面。

NativeWindow 类提供以下属性和方法来设置一个窗口相对于其他窗口的显示顺序：

成员	说明
alwaysInFront 属性	指定窗口是否显示在最上面的窗口组中。 几乎在所有情况下， false 都是最佳设置。将值从 false 更改为 true 会将窗口置于所有其他窗口的前面 （但不会激活该窗口）。将值从 true 更改为 false 会将窗口的顺序排在最上面组中其余窗口的后面，但仍位于其他窗口的前面。将窗口的该属性设置为其当前值不会更改窗口显示顺序。 alwaysInFront 设置对其他窗口所拥有的窗口没有任何影响。
orderToFront()	将窗口置于前面。
orderInFrontOf()	将窗口置于紧靠特定窗口前面。
orderToBack()	将窗口发送到其他窗口后面。
orderBehind()	将窗口发送到紧靠特定窗口后面。
activate()	将窗口置于前面 （同时使该窗口可见并分配焦点）。

注：如果窗口处于隐藏 （visible 为 false）或最小化状态，则调用显示顺序方法无效。

在 Linux 操作系统中，不同的窗口管理器对于窗口显示顺序实施不同的规则：

- 在某些窗口管理器中，实用程序窗口始终显示于普通窗口之前。
- 在某些窗口管理器中，将 alwaysInFront 设置为 true 的全屏窗口始终显示于其他同样将 alwaysInFront 设置为 true 的窗口之前。

关闭窗口

Adobe AIR 1.0 和更高版本

若要关闭窗口，请使用 NativeWindow.close() 方法。

关闭窗口会卸载该窗口的内容，但如果其他对象引用了此内容，则不会破坏内容对象。 NativeWindow.close() 方法以异步方式执行，该窗口中包含的应用程序在关闭过程中继续运行。该 close 方法在关闭操作完成时调度 close 事件。从技术角度而言，NativeWindow 对象仍然有效，但访问已关闭窗口上的大多数属性和方法会生成 IllegalOperationError。不能重新打开已关闭窗口。检查窗口的 closed 属性以测试窗口是否已关闭。若要仅从视图中隐藏窗口，请将 NativeWindow.visible 属性设置为 false。

如果 Nativeapplication.autoExit 属性为 true （默认情况），则应用程序在其最后一个窗口关闭后退出。

当所有者关闭时，该所有者所拥有的所有窗口也将同时关闭。所拥有的窗口不会调度关闭事件，因此无法阻止关闭这些窗口。已调度 close 事件。

允许取消窗口操作

Adobe AIR 1.0 和更高版本

在窗口使用系统镶边时，可以通过侦听和取消相应事件的默认行为来取消用户与该窗口的交互。例如，在用户单击系统镶边关闭按钮后，将调度 closing 事件。如果任何已注册的侦听器调用了事件的 preventDefault() 方法，则该窗口不会关闭。

在窗口不使用系统镶边时，不会在执行预期更改之前自动调度这些更改的通知事件。因此，如果调用关闭窗口、更改窗口状态的方法，或者设置任何窗口范围属性，则无法取消更改。若要在执行窗口更改前通知应用程序中的组件，则应用程序逻辑可以使用窗口的 dispatchEvent() 方法调度相关的通知事件。

例如，以下逻辑实现窗口关闭按钮的可取消事件处理函数：

```
public function onCloseCommand(event:MouseEvent):void{
    var closingEvent:Event = new Event(Event.CLOSING,true,true);
    dispatchEvent(closing);
    if(!closingEvent.isDefaultPrevented()){
        win.close();
    }
}
```

如果侦听器调用事件的 preventDefault() 方法，则 dispatchEvent() 方法返回 false。但是，由于其他原因，它还可以返回 false，因此最好明确使用 isDefaultPrevented() 方法来测试是否应取消更改。

最大化、最小化和还原窗口

Adobe AIR 1.0 和更高版本

若要最大化窗口，请使用 NativeWindow maximize() 方法。

```
myWindow.maximize();
```

若要最小化窗口，请使用 NativeWindow minimize() 方法。

```
myWindow.minimize();
```

若要还原窗口 （即，使其返回最小化或最大化操作之前的大小），请使用 NativeWindow restore() 方法。

```
myWindow.restore();
```

当所有者窗口最小化或还原时，所拥有的窗口也将最小化和还原。所拥有的窗口在最小化时不会调度任何事件，因为是对其所有者执行的最小化操作。

注 最大化 AIR 窗口导致的行为与 Mac OS X 标准行为不同。AIR 窗口不是在应用程序定义的 “ 标准 ” 大小和用户最后设置的大小之间切换，而是在应用程序或用户最后设置的大小和屏幕的完整可用区域之间切换。

在 Linux 操作系统中，不同的窗口管理器对于设置窗口显示状态实施不同的规则：

- 在某些窗口管理器中无法将实用程序窗口最大化。
- 如果为窗口设置了最大大小，则某些窗口不允许将窗口最大化。某些其他窗口管理器将显示状态设置为最大化，但不调整窗口大小。在这两种情况下，都不会调度显示状态更改事件。
- 某些窗口管理器不遵守窗口的 maximizable 或 minimizable 设置。

注：在 Linux 中，更改窗口属性是异步进行的。如果在程序的一行中更改窗口的显示状态，并在下一行读取该值，则对值的读取仍将受旧的设置影响。在所有平台上，当显示状态发生更改时， NativeWindow 对象将调度 displayStateChange 事件。如果您需要根据窗口的新状态执行某种动作，始终在 displayStateChange 事件处理函数中执行。请参阅第 782 页的 “ 侦听窗口事件 ”。

示例：最小化、最大化、还原和关闭窗口

Adobe AIR 1.0 和更高版本

以下简短的 MXML 应用程序演示 Window maximize()、 minimize()、 restore() 和 close() 方法：

```
<?xml version="1.0" encoding="utf-8"?>

<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical">


    <mx:Script>
    <![CDATA[
    public function minimizeWindow():void
    {
        this.stage.nativeWindow.minimize();
    }

    public function maximizeWindow():void
    {
        this.stage.nativeWindow.maximize();
    }

    public function restoreWindow():void
    {
        this.stage.nativeWindow.restore();
    }

    public function closeWindow():void
    {
        this.stage.nativeWindow.close();
    }
    ]]>
    </mx:Script>

    <mx:VBox>
        <mx:Button label="Minimize" click="minimizeWindow()"/>
        <mx:Button label="Restore" click="restoreWindow()"/>
        <mx:Button label="Maximize" click="maximizeWindow()"/>
        <mx:Button label="Close" click="closeWindow()"/>
    </mx:VBox>

</mx:WindowedApplication>
```

以下用于 Flash 的 ActionScript 示例创建四个可单击的文本字段，这些字段分别触发 NativeWindow minimize()、maximize()、 restore() 和 close() 方法：

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.text.TextField;

    public class MinimizeExample extends Sprite
    {
        public function MinimizeExample():void
        {
            var minTextBtn:TextField = new TextField();
            minTextBtn.x = 10;
            minTextBtn.y = 10;
            minTextBtn.text = "Minimize";
            minTextBtn.background = true;
            minTextBtn.border = true;
            minTextBtn.selectable = false;
            addChild(minTextBtn);
            minTextBtn.addEventListener(MouseEvent.CLICK, onMinimize);
```

```
            var maxTextBtn:TextField = new TextField();
            maxTextBtn.x = 120;
            maxTextBtn.y = 10;
            maxTextBtn.text = "Maximize";
            maxTextBtn.background = true;
            maxTextBtn.border = true;
            maxTextBtn.selectable = false;
            addChild(maxTextBtn);
            maxTextBtn.addEventListener(MouseEvent.CLICK, onMaximize);

            var restoreTextBtn:TextField = new TextField();
            restoreTextBtn.x = 230;
            restoreTextBtn.y = 10;
            restoreTextBtn.text = "Restore";
            restoreTextBtn.background = true;
            restoreTextBtn.border = true;
            restoreTextBtn.selectable = false;
            addChild(restoreTextBtn);
            restoreTextBtn.addEventListener(MouseEvent.CLICK, onRestore);

            var closeTextBtn:TextField = new TextField();
            closeTextBtn.x = 340;
            closeTextBtn.y = 10;
            closeTextBtn.text = "Close Window";
            closeTextBtn.background = true;
            closeTextBtn.border = true;
            closeTextBtn.selectable = false;
            addChild(closeTextBtn);
            closeTextBtn.addEventListener(MouseEvent.CLICK, onCloseWindow);
        }
        function onMinimize(event:MouseEvent):void
        {
            this.stage.nativeWindow.minimize();
        }
        function onMaximize(event:MouseEvent):void
        {
            this.stage.nativeWindow.maximize();
        }
        function onRestore(event:MouseEvent):void
        {
            this.stage.nativeWindow.restore();
        }
        function onCloseWindow(event:MouseEvent):void
        {
            this.stage.nativeWindow.close();
        }
    }
}
```

调整窗口大小和移动窗口

Adobe AIR 1.0 和更高版本

在窗口使用系统镶边时，该镶边提供用于调整窗口大小和在桌面范围内移动窗口的拖动控件。如果窗口不使用系统镶边，则必须添加您自己的控件以允许用户调整窗口大小和移动窗口。

注：若要调整窗口大小或移动窗口，必须首先获取对 NativeWindow 实例的引用。有关如何获取窗口引用的信息，请参阅第 775 页的 “ 获取 NativeWindow 实例 ”。

调整窗口大小

若要使用户可以交互地调整窗口大小，请使用 NativeWindow startResize() 方法。如果此方法是从 mouseDown 事件中调用的，则调整大小操作由鼠标驱动并在操作系统收到 mouseUp 事件时完成。在调用 startResize() 时，传入用于指定所调整窗口大小的起始边或角的参数。

若要以编程方式设置窗口大小，请将窗口的 width、 height 或 bounds 属性设置为所需尺寸。设置范围时，可以同时更改窗口的大小和位置。但是，无法保证发生更改的顺序。某些 Linux 窗口管理器不允许窗口扩展到桌面屏幕范围之外。在这些情况下，即使更改的最终效果以其他方式产生了合法的窗口，最终窗口大小也可能因属性的设置顺序而受到限制。例如，如果同时更改靠近屏幕底部的窗口的高度和 Y 轴位置，那么在 Y 轴位置更改之前应用高度更改时，可能不会进行完整的高度更改。

注：在 Linux 中，更改窗口属性是异步进行的。如果在程序的一行中调整窗口大小，并在下一行读取尺寸，则这些尺寸仍将受旧的设置影响。在所有平台上，当调整窗口大小时， NativeWindow 对象将调度 resize 事件。如果您需要根据窗口的新大小或状态执行某种动作 （如设置窗口中控件的布局），始终在 resize 事件处理函数中执行。请参阅第 782 页的 " 侦听窗口事件 "。

舞台的缩放模式确定调整窗口大小时窗口舞台及其内容的行为方式。请记住，舞台缩放模式旨在用于应用程序不控制其显示区大小或高宽比的情况，例如 Web 浏览器。通常，通过将舞台的 scaleMode 属性设置为 StageScaleMode.NO_SCALE 来获得最佳效果。如果要缩放窗口的内容，则仍可以根据窗口范围的更改来设置内容的 scaleX 和 scaleY 参数。

移动窗口

要移动窗口而不调整其大小，请使用 NativeWindow startMove() 方法。与 startResize() 方法相似，在从 mouseDown 事件调用 startMove() 方法时，移动过程由鼠标驱动并在操作系统收到 mouseUp 事件时完成。

有关 startResize() 和 startMove() 方法的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

若要以编程方式移动窗口，将窗口的 x、 y 或 bounds 属性设置为所需的位置。设置范围时，可以同时更改窗口的大小和位置。

注：在 Linux 中，更改窗口属性是异步进行的。如果在程序的一行中移动窗口，并在下一行读取该位置，则对值的读取仍将受旧的设置影响。在所有平台上，当更改位置时， NativeWindow 对象将调度 move 事件。如果您需要根据窗口的新位置执行某种动作，始终在 move 事件处理函数中执行。请参阅第 782 页的 " 侦听窗口事件 "。

示例：调整窗口大小和移动窗口

Adobe AIR 1.0 和更高版本

以下示例说明如何对窗口启动调整大小和移动操作：

```
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    import flash.display.NativeWindowResize;

    public class NativeWindowResizeExample extends Sprite
    {
        public function NativeWindowResizeExample():void
        {
            // Fills a background area.
            this.graphics.beginFill(0xFFFFFF);
            this.graphics.drawRect(0, 0, 400, 300);
            this.graphics.endFill();

            // Creates a square area where a mouse down will start the resize.
            var resizeHandle:Sprite =
                createSprite(0xCCCCCC, 20, this.width - 20, this.height - 20);
            resizeHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartResize);

            // Creates a square area where a mouse down will start the move.
            var moveHandle:Sprite = createSprite(0xCCCCCC, 20, this.width - 20, 0);
```

```
            moveHandle.addEventListener(MouseEvent.MOUSE_DOWN, onStartMove);
        }

        public function createSprite(color:int, size:int, x:int, y:int):Sprite
        {
            var s:Sprite = new Sprite();
            s.graphics.beginFill(color);
            s.graphics.drawRect(0, 0, size, size);
            s.graphics.endFill();
            s.x = x;
            s.y = y;
            this.addChild(s);
            return s;
        }

        public function onStartResize(event:MouseEvent):void
        {
            this.stage.nativeWindow.startResize(NativeWindowResize.BOTTOM_RIGHT);
        }

        public function onStartMove(event:MouseEvent):void
        {
            this.stage.nativeWindow.startMove();
        }
    }
}
```

侦听窗口事件

Adobe AIR 1.0 和更高版本

若要侦听窗口调度的事件，请向窗口实例注册侦听器。例如，若要侦听 closing 事件，请按如下方式向窗口实例注册侦听器：

```
myWindow.addEventListener(Event.CLOSING, onClosingEvent);
```

在调度事件时， target 属性引用发送该事件的窗口。

大多数窗口事件都有两条相关消息。第一条消息发出即将发生窗口更改 （可以取消）的信号通知，第二条消息发出更改已发生的信号通知。例如，在用户单击窗口的关闭按钮后，将调度 closing 事件消息。如果侦听器未取消该事件，则窗口将关闭并且 close 事件将调度到所有侦听器。

通常，仅在已使用系统镶边触发事件时才调度 closing 等警告事件。例如，调用 window close() 方法不会自动调度 closing 事件，而只调度 close 事件。但是，您可以构造 closing 事件对象并使用 window dispatchEvent() 方法来调度它。

调度 Event 对象的窗口事件包括：

事件	说明
activate	在窗口收到焦点时调度。
deactivate	在窗口失去焦点时调度
closing	在窗口即将关闭时调度。仅当在按下系统镶边关闭按钮时或者在 Mac OS X 中调用 Quit 命令时，此事件才自动发生。
close	在窗口关闭时调度。

调度 NativeWindowBoundsEvent 对象的窗口事件包括：

事件	说明
moving	在窗口左上角由于移动窗口、调整窗口大小或更改窗口显示状态而更改位置的前一刻调度。
move	在左上角更改位置之后调度。
resizing	在窗口宽度或高度由于调整大小或显示状态更改而发生更改的前一刻调度。
resize	在窗口更改大小之后调度。

对于 NativeWindowBoundsEvent 事件，可以使用 beforeBounds 和 afterBounds 属性确定即将进行更改或完成更改之前和之后的窗口范围。

调度 NativeWindowDisplayStateEvent 对象的窗口事件包括：

事件	说明
displayStateChanging	在窗口显示状态更改的前一刻调度。
displayStateChange	在窗口显示状态更改之后调度。

对于 NativeWindowDisplayStateEvent 事件，可以使用 beforeDisplayState 和 afterDisplayState 属性确定即将进行更改或完成更改之前和之后的窗口显示状态。

在某些 Linux 窗口管理器中，将具有最大化大小设置的窗口最大化时并不调度显示状态更改事件。（窗口设置为最大化显示状态，但并不调整其大小。）

显示全屏窗口

Adobe AIR 1.0 和更高版本

将 Stage 的 displayState 属性设置为 StageDisplayState.FULL_SCREEN_INTERACTIVE 会使窗口进入全屏模式，在此模式下允许键盘输入。（在浏览器中运行的 SWF 内容中，不允许键盘输入）。若要退出全屏模式，用户需要按 Esc 键。

注：如果为窗口设置了最大大小，某些 Linux 窗口管理器不会更改窗口尺寸以适应屏幕 （但会删除窗口的系统镶边）。

例如，以下 Flex 代码定义用于设置简单全屏端点的简单 AIR 应用程序：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()" backgroundColor="0x003030" focusRect="false">
    <mx:Script>
        <![CDATA[
            private function init():void
            {
                stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
                focusManager.setFocus(terminal);
                terminal.text = "Welcome to the dumb terminal app. Press the ESC key to exit..\n";
                terminal.selectionBeginIndex = terminal.text.length;
                terminal.selectionEndIndex = terminal.text.length;
            }
        ]]>
    </mx:Script>
    <mx:TextArea
        id="terminal"
        height="100%" width="100%"
        scroll="false"
        backgroundColor="0x003030"
        color="0xCCFF00"
        fontFamily="Lucida Console"
        fontSize="44"/>
</mx:WindowedApplication>
```

以下用于 Flash 的 ActionScript 示例模拟简单全屏文本端点：

```
import flash.display.Sprite;
import flash.display.StageDisplayState;
import flash.text.TextField;
import flash.text.TextFormat;

public class FullScreenTerminalExample extends Sprite
{
    public function FullScreenTerminalExample():void
    {
        var terminal:TextField = new TextField();
        terminal.multiline = true;
        terminal.wordWrap = true;
        terminal.selectable = true;
        terminal.background = true;
        terminal.backgroundColor = 0x00333333;

        this.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;

        addChild(terminal);
        terminal.width = 550;
        terminal.height = 400;

        terminal.text = "Welcome to the dumb terminal application. Press the ESC key to exit.\n_";

        var tf:TextFormat = new TextFormat();
        tf.font = "Courier New";
        tf.color = 0x00CCFF00;
        tf.size = 12;
        terminal.setTextFormat(tf);

        terminal.setSelection(terminal.text.length - 1, terminal.text.length);
    }
}
```

第 53 章：AIR 中的显示屏幕

Adobe AIR 1.0 和更高版本

使用 Adobe® AIR® Screen 类访问关于连接到计算机或设备的显示屏幕的信息。

更多帮助主题
flash.display.Screen

AIR 中的显示屏幕的基础知识

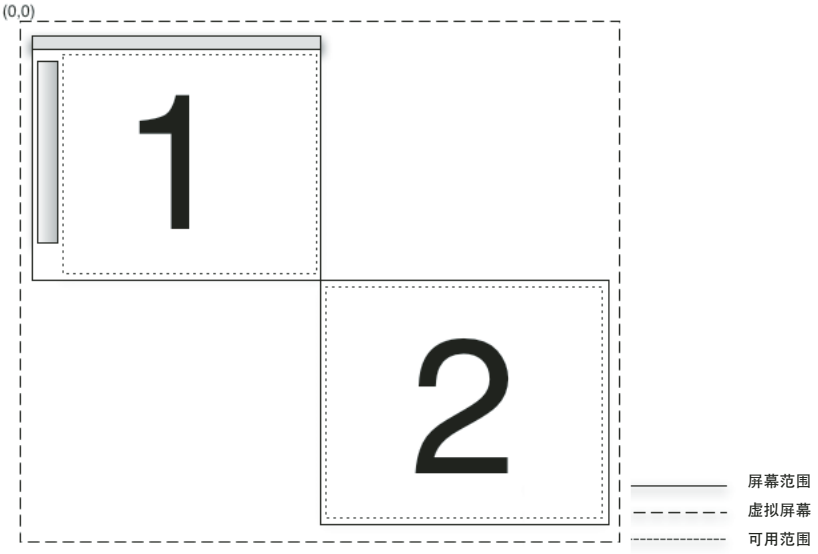
Adobe AIR 1.0 和更高版本

- 测量虚拟桌面 (Flex)
- 测量虚拟桌面 (Flash)

屏幕 API 只包含单个 Screen 类，该类提供用于获取系统屏幕信息的静态成员以及用于描述特定屏幕的实例成员。

计算机系统可以连接多台监视器或显示器，这些监视器或显示器对应于虚拟空间中排列的多个桌面屏幕。 AIR Screen 类提供有关这些屏幕、屏幕的相对排列及其可用空间的信息。如果多台监视器映射到同一屏幕，则只存在一个屏幕。如果屏幕的尺寸大于监视器的显示区域，则无法确定当前可以看到屏幕的哪个部分。

屏幕表示独立的桌面显示区域。屏幕被描述为虚拟桌面内部的矩形。被指定为主显示屏的屏幕的左上角是虚拟桌面坐标系的原点。用于描述屏幕的所有值均以像素为单位提供。



在此屏幕排列中，虚拟桌面中存在两个屏幕。主屏幕 (#1) 左上角的坐标始终是 (0,0)。如果屏幕排列更改为指定屏幕 #2 作为主屏幕，则屏幕 #1 的坐标将变为负值。报告屏幕的可用范围时将排除菜单栏、任务栏和停靠栏。

有关屏幕 API 类、方法、属性和事件的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

枚举屏幕

Adobe AIR 1.0 和更高版本

可以使用以下屏幕方法和属性枚举虚拟桌面的屏幕：

方法或属性	说明
Screen.screens	提供描述可用屏幕的 Screen 对象的数组。数组的顺序不重要。
Screen.mainScreen	提供主屏幕的 Screen 对象。在 Mac OS X 中，主屏幕是指显示菜单栏的屏幕。在 Windows 中，主屏幕是指系统指定的主屏幕。
Screen.getScreensForRectangle()	提供描述与给定矩形相交的屏幕的 Screen 对象的数组。传递给此方法的矩形位于虚拟桌面上的像素坐标中。如果没有屏幕与该矩形相交，则该数组为空。可以使用此方法确定窗口显示在哪些屏幕上。

不要保存 Screen 类方法和属性返回的值。用户或操作系统可随时更改可用屏幕及其排列方式。

以下示例使用屏幕 API 在多个屏幕间移动窗口，以响应箭头键的按键操作。该示例获取 screens 数组并将其在垂直或水平方向排序 （取决于所按的箭头键），以便将窗口移动到下一个屏幕。代码随后将遍历排序后的数组，将每个屏幕与当前屏幕的坐标进行比较。该示例调用 Screen.getScreensForRectangle()，传入窗口范围，以便识别窗口的当前屏幕。

```
package {
    import flash.display.Sprite;
    import flash.display.Screen;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;

    public class ScreenExample extends Sprite
    {
        public function ScreenExample()
        {
                stage.align = StageAlign.TOP_LEFT;
                stage.scaleMode = StageScaleMode.NO_SCALE;

                stage.addEventListener(KeyboardEvent.KEY_DOWN,onKey);
        }

        private function onKey(event:KeyboardEvent):void{
            if(Screen.screens.length > 1){
                switch(event.keyCode){
                    case Keyboard.LEFT :
                        moveLeft();
                        break;
                    case Keyboard.RIGHT :
                        moveRight();
                        break;
                    case Keyboard.UP :
                        moveUp();
                        break;
                    case Keyboard.DOWN :
                        moveDown();
                        break;
                }
            }
        }

        private function moveLeft():void{
            var currentScreen = getCurrentScreen();
```

```
        var left:Array = Screen.screens;
        left.sort(sortHorizontal);
        for(var i:int = 0; i < left.length - 1; i++){
            if(left[i].bounds.left < stage.nativeWindow.bounds.left){
                stage.nativeWindow.x +=
                    left[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
            }
        }
    }

    private function moveRight():void{
        var currentScreen:Screen = getCurrentScreen();
        var left:Array = Screen.screens;
        left.sort(sortHorizontal);
        for(var i:int = left.length - 1; i > 0; i--){
            if(left[i].bounds.left > stage.nativeWindow.bounds.left){
                stage.nativeWindow.x +=
                    left[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += left[i].bounds.top - currentScreen.bounds.top;
            }
        }
    }

    private function moveUp():void{
        var currentScreen:Screen = getCurrentScreen();
        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = 0; i < top.length - 1; i++){
            if(top[i].bounds.top < stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function moveDown():void{
        var currentScreen:Screen = getCurrentScreen();

        var top:Array = Screen.screens;
        top.sort(sortVertical);
        for(var i:int = top.length - 1; i > 0; i--){
            if(top[i].bounds.top > stage.nativeWindow.bounds.top){
                stage.nativeWindow.x += top[i].bounds.left - currentScreen.bounds.left;
                stage.nativeWindow.y += top[i].bounds.top - currentScreen.bounds.top;
                break;
            }
        }
    }

    private function sortHorizontal(a:Screen,b:Screen):int{
        if (a.bounds.left > b.bounds.left){
            return 1;
        } else if (a.bounds.left < b.bounds.left){
```

```
                return -1;
            } else {return 0;}
        }

        private function sortVertical(a:Screen,b:Screen):int{
            if (a.bounds.top > b.bounds.top){
                return 1;
            } else if (a.bounds.top < b.bounds.top){
                return -1;
            } else {return 0;}
        }

        private function getCurrentScreen():Screen{
            var current:Screen;
            var screens:Array = Screen.getScreensForRectangle(stage.nativeWindow.bounds);
            (screens.length > 0) ? current = screens[0] : current = Screen.mainScreen;
            return current;
        }
    }
}
```

第 54 章：打印

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash 运行时 （如 Adobe® Flash® Player 和 Adobe® AIR™）可与操作系统的打印接口通信，使您能将页面传递到打印后台处理程序。发送到后台处理程序的每个页面都可能包含可见的、动态的或屏幕上未显示给用户的内容，其中包括数据库值和动态文本。另外， flash.printing.PrintJob 类的属性包含基于用户打印机设置的值，以便您能适当设置页面的格式。

以下内容详细介绍使用 flash.printing.PrintJob 类方法和属性创建打印作业、读取用户的打印设置，并基于来自 Flash 运行时和用户操作系统的反馈来调整打印作业的策略。

打印基础知识

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 ActionScript 3.0 中，可以使用 PrintJob 类来创建显示内容的快照以转换为打印输出中的墨水和纸张表示形式。在某些方面，设置要打印的内容与设置在屏幕上显示的内容是相同的；即可以放置元素和调整其大小以创建所需的布局。但是，打印具有某些特性，而使其不同于屏幕布局。例如，打印机使用的分辨率不同于计算机显示器；计算机屏幕的内容是动态的并且可能会发生变化，而打印的内容本身静态的；在计划打印时，请考虑固定纸张大小的限制和多页打印的可能性。

即使这些区别看起来很明显，但在使用 ActionScript 设置打印时一定要记住这些不同之处。精确打印取决于您指定的值和用户打印机的特性。 PrintJob 类包括允许您确定用户打印机的重要特性的属性。

重要概念和术语

以下参考列表包含与打印相关的重要术语：

后台处理程序 操作系统或打印机驱动程序软件的一部分，用于在等待打印时跟踪页面，并在打印机可用时将页面发送到打印机。

页面方向 打印的内容相对于纸张的旋转角度 — 水平 （横向）或垂直 （纵向）。

打印作业 组成单一打印输出的一个页面或一组页面。

打印页面

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 PrintJob 类的实例来处理打印。要通过 Flash Player 或 AIR 打印基本页面，请依次使用下面四个语句：

- new PrintJob()：创建指定的打印作业名称的新实例。
- PrintJob.start()：启动操作系统的打印过程、为用户调用打印对话框并填充打印作业的只读属性。
- PrintJob.addPage()：包含有关打印作业内容的详细信息，包括 Sprite 对象 （及其包含的任何子对象）、打印区域的大小以及打印机将图像作为矢量打印还是作为位图打印。您可以使用对 addPage() 的连续调用，在多个页面上打印多个 sprite。
- PrintJob.send()：将页面发送到操作系统的打印机。

因此，举例来说，一份简单的打印作业脚本如下 （包括用于编译的 package、 import 和 class 语句）：

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;

    public class BasicPrintExample extends Sprite
    {
        var myPrintJob:PrintJob = new PrintJob();
        var mySprite:Sprite = new Sprite();

        public function BasicPrintExample()
        {
            myPrintJob.start();
            myPrintJob.addPage(mySprite);
            myPrintJob.send();
        }
    }
}
```

注：此示例是为了说明打印作业脚本的基本元素，不包含任何错误处理。要生成脚本以正确响应用户取消打印作业的操作，请参阅第 790 页的 “ 处理异常和返回值 ”。

由于某种原因要清除 PrintJob 对象的属性，请将 PrintJob 变量设置为 null （如 myPrintJob = null 中所示）。

Flash 运行时任务和系统打印

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

由于 Flash 运行时向操作系统的打印接口调度页面，因此请了解 Flash 运行时管理的任务和操作系统自己的打印接口管理的任务。 Flash 运行时可以启动打印作业、读取打印机的一些页面设置、将打印作业的内容传递到操作系统以及验证用户或系统是否取消了打印作业。其他过程 （例如显示特定于打印机的对话框、取消后台打印的打印作业或报告打印机的状态）均由操作系统进行处理。如果出现打印作业启动或格式设置问题， Flash 运行时能够做出响应，但只能返回关于操作系统打印接口的某些属性或条件的报告。作为开发人员，您的代码需要响应这些属性或条件。

处理异常和返回值

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在用户已取消打印作业的情况下，检查在执行 addPage() 和 send() 调用之前，PrintJob.start() 方法是否返回 true。一种在继续之前检查是否已取消这些方法的简单方式是将这些方法包在 if 语句中，如下所示：

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

如果 PrintJob.start() 为 true，则用户选择 Print （或 Flash 运行时，例如 Flash Player 或 AIR，已启动 Print 命令）。因此，可以调用 addPage() 和 send() 方法。

此外，为帮助管理打印过程， Flash 运行时为 PrintJob.addPage() 方法引发异常，以使您能捕获错误并向用户提供信息和选项。如果 PrintJob.addPage() 方法失败，则可以调用其他函数或停止当前打印作业。通过将 addPage() 调用嵌入 try..catch 语句，可以捕获这些异常，如下面的示例所示。在该例中， [params] 是指定实际打印内容的参数的占位符：

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

打印作业启动后，您可以使用 PrintJob.addPage() 添加内容并查看是否生成异常 （例如，如果用户已取消打印作业）。如果产生异常，您可以向 catch 语句添加逻辑，从而向用户 （或 Flash 运行时）提供信息和选项，您也可以停止当前打印作业。如果成功添加了页面，则可以继续使用 PrintJob.send() 将页面发送到打印机。

如果 Flash 运行时在向打印机发送打印作业时遇到问题 （例如，如果打印机脱机），您也可以捕获到该异常，并提供详细信息或更多选项 （例如显示消息文本或在动画中提供警告）。例如，您可以为 if..else 语句中的文本字段分配新文本，如下面的代码所示：

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

有关运行正常的示例，请参阅第 798 页的 “ 打印示例：缩放、裁剪和响应 ”。

使用页面属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用户在 “ 打印 ” 对话框中单击 “ 确定 ” 且 PrintJob.start() 返回 true 后，您就可以访问由打印机设置定义的属性。这些设置包括纸张宽度、纸张高度 （pageHeight 和 pageWidth）以及纸张上的内容方向。由于这些设置是打印机设置，不受 Flash 运行时控制，因此您无法更改它们；但是，您可以使用它们调整发送到打印机的内容以与当前设置匹配。有关详细信息，请参阅第 792 页的 “ 设置大小、缩放和方向 ”。

设置矢量或位图呈示

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以将打印作业手动设置为将每个页面在后台处理为矢量图形或位图图像。在某些情况下，相对于位图打印而言，矢量打印产生的后台处理文件更小，且图像效果更好。但是，如果您的内容中包含位图图像，并希望保留任何 Alpha 透明度或颜色效果，则请将此页打印为位图图像。另外，非 PostScript 打印机会自动将任何矢量图形转换为位图图像。

可以通过将 PrintJobOptions 对象作为 PrintJob.addPage() 的第三个参数传递来指定位图打印。

对于 Flash Player 和早于 AIR 2 的 AIR，请将 PrintJobOptions 对象的 printAsBitmap 参数集设置为 true，如下所示：

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

如果没有为第三个参数指定值，则打印作业会使用默认打印，即矢量打印。

对于 AIR 2 及更高版本，请使用 PrintJobOptions 对象的 printMethod 属性来指定打印方法。此属性接受三个值，这三个值定义为 PrintMethod 中的常量：

- PrintMethod.AUTO：根据要打印的内容自动选择最佳的打印方法。例如，如果页面包含文本，则选择矢量打印方法。然而，如果带有 alpha 透明度的水印图像叠加在文本上，则会选择位图打印以保留该透明度。
- PrintMethod.BITMAP：无论内容如何，强制进行位图打印
- PrintMethod.VECTOR：无论内容如何，强制进行矢量打印

为打印作业语句定时

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

与 ActionScript 的先前版本不同， ActionScript 3.0 未将 PrintJob 对象限定在单帧。然而，因为用户在 " 打印 " 对话框中单击 " 确定 " 按钮后，操作系统会向用户显示打印状态信息，所以请尽快调用 PrintJob.addPage() 和 PrintJob.send() 以将页面发送到后台处理程序。如果到达包含 PrintJob.send() 调用的帧时发生延迟，则会延迟打印过程。

在 ActionScript 3.0 中，脚本超时限制为 15 秒。因此，打印作业序列中各个主要语句间的时间间隔不能超过 15 秒。也就是说， 15 秒脚本超时限制适用于以下时间间隔：

- PrintJob.start() 和第一个 PrintJob.addPage() 之间
- PrintJob.addPage() 和下一个 PrintJob.addPage() 之间
- 最后一个 PrintJob.addPage() 和 PrintJob.send() 之间

如果以上任何一个间隔时间超过 15 秒，则对 PrintJob 实例的下一次 PrintJob.start() 调用会返回 false，并且下次对 PrintJob 实例执行 PrintJob.addPage() 时会导致 Flash Player 或 AIR 引发运行时异常。

设置大小、缩放和方向

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

第 789 页的 " 打印页面 " 一节详细介绍基本打印作业的步骤，在基本打印作业中，输出直接反映了指定的 Sprite 的屏幕大小和位置的打印效果。然而，打印机使用不同的分辨率进行打印，并且可以具有对打印 sprite 的外观有不利影响的设置。

Flash 运行时可以读取操作系统的打印设置，但是请注意这些属性是只读的：虽然您可以响应它们的值，但无法对它们进行设置。例如，您可以查明打印机的页面大小设置，并调整内容以适合该大小。您还可以确定打印机的边距设置和页面方向。要响应打印机设置，请指定打印区域，调整屏幕的分辨率和打印机的点测量之间的差异，或者转换您的内容以适合用户打印机的大小或方向设置。

为打印区域使用矩形

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 PrintJob.addPage() 方法可以指定要打印的 sprite 的区域。第二个参数 printArea 为 Rectangle 对象的形式。您可以选择三种方法来提供该参数的值：

- 创建一个具有特定属性的 Rectangle 对象，然后将该矩形用于 addPage() 调用，如下例所示：

```
private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
myPrintJob.addPage(sheet, rect1);
```

- 如果您尚未指定 Rectangle 对象，则可以在该调用本身内指定，如下面的示例所示：

```
myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
```

- 如果您打算为 addPage() 调用中的第三个参数提供值，但不想指定矩形，则可以对第二个参数使用 null，如下所示：

```
myPrintJob.addPage(sheet, null, options);
```

比较点和像素

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

矩形的宽度和高度以像素为单位。打印机使用点来作为打印的度量单位。点的实际大小是固定的 （1/72 英寸），但是在屏幕上，像素的大小取决于特定屏幕的分辨率。像素和点之间的转换比率取决于打印机设置以及 sprite 是否经过缩放。一个 72 像素宽的 sprite 在未经缩放的情况下打印输出为一英寸宽，这时，一点等于一像素，与屏幕分辨率无关。

您可以使用以下换算公式将英寸或厘米转换为缇或点 （1 缇为 1/20 点）：

- 1 点 = 1/72 英寸 = 20 缇
- 1 英寸 = 72 点 = 1440 缇
- 1 厘米 = 567 缇

如果省略了 printArea 参数或错误地传递了该参数，将打印 sprite 的整个区域。

缩放

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果要在打印前对 Sprite 对象进行缩放，请在调用 PrintJob.addPage() 方法之前设置缩放属性 （请参阅第 149 页的 “ 处理大小和缩放对象 ”），并在打印后将这些属性设置回原始值。Sprite 对象的缩放与 printArea 属性无关。也就是说，如果指定一个 50 x 50 个像素的打印区域，则会打印 2500 个像素。如果对 Sprite 对象进行缩放，则同样会打印 2500 个像素，但按缩放后的大小打印 Sprite 对象。

有关示例，请参阅第 798 页的 “ 打印示例：缩放、裁剪和响应 ”。

横向或纵向打印

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

由于 Flash Player 和 AIR 可以检测方向设置，因此，您可以在 ActionScript 中构建逻辑来调整内容大小或旋转以响应打印机设置，如下面的示例所示：

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

注：如果您计划读取纸张上内容方向的系统设置，请记得导入 PrintJobOrientation 类。 PrintJobOrientation 类提供了定义页面上的内容方向的常量值。使用以下语句导入该类：

```
import flash.printing.PrintJobOrientation;
```

响应页面高度和宽度

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用类似于处理打印机方向设置的策略，可以读取页面高度和宽度设置，并通过将某些逻辑嵌入 if 语句中来响应这些设置。下面的代码显示了一个示例：

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

此外，还可以通过比较页面尺寸和纸张尺寸来确定页面的边距设置，如下面的示例所示：

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

高级打印技术

Adobe AIR 2 和更高版本

从 Adobe AIR 2 开始， PrintJob 类包含附加属性和方法，并支持三个附加类：PrintUIOptions、 PaperSize 和 PrintMethod。这些更改支持附加打印机工作流程，使作者可以更好地控制打印过程。这些更改包括：

- “ 页面设置 ” 对话框: 可显示标准和自定义 “ 页面设置 ” 对话框。在打印之前，用户可以设置页面范围、页面大小、页面方向和页面缩放。
- 打印视图：一种查看模式，可准确显示页面大小、页面边距和内容在页面上的位置。
- 受限打印：作者可以限制打印选项，如可打印页面范围。
- “ 品质 ” 选项：作者可调整文档的打印品质并允许用户选择分辨率和颜色选项。
- 多个打印会话: 现在单个 PrintJob 实例可用于多个打印会话。每次显示 “ 页面设置 ” 和 “ 打印 ” 对话框时，应用程序都可提供一致的设置。

打印工作流更改

新建打印工作流程包括下列步骤：

- new PrintJob()：创建 PrintJob 实例（或重用现有的实例）。在启动打印作业之前或在打印期间可以使用许多新的 PrintJob 属性和方法，例如 selectPaperSize()。
- PrintJob.showPageSetupDialog()：（可选）在不启动打印作业的情况下显示 “ 页面设置 ” 对话框。
- PrintJob.start() 或 PrintJob.start2()：除了 start() 方法之外，还可以使用 start2() 方法来启动后台打印过程。借助 start2() 方法，您可以选择是否显示 “ 打印 ” 对话框，如果不显示该对话框，是否对其进行自定义。
- PrintJob.addPage()：将内容添加到打印作业。不会更改现有进程。

- PrintJob.send() 或 PrintJob.terminate()：将页面发送到所选的打印机或在不发送的情况下终止打印作业。出现错误时，终止打印作业。如果终止 PrintJob 实例，仍然可以重用它。无论是将打印作业发送到打印机还是终止打印作业，重用 PrintJob 实例时，都会保留当前打印设置。

“ 页面设置 ” 对话框

showPageSetupDialog() 方法将显示操作系统的 “ 页面设置 ” 对话框，如果当前环境支持此功能。在调用此方法之前，始终先查看 supportsPageSetupDialog 属性。以下是一个简单示例：

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
//check for static property supportsPageSetupDialog of PrintJob class
if (PrintJob.supportsPageSetupDialog) {
    myPrintJob.showPageSetupDialog();
}
```

可使用 PrintUIOptions 类属性选择性地调用此方法以控制在 “ 页面设置 ” 对话框中显示哪些选项。可设置最小和最大页码。在以下示例中，限制只打印前三个页面：

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
if (PrintJob.supportsPageSetupDialog) {
    var uiOpt:PrintUIOptions = new PrintUIOptions();
    uiOpt.minPage = 1;
    uiOpt.maxPage = 3;
    myPrintJob.showPageSetupDialog(uiOpt);
}
```

更改打印设置

在构建 PrintJob 实例之后，可以随时更改其设置。这包括在 addPage() 调用之间以及在发送或终止打印作业之后更改设置。一些设置 （例如 printer 属性）应用于整个打印作业而不是单个页面。这些设置必须在调用 start() 或 start2() 之前设置。

通过调用 selectPaperSize() 方法，可以设置 “ 页面设置 ” 对话框和 “ 打印 ” 对话框中的默认页面大小。也可以在打印作业期间调用该方法以设置一系列页面的页面大小。使用在 PaperSize 类中定义的常量调用它，如在以下示例中，选择了 10 号信封尺寸：

```
import flash.printing.PrintJob;
import flash.printing.PaperSize;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.selectPaperSize(PaperSize.ENV_10);
```

使用 printer 属性获取或设置当前打印作业的打印机名称。默认情况下，设置为默认打印机的名称。如果打印机不可用或系统不支持打印，则 printer 属性为 null。要更改打印机，首先使用 printers 属性获取可用打印机列表。该属性是其 String 元素为可用打印机名称的矢量。将 printer 属性设置为这些 String 值之一以使该打印机成为活动打印机。活动打印作业的 printer 属性无法更改。在成功调用 start() 或 start2() 之后以及在发送或终止作业之前，任何对此属性的更改尝试都将失败。在以下示例中，对此属性进行了设置：

```
import flash.printing.PrintJob;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.printer = "HP_LaserJet_1";
myPrintJob.start();
```

copies 属性获取在操作系统的 “ 打印 ” 对话框中设置的打印份数值。firstPage 和 lastPage 属性获取页面范围。orientation 属性获取纸张方向设置。可以对这些属性进行设置以覆盖 “ 打印 ” 对话框中的值。在以下示例中，设置了这些属性：

```
import flash.printing.PrintJob;
import flash.printing.PrintJobOrientation;

var myPrintJob:PrintJob = new PrintJob();
myPrintJob.copies = 3;
myPrintJob.firstPage = 1;
myPrintJob.lastPage = 3;
myPrintJob.orientation = PrintJobOrientation.LANDSCAPE;
```

通过下列与 PrintJob 关联的只读设置，可以了解有关当前打印机设置的有用信息：

- paperArea：打印机媒体的矩形范围 （以点为单位）。
- printableArea：可打印区域的矩形范围 （以点为单位）。
- maxPixelsPerInch：当前打印机的物理分辨率 （以每英寸像素为单位）。
- isColor：当前打印机能否打印颜色 （如果可以，则返回 true）。

请参阅第 799 页的 “ 打印示例：页面设置和打印选项 ”。

打印示例：多页面打印

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

打印多页内容时，可以将每一页内容与不同的 sprite （本例中为 sheet1 和 sheet2）关联，然后为每个 sprite 使用 PrintJob.addPage()。以下代码说明了这种方法：

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;

    public class PrintMultiplePages extends MovieClip
    {
        private var sheet1:Sprite;
        private var sheet2:Sprite;

        public function PrintMultiplePages():void
        {
            init();
            printPages();
        }

        private function init():void
        {
            sheet1 = new Sprite();
            createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, height:130});
            sheet2 = new Sprite();
            createSheet(sheet2, "There was a great story to tell, and it ended quickly.\n\nThe end.", null);
        }

        private function createSheet(sheet:Sprite, str:String, imgValue:Object):void
        {
            sheet.graphics.beginFill(0xEEEEEE);
```

```
        sheet.graphics.lineStyle(1, 0x000000);
        sheet.graphics.drawRect(0, 0, 100, 200);
        sheet.graphics.endFill();

        var txt:TextField = new TextField();
        txt.height = 200;
        txt.width = 100;
        txt.wordWrap = true;
        txt.text = str;

        if (imgValue != null)
        {
            var img:Sprite = new Sprite();
            img.graphics.beginFill(0xFFFFFF);
            img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, imgValue.height);
            img.graphics.endFill();
            sheet.addChild(img);
        }
        sheet.addChild(txt);
    }

    private function printPages():void
    {
        var pj:PrintJob = new PrintJob();
        var pagesToPrint:uint = 0;
        if (pj.start())
        {
            if (pj.orientation == PrintJobOrientation.LANDSCAPE)
            {
                throw new Error("Page is not set to an orientation of portrait.");
            }

            sheet1.height = pj.pageHeight;
            sheet1.width = pj.pageWidth;
            sheet2.height = pj.pageHeight;
            sheet2.width = pj.pageWidth;

            try
            {
                pj.addPage(sheet1);
                pagesToPrint++;
            }
            catch (error:Error)
            {
            // Respond to error.
```

```
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (error:Error)
                {
                    // Respond to error.
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

打印示例：缩放、裁剪和响应

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在某些情况下，打印时您希望调整显示对象的大小 （或其他属性），以消除显示对象在屏幕上与打印纸张上的显示差异。在打印之前调整显示对象的属性时 （例如，通过使用 scaleX 和 scaleY 属性），请注意，如果对象缩放后大于为打印区域定义的矩形，则会裁剪此对象。页面打印完毕后，您可能还需要重置属性。

以下代码缩放 txt 显示对象 （但不缩放绿色框背景）的尺寸，最终按照指定矩形的尺寸对文本字段进行裁剪。打印后，文本字段将返回到在屏幕上显示的原始大小。如果用户从操作系统的 “ 打印 ” 对话框中取消打印作业， Flash 运行时中的内容将更改以警告用户此作业已取消。

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
```

```
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);

                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job canceled";
            }
            // Reset the txt scale properties.
            txt.scaleX = 1;
            txt.scaleY = 1;
        }

        private function init():void
        {
            bg = new Sprite();
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt = new TextField();
            txt.border = true;
            txt.text = "Hello World";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

打印示例：页面设置和打印选项

Adobe AIR 2 和更高版本

在以下示例中，对下列 PrintJob 设置进行了初始化：打印份数、页面大小 （法律专用纸）和页面方向 （横向）。此示例强制首先显示 “ 页面设置 ” 对话框，然后显示 “ 打印 ” 对话框以启动打印作业。

```
package
{
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.printing.PaperSize;
    import flash.printing.PrintUIOptions;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintAdvancedExample extends Sprite
    {
        private var bg:Sprite = new Sprite();
        private var txt:TextField = new TextField();
        private var pj:PrintJob = new PrintJob();
        private var uiOpt:PrintUIOptions = new PrintUIOptions();

        public function PrintAdvancedExample():void
        {
            initPrintJob();
            initContent();
            draw();
            printPage();
        }

        private function printPage():void
        {
            //test for dialog support as a static property of PrintJob class
            if (PrintJob.supportsPageSetupDialog)
            {
                pj.showPageSetupDialog();
            }
            if (pj.start2(uiOpt, true))
            {
                try
                {
                    pj.addPage(this, new Rectangle(0, 0, 100, 100));
                }
                catch (error:Error)
                {
                    // Do nothing.
                }
                pj.send();
            }
            else
            {
                txt.text = "Print job terminated";
                pj.terminate();
            }
        }

        private function initContent():void
        {
            bg.graphics.beginFill(0x00FF00);
            bg.graphics.drawRect(0, 0, 100, 200);
            bg.graphics.endFill();

            txt.border = true;
```

```
            txt.text = "Hello World";
        }

        private function initPrintJob():void
        {
            pj.selectPaperSize(PaperSize.LEGAL);
            pj.orientation = PrintJobOrientation.LANDSCAPE;
            pj.copies = 2;
            pj.jobName = "Flash test print";
        }

        private function draw():void
        {
            addChild(bg);
            addChild(txt);
            txt.x = 50;
            txt.y = 50;
        }
    }
}
```

第 55 章 : Geolocation

如果设备支持 geolocation，您可以使用 geolocation API 获得设备的当前地理位置。如果设备支持此功能，您可以获取 geolocation 信息。此信息包括位置在高度、精度、标题、速度和时间戳上的最新更改。

为响应设备的位置传感器， Geolocation 类调度 update 事件。 update 事件是一个 GeolocationEvent 对象。

更多帮助主题

flash.sensors.Geolocation

flash.events.GeolocationEvent

检测 geolocation 更改

要使用 geolocation 传感器，请实例化 Geolocation 对象并注册该对象调度的 update 事件。update 事件是一个 Geolocation 事件对象。此事件包含下列八个属性：

- altitude — 高度 （以米为单位）。
- heading — 相对于正北的移动方向 （以度为单位）。
- horizontalAccuracy — 水平精度 （以米为单位）。
- latitude — 纬度 （以度为单位）。
- longitude — 经度 （以度为单位）。
- speed — 速度 （以米 / 秒为单位）。
- timestamp — 自初始化运行时后事件的毫秒数。
- verticalAccuracy — 垂直精度 （以米为单位）。

timestamp 属性是一个 int 对象。其他属性是 Number 对象。

以下是在文本字段中显示 geolocation 数据的一个基本示例：

```
var geo:Geolocation;
if (Geolocation.isSupported)
{
    geo = new Geolocation();
    geo.addEventListener(GeolocationEvent.UPDATE, updateHandler);
}
else
{
    geoTextField.text = "Geolocation feature not supported";
}
function updateHandler(event:GeolocationEvent):void
{
    geoTextField.text = "latitude: " + event.latitude.toString() + "\n"
            + "longitude: " + event.longitude.toString() + "\n"
            + "altitude: " + event.altitude.toString()
            + "speed: " + event.speed.toString()
            + "heading: " + event.heading.toString()
            + "horizontal accuracy: " + event.horizontalAccuracy.toString()
            + "vertical accuracy: " + event.verticalAccuracy.toString()
}
```

要使用此示例，请确保创建 geoTextField 文本字段并在使用此代码之前将该文本字段添加到显示列表。

您可以通过调用 Geolocation 对象的 setRequestedUpdateInterval() 方法调整 geolocation 事件的所需时间间隔。此方法使用一个参数 interval，该参数是请求的更新时间间隔 （以毫秒为单位）：

```
var geo:Geolocation = new Geolocation();
geo.setRequestedUpdateInterval(10000);
```

Geolocation 更新之间的实际时间可能大于或小于此值。更新间隔的任何更改都会影响所有注册侦听器。如果您不调用 setRequestedUpdateInterval() 方法，应用程序将根据设备的默认时间间隔接收更新。

用户可以阻止应用程序访问 geolocation 数据。例如，应用程序尝试获取 geolocation 数据时， iPhone 会提示用户。为响应该提示，用户可以拒绝应用程序访问 geolocation 数据。当用户不能访问 geolocation 数据时， Geolocation 对象将调度 status 事件。另外， Geolocation 对象还包含 muted 属性，当 geolocation 传感器不可用时，该属性设置为 true。当 muted 属性发生更改时， Geolocation 对象将调度 status 事件。以下代码显示如何检测 geolocation 数据在哪些情况下不可用：

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.GeolocationEvent;
    import flash.events.MouseEvent;
    import flash.events.StatusEvent;
    import flash.sensors.Geolocation;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class GeolocationTest extends Sprite
    {

        private var geo:Geolocation;
        private var log:TextField;

        public function GeolocationTest()
        {
            super();
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;
            setUpTextField();

            if (Geolocation.isSupported)
            {
                geo = new Geolocation();
                if (!geo.muted)
                {
                    geo.addEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
                }
                geo.addEventListener(StatusEvent.STATUS, geoStatusHandler);
            }
            else
            {
                log.text = "Geolocation not supported";
            }
        }

        public function geoUpdateHandler(event:GeolocationEvent):void
        {
            log.text = "latitude : " + event.latitude.toString() + "\n";
            log.appendText("longitude : " + event.longitude.toString() + "\n");
        }
```

```
        public function geoStatusHandler(event:StatusEvent):void
        {
            if (geo.muted)
                geo.removeEventListener(GeolocationEvent.UPDATE, geoUpdateHandler);
            else
                geo.addEventListener(GeolocationEvent.UPDATE, geoStatusHandler);
        }

        private function setUpTextField():void
        {
            log = new TextField();
            var format:TextFormat = new TextFormat("_sans", 24);
            log.defaultTextFormat = format;
            log.border = true;
            log.wordWrap = true;
            log.multiline = true;
            log.x = 10;
            log.y = 10;
            log.height = stage.stageHeight - 20;
            log.width = stage.stageWidth - 20;
            log.addEventListener(MouseEvent.CLICK, clearLog);
            addChild(log);
        }
        private function clearLog(event:MouseEvent):void
        {
            log.text = "";
        }
    }
}
```

注：不包括 GPS 单位的第一代 iPhone 仅偶尔调度 update 事件。在这些设备上， Geolocation 对象最初调度一个或两个 update 事件。随后，当信息明显更改时将调度 update 事件。

检查 geolocation 支持

使用 Geolocation.isSupported 属性测试运行时环境是否能够使用此功能：

```
if (Geolocation.isSupported)
{
    // Set up geolocation event listeners and code.
}
```

目前，仅 iPhone 的基于 ActionScript 的应用程序和 Flash Lite 4 支持此 geolocation。如果 Geolocation.isSupported 在运行时为 true，则存在 geolocation 支持。

某些 iPhone 模型没有 GPS 单位。这些模型使用其他方法 （如移动电话三定位）获取 geolocation 数据。对于这些模型或任何已禁用 GPS 的 iPhone， Geolocation 对象可能只调度一个或两个初始 update 事件。

第 56 章：应用程序国际化

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

使用 flash.globalization 包，可以更轻松地创建适合不同语言和区域的约定的国际软件。

更多帮助主题

flash.globalization 包

第 819 页的 " 本地化应用程序 "

Charles Bihis：是否希望将您的 Flex/AIR 应用程序本地化？

应用程序国际化基础知识

术语 " 全球化 " 和 " 国际化 " 有时会互换使用。但这些术语被广泛接受的定义是，" 全球化 " 是指业务过程和工程过程的组合，而 " 国际化 " 仅指工程过程。

以下是一些重要术语定义：

全球化 在全球范围内准备以及启动产品和公司活动所需的大量工程和业务过程。全球化包含工程活动 （如国际化、本地化和文明化等）和业务活动 （如产品管理、财务规划、市场营销和法律事务等）。" 全球化 " 有时缩写为 G11n （代表字母 G，中间 11 个字母，然后是字母 n）。" 全球化是业务行为。 "

国际化 推广产品的工程过程，从而使产品无需重新设计或重新编译，就可以处理多种语言、脚本和文化约定 （包括货币、排序规则、数字和日期格式等）。此过程可分为两组活动：授权和本地化。国际化有时称为全球可用，有时缩写为 I18n。" 国际化是工程师行为。 "

本地化 使产品或服务适合特定语言、文化和本地外观要求的过程。本地化有时缩写为 L10n。" 本地化是翻译人员行为。 "

文明化 开发特定功能或使其满足独特文化需求的工程过程。例如， Adobe InDesign 中的日语发布功能以及 Adobe Acrobat 中的 Hanko 支持功能。

其他一些重要国际化术语的定义如下：

字符集 一种语言或一组语言使用的字符。字符集包括国家字符、特殊字符 （如标点符号和数学符号）、数字和计算机控制字符。

排序规则 针对给定区域设置，将文本排序为适当顺序。

区域设置 表示在地理、政治或文化区域 （在许多情况下，指示一个国家 / 地区）中使用的语言和文化约定的值。唯一区域设置标识符 （区域设置 ID）表示此值。区域设置 ID 用于查找一组提供区域设置特定支持的区域设置数据此支持适用于度量单位、解析数字和日期并为它们设置格式等等。

资源捆绑 一组已存储区域设置特定的元素，这些元素针对将在其中使用应用程序的区域设置而创建。通常，资源捆绑包含应用程序用户界面中的所有文本元素。在捆绑中，这些元素被翻译为给定区域设置的相应语言。它还包含其他设置，可根据特定区域设置改变用户界面的布局或行为。资源捆绑可包含区域设置特定的其他媒体类型或对其他媒体类型的引用。

flash.globalization 包概述

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

flash.globalization 包可利用基础操作系统的文化支持功能。借助此包，可更轻松地编写符合各个用户的文化约定的应用程序。

此包中的主类包括：

- Collator 类，用于控制字符串的排序和匹配
- CurrencyFormatter 类，用于将数字设置为货币金额字符串格式，并解析输入字符串中的货币金额和货币符号
- 用于设置日期值的格式的 DateTimeFormatter 类
- 用于检索有关特定区域设置信息的 LocaleID 类
- NumberFormatter 类，用于设置数值的格式并对数值进行分析
- StringTools 类，用于处理受区域设置影响的字符串大小写转换

flash.globalization 包和资源本地化

flash.globalization 包不处理资源本地化。但是，您可以将 flash.globalization 区域设置 ID 用作键值，以便使用其他技术检索本地化的资源。例如，您可以使用 ResourceManager 和 ResourceBundle 类本地化使用 Flex 构建的应用程序资源。有关详细信息，请参阅本地化 Flex 应用程序。

Adobe AIR 1.1 还包含某些帮助本地化 AIR 应用程序的功能，在第 820 页的 " 本地化 AIR 应用程序 " 中对此进行了介绍。

应用程序国际化的一般方法

下列步骤介绍了使用 flash.globalization 包国际化应用程序的高级别常见方法：

1. 确定或设置区域设置。
2. 创建服务类 （Collator、 CurrencyFormatter、 DateTimeFormatter、 NumberFormatter 或 StringTools）的实例。
3. 使用 lastOperationStatus 属性检查错误和回退。
4. 使用区域设置特定的设置格式化并显示信息。

下一步是加载和显示特定于区域设置的字符串和用户界面资源。此步骤可能包括下列任务，例如：

- 使用自动布局功能调整 UI 的大小以适应字符串长度
- 选择正确的字体并支持字体回退
- 使用 FTE 文本引擎支持其他编写系统
- 确保正确处理输入法编辑器

检查错误和回退

flash.globalization 服务类全部按照相似的模式标识错误。它们还共享一种模式，即在请求的区域设置不可用时回退到用户操作系统支持的区域设置。

以下示例显示了实例化服务类时如何检查错误和回退。每个服务类都包含一个 lastOperationStatus 属性，指示最近的方法调用触发的是错误还是警告。

```
var nf:NumberFormatter = new NumberFormatter("de-DE");
if(nf.lastOperationStatus != LastOperationStatus.NO_ERROR)
{
    if(nf.lastOperationStatus == LastOperationStatus.USING_FALLBACK_WARNING)
    {
        // perform fallback logic here, if needed
        trace("Warning - Fallback locale ID: " + nf.actualLocaleIDName);
    }
    else
    {
        // perform error handling logic here, if needed
        trace("Error: " + nf.lastOperationStatus);
    }
}
```

如果使用回退区域设置 ID 或存在错误，此示例将只跟踪消息。如果需要，您的应用程序可以执行其他错误处理逻辑。例如，您可以向用户显示消息或强制应用程序使用特定的受支持区域设置。

确定区域设置

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

区域设置标识某个国家 / 地区或区域的语言和文化约定的特定组合。

可以将区域设置标识符作为字符串安全地管理。但您可以使用 LocaleID 类获取与区域设置有关的其他信息。

您可以按如下方式创建 LocaleID 对象：

```
var locale:LocaleID = new LocaleID("es-MX");
```

创建 LocaleID 对象后，您可以检索有关区域设置 ID 的数据。使用 getKeysAndValues()、 getLanguage()、 getRegion()、 getScript()、 getVariant() 和 isRightToLeft() 方法以及 name 属性。

从这些方法和属性检索的值可以反映不能直接从区域设置标识符提取的有关区域设置的其他信息。

应用程序创建区域设置感知服务 （例如日期格式化程序）时，必须指定要使用的区域设置。支持的区域设置列表因操作系统而异，因此请求的区域设置可能不可用。

Flash Player 首先尝试匹配请求的区域设置的语言代码。然后，尝试通过查找匹配的编写系统 （脚本）和区域进一步确定区域设置。例如：

```
var loc:LocaleID = new LocaleID("es");
trace(loc.getLanguage()); // es
trace(loc.getScript()); // Latn
trace(loc.getRegion()); // ES
```

在此示例中， LocaleID() 构造函数检索有关与该用户的语言代码 “es” 最匹配的区域设置的数据。

设置区域设置 ID

有多个方法可以设置应用程序的当前区域设置，包括：

- 将单个区域设置 ID 硬编码到应用程序中。虽然这种方法很常见，但它不支持应用程序的国际化。
- 使用用户的操作系统、浏览器或其他用户首选项中的区域设置 ID 首选参数。此技术通常为用户提供最佳的区域设置设置，但并非始终是准确的。操作系统设置有可能不反映用户的实际首选参数。例如，用户可能会使用共享计算机并且无法更改操作系统的首选区域设置。

- 在根据用户的首选参数设置区域设置 ID 后，允许用户从支持的区域设置列表中进行选择。如果您的应用程序可以支持多个区域设置，此策略通常是最佳选择。

您可以按如下方式实现第三个选项：

1 从用户配置文件、浏览器设置、操作系统设置或 Cookie 中检索用户的首选区域设置或语言的列表。（您的应用程序需要自行实现此逻辑。 flash.globalization 库不支持直接读取此类首选参数。）

2 确定您的应用程序支持其中哪些区域设置并默认选择最佳区域设置。使用 LocaleID.determinePreferredLocales() 根据用户首选区域设置和操作系统支持的区域设置为用户查找最佳的区域设置。

3 如果默认区域设置不合适，为用户提供更改默认区域设置设置的方法。

其他区域设置和语言类的限制

通过 fl.lang.Locale 类，您可以使用包含字符 值的资源捆绑，基于区域设置替换文本字符串。但是，此类不支持数字、货币或日期格式化、排序和匹配等其他国际化功能。此外，此类仅用于 Flash Professional。

您还可以使用 flash.system.Capabilities.language 属性检索操作系统的当前语言代码设置。但是，此属性仅检索双字符 ISO 639-1 语言代码，而不是完整的区域设置 ID，并且只支持特定的区域设置组。

通过 AIR 1.5，您可以使用 flash.system.Capabilities.languages 属性。此属性提供一个用户的首选用户界面语言的数组。因此，它不受 Capabilities.language 的限制。

设置数字格式

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

根据区域的不同，数值的显示格式也大不相同。例如，以下是针对特定区域设置对数字 123456.78 进行格式设置的方式：

区域设置	数字格式
en-US （英语，美国）	-123,456.78
de-DE （德语，德国）	-123.456,78
fr-FR （法语，法国）	-123 456,78
de-CH （德语，瑞士）	-123'456.78
en-IN （英语，印度）	-1,23,456.78
许多阿拉伯区域设置	123,456.78-

影响数字格式的因素有许多，包括：

- 分隔符。小数点分隔符置于数字的整数和小数部分之间。它可以是句号、逗号或其他字符。分组分隔符或千位分隔符可以是句号、逗号、不间断空格或其他字符。
- 分组模式。小数点左侧的每个分组分隔符之间的数位可以是两位或三位，或者其他值。
- 负数指示符。负数可以显示为数字左侧或右侧带有一个减号，或者用括号表示 （对于财务应用程序）。例如，负 19 可以显示为 -19、 19- 或 (19)。
- 前导零和尾部零。某些文化约定对显示的数字添加前导零或尾部零。例如，值 0.17 可能显示为 .17、 0.17 或 0.170 等。
- 数字字符集。许多语言，包括印地语、阿拉伯语和日语，使用不同的数字字符集。 flash.globalization 包支持任何映射到数字 0-9 的数字字符集。

格式化数值时， NumberFormatter 类将考虑所有这些因素。

使用 NumberFormatter 类

NumberFormatter 类将根据特定区域设置的约定格式化数值 （类型为 int、 uint 或 Number）。

以下示例显示了使用用户的操作系统提供的默认格式设置属性格式化数字的最简单方式：

```
var nf:NumberFormatter = new NumberFormatter(LocaleID.DEFAULT);
trace(nf.formatNumber(-123456.789))
```

结果根据用户的区域设置设置和用户首选项而变化。例如，如果用户的区域设置是 fr-FR，则格式化的值将是：

-123.456,789

如果您只希望格式化特定区域设置的数字，而不管用户的设置如何，则专门设置区域设置名称。例如：

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
trace(nf.formatNumber(-123456.789));
```

在这种情况下，结果为：

-123'456.789

formatNumber() 方法将采用 Number 作为参数。 NumberFormatter 类还具有采用 int 作为输入的 formatInt() 方法，以及采用 uint 的 formatUint() 方法。

您可以通过设置 NumberFormatter 类的属性明确控制格式化逻辑，如此示例中所示：

```
var nf:NumberFormatter = new NumberFormatter("de-CH");
nf.negativeNumberFormat = 0;
nf.fractionalDigits = 5;
nf.trailingZeros = true;
nf.decimalSeparator = ",";
nf.useGrouping = false;
trace(nf.formatNumber(-123456.789)); //(123456.78900)
```

此示例首先创建一个 NumberFormatter 对象，然后：

- 将负数格式设置为使用括号 （与财务应用程序相同）；
- 将小数点分隔符之后的数位设置为 5 ；
- 指定使用尾部零以确保 5 位小数；
- 将小数点分隔符设置为逗号；
- 要求格式化程序不使用任何分组分隔符。

注：当更改其中一些属性时，生成的数字格式将不再对应于指定区域设置的首选格式。仅在下列情况下使用其中一些属性：当区域设置感知不重要时；当您需要详细控制格式的一个方面 （例如尾部零的数量）时；或者当用户直接请求更改 （例如，通过 Windows 控制面板进行更改）时。

分析包含数值的字符串

NumberFormatter 类还可以从符合区域设置特定格式化要求的字符串提取数值。 NumberFormatter.parseNumber() 方法从字符串提取单个数值。例如：

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "-1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // -1234567.89
trace("Status:" + nf.lastOperationStatus); // noError
```

parseNumber() 方法处理仅包含位数和数字格式化字符 （例如负号和分隔符）的字符串。如果字符串包含其他字符，则设置错误代码：

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String =  "The value is 1,234,567.890"
var parsedNumber:Number = nf.parseNumber(inputNumberString);
trace("Value:" + parsedNumber); // NaN
trace("Status:" + nf.lastOperationStatus); // parseError
```

要从包含其他字母字符的字符串中提取数字，请使用 NumberFormatter.parse() 方法：

```
var nf:NumberFormatter = new NumberFormatter( "en-US" );
var inputNumberString:String = "The value is 123,456,7.890";
var parseResult:NumberParseResult = nf.parse(inputNumberString);
trace("Value:" + parseResult.value); // 1234567.89
trace("startIndex: " + parseResult.startIndex); // 14
trace("Status:" + nf.lastOperationStatus); // noError
```

parse() 方法返回在其值属性中包含分析的数值的 NumberParseResult 对象。 startIndex 属性指示找到的第一个数字字符的索引。您可以使用 startIndex 和 endIndex 属性提取字符串中数位之前和之后的部分。

设置货币值格式

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

货币值的显示格式与数字格式一样多种多样。例如，格式化某些区域设置的美元值 $123456.78 的方式如下：

区域设置	数字格式
en-US （英语，美国）	$123,456.78
de-DE （德语，德国）	123.456,78 $
en-IN （英语，印度）	$ 1,23,456.78

货币格式化包含数字格式化的所有因素，同时还有以下其他因素：

- 货币 ISO 代码。实际区域设置使用的三个字母 ISO 4217 货币代码，例如 USD 或 EUR。
- 货币符号。实际区域设置使用的货币符号或字符串，例如 $ 或 €。
- 负货币格式。定义与货币金额的数字部分相关的货币符号和负号或括号的位置。
- 正货币格式。定义货币符号相对于货币金额的数字部分的位置。

使用 CurrencyFormatter 类

CurrencyFormatter 类根据特定区域设置的约定将数值格式化为包含货币字符串和格式化的数字的字符串。

实例化新的 CurrencyFormatter 对象时，该对象将其货币设置为给定区域设置的默认货币。

以下示例显示了使用德语区域设置创建的 CurrencyFormatter 对象，假定货币金额以欧元表示：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "de-DE" );
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

在大多数情况下，不要依赖区域设置的默认货币。如果不支持用户的默认区域设置，则 CurrencyFormatter 类将分配回退区域设置。回退区域设置可能包含不同的默认货币。此外，即使金额格式不符合用户的本地货币，您通常也希望货币格式在用户看来是正确的。例如，加拿大用户可能希望看到以欧元表示的德国公司的报价，但格式采用加拿大样式。

CurrencyFormatter.setCurrency() 方法指定要使用的确切货币字符串和货币符号。

以下示例将向加拿大法语地区的用户显示以欧元为单位的货币金额：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "fr-CA" );
cf.setCurrency("EUR", "€");
trace(cf.format(1234567.89)); // 1.234.567,89 EUR
```

setCurrency() 方法还可用于通过设置明确的货币符号减少混淆。例如：

```
cf.setCurrency("USD","US$");
```

默认情况下， format() 方法将显示三个字符的 ISO 4217 货币代码，而不是货币符号。 ISO 4217 代码是明确的，不需要本地化。但是，许多用户更喜欢查看货币符号而不是 ISO 代码。

CurrencyFormatter 类可以帮助您决定格式化的货币字符串使用的是一种货币符号 （如美元符号或欧元符号），还是三个字符的 ISO 货币字符串 （例如 USD 或 EUR）。例如，对于加拿大用户，以加拿大元表示的金额可能显示为 $200。而对于美国用户，可能显示为 CAD 200。使用 formattingWithCurrencySymbolIsSafe() 方法根据用户的区域设置设置确定金额的货币符号不明确还是不正确。

以下示例将以欧元表示的值格式化为 en-US 区域设置的格式。根据用户的区域设置，输出字符串使用 ISO 货币代码或货币符号。

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-CA");

if (cf.formattingWithCurrencySymbolIsSafe("USD"))
{
    trace(cf.format(1234567.89, true)); // $1,234,567.89
}
else
{
    cf.setCurrency("USD", "$");
    trace(cf.format(1234567.89)); // USD1,234,567.89
}
```

分析包含货币值的字符串

CurrencyFormatter 类还可以从符合区域设置特定格式化要求的输入字符串提取货币金额和货币字符串。 CurrencyFormatter.parse() 方法在 CurrencyParseResult 对象中存储分析的金额和货币字符串，如下所示：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "(GBP 123,56,7.890)";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("parsed amount: " + parseResult.value); // -1234567.89
trace("currencyString: " + parseResult.currencyString ); // GBP
```

输入字符串的货币字符串部分可以包含货币符号、货币 ISO 代码及其他文本字符。货币字符串、负数指示符和数值的位置与 negativeCurrencyFormat 和 positiveCurrencyFormat 属性指定的格式相匹配。例如：

```
var cf:CurrencyFormatter = new CurrencyFormatter( "en-US" );
var inputCurrencyString:String = "Total $-123,56,7.890";
var parseResult:CurrencyParseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // parseError
trace("parsed amount: " + parseResult.value); // NaN
trace("currencyString: " + parseResult.currencyString ); //
cf.negativeCurrencyFormat = 2;
parseResult = cf.parse(inputCurrencyString);
trace("status: " + cf.lastOperationStatus ); // noError
trace("parsed amount: " + parseResult.value); // -123567.89
trace("currencyString: " + parseResult.currencyString ); // Total $
```

在此示例中，输入字符串包含后面跟有负号和数字的货币字符串。但是， en-US 区域设置的默认 negativeCurrencyFormat 值指定负数指示符放在最前面。因此， parse() 方法会生成一个错误，并且分析的值是 NaN。

将 negativeCurrencyFormat 设置为 2 后，会指定货币字符串放在最前面， parse() 方法就会成功执行。

设置日期和时间格式

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

根据区域的不同，日期和时间值的显示格式也大不相同。例如，以下是如何以短格式显示某些区域设置的 1962 年 1 月 2 日的 1:01 PM：

区域设置	日期和时间格式
en-US （英语，美国）	1/2/62 1:01pm
fr-FR （法语，法国）	2/1/62 13:01
ja-JP （日语，日本）	1962/2/1 13:01

使用 DateTimeFormatter 类

DateTimeFormatter 类根据特定区域设置的约定将日期值格式化为日期和时间字符串。

格式化遵循模式字符串，该模式字符串包含要用日期或时间值替换的字母序列。例如，在 “yyyy/MM” 模式中，字符 “yyyy” 被替换为 4 位数字的年份，后面跟有 “/” 字符和 2 位数字的月份。

可以使用 setDateTimePattern() 方法明确设置模式字符串。但是，最好根据用户的区域设置和操作系统首选项自动设置模式。此做法有助于确保结果符合当地文化。

DateTimeFormatter 可以以三种标准样式 （LONG、 MEDIUM 和 SHORT）表示日期和时间，也可以使用 CUSTOM 模式。其中一种样式用于表示日期，另一种样式用于表示时间。用于每种样式的实际模式在一定程度上取决于操作系统。

您可以在创建 DateTimeFormatter 对象时指定样式。如果没有指定样式参数，则默认情况下，将这些样式参数设置为 DateTimeStyle.LONG。稍后您可以通过使用 setDateTimeStyles() 方法更改样式，如以下示例中所示：

```
var date:Date = new Date(2009, 2, 27, 13, 1);
var dtf:DateTimeFormatter = new DateTimeFormatter("en-US",
    DateTimeStyle.LONG, DateTimeStyle.LONG);

var longDate:String = dtf.format(date);
trace(longDate); // March 27, 2009 1:01:00 PM

dtf.setDateTimeStyles(DateTimeStyle.SHORT, DateTimeStyle.SHORT);
var shortDate:String = dtf.format(date);
trace(shortDate); // 3/27/09 1:01 PM
```

本地化月份名称和日期名称

许多应用程序在日历显示或下拉列表中使用月份名称和周日期名称的列表。

您可以使用 DateTimeFormatter.getMonthNames() 方法检索本地化的月份名称列表。根据具体操作系统，可能可以使用完整形式和缩写形式。传递 DateTimeNameStyle.FULL 值获得完整长度的月份名称。传递 DateTimeNameStyle.LONG_ABBREVIATION 或 DateTimeNameStyle.SHORT_ABBREVIATION 值以获得较短的版本。

在某些语言中，月份名称在放在日期格式中的日期值旁边时会发生更改 （更改为其所有格形式）。如果您计划单独使用月份名称，请将 DateTimeNameContext.STANDALONE 值传递到 getMonthNames() 方法。但是，如果您在格式化的日期中使用月份名称，则应传递 DateTimeNameContext.FORMAT 值。

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var months:Vector.<String> = dtf.getMonthNames(DateTimeNameStyle.FULL,
        DateTimeNameContext.STANDALONE);
trace(months[0]); // janvier
months = dtf.getMonthNames(DateTimeNameStyle.SHORT_ABBREVIATION,
         DateTimeNameContext.STANDALONE);
trace(months[0]); // janv.
```

DateTimeFormatter.getWeekdayNames() 方法提供了本地化的周日期名称的列表。 getWeekdayNames() 方法接受的 nameStyle 和上下文参数与 getMonthNames() 方法所接受的相同。

```
var dtf:DateTimeFormatter = new DateTimeFormatter("fr-FR");
var weekdays:Vector.<String> = dtf.getWeekdayNames(DateTimeNameStyle.FULL,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dimanche
weekdays = dtf.getWeekdayNames(DateTimeNameStyle.LONG_ABBREVIATION,
           DateTimeNameContext.STANDALONE);
trace(weekdays[0]); // dim.
```

此外， getFirstWeekday() 方法返回某个日期的索引值，这一天传统上标记为所选区域设置中一周开始的那天。

排序和比较字符串

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

排序是以适当的顺序排列各项的过程。排序规则因区域设置而大不相同。这些规则还根据您是否对列表排序或者是否匹配相似项 （例如在文本搜索算法中）而有所不同。

排序时，大小写字母或音调符号 （例如重音符号）等微小的区别通常很重要。例如，字母 ö （带有分音符的 o）在法语或英语中视为与普通字母 o 几乎等效。但同一字母在瑞典语中跟在字母 z 后面。另外，在法语和一些其他语言中，单词中最后一个重音差异决定该单词在排序列表中的顺序。

搜索时，您经常需要忽略大小写或音调区别，以便增加找到相关匹配项的机会。例如，在法语文档中搜索字符 “cote” 很可能会返回 “cote”、 “côte” 和 “coté” 的匹配项。

使用 Collator 类

Collator 类的主要方法是 compare() 方法 （主要用于排序）和 equals() 方法 （用于匹配值）。

以下示例显示了 compare() 和 equals() 方法的不同行为。

```
var words:Array = new  Array("coté", "côte");

var sorter:Collator = new Collator("fr-FR", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // côte,coté

var matcher:Collator = new Collator("fr-FR", CollatorMode.MATCHING);
if (matcher.equals(words[0], words[1]))
{
    trace(words[0] + " = " + words[1]); // côte = coté
}
```

此示例首先针对法语 - 法国区域设置在 SORTING 模式中创建 Collator 对象。然后对只有变音符号不同的两个单词进行排序。这说明 SORTING 比较会区分重音和非重音字符。

通过将对 Collator 对象的 sort() 方法的引用作为参数传递到 Array.sort() 方法来执行排序。此技巧是使用 Collator 对象控制排序顺序最有效的方法之一。

然后，该示例在 MATCHING 模式中创建 Collator 对象。该 Collator 对象比较这两个单词时，认为二者等同。这说明 MATCHING 比较将重音和非重音字符视为相同的。

自定义 Collator 类的行为

默认情况下， Collator 类根据区域设置和用户的系统首选参数使用从操作系统获取的字符串比较规则。您可以通过明确设置多种属性自定义 compare() 和 equals() 方法的行为。下表列出了各个属性和比较后的效果：

Collator 属性	效果
numericComparison	控制数字字符被视为数字还是文本。
ignoreCase	控制是否忽略大小写区别。
ignoreCharacterWidth	控制某些中文和日语字符的全角和半角格式是否视为等同。
ignoreDiacritics	控制使用相同基本字符但使用不同的重音或其他音调符号的字符串是否视为等同。
ignoreKanaType	控制使用的仅 kana 字符类型不同的字符串是否视为等同。
ignoreSymbols	控制是否忽略符号字符 （例如空格、货币符号、数学符号及其他字符）。

以下代码显示将 ignoreDiacritics 属性设置为 true 将更改法语单词列表的排序顺序：

```
var words:Array = new  Array("COTE", "coté", "côte", "Coté","cote");
var sorter:Collator = new Collator("fr-CA", CollatorMode.SORTING);
words.sort(sorter.compare);
trace(words); // cote,COTE,côte,coté,Coté

sorter.ignoreDiacritics = true;
words.sort(sorter.compare);
trace(words); // côte,coté,cote,Coté,COTE
```

大小写转换

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

在大写格式 (majiscules) 和小写格式 (miniscules) 之间转换字母的语言规则也有所不同。

例如，在使用拉丁字母表的大多数语言中，大写字母 “I” 的小写格式是 “i”。但是，在某些语言 （例如土耳其语和阿塞拜疆语）中，还存在一个无点字母 “ı”。因此，在这些语言中，小写无点字母 “ı” 转换为大写字母 “I”。小写字母 “i” 转换为带点的大写字母 “İ”。

StringTools 类提供了使用特定语言的规则执行此类转换的方法。

使用 StringTools 类

StringTools 类提供了两个方法来执行大小写转换：toLowerCase() 和 toUpperCase()。您可以通过调用带有区域设置 ID 的构造函数创建 StringTools 对象。 StringTools 类从操作系统检索区域设置 （或回退区域设置）的大小写转换规则。不能进一步自定义大小写转换算法。

以下示例使用 toUpperCase() 和 toLowerCase() 方法转换包括字母 “ß”(sharp S) 的德语短语。

```
var phrase:String = "Schloß Neuschwanstein";
var converter:StringTools = new StringTools("de-DE");

var upperPhrase:String = converter.toUpperCase(phrase);
trace(upperPhrase); // SCHLOSS NEUSCHWANSTEIN

var lowerPhrase:String = converter.toLowerCase(upperPhrase);
trace(lowerPhrase);// schloss neuschwanstein
```

toUpperCase() 方法将小写字母 “ß” 转换为大写字母 “SS”。此转换只能单向进行。将字母 “SS” 转换回小写时，结果为 “ss” 而非 “ß”。

示例：国际化股票报价应用程序

Flash Player 10.1 和更高版本， **Adobe AIR 2.0** 和更高版本

全球股票报价应用程序在以下三个不同的股票市场上检索并显示虚构的股票相关数据：美国、日本和欧洲。它根据多种区域设置的约定设置数据格式。

此示例说明了 flash.globalization 包的下列功能：

- 区域设置感知数字格式化
- 区域设置感知货币格式化
- 设置货币 ISO 代码和货币符号
- 区域设置感知日期格式化
- 检索并显示相应的月份名称缩写

若要获取此范例的应用程序文件，请参阅 www.adobe.com/go/learn_programmingAS3samples_flash_cn。可以在 Samples/GlobalStockTicker 文件夹中找到全球股票报价应用程序文件。该应用程序包含以下文件：

文件	说明
GlobalStockTicker.mxml 或 GlobalStockTicker.fla	适用于 Flex (MXML) 或 Flash (FLA) 的应用程序的用户界面。
styles.css	应用程序用户界面的样式 （仅限 Flex）。
com/example/programmingas3/stockticker/flex/FinGraph.mxml	显示模拟股票数据图表的 MXML 组件，仅限 Flex。
com/example/programmingas3/stockticker/flash/GlobalStockTicker.as	包含应用程序的用户界面逻辑的 Document 类 （仅限 Flash）。
comp/example/programmingas3/stockticker/flash/RightAlignedColumn.as	Flash DataGrid 组件的自定义单元格渲染器 （仅限 Flash）。

文件	说明
com/example/programmingas3/stockticker/FinancialGraph.as	用于绘制模拟股票数据图表的 ActionScript 类。
com/example/programmingas3/stockticker/Localizer.as	ActionScript 类，用于管理区域设置和货币并处理本地化的数字、货币金额和日期的格式化。
com/example/programmingas3/stockticker/StockDataModel.as	ActionScript 类，用于包含全球股票报价示例的所有采样数据。

了解用户界面和采样数据

应用程序的主要用户界面元素是：

- 用于选择区域设置的组合框
- 用于选择市场的组合框
- 用于显示每个市场中六家公司的数据的 DataGrid
- 用于显示所选公司股票的模拟历史数据的图表

应用程序将存储 StockDataModel 类中有关区域设置、市场和公司股票的所有采样数据。实际应用程序将从服务器检索数据，然后将数据存储在类 （如 StockDataModel 类）中。在此示例中，所有数据都硬编码到 StockDataModel 类中。

注：此财务表中显示的数据不一定匹配 DataGrid 控件中显示的数据。每次选择其他公司时，都将随机重绘图表。仅用作说明目的。

设置区域设置

在一些初始设置工作后，应用程序将调用 Localizer.setLocale() 方法以创建默认区域设置的格式化程序对象。用户每次从区域设置组合框选择新值时，也会调用 setLocale() 方法。

```
public function setLocale(newLocale:String):void
{
    locale  = new LocaleID(newLocale);

    nf = new NumberFormatter(locale.name);
    traceError(nf.lastOperationStatus, "NumberFormatter", nf.actualLocaleIDName);

    cf = new CurrencyFormatter(locale.name);
    traceError(cf.lastOperationStatus, "CurrencyFormatter", cf.actualLocaleIDName);
    symbolIsSafe = cf.formattingWithCurrencySymbolIsSafe(currentCurrency);
    cf.setCurrency(currentCurrency, currentSymbol);
    cf.fractionalDigits = currentFraction;

    df = new DateTimeFormatter(locale.name, DateTimeStyle.LONG, DateTimeStyle.SHORT);
    traceError(df.lastOperationStatus, "DateTimeFormatter", df.actualLocaleIDName);
    monthNames = df.getMonthNames(DateTimeNameStyle.LONG_ABBREVIATION);
}

public function traceError(status:String, serviceName:String, localeID:String) :void
{
    if(status != LastOperationStatus.NO_ERROR)
    {
        if(status == LastOperationStatus.USING_FALLBACK_WARNING)
```

```
        {
            trace("Warning - Fallback locale ID used by "
                    + serviceName + ": " + localeID);
        }
        else if (status == LastOperationStatus.UNSUPPORTED_ERROR)
        {
            trace("Error in " + serviceName + ": " + status);
            //abort application
            throw(new Error("Fatal error", 0));
        }
        else
        {
            trace("Error in " + serviceName + ": " + status);
        }
    }
    else
    {
        trace(serviceName + " created for locale ID: " + localeID);
    }
}
```

首先， setLocale() 方法创建 LocaleID 对象。借助此对象，以后更容易获得有关实际区域设置的详细信息 （如果需要）。

接下来，为区域设置创建新的 NumberFormatter、 CurrencyFormatter 和 DateTimeFormatter 对象。在创建每个格式化程序对象后，将调用 traceError() 方法。如果请求的区域设置有问题，该方法会在控制台中显示错误和警告消息。（实际的应用程序应该根据此类错误做出响应，而不只是跟踪它们。）

在创建 CurrencyFormatter 对象后， setLocale() 方法会将格式化程序的货币 ISO 代码、货币符号和 fractionalDigits 属性设置为先前确定的值。（用户每次从市场组合框中选择新市场时，都会设置这些值）。

创建 DateTimeFormatter 对象后， setLocale() 方法还会检索本地化的月份名称缩写的数组。

格式化数据

格式化的股票数据显示在 DataGrid 控件中。每个 DataGrid 列将调用使用适当的格式化程序对象设置列值格式的标签函数。

例如，在 Flash 版本中，以下代码设置 DataGrid 列：

```
var col1:DataGridColumn = new DataGridColumn("ticker");
col1.headerText = "Company";
col1.sortOptions = Array.NUMERIC;
col1.width = 200;

var col2:DataGridColumn = new DataGridColumn("volume");
col2.headerText = "Volume";
col2.width = 120;
col2.cellRenderer = RightAlignedCell;
col2.labelFunction = displayVolume;

var col3:DataGridColumn = new DataGridColumn("price");
col3.headerText = "Price";
col3.width = 70;
col3.cellRenderer = RightAlignedCell;
col3.labelFunction = displayPrice;

var col4:DataGridColumn = new DataGridColumn("change");
col4.headerText = "Change";
col4.width = 120;
col4.cellRenderer = RightAlignedCell;
col4.labelFunction = displayPercent;
```

此示例的 Flex 版本在 MXML 中声明其 DataGrid。它还为每列定义类似的标签函数。

labelFunction 属性引用下列函数，这些函数将调用 Localizer 类的格式化方法：

```
private function displayVolume(item:Object):String
{
    return localizer.formatNumber(item.volume, 0);
}

private function displayPercent(item:Object):String
{
    return localizer.formatPercent(item.change )  ;
}

  private function displayPrice(item:Object):String
{
    return localizer.formatCurrency(item.price);
}
```

然后， Localizer 方法将设置并调用适当的格式化程序：

```
public function formatNumber(value:Number, fractionalDigits:int = 2):String
{
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value);
}

public function formatPercent(value:Number, fractionalDigits:int = 2):String
{
    // HACK WARNING: The position of the percent sign, and whether a space belongs
    // between it and the number, are locale-sensitive decisions. For example,
    // in Turkish the positive format is %12 and the negative format is -%12.
    // Like most operating systems, flash.globalization classes do not currently
    // provide an API for percentage formatting.
    nf.fractionalDigits = fractionalDigits;
    return nf.formatNumber(value) + "%";
}

public function formatCurrency(value:Number):String
{
    return cf.format(value, symbolIsSafe);
}

public function formatDate(dateValue:Date):String
{
    return df.format(dateValue);
}
|
```

第 57 章：本地化应用程序

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

本地化即包含资源以支持多种区域设置的过程。区域设置为语言和国家 / 地区代码的组合。例如，en_US 指美国英语，fr_FR 指法国法语。若要针对这些区域设置本地化应用程序，则应提供两组资源：一组用于 en_US 区域设置，一组用于 fr_FR 区域设置。

区域设置可以共享语言。例如， en_US 和 en_GB （英国）是不同的区域设置。在此情况下，虽然这两个区域设置都使用英语，但国家 / 地区代码指示它们为不同的区域设置，并可能因此使用不同的资源。例如， en_US 区域设置下的应用程序可能将单词拼写为 “color”，而在 en_GB 区域设置下则可能拼写为 “colour”。同样，根据区域设置的不同，货币单位也将以美元或英镑表示，并且日期和时间的格式可能也有所不同。

您也可以在不指定国家 / 地区代码的情况下为一种语言提供一组资源。例如，您可以为英语提供 en 资源并为 en_US 区域设置提供特定于美式英语的其他资源。

本地化不仅仅是翻译应用程序中使用的字符串。它还包括对任何类型的资源 （例如音频文件、图像和视频）的翻译。

选择区域设置

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要确定内容或应用程序使用的区域设置，可使用下列方法之一：

- flash.globalization 包 — 使用 flash.globalization 包中的区域设置感知类可以根据操作系统和用户首选项为用户检索默认区域设置。这是将在 Flash Player 10.1 或更高版本或者 AIR 2.0 或更高版本运行时上运行的应用程序的首选方法。有关详细信息，请参阅第 807 页的 “ 确定区域设置 ”。
- 用户提示符 — 您可以在某些默认区域设置下启动应用程序，然后请用户选择其首选区域设置。
- Capabilities.languages （仅限 AIR） — Capabilities.languages 属性列出用户首选语言中提供的语言数组，与通过操作系统设置的一样。字符串包含 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt) 定义的语言标记 （如果适用，还包括脚本及区域信息）。这些字符串使用连字符作为分隔符 （例如，"en-US" 或 "ja-JP"）。返回的数组中的第一项具有与 language 属性相同的主语言 ID。例如，如果 languages[0] 设置为 "en-US"，则 language 属性将设置为 "en"。但是，如果语言属性设置为 "xu" （指定未知语言），则 languages 数组中的第一个元素不同。
- Capabilities.language — Capabilities.language 提供了操作系统的用户界面语言代码。但是，此属性限制为 20 种已知语言。并且在英文系统中，此属性仅返回语言代码，而不返回国家 / 地区代码。出于以上原因，最好使用 Capabilities.languages 数组中的第一个元素。

本地化 Flex 内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe Flex 包含一个用于本地化 Flex 内容的框架。此框架包括 Locale、 ResourceBundle 和 ResourceManagerImpl 类以及 IResourceBundle、 IResourceManagerImpl 接口。

Google 代码 (http://code.google.com/p/as3localelib/) 中提供了一个 Flex 本地化库，其中包含用于排序应用程序区域设置的实用程序类。

更多帮助主题
http://code.google.com/p/as3localelib/

本地化 Flash 内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Adobe Flash Professional 在 ActionScript 3.0 组件中包含一个 Locale 类。使用 Locale 类可以控制 SWF 文件显示多语言文本的方式。借助 “Flash 字符串 ” 面板，可以在动态文本字段中使用字符串 ID 替代字符串。使用此工具，您可以创建一个 SWF 文件，用它来显示从特定语言的 XML 文件加载的文本。有关使用 Locale 类的信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考。

本地化 AIR 应用程序

Adobe AIR 1.0 和更高版本

AIR SDK 提供了一个 HTML 本地化框架（包含在 AIRLocalizer.js 文件中）。该框架包括 API，可帮助处理基于 HTML 的应用程序中的多个区域设置。以下网址提供了一个用于排序区域设置的 ActionScript 库：
http://code.google.com/p/as3localelib/。

更多帮助主题
http://code.google.com/p/as3localelib/

对日期、时间和货币进行本地化

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

应用程序为每个区域设置显示日期、时间和货币的方式均有很大不同。例如，美国标准表示日期的方式为月 / 日 / 年，而欧洲标准表示日期的方式为日 / 月 / 年。

您可以编写代码以设置日期、时间和货币的格式。例如，以下代码将 Date 对象转换为月 / 日 / 年格式或日 / 月 / 年格式。如果将 locale 变量 （表示区域设置）设置为 "en_US"，则函数会返回月 / 日 / 年格式。该示例将 Date 对象转换为所有其他区域设置的日 / 月 / 年格式：

```
function convertDate(date)
{
    if (locale == "en_US")
    {
        return (date.getMonth() + 1) + "/" + date.getDate() + "/" + date.getFullYear();
    }
    else
    {
        return date.getDate() + "/" + (date.getMonth() + 1) + "/" + date.getFullYear();
    }
}
```

ADOBE FLEX

Flex 框架包括设置日期、时间和货币的格式的控件。这些控件包括 DateFormatter 控件和 CurrencyFormatter 控件。

- mx:DateFormatter
- mx:CurrencyFormatter

第 58 章：关于 HTML 环境

Adobe AIR 1.0 和更高版本

Adobe®AIR® 使用 WebKit （www.webkit.org， Safari Web 浏览器也使用它）分析、布局和呈现 HTML 和 JavaScript 内容。在 HTML 内容中使用 AIR API 是可选的。您可以完全使用 HTML 和 JavaScript 在 HTMLLoader 对象或 HTML 窗口的内容中编程。大多数现有 HTML 页和应用程序只需少量更改即可运行 （假定它们使用的 HTML、 CSS、 DOM 和 JavaScript 功能与 WebKit 兼容）。

重要说明: Adobe AIR 运行时的新版本可能包含 WebKit 的更新版本。AIR 新版本中的 WebKit 更新可能会对已部署的 AIR 应用程序造成意外更改。这些更改可能会影响应用程序中 HTML 内容的行为或外观。例如， WebKit 呈现中的改进或更正可能会更改应用程序用户界面中元素的布局。为此，我们强烈建议您在应用程序中提供一个更新机制。如果因 AIR 中包含的 WebKit 版本发生更改而需要更新应用程序， AIR 更新机制可提示用户安装应用程序的新版本。

下表列出了所使用的 WebKit 版本与 AIR 中使用的 WebKit 版本相同的 Safari Web 浏览器版本:

AIR 版本	Safari 版本
1.0	2.04
1.1	3.04
1.5	4.0 测试版
2.0	4.03
2.5	4.03
2.6	4.03
2.7	4.03
3	5.0.3

您始终可以通过检查由 HTMLLoader 对象返回的默认用户代理字符串来确定 WebKit 的已安装版本:

```
var htmlLoader:HTMLLoader = new HTMLLoader();
trace( htmlLoader.userAgent );
```

请记住，AIR 中使用的 WebKit 版本与开放源版本不同。AIR 中不支持某些功能，并且 AIR 版本可以包括在相应 WebKit 版本中尚不可用的安全性和错误修复功能。请参阅 第 834 页的 “AIR 中不支持的 WebKit 功能 ”。

由于 AIR 应用程序直接在桌面上运行，且具有对文件系统的完全访问权限，因此， HTML 内容的安全模型比典型 Web 浏览器的安全模型更加严格。在 AIR 中，只有从应用程序安装目录加载的内容才会被放置到应用程序沙箱 中。应用程序沙箱具有最高级别的权限，且允许访问 AIR API。AIR 根据其他内容的来源将这些内容放置到隔离沙箱中。从文件系统加载的文件放置到本地沙箱中。使用 http: 或 https: 协议从网络加载的文件则根据远程服务器的域放置到相应沙箱中。禁止这些非应用程序沙箱中的内容访问任何 AIR API，且其运行方式与在典型 Web 浏览器中几乎一样。

如果应用 Alpha、缩放或透明度设置，则 AIR 中的 HTML 内容不显示 SWF 或 PDF 内容。有关详细信息，请参阅第 860 页的 “ 在 HTML 页中加载 SWF 或 PDF 内容时的注意事项 ” 和第 765 页的 “ 窗口透明度 ”。

更多帮助主题

Webkit DOM 参考

Safari HTML 参考

Safari CSS 参考

www.webkit.org

HTML 环境概述

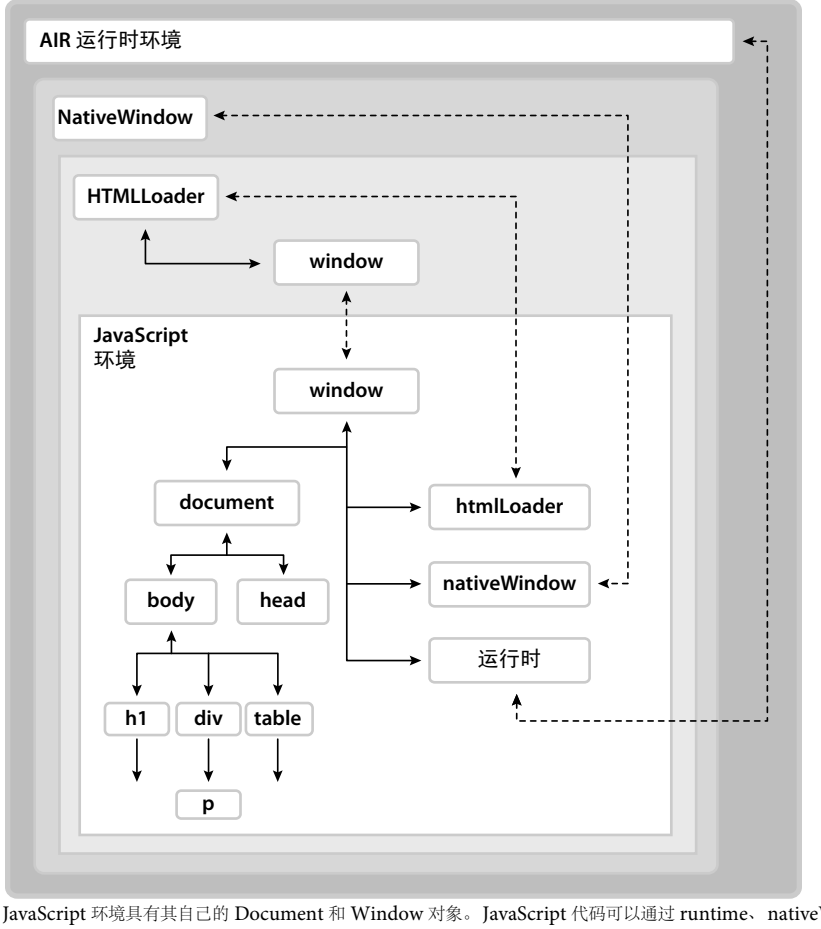
Adobe AIR 1.0 和更高版本

Adobe AIR 使用 HTML 渲染器、文档对象模型和 JavaScript 解释程序提供与浏览器完全相似的 JavaScript 环境。JavaScript 环境通过 AIR HTMLLoader 类表示。在 HTML 窗口中，HTMLLoader 对象包含所有 HTML 内容，而该对象又包含在 NativeWindow 对象中。在 SWF 内容中，扩展 Sprite 类的 HTMLLoader 类可以与任何其他显示对象一样添加到舞台显示列表中。 Adobe® ActionScript® 3.0 类属性在第 859 页的 " 为 AIR HTML 容器编写脚本 " 和用于 Adobe® Flash® Platform 的 ActionScript® 3.0 参考中都有所介绍。在 Flex 框架中， AIR HTMLLoader 类包装在 mx:HTML 组件中。mx:HTML 组件扩展 UIComponent 类，因此可以直接与其他 Flex 容器配合使用。mx:HTML 组件中的 JavaScript 环境则是相同的。

关于 JavaScript 环境及其与 AIR 主机之间的关系

Adobe AIR 1.0 和更高版本

下图演示了 JavaScript 环境和 AIR 运行时环境的关系。虽然只显示了一个本机窗口，但是一个 AIR 应用程序可以包含多个窗口。（并且一个窗口可以包含多个 HTMLLoader 对象。）



JavaScript 环境具有其自己的 Document 和 Window 对象。 JavaScript 代码可以通过 runtime、 nativeWindow 和 htmlLoader 属性与 AIR 运行时环境交互。 ActionScript 代码可以通过 HTMLLoader 对象的 window 属性与 JavaScript 环境交互，该属性是对 JavaScript Window 对象的引用。此外，ActionScript 和 JavaScript 对象均可以侦听由 AIR 和 JavaScript 对象调度的事件。

runtime 属性提供了对 AIR API 类的访问权限，使您可以新建 AIR 对象和访问类 （也称为静态）成员。若要访问 AIR API，请向 runtime 属性添加该类的名称和包。例如，若要创建 File 对象，您将使用以下语句：

```
var file = new window.runtime.filesystem.File();
```

注：AIR SDK 提供了一个 JavaScript 文件，即 AIRAliases.js，该文件为最常用的 AIR 类定义了更便于使用的别名。导入该文件时，您可以使用 air.Class 简短形式替换 window.runtime.package.Class。例如，您可以使用 new air.File() 创建 File 对象。

NativeWindow 对象提供了用于控制桌面窗口的属性。您可以使用 window.nativeWindow 属性从 HTML 页内部访问包含的 NativeWindow 对象。

HTMLLoader 对象提供了用于控制内容的加载方式和呈现方式的属性、方法和事件。您可以使用 window.htmlLoader 属性从 HTML 页内部访问父 HTMLLoader 对象。

重要说明：仅当作为应用程序的一部分安装的页作为顶级文档加载时，该页才具有 htmlLoader、 nativeWindow 或 runtime 属性。将文档加载到 frame 或 iframe 中时不会添加这些属性。（只要子文档与父文档位于相同安全沙箱中，子文档即可访问父文档的这些属性。例如，加载到 frame 中的文档可以使用 parent.runtime 访问其父级的 runtime 属性。）

关于安全性

Adobe AIR 1.0 和更高版本

AIR 根据原始域在安全沙箱中执行所有代码。应用程序内容限制为从应用程序安装目录加载的内容，该内容放置到应用程序 沙箱中。只有在此沙箱中运行的 HTML 和 JavaScript 才能访问运行时环境和 AIR API。同时，在页 load 事件的所有处理函数返回之后，将在应用程序沙箱中阻止 JavaScript 的大多数动态计算和执行操作。

通过将应用程序页加载到 frame 或 iframe 中，并对 frame 设置特定于 AIR 的 sandboxRoot 和 documentRoot 属性，可以将该应用程序页映射到非应用程序沙箱中。通过将 sandboxRoot 值设置为实际远程域，您可以使沙箱中的内容跨脚本访问该域中的内容。当加载远程内容或撰写远程内容脚本时 （例如，在 mash-up 应用程序中），使用上述方法映射页非常有用。

允许应用程序和非应用程序内容相互跨脚本访问的另一方法是创建沙箱桥，这也是向 AIR API 授予非应用程序内容访问权限的唯一方法。使用父级到子级 桥，子 frame、 iframe 或窗口中的内容能够访问在应用程序沙箱中定义的指定方法和属性。反之，使用子级到父级 桥，应用程序内容能够访问在子级的沙箱中定义的指定方法和属性。通过设置窗口对象的 parentSandboxBridge 和 childSandboxBridge 属性可以建立沙箱桥。有关详细信息，请参阅 第 925 页的 “Adobe AIR 中的 HTML 安全性 ” 以及 第 831 页的 “HTML frame 和 iframe 元素 ”。

关于插件和嵌入对象

Adobe AIR 1.0 和更高版本

AIR 支持 Adobe® Acrobat® 插件。用户必须安装有 Acrobat 或 Adobe® Reader® 8.1 （或更高版本）才能显示 PDF 内容。HTMLLoader 对象提供了用于查看用户系统是否可以显示 PDF 的属性。 SWF 文件内容也可以在 HTML 环境中显示，但 AIR 中构建有此功能，因此无需使用外部插件。

AIR 中不支持任何其他 WebKit 插件。

更多帮助主题

第 925 页的 “Adobe AIR 中的 HTML 安全性 ”

第 825 页的 “HTML 沙箱 ”

第 831 页的 “HTML frame 和 iframe 元素 ”

第 829 页的 “JavaScript Window 对象 ”

第 826 页的 “XMLHttpRequest 对象 ”

第 469 页的 “ 在 AIR 中添加 PDF 内容 ”

AIR 和 WebKit

Adobe AIR 1.0 和更高版本

Adobe AIR 中使用开放源代码 WebKit 引擎，该引擎也用于 Safari Web 浏览器。 AIR 添加了若干扩展功能，以便允许访问运行时类和对象，并增强了安全性。此外， WebKit 自身还针对 HTML、 CSS 和 JavaScript 添加了 W3C 标准中不包括的功能。

此处仅包含 AIR 新增功能和最显著的 WebKit 扩展功能；有关非标准 HTML、 CSS 和 JavaScript 的其他文档，请参阅 www.webkit.org 和 developer.apple.com。有关标准信息，请参阅 W3C 网站。 Mozilla 还提供了有关 HTML、 CSS 和 DOM 主题的重要一般参考 （当然， WebKit 引擎不同于 Mozilla 引擎）。

AIR 中的 JavaScript

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

AIR 对通用 JavaScript 对象的典型行为进行了若干更改。其中，很多更改都是为了在 AIR 中更方便地编写安全应用程序。同时，这些行为差异表示某些通用 JavaScript 编码模式和使用这些模式的现有 Web 应用程序在 AIR 中可能始终不会按预期方式执行。有关更正这些问题类型的信息，请参阅第 841 页的 “ 避免与安全相关的 JavaScript 错误 ”。

HTML 沙箱

Adobe AIR 1.0 和更高版本

AIR 根据内容的原始位置将该内容放置到隔离沙箱中。沙箱规则与大多数 Web 浏览器实现的具有相同原始位置的策略一致，并且与 Adobe Flash Player 实现的沙箱规则一致。此外， AIR 还提供了一个包含并保护应用程序内容的新应用程序 沙箱类型。有关在开发 AIR 应用程序时可能遇到的沙箱类型的详细信息，请参阅第 895 页的 “ 安全沙箱 ”。

只有在应用程序沙箱中运行的 HTML 和 JavaScript 才能访问运行时环境和 AIR API。但同时，出于安全原因，动态计算和执行各种形式的 JavaScript 在应用程序沙箱内大幅受限。无论您的应用程序实际上是否从服务器直接加载信息，这些限制都将发挥作用。（即使是文件内容、粘贴的字符串和直接用户输入也可能不可靠。）

页内容的原始位置确定要将该内容放置到哪个沙箱。只能将从应用程序目录 （app: URL 方案引用的安装目录）加载的内容放置到应用程序沙箱中。从文件系统加载的内容则放置到与本地文件系统内容交互的 沙箱或受信任的本地 沙箱中，以便访问本地文件系统上的内容 （而不是远程内容），并与其进行交互。从网络加载的内容则放置到与其原始域对应的远程沙箱中。

若要使应用程序页与远程沙箱中的内容自由交互，则可以将该页映射到远程内容所在的相同域中。例如，如果编写显示 Internet 服务的映射数据的应用程序，则可以将加载和显示该服务内容的应用程序页映射到该服务域中。用于将页映射到远程沙箱和域的属性是 frame 和 iframe HTML 元素的新增属性。

若要使非应用程序沙箱中的内容安全地使用 AIR 功能，则可以设置父沙箱桥。若要使应用程序内容安全地调用其他沙箱中的内容的方法并访问其属性，则可以设置子沙箱桥。此处所述的安全性意味着远程内容不能意外获取对未明确公开的对象、属性或方法的引用。通过沙箱桥只能传递简单数据类型、函数和匿名对象。但是，您仍然必须避免明确公开潜在的危险函数。例如，如果公开的接口允许远程内容读取或写入用户系统中任意位置的文件，则可能会使远程内容严重危害用户。

JavaScript eval() 函数

Adobe AIR 1.0 和更高版本

完成加载页之后，将在应用程序沙箱中限制使用 eval() 函数。在某些情况下允许使用该函数，以便可以安全地分析 JSON 格式数据，但是，生成可执行语句的任何计算将生成错误。第 927 页的 " 对不同沙箱中的内容的代码限制 " 介绍了允许使用 eval() 函数的情况。

函数构造函数

Adobe AIR 1.0 和更高版本

在应用程序沙箱中，可以在完成加载页之前使用函数构造函数。在所有页 load 事件处理函数完成之后，无法创建新的函数。

加载外部脚本

Adobe AIR 1.0 和更高版本

应用程序沙箱中的 HTML 页无法使用 script 标签从应用程序目录外部加载 JavaScript 文件。为使应用程序中的页能够从应用程序目录外部加载脚本，必须将该页映射到非应用程序沙箱。

XMLHttpRequest 对象

Adobe AIR 1.0 和更高版本

AIR 提供了应用程序可用于执行数据请求的 XMLHttpRequest (XHR) 对象。下面的示例演示简单数据请求：

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "http:/www.example.com/file.data", true);
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        //do something with data...
    }
}
xmlhttp.send(null);
```

与浏览器不同， AIR 允许在应用程序沙箱中运行的内容请求任何域中的数据。对于包含 JSON 字符串的 XHR 结果，除非该结果还包含可执行代码，否则可以计算出该结果的数据对象。如果 XHR 结果中存在可执行语句，则会引发错误，且计算尝试失败。

若要防止从远程源意外注入代码，在完成加载页之前执行同步 XHR 时将返回空结果。异步 XHR 将始终在加载页之后返回。

默认情况下， AIR 阻止在非应用程序沙箱中执行跨域 XMLHttpRequest。应用程序沙箱中的父窗口可以选择在包含非应用程序沙箱内容的子 frame 中允许跨域请求，方法是在包含非应用程序沙箱内容的 frame 或 iframe 元素中将 AIR 添加的 allowCrossDomainXHR 属性设置为 true：

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    allowCrossDomainXHR="true"
</iframe>
```

注：还可以根据需要使用 AIR URLStream 类下载数据。

如果从包含已映射到远程沙箱的应用程序内容的 frame 或 iframe 对远程服务器调度 XMLHttpRequest，请确保映射 URL 不会遮蔽在 XHR 中使用的服务器地址。例如，请考虑以下 iframe 定义，该定义将应用程序内容映射到 example.com 域所对应的远程沙箱：

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/"
    allowCrossDomainXHR="true"
</iframe>
```

由于 sandboxRoot 属性重新映射 www.example.com 地址的根 URL，因此所有请求都将从应用程序目录加载，而不是从远程服务器加载。无论请求是派生自页导航还是 XMLHttpRequest，都将重新映射这些请求。

若要避免意外阻止对远程服务器的数据请求，请将 sandboxRoot 映射到远程 URL 的子目录，而不是根目录。该目录不一定存在。例如，若要允许从远程服务器加载对 www.example.com 的请求，而不是从应用程序目录加载，请将上面的 iframe 更改为以下内容：

```
<iframe id="mashup"
    src="http://www.example.com/map.html"
    documentRoot="app:/sandbox/"
    sandboxRoot="http://www.example.com/air/"
    allowCrossDomainXHR="true"
</iframe>
```

在本例中，仅在本地加载 air 子目录中的内容。

有关沙箱映射的详细信息，请参阅 第 831 页的 “HTML frame 和 iframe 元素 ” 以及 第 925 页的 “Adobe AIR 中的 HTML 安全性 ”。

Cookie

Adobe AIR 1.0 和更高版本

在 AIR 应用程序中，只有远程沙箱中的内容 （从 http: 和 https: 源加载的内容）才能使用 cookie （即 document.cookie 属性）。在应用程序沙箱中，还提供了存储永久性数据的其他方法，包括 EncryptedLocalStore、SharedObject 和 FileStream 类。

Clipboard 对象

Adobe AIR 1.0 和更高版本

WebKit Clipboard API 由以下事件驱动: copy、cut 和 paste。在这些事件中传递的事件对象通过 clipboardData 属性提供对剪贴板的访问。使用 clipboardData 对象的以下方法可以读取或写入剪贴板数据：

方法	说明
clearData(mimeType)	清除剪贴板数据。将 mimeType 参数设置为要清除的数据的 MIME 类型。
getData(mimeType)	获取剪贴板数据。该方法只能在 paste 事件的处理函数中调用。将 mimeType 参数设置为要返回的数据的 MIME 类型。
setData(mimeType, data)	将数据复制到剪贴板。将 mimeType 参数设置为数据的 MIME 类型。

应用程序沙箱外部的 JavaScript 代码只能通过上述事件访问剪贴板。但是，应用程序沙箱中的内容可以使用 AIR Clipboard 类直接访问系统剪贴板。例如，您可以使用以下语句获取剪贴板上的文本格式数据：

```
var clipping = air.Clipboard.generalClipboard.getData("text/plain",
                                air.ClipboardTransferMode.ORIGINAL_ONLY);
```

有效的数据 MIME 类型为：

MIME 类型	值
文本	"text/plain"
HTML	"text/html"
URL	"text/uri-list"
位图	"image/x-vnd.adobe.air.bitmap"
文件列表	"application/x-vnd.adobe.air.file-list"

重要说明：只有应用程序沙箱中的内容才能访问剪贴板上的文件数据。如果非应用程序内容尝试访问剪贴板上的文件对象，则会引发安全错误。

有关使用剪贴板的详细信息，请参阅第 509 页的 " 复制和粘贴 " 以及 Using the Pasteboard from JavaScript（Apple 开发人员中心）。

拖放

Adobe AIR 1.0 和更高版本

进出 HTML 的拖放手势生成下列 DOM 事件：dragstart、drag、dragend、dragenter、dragover、dragleave 和 drop。在这些事件中传递的事件对象通过 dataTransfer 属性提供对被拖动数据的访问。 dataTransfer 属性引用的对象提供与剪贴板事件关联的 clipboardData 对象相同的方法。例如，您可以使用以下函数获取 drop 事件中的文本格式数据：

```
function onDrop(dragEvent){
    return dragEvent.dataTransfer.getData("text/plain",
            air.ClipboardTransferMode.ORIGINAL_ONLY);
}
```

dataTransfer 对象包含下列重要成员：

成员	说明
clearData(mimeType)	清除数据。将 mimeType 参数设置为要清除的数据表示形式的 MIME 类型。
getData(mimeType)	获取拖动的数据。该方法只能在 drop 事件的处理函数中调用。将 mimeType 参数设置为要获取的数据的 MIME 类型。
setData(mimeType, data)	设置要拖动的数据。将 mimeType 参数设置为数据的 MIME 类型。
类型	一个字符串数组，其中包含 dataTransfer 对象中当前可用的所有数据表示形式的 MIME 类型。
effectsAllowed	指定是否可以复制、移动、链接拖动数据，或者是否可以对拖动数据执行任何组合操作。设置 dragstart 事件的处理函数中的 effectsAllowed 属性。
dropEffect	指定拖动目标支持哪些允许的拖动效果。设置 dragEnter 事件的处理函数中的 dropEffect 属性。拖动期间，光标将发生更改，以便指示在用户释放鼠标后将产生的效果。如果未指定任何 dropEffect，则选择 effectsAllowed 属性效果。复制效果的优先级高于移动效果，而移动效果自身的优先级又高于链接效果。用户可以使用键盘修改默认优先级。

有关向 AIR 应用程序添加拖放支持的详细信息，请参阅第 519 页的 "AIR 中的拖放 " 和通过 JavaScript 使用拖放 （Apple 开发人员中心）。

innerHTML 和 outerHTML 属性

Adobe AIR 1.0 和更高版本

对于在应用程序沙箱中运行的内容， AIR 会对 innerHTML 和 outerHTML 属性的使用施加一些安全限制。在执行页 load 事件之前以及在执行任何 load 事件处理函数期间，innerHTML 和 outerHTML 属性的使用不受限制。但是，一旦完成页加载，则只能使用 innerHTML 或 outerHTML 属性向文档添加静态内容。分配给 innerHTML 或 outerHTML 的字符串中计算结果为可执行代码的任何语句都将被忽略。例如，如果在元素定义中包含事件回调属性，则不会添加事件侦听器。同样，不会计算嵌入的 <script> 标签。有关详细信息，请参阅 第 925 页的 “Adobe AIR 中的 HTML 安全性 ”。

Document.write() 和 Document.writeln() 方法

Adobe AIR 1.0 和更高版本

在执行页的 load 事件之前，可以在应用程序沙箱中不受限制地使用 write() 和 writeln() 方法。但是，一旦完成页加载，调用上述任一方法将不会清除页或创建新页。在非应用程序沙箱中，在完成页加载之后调用 document.write() 或 writeln() 将清除当前页或打开一个新的空白页，这与在大多数 Web 浏览器中相同。

Document.designMode 属性

Adobe AIR 1.0 和更高版本

将 document.designMode 属性设置为值 on 可以编辑文档中的所有元素。内置编辑器支持包括文本编辑、复制、粘贴和拖放。将 designMode 设置为 on 等效于将 body 元素的 contentEditable 属性设置为 true。您可以对大多数 HTML 元素使用 contentEditable 属性，以便定义可以编辑文档的哪些部分。有关其他信息，请参阅第 833 页的 “HTML contentEditable 属性 ”。

unload 事件 （对于 body 和 frameset 对象）

Adobe AIR 1.0 和更高版本

在窗口 （包括应用程序主窗口）的顶级 frameset 或 body 标签中，切勿使用 unload 事件响应要关闭的窗口 （或应用程序），而应使用 NativeApplication 对象的 exiting 事件检测关闭某一应用程序的时间。或者使用 NativeWindow 对象的 closing 事件检测关闭某一窗口的时间。例如，以下 JavaScript 代码将在用户关闭应用程序时显示 ("Goodbye.") 消息：

```
var app = air.NativeApplication.nativeApplication;
app.addEventListener(air.Event.EXITING, closeHandler);
function closeHandler(event)
{
    alert("Goodbye.");
}
```

但是，脚本能够 成功响应因导航 frame、 iframe 或顶级窗口内容而引起的 unload 事件。

注：Adobe AIR 的将来版本可能会删除这些限制。

JavaScript Window 对象

Adobe AIR 1.0 和更高版本

Window 对象在 JavaScript 执行上下文中保持为全局对象。在应用程序沙箱中， AIR 向 JavaScript Window 对象添加了新属性，以便提供对 AIR 内置类和重要主机对象的访问。此外，某些方法和属性的行为会有所不同，这取决于它们是否位于应用程序沙箱中。

Window.runtime 属性 通过 runtime 属性，您可以从应用程序沙箱内部实例化和使用内置运行时类。这些类包括 AIR 和 Flash Player API，但不包括 Flex 框架等。例如，以下语句可创建一个 AIR 文件对象：

```
var preferencesFile = new window.runtime.flash.filesystem.File();
```

AIR SDK 提供的 AIRAliases.js 文件所包含的别名定义使您可以缩短这些引用。例如，将 AIRAliases.js 导入到页后，可以使用以下语句创建 File 对象：

```
var preferencesFile = new air.File();
```

window.runtime 属性仅针对应用程序沙箱中的内容和带有 frame 或 iframe 的父页文档定义。

请参阅第 845 页的 " 使用 AIRAliases.js 文件 "。

Window.nativeWindow 属性 nativeWindow 属性提供对基础本机 Window 对象的引用。使用该属性，您可以为屏幕位置、大小和可见性等窗口函数和属性撰写脚本，并处理关闭、调整大小和移动等窗口事件。例如，以下语句可关闭窗口：

```
window.nativeWindow.close();
```

注：NativeWindow 对象提供的窗口控制功能与 JavaScript Window 对象提供的功能重叠。在这种情况下，您可以根据需要选择其中一种方法。

window.nativeWindow 属性仅针对应用程序沙箱中的内容和带有 frame 或 iframe 的父页文档定义。

Window.htmlLoader 属性 htmlLoader 属性提供对包含 HTML 内容的 AIR HTMLLoader 对象的引用。使用该属性，您可以为 HTML 环境的外观和行为撰写脚本。例如，您可以使用 htmlLoader.paintsDefaultBackground 属性确定该控件是否绘制默认的白色背景：

```
window.htmlLoader.paintsDefaultBackground = false;
```

注：HTMLLoader 对象自身具有一个 window 属性，该属性引用该对象包含的 HTML 内容的 JavaScript Window 对象。您可以使用该属性通过对包含 HTMLLoader 的引用来访问 JavaScript 环境。

window.htmlLoader 属性仅针对应用程序沙箱中的内容和带有 frame 或 iframe 的父页文档定义。

Window.parentSandboxBridge 和 Window.childSandboxBridge 属性 使用 parentSandboxBridge 和 childSandboxBridge 属性，您可以定义父帧和子帧之间的接口。有关详细信息，请参阅第 855 页的 " 跨脚本访问不同安全沙箱中的内容 "。

Window.setTimeout() 和 Window.setInterval() 函数 AIR 对 setTimeout() 和 setInterval() 函数在应用程序沙箱内的使用施加了一些安全限制。当调用 setTimeout() 或 setInterval() 时，您不能定义作为字符串执行的代码。您必须使用函数引用。有关详细信息，请参阅第 843 页的 "setTimeout() 和 setInterval()"。

Window.open() 函数 当在非应用程序沙箱中运行的代码调用 open() 方法时，该方法在调用时仅打开一个窗口，以作为用户交互 （例如鼠标单击或按键）的结果。此外，窗口标题采用应用程序标题作为前缀，以免远程内容打开的窗口模拟应用程序打开的窗口。有关详细信息，请参阅第 930 页的 " 调用 JavaScript window.open() 方法的限制 "。

air.NativeApplication 对象

Adobe AIR 1.0 和更高版本

NativeApplication 对象提供有关应用程序状态的信息，调度若干重要应用程序级别的事件，并提供用于控制应用程序行为的有用函数。将自动创建 NativeApplication 对象的单个实例，并且可通过用户定义的 NativeApplication.nativeApplication 属性访问该实例。

若要通过 JavaScript 代码访问该对象，您可以使用：

```
var app = window.runtime.flash.desktop.NativeApplication.nativeApplication;
```

或者，如果已导入 AIRAliases.js 脚本，则可以使用以下简短形式：

```
var app = air.NativeApplication.nativeApplication;
```

NativeApplication 对象只能从应用程序沙箱内部访问。有关 NativeApplication 对象的详细信息，请参阅第 759 页的 " 处理 AIR 运行时和操作系统信息 "。

JavaScript URL 方案

Adobe AIR 1.0 和更高版本

在应用程序沙箱内部阻止执行在 JavaScript URL 方案 （如在 href="javascript:alert('Test')" 中）定义的代码。不引发错误。

AIR 中的 HTML

Adobe AIR 1.0 和更高版本

AIR 和 WebKit 定义了几个非标准 HTML 元素和属性，包括：

第 831 页的 “HTML frame 和 iframe 元素 ”

第 832 页的 “HTML 元素事件处理函数 ”

HTML frame 和 iframe 元素

Adobe AIR 1.0 和更高版本

AIR 向应用程序沙箱中的内容的 frame 和 iframe 元素添加了以下新属性：

sandboxRoot 属性 sandboxRoot 属性为帧的 src 属性指定的文件指定一个替代非应用程序原始域。该文件加载到与指定域对应的非应用程序沙箱中。该文件中的内容和从指定域加载的内容可以相互跨脚本访问对方。

重要说明：如果将 sandboxRoot 的值设置为该域的基本 URL，则从应用程序目录而不是远程服务器加载对该域中的内容的所有请求 （无论相应请求是通过页导航、 XMLHttpRequest 还是任何其他内容加载方式生成）。

documentRoot 属性 documentRoot 属性指定本地目录，将通过该目录加载解析到 sandboxRoot 指定的位置内文件的 URL。

当解析帧的 src 属性中或加载到帧中的内容中的 URL 时，与 sandboxRoot 中的指定值匹配的 URL 部分将替换为 documentRoot 中的指定值。因此，在以下 frame 标签中：

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

child.html 从应用程序安装文件夹的 sandbox 子目录加载。 child.html 中的相对 URL 基于 sandbox 目录进行解析。请注意，在此帧中无法访问位于 www.example.com/air 的远程服务器上的任何文件，这是因为 AIR 将尝试从 app:/sandbox/ 目录加载这些文件。

allowCrossDomainXHR 属性 在帧开始标签中包含 allowCrossDomainXHR="allowCrossDomainXHR"，以便允许该帧中的内容对任何远程域执行 XMLHttpRequest 操作。默认情况下，非应用程序内容只能对其自身的原始域执行这样的请求。允许跨域 XHR 涉及严重的安全影响。页中的代码能够与任何域交换数据。如果由于某种原因向页注入恶意内容，则会削弱当前沙箱中可供代码访问的任何数据的安全性。仅当确实需要执行跨域数据加载时才仅针对您创建和控制的页启用跨域 XHR。此外，请仔细验证由页加载的所有外部数据，以防止代码注入或其他形式的攻击。

重要说明：如果 allowCrossDomainXHR 属性包含在 frame 或 iframe 元素中，则启用跨域 XHR （除非分配的值为 “0” 或者以字母 “f” 或 “n” 开头）。例如，将 allowCrossDomainXHR 设置为 “deny” 仍将启用跨域 XHR。如果要禁用跨域请求，请将该属性完全置于元素声明之外。

ondominitialize 属性 为帧的 dominitialize 事件指定事件处理函数。该事件是在创建帧的窗口和文档对象之后以及在分析任何脚本或创建文档元素之前引发的特定于 AIR 的事件。

帧会在加载序列中尽早调度 dominitialize 事件，以使子页中的任何脚本均可引用由 dominitialize 处理函数添加到子文档的对象、变量和函数。父页必须与子页位于相同沙箱中才能直接添加或访问子文档中的任何对象。但是，应用程序沙箱中的父级可以建立沙箱桥，以便与非应用程序沙箱中的内容通信。

下面的示例演示 AIR 中 iframe 标签的用法：

在无需映射到远程服务器上的实际域的情况下将 child.html 放置到远程沙箱中：

```
<iframe src="http://localhost/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://localhost/air/"/>
```

在仅允许对 www.example.com 执行 XMLHttpRequest 的情况下将 child.html 放置到远程沙箱中：

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"/>
```

在允许对任何远程域执行 XMLHttpRequest 的情况下将 child.html 放置到远程沙箱中：

```
<iframe src="http://www.example.com/air/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="http://www.example.com/air/"
            allowCrossDomainXHR="allowCrossDomainXHR"/>
```

将 child.html 放置到只能与本地文件系统内容交互的沙箱中：

```
<iframe     src="file:///templates/child.html"
            documentRoot="app:/sandbox/"
            sandboxRoot="app-storage:/templates/"/>
```

使用 dominitialize 事件建立沙箱桥，并将 child.html 放置到远程沙箱中：

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

以下 child.html 文档说明子级内容如何访问父级沙箱桥：

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

有关详细信息，请参阅第 855 页的 “ 跨脚本访问不同安全沙箱中的内容 ” 和 第 925 页的 “Adobe AIR 中的 HTML 安全性 ”。

HTML 元素事件处理函数

Adobe AIR 1.0 和更高版本

AIR 和 WebKit 中的 DOM 对象调度未包含在标准 DOM 事件模型中的某些事件。下表列出了为这些事件指定处理函数可使用的相关事件属性：

回调属性名称	说明
oncontextmenu	当调用上下文菜单时调用，例如通过右键单击或按住 Command 并单击所选文本。
oncopy	当复制元素中的所选内容时调用。
oncut	当剪切元素中的所选内容时调用。
ondominitialize	在创建加载到 frame 或 iframe 的文档的 DOM 之后以及在创建任何 DOM 元素或分析脚本之前调用。
ondrag	当拖动元素时调用。
ondragend	当释放拖动动作时调用。
ondragenter	当拖动手势进入元素范围时调用。
ondragleave	当拖动手势离开元素范围时调用。
ondragover	当拖动手势位于元素范围内部时持续调用。
ondragstart	当拖动手势开始时调用。
ondrop	当在元素上释放拖动手势时调用。
onerror	当在加载元素期间出错时调用。
oninput	当在表单元素中输入文本时调用。
onpaste	将项目粘贴到元素中时调用。
onscroll	当滚动可滚动元素的内容时调用。
onselectstart	当开始选择时调用。

HTML contentEditable 属性

Adobe AIR 1.0 和更高版本

您可以向任何 HTML 元素添加 contentEditable 属性，以便允许用户编辑该元素的内容。例如，以下 HTML 示例代码将除第一个 p 元素以外的整个文档设置为可编辑。

```
<html>
<head/>
<body contentEditable="true">
    <h1>de Finibus Bonorum et Malorum</h1>
    <p contentEditable="false">Sed ut perspiciatis unde omnis iste natus error.</p>
    <p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis.</p>
</body>
</html>
```

注：如果将 document.designMode 属性设置为 on，则该文档中的所有元素都是可编辑的，而不管各元素的 contentEditable 设置如何。但是，将 designMode 设置为 off 不会禁用对 contentEditable 为 true 的元素的编辑。有关其他信息，请参阅第 829 页的 “Document.designMode 属性 ”。

Data: URL

Adobe AIR 2 和更高版本

AIR 支持下列元素的 data: URL：

- img

- input type=”image”
- CSS 规则，允许使用图像 （例如，背景图像）

数据 URL 允许您将二进制图像数据作为 Base64 编码的字符串直接插入到 CSS 或 HTML 文档。以下示例使用 data: URL 作为重复的背景：

```
<html>
<head>
<style>
body {
background-
image:url('
ZSBJbWFnZVJlYWR5ccllPAAAAAZQTFRF%2F6cA%2F%2F%2F%2Fgxp3lwAAAAJ0Uk5T%2FwDltzBKAAABF0lEQVR42uzZQQ7CMAxE0e%2F
7X5oNCyRocWzPiJbMBZ6qpIljE%2BnwklgKG7kwUjc2IkIaxkY0CPdEsCCasws6ShXBgmBBmEagpXQQLAgWBAuSY2gaKaWPYEGwIEwg0F
RmECwIFoQeQjJlhJWUEFazjFDJCkI5WYRWMgjtfEGYyQnCXD4jTCdm1zmngFpBFznwVNi5RPSbwbWnpYr%2BBHi%2FtCTfgPLEPL7jBct
AKBRptXJ8M%2BprIuZKu%2BUKcg4YK1PLz7kx4bSqHyPaT4d%2B28OCJJiRBo4FCQsSA0bziT3XubMgYUG6fc5fatmGBQkL0hoJ1IaZMi
QsSFiQ8vRscTjlQOI2iHZwtpHuf%2BJAYiOiJSkj8Z%2FIQ4ABANvXGLd3%2BZMrAAAAAElFTkSuQmCC');
background-repeat:repeat;
}
</style>
</head>
<body>
</body>
</html>
```

使用 data: URL 时，一定要注意多余的空格。例如，数据字符串必须作为单一、完整的行输入。否则，换行符将被视为数据的一部分，并且无法解码图像。

AIR 中的 CSS

Adobe AIR 1.0 和更高版本

WebKit 支持多个扩展 CSS 属性。其中，许多扩展名使用以下前缀：-webkit。请注意，其中某些扩展名本质上是实验性的，并可能从 WebKit 的将来版本中删除。有关 CSS 支持的 Webkit 和适用于 CSS 的 Webkit 扩展名的详细信息，请参阅 Safari CSS 参考。

AIR 中不支持的 WebKit 功能

Adobe AIR 1.0 和更高版本

AIR 不支持 WebKit 或 Safari 4 中提供的下列功能：

- 通过 window.postMessage 进行跨域消息传递 （AIR 提供自己的跨域通信 API）
- CSS 变量
- Web 开放字体格式 (WOFF) 和 SVG 字体。
- HTML 视频和音频标签
- 媒体设备查询
- 脱机应用程序缓存
- 打印 （AIR 提供自己的 PrintJob API）
- 拼写和语法检查
- SVG
- WAI-ARIA

- WebSocket （AIR 提供自己的 Socket API）
- Web workers
- WebKit SQL API （AIR 提供自己的 API）
- WebKit geolocation API （在受支持的设备中， AIR 提供自己的 geolocation API）
- WebKit 多文件上传 API
- WebKit 触摸事件 （AIR 提供自己的触摸事件）
- 无线标记语言 (WML)

下面的列表包含 AIR 不支持的特定 JavaScript API、 HTML 元素以及 CSS 属性和值：

不支持的 **JavaScript Window** 对象成员：

- applicationCache()
- console
- openDatabase()
- postMessage()
- document.print()

不支持的 **HTML** 标签：

- audio
- video

不支持的 **HTML** 属性：

- aria-*
- draggable
- formnovalidate
- list
- novalidate
- onbeforeload
- onhashchange
- onorientationchange
- onpagehide
- onpageshow
- onpopstate
- ontouchstart
- ontouchmove
- ontouchend
- ontouchcancel
- onwebkitbeginfullscreen
- onwebkitendfullscreen
- pattern

- required
- sandbox

不支持的 **JavaScript** 事件：
- beforeload
- hashchange
- orientationchange
- pagehide
- pageshow
- popstate
- touchstart
- touchmove
- touchend
- touchcancel
- webkitbeginfullscreen
- webkitendfullscreen

不支持的 **CSS** 属性：
- background-clip
- background-origin （使用 -webkit-background-origin）
- background-repeat-x
- background-repeat-y
- background-size （使用 -webkit-background-size）
- border-bottom-left-radius
- border-bottom-right-radius
- border-radius
- border-top-left-radius
- border-top-right-radius
- text-rendering
- -webkit-animation-play-state
- -webkit-background-clip
- -webkit-color-correction
- -webkit-font-smoothing

不支持的 **CSS** 值：
- 外观属性值：
  - media-volume-slider-container

  - media-volume-slider
  - media-volume-sliderthumb
  - outer-spin-button
- border-box （background-clip 和 background-origin）
- contain (background-size)
- content-box （background-clip 和 background-origin）
- cover (background-size)
- 列表属性值:
  - afar
  - amharic
  - amharic-abegede
  - cjk-earthly-branch
  - cjk-heavenly-stem
  - ethiopic
  - ethiopic-abegede
  - ethiopic-abegede-am-et
  - ethiopic-abegede-gez
  - ethiopic-abegede-ti-er
  - ethiopic-abegede-ti-et
  - ethiopic-halehame-aa-er
  - ethiopic-halehame-aa-et
  - ethiopic-halehame-am-et
  - ethiopic-halehame-gez
  - ethiopic-halehame-om-et
  - ethiopic-halehame-sid-et
  - ethiopic-halehame-so-et
  - ethiopic-halehame-ti-er
  - ethiopic-halehame-ti-et
  - ethiopic-halehame-tig
  - hangul
  - hangul-consonant
  - lower-norwegian
  - oromo
  - sidama
  - somali

  - tigre
  - tigrinya-er
  - tigrinya-er-abegede
  - tigrinya-et
  - tigrinya-et-abegede
  - upper-greek
  - upper-norwegian
- -wap-marquee （显示属性）

第 59 章：在 AIR 中进行 HTML 和 JavaScript 编程

Adobe AIR 1.0 和更高版本

许多编程主题是专门针对使用 HTML 和 JavaScript 开发 Adobe® AIR® 应用程序而编写的。无论是要编写基于 HTML 的 AIR 应用程序，还是要编写使用 HTMLLoader 类（或 mx:HTML Flex™ 组件）运行 HTML 和 JavaScript 的基于 SWF 的 AIR 应用程序，以下信息都非常重要。

关于 HTMLLoader 类

Adobe AIR 1.0 和更高版本

Adobe AIR 的 HTMLLoader 类定义可在 AIR 应用程序中显示 HTML 内容的显示对象。基于 SWF 的应用程序可以向现有窗口中添加一个 HTMLLoader 控件，也可以使用 HTMLLoader.createRootWindow() 创建 HTML 窗口，该窗口会自动包含 HTMLLoader 对象。可以通过 JavaScript window.htmlLoader 属性从加载的 HTML 页内部访问 HTMLLoader 对象。

从 URL 加载 HTML 内容

Adobe AIR 1.0 和更高版本

以下代码将 URL 加载到 HTMLLoader 对象中（添加 HTMLLoader 作为舞台或其他显示对象容器的子容器，以便在应用程序中显示 HTML 内容）：

```
import flash.html.HTMLLoader;

var html:HTMLLoader = new HTMLLoader;
html.width = 400;
html.height = 600;
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
```

HTMLLoader 对象的 width 和 height 属性默认情况下均设置为 0。向舞台添加 HTMLLoader 对象时需要设置这些尺寸。HTMLLoader 在加载页面时会调度多个事件。可以使用这些事件来确定何时与加载的页面进行交互是安全的。第 874 页的 “处理 AIR 中与 HTML 相关的事件” 中对这些事件进行了介绍。

注：在 Flex 框架中，只有 UIComponent 类的扩展类才可以作为 Flex 容器组件的子组件进行添加。因此，无法将 HTMLLoader 作为 Flex 容器组件的子组件直接进行添加；但是可以通过以下方式添加：使用 Flex mx:HTML 控件；构建一个自定义类对 UIComponent 进行扩展，并将 HTMLLoader 作为 UIComponent 的子组件包含在其中；将 HTMLLoader 作为 UIComponent 的子组件进行添加，然后将 UIComponent 添加到 Flex 容器中。

您也可以使用 TextField 类来呈现 HTML 文本，但该类的功能受到限制。Adobe® Flash® Player 的 TextField 类支持 HTML 标记的子集，但是由于大小限制，该类的功能受到限制。（Flash Player 不支持 Adobe AIR 中包含的 HTMLLoader 类。）

从字符串加载 HTML 内容

Adobe AIR 1.0 和更高版本

HTMLLoader 对象的 loadString() 方法可以将 HTML 内容字符串加载到 HTMLLoader 对象中：

```
var html:HTMLLoader = new HTMLLoader();
var htmlStr:String = "<html><body>Hello <b>world</b>.</body></html>";
html.loadString(htmlStr);
```

默认情况下，通过 loadString() 方法加载的内容放置在非应用程序沙箱中，并具有以下特征：

- 有权从网络加载内容 （但不能从文件系统加载）。
- 无法使用 XMLHttpRequest 加载数据。
- window.location 属性设置为 "about:blank"。
- 内容无法访问 window.runtime 属性 （像任何非应用程序沙箱中的内容那样）。

在 AIR 1.5 中，HTMLLoader 类包含 placeLoadStringContentInApplicationSandbox 属性。对 HTMLLoader 对象将此属性设置为 true 时，将通过 loadString() 方法加载的内容放置在应用程序沙箱中。（默认值为 false。）这样就使通过 loadString() 方法加载的内容有权访问 window.runtime 属性和所有 AIR API。如果将此属性设置为 true，请确保在调用 loadString() 方法时所使用的字符串的数据源受信任。此属性设置为 true 时，将以完全应用程序权限执行 HTML 字符串中的代码语句。请仅在确定字符串不会包含有害代码时才将此属性设置为 true。

在用 AIR 1.0 或 AIR 1.1 SDK 编译的应用程序中，通过 loadString() 方法加载的内容放置在应用程序沙箱中。

在 AIR 应用程序中使用 HTML 的重要安全规则

Adobe AIR 1.0 和更高版本

随 AIR 应用程序一起安装的文件能够访问 AIR API。出于安全方面的考虑，来自其他源的内容不能访问 AIR API。例如，此限制将阻止远程域 （例如 http://example.com）中的内容读取用户桌面目录中的内容 （也可能是更严重的情况）。

由于存在可通过调用 eval() 函数 （及相关 API）来利用的安全漏洞，因此，默认情况下限制使用这些方法。但是，某些 Ajax 框架会调用 eval() 函数和相关 API。

为确保结构内容在 AIR 应用程序中能够正常工作，必须考虑对来自不同源的内容制订相应的安全限制规则。来自不同源的内容分别放置在不同的安全性分类中，称为沙箱 （请参阅第 895 页的 “ 安全沙箱 ”）。默认情况下，随应用程序一起安装的内容安装在称为应用程序 的沙箱中，这将授予该内容访问 AIR API 的权限。应用程序沙箱通常是最安全的沙箱，设计了一些限制，可阻止不受信任代码的执行。

运行时允许将随应用程序一起安装的内容加载到应用程序沙箱之外的沙箱中。非应用程序沙箱中的内容在类似于典型 Web 浏览器的安全环境中运行。例如，非应用程序沙箱中的代码可以使用 eval() 和相关方法 （但不允许该代码访问 AIR API）。运行时包含有相关方法，可以让不同沙箱中的内容安全地进行通信 （例如，不将 AIR API 公开给非应用程序内容）。有关详细信息，请参阅第 855 页的 “ 跨脚本访问不同安全沙箱中的内容 ”。

如果出于安全方面的考虑，限制在沙箱中使用所调用的代码，则运行时将调度 JavaScript 错误：“Adobe AIR runtime security violation for JavaScript code in the application security sandbox”（应用程序安全沙箱中存在针对 JavaScript 代码的 Adobe AIR 运行时安全侵犯）。

为了避免此错误，请按照下一部分第 841 页的 “ 避免与安全相关的 JavaScript 错误 ” 中介绍的代码编写方法进行操作。

有关详细信息，请参阅 第 925 页的 “Adobe AIR 中的 HTML 安全性 ”。

避免与安全相关的 JavaScript 错误

Adobe AIR 1.0 和更高版本

如果由于这些安全限制而限制在沙箱中使用所调用的代码，则运行时将调度 JavaScript 错误："Adobe AIR runtime security violation for JavaScript code in the application security sandbox"（应用程序安全沙箱中存在针对 JavaScript 代码的 Adobe AIR 运行时安全侵犯）。为了避免此错误，请按照这些代码编写方法进行操作。

产生与安全相关的 JavaScript 错误的原因

Adobe AIR 1.0 和更高版本

一旦触发文档 load 事件并退出所有 load 事件处理函数，将限制应用程序沙箱中执行的代码执行涉及计算和执行字符串的大多数操作。如果尝试使用以下类型的可计算和执行潜在不安全字符串的 JavaScript 语句，则会生成 JavaScript 错误：

- eval() 函数
- setTimeout() 和 setInterval()
- Function 构造函数

此外，以下类型的 JavaScript 语句也会失败，但不会生成不安全的 JavaScript 错误：

- javascript: URL
- innerHTML 和 outerHTML 语句中通过 onevent 属性分配的事件回调
- 从应用程序安装目录外部加载 JavaScript 文件
- document.write() 和 document.writeln()
- load 事件之前或 load 事件处理函数中的同步 XMLHttpRequests
- 动态创建的脚本元素

注：在某些受限制的情况下，允许执行字符串运算。有关详细信息，请参阅第 927 页的 " 对不同沙箱中的内容的代码限制 "。

Adobe 维护了一个已知的支持应用程序安全沙箱的 Ajax 框架列表，可以通过 http://www.adobe.com/go/airappsandboxframeworks_cn 访问该列表。

以下部分介绍了如何针对应用程序沙箱中运行的代码改写脚本，以避免这些不安全的 JavaScript 错误和无提示失败。

将应用程序内容映射到其他沙箱

Adobe AIR 1.0 和更高版本

在大多数情况下，可以改写或重构应用程序以避免与安全相关的 JavaScript 错误。但是，如果无法进行改写或重构，则可以采用第 856 页的 " 将应用程序内容加载到非应用程序沙箱 "中介绍的技术将应用程序内容加载到其他沙箱。如果该内容还必须访问 AIR API，则可以按照第 856 页的 " 设置沙箱桥接口 "中的说明创建一个沙箱桥。

eval() 函数

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在应用程序沙箱中， eval() 函数只能用在页面 load 事件之前或用在 load 事件处理函数中。在页面加载之后，调用 eval() 将不会执行代码。但是，在下面的情况中，可以通过改写代码来避免使用 eval()。

将属性分配给对象

Adobe AIR 1.0 和更高版本

不再通过分析字符串来构建属性存取器：

```
eval("obj." + propName + " = " + val);
```

而是使用中括号记号来访问属性：

```
obj[propName] = val;
```

创建从上下文中获得变量的函数

Adobe AIR 1.0 和更高版本

将如下所示的语句：

```
function compile(var1, var2){
    eval("var fn = function(){ this."+var1+"(var2) }");
    return fn;
}
```

替换为：

```
function compile(var1, var2){
    var self = this;
    return function(){ self[var1](var2) };
}
```

创建使用类名称作为字符串参数的对象

Adobe AIR 1.0 和更高版本

假设有一个 JavaScript 类，其代码定义如下：

```
var CustomClass =
    {
        Utils:
        {
            Parser: function(){ alert('constructor') }
        },
        Data:
        {

        }
    };
var constructorClassName = "CustomClass.Utils.Parser";
```

最简单的实例创建方法是使用 eval()：

```
var myObj;
eval('myObj=new ' + constructorClassName +'()')
```

但是，通过分析类名称的各个部分并使用中括号记号构建新对象，可以避免调用 eval()：

```
function getter(str)
{
    var obj = window;
    var names = str.split('.');
    for(var i=0;i<names.length;i++){
        if(typeof obj[names[i]]=='undefined'){
            var undefstring = names[0];
            for(var j=1;j<=i;j++)
                undefstring+="."+names[j];
            throw new Error(undefstring+" is undefined");
        }
        obj = obj[names[i]];
    }
    return obj;
}
```

若要创建实例，可使用：

```
try{
    var Parser = getter(constructorClassName);
    var a = new Parser();
    }catch(e){
        alert(e);
}
```

setTimeout() 和 setInterval()

Adobe AIR 1.0 和更高版本

将作为处理函数传递的字符串替换为函数引用或对象。例如，将以下语句：

```
setTimeout("alert('Timeout')", 100);
```

替换为：

```
setTimeout(function(){alert('Timeout')}, 100);
```

或者，如果函数要求 this 对象由调用方设置，则将以下语句：

```
this.appTimer = setInterval("obj.customFunction();", 100);
```

替换为：

```
var _self = this;
this.appTimer = setInterval(function(){obj.customFunction.apply(_self);}, 100);
```

Function 构造函数

Adobe AIR 1.0 和更高版本

对 new Function(param, body) 的调用可以替换为内联函数声明或仅在处理页面 load 事件之前使用。

javascript: URL

Adobe AIR 1.0 和更高版本

在应用程序沙箱中，将忽略在使用 javascript: URL 方案的链接中定义的代码。不会生成任何不安全的 JavaScript 错误。可以将如下所示的使用 javascript: URL 的链接：

```
<a href="javascript:code()">Click Me</a>
```

替换为：

```
<a href="#" onclick="code()">Click Me</a>
```

innerHTML 和 outerHTML 语句中通过 onevent 属性分配的事件回调

Adobe AIR 1.0 和更高版本

使用 innerHTML 或 outerHTML 向文档的 DOM 中添加元素时，将忽略在语句内分配的任何事件回调 （如 onclick 或 onmouseover）。不会生成任何安全错误。可以改为向新元素分配 id 属性，并使用 addEventListener() 方法设置事件处理函数回调函数。

例如，在文档中给定一个目标元素，如下所示：

```
<div id="container"></div>
```

将如下所示的语句：

```
document.getElementById('container').innerHTML =
    '<a href="#" onclick="code()">Click Me.</a>';
```

替换为：

```
document.getElementById('container').innerHTML = '<a href="#" id="smith">Click Me.</a>';
document.getElementById('smith').addEventListener("click", function() { code(); });
```

从应用程序安装目录外部加载 JavaScript 文件

Adobe AIR 1.0 和更高版本

不允许从应用程序沙箱外部加载脚本文件。不会生成任何安全错误。在应用程序沙箱中运行的所有脚本文件都必须安装在应用程序目录中。若要在页面中使用外部脚本，必须将页面映射到其他沙箱。请参阅第 856 页的 “ 将应用程序内容加载到非应用程序沙箱 ”。

document.write() 和 document.writeln()

Adobe AIR 1.0 和更高版本

如果页面 load 事件已处理完毕，则将忽略对 document.write() 或 document.writeln() 的调用。不会生成任何安全错误。作为一种替代方法，可以加载新文件，或者使用 DOM 操作技术替换文档的正文。

load 事件之前或 load 事件处理函数中的同步 XMLHttpRequests

Adobe AIR 1.0 和更高版本

在页面 load 事件之前或在 load 事件处理函数中启动的同步 XMLHttpRequests 不会返回任何内容。可以启动异步 XMLHttpRequests，但在 load 事件之前不会返回内容。在处理 load 事件之后，同步 XMLHttpRequests 才能正常工作。

动态创建的脚本元素

Adobe AIR 1.0 和更高版本

动态创建的脚本元素 （例如，使用 innerHTML 或 document.createElement() 方法创建的元素）将被忽略。

通过 JavaScript 访问 AIR API 类

Adobe AIR 1.0 和更高版本

除 Webkit 的标准元素和扩展元素之外，HTML 和 JavaScript 代码还可以访问运行时提供的主机类。通过这些类，可以访问 AIR 提供的高级功能，包括：

- 访问文件系统
- 使用本地 SQL 数据库
- 控制应用程序和窗口菜单
- 访问网络套接字
- 使用用户定义的类和对象
- 声音功能

例如，AIR 文件 API 包含一个 File 类，该类包含在 flash.filesystem 包中。可以在 JavaScript 中创建一个如下所示的 File 对象：

```
var myFile = new window.runtime.flash.filesystem.File();
```

runtime 对象是一个特殊的 JavaScript 对象，可用于在 AIR 应用程序沙箱中运行的 HTML 内容。使用该对象，可以通过 JavaScript 访问运行时类。runtime 对象的 flash 属性提供了对 Flash 包的访问。同样，runtime 对象的 flash.filesystem 属性提供了对 flash.filesystem 包（此包包含 File 类）的访问。包是一种对 ActionScript 中使用的类进行组织的方式。

注：不会自动向 frame 或 iframe 中加载的页面的窗口对象添加 runtime 属性。但是，只要子级文档位于应用程序沙箱中，子级文档就可以访问父级文档的 runtime 属性。

由于运行时类的包结构要求开发人员键入长字符串的 JavaScript 代码字符串（如 window.runtime.flash.desktop.NativeApplication）来访问各个类，因此，AIR SDK 提供了一个 AIRAliases.js 文件，使用该文件，可以更方便地访问运行时类（例如，只需键入 air.NativeApplication 即可）。

本指南主要讨论 AIR API 类。HTML 开发人员可能对 Flash Player API 中的其他类感兴趣，将在《针对 HTML 开发人员的 Adobe AIR API 参考》中介绍这些类。SWF (Flash Player) 内容所使用的语言为 ActionScript。但是，JavaScript 和 ActionScript 语法是类似的。（它们都基于 ECMAScript 语言版本。）JavaScript（在 HTML 内容中）和 ActionScript（在 SWF 内容中）均包含所有内置类。

注：JavaScript 代码无法使用 Dictionary、XML 和 XMLList 类，但这些类在 ActionScript 中是可用的。

使用 AIRAliases.js 文件

Adobe AIR 1.0 和更高版本

运行时类采用包结构的形式进行组织，如下所示：

- window.runtime.flash.desktop.NativeApplication
- window.runtime.flash.desktop.ClipboardManager
- window.runtime.flash.filesystem.FileStream
- window.runtime.flash.data.SQLDatabase

AIR SDK 中包含的 AIRAliases.js 文件提供了“别名”定义，使用这些定义，只需键入很短的内容就可以访问运行时类。例如，只需键入以下内容就可以访问上面列出的类：

- air.NativeApplication
- air.Clipboard

- air.FileStream
- air.SQLDatabase

此列表只列出了 AIRAliases.js 文件中一小部分类。《针对 HTML 开发人员的 Adobe AIR API 参考》中提供了类和包级别函数的完整列表。

除了常用的运行时类之外， AIRAliases.js 文件还包括以下常用包级别函数的别名：window.runtime.trace()、window.runtime.flash.net.navigateToURL() 和 window.runtime.flash.net.sendToURL()，这些函数对应的别名为 air.trace()、air.navigateToURL() 和 air.sendToURL()。

若要使用 AIRAliases.js 文件，请在 HTML 页中包括以下 script 引用：

```
<script src="AIRAliases.js"></script>
```

根据需要调整 src 引用中的路径。

重要说明：如果未明确声明，此文档中的 JavaScript 示例代码均假定已在 HTML 页中包含 AIRAliases.js 文件。

关于 AIR 中的 URL

Adobe AIR 1.0 和更高版本

在 AIR 中运行的 HTML 内容中，可以在定义 img、frame、iframe 和 script 标签的 src 属性时、在 link 标签的 href 属性中，以及可以提供 URL 的其他任何地方，使用以下任意 URL 方案：

URL 方案	说明	示例
file	相对于文件系统根目录的相对路径。	`file:///c:/AIR Test/test.txt`
app	相对于应用程序根安装目录的相对路径。	`app:/images`
app-storage	相对于应用程序存储目录的相对路径。对于每个安装的应用程序，AIR 定义了一个唯一的应用程序存储目录，此目录对于存储特定于该应用程序的数据很有用。	`app-storage:/settings/prefs.xml`
http	标准 HTTP 请求。	`http://www.adobe.com`
https	标准 HTTPS 请求。	`https://secure.example.com`

有关在 AIR 中使用 URL 方案的详细信息，请参阅第 698 页的 “URI 方案 ”。

许多 AIR API（包括 File、Loader、URLStream 和 Sound 类）使用的是 URLRequest 对象，而不是包含 URL 的字符串。URLRequest 对象本身使用字符串进行初始化，在字符串中可以使用以上任意 URL 方案。例如，以下语句创建的 URLRequest 对象可用于请求 Adobe 主页：

```
var urlReq = new air.URLRequest("http://www.adobe.com/");
```

有关 URLRequest 对象的信息，请参阅 第 696 页的 “HTTP 通信 ”。

使 ActionScript 对象可用于 JavaScript

Adobe AIR 1.0 和更高版本

HTMLLoader 对象加载的 HTML 页中的 JavaScript，可以使用 HTML 页的 window.runtime、 window.htmlLoader 和 window.nativeWindow 属性来调用 ActionScript 执行上下文中定义的类、对象和函数。还可以通过在 JavaScript 执行上下文中创建对 ActionScript 对象和函数的引用，从而使其可用于 JavaScript。

从 ActionScript 访问 JavaScript 对象的基本示例

Adobe AIR 1.0 和更高版本

以下示例说明如何添加相关属性，将 ActionScript 对象引用到 HTML 页的全局 window 对象：

```
 var html:HTMLLoader = new HTMLLoader();
var foo:String = "Hello from container SWF."
function helloFromJS(message:String):void {
    trace("JavaScript says:", message);
}
var urlReq:URLRequest = new URLRequest("test.html");
html.addEventListener(Event.COMPLETE, loaded);
html.load(urlReq);

function loaded(e:Event):void{
    html.window.foo = foo;
    html.window.helloFromJS = helloFromJS;
}
```

上例中加载到 HTMLLoader 对象的 HTML 内容（位于名为 test.html 的文件中），可以访问父级 SWF 文件中定义的 foo 属性和 helloFromJS() 方法：

```
 <html>
    <script>
        function alertFoo() {
            alert(foo);
        }
    </script>
    <body>
        <button onClick="alertFoo()">
            What is foo?
        </button>
        <p><button onClick="helloFromJS('Hi.')">
            Call helloFromJS() function.
        </button></p>
    </body>
</html>
```

访问加载文档的 JavaScript 上下文时，可以使用 htmlDOMInitialize 事件按照适当的页面构造顺序尽早创建对象，以便页面中定义的所有脚本均能访问这些对象。如果等待 complete 事件，则只有在页面 load 事件之后运行的页面中的脚本才能访问所添加的对象。

使类定义可用于 JavaScript

Adobe AIR 1.0 和更高版本

若要使应用程序的 ActionScript 类可用于 JavaScript，可以将加载的 HTML 内容分配给包含类定义的应用程序域。JavaScript 执行上下文的应用程序域可以通过 HTMLLoader 对象的 runtimeApplicationDomain 属性进行设置。若要将应用程序域设置为主应用程序域，例如将 runtimeApplicationDomain 设置为 ApplicationDomain.currentDomain，则按以下代码所示：

```
html.runtimeApplicationDomain = ApplicationDomain.currentDomain;
```

一旦设置了 runtimeApplicationDomain 属性， JavaScript 上下文将与所分配的域共享类定义。若要在 JavaScript 中创建自定义类的实例，需通过 window.runtime 属性引用类定义并使用 new 运算符：

```
var customClassObject = new window.runtime.CustomClass();
```

HTML 内容必须来自兼容的安全域。如果 HTML 内容来自所分配的应用程序域之外的其他安全域，则页面将改为使用默认的应用程序域。例如，如果从 Internet 加载远程页面，则不能将 ApplicationDomain.currentDomain 分配为该页面的应用程序域。

删除事件侦听器

Adobe AIR 1.0 和更高版本

在将 JavaScript 事件侦听器添加到当前页面外部的对象时 （包括运行时对象、加载的 SWF 内容中的对象，甚至是其他页面中运行的 JavaScript 对象），则均应在页面卸载时删除这些事件侦听器。否则，事件侦听器会将事件调度给不再存在的处理函数。如果出现这种情况，将会看到以下错误消息：“The application attempted to reference a JavaScript object in an HTML page that is no longer loaded”（应用程序尝试引用已不再加载的 HTML 页中的 JavaScript 对象）。删除不必要的事件侦听器还能使 AIR 回收关联的内存。有关详细信息，请参阅第 878 页的 “ 删除执行导航的 HTML 页面中的事件侦听器 ”。

从 ActionScript 访问 HTML DOM 和 JavaScript 对象

Adobe AIR 1.0 和更高版本

当 HTMLLoader 对象调度 complete 事件之后，即可访问页面的 HTML DOM （文档对象模型）中的所有对象。可访问的对象包括显示元素 （例如页面中的 div 和 p 对象）以及 JavaScript 变量和函数。complete 事件对应于 JavaScript 页面的 load 事件。在调度 complete 之前，可能尚未分析或创建 DOM 元素、变量和函数。如果可能，请等待 complete 事件，然后再访问 HTML DOM。

例如，请看以下 HTML 页：

```
<html>
    <script>
        foo = 333;
        function test() {
            return "OK.";
        }
    </script>
    <body>
        <p id="p1">Hi.</p>
    </body>
</html>
```

这个简单的 HTML 页定义了名为 foo 的 JavaScript 变量和名为 test() 的 JavaScript 函数。两者均为该页面的全局 window 对象的属性。此外，window.document 对象还包括一个名为 P 的元素 （ID 为 p1），可以使用 getElementById() 方法来访问该元素。当页面加载后（HTMLLoader 对象调度 complete 事件时），可以从 ActionScript 访问各个对象，如以下 ActionScript 代码所示：

```
var html:HTMLLoader = new HTMLLoader();
html.width = 300;
html.height = 300;
html.addEventListener(Event.COMPLETE, completeHandler);
var xhtml:XML =
    <html>
        <script>
            foo = 333;
            function test() {
                return "OK.";
            }
        </script>
        <body>
            <p id="p1">Hi.</p>
        </body>
    </html>;
html.loadString(xhtml.toString());

function completeHandler(e:Event):void {
    trace(html.window.foo); // 333
    trace(html.window.document.getElementById("p1").innerHTML); // Hi.
    trace(html.window.test()); // OK.
}
```

若要访问 HTML 元素的内容，可使用 innerHTML 属性。例如，上面的代码使用 html.window.document.getElementById("p1").innerHTML 来获取名为 p1 的 HTML 元素的内容。

还可以从 ActionScript 设置 HTML 页的属性。例如，下面的示例使用对包含 HTMLLoader 对象的引用来设置页面上 p1 元素的内容和 foo JavaScript 变量的值：

```
html.window.document.getElementById("p1").innerHTML = "Goodbye";
html.window.foo = 66;
```

在 HTML 中嵌入 SWF 内容

Adobe AIR 1.0 和更高版本

在 AIR 应用程序中，可以像在浏览器中那样，在 HTML 中嵌入 SWF 内容。使用 object 标签、embed 标签或同时二者可嵌入 SWF 内容。

注：常见的 Web 开发做法是同时使用 object 标签和 embed 标签在 HTML 页中显示 SWF 内容。此做法在 AIR 中毫无益处。您可以只使用 W3C 标准的 object 标签在 AIR 中显示内容。同时，对于浏览器中显示的 HTML 内容，您可以继续同时使用 object 和 embed 标签 （如果需要）。

如果在显示 HTML 和 SWF 内容的 NativeWindow 对象中启用了透明度，则用于嵌入内容的窗口模式 (wmode) 设置为 window 时， AIR 不会显示 SWF 内容。要在透明窗口的 HTML 页中显示 SWF 内容，请将 wmode 参数设置为 opaque 或 transparent。 window 是 wmode 的默认值，因此如果不指定值，则可能不会显示所需内容。

以下示例说明如何使用 HTML object 标签在 HTML 内容中显示 SWF 文件。应将 wmode 参数设置为 opaque 才可显示内容，即使基础 NativeWindow 对象是透明的。SWF 文件加载自应用程序目录，但您可以使用 AIR 支持的任何 URL 方案。（SWF 文件的加载位置决定了 AIR 放置内容的安全沙箱。）

```
<object type="application/x-shockwave-flash" width="100%" height="100%">
    <param name="movie" value="app:/SWFFile.swf"></param>
    <param name="wmode" value="opaque"></param>
</object>
```

还可以使用脚本动态地加载内容。以下示例创建了一个 object 节点，用于显示 urlString 参数中指定的 SWF 文件。该示例将此节点添加为页面元素的子元素，并使用 elementID 参数来指定 ID：

```
<script>
function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    var flash = createSWFObject(urlString, 'opaque', 650, 650);
    displayContainer.appendChild(flash);
}
function createSWFObject(urlString, wmodeString, width, height){
    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");
    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);
    var wmodeParam = document.createElement("param");
    wmodeParam.setAttribute("name","wmode");
    wmodeParam.setAttribute("value",wmodeString);
    SWFObject.appendChild(wmodeParam);
    return SWFObject;
}
</script>
```

如果缩放或旋转 HTMLLoader 对象或者将 alpha 属性设置为 1.0 之外的任何值，则不会显示 SWF 内容。在早于 AIR 1.5.2 的版本中，无论将 wmode 值设置为何值，透明窗口中都不显示 SWF 内容。

注：当嵌入式 SWF 对象尝试加载外部资源 （如视频文件）时，如果 HTML 文件中未提供视频文件的绝对路径，可能无法正确呈现 SWF 内容。不过，嵌入式 SWF 对象可以使用相对路径加载外部图像文件。

下面的示例描述了如何通过嵌入到 HTML 内容中的 SWF 对象加载外部资源：

```
var imageLoader;

function showSWF(urlString, elementID){
    var displayContainer = document.getElementById(elementID);
    imageLoader = createSWFObject(urlString,650,650);
    displayContainer.appendChild(imageLoader);
}

function createSWFObject(urlString, width, height){

    var SWFObject = document.createElement("object");
    SWFObject.setAttribute("type","application/x-shockwave-flash");
    SWFObject.setAttribute("width","100%");
    SWFObject.setAttribute("height","100%");

    var movieParam = document.createElement("param");
    movieParam.setAttribute("name","movie");
    movieParam.setAttribute("value",urlString);
    SWFObject.appendChild(movieParam);

    var flashVars = document.createElement("param");
    flashVars.setAttribute("name","FlashVars");

    //Load the asset inside the SWF content.
    flashVars.setAttribute("value","imgPath=air.jpg");
    SWFObject.appendChild(flashVars);

    return SWFObject;
}
function loadImage()
{
  showSWF("ImageLoader.swf", "imageSpot");

}
```

在下面的 ActionScript 示例中，读取 HTML 文件传递的图像路径，并将该图像加载到舞台上：

```
package
{
      import flash.display.Sprite;
      import flash.display.LoaderInfo;
      import flash.display.StageScaleMode;
      import flash.display.StageAlign;
      import flash.display.Loader;
      import flash.net.URLRequest;

      public class ImageLoader extends Sprite
      {
            public function ImageLoader()
            {

                  var flashvars = LoaderInfo(this.loaderInfo).parameters;

                  if(flashvars.imgPath){
                        var imageLoader = new Loader();
                        var image = new URLRequest(flashvars.imgPath);
                        imageLoader.load(image);
                        addChild(imageLoader);
                        imageLoader.x = 0;
                        imageLoader.y = 0;
                        stage.scaleMode=StageScaleMode.NO_SCALE;
                        stage.align=StageAlign.TOP_LEFT;
                  }
            }
      }
}
```

在 HTML 页中使用 ActionScript 库

Adobe AIR 1.0 和更高版本

AIR 对 HTML 脚本元素进行了扩展，以便页面可以导入编译的 SWF 文件中的 ActionScript 类。例如，若要导入名为 myClasses.swf 的库 （位于应用程序根文件夹的 lib 子目录中），则应在 HTML 文件中包含以下 script 标签：

```
<script src="lib/myClasses.swf" type="application/x-shockwave-flash"></script>
```

重要说明：类型属性必须为 type="application/x-shockwave-flash"，才能正确加载库。

如果将 SWF 内容编译为 Flash Player 10 或 AIR 1.5 SWF，则必须将应用程序描述符文件的 XML 命名空间设置为 AIR 1.5 命名空间。

在对 AIR 文件进行打包时，也必须包含 lib 目录和 myClasses.swf 文件。

通过 JavaScript Window 对象的 runtime 属性访问导入的类：

```
var libraryObject = new window.runtime.LibraryClass();
```

如果 SWF 文件中的类已组织到包中，则同时还必须包含包名称。例如，如果 LibraryClass 定义位于名为 utilities 的包中，则需要使用以下语句来创建该类的实例：

```
var libraryObject = new window.runtime.utilities.LibraryClass();
```

注：若要编译 ActionScript SWF 库使其作为 AIR 中的 HTML 页的一部分，请使用 acompc 编译器。 acompc 实用程序是 Flex SDK 的一部分， Flex SDK 文档中对此有相关说明。

从导入的 ActionScript 文件访问 HTML DOM 和 JavaScript 对象

Adobe AIR 1.0 和更高版本

若要在使用 <script> 标签导入页面的 SWF 文件中从 ActionScript 访问 HTML 页中的对象，需将对 JavaScript 对象的引用（例如 window 或 document）传递给 ActionScript 代码中定义的函数。在函数中使用引用来访问 JavaScript 对象 （或通过传入的引用可访问的其他对象）。

例如，请看以下 HTML 页：

```
<html>
    <script src="ASLibrary.swf" type="application/x-shockwave-flash"></script>
    <script>
        num = 254;
        function getStatus() {
            return "OK.";
        }
        function runASFunction(window){
            var obj = new runtime.ASClass();
            obj.accessDOM(window);
        }
    </script>
    <body onload="runASFunction">
        <p id="p1">Body text.</p>
    </body>
</html>
```

这个简单的 HTML 页包含名为 num 的 JavaScript 变量和名为 getStatus() 的 JavaScript 函数。两者均为该页面的全局 window 对象的属性。此外， window.document 对象还包括一个名为 P 的元素 （ID 为 p1）。

该页加载了一个 ActionScript 文件 “ASLibrary.swf”，其中包含类 ASClass。 ASClass 定义了一个名为 accessDOM() 的函数，该函数可以轻松跟踪这些 JavaScript 对象的值。 accessDOM() 方法将 JavaScript Window 对象作为一个参数。使用此 Window 引用，可访问页面中的其他对象，包括变量、函数和 DOM 元素，如以下定义所示：

```
public class ASClass{
    public function accessDOM(window:*):void {
        trace(window.num); // 254
        trace(window.document.getElementById("p1").innerHTML); // Body text..
        trace(window.getStatus()); // OK.
    }
}
```

可以通过导入的 ActionScript 类同时获取和设置 HTML 页的属性。例如，以下函数设置了页面上 p1 元素的内容，并设置了页面上 foo JavaScript 变量的值：

```
public function modifyDOM(window:*):void {
    window.document.getElementById("p1").innerHTML = "Bye";
    window.foo = 66;
```

转换 Date 和 RegExp 对象

Adobe AIR 1.0 和更高版本

JavaScript 和 ActionScript 语言均定义了 Date 和 RegExp 类，但这些类型的对象并不能自动在两种执行上下文之间进行转换。必须将 Date 和 RegExp 对象转换为等效类型，然后才能在替代执行上下文中使用它们来设置属性或函数参数。

例如，以下 ActionScript 代码可将名为 jsDate 的 JavaScript Date 对象转换为 ActionScript Date 对象：

```
var asDate:Date = new Date(jsDate.getMilliseconds());
```

以下 ActionScript 代码可将名为 jsRegExp 的 JavaScript RegExp 对象转换为 ActionScript RegExp 对象：

```
var flags:String = "";
if (jsRegExp.dotAll) flags += "s";
if (jsRegExp.extended) flags += "x";
if (jsRegExp.global) flags += "g";
if (jsRegExp.ignoreCase) flags += "i";
if (jsRegExp.multiline) flags += "m";
var asRegExp:RegExp = new RegExp(jsRegExp.source, flags);
```

从 ActionScript 操作 HTML 样式表

Adobe AIR 1.0 和更高版本

当 HTMLLoader 对象调度 complete 事件之后，则可以检查和操作页面中的 CSS 样式。

例如，请看以下简单的 HTML 文档：

```
<html>
<style>
    .style1A { font-family:Arial; font-size:12px }
    .style1B { font-family:Arial; font-size:24px }
</style>
<style>
    .style2 { font-family:Arial; font-size:12px }
</style>
<body>
    <p class="style1A">
        Style 1A
    </p>
    <p class="style1B">
        Style 1B
    </p>
    <p class="style2">
        Style 2
    </p>
</body>
</html>
```

当 HTMLLoader 对象加载此内容后，可以通过 window.document.styleSheets 数组的 cssRules 数组来操作页面中的 CSS 样式，如下所示：

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event):void {
    var styleSheet0:Object = html.window.document.styleSheets[0];
    styleSheet0.cssRules[0].style.fontSize = "32px";
    styleSheet0.cssRules[1].style.color = "#FF0000";
    var styleSheet1:Object = html.window.document.styleSheets[1];
    styleSheet1.cssRules[0].style.color = "blue";
    styleSheet1.cssRules[0].style.font-family = "Monaco";
}
```

此代码调整了 CSS 样式，从而生成如下所示的 HTML 文档：

Style 1A

Style 1B

Style 2

请记住，当 HTMLLoader 对象调度 complete 事件之后，该代码可以向页面中添加样式。

跨脚本访问不同安全沙箱中的内容

Adobe AIR 1.0 和更高版本

运行时安全模型将代码与不同的源隔离开来。通过跨脚本访问不同安全沙箱中的内容，可允许一个安全沙箱中的内容访问另一个沙箱中的所选属性和方法。

AIR 安全沙箱和 JavaScript 代码

Adobe AIR 1.0 和更高版本

AIR 强制实施同源策略，以防止一个域中的代码与另一个域中的内容进行交互。所有文件根据其来源放置在相应的沙箱中。通常，应用程序沙箱中的内容不能违反同源原则，并且不能跨脚本访问从应用程序安装目录外部加载的内容。但是， AIR 提供了一些方法，可让您跨脚本访问非应用程序内容。

一种方法是使用 frame 或 iframe 将应用程序内容映射到其他安全沙箱。从应用程序的沙箱区域加载的任何页的行为与从远程域加载该页的行为相同。例如，通过将应用程序内容映射到 example.com 域，该内容可以跨脚本访问从 example.com 加载的页。

由于此方法将应用程序内容放置到其他沙箱中，因此该内容中的代码也不再受计算出的字符串中对代码执行的限制。即使不需要跨脚本访问远程内容，也可以使用这种沙箱映射方法来减弱这些限制。当使用多个 JavaScript 框架中的一个框架或者使用依赖于计算字符串的现有代码时，采用此方法映射内容特别有用。但是，应考虑并防止运行应用程序沙箱以外的内容时可能插入和执行不可信内容的额外风险。

同时，映射到其他沙箱的应用程序内容将失去访问 AIR API 的权利，因此沙箱映射方法不能用于向在应用程序沙箱外部执行的代码公开 AIR 功能。

另一种跨脚本访问的方法，是在非应用程序沙箱中的内容与其在应用程序沙箱中的父级文档之间创建一个名为沙箱桥 的接口。沙箱桥允许子级内容访问父级内容所定义的属性和方法，或允许父级内容访问子级内容所定义的属性和方法，或者两者同时允许。

最后，还可以从应用程序沙箱和其他沙箱 （可选）执行跨域 XMLHttpRequest。

有关详细信息，请参阅 第 831 页的 “HTML frame 和 iframe 元素 ”、第 925 页的 “Adobe AIR 中的 HTML 安全性 ” 和 第 826 页的 “XMLHttpRequest 对象 ”。

将应用程序内容加载到非应用程序沙箱

Adobe AIR 1.0 和更高版本

若要允许应用程序内容安全地跨脚本访问从应用程序安装目录外部加载的内容，可以使用 frame 或 iframe 元素将应用程序内容加载到与外部内容相同的安全沙箱。如果不需要跨脚本访问远程内容，但仍希望加载应用程序沙箱外部的应用程序页，则可以使用同一方法，指定 http://localhost/ 或某些无不利影响的其他值作为源域。

AIR 将向 frame 元素添加新的 sandboxRoot 和 documentRoot 属性，以便允许您指定是否应将加载到该框架中的应用程序文件映射到非应用程序沙箱。对于解析为 sandboxRoot URL 之下的路径的文件，将改为从 documentRoot 目录加载。出于安全方面的考虑，使用此方法加载的应用程序内容将被视为是从 sandboxRoot URL 实际加载的。

sandboxRoot 属性指定用于确定放置框架内容的沙箱和域的 URL。必须使用 file:、http: 或 https: URL 方案。如果您指定的是相对 URL，则内容将保留在应用程序沙箱中。

documentRoot 属性指定从中加载框架内容的目录。必须使用 file:、 app: 或 app-storage: URL 方案。

以下示例对要在远程沙箱中运行的应用程序的 sandbox 子目录中安装的内容以及 www.example.com 域进行了映射：

```
<iframe
    src="http://www.example.com/local/ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

ui.html 页可以使用以下脚本标签从本地的 sandbox 文件夹加载 javascript 文件：

```
<script src="http://www.example.com/local/ui.js"></script>
```

它还可以使用以下脚本标签从远程服务器的目录加载内容：

```
<script src="http://www.example.com/remote/remote.js"></script>
```

sandboxRoot URL 将遮盖远程服务器上位于相同 URL 中的所有内容。在上例中，您不能访问位于 www.example.com/local/（或其子目录中）中的任何远程内容，因为 AIR 会将请求重新映射到本地应用程序目录：无论是页面导航、XMLHttpRequest 还是采用其他内容加载手段所派生的请求，都会重新映射。

设置沙箱桥接口

Adobe AIR 1.0 和更高版本

如果应用程序沙箱中的内容必须访问非应用程序沙箱中的内容所定义的属性或方法，或者如果非应用程序内容必须访问应用程序沙箱中的内容所定义的属性或方法，则可以使用沙箱桥。使用任何子级文档的 window 对象的 childSandboxBridge 和 parentSandboxBridge 属性可创建沙箱桥。

建立子级沙箱桥

Adobe AIR 1.0 和更高版本

childSandboxBridge 属性允许子级文档向父级文档中的内容公开接口。若要公开接口，需将 childSandbox 属性设置为子级文档中的函数或对象。然后，可以从父级文档中的内容访问该对象或函数。以下示例显示了子级文档中运行的脚本如何向其父级文档公开包含函数和属性的对象：

```
var interface = {};
interface.calculatePrice = function(){
    return ".45 cents";
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

如果此子级内容加载到 iframe （分配的 ID 为 “child”），则可以通过读取 frame 的 childSandboxBridge 属性从父级内容来访问接口：

```
var childInterface = document.getElementById("child").contentWindow.childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces ".45 cents"
air.trace(childInterface.storeID)); //traces "abc"
```

建立父级沙箱桥

Adobe AIR 1.0 和更高版本

parentSandboxBridge 属性允许父级文档向子级文档中的内容公开接口。若要公开接口，父级文档需将子级文档的 parentSandbox 属性设置为父级文档中定义的函数或对象。然后，可以从子级文档中的内容访问该对象或函数。以下示例显示了父级 frame 中运行的脚本如何向其子级文档公开包含函数的对象：

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").contentWindow.parentSandboxBridge = interface;
```

使用此接口，子级 frame 中的内容可以将文本保存到名为 save.txt 的文件，但对文件系统不具备任何其他访问权利。子级内容可以调用 save 函数，如下所示：

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

应用程序内容向其他沙箱公开的接口应越窄越好。应考虑到非应用程序内容本身并不可靠，因为它可能遭到意外或恶意代码注入。必须采取适当的防护措施，以防止误用通过父级沙箱桥公开的接口。

在页面加载过程中访问父级沙箱桥

Adobe AIR 1.0 和更高版本

为了使子级文档中的脚本能够访问父级沙箱桥，必须先设置沙箱桥，然后才能运行脚本。创建新页 DOM 之后，在分析任何脚本或添加 DOM 元素之前，Window、frame 和 iframe 对象将调度 dominitialize 事件。可以使用 dominitialize 事件按照适当的页面构造顺序尽早建立沙箱桥，以便页面中定义的所有脚本均能访问该沙箱桥。

以下示例说明如何创建父级沙箱桥，以响应从子级 frame 调度的 dominitialize 事件：

```
<html>
<head>
<script>
var bridgeInterface = {};
bridgeInterface.testProperty = "Bridge engaged";
function engageBridge(){
    document.getElementById("sandbox").contentWindow.parentSandboxBridge = bridgeInterface;
}
</script>
</head>
<body>
<iframe id="sandbox"
            src="http://www.example.com/air/child.html"
            documentRoot="app:/"
            sandboxRoot="http://www.example.com/air/"
            ondominitialize="engageBridge()"/>
</body>
</html>
```

以下 child.html 文档说明子级内容如何访问父级沙箱桥：

```
<html>
    <head>
        <script>
            document.write(window.parentSandboxBridge.testProperty);
        </script>
    </head>
    <body></body>
</html>
```

若要侦听子级窗口（而非框架）上的 dominitialize 事件，必须将侦听器添加到通过 window.open() 函数创建的新子级 window 对象：

```
var childWindow = window.open();
childWindow.addEventListener("dominitialize", engageBridge());
childWindow.document.location = "http://www.example.com/air/child.html";
```

在这种情况下，无法将应用程序内容映射到非应用程序沙箱。只有在从应用程序目录外部加载 child.html 时，此方法才有用。仍可将窗口中的应用程序内容映射到非应用程序沙箱，但必须首先加载一个中间页，在中间页中使用框架来加载子级文档并将其映射到所需的沙箱。

如果使用 HTMLLoader 类的 createRootWindow() 函数来创建窗口，则新窗口不是从中调用 createRootWindow() 的文档的子级。因此，无法从调用窗口建立到加载到新窗口中的非应用程序内容的沙箱桥。而是必须在新窗口中加载一个中间页，在中间页中使用框架来加载子级文档。这样，就可以在新窗口的父级文档与加载到框架中的子级文档之间建立沙箱桥。

第 60 章：为 AIR HTML 容器编写脚本

Adobe AIR 1.0 和更高版本

在 Adobe® AIR® 中， HTMLLoader 类用作 HTML 内容的容器。该类提供了许多属性和方法 （继承自 Sprite 类），用于控制对象在 ActionScript® 3.0 显示列表上的行为和外观。此外，该类还为加载 HTML 内容并与之交互以及管理历史记录等任务定义了相关属性和方法。

HTMLHost 类为 HTMLLoader 定义了一组默认行为。在创建 HTMLLoader 对象时，未提供任何 HTMLHost 实现。因此，当 HTML 内容触发某一默认行为时 （如更改窗口位置或窗口标题），不会发生任何变化。可以对 HTMLHost 类进行扩展，为您的应用程序定义所需的行为。

对于 AIR 创建的 HTML 窗口，提供了 HTMLHost 的默认实现。通过将 defaultBehavior 参数设置为 true 来创建新的 HTMLHost 对象，并使用新创建的 HTMLHost 对象设置 HTMLLoader 对象的 htmlHost 属性，可以将默认 HTMLHost 实现分配给其他 HTMLLoader 对象。

注：在 Adobe® Flex™ 框架中， HTMLLoader 对象采用 mx:HTML 组件进行包装。使用 Flex 时，请使用 HTML 组件。

HTMLLoader 对象的显示属性

Adobe AIR 1.0 和更高版本

HTMLLoader 对象继承 Adobe® Flash® Player Sprite 类的显示属性。例如，可以调整大小、移动、隐藏和更改背景颜色，也可以应用滤镜、遮罩、缩放和旋转等高级效果。在应用效果时，应考虑对易读性的影响。在应用某些效果时，无法显示加载到 HTML 页中的 SWF 和 PDF 内容。

HTML 窗口包含用于呈现 HTML 内容的 HTMLLoader 对象。此对象被限制在窗口区域内，因此，更改尺寸、位置、旋转或缩放系数并不一定能得到令人满意的结果。

基本显示属性

Adobe AIR 1.0 和更高版本

通过 HTMLLoader 的基本显示属性，可以定位控件在其父显示对象中的位置，设置大小以及显示或隐藏控件。不应更改 HTML 窗口的 HTMLLoader 对象的这些属性。

基本属性包括：

属性	备注
x、 y	定位对象在其父容器中的位置。
width、 height	更改显示区域的尺寸。
visible	控制对象及所其包含的所有内容的可见性。

在 HTML 窗口外部，HTMLLoader 对象的 width 和 height 属性的默认值为 0。必须设置宽度和高度才能看到加载的 HTML 内容。HTML 内容根据 HTMLLoader 大小进行绘制，并根据内容中的 HTML 和 CSS 属性进行布置。更改 HTMLLoader 大小会重新填充内容。

在向新的 HTMLLoader 对象 （width 仍设置为 0）中加载内容时，使用 contentWidth 和 contentHeight 属性设置 HTMLLoader 的显示宽度和高度是一种很不错的做法。此项技术适用于根据 HTML 和 CSS 流规则进行布置时具有合理的最小宽度的页。不过，在缺少 HTMLLoader 提供的合理宽度时，有些页会生成窄而长的布局。

注：当更改 HTMLLoader 对象的宽度和高度时， scaleX 和 scaleY 值不会发生更改，大多数其他类型的显示对象也存在此现象。

HTMLLoader 内容的透明度

Adobe AIR 1.0 和更高版本

HTMLLoader 对象的 paintsDefaultBackground 属性 （默认情况下为 true）确定 HTMLLoader 对象是否绘制不透明背景。当 paintsDefaultBackground 为 false 时，背景是透明的。显示对象容器或 HTMLLoader 对象下的其他显示对象在 HTML 内容的前景元素后是可见的。

如果 body 元素或 HTML 文档的任何其他元素指定了背景颜色（例如，使用 style="background-color:gray"），则 HTML 的该部分背景是不透明的，并使用指定的背景颜色呈现。如果设置了 HTMLLoader 对象的 opaqueBackground 属性，并且 paintsDefaultBackground 为 false，则为 opaqueBackground 设置的颜色是可见的。

注: 可以使用透明的 PNG 格式的图形为 HTML 文档中的元素提供 Alpha 混合背景。不支持对 HTML 元素设置不透明样式。

缩放 HTMLLoader 内容

Adobe AIR 1.0 和更高版本

在对 HTMLLoader 对象进行缩放时，缩放系数应避免超过 1.0。如果对 HTMLLoader 对象进行放大，则 HTMLLoader 内容中的文本将以特定的分辨率呈现，从而产生像素化效果。在调整窗口大小时，为防止 HTMLLoader 及其内容缩放，请将舞台的 scaleMode 属性设置为 StageScaleMode.NO_SCALE。

在 HTML 页中加载 SWF 或 PDF 内容时的注意事项

Adobe AIR 1.0 和更高版本

在以下情况下，加载到 HTMLLoader 对象中的 SWF 和 PDF 内容将消失：

- HTMLLoader 对象的缩放系数不为 1.0。
- 将 HTMLLoader 对象的 alpha 属性设置为 1.0 之外的值。
- 旋转 HTMLLoader 内容。

如果删除出错的属性设置并删除活动滤镜，可重新显示内容。

此外，运行时无法在透明窗口中显示 PDF 内容。如果将 object 或 embed 标签的 wmode 参数设置为 opaque 或 transparent，则运行时仅显示在 HTML 页中嵌入的 SWF 内容。因为 wmode 的默认值是 window，所以在透明窗口中不显示 SWF 内容，除非明确设置 wmode 参数。

注：在早于 AIR 1.5.2 的版本中，无论使用哪种 wmode 值，都不显示在 HTML 中嵌入的 SWF。

有关在 HTMLLoader 中加载这些类型的媒体的详细信息，请参阅第 849 页的 “ 在 HTML 中嵌入 SWF 内容 ” 和第 469 页的 “ 在 AIR 中添加 PDF 内容 ”。

高级显示属性

Adobe AIR 1.0 和更高版本

HTMLLoader 类继承了一些可用于生成特殊效果的方法。通常，这些效果在用于 HTMLLoader 显示时存在一些限制，但对于生成过渡或其他临时效果会非常有用。例如，如果显示一个对话窗口来收集用户输入内容，则可以在用户关闭对话之前模糊显示主窗口。同样，在关闭窗口时可以淡出显示。

高级显示属性包括：

属性	限制
alpha	会降低 HTML 内容的易读性
filters	在 HTML 窗口中，外部效果沿窗口边缘剪裁
graphics	使用图形命令绘制的形状显示在 HTML 内容下方 （包括默认背景）。paintsDefaultBackground 属性必须为 false，绘制的形状才可见。
opaqueBackground	不更改默认背景的颜色。paintsDefaultBackground 属性必须为 false，此颜色层才可见。
rotation	矩形 HTMLLoader 区域的各个角会沿窗口边缘剪裁。不显示 HTML 内容中加载的 SWF 和 PDF 内容。
scaleX、 scaleY	当缩放系数大于 1 时，会呈现像素化效果。不显示 HTML 内容中加载的 SWF 和 PDF 内容。
transform	会降低 HTML 内容的易读性。 HTML 显示会沿窗口边缘剪裁。如果转换涉及旋转、缩放或倾斜，则不显示 HTML 内容中加载的 SWF 和 PDF 内容。

下面的示例说明如何设置 filters 数组使整个 HTML 模糊显示：

```
 var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("http://www.adobe.com/");
html.load(urlReq);
html.width = 800;
html.height = 600;

var blur:BlurFilter = new BlurFilter(8);
var filters:Array = [blur];
html.filters = filters;
```

滚动 HTML 内容

Adobe AIR 1.0 和更高版本

HTMLLoader 类包含以下属性，通过这些属性可以控制 HTML 内容的滚动：

属性	说明
contentHeight	HTML 内容的高度 （以像素为单位）。
contentWidth	HTML 内容的宽度 （以像素为单位）。
scrollH	HTMLLoader 对象中的 HTML 内容的水平滚动位置。
scrollV	HTMLLoader 对象中的 HTML 内容的垂直滚动位置。

以下代码设置 scrollV 属性，以便 HTML 内容滚动到页面底部：

```
 var html:HTMLLoader = new HTMLLoader();
html.addEventListener(Event.HTML_BOUNDS_CHANGE, scrollHTML);

const SIZE:Number = 600;
html.width = SIZE;
html.height = SIZE;

var urlReq:URLRequest = new URLRequest("http://www.adobe.com");
html.load(urlReq);
this.addChild(html);

function scrollHTML(event:Event):void
{
    html.scrollV = html.contentHeight - SIZE;
}
```

HTMLLoader 不包含水平和垂直滚动条。可以使用 ActionScript 或 Flex 组件来实现滚动条。 Flex HTML 组件会自动为 HTML 内容附加滚动条。还可以使用 HTMLLoader.createRootWindow() 方法创建包含带有滚动条的 HTMLLoader 对象的窗口 （请参阅第 871 页的 “ 创建具有滚动 HTML 内容的窗口 ”）。

访问 HTML 历史记录列表

Adobe AIR 1.0 和更高版本

在 HTMLLoader 对象中加载新页时，运行时将为该对象维护一份历史记录列表。历史记录列表对应于 HTML 页中的 window.history 对象。 HTMLLoader 类包含以下属性和方法，可用于操作 HTML 历史记录列表：

类成员	说明
historyLength	历史记录列表的总长度，包括向后和向前的条目。
historyPosition	历史记录列表中的当前位置。位于此位置之前的历史记录项表示 “ 向后 ” 导航，位于此位置之后的项表示 “ 向前 ” 导航。
getHistoryAt()	返回与历史记录列表中指定位置的历史记录条目对应的 URLRequest 对象。
historyBack()	如果可能，在历史记录列表中向后导航。
historyForward()	如有可能，请在历史记录列表中向前导航。
historyGo()	在浏览器历史记录中按指示的步数导航。如果为正数，则向前导航；如果为负数，则向后导航。导航到零将重新加载页面。如果指定的位置超出末尾位置，则将导航到列表末尾。

历史记录列表中的项目作为 HTMLHistoryItem 类型的对象存储。 HTMLHistoryItem 类包含下列属性：

属性	说明
isPost	如果 HTML 页包括 POST 数据，则设置为 true。
originalUrl	在进行任何重定向之前， HTML 页的原始 URL。
title	HTML 页的标题。
url	HTML 页的 URL。

设置在加载 HTML 内容时使用的用户代理

Adobe AIR 1.0 和更高版本

HTMLLoader 类具有 userAgent 属性，通过该属性可以设置 HTMLLoader 使用的用户代理字符串。需在调用 load() 方法之前设置 HTMLLoader 对象的 userAgent 属性。如果对 HTMLLoader 实例设置此属性，则不 使用传递给 load() 方法的 URLRequest 的 userAgent 属性。

通过设置 URLRequestDefaults.userAgent 属性，可以设置应用程序域中所有 HTMLLoader 对象使用的默认用户代理字符串。URLRequestDefaults 静态属性作为默认属性应用于所有 URLRequest 对象，不只是与 HTMLLoader 对象的 load() 方法一起使用的 URLRequest 对象。设置 HTMLLoader 的 userAgent 属性将覆盖 URLRequestDefaults.userAgent 默认设置。

如果既未为 HTMLLoader 对象的 userAgent 属性设置用户代理值，也未为 URLRequestDefaults.userAgent 设置用户代理值，则将使用默认的 AIR 用户代理值。此默认值随着运行时操作系统（如 Mac OS 或 Windows）、运行时语言和运行时版本而变化，如下面两个示例所示：

- "Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"
- "Mozilla/5.0 (Windows; U; en) AppleWebKit/420+ (KHTML, like Gecko) AdobeAIR/1.0"

设置用于 HTML 内容的字符编码

Adobe AIR 1.0 和更高版本

HTML 页通过包括 meta 标签可以指定其使用的字符编码，如下所示：

```
meta http-equiv="content-type" content="text/html" charset="ISO-8859-1";
```

通过设置 HTMLLoader 对象的 textEncodingOverride 属性覆盖页面设置，确保使用特定的字符编码：

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingOverride = "ISO-8859-1";
```

使用 HTMLLoader 对象的 textEncodingFallback 属性，指定当 HTML 页未指定字符编码设置时要对 HTMLLoader 内容使用的字符编码：

```
var html:HTMLLoader = new HTMLLoader();
html.textEncodingFallback = "ISO-8859-1";
```

textEncodingOverride 属性将覆盖 HTML 页中的设置。并且 textEncodingOverride 属性和 HTML 页中的设置将覆盖 textEncodingFallback 属性。

需在加载 HTML 内容之前设置 textEncodingOverride 属性或 textEncodingFallback 属性。

为 HTML 内容定义类似于浏览器的用户界面

Adobe AIR 1.0 和更高版本

JavaScript 提供了多个 API 来控制显示 HTML 内容的窗口。在 AIR 中，可以通过实现自定义 HTMLHost 类覆盖这些 API。

关于扩展 HTMLHost 类

Adobe AIR 1.0 和更高版本

例如，如果您的应用程序在选项卡式界面中呈现多个 HTMLLoader 对象，可能希望根据所加载的 HTML 页，使用其标题来更改选项卡的标签，而不是主窗口的标题。同样，您的代码可通过以下方式响应 window.moveTo() 调用：在父显示对象容器中重新定位 HTMLLoader 对象、移动包含 HTMLLoader 对象的窗口、不执行任何操作或执行完全不相干的操作。

AIR HTMLHost 类控制以下 JavaScript 属性和方法：

- window.status
- window.document.title
- window.location
- window.blur()
- window.close()
- window.focus()
- window.moveBy()
- window.moveTo()
- window.open()
- window.resizeBy()
- window.resizeTo()

在使用 new HTMLLoader() 创建 HTMLLoader 对象时，不会启用所列的 JavaScript 属性或方法。HTMLHost 类提供了这些 JavaScript API 的类似于浏览器的默认实现。还可以扩展 HTMLHost 类以自定义行为。若要创建支持默认行为的 HTMLHost 对象，请在 HTMLHost 构造函数中将 defaultBehaviors 参数设置为 true：

```
var defaultHost:HTMLHost = new HTMLHost(true);
```

在 AIR 中，使用 HTMLLoader 类的 createRootWindow() 方法创建 HTML 窗口时，将自动分配支持默认行为的 HTMLHost 实例。可以通过向 HTMLLoader 的 htmlHost 属性分配不同的 HTMLHost 实现来更改主机对象行为，也可以分配 null 以禁用整个功能。

注：AIR 将默认的 HTMLHost 对象分配给为基于 HTML 的 AIR 应用程序创建的初始窗口以及使用 JavaScript 的 window.open() 方法的默认实现创建的所有窗口。

示例：扩展 HTMLHost 类

Adobe AIR 1.0 和更高版本

下面的示例说明如何通过扩展 HTMLHost 类来自定义 HTMLLoader 对象影响用户界面的方式：

Flex 示例：

1. 创建一个 HTMLHost 类的扩展类 （子类）。
2. 覆盖新类的方法以处理用户界面相关设置中的更改。例如，以下 CustomHost 类定义调用 window.open() 和更改 window.document.title 的行为。调用 window.open() 将在新窗口中打开 HTML 页，更改 window.document.title （包括 HTML 页的 <title> 元素的设置）将设置该窗口的标题。

```
package
{
    import flash.html.*;
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;

    public class CustomHost extends HTMLHost
    {
        import flash.html.*;
        override public function
            createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                                 windowCreateOptions.y,
                                                 windowCreateOptions.width,
                                                 windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                         windowCreateOptions.scrollBarsVisible, bounds);
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title;
        }
    }
}
```

3 在包含 HTMLLoader 的代码（不是新建的 HTMLHost 子类的代码）中，创建新类的对象。将新对象分配给 HTMLLoader 的 htmlHost 属性。以下 Flex 代码使用上一步中定义的 CustomHost 类：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    applicationComplete="init()">
    <mx:Script>
        <![CDATA[
            import flash.html.HTMLLoader;
            import CustomHost;
            private function init():void
            {
                var html:HTMLLoader = new HTMLLoader();
                html.width = container.width;
                html.height = container.height;
                var urlReq:URLRequest = new URLRequest("Test.html");
                html.htmlHost = new CustomHost();
                html.load(urlReq);
                container.addChild(html);
            }
        ]]>
    </mx:Script>
    <mx:UIComponent id="container" width="100%" height="100%"/>
</mx:WindowedApplication>
```

若要测试此处所述的代码，请将具有以下内容的 HTML 文件放在应用程序目录下：

```
<html>
    <head>
        <title>Test</title>
    </head>
    <script>
        function openWindow()
        {
            window.runtime.trace("in");
            document.title = "foo"
            window.open('Test.html');
            window.runtime.trace("out");
        }
    </script>
    <body>
        <a href="#" onclick="openWindow()">window.open('Test.html')</a>
    </body>
</html>
```

Flash Professional 示例：

1 为 AIR 创建一个 Flash 文件。将其文档类设置为 CustomHostExample，然后将文件另存为 CustomHostExample.fla。

2 创建一个名为 CustomHost.as 的 ActionScript 文件，该文件包含一个 HTMLHost 类的扩展类 （子类）。此类将覆盖新类的某些方法，以处理用户界面相关设置中的更改。例如，以下 CustomHost 类定义调用 window.open() 和更改 window.document.title 的行为。调用 window.open() 方法将在新窗口中打开 HTML 页，更改 window.document.title 属性（包括 HTML 页的 <title> 元素的设置）将设置该窗口的标题。

```
package
{
    import flash.display.StageScaleMode;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.events.Event;
    import flash.events.NativeWindowBoundsEvent;
    import flash.geom.Rectangle;
    import flash.html.HTMLLoader;
    import flash.html.HTMLHost;
    import flash.html.HTMLWindowCreateOptions;
    import flash.text.TextField;

    public class CustomHost extends HTMLHost
    {
        public var statusField:TextField;

        public function CustomHost(defaultBehaviors:Boolean=true)
        {
            super(defaultBehaviors);
        }
        override public function windowClose():void
        {
            htmlLoader.stage.nativeWindow.close();
        }
        override public function createWindow(
                                windowCreateOptions:HTMLWindowCreateOptions ):HTMLLoader
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(windowCreateOptions.x,
                                        windowCreateOptions.y,
                                        windowCreateOptions.width,
                                        windowCreateOptions.height);
            var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                    windowCreateOptions.scrollBarsVisible, bounds);
```

```
            htmlControl.htmlHost = new HTMLHostImplementation();
            if(windowCreateOptions.fullscreen){
                htmlControl.stage.displayState =
                    StageDisplayState.FULL_SCREEN_INTERACTIVE;
            }
            return htmlControl;
        }
        override public function updateLocation(locationURL:String):void
        {
            trace(locationURL);
        }
        override public function set windowRect(value:Rectangle):void
        {
            htmlLoader.stage.nativeWindow.bounds = value;
        }
        override public function updateStatus(status:String):void
        {
            statusField.text = status;
            trace(status);
        }
        override public function updateTitle(title:String):void
        {
            htmlLoader.stage.nativeWindow.title = title + "- Example Application";
        }
        override public function windowBlur():void
        {
            htmlLoader.alpha = 0.5;
        }
        override public function windowFocus():void
        {
            htmlLoader.alpha = 1;
        }
    }
}
```

3 创建另一个名为 CustomHostExample.as 的 ActionScript 文件，以包含应用程序的文档类。此类将创建一个 HTMLLoader 对象，并将其主机属性设置为上一步中定义的 CustomHost 类的一个实例：

```
package
{
    import flash.display.Sprite;
    import flash.html.HTMLLoader;
    import flash.net.URLRequest;
    import flash.text.TextField;

    public class CustomHostExample extends Sprite
    {
        function CustomHostExample():void
        {
            var html:HTMLLoader = new HTMLLoader();
            html.width = 550;
            html.height = 380;
            var host:CustomHost = new CustomHost();
            html.htmlHost = host;

            var urlReq:URLRequest = new URLRequest("Test.html");
            html.load(urlReq);

            addChild(html);

            var statusTxt:TextField = new TextField();
            statusTxt.y = 380;
            statusTxt.height = 20;
            statusTxt.width = 550;
            statusTxt.background = true;
            statusTxt.backgroundColor = 0xEEEEEEEE;
            addChild(statusTxt);

            host.statusField = statusTxt;
        }
    }
}
```

若要测试此处所述的代码，请将具有以下内容的 HTML 文件放在应用程序目录下：

```
<html>
     <head>
     <title>Test</title>
     <script>
     function openWindow()
     {
     document.title = "Test"
     window.open('Test.html');
     }
     </script>
     </head>
     <body bgColor="#EEEEEE">
     <a href="#" onclick="window.open('Test.html')">window.open('Test.html')</a>
     <br/><a href="#" onclick="window.document.location='http://www.adobe.com'">
     window.document.location = 'http://www.adobe.com'</a>
     <br/><a href="#" onclick="window.moveBy(6, 12)">moveBy(6, 12)</a>
     <br/><a href="#" onclick="window.close()">window.close()</a>
     <br/><a href="#" onclick="window.blur()">window.blur()</a>
     <br/><a href="#" onclick="window.focus()">window.focus()</a>
     <br/><a href="#" onclick="window.status = new Date().toString()">window.status=new
Date().toString()</a>
     </body>
</html>
```

处理对 window.location 属性的更改

Adobe AIR 1.0 和更高版本

覆盖 locationChange() 方法以处理对 HTML 页的 URL 的更改。当某页中的 JavaScript 更改了 window.location 的值时，将调用 locationChange() 方法。以下示例仅加载了请求的 URL：

```
override public function updateLocation(locationURL:String):void
{
    htmlLoader.load(new URLRequest(locationURL));
}
```

注：可以使用 HTMLHost 对象的 htmlLoader 属性来引用当前的 HTMLLoader 对象。

处理对 window.moveBy()、 window.moveTo()、 window.resizeTo()、window.resizeBy() 的 JavaScript 调用

Adobe AIR 1.0 和更高版本

覆盖 set windowRect() 方法以处理 HTML 内容范围的更改。当某页中的 JavaScript 调用 window.moveBy()、window.moveTo()、 window.resizeTo() 或 window.resizeBy() 时，将调用 set windowRect() 方法。以下示例仅更新了桌面窗口范围：

```
override public function set windowRect(value:Rectangle):void
{
    htmlLoader.stage.nativeWindow.bounds = value;
}
```

处理对 window.open() 的 JavaScript 调用

Adobe AIR 1.0 和更高版本

覆盖 createWindow() 方法以处理对 window.open() 的 JavaScript 调用。 createWindow() 方法的实现负责创建和返回新的 HTMLLoader 对象。通常，将在新窗口中显示 HTMLLoader，但不需要创建一个窗口。

以下示例说明如何通过使用 HTMLLoader.createRootWindow() 创建窗口和 HTMLLoader 对象来实现 createWindow() 函数。还可以单独创建一个 NativeWindow 对象，然后将 HTMLLoader 添加到窗口舞台。

```
override public function createWindow(windowCreateOptions:HTMLWindowCreateOptions):HTMLLoader{
    var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
    var bounds:Rectangle = new Rectangle(windowCreateOptions.x, windowCreateOptions.y,
                                 windowCreateOptions.width, windowCreateOptions.height);
    var htmlControl:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions,
                                     windowCreateOptions.scrollBarsVisible, bounds);
    htmlControl.htmlHost = new HTMLHostImplementation();
    if(windowCreateOptions.fullscreen){
        htmlControl.stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
    }
    return htmlControl;
}
```

注：本示例将自定义 HTMLHost 实现分配给使用 window.open() 创建的所有新窗口。如果需要，还可以对新窗口使用不同的实现或将 htmlHost 属性设置为 null。

作为参数传递到 createWindow() 方法的对象是一个 HTMLWindowCreateOptions 对象。HTMLWindowCreateOptions 类包含相关属性，可报告在对 window.open() 的调用中， features 参数字符串中设置的值：

HTMLWindowCreateOptions 属性	在对 window.open() 的 JavaScript 调用中， features 字符串中的相应设置
fullscreen	fullscreen
height	height
locationBarVisible	location
menuBarVisible	menubar
resizeable	resizable
scrollBarsVisible	scrollbars
statusBarVisible	status
toolBarVisible	toolbar
width	width
x	left 或 screenX
y	top 或 screenY

HTMLLoader 类并不会实现可在 feature 字符串中指定的所有功能。您的应用程序必须在适当的时候提供滚动条、位置栏、菜单栏、状态栏和工具栏。

JavaScript window.open() 方法的其他参数由系统处理。 createWindow() 实现不应在 HTMLLoader 对象中加载内容或设置窗口标题。

处理对 window.close() 的 JavaScript 调用

Adobe AIR 1.0 和更高版本

覆盖 windowClose() 以处理对 window.close() 方法的 JavaScript 调用。以下示例在调用 window.close() 方法时将关闭桌面窗口。

```
override public function windowClose():void
{
    htmlLoader.stage.nativeWindow.close();
}
```

对 window.close() 的 JavaScript 调用不必关闭包含窗口。例如，可以从显示列表中删除 HTMLLoader，保持窗口 （可能包含其他内容）处于打开状态，如以下代码所示：

```
override public function windowClose():void
{
    htmlLoader.parent.removeChild(htmlLoader);
}
```

处理对 window.status 属性的更改

Adobe AIR 1.0 和更高版本

覆盖 updateStatus() 方法以处理对 window.status 值的 JavaScript 更改。以下示例跟踪状态值：

```
 override public function updateStatus(status:String):void
{
    trace(status);
}
```

请求的状态作为字符串传递给 updateStatus() 方法。

HTMLLoader 对象不提供状态栏。

处理对 window.document.title 属性的更改

Adobe AIR 1.0 和更高版本

覆盖 updateTitle() 方法以处理对 window.document.title 值的 JavaScript 更改。以下示例更改窗口标题并向标题追加 “Sample” 字符串：

```
 override public function updateTitle(title:String):void
{
    htmlLoader.stage.nativeWindow.title = title + " - Sample";
}
```

在 HTML 页上设置 document.title 时，请求的标题将作为字符串传递给 updateTitle() 方法。

更改 document.title 时不必更改包含 HTMLLoader 对象的窗口的标题。可以更改其他界面元素，如文本字段。

处理对 window.blur() 和 window.focus() 的 JavaScript 调用

Adobe AIR 1.0 和更高版本

覆盖 windowBlur() 和 windowFocus() 方法以处理对 window.blur() 和 window.focus() 的 JavaScript 调用，如下例所示：

```
 override public function windowBlur():void
{
    htmlLoader.alpha = 0.5;
}
override public function windowFocus():void
{
    htmlLoader.alpha = 1.0;
    NativeApplication.nativeApplication.activate(htmlLoader.stage.nativeWindow);
}
```

注：AIR 不提供用于取消激活窗口或应用程序的 API。

创建具有滚动 HTML 内容的窗口

Adobe AIR 1.0 和更高版本

HTMLLoader 类包含一个静态方法 HTMLLoader.createRootWindow()，使用该方法，可以打开一个包含 HTMLLoader 对象的新窗口 （由 NativeWindow 对象表示）并为该窗口定义一些用户界面设置。该方法采用四个参数，可以通过这些参数来定义用户界面：

参数	说明
visible	一个布尔值，它指定窗口最初是 (true) 否 (false) 可见。
windowInitOptions	一个 NativeWindowInitOptions 对象。 NativeWindowInitOptions 类为 NativeWindow 对象定义初始化选项，包括以下内容：窗口是否可最小化、可最大化或可调整大小，窗口是否有系统镶边或自定义镶边，窗口是否透明 （对于不使用系统镶边的窗口）以及窗口的类型。
scrollBarsVisible	是 (true) 否 (false) 有滚动条。
bounds	一个用于定义新窗口的位置和大小的 Rectangle 对象。

例如，以下代码使用 HTMLLoader.createRootWindow() 方法创建带有使用滚动条的 HTMLLoader 内容的窗口：

```
var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
var bounds:Rectangle = new Rectangle(10, 10, 600, 400);
var html2:HTMLLoader = HTMLLoader.createRootWindow(true, initOptions, true, bounds);
var urlReq2:URLRequest = new URLRequest("http://www.example.com");
html2.load(urlReq2);
html2.stage.nativeWindow.activate();
```

注：通过直接在 JavaScript 中调用 createRootWindow() 创建的窗口将独立于打开的 HTML 窗口。例如，JavaScript Window opener 和 parent 的属性为 null。不过，如果通过覆盖 HTMLHost createWindow() 方法间接地调用 createRootWindow()，opener 和 parent 将引用打开的 HTML 窗口。

创建 HTMLLoader 类的子类

Adobe AIR 1.0 和更高版本

可以通过创建 HTMLLoader 类的子类来创建新行为。例如，可以创建一个用于定义 HTMLLoader 事件 （例如在呈现 HTML 或用户单击链接时调度的那些事件）的默认事件侦听器的子类。

以下示例在调用 JavaScript window.open() 方法时，通过扩展 HTMLHost 类来提供正常 行为。然后，本示例定义使用自定义 HTMLHost 实现类的 HTMLLoader 的一个子类：

```
package
{
       import flash.html.HTMLLoader;
    public class MyHTMLHost extends HTMLHost
       {
           public function MyHTMLHost()
           {
               super(false);
        }
        override public function createWindow(opts:HTMLWindowCreateOptions):void
        {
            var initOptions:NativeWindowInitOptions = new NativeWindowInitOptions();
            var bounds:Rectangle = new Rectangle(opts.x, opts.y, opts.width, opts.height);
            var html:HTMLLoader = HTMLLoader.createRootWindow(true,
                                                    initOptions,
                                                    opts.scrollBarsVisible,
                                                    bounds);
            html.stage.nativeWindow.orderToFront();
            return html
       }
}
```

下面定义 HTMLLoader 类的一个子类，该子类将一个 MyHTMLHost 对象分配给其 htmlHost 属性：

```
package
{
        import flash.html.HTMLLoader;
    import MyHTMLHost;
    import HTMLLoader;
       public class MyHTML extends HTMLLoader
       {
           public function MyHTML()
           {
               super();
               htmlHost = new MyHTMLHost();
           }
       }
}
```

有关本示例中使用的 HTMLHost 类和 HTMLLoader.createRootWindow() 方法的详细信息，请参阅第 863 页的 “ 为 HTML 内容定义类似于浏览器的用户界面 ”。

第 61 章：处理 AIR 中与 HTML 相关的事件

Adobe AIR 1.0 和更高版本

利用事件处理系统，程序员可以十分方便地响应用户输入和系统事件。 Adobe® AIR® 事件模型不仅方便，而且符合标准。事件模型基于文档对象模型 (DOM) 第 3 级事件规范，是业界标准的事件处理体系结构，为程序员提供了强大而直观的事件处理工具。

HTMLLoader 事件

Adobe AIR 1.0 和更高版本

HTMLLoader 对象调度以下 Adobe® ActionScript® 3.0 事件：

事件	说明
htmlDOMInitialize	在创建 HTML 文档时调度，调度时未分析任何脚本或未将 DOM 节点添加到页面。
complete	在为响应加载操作而创建 HTML DOM 后，紧接在 HTML 页面中的 onload 事件后调度。
htmlBoundsChanged	在 contentWidth 和 / 或 contentHeight 属性发生了变化时调度。
locationChange	在 HTMLLoader 的 location 属性发生了变化时调度。
locationChanging	由于用户导航、 JavaScript 调用或重定向，在 HTMLLoader 的位置发生变化前调度。当您调用 load()、loadString()、reload()、historyGo()、historyForward() 或 historyBack() 方法时未调度 locationChanging 事件。 调用所调度事件对象的 preventDefault() 方法会取消导航。 如果系统浏览器中打开某个链接，则不会调度 locationChanging 事件，因为 HTMLLoader 不会改变位置。
scroll	只要 HTML 引擎更改滚动位置，便会调度此事件。 Scroll 事件的引发可能是由于导航到页面中的锚记链接 （# 链接），也可能是由于调用 window.scrollTo() 方法。在文本输入或文本区域中输入文本也可能会引发 scroll 事件。
uncaughtScriptException	当在 HTMLLoader 中发生 JavaScript 异常，并且在 JavaScript 代码中未捕获到该异常时调度。

您也可以为 JavaScript 事件 （如 onClick）注册 ActionScript 函数。有关详细信息，请参阅第 874 页的 “ 使用 ActionScript 处理 DOM 事件 ”。

使用 ActionScript 处理 DOM 事件

Adobe AIR 1.0 和更高版本

可以注册 ActionScript 函数以响应 JavaScript 事件。以下面的 HTML 内容为例：

```
<html>
<body>
    <a href="#" id="testLink">Click me.</a>
</html>
```

您可以将 ActionScript 函数注册为页面中任何事件的处理函数。例如，下面的代码添加 clickHandler() 函数，将其作为 HTML 页面中 testLink 元素的 onclick 事件的侦听器：

```
var html:HTMLLoader = new HTMLLoader( );
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event):void {
    html.window.document.getElementById("testLink").onclick = clickHandler;
}

function clickHandler( event:Object ):void {
    trace("Event of type: " + event.type );
}
```

调度的事件对象不属于类型 flash.events.Event，也不是 Event 的一个子类。使用 Object 类为事件处理函数参数声明一个类型。

您也可以使用 addEventListener() 方法来注册这些事件。例如，您可以将上一示例中的 completeHandler() 方法替换为下面的代码：

```
function completeHandler(event:Event):void {
    var testLink:Object = html.window.document.getElementById("testLink");
    testLink.addEventListener("click", clickHandler);
}
```

当侦听器引用特定的 DOM 元素时，先等父级 HTMLLoader 调度 complete 事件之后再添加事件侦听器是个好习惯。HTML 页面通常会加载多个文件，HTML DOM 的构建在所有文件的加载和分析完成以前不会全部完成。HTMLLoader 会在所有元素都创建完毕后调度 complete 事件。

响应未捕获的 JavaScript 异常

Adobe AIR 1.0 和更高版本

以下面的 HTML 为例：

```
<html>
<head>
    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

它包含 JavaScript 函数 throwError()，该函数引用一个未知变量 melbaToast：

```
var x = 400 * melbaToast;
```

如果 JavaScript 操作遇到一个非法操作，这个非法操作在带有 try/catch 结构的 JavaScript 代码中没有捕获到，则包含相应页面的 HTMLLoader 对象就会调度 HTMLUncaughtScriptExceptionEvent 事件。您可以注册此事件的处理函数，如下面的代码所示：

```
var html:HTMLLoader = new HTMLLoader();
var urlReq:URLRequest = new URLRequest("test.html");
html.load(urlReq);
html.width = container.width;
html.height = container.height;
container.addChild(html);
html.addEventListener(HTMLUncaughtScriptExceptionEvent.UNCAUGHT_SCRIPT_EXCEPTION,
                          htmlErrorHandler);
function htmlErrorHandler(event:HTMLUncaughtJavaScriptExceptionEvent):void
{
    event.preventDefault();
    trace("exceptionValue:", event.exceptionValue)
    for (var i:int = 0; i < event.stackTrace.length; i++)
    {
        trace("sourceURL:", event.stackTrace[i].sourceURL);
        trace("line:", event.stackTrace[i].line);
        trace("function:", event.stackTrace[i].functionName);
    }
}
```

在 JavaScript 中，您可以使用 window.htmlLoader 属性处理同一事件：

```
<html>
<head>
<script language="javascript" type="text/javascript" src="AIRAliases.js"></script>

    <script>
        function throwError() {
            var x = 400 * melbaToast;
        }

        function htmlErrorHandler(event) {
            event.preventDefault();
            var message = "exceptionValue:" + event.exceptionValue + "\n";
            for (var i = 0; i < event.stackTrace.length; i++){
                message += "sourceURL:" + event.stackTrace[i].sourceURL +"\n";
                message += "line:" + event.stackTrace[i].line +"\n";
                message += "function:" + event.stackTrace[i].functionName + "\n";
            }
            alert(message);
        }

        window.htmlLoader.addEventListener("uncaughtScriptException", htmlErrorHandler);
    </script>
</head>
<body>
    <a href="#" onclick="throwError()">Click me.</a>
</html>
```

htmlErrorHandler() 事件处理函数取消事件的默认行为 （即将 JavaScript 错误消息发送到 AIR trace 输出），然后生成自己的输出消息。它输出 HTMLUncaughtScriptExceptionEvent 对象的 exceptionValue 的值。它输出 stackTrace 数组中每个对象的属性：

```
 exceptionValue: ReferenceError: Can't find variable: melbaToast
sourceURL: app:/test.html
line: 5
function: throwError
sourceURL: app:/test.html
line: 10
function: onclick
```

使用 JavaScript 处理运行时事件

Adobe AIR 1.0 和更高版本

运行时类支持使用 addEventListener() 方法添加事件处理函数。若要为某个事件添加处理函数，请调用调度该事件的对象的 addEventListener() 方法，同时提供事件类型和处理函数。例如，若要侦听用户单击标题栏上的窗口关闭按钮时调度的 closing 事件，请使用下面的语句：

```
window.nativeWindow.addEventListener(air.NativeWindow.CLOSING, handleWindowClosing);
```

创建事件处理函数

Adobe AIR 1.0 和更高版本

下面的代码创建一个简单的 HTML 文件，用于显示有关主窗口位置的信息。一个名为 moveHandler() 的处理函数侦听主窗口的 move 事件 （由 NativeWindowBoundsEvent 类定义）。

```
<html>
    <script src="AIRAliases.js" />
    <script>
        function init() {
            writeValues();
            window.nativeWindow.addEventListener(air.NativeWindowBoundsEvent.MOVE,
                                                 moveHandler);
        }
        function writeValues() {
            document.getElementById("xText").value = window.nativeWindow.x;
            document.getElementById("yText").value = window.nativeWindow.y;
        }
        function moveHandler(event) {
            air.trace(event.type); // move
            writeValues();
        }
    </script>
    <body onload="init()" />
        <table>
            <tr>
                <td>Window X:</td>
                <td><textarea id="xText"></textarea></td>
            </tr>
            <tr>
                <td>Window Y:</td>
                <td><textarea id="yText"></textarea></td>
            </tr>
        </table>
    </body>
</html>
```

当用户移动此窗口时， textarea 元素会显示此窗口的更新的 X 位置和 Y 位置：

请注意，事件对象作为实参传递给 moveHandler() 方法。利用 event 参数，处理函数可以检查事件对象。在此示例中，使用事件对象的 type 属性报告该事件为 move 事件。

删除事件侦听器

Adobe AIR 1.0 和更高版本

可以使用 removeEventListener() 方法删除不再需要的事件侦听器。建议删除将不再使用的所有侦听器。必需的参数包括 eventName 和 listener 参数，这些参数与 addEventListener() 方法的必需参数相同。

删除执行导航的 HTML 页面中的事件侦听器

Adobe AIR 1.0 和更高版本

当 HTML 内容进行导航时，或者因包含 HTML 内容的窗口关闭而丢弃这些 HTML 内容时，不会自动删除引用已卸载的页面中对象的事件侦听器。当对象向已卸载的处理函数调度事件时，会显示下面的错误消息：“ 应用程序尝试引用不再处于已加载状态的 HTML 页面中的 JavaScript 对象。”(The application attempted to reference a JavaScript object in an HTML page that is no longer loaded.)

为避免出现此错误，请在 HTML 页面退出之前删除其中的 JavaScript 事件侦听器。如果发生页面导航（在 HTMLLoader 对象中），请在 window 对象的 unload 事件发生期间删除事件侦听器。

例如，下面的 JavaScript 代码删除 uncaughtScriptException 事件的事件侦听器：

```
window.onunload = cleanup;
window.htmlLoader.addEventListener('uncaughtScriptException', uncaughtScriptException);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                              uncaughtScriptExceptionHandler);
}
```

为避免在关闭包含 HTML 内容的窗口时发生错误，请调用 cleanup 函数以响应 NativeWindow 对象 (window.nativeWindow) 的 closing 事件。例如，下面的 JavaScript 代码删除 uncaughtScriptException 事件的事件侦听器：

```
 window.nativeWindow.addEventListener(air.Event.CLOSING, cleanup);
function cleanup()
{
    window.htmlLoader.removeEventListener('uncaughtScriptException',
                              uncaughtScriptExceptionHandler);
}
```

为了防止发生此错误，您还可以在事件侦听器运行时将其删除 （如果该事件只需要处理一次）。例如，下面的 JavaScript 代码通过调用 HTMLLoader 类的 createRootWindow() 方法创建一个 html 窗口，并为 complete 事件添加一个事件侦听器。在调用 complete 事件处理函数时，它会使用 removeEventListener() 函数删除它自己的事件侦听器：

```
 var html = runtime.flash.html.HTMLLoader.createRootWindow(true);
html.addEventListener('complete', htmlCompleteListener);
function htmlCompleteListener()
{
    html.removeEventListener(complete, arguments.callee)
    // handler code..
}
html.load(new runtime.flash.net.URLRequest("second.html"));
```

如果删除不需要的事件侦听器，则还会使系统垃圾回收器能回收与这些侦听器关联的任何内存空间。

检查有无现有的事件侦听器

Adobe AIR 1.0 和更高版本

hasEventListener() 方法用于检查某个对象是否存在事件侦听器。

第 62 章：在移动应用程序中显示 HTML 内容

Adobe AIR 2.5 和更高版本

StageWebView 类在移动设备上使用系统浏览器控件显示 HTML 内容，在桌面计算机上使用标准 Adobe® AIR® HTMLLoader 控件显示 HTML 内容。检查 StageWebView.isSupported 属性以确定当前设备是否支持该类。不保证在移动配置文件中为所有设备提供支持。

在所有配置文件中，StageWebView 类仅支持 HTML 内容和应用程序其他内容之间的有限交互。您可以控制导航，但不允许脚本交叉或直接交换数据。您可以从本地或远程 URL 加载内容，或者传入 HTML 字符串。

StageWebView 对象

StageWebView 对象不是显示对象，无法添加到显示列表。相反，它作为直接附加到舞台的视口工作。 StageWebView 内容在任何显示列表内容的前面绘制。无法控制多个 StageWebView 对象的绘制顺序。

要显示 StageWebView 对象，您对将显示此对象的舞台赋予 StageWebView 的 Stage 属性。使用 viewPort 属性设置显示的大小。

将 viewPort 属性的 x 和 y 坐标设置为介于 -8192 至 8191 之间的值。舞台宽度和高度的最大值为 8191。如果大小超出了最大值，将引发异常。

以下示例创建一个 StageWebView 对象，设置 Stage 和 viewPort 属性，并显示 HTML 字符串：

```
var webView:StageWebView = new StageWebView();
webView.viewPort = new Rectangle( 0, 0, this.stage.stageWidth, this .stage.stageHeight);
webView.stage = this.stage;
var htmlString:String = "<!DOCTYPE HTML>" +
                        "<html><body>" +
                        "<p>King Philip could order five good steaks.</p>" +
                        "</body></html>";
webView.loadString( htmlString );
```

要隐藏 StageWebView 对象，请将其 stage 属性设置为 null。要彻底销毁对象，请调用 dispose() 方法。调用 dispose() 是可选的，但调用它有助于垃圾回收器尽快回收对象使用的内存。

内容

您可以使用两个方法将内容加载到 StageWebView 对象：loadURL() 和 loadString()。

loadURL() 方法在指定的 URL 上加载资源。您可以使用系统 Web 浏览器控件支持的任何 URI 方案，包括 data:、 file:、 http:、 https: 和 JavaScript:。不支持 app: 和 app-storage: 方案。 AIR 不验证 URL 字符串。

loadString() 方法加载包含 HTML 内容的文本字符串。使用此方法加载的页面的位置表示为：

- 在桌面操作系统上：about:blank
- 在 iOS 上：htmlString
- 在 Android 上：编码的 htmlString 的数据 URI 格式

URI 方案确定加载嵌入内容或数据的规则。

URI 方案	加载本地资源	加载远程资源	本地 XMLHttpRequest	远程 XMLHttpRequest
数据:	否	是	否	否
文件:	是	是	是	是
http:，https:	否	是	否	同一域
关于:（loadString() 方法）	否	是	否	否

注：如果舞台的 displayState 属性设置为 FULL_SCREEN，则在桌面操作系统中，您不能在 StageWebView 中显示的文本字段中键入内容。但是，在 iOS 和 Android 中，即使舞台的 displayState 设置为 FULL_SCREEN，您也可以在 StageWebView 上的文本字段中键入内容。

下面的示例使用 StageWebView 对象显示 Adobe 的网站：

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );
        }
    }
}
```

在 Android 设备上，您必须指定 Android INTERNET 权限才能使应用程序成功加载远程资源。

在 Android 3.0 以上的版本中，应用程序必须在 AIR 应用程序描述符的 Android manifestAdditions 元素中启用硬件加速，才能在 StageWebView 对象中显示插件内容。请参阅在 StageWebView 对象中启用 Flash Player 和其他插件。

JavaScript URI

可以使用 JavaScript URI 来调用在 HTML 页中定义并由 StageWebView 对象加载的函数。使用 JavaScript URI 调用的函数将在所加载网页的上下文中运行。以下示例使用 StageWebView 对象调用一个 JavaScript 函数：

```
package {
    import flash.display.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    public class WebView extends Sprite
    {
        public var webView:StageWebView = new StageWebView();
        public function WebView()
        {
            var htmlString:String = "<!DOCTYPE HTML>" +
            "<html><script type=text/javascript>" +
            "function callURI(){" +
            "alert(\"You clicked me!!\");"+
            "}</script><body>" +
            "<p><a href=javascript:callURI()>Click Me</a></p>" +
            "</body></html>";
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadString( htmlString );
        }
    }
}
```

更多帮助主题

Sean Voisen：制作大部分 StageWebView

导航事件

当用户单击 HTML 中的链接时， StageWebView 对象调度一个 locationChanging 事件。您可以调用事件对象的 preventDefault() 方法停止导航。否则， StageWebView 对象将导航到新页面并调度一个 locationChange 事件。当页面加载完成时， StageWebView 调度一个 complete 事件。

locationChanging 事件会在每次 HTML 重定向时调度。在适当的时间调度 locationChange 和 complete 事件。

在 iOS 上，除了首个 loadURL() 或 loadString() 方法以外， locationChanging 事件会在 locationChange 事件之前调度。通过 iFrame 和 Frame 进行导航更改时，也会调度 locationChange 事件。

以下示例显示如何防止位置更改，并在系统浏览器中打开新页面。

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.events.LocationChangeEvent;
    import flash.geom.Rectangle;
    import flash.net.navigateToURL;
    import flash.net.URLRequest;

    public class StageWebViewNavEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();

        public function StageWebViewNavEvents() {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
        webView.addEventListener( LocationChangeEvent.LOCATION_CHANGING, onLocationChanging );
            webView.loadURL( "http://www.adobe.com" );
        }
        private function onLocationChanging( event:LocationChangeEvent ):void
        {
            event.preventDefault();
            navigateToURL( new URLRequest( event.location ) );
        }
    }
}
```

历史记录

当用户单击在 StageWebView 对象中显示的内容中的链接时，控件会保存向前和向后的历史记录堆栈。以下示例显示如何通过二个历史记录堆栈导航。示例使用 “ 后退 ”(Back) 和 “ 搜索 ”(Search) 软键。

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;

    public class StageWebViewExample extends MovieClip{

        var webView:StageWebView = new StageWebView();

        public function StageWebViewExample()
        {
            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 0, stage.stageWidth, stage.stageHeight );
            webView.loadURL( "http://www.adobe.com" );

            stage.addEventListener( KeyboardEvent.KEY_DOWN, onKey );
        }

        private function onKey( event:KeyboardEvent ):void
        {
            if( event.keyCode == Keyboard.BACK && webView.isHistoryBackEnabled )
            {
                trace("back");
                webView.historyBack();
                event.preventDefault();
            }
            if( event.keyCode == Keyboard.SEARCH && webView.isHistoryForwardEnabled )
            {
                trace("forward");
                webView.historyForward();
            }
        }
    }
}
```

焦点

尽管 StageWebView 类不是显示对象，但它包含的成员允许您管理控件的焦点进出。

当 StageWebView 对象获得焦点时，它调度一个 focusIn 事件。您使用此事件管理应用程序中的焦点元素 （如果需要）。

当 StageWebView 放弃焦点时，它调度一个 focusOut 事件。当用户使用设备轨迹球或方向箭头以 Tab 键方式向上通过页面上的第一个或向下通过最后一个控件时， StageWebView 实例会放弃焦点。事件对象的 direction 属性告诉您焦点移动方向是向上通过页面的顶部还是向下通过页面的底部。使用此信息将焦点赋予 StageWebView 上方或下方适当的显示对象上。

在 iOS 上，不能通过编程方式设置焦点。 StageWebView 会调度 focusIn 和 focusOut 事件，并将 FocusEvent 的 direction 属性设置为 none。如果用户触摸 StageWebView 内部，将调度 focusIn 事件。如果用户触摸 StageWebView 外部，将调度 focusOut 事件。

以下示例显示焦点如何从 StageWebView 对象传递到 Flash 显示对象：

```
package  {
    import flash.display.MovieClip;
    import flash.media.StageWebView;
    import flash.geom.Rectangle;
    import flash.events.KeyboardEvent;
    import flash.ui.Keyboard;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.FocusEvent;
    import flash.display.FocusDirection;
    import flash.events.LocationChangeEvent;

    public class StageWebViewFocusEvents extends MovieClip{
        var webView:StageWebView = new StageWebView();
        var topControl:TextField = new TextField();
        var bottomControl:TextField = new TextField();

        public function StageWebViewFocusEvents()
        {
            trace("Starting");
            topControl.type = TextFieldType.INPUT;
            addChild( topControl );
            topControl.height = 60;
            topControl.width = stage.stageWidth;
            topControl.background = true;
            topControl.text = "One control on top.";
            topControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            topControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );

            webView.stage = this.stage;
            webView.viewPort = new Rectangle( 0, 60, stage.stageWidth, stage.stageHeight
- 120 );
            webView.addEventListener( FocusEvent.FOCUS_IN, webFocusIn );
            webView.addEventListener(FocusEvent.FOCUS_OUT, webFocusOut );
            webView.addEventListener(LocationChangeEvent.LOCATION_CHANGING,
                                     function( event:LocationChangeEvent ):void
                                     {
                                        event.preventDefault();
                                     } );
            webView.loadString("<form action='#'><input/><input/><input/></form>");
            webView.assignFocus();

            bottomControl.type = TextFieldType.INPUT;
            addChild( bottomControl );
            bottomControl.y = stage.stageHeight - 60;
            bottomControl.height = 60;
            bottomControl.width = stage.stageWidth;
            bottomControl.background = true;
            bottomControl.text = "One control on the bottom.";
            bottomControl.addEventListener( FocusEvent.FOCUS_IN, flashFocusIn );
            bottomControl.addEventListener( FocusEvent.FOCUS_OUT, flashFocusOut );}

        private function webFocusIn( event:FocusEvent ):void
        {
            trace("Web focus in");
        }

        private function webFocusOut( event:FocusEvent ):void
        {
            trace("Web focus out: " + event.direction);
            if( event.direction == FocusDirection.TOP )
            {
```

```
                stage.focus = topControl;
            }
            else
            {
                stage.focus = bottomControl;
            }
        }

        private function flashFocusIn( event:FocusEvent ):void
        {
            trace("Flash focus in");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xff5566;
        }

        private function flashFocusOut( event:FocusEvent ):void
        {
            trace("Flash focus out");
            var textfield:TextField = event.target as TextField;
            textfield.backgroundColor = 0xffffff;
        }

    }
}
```

位图捕获

StageWebView 对象会在所有显示列表内容之上呈现。您不能在 StageWebView 对象之上添加内容。例如，您不能在 StageWebView 之上展开一个下拉列表。要解决此问题，请捕获 StageWebView 的快照。然后，隐藏 StageWebView 并添加替代的位图快照。

以下示例演示了如何使用 drawViewPortToBitmapData 方法捕获 StageWebView 对象的快照。该示例通过将舞台设置为 null 来隐藏 StageWebView 对象。完全加载网页后，该示例调用了一个捕获并显示位图的函数。运行时，此代码会显示两个标签：Google 和 Facebook。点击标签将捕获相应的网页，并将其在舞台上作为快照显示。

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.geom.Rectangle;
    import flash.media.StageWebView;
    import flash.net.*;
    import flash.text.TextField;
    public class stagewebview extends Sprite
    {
        public var webView:StageWebView=new StageWebView();
        public var textGoogle:TextField=new TextField();
        public var textFacebook:TextField=new TextField();
        public function stagewebview()
        {
            textGoogle.htmlText="<b>Google</b>";
            textGoogle.x=300;
            textGoogle.y=-80;
            addChild(textGoogle);
            textFacebook.htmlText="<b>Facebook</b>";
            textFacebook.x=0;
            textFacebook.y=-80;
```

```
            addChild(textFacebook);
            textGoogle.addEventListener(MouseEvent.CLICK,goGoogle);
            textFacebook.addEventListener(MouseEvent.CLICK,goFaceBook);
            webView.stage = this.stage;
            webView.viewPort = new Rectangle(0, 0, stage.stageWidth, stage.stageHeight);
        }
    public function goGoogle(e:Event):void
        {
            webView.loadURL("http://www.google.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }

    public function goFaceBook(e:Event):void
        {
            webView.loadURL("http://www.facebook.com");
            webView.stage = null;
            webView.addEventListener(Event.COMPLETE,handleLoad);
        }
    public function handleLoad(e:Event):void
        {
            var bitmapData:BitmapData = new BitmapData(webView.viewPort.width, webView.viewPort.height);
            webView.drawViewPortToBitmapData(bitmapData);
            var webViewBitmap:Bitmap=new Bitmap(bitmapData);
            addChild(webViewBitmap);
        }
    }
}
```

第 63 章：将 worker 用于并发

用于桌面平台的 **Flash Player 11.4** 和更高版本以及 **Adobe AIR 13.4** 和更高版本

ActionScript worker 使并发执行代码成为可能，换句话说，即可在不中断主代码执行的情况下在后台执行代码。

ActionScript 并发 api 仅在 Flash Player 11.4 和更高版本以及 AIR 3.4 和更高版本的桌面平台上可用。用于移动平台的 AIR 中不支持并发。

了解 worker 与并发

用于桌面平台的 **Flash Player 11.4** 和更高版本以及 **Adobe AIR 13.4** 和更高版本

当某应用程序不使用 worker 时，该应用程序的代码在称为执行线程的单个线性块的执行步骤中执行。该线程可执行开发人员编写的代码。它还执行作为运行时一部分的大部分代码，尤其是当显示对象的属性更改时更新屏幕的代码。虽然代码用块的形式以方法和类进行编写，但在运行时，代码一次执行一行，就好像其是用一长串步骤编写而成。请考虑下面应用程序执行的步骤假设示例：

1. 输入帧：运行时调用任意 enterFrame 事件处理函数并一次运行其一个代码
2. 鼠标事件：用户移动鼠标，并当发生各种 rollover 和 rollout 事件时，运行时调用任意鼠标事件处理函数
3. 加载完整事件：从 url 中加载 xml 文件的请求返回并带有加载的文件数据。事件处理函数被调用并运行其步骤，读取 xml 内容并从 xml 中创建一组对象。
4. 鼠标事件：鼠标再次移动，因此运行时调用相关鼠标事件处理函数
5. 渲染：没有更多事件在等待，因此运行时根据对显示对象所做的任何更改来更新屏幕
6. 输入帧：循环再次开始

如示例中所述，假设步骤 1-5 在称为帧的单个时间块内按顺序运行。由于它们在单线程中按顺序运行，因此运行时无法中断过程的一个步骤以运行另一个步骤。在每秒 30 帧的帧速率下，运行时有不到三十分之一秒的时间来执行所有这些操作。在许多有足够时间运行代码的示例中，在剩余时间期间运行时只是等待。但是，假设在步骤 3 中加载的 xml 数据非常大，深深嵌入 xml 结构。随着代码在 xml 中循环并创建对象，可以想得到其可能花费比三十分之一秒更长的时间来执行这项工作。在这种情况下，后面的步骤 （响应鼠标并刷新屏幕）不会以应有的速度发生。这将由于屏幕刷新不够快速响应用户移动鼠标而导致屏幕冻结和不连贯。

如果所有代码都在同一线程中执行，则只有一种方法可避免临时不连贯和冻结。这便是不执行耗时操作，如在大量数据中循环。 ActionScript worker 可提供另一种解决方案。使用 worker，可以在单独的 worker 中执行长时间运行的代码。每个 worker 都在单独线程中运行，因此后台 worker 在其自身线程中执行耗时操作。这样释放了主 worker 的执行线程以刷新屏幕每帧，而不会受到其他工作的阻碍。

这种方法中同时运行多个代码操作的能力称为并发。当后台 worker 完成其工作或者处于途中 “ 进行 ” 点时，可以发送主 worker 通知和数据。这样，您可以编写复杂或耗时操作的代码，但可以避免因缺乏用户经验而使屏幕冻结。

Worker 非常有用，因为它们可以减少由于主渲染线程受其他代码阻止而使帧速率下降的可能。但是， worker 需要使用额外系统内存和 CPU，这可能使整体应用程序性能代价昂贵。由于每个 worker 均使用其自身的运行时虚拟机实例，因此即使不重要的 worker 开销也可能很大。当使用 worker 时，请在所有目标平台测试代码，以确保对系统的需求不是太大。 Adobe 建议在典型情境下请勿使用超过一个或两个后台 worker。

创建和管理 worker

用于桌面平台的 **Flash Player 11.4** 和更高版本以及 **Adobe AIR 13.4** 和更高版本

将 worker 用于并发的第一步是创建后台 worker。您可以使用两种类型的对象来创建 worker。第一种是您所创建的 Worker 实例。另一种是 WorkerDomain 对象，其可创建 Worker 并管理应用程序中的运行 Worker 对象。

当运行时加载时，其将自动创建 WorkerDomain 对象。运行时还可自动为应用程序的主 swf 创建 worker。第一个 worker 称为 原始 worker。

因为应用程序只有一个 WorkerDomain 对象，您可以使用静态 WorkerDomain.current 属性访问 WorkerDomain 实例。

您可以使用静态 Worker.current 属性，随时访问当前 Worker 实例 （当前代码在其中运行的 worker）。

从 swf 中创建 Worker 对象

正当主 swf 在原始 worker 内运行时，后台 worker 执行单个 swf 文件的代码。要使用后台 worker，您必须将 worker 代码创作和编译为 swf 文件。要创建后台 worker，父 worker 需要作为 ByteArray 对象访问该 swf 文件的字节。将该 ByteArray 传递到 WorkerDomain 对象的 createWorker() 方法，从而实际创建 worker。

有三种主要方法可以作为 ByteArray 对象得到后台 worker swf。

嵌入 worker swf

使用 [Embed] 元标签可以将 worker swf 作为 ByteArray 嵌入主 swf:

```
[Embed(source="../swfs/BgWorker.swf", mimeType="application/octet-stream")]
private static var BgWorker_ByteClass:Class;
private function createWorker():void
{
    var workerBytes:ByteArray = new BgWorker_ByteClass();
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

worker swf 作为名为 BgWorker_ByteClass 的 ByteArray 子类编译进主 swf。创建该类实例可为您提供预先填充了 worker swf 字节的 ByteArray。

加载外部 worker swf

使用 URLLoader 对象加载外部 swf 文件。swf 文件必须来自相同的安全域，例如与主 swf 加载自同一个 Internet 域的 swf 文件或包含在 AIR 应用程序包中的 swf 文件。

```
var workerLoader:URLLoader = new URLLoader();
workerLoader.dataFormat = URLLoaderDataFormat.BINARY;
workerLoader.addEventListener(Event.COMPLETE, loadComplete);
workerLoader.load(new URLRequest("BgWorker.swf"));

private function loadComplete(event:Event):void
{
    // create the background worker
    var workerBytes:ByteArray = event.target.data as ByteArray;
    var bgWorker:Worker = WorkerDomain.current.createWorker(workerBytes);

    // ... set up worker communication and start the worker
}
```

当 URLLoader 完成加载 swf 文件后，swf 的字节在 URLLoader 对象的 data 属性（在本示例中为 e ent.target.data）中可用。

将主 swf 用作 worker swf

您可以将单个 swf 用作主 swf 和 worker swf。使用主显示类的 loaderInfo.bytes 属性可以访问 swf 的字节。

```
// The primordial worker's main class constructor
public function PrimordialWorkerClass()
{
    init();
}

private function init():void
{
    var swfBytes:ByteArray = this.loaderInfo.bytes;

    // Check to see if this is the primordial worker or the background worker
    if (Worker.current.isPrimordial)
    {
        // create a background worker
        var bgWorker:Worker = WorkerDomain.current.createWorker(swfBytes);

        // ... set up worker communication and start the worker
    }
    else // entry point for the background worker
    {
        // set up communication between workers using getSharedProperty()
        // ... (not shown)

        // start the background work
    }
}
```

如果使用该技巧，那么使用 if 语句可以在主类的构造函数或其调用的方法中分支 swf 文件代码。要确定代码是在主 worker 中运行还是在后台 worker 中运行，请检查当前 Worker 对象的 isPrimordial 属性，如示例中所示。

开始 worker 执行

在创建了 worker 之后，通过调用 Worker 对象的 start() 方法开始其代码执行。 start() 操作不会立即发生。要了解 worker 何时运行，请为 Worker 对象的 workerState 事件注册一个侦听器。当 Worker 对象在其生存期内切换状态时 （例如，当其开始执行代码时），对该事件进行调度。在 workerState 事件处理函数中，请检查 Worker 对象的 state 属性是否为 WorkerState.RUNNING。在那时， worker 正在运行并且其主类的构造函数已经运行。以下代码列表显示注册 workerState 事件和调用 start() 方法的示例：

```
// listen for worker state changes to know when the worker is running
bgWorker.addEventListener(Event.WORKER_STATE, workerStateHandler);
// set up communication between workers using
// setSharedProperty(), createMessageChannel(), etc.
// ... (not shown)
bgWorker.start();
private function workerStateHandler(event:Event):void
{
    if (bgWorker.state == WorkerState.RUNNING)
    {
        // The worker is running.
        // Send it a message or wait for a response.
    }
}
```

管理 worker 执行

使用 WorkerDomain 类的 listWorkers() 方法，您可以随时访问应用程序中运行的一组 worker。该方法返回 state 属性为 WorkerState.RUNNING 的一组 worker，包括原始 worker。如果 worker 尚未开始或已停止执行，则其不包含在内。

如果您不再需要 worker，则可以调用 Worker 对象的 terminate() 方法以关闭 worker 并释放其内存和其他系统资源。

在 worker 之间进行通信

用于桌面平台的 **Flash Player 11.4** 和更高版本以及 **Adobe AIR 13.4** 和更高版本

虽然 worker 在单独的执行线程中执行其代码，但如果完全将其相互隔离，它们将不具有任何优势。worker 之间的通信最终意味着在 worker 之间传递数据。可使用三种主要机制在 worker 之间获取数据。

确定适合于特定数据传递需求的数据共享技术时，请考虑这些技术之间的两大差异。一项差异在于是否存在一个事件来通知接收器新数据可用或接收 worker 是否必须检查更新。数据共享技术之间的另一项差异与数据实际传递的方式相关。在某些情况下，接收 worker 获取共享数据的副本，这意味着需要创建更多对象，从而占用更多内存和 CPU 周期。在其他情况下，worker 可对引用相同系统基础内存的对象进行访问，这意味着创建的对象有所减少，整体而言减少了所使用的内存。此处对这些差异进行了概括：

通信技术	接收数据时调度事件	在 worker 之间共享内存
Worker 共享属性	否	否，对象是副本，不是引用
MessageChannel	是	否，对象是副本，不是引用
可共享 ByteArray	否	是，内存已共享

使用共享属性传递数据

在 worker 之间共享数据最基本的方法是使用共享属性。每个 worker 均保留共享属性值的内部词典。使用字符串键名存储属性以区分属性。要将 worker 中的对象存储为共享属性，请使用两个参数 （键名和要存储的值）调用 Worker 对象的 setSharedProperty() 方法：

```
// code running in the parent worker
bgWorker.setSharedProperty("sharedPropertyName", someObject);
```

在设置了共享属性后，可通过调用 Worker 对象的 getSharedProperty() 方法来读取该值，从而传递键名：

```
// code running in the background worker
receivedProperty = Worker.current.getSharedProperty("sharedPropertyName");
```

worker 读取或设置属性值无限制。例如，后台 worker 中的代码可以调用其 setSharedProperty() 方法以存储某值。然后，在父 worker 中运行的代码可以使用 getSharedProperty() 以接收该数据。

传递到 setSharedProperty() 方法中的值几乎可以是任何类型的对象。除了一些特殊情况外，当调用 getSharedProperty() 方法时，所返回的对象是传递到 setSharedProperty() 的对象副本，而不是对同一对象的引用。第 893 页的 “ 共享引用和复制值 ” 中介绍了有关如何共享数据的细节。

使用共享属性在 worker 之间传递数据的最大优势是，即使在 worker 运行之前其也可用。即使在 worker 运行之前，您也可以调用后台 Worker 对象的 setSharedProperty() 方法来设置共享属性。当父 worker 调用 Worker 的 start() 方法时，运行时调用子 worker 主类的构造函数。在调用 start() 之前设置的任何共享属性均可用于读取子 worker 中的代码。

使用 MessageChannel 传递数据

消息通道可在两个 worker 之间提供单向数据传递链接。使用 MessageChannel 对象在 worker 之间传递数据具有一个主要优势。当使用消息通道发送消息 （对象）时， MessageChannel 对象将调度 channelMessage 事件。接收 worker 中的代码可以侦听该事件以了解数据何时可用。这样，接收 worker 不需要持续检查数据更新。

消息通道仅与两个 worker 相关联，即发送器和接收器。要创建 MessageChannel 对象，请调用发送 Worker 对象的 createMessageChannel() 方法，将接收 worker 作为参数进行传递：

```
// In the sending worker swf
var sendChannel:MessageChannel;
sendChannel = Worker.current.createMessageChannel(receivingWorker);
```

两个 worker 均需有访问 MessageChannel 对象的权限。做这件事最简单的方法便是使用 setSharedProperty() 方法传递 MessageChannel 对象。

```
receivingWorker.setSharedProperty("incomingChannel", sendChannel);
```

在接收 worker 中，为 MessageChannel 对象的 channelMessage 事件注册一个侦听器。当发送 worker 通过消息通道发送数据时，将会对该事件进行调度。

```
// In the receiving worker swf
var incomingChannel:MessageChannel;
incomingChannel = Worker.current.getSharedProperty("incomingChannel");
incomingChannel.addEventListener(Event.CHANNEL_MESSAGE, handleIncomingMessage);
```

要实际发送数据，请在发送 worker 中调用 MessageChannel 对象的 send() 方法：

```
// In the sending worker swf
sendChannel.send("This is a message");
```

在接收 worker 中， MessageChannel 调用 channelMessage 事件处理函数。然后，接收 worker 可以通过调用 MessageChannel 对象的 receive() 方法来获取数据。

```
private function handleIncomingMessage(event:Event):void
{
    var message:String = incomingChannel.receive() as String;
}
```

接收方法所返回的对象与传递到 send() 方法的对象具有相同的数据类型。接收到的对象是发送器所传入对象的副本而不是对发送 worker 中对象的引用，除非其是第 893 页的 “ 共享引用和复制值 ”中所述的一些数据类型之一。

使用可共享 ByteArray 共享数据

在两个 worker 之间传递对象时，接收 worker 获得新对象，该对象是原始对象的副本。两个对象存储在系统内存中的不同位置。因此，接收到的对象的每份副本增加了运行时所使用的总内存。此外，您在一个 worker 中针对对象所做的任何更改并不影响另一个 worker 中的副本。有关如何复制数据的详细信息，请参阅第 893 页的 “ 共享引用和复制值 ”。

默认情况下， ByteArray 对象使用相同行为。如果您将 ByteArray 实例传递给 Worker 对象的 setSharedProperty() 方法或 MessageChannel 对象的 send() 方法，则运行时在计算机内存中创建新 ByteArray，而接收 worker 获得 ByteArray 实例（此实例是该新 ByteArray 的引用）。然而，您可以通过将 ByteArray 对象的 shareable 属性设置为 true，更改其行为。

在 worker 之间传递可共享 ByteArray 对象时，接收 worker 中的 ByteArray 实例是发送 worker 中 ByteArray 实例所用的相同操作系统基础内存的引用。当一个 worker 中的代码更改字节数组的内容时，这些更改在访问该共享字节数组的其他 worker 中立即生效。

由于各 worker 同时执行其代码，所以可能发生两个 worker 同时尝试访问字节数组中相同字节的情况。这可能会导致数据丢失或损坏。您可以使用一些 API 对共享资源的访问进行管理，从而避免这些问题。

ByteArray 类包含多个可在单个操作中验证并更改字节数组内容的方法：

- atomicCompareAndSwapIntAt() 方法

- atomicCompareAndSwapLength() 方法

此外，flash.concurrent 包包含可提供共享资源访问控制的类。

- Mutex 类
- Condition 类

共享引用和复制值

在正常情况下，当调用 Worker.setSharedProperty() 或 MessageChannel.send() 时，传递到接收 worker 的对象通过以 AMF 格式序列化而进行传递。这将带来一些后果：

- 当调用其 getSharedProperty() 方法时，在接收 worker 中创建的对象从 AMF 字节中进行反序列化。其是原始对象的副本，而不是对对象的引用。在任一 worker 中对对象所做的更改在另一 worker 的副本中不会更改。
- 无法以 AMF 格式进行序列化的对象 （如显示对象）无法使用 Worker.setSharedProperty() 或 MessageChannel.send() 传递到 worker。
- 为了对自定义类进行正确反序列化，必须使用 flash.net.registerClassAlias() 函数或 [RemoteClass] 元数据注册类定义。该类的两个 worker 版本必须使用相同的别名。

worker 之间真正实现对象共享而非复制的特殊情况有五种：

- Worker 对象
- MessageChannel 对象
- 可共享字节数组 （shareable 属性为 true 的 ByteArray 对象）
- Mutex 对象
- Condition 对象

当使用 Worker.setSharedProperty() 方法或 MessageChannel.send() 方法传递这些对象之一的实例时，每个 worker 均引用同一个基础对象。在一个 worker 中对实例所做的更改可立即用于另一 worker 中。此外，如果将这些对象之一的相同实例不止一次传递到 worker，运行时不会在接收 worker 中创建该对象的新副本。相反，将重新使用相同引用。

其他数据共享技巧

除了用于传递数据的特定于 worker 的机制之外，worker 还可使用支持在两个 swf 应用程序之间共享数据的任何现有 api 来交换数据，如下：

- 本地共享对象
- 在一个 worker 中将数据编入文件并在另一个 worker 中从该文件中读取
- 将数据存入 SQLite 数据库并从中读取数据

当在两个或更多 worker 中共享资源时，通常需要避免让多个 worker 同时访问资源。例如，让多个 worker 访问本地文件系统中的文件时，可能导致数据丢失或损坏或者不受操作系统支持。

要防止并发访问问题，使用 flash.concurrent 包的 Mutex 类和 Condition 类，以提供共享资源访问控制。

与其他数据共享机制不同，SQLite 数据库引擎旨在用于并发访问，并具有其自身内置事务支持。多个 worker 可访问 SQLite 数据库，而不会有损坏数据的风险。由于 worker 使用不同的 SQLConnection 实例，因此每个 worker 在单独事务中访问数据库。同步数据处理操作不会影响数据的完整性。

另请参阅

第 613 页的 “ 在 AIR 中使用本地 SQL 数据库 ”
flash.concurrent 包

第 64 章：安全性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

安全性是 Adobe、用户、网站所有者和内容开发人员关注的焦点。出于此原因，Adobe® Flash® Player 和 Adobe® AIR™ 包含一组安全规则和控制，以保护用户、网站所有者和内容开发人员的利益。除非另行声明，否则此讨论介绍的安全模型仅适用于 ActionScript 3.0 发布的且在 Flash Player 9.0.124.0 或更高版本中运行的 SWF 文件以及在 AIR 1.0 或更高版本中运行的 SWF、 HTML 和 JavaScript 文件。

此讨论概述了安全性；它并不试图全面阐述所有实现详细信息、使用方案或者使用某些 API 的后果。有关 Flash Player 安全性概念的详尽介绍，请参阅 Flash Player 开发人员中心主题 " 安全性 "，网址为 www.adobe.com/go/devnet_security_cn。

Flash Platform 安全概述

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 和 AIR 运行时使用的大部分安全模型以加载的 SWF 文件、 HTML、介质及其他资源的原始域为基础。特定 Internet 域 （例如 www.example.com）的文件中的可执行代码，始终可以访问该域的所有数据。这些资源放置在相同的安全分组中，该分组称为 " 安全沙箱 "。（有关详细信息，请参阅第 895 页的 " 安全沙箱 "。）

例如， SWF 文件中的 ActionScript 代码可以加载自己域中的 SWF 文件、位图、音频、文本文件及任何其他资源。此外，只要同一域中的两个 SWF 文件都是使用 ActionScript 3.0 编写的，则始终可以在这两个文件之间执行跨脚本访问操作。" 跨脚本访问 " 是指一个文件中的代码能够访问另一个文件中的代码所定义的属性、方法和对象。

对于使用 ActionScript 3.0 编写的 SWF 文件与使用 ActionScript 早期版本编写的 SWF 文件，它们之间不支持跨脚本访问；但是，可以通过使用 LocalConnection 类在这些文件之间进行通信。另外，默认情况下，禁止 SWF 文件跨脚本访问其他域的 ActionScript 3.0 SWF 文件，也禁止其加载其他域的数据；但是可以通过在加载的 SWF 文件中调用 Security.allowDomain() 方法来授予这种访问权。有关详细信息，请参阅第 911 页的 " 跨脚本访问 "。

默认情况下，以下基本安全规则始终适用：

- 位于相同安全沙箱中的资源始终可以互相访问。
- 远程沙箱的文件中的可执行代码始终不能访问本地文件和数据。

Flash Player 和 AIR 运行时认为下列域为单独域，并为其设置单独的安全沙箱：

- http://example.com
- http://www.example.com
- http://store.example.com
- https://www.example.com
- http://192.0.34.166

即使某个命名的域 （如 http://example.com）映射到特定 IP 地址 （例如 http://192.0.34.166），运行时也会为其设置单独的安全沙箱。

开发人员可以使用两种基本方法为 SWF 文件授予访问权限，使之能够访问除该 SWF 文件所属沙箱之外的其他沙箱中的资源：

- Security.allowDomain() 方法 （请参阅第 904 页的 " 作者 （开发人员）控制 "）
- URL 策略文件 （请参阅第 901 页的 " 网站控制 （策略文件） "）

在 Flash Player 和 AIR 运行时安全模型中，加载内容与提取或访问数据之间存在区别。“ 内容 ”被定义为媒体，包括运行时可显示的可视化媒体、音频、视频或包含显示媒体的 SWF 文件或 HTML。“ 数据 ”被定义为只有代码才能访问的内容。内容和数据的加载方式不同。

- 加载内容 — 可使用 Loader、 Sound 和 NetStream 等类加载内容；使用 Flex 时通过 MXML 标签加载；或在 AIR 应用程序中通过 HTML 标签加载。
- 提取数据 — 可以通过使用 Bitmap 对象、BitmapData.draw() 和 BitmapData.drawWithQuality() 方法、Sound.id3 属性或者 SoundMixer.computeSpectrum() 方法从加载的媒体内容中提取数据。 drawWithQuality 方法可用于 Flash Player 11.3 和更高版本、 AIR 3.3 和更高版本。
- 访问数据 — 可使用 URLStream、 URLLoader、 FileReference、 Socket 和 XMLSocket 等类从外部文件 （如 XML 文件）加载数据，从而直接对其进行访问。 AIR 提供了用于加载数据的其他类，如 FileStream 和 XMLHttpRequest。

Flash Player 安全模型针对加载内容和访问数据定义了不同的规则。通常，加载内容的限制要比访问数据的限制少一些。

通常，可从任意位置加载内容 （SWF 文件、位图、 mp3 文件和视频），但如果内容来自加载代码或内容的域之外的域，则会将内容划分到单独的安全沙箱中。

下面是加载内容的一些限制：

- 默认情况下，本地 SWF 文件 （从非网络地址加载的文件，例如用户硬盘上的文件）会被分类到只能与本地文件系统内容交互的沙箱中。这些文件无法从网络加载内容。有关详细信息，请参阅第 896 页的 “ 本地沙箱 ”。
- 实时消息传递协议 (RTMP) 服务器可以限制对内容的访问。有关详细信息，请参阅第 910 页的 “ 使用 RTMP 服务器传送的内容 ”。

如果加载的媒体为图像、音频或视频，则其安全沙箱之外的 SWF 文件无法访问该媒体的数据 （如像素数据和声音数据），除非该 SWF 文件的域已包含在该媒体原始域的 URL 策略文件中。有关详细信息，请参阅第 913 页的 “ 作为数据访问加载的媒体 ”。

加载数据的其他格式包括文本文件或 XML 文件，这些文件可使用 URLLoader 对象来加载。同样，这种情况下要访问其他安全沙箱中的任何数据，必须通过原始域中的 URL 策略文件来授予权限。有关详细信息，请参阅第 915 页的 “ 使用 URLLoader 和 URLStream”。

注：在 AIR 应用程序沙箱中执行的代码加载远程内容或数据时始终不需要策略文件。

安全沙箱

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

客户端计算机可以获得包含来自许多源 （如外部网站、本地文件系统或安装的 AIR 应用程序）的代码、内容和数据的各个文件。Flash Player 和 AIR 运行时会根据代码文件及其他资源（如共享对象，位图、声音、视频和数据文件）加载时的来源分别将其分配到安全沙箱中。以下部分介绍运行时强制执行的规则，这些规则控制着在给定沙箱内执行的代码或内容可以访问哪些内容。

有关 Flash Player 安全性的详细信息，请参阅 Flash Player 开发人员中心主题 “ 安全性 ”，网址为 www.adobe.com/go/devnet_security_cn。

远程沙箱

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 和 AIR 运行时将来自 Internet 的资源 （包括 SWF 文件）分类到与其原始域对应的单独沙箱中。例如，会将从 example.com 加载的资源放置到与从 foo.org 加载的资源不同的安全沙箱中。默认情况下，对这些文件授予访问其自身所在服务器中任何资源的权限。通过显式的 Web 站点许可和作者许可 （例如 URL 策略文件和 Security.allowDomain() 方法），可以允许远程 SWF 文件访问其他域的其他数据。有关详细信息，请参阅第 901 页的 " 网站控制 （策略文件） " 和第 904 页的 " 作者 （开发人员）控制 "。

远程 SWF 文件无法加载任何本地文件或资源。

有关 Flash Player 安全性的详细信息，请参阅 Flash Player 开发人员中心主题 " 安全性 "，网址为 www.adobe.com/go/devnet_security_cn。

本地沙箱

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

" 本地文件 " 是指通过使用 file: 协议或统一命名约定 (UNC) 路径引用的任何文件。本地 SWF 文件放置在四个本地沙箱中的一个内：

- 只能与本地文件系统内容交互的沙箱 — 出于安全方面的考虑，默认情况下，Flash Player 和 AIR 运行时会将所有本地文件放置在只能与本地文件系统内容交互的沙箱中。通过此沙箱，可执行代码可以读取本地文件 （例如通过使用 URLLoader 类），但无法通过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他方式不适当地共享。
- 只能与远程内容交互的沙箱 — 编译 SWF 文件时，可以指定该文件作为本地文件运行时拥有网络访问权限 （请参阅第 898 页的 " 设置本地 SWF 文件的沙箱类型 "）。这些文件位于只能与远程内容交互的沙箱中。分配到只能与远程内容交互的沙箱中的 SWF 文件将失去其本地文件访问权限，但允许这些 SWF 文件访问网络中的数据。不过，只有通过 URL 策略文件或调用 Security.allowDomain() 方法来授予操作权限，才允许远程内容交互的 SWF 文件读取源自网络的数据。为授予此类权限， URL 策略文件必须向 " 所有 " 域授予权限，方法是使用 <allow-access-from domain="*"/> 或使用 Security.allowDomain("*")。有关详细信息，请参阅第 901 页的 " 网站控制 （策略文件） " 和第 904 页的 " 作者 （开发人员）控制 "。
- 受信任的本地沙箱 — 注册为受信任 （由用户或安装程序注册）的本地 SWF 文件放置在受信任的本地沙箱中。系统管理员和用户还可以根据安全注意事项将本地 SWF 文件重新分配 （移动）到受信任的本地沙箱，或者从受信任的本地沙箱重新分配 （移动）本地 SWF 文件 （请参阅第 899 页的 " 管理员控制 " 和第 900 页的 " 用户控制 "）。分配到受信任的本地沙箱的 SWF 文件可以与其他任何 SWF 文件交互，也可以从任何位置 （远程或本地）加载数据。
- AIR 应用程序安全沙箱 — 该沙箱包含在运行 AIR 应用程序时安装的内容。默认情况下，在 AIR 应用程序沙箱中执行的代码可跨脚本访问来自任何域的代码。但是，不允许 AIR 应用程序沙箱外的文件跨脚本访问应用程序沙箱内的代码。默认情况下， AIR 应用程序沙箱内的代码和内容可加载来自任何域的内容和数据。

只能与远程内容交互的沙箱和只能与本地文件系统内容交互的沙箱之间的通信以及只能与本地文件系统内容交互的沙箱和远程沙箱之间的通信是严格禁止的。运行于 Flash Player 中的应用程序或用户 / 管理员不能授予允许此类通信的权限。

在本地 HTML 文件和本地 SWF 文件之间以任一方向访问脚本（例如使用 ExternalInterface 类）均要求涉及的 HTML 文件和 SWF 文件应位于受信任的本地沙箱中。这是因为浏览器的本地安全模型与 Flash Player 本地安全模型不同。

只能与远程内容交互的沙箱中的 SWF 文件无法加载只能与本地文件系统内容交互的沙箱中的 SWF 文件。只能与本地文件系统内容交互的沙箱中的 SWF 文件无法加载只能与远程内容交互的沙箱中的 SWF 文件。

AIR 应用程序沙箱

Adobe AIR 1.0 和更高版本

Adobe AIR 运行时可以向 Flash Player 安全沙箱模型添加其他的沙箱，称为应用程序沙箱。作为 AIR 应用程序一部分安装的文件将加载到应用程序沙箱内。由应用程序加载的任何其他文件具有的安全限制与常规 Flash Player 安全模型指定的安全限制相对应。

安装应用程序时， AIR 包中包括的所有文件都将安装到用户计算机上的应用程序目录下。开发人员在代码中可以通过 app:/ URL 方案引用此目录（请参阅 第 698 页的 "URI 方案 "）。在应用程序运行时，应用程序目录树中的所有文件都会分配到应用程序沙箱中。应用程序沙箱中的内容具有 AIR 应用程序的完全访问权限，包括与本地文件系统内容进行交互。

许多 AIR 应用程序只能使用这些本地安装的文件来运行应用程序。但是，不会限制 AIR 应用程序仅加载应用程序目录中的文件，它们可以加载任意源中任何类型的文件。其中包括用户计算机上的本地文件以及可用外部源中的文件 （例如本地网络或 Internet 上的文件）。文件类型不会对安全限制产生影响；加载的 HTML 文件与从相同源加载的 SWF 文件具有相同的安全权限。

应用程序安全沙箱中的内容可以访问 AIR API，而其他沙箱中的内容则无法访问。例如，限制 air.NativeApplication.nativeApplication.applicationDescriptor 属性 （该属性返回应用程序的应用程序描述符文件的内容）访问应用程序安全沙箱中的内容。另一个示例是受限制的 API 为 FileStream 类，其中包含用于读取和写入本地文件系统的方法。

在用于 Adobe Flash Platform 的 ActionScript 3.0 参考 中，仅用于应用程序安全沙箱中的内容的 ActionScript API 将使用 AIR 徽标指示。在其他沙箱中使用这些 API 会导致运行时引发 SecurityError 异常。

对于 HTML 内容 （位于 HTMLLoader 对象中），所有 AIR JavaScript API （使用 AIRAliases.js 文件时通过 window.runtime 属性或 air 对象可用）都可用于应用程序安全沙箱中的内容。其他沙箱中的 HTML 内容无权访问 window.runtime 属性，因此该内容无法访问 AIR 或 Flash Player API。

在 AIR 应用程序沙箱内执行内容具有下列额外的限制：

- 对于应用程序安全沙箱中的 HTML 内容，在加载代码后使用可将字符串动态转换为可执行代码的 API 时存在一些限制。这是为了阻止应用程序从非应用程序源 （例如潜在不安全网络域）意外插入 （及执行）代码。使用 eval() 函数是一个示例。有关详细信息，请参阅第 927 页的 " 对不同沙箱中的内容的代码限制 "。
- 为了阻止潜在仿冒攻击，在应用程序安全沙箱的 SWF 内容中忽略了 ActionScript TextField 对象的 HTML 内容中的 img 标签。
- 应用程序沙箱中的内容无法在 ActionScript 2.0 文本字段的 HTML 内容中使用 asfunction 协议。
- 应用程序沙箱中的 SWF 内容无法使用跨域缓存，这是一项增加到 Flash Player 9 Update 3 的功能。 Flash Player 通过此功能可以永久缓存 Adobe 平台组件内容，并根据需要在加载的 SWF 内容中重复使用该内容（无需多次重新加载该内容）。

对 AIR 内的 JavaScript 的限制

Adobe AIR 1.0 和更高版本

与应用程序安全沙箱中的内容不同，非应用程序安全沙箱中的 JavaScript 内容随时都可以调用 eval() 函数来执行动态生成的代码。但是，对 AIR 内在非应用程序安全沙箱中运行的 JavaScript 具有限制。这些项目包括：

- 非应用程序沙箱中的 JavaScript 代码无法访问 window.runtime 对象，也无法执行 AIR API。
- 默认情况下，非应用程序安全沙箱中的内容无法使用 XMLHttpRequest 调用从调用该请求的域之外的其他域加载数据。但是，通过设置包含 frame 或 iframe 中的 allowCrossdomainXHR 属性，应用程序代码可以授予非应用程序内容执行此操作的权限。有关详细信息，请参阅第 927 页的 " 对不同沙箱中的内容的代码限制 "。
- 调用 JavaScript window.open() 方法时存在一些限制。有关详细信息，请参阅第 930 页的 " 调用 JavaScript window.open() 方法的限制 "。
- 远程 （网络）安全沙箱中的 HTML 内容只能从远程域 （网络 URL）加载 CSS、 frame、 iframe 和 img 内容。

- 只能与本地文件系统内容交互的沙箱、只能与远程内容交互的沙箱或受信任的本地沙箱中的 HTML 内容只能从本地沙箱（而不是应用程序或网络 URL）加载 CSS、 frame、 iframe 和 img 内容。

有关详细信息，请参阅第 927 页的 " 对不同沙箱中的内容的代码限制 "。

设置本地 SWF 文件的沙箱类型

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

最终用户或计算机管理员可以指定某个本地 SWF 文件是受信任的，以允许该文件从所有域 （本地和网络）加载数据。这一点在 " 全局 Flash Player 信任 " 目录和 " 用户 Flash Player 信任 " 目录中指定。有关详细信息，请参阅第 899 页的 " 管理员控制 " 和第 900 页的 " 用户控制 "。

有关本地沙箱的详细信息，请参阅第 896 页的 " 本地沙箱 "。

Adobe Flash Professional

通过在创作工具中设置文档发布设置，您可以将 SWF 文件配置在只能与本地文件系统内容交互的沙箱或只能与远程内容交互的沙箱中。

Adobe Flex

通过在 Adobe Flex 编译器中设置 use-network 标志，您可以配置只能与本地文件系统内容交互的沙箱或只能与远程内容交互的沙箱的 SWF 文件。有关详细信息，请参阅 《构建和部署 Adobe Flex 3 应用程序》中的 " 关于应用程序编译器选项 "。

Security.sandboxType 属性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

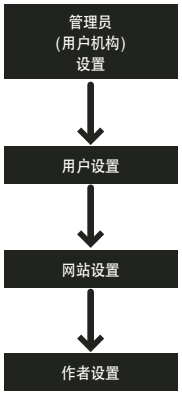
SWF 文件的作者可使用只读静态 Security.sandboxType 属性确定 Flash Player 或 AIR 运行时已向其分配 SWF 文件的沙箱类型。 Security 类包括表示 Security.sandboxType 属性可能值的常量，如下所示：

- Security.REMOTE — SWF 文件来自 Internet URL，并遵守基于域的沙箱规则。
- Security.LOCAL_WITH_FILE — SWF 文件是本地文件，但尚未受到用户信任，且没有使用网络名称进行发布。此 SWF 文件可以从本地数据源读取数据，但无法与 Internet 进行通信。
- Security.LOCAL_WITH_NETWORK — SWF 文件是本地文件，且尚未受到用户信任，但已使用网络名称进行发布。此 SWF 文件可与 Internet 通信，但不能从本地数据源读取数据。
- Security.LOCAL_TRUSTED — SWF 文件是本地文件，且已使用 " 设置管理器 " 或 Flash Player 信任配置文件受到用户信任。此 SWF 文件既可以从本地数据源读取数据，也可以与 Internet 进行通信。
- Security.APPLICATION — SWF 文件在 AIR 应用程序中运行，并且随该应用程序的包 （AIR 文件）一起安装。默认情况下， AIR 应用程序沙箱中的文件可以跨脚本访问任何域中的任何文件。但是， AIR 应用程序沙箱以外的文件不会获许跨脚本访问 AIR 文件。默认情况下， AIR 应用程序沙箱中的文件可以加载任何域中的内容和数据。

权限控制

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 客户端运行时安全模型是围绕 SWF 文件、本地数据和 Internet URL 等这些对象资源设计而成的模型。“ 资源持有者 ” 是指拥有或使用这些资源的各方。资源持有者可以对其自己的资源进行控制 （安全设置），每种资源有四个持有者。Flash Player 对这些控制严格采用一种权利层次，如下图所示：



安全控制层次

该图说明，如果管理员限制对资源的访问，则任何持有者都不能覆盖该限制。

对于 AIR 应用程序，这些权限控制只适用于在 AIR 应用程序沙箱外运行的内容。

管理员控制

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

计算机的管理用户 （使用管理权限登录的用户）可以应用能影响计算机所有用户的 Flash Player 安全设置。在非企业环境（例如家庭计算机）中，通常只有一个用户，该用户也拥有管理访问权限。即使是在企业环境中，单个用户也可以拥有计算机管理权限。

管理用户控制有两种类型：

- mms.cfg 文件
- “ 全局 Flash Player 信任 ” 目录

mms.cfg 文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

mms.cfg 文件是一个文本文件，由管理员用于启用或限制用户对各种功能的访问。 Flash Player 启动时，将从此文件中读取其安全设置，然后使用这些设置限制功能。 mms.cfg 文件包含管理员用于管理隐私控制、本地文件安全性、套接字连接等功能。

SWF 文件可通过调用 Capabilities.avHardwareDisable 和 Capabilities.localFileReadDisable 属性来访问已禁用功能的某些信息。但是， mms.cfg 文件中的大部分设置无法通过 ActionScript 进行查询。

为对计算机强制执行与应用程序无关的安全和隐私策略，只能由系统管理员修改 mms.cfg 文件。 mms.cfg 文件不能用于安装应用程序。虽然使用管理权限运行的安装程序可以修改 mms.cfg 文件的内容，但是 Adobe 将此类使用视为违反用户的信任，并且劝告安装程序的创建者决不要修改 mms.cfg 文件。

mms.cfg 文件存储在以下位置：

- Windows：system\Macromed\Flash\mms.cfg

  （例如， C:\WINDOWS\system32\Macromed\Flash\mms.cfg）

- Mac：app support/Macromedia/mms.cfg

  （例如， /Library/Application Support/Macromedia/mms.cfg）

有关 mms.cfg 文件的详细信息，请参阅 《Flash Player 管理指南》，网址为 www.adobe.com/go/flash_player_admin_cn。

“ 全局 Flash Player 信任 ” 目录

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

管理用户和安装应用程序可以针对所有用户将指定的本地 SWF 文件注册为受信任。这些 SWF 文件会被分配到受信任的本地沙箱。它们可以与任何其他 SWF 文件进行交互，也可以从任意位置 （远程或本地）加载数据。文件在 “ 全局 Flash Player 信任 ” 目录中被指定为受信任，位置如下：

- Windows：system\Macromed\Flash\FlashPlayerTrust

  （例如， C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust）

- Mac：app support/Macromedia/FlashPlayerTrust

  （例如， /Library/Application Support/Macromedia/FlashPlayerTrust）

“Flash Player 信任 ” 目录可以包含任意数目的文本文件，每个文件均列出受信任的路径，一个路径占一行。每个路径可以是单个的 SWF 文件、HTML 文件，也可以是目录。注释行以 # 号开头。例如，包含以下文本的 Flash Player 信任配置文件表示将向指定目录及所有子目录中的所有文件授予受信任状态：

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

信任配置文件中列出的路径应始终是本地路径或 SMB 网络路径。信任配置文件中的任何 HTTP 路径均会被忽略；只能信任本地文件。

为避免发生冲突，应为每个信任配置文件指定一个与安装应用程序相应的文件名，并且使用 .cfg 文件扩展名。

由于开发人员通过安装应用程序分发本地运行的 SWF 文件，因此可以让安装应用程序向 “ 全局 Flash Player 信任 ” 目录添加一个配置文件，为要分发的文件授予完全访问权限。安装应用程序必须由拥有管理权限的用户来运行。与 mms.cfg 文件不同，包含 “ 全局 Flash Player 信任 ” 目录是为了让安装应用程序授予信任权限。管理用户和安装应用程序都可以使用 “ 全局 Flash Player 信任 ” 目录指定受信任的本地应用程序。

此外，还有适用于单个用户的 “Flash Player 信任 ” 目录 （请参阅下一节第 900 页的 “ 用户控制 ”）。

用户控制

Flash Player 9 和更高版本

Flash Player 提供三种不同的用户级别权限设置机制：“ 设置 UI”、“ 设置管理器 ” 和 “ 用户 Flash Player 信任 ” 目录。

设置 UI 和设置管理器

Flash Player 9 和更高版本

“设置 UI”是一种用于配置特定域设置的快速交互机制。“设置管理器”显示一个更详细的界面，并提供全局更改功能，全局更改可影响对许多域或所有域拥有的权限。另外，当 SWF 文件请求新的权限，要求有关安全或隐私的运行时决策时，程序会显示一些对话框，用户可以在这些对话框中调整某些 Flash Player 设置。

设置管理器和设置 UI 提供了与安全相关的选项，如摄像机和麦克风设置、共享对象存储设置、与旧内容相关的设置等。设置管理器和设置 UI 在 AIR 应用程序中都不可用。

注：在 mms.cfg 文件中所进行的任何设置 （请参阅第 899 页的“管理员控制”）都不会反映在设置管理器中。

有关设置管理器的详细信息，请访问 www.adobe.com/go/settingsmanager_cn。

“用户 Flash Player 信任”目录

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用户和安装应用程序可以将指定的本地 SWF 文件注册为受信任。这些 SWF 文件会被分配到受信任的本地沙箱。它们可以与任何其他 SWF 文件进行交互，也可以从任意位置 （远程或本地）加载数据。用户在“用户 Player 信任”目录中将文件指定为受信任，该目录与 Flash 共享对象存储区域的所在目录相同，位置 （特定于当前用户）如下：

- Windows：app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  （例如，在 Windows XP 上：C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust，在 Windows Vista 上：C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust）

  在 Windows 中， Application Data 默认为隐藏状态。若要显示隐藏的文件夹和文件，请选择“我的电脑”打开 Windows 资源管理器，选择“工具”>“文件夹选项”，然后选择“查看”选项卡。在“查看”选项卡之下，选择“显示所有文件和文件夹”单选按钮。

- Mac：app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  （例如， /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust）

  这些设置只会影响当前用户，不会影响登录到计算机的其他用户。如果没有管理权限的用户在属于他们自己的系统中安装了某个应用程序，则“用户 Flash Player 信任”目录允许安装程序将该应用程序注册为该用户的受信任程序。

  由于开发人员通过安装应用程序分发本地运行的 SWF 文件，因此可以让安装应用程序向“用户 Flash Player 信任”目录添加一个配置文件，为要分发的文件授予完全访问权限。即使在这种情况下，也将“用户 Flash Player 信任”目录文件视为用户控制，原因是用户操作 （安装）启动了它。

  此外，还有一个“全局 Flash Player 信任”目录，管理用户或安装程序可使用该目录为所有计算机用户注册一个应用程序（请参阅第 899 页的“管理员控制”）。

网站控制 （策略文件）

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要使来自 Web 服务器的数据可用于来自其他域的 SWF 文件，可以在服务器上创建一个策略文件。“策略文件”是位于服务器上特定位置的 XML 文件。

策略文件可影响对许多资源的访问，其中包括以下内容：

- 位图、声音和视频中的数据
- 加载 XML 和文本文件

- 将 SWF 文件从其他安全域导入到执行加载的 SWF 文件所在的安全域
- 对套接字和 XML 套接字连接的访问

ActionScript 对象可实例化两种不同的服务器连接：基于文档的服务器连接和套接字连接。 Loader、 Sound、 URLLoader 和 URLStream 等 ActionScript 对象可实例化基于文档的服务器连接，这些对象均根据 URL 加载文件。 ActionScript Socket 和 XMLSocket 对象进行套接字连接，这些对象操作的是数据流而非加载的文档。

由于 Flash Player 支持两种服务器连接，因此有两种策略文件 — URL 策略文件和套接字策略文件。

- 基于文档的连接需要 “URL 策略文件 ”。服务器通过这些文件指示其数据和文档是否可用于特定域或所有域提供的 SWF 文件。
- 套接字连接需要 “ 套接字策略文件 ”，套接字策略文件可以在较低的 TCP 套接字级别使用 Socket 类和 XMLSocket 类直接启用网络。

Flash Player 要求使用尝试的连接希望使用的协议来传输策略文件。例如，如果将策略文件放置在您的 HTTP 服务器上，则允许其他域中的 SWF 文件从该服务器 （作为 HTTP 服务器）加载数据。但是，如果在同一服务器上未提供套接字策略文件，则禁止其他域的 SWF 文件在套接字级别连接到该服务器。换言之，检索策略文件的方法必须与连接方法相匹配。

策略文件的用法和语法将在本节其余部分加以概述，主要讨论为 Flash Player 10 发布的 SWF 文件的用法和语法。（在早期的 Flash Player 版本中，策略文件实现稍有不同，后续的版本加强了 Flash Player 安全性。）有关策略文件的详细信息，请参阅 Flash Player 开发人员中心主题 “Flash Player 9 中策略文件的更改 ”，网址为 www.adobe.com/go/devnet_security_cn。

在 AIR 应用程序沙箱中执行的代码不需要策略文件就可从 URL 或套接字访问数据。在非应用程序沙箱内执行的 AIR 应用程序中的代码要求使用策略文件。

主策略文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下， Flash Player （和 AIR 应用程序沙箱以外的 AIR 内容）首先在服务器的根目录中查找名为 crossdomain.xml 的 URL 策略文件，然后在端口 843 上查找套接字策略文件。这些位置上的文件称为 “ 主策略文件 ”。（对于套接字连接， Flash Player 还会在与主连接相同的端口上查找套接字策略文件。但是，在该端口上找到的策略文件并不视为主策略文件。）

除指定访问权限外，主策略文件还包含一条 “ 元策略 ” 语句。元策略指定哪些位置可包含策略文件。 URL 策略文件的默认元策略为 “master-only”，它表示 /crossdomain.xml 是服务器上唯一允许的策略文件。套接字策略文件的默认元策略为 “all”，它表示主机上的任何套接字都能提供套接字策略文件。

注：在 Flash Player 9 及更低版本中， URL 策略文件的默认元策略为 “all”，它表示任何目录都可以包含策略文件。如果部署的应用程序不是从默认的 /crossdomain.xml 文件而是从其他位置加载策略文件，并且这些应用程序目前可能运行于 Flash Player 10 中，请您 （或服务器管理员）务必修改主策略文件以允许其他策略文件。有关如何指定不同元策略的信息，请参阅 Flash Player 开发人员中心主题 “Flash Player 9 中策略文件的更改 ”，网址为 www.adobe.com/go/devnet_security_cn。

SWF 文件可以通过调用 Security.loadPolicyFile() 方法检索其他策略文件名或其他目录位置。但是，如果主策略文件未指定目标位置能提供策略文件，则调用 loadPolicyFile() 无效，即使该位置有策略文件。在尝试需要策略文件的任何网络操作之前，请先调用 loadPolicyFile()。 Flash Player 自动将网络请求加入队列并排在对应的策略文件尝试之后。例如，可以在调用 Security.loadPolicyFile() 之后立即进行网络操作。

在检索主策略文件时， Flash Player 会用三秒钟等待服务器响应。如果未接收到响应， Flash Player 则假定主策略文件不存在。但是，对 loadPolicyFile() 的调用没有默认超时值； Flash Player 假定调用的文件存在，在加载文件之前会一直等待。因此，如果要确保加载主策略文件，请使用 loadPolicyFile() 来明确调用主策略文件。

尽管该方法的名称是 Security.loadPolicyFile()，如果不发出需要策略文件的网络调用，也不能加载策略文件。对 loadPolicyFile() 的调用只是告知 Flash Player 到何处查找所需的策略文件。

系统不会通知您何时启动或完成策略文件请求，也没有必要这样做。 Flash Player 执行异步策略检查，自动等待连接的启动直到策略文件检索成功完成。

下面几节介绍只适用于 URL 策略文件的相关信息。有关套接字策略文件的详细信息，请参阅第 916 页的 “ 连接到套接字 ”。

URL 策略文件范围

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

URL 策略文件仅适用于从其中加载该文件的目录及其子目录。根目录中的策略文件适用于整个服务器；而从任意子目录加载的策略文件仅适用于该目录及其子目录。

策略文件仅影响对其所在特定服务器的访问。例如，位于 https://www.adobe.com:8080/crossdomain.xml 的策略文件仅适用于在端口 8080 通过 HTTPS 对 www.adobe.com 进行的数据加载调用。

在 URL 策略文件中指定访问权限

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

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）。可采用下列两种方式之一来指示通配符域：

- 单个星号 (*)，表示匹配所有域和所有 IP 地址
- 后接后缀的星号，表示只匹配那些以指定后缀结尾的域

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， xyz.com 会被看作是 *.xyz.com 的一部分。 IP 域规范中不允许使用通配符。

下面的示例显示了一个 URL 策略文件，该文件允许访问源自 *.example.com、 www.friendOfExample.com 和 192.0.34.166 的 SWF 文件。

```
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*.example.com" />
    <allow-access-from domain="www.friendOfExample.com" />
    <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

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的 SWF 文件授予访问权限。 Flash Player 不执行 DNS 解析。

您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，其默认值为 true。如果您的策略文件在 HTTPS 服务器上，并且要允许非 HTTPS 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。特别是将此属性设置为 false 时，会使安全内容受到电子欺骗和窃听攻击。 Adobe 强烈建议不要将 secure 属性设置为 false。

如果要加载的数据在 HTTPS 服务器上，但是加载数据的 SWF 文件在 HTTP 服务器上， Adobe 建议将加载数据的 SWF 文件移至 HTTPS 服务器。这样，可以使安全数据的所有副本得到 HTTPS 的保护。但是，如果决定必须将要执行加载的 SWF 文件保存在 HTTP 服务器上，则需将 secure="false" 属性添加到 <allow-access-from> 标签，如以下代码所示：

```
<allow-access-from domain="www.example.com" secure="false" />
```

另一种可用于允许访问权限的元素是 allow-http-request-headers-from 标签。此元素授权客户端 （此客户端承载其他权限域的内容）向您的域发送用户定义的标头。 <allow-access-from> 标签授权其他域提取您域中的数据，而 allow-http-request-headers-from 标签授权其他域将数据以标头的形式发送到您的域中。在下面的示例中，任何域都可以向当前域发送 SOAPAction 标头：

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

如果 allow-http-request-headers-from 语句存在于主策略文件中，它将应用于主机上的所有目录。否则，它只应用于包含此语句的策略文件所在的目录和子目录。

预加载策略文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

从服务器加载数据或连接到套接字是一种异步操作。 Flash Player 只是等待策略文件完成下载，然后才开始主操作。但是，从图像中提取像素数据或从声音中提取采样数据是一种同步操作。必须先加载策略文件才能提取数据。加载媒体时，请指定媒体检索策略文件：

- 使用 Loader.load() 方法时，设置 context 参数的 checkPolicyFile 属性，该参数是一个 LoaderContext 对象。
- 使用 <img> 标签在文本字段中嵌入图像时，将 <img> 标签的 checkPolicyFile 属性设置为 "true"，如下所示：

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- 使用 Sound.load() 方法时，设置 context 参数的 checkPolicyFile 属性，该参数是一个 SoundLoaderContext 对象。
- 使用 NetStream 类时，设置 NetStream 对象的 checkPolicyFile 属性。

设置上述参数时， Flash Player 首先会检查是否已经为该域下载了任何策略文件。然后，它查找服务器上默认位置中的策略文件，同时检查 <allow-access-from> 语句和是否存在元策略。最后，考虑对 Security.loadPolicyFile() 方法的所有未处理的调用，以查看它们是否在范围内。

作者 （开发人员）控制

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

用于授予安全权限的主 ActionScript API 是 Security.allowDomain() 方法，它将向指定域中的 SWF 文件授予权限。在下面的示例中， SWF 文件向 www.example.com 域提供的 SWF 文件授予访问权限：

```
Security.allowDomain("www.example.com")
```

此方法为下列各项授予权限：

- SWF 文件之间的跨脚本访问 （请参阅第 911 页的 “ 跨脚本访问 ”）
- 显示列表访问 （请参阅第 913 页的 “ 遍历显示列表 ”）
- 事件检测 （请参阅第 913 页的 “ 事件安全性 ”）
- 对 Stage 对象的属性和方法的完全访问 （请参阅 第 912 页的 “Stage 安全性 ”）

调用 Security.allowDomain() 方法的主要目的是为外部域中的 SWF 文件授予权限以访问调用 Security.allowDomain() 方法的 SWF 文件的脚本。有关详细信息，请参阅第 911 页的 “ 跨脚本访问 ”。

如果将 IP 地址指定为 Security.allowDomain() 方法的参数，则不允许任何源自该指定 IP 地址的访问方进行访问。相反，只允许 URL 中包含该指定 IP 地址的访问方进行访问，而不允许其域名映射到该 IP 地址的访问方进行访问。例如，如果域名 www.example.com 映射到 IP 地址 192.0.34.166，则对 Security.allowDomain("192.0.34.166") 的调用不会授予对 www.example.com 的访问权限。

可以将通配符 "*" 传递给 Security.allowDomain() 方法以允许从所有域进行访问。由于这种方式会为 “ 所有 ” 域中的 SWF 文件授予访问执行调用的 SWF 文件的脚本的权限，因此请谨慎使用通配符 "*"。

ActionScript 还包括一个权限 API，称为 Security.allowInsecureDomain()。此方法与 Security.allowDomain() 方法的作用相同，只是从安全 HTTPS 连接提供的 SWF 文件调用时，此方法还会允许非安全协议 （例如 HTTP）提供的其他 SWF 文件访问执行调用的 SWF 文件。但是，在安全协议 (HTTPS) 中的文件与非安全协议 （例如 HTTP）中的文件之间执行脚本访问操作并不是一种好的安全性做法；这样做会使安全内容受到电子欺骗和窃听攻击。下面是此类攻击的作用方式：由于 Security.allowInsecureDomain() 方法允许通过 HTTP 连接提供的 SWF 文件访问安全 HTTPS 数据，因此介入 HTTP 服务器和用户之间的攻击者能够将 HTTP SWF 文件替换为它们自己的文件，这样便可访问您的 HTTPS 数据。

重要说明：不允许在 AIR 应用程序沙箱中执行的代码调用 Security 类的 allowDomain() 或 allowInsecureDomain() 方法。

另一种与安全性相关的重要方法是 Security.loadPolicyFile() 方法，该方法可让 Flash Player 在非标准位置检查是否存在策略文件。有关详细信息，请参阅第 901 页的 “ 网站控制 （策略文件） ”。

限制网络 API

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可通过两种方式限制网络 API。为防止恶意行为，需阻止对常用保留端口进行访问；您不能在自己的代码中覆盖这些阻止设置。要控制 SWF 文件访问与其他端口相关的网络功能，可以使用 allowNetworking 设置。

阻止的端口

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 和 Adobe AIR 对某些端口的 HTTP 访问设有限制，与浏览器类似。通常用于非 HTTP 类型服务器的某些标准端口上不允许使用 HTTP 请求。

任何访问网络 URL 的 API 都会受到这些端口阻止限制的约束。只有直接调用套接字的 API 例外，如 Socket.connect() 和 XMLSocket.connect()，或 Security.loadPolicyFile() 调用 （这种情况是加载套接字策略文件）。对套接字连接的允许或拒绝是通过在目标服务器上使用套接字策略文件实现的。

下面的列表显示了适用端口阻止的 ActionScript 3.0 API：

FileReference.download()、 FileReference.upload()、 Loader.load()、 Loader.loadBytes()、 navigateToURL()、 NetConnection.call()、 NetConnection.connect()、 NetStream.play()、 Security.loadPolicyFile()、 sendToURL()、 Sound.load()、 URLLoader.load()、 URLStream.load()

端口阻止也适用于共享库的导入、文本字段中 <img> 标签的使用、HTML 页中 SWF 文件的加载（使用 <object> 和 <embed> 标签）。

端口阻止也适用于文本字段中 <img> 标签的使用、 HTML 页中 SWF 文件的加载 （使用 <object> 和 <embed> 标签）。

下面的列表显示了受阻止的端口：

HTTP：20 （ftp 数据）、 21 （ftp 控制）

HTTP 和 FTP：1 (tcpmux)、 7 (echo)、 9 (discard)、 11 (systat)、 13 (daytime)、 15 (netstat)、 17 (qotd)、 19 (chargen)、 22 (ssh)、 23 (telnet)、 25 (smtp)、 37 (time)、 42 (name)、 43 (nicname)、 53 (domain)、 77 (priv-rjs)、 79 (finger)、 87 (ttylink)、 95 (supdup)、 101 (hostriame)、 102 (iso-tsap)、 103 (gppitnp)、 104 (acr-nema)、 109 (pop2)、 110 (pop3)、 111 (sunrpc)、 113 (auth)、 115 (sftp)、 117 (uucp-path)、 119 (nntp)、 123 (ntp)、 135 (loc-srv / epmap)、 139 (netbios)、 143 (imap2)、 179 (bgp)、 389 (ldap)、 465 (smtp+ssl)、 512 (print / exec)、

513 (login)、 514 (shell)、 515 (printer)、 526 (tempo)、 530 (courier)、 531 (chat)、 532 (netnews)、 540 (uucp)、 556 (remotefs)、 563 (nntp+ssl)、 587 (smtp)、 601 (syslog)、 636 (ldap+ssl)、 993 (ldap+ssl)、 995 (pop3+ssl)、 2049 (nfs)、 4045 (lockd)、 6000 (x11)

使用 allowNetworking 参数

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

您可以通过在包含 SWF 内容的 HTML 页中的 <object> 和 <embed> 标签中设置 allowNetworking 参数来控制 SWF 文件访问网络的功能。

allowNetworking 的可能的值为：

- "all" （默认值） — 在 SWF 中允许所有的网络 API。
- "internal"— SWF 文件可能不调用浏览器导航或浏览器交互 API （在本节后面部分中列出），但是它会调用任何其他网络 API。
- "none"— SWF 文件可能不调用浏览器导航或浏览器交互 API （在本节后面部分中列出），并且它无法使用任何 SWF 到 SWF 通信 API （也在本节后面部分中列出）。

allowNetworking 参数主要在 SWF 文件及所在的 HTML 页来自不同的域时使用。当要加载的 SWF 文件与其所在的 HTML 页来自同一个域时，不建议使用 "internal" 或 "none" 值，原因是您不能保证始终同时加载 SWF 文件和想要的 HTML 页。不受信任方可以从您的域中加载未包含在 HTML 中的 SWF 文件，这种情况下， allowNetworking 限制不会按预期发挥作用。

调用被禁止的 API 会引发 SecurityError 异常。

在包含 SWF 文件引用的 HTML 页的 <object> 和 <embed> 标签中添加 allowNetworking 参数并设置该参数的值，如下面的示例中所示：

```
<object classic="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    Code base="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,0"
    width="600" height="400" ID="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML 页也可以使用脚本生成 SWF 嵌入标签。您需要更改脚本以使其插入正确的 allowNetworking 设置。Adobe Flash Professional 和 Adobe Flash Builder 生成的 HTML 页面使用 AC_FL_RunContent() 函数嵌入对 SWF 文件的引用。向此脚本添加 allowNetworking 参数设置，如下所示：

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

当 allowNetworking 设置为 "internal" 时，下列 API 被阻止：

navigateToURL()、 fscommand()、 ExternalInterface.call()

当 allowNetworking 设置为 "none" 时，除了上面所列的 API 外，还会禁止下列 API：

sendToURL()、 FileReference.download()、 FileReference.upload()、 Loader.load()、 LocalConnection.connect()、 LocalConnection.send()、 NetConnection.connect()、 NetStream.play()、 Security.loadPolicyFile()、 SharedObject.getLocal()、 SharedObject.getRemote()、 Socket.connect()、 Sound.load()、 URLLoader.load()、 URLStream.load()、 XMLSocket.connect()

即使所选的 allowNetworking 设置允许 SWF 文件使用网络 API，可能仍有其他基于安全沙箱限制的限制 （请参阅第 895 页的 “ 安全沙箱 ”）。

当 allowNetworking 设置为 "none" 时，无法在 TextField 对象 htmlText 属性的 <img> 标签中引用外部媒体 （这会引发 SecurityError 异常）。

当 allowNetworking 设置为 "none" 时，在 Flash Professional（而不是 ActionScript）中添加的导入的共享库中的元件在运行时被阻止。

全屏模式安全性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 9.0.27.0 和更高版本支持全屏模式，在该模式中， Flash Player 中运行的内容可以填满整个屏幕。要进入全屏模式，需将 Stage 的 displayState 属性设置为 StageDisplayState.FULL_SCREEN 常量。有关详细信息，请参阅第 139 页的 “ 使用全屏模式 ”。

对于在远程沙箱中运行的 SWF 文件，存在一些安全注意事项。

要启用全屏模式，请在包含 SWF 文件引用的 HTML 页的 <object> 和 <embed> 标签中添加 allowFullScreen 参数，并将参数值设置为 "true" （默认值为 "false"），如下例所示：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML 页也可以使用脚本生成 SWF 嵌入标签。您必须更改该脚本，以便让它能够插入适当的 allowFullScreen 设置。 Flash Professional 和 Flash Builder 生成的 HTML 页面使用 AC_FL_RunContent() 函数嵌入对 SWF 文件的引用，您需要添加 allowFullScreen 参数设置，如下所示：

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

仅当在响应鼠标事件或键盘事件时才会调用启动全屏模式的 ActionScript。如果在其他情况中调用， Flash Player 会引发异常。

当内容进入全屏模式时，程序会显示一条消息，指导用户如何退出和返回标准模式。该消息将显示几秒钟，然后淡出。

对于在浏览器中运行的内容，在全屏模式下将限制使用键盘。在 Flash Player 9 中，只支持使应用程序返回正常模式 （如通过按 Escape 键）的快捷键。用户不能在文本字段中输入文本，也不能在屏幕上导航。在 Flash Player 10 和更高版本中，支持某些非打印键 （具体来说是方向键、空格键和 Tab 键）。但仍禁止文本输入。

在独立的播放器或放映文件中始终允许全屏模式。此外，这些环境中还完全支持使用键盘 （包括文本输入）。

调用 Stage 对象的 displayState 属性会因任何调用方没有与 Stage 所有者（主 SWF 文件）位于同一安全沙箱而引发异常。有关详细信息，请参阅 第 912 页的 “Stage 安全性 ”。

管理员可以通过在 mms.cfg 文件中设置 FullScreenDisable = 1 对浏览器中运行的 SWF 文件禁用全屏模式。有关详细信息，请参阅第 899 页的 “ 管理员控制 ”。

在浏览器中，必须在 HTML 页面中包含 SWF 文件，才能进入全屏模式。

全屏交互模式安全性

Flash Player 11.3 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 11.3 和更高版本支持全屏交互模式，在该模式中， Flash Player 中运行的内容可以填满整个屏幕并接受文本输入。若要进入全屏交互模式，需要将 Stage 的 displayState 属性设置为 StageDisplayState.FULL_SCREEN_INTERACTIVE 常量。有关详细信息，请参阅第 139 页的 " 使用全屏模式 "。

对于在远程沙箱中运行的 SWF 文件，存在一些安全注意事项。

若要启用全屏模式，请在包含 SWF 文件引用的 HTML 页中的 <object> 和 <embed> 标签中添加 allowFullScreenInteractive 参数，并将其值设置为 "true" （默认值为 "false"），如下例所示：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,18,0"
    width="600" height="400" id="test" align="middle">
<param name="allowFullScreenInteractive" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
    width="600" height="400"
    name="test" align="middle" type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML 页也可以使用脚本生成 SWF 嵌入标签。您必须更改该脚本，以便它能够插入适当的 allowFullScreenInteractive 设置。Flash Professional 和 Flash Builder 生成的 HTML 页使用 AC_FL_RunContent() 函数嵌入对 SWF 文件的引用，并且您需要添加 allowFullScreenInteractive 参数设置，如下所示：

```
AC_FL_RunContent( ... "allowFullScreenInteractive", "true", ...)
```

仅当在响应鼠标事件或键盘事件时，才能调用启动全屏交互模式的 ActionScript。如果在其他情况中调用， Flash Player 会引发异常。

当内容进入全屏交互模式时，会出现叠加消息。此消息显示全屏页面的域、有关如何退出全屏模式的说明以及 " 允许 " 按钮。叠加持续到用户单击 " 允许 " 时，确认他们处于全屏交互模式。

管理员可以通过在 mms.cfg 文件中设置 FullScreenInteractiveDisable = 1 对浏览器中运行的 SWF 文件禁用全屏交互模式。有关详细信息，请参阅第 899 页的 " 管理员控制 "。

在浏览器中， SWF 文件必须包含在 HTML 页中才能进入全屏交互模式。

加载内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 和 AIR 内容可加载多种类型的其他内容，其中包括：

- SWF 文件
- 图像
- Sound
- 视频
- HTML 文件 （仅限 AIR)
- JavaScript （仅限 AIR)

使用 Loader 类加载 SWF 文件和图像

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

使用 Loader 类加载 SWF 文件和图像（JPG、GIF 或 PNG 文件）。除只能与本地文件系统内容交互的沙箱中的 SWF 文件之外，其他所有 SWF 文件都可以从任何网络域加载 SWF 文件和图像。只有本地沙箱中的 SWF 文件才能从本地文件系统中加载 SWF 文件和图像。但是，只能与远程内容交互的沙箱中的文件只能加载位于受信任的本地沙箱或只能与远程内容交互的沙箱中的本地 SWF 文件。只能与远程内容交互的沙箱中的 SWF 文件可加载非 SWF 文件（例如图像）的本地内容，但是无法访问所加载内容中的数据。

从不受信任的来源 （如 Loader 对象的根 SWF 文件所在域以外的域）加载 SWF 文件时，您可能需要为 Loader 对象定义遮罩，以防止加载的内容 （Loader 对象的子级）绘制到该遮罩之外的 Stage 部分中，如以下代码所示：

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

当调用 Loader 对象的 load() 方法时，可以指定一个 context 参数，该参数是一个 LoaderContext 对象。LoaderContext 类包括三个属性，用于定义如何使用加载的内容的上下文：

- checkPolicyFile：仅当加载图像文件 （不是 SWF 文件）时才会使用此属性。如果图像文件所在的域与包含 Loader 对象的文件所在的域不同，则指定此属性。如果将此属性设置为 true， Loader 将检查 URL 策略文件的原始服务器 （请参阅第 901 页的 “ 网站控制 （策略文件） ”）。如果服务器授予 Loader 域适当权限，则来自 Loader 域中 SWF 文件的 ActionScript 可以访问所加载图像中的数据。换言之，您可以使用 Loader.content 属性获取对表示所加载图像的 Bitmap 对象的引用，或者使用 BitmapData.draw() 或 BitmapData.drawWithQuality() 方法访问所加载图像中的像素。drawWithQuality 方法可用于 Flash Player 11.3 和更高版本、 AIR 3.3 和更高版本。
- securityDomain：仅当加载 SWF 文件 （不是图像）时才会使用此属性。如果 SWF 文件所在的域与包含 Loader 对象的文件所在的域不同，则指定此属性。对于 securityDomain 属性而言，目前仅支持以下两个值：null （默认值）和 SecurityDomain.currentDomain。如果指定 SecurityDomain.currentDomain，则要求加载的 SWF 文件应 “ 导入 ” 到执行加载的 SWF 文件所在的沙箱中，这意味着其运行方式就像它已从执行加载的 SWF 文件自己的服务器中加载一样。只有在位于加载的 SWF 文件的服务器上找到 URL 策略文件时才允许这样做，从而允许执行加载的 SWF 文件所在的域进行访问。如果找到所需的策略文件，则一旦加载开始，加载方和被加载方可以自由地互相访问脚本，原因是它们位于同一沙箱中。请注意，多数情况可以通过执行普通加载操作然后让加载的 SWF 文件调用 Security.allowDomain() 方法来取代沙箱导入。由于加载的 SWF 文件将位于自己的原始沙箱中，并因而能够访问自己实际服务器上的资源，因此后一种方法会更易于使用。
- applicationDomain: 仅当加载使用 ActionScript 3.0 编写的 SWF 文件（不是图像或使用 ActionScript 1.0 或 2.0 编写的 SWF 文件）时才会使用此属性。当加载文件时，可以指定文件应放置在特定的应用程序域中，而不是默认放置在一个新的应用程序域中，这个新的应用程序域是执行加载的 SWF 文件所在应用程序域的子域。请注意，应用程序域是安全域的子单位，因此仅当要加载的 SWF 文件由于以下原因来自您自己的安全域时，才能指定目标应用程序域：该文件来自您自己的服务器，或者使用 securityDomain 属性已成功地将该文件导入到您的安全域中。如果指定应用程序域，但加载的 SWF 文件属于其他安全域，则在 applicationDomain 中指定的域将被忽略。有关详细信息，请参阅第 123 页的 “ 使用应用程序域 ”。

有关详细信息，请参阅第 166 页的 “ 指定加载上下文 ”。

Loader 对象的一个重要属性就是 contentLoaderInfo 属性，该属性是一个 LoaderInfo 对象。与大部分对象不同，LoaderInfo 对象在执行加载的 SWF 文件和被加载的内容之间共享，并且双方始终可以访问该对象。当被加载的内容为 SWF 文件时，它可以通过 DisplayObject.loaderInfo 属性访问 LoaderInfo 对象。 LoaderInfo 对象包括诸如加载进度、加载方和被加载方的 URL、加载方和被加载方之间的信任关系等信息及其他信息。有关详细信息，请参阅第 165 页的 “ 监视加载进度 ”。

加载声音和视频

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

允许任何内容使用 Sound.load()、NetConnection.connect() 和 NetStream.play() 方法从网络源加载声音和视频，只能与本地文件系统内容交互的沙箱中的内容除外。

只有只能与本地文件系统内容交互的沙箱和 AIR 应用程序沙箱中的内容可以从本地文件系统中加载媒体。只有只能与本地文件系统内容交互的沙箱、 AIR 应用程序沙箱或受信任的本地沙箱中的内容可以访问这些加载的文件中的数据。

对加载的媒体还存在一些其他数据访问限制。有关详细信息，请参阅第 913 页的 " 作为数据访问加载的媒体 "。

使用文本字段中的 <img> 标签加载 SWF 文件和图像

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过使用 <img> 标签，可以将 SWF 文件和位图加载到文本字段中，如以下代码所示：

```
<img src = 'filename.jpg' id = 'instanceName' >
```

通过使用 TextField 实例的 getImageReference() 方法，可以访问以这种方式加载的内容，如以下代码所示：

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

但是请注意，以这种方式加载的 SWF 文件和图像会被放入与各自来源相应的沙箱中。

当在文本字段中使用 <img> 标签加载图像文件时，通过 URL 策略文件可以允许访问图像中的数据。通过将 checkPolicyFile 属性添加到 <img> 标签上，可以检查是否存在策略文件，如以下代码所示：

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

当在文本字段中使用 <img> 标签加载 SWF 时，可以允许通过调用 Security.allowDomain() 方法来访问该 SWF 文件的数据。

当在文本字段中使用 <img> 标签加载外部文件时 （相对于使用嵌在 SWF 文件中的 Bitmap 类），会自动创建一个 Loader 对象作为 TextField 对象的子对象，并且会将外部文件加载到该 Loader 对象中，就如同使用了 ActionScript 中的 Loader 对象来加载文件一样。在这种情况下，getImageReference() 方法返回自动创建的 Loader。由于此 Loader 对象与调用代码位于同一安全沙箱中，因此访问此对象不需要任何安全检查。

但是，当引用 Loader 对象的 content 属性来访问加载的媒体时，需要应用安全规则。如果内容是图像，则需要实现 URL 策略文件；如果内容是 SWF 文件，则需要让 SWF 文件中的代码调用 allowDomain() 方法。

Adobe AIR

在应用程序沙箱内，将忽略文本字段中的 <img> 标签以防止仿冒攻击。此外，不允许在应用程序沙箱中运行的代码调用 Security allowDomain() 方法。

使用 RTMP 服务器传送的内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Media Server 使用实时媒体协议 (RTMP) 提供数据、音频和视频。您可以使用 NetConnection 类的 connect() 方法将 RTMP URL 作为参数传递来加载此媒体。 Flash Media Server 可以根据所请求文件的域来限制连接并防止内容被下载。有关详细信息，请参阅 www.adobe.com/go/learn_fms_docs_cn 上提供的在线 Flash Media Server 文档。

若要使用 BitmapData.draw()、BitmapData.drawWithQuality() 和 SoundMixer.computeSpectrum() 方法从 RTMP 流提取运行时图形和声音数据，您必须授予对服务器的访问权限。使用服务器端 ActionScript Client.videoSampleAccess 和 Client.audioSampleAccess 属性允许访问 Flash Media Server 上的特定目录。有关详细信息，请参阅 Server-Side ActionScript Language Reference。（drawWithQuality 方法可用于 Flash Player 11.3 和更高版本、AIR 3.3 和更高版本。）

跨脚本访问

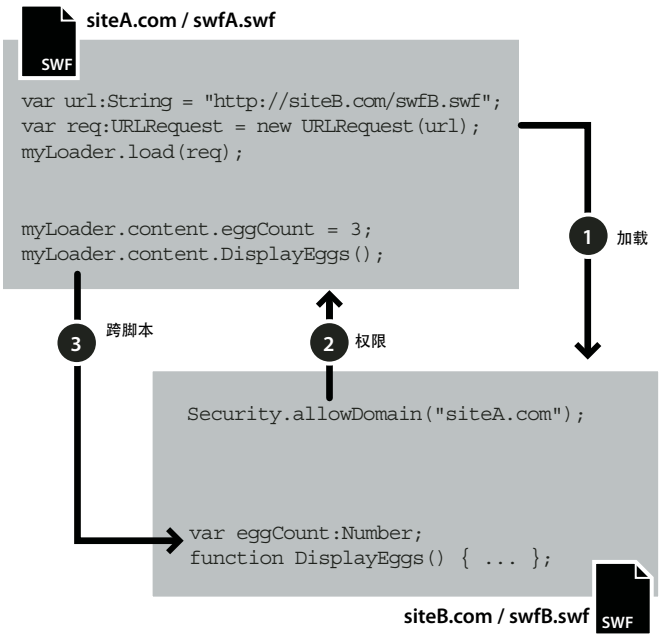
Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

如果使用 ActionScript 3.0 编写的两个 SWF 文件或在 AIR 中运行的两个 HTML 文件来自同一域（例如，一个 SWF 文件的 URL 是 http://www.example.com/swfA.swf，另一个的是 http://www.example.com/swfB.swf），则在其中一个文件中定义的代码可检查并修改另一个文件中的变量、对象、属性、方法等，反之亦然。这称为“跨脚本访问”。

如果这两个文件来自不同的域（例如， http://siteA.com/swfA.swf 和 http://siteB.com/swfB.swf），那么，在默认情况下，Flash Player 和 AIR 不允许 swfA.swf 编写 swfB.swf 的脚本，也不允许 swfB.swf 编写 swfA.swf 的脚本。通过调用 Security.allowDomain()，一个 SWF 文件可授予其他域中的 SWF 文件编写其脚本的权限。通过调用 Security.allowDomain("siteA.com")， swfB.swf 向来自 siteA.com 的 SWF 文件授予访问其脚本的权限。

在 AVM1 SWF 文件和 AVM2 SWF 文件之间不支持跨脚本访问。 AVM1 SWF 文件是使用 ActionScript 1.0 或 ActionScript 2.0 创建的文件。（AVM1 和 AVM2 指的是 ActionScript 虚拟机。）但是，可以使用 LocalConnection 类在 AVM1 和 AVM2 之间发送数据。

在任何跨域的情况下，明确所涉及的双方非常重要。为了便于进行此讨论，我们将执行跨脚本访问的一方称为“访问方”（通常是执行访问的 SWF），将另一方称为“被访问方”（通常是被访问的 SWF）。当 siteA.swf 访问 siteB.swf 的脚本时，siteA.swf 是访问方， siteB.swf 是被访问方，如下图所示：



使用 Security.allowDomain() 方法建立的跨域权限是不对称的。在上例中，siteA.swf 可以访问 siteB.swf 的脚本，但 siteB.swf 无法访问 siteA.swf 的脚本，这是因为 siteA.swf 未调用 Security.allowDomain() 方法来授予 siteB.com 中的 SWF 文件访问其脚本的权限。可以通过使两个 SWF 文件都调用 Security.allowDomain() 方法来设置对称权限。

除了防止 SWF 文件受到其他 SWF 文件发起的跨域脚本编写外，Flash Player 还防止 SWF 文件受到 HTML 文件发起的跨域脚本编写。可以通过 ExternalInterface.addCallback() 方法建立的回调执行 HTML 到 SWF 的脚本访问。当 HTML 到 SWF 的脚本访问跨域时，被访问的 SWF 文件必须调用 Security.allowDomain() 方法 （这与访问方是 SWF 文件时一样），否则操作将失败。有关详细信息，请参阅第 904 页的 " 作者 （开发人员）控制 "。

此外， Flash Player 还对 SWF 到 HTML 的脚本访问提供安全控制。有关详细信息，请参阅第 919 页的 " 控制外出 URL 访问 "。

Stage 安全性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Stage 对象的某些属性和方法可用于显示列表中的任何 sprite 或影片剪辑。

但是，我们说 Stage 对象有一个所有者，即加载的第一个 SWF 文件。默认情况下，Stage 对象的以下属性和方法只能用于与舞台所有者位于同一安全沙箱中的 SWF 文件：

属性	方法
align	addChild()
displayState	addChildAt()
frameRate	addEventListener()
height	dispatchEvent()
mouseChildren	hasEventListener()
numChildren	setChildIndex()
quality	willTrigger()
scaleMode	
showDefaultContextMenu	
stageFocusRect	
stageHeight	
stageWidth	
tabChildren	
textSnapshot	
width	

为使与 Stage 所有者不在同一沙箱中的 SWF 文件能够访问这些属性和方法，舞台所有者 SWF 文件必须调用 Security.allowDomain() 方法来允许外部沙箱的域。有关详细信息，请参阅第 904 页的 " 作者 （开发人员）控制 "。

frameRate 属性是一种特殊情况：任何 SWF 文件均能读取 frameRate 属性。但是，只有位于 Stage 所有者的安全沙箱中的文件（或通过调用 Security.allowDomain() 方法被授予权限的文件）才能更改该属性。

此外，对于 Stage 对象的 removeChildAt() 和 swapChildrenAt() 方法还存在一些限制，但是这些限制与其他限制不同。要调用这些方法，不需要与 Stage 所有者位于同一域中，而是代码必须与受影响的子对象位于同一域中，或者子对象可以调用 Security.allowDomain() 方法。

遍历显示列表

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

一个 SWF 文件能够访问从其他沙箱中加载的显示对象受到一定限制。为使 SWF 文件能够访问由其他沙箱中的另一个 SWF 文件创建的显示对象，被访问的 SWF 文件必须调用 Security.allowDomain() 方法向进行访问的 SWF 文件所在的域授予访问权限。有关详细信息，请参阅第 904 页的 " 作者 （开发人员）控制 "。

要访问由 Loader 对象加载的 Bitmap 对象，图像文件的原始服务器上必须存在 URL 策略文件，并且该 URL 策略文件必须为尝试访问该 Bitmap 对象的 SWF 文件所在的域授予访问权限 （请参阅第 901 页的 " 网站控制 （策略文件） "）。

与加载的文件 （和 Loader 对象）相对应的 LoaderInfo 对象包括以下三种属性 （定义加载的对象和 Loader 对象之间的关系）： childAllowsParent、 parentAllowsChild 和 sameDomain。

事件安全性

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

根据调度事件的显示对象所在的沙箱，与显示列表相关的事件具有一定的安全性访问限制。显示列表中的事件具有冒泡阶段和捕获阶段 （在第 106 页的 " 处理事件 " 中介绍）。在冒泡阶段和捕获阶段期间，事件从源显示对象开始迁移，并经过显示列表中的父显示对象。如果父对象与源显示对象位于不同的安全沙箱中，则捕获阶段和冒泡阶段在父对象的下方停止，除非父对象的所有者与源对象的所有者互相信任。这种互相信任可以通过以下方式来建立：

1 拥有父对象的 SWF 文件必须调用 Security.allowDomain() 方法，以信任拥有源对象的 SWF 文件所在的域。

2 拥有源对象的 SWF 文件必须调用 Security.allowDomain() 方法，以信任拥有父对象的 SWF 文件所在的域。

与加载的文件 （和 Loader 对象）相对应的 LoaderInfo 对象包括以下两种属性 （定义加载的对象和 Loader 对象之间的关系）： childAllowsParent 和 parentAllowsChild。

对于从显示对象以外的对象调度的事件，不存在任何安全检查或与安全相关的含义。

作为数据访问加载的媒体

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

若要访问加载数据，请使用 BitmapData.draw()、 BitmapData.drawWithQuality() 和 SoundMixer.computeSpectrum() 方法。默认情况下，您无法从其他沙箱中加载的媒体呈现或播放的图形或音频对象中获取像素数据或音频数据。但是，您可以使用下列方法授予跨沙箱边界访问此类数据的权限：

- 在呈现或播放要访问的数据的内容中，调用 Security.allowDomain() 方法以授予对其他域中的内容的数据访问权限。
- 对于加载的图像、声音或视频，在被加载文件所在的服务器上添加 URL 策略文件。此策略文件必须向试图调用 BitmapData.draw()、 BitmapData.drawWithQuality() 或 SoundMixer.computeSpectrum() 方法的 SWF 文件所在的域授予访问权限，才能从文件中提取数据。 drawWithQuality 方法可用于 Flash Player 11.3 和更高版本、 AIR 3.3 和更高版本。

以下各节提供有关访问位图、声音和视频数据的详细信息。

访问位图数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

借助 BitmapData 对象的 draw() 和 drawWithQuality() （Flash Player 11.3、 AIR 3.3）方法，可以将任何显示对象的当前显示像素绘制到 BitmapData 对象。这可能包括 MovieClip 对象、Bitmap 对象或任何显示对象的像素。若要使用这些方法将像素绘制到 BitmapData 对象，必须满足以下条件：

- 如果是除所加载位图之外的源对象，则该源对象及其（如果是 Sprite 或 MovieClip 对象）所有子对象必须与调用 draw 方法的对象位于同一域中，或者它们必须位于调用方通过调用 Security.allowDomain() 方法可访问的 SWF 文件中。
- 如果是已加载的位图源对象，则该源对象必须与调用 draw 方法的对象位于同一域中，或者其源服务器必须包含一个授予调用域访问权限的 URL 策略文件。

如果不满足上述条件，则会引发 SecurityError 异常。

当使用 Loader 类的 load() 方法加载图像时，可以指定一个 context 参数，该参数是一个 LoaderContext 对象。如果将 LoaderContext 对象的 checkPolicyFile 属性设为 true，则 Flash Player 将在从其中加载图像的服务器上查找 URL 策略文件。如果存在策略文件且该文件允许执行加载的 SWF 文件所在的域进行访问，则会允许该文件访问 Bitmap 对象中的数据，否则就不允许。

此外，还可以通过文本字段中的 <img> 标签在加载的图像中指定 checkPolicyFile 属性。有关详细信息，请参阅第 910 页的 “ 使用文本字段中的 <img> 标签加载 SWF 文件和图像 ”。

访问声音数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

以下与声音相关的 ActionScript 3.0 API 存在一些安全限制：

- SoundMixer.computeSpectrum() 方法 — 对于与声音文件在同一安全沙箱中运行的代码，始终允许使用此方法。对于在其他沙箱中运行的代码，需要进行安全检查。
- SoundMixer.stopAll() 方法 — 对于与声音文件在同一安全沙箱中运行的代码，始终允许使用此方法。对于其他沙箱中的文件，则需经过安全检查。
- Sound 类的 id3 属性 — 对于与声音文件位于同一安全沙箱的 SWF 文件，始终允许使用该属性。对于在其他沙箱中运行的代码，需要进行安全检查。

每个声音都具有两种与之关联的沙箱 （一个内容沙箱和一个所有者沙箱）：

- 声音的源域确定内容沙箱，内容沙箱则确定是否可以通过声音的 id3 属性和 SoundMixer.computeSpectrum() 方法提取声音中的数据。
- 启动声音播放的对象确定所有者沙箱，所有者沙箱则确定是否可以使用 SoundMixer.stopAll() 方法停止声音。

当使用 Sound 类的 load() 方法加载声音时，可以指定一个 context 参数，该参数是一个 SoundLoaderContext 对象。如果将 SoundLoaderContext 对象的 checkPolicyFile 属性设置为 true，运行时会在从其加载声音的服务器上检查是否存在 URL 策略文件。如果存在策略文件，并且该文件支持加载代码的域，则允许该代码访问 Sound 对象的 id 属性；否则，将不允许。此外，设置 checkPolicyFile 属性可以为加载的声音启用 SoundMixer.computeSpectrum() 方法。

可以使用 SoundMixer.areSoundsInaccessible() 方法确定对 SoundMixer.stopAll() 方法的调用是否会因调用方无法访问一个或多个声音所有者的沙箱而停止全部声音。

调用 SoundMixer.stopAll() 方法会停止与 stopAll() 的调用方位于同一所有者沙箱中的那些声音。它还会停止由调用 Security.allowDomain() 方法的 SWF 文件来启动播放的声音，以允许调用 stopAll() 方法的 SWF 文件所在的域进行访问。任何其他声音均不会停止，可以通过调用 SoundMixer.areSoundsInaccessible() 方法来确定此类声音是否存在。

调用 computeSpectrum() 方法要求播放的每个声音应与调用该方法的对象位于同一沙箱中，或者位于已向调用方的沙箱授予权限的源；否则会引发 SecurityError 异常。对于从 SWF 文件库中嵌入声音加载的声音，通过在加载的 SWF 文件中调用 Security.allowDomain() 方法授予权限。对于从非 SWF 文件的源 （源自加载的 mp3 文件或视频文件）中加载的声音，源服务器上的 URL 策略文件将授予访问所加载媒体中数据的权限。

有关详细信息，请参阅第 904 页的 " 作者 （开发人员）控制 " 和第 901 页的 " 网站控制 （策略文件） "。

要从 RTMP 流访问声音数据，必须拥有对服务器的访问权限。使用服务器端 ActionScript Client.audioSampleAccess 属性允许访问 Flash Media Server 上的特定目录。有关详细信息，请参阅 Server-Side ActionScript Language Reference。

访问视频数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以使用 BitmapData.draw() 或 BitmapData.drawWithQuality() 方法捕获当前视频帧中的像素数据。（drawWithQuality 方法可用于 Flash Player 11.3 和更高版本、 AIR 3.3 和更高版本。）

视频有两种不同形式：

- 从 Flash Media Server 通过 RTMP 流式传输的视频
- 从 FLV 或 F4V 文件加载的渐进式视频

若要使用 draw 方法从 RTMP 流提取运行时图形，您必须授予对服务器的访问权限。使用服务器端 ActionScript Client.videoSampleAccess 属性允许访问 Flash Media Server 上的特定目录。有关详细信息，请参阅 Server-Side ActionScript Language Reference。

当调用 draw 方法，并且 source 参数为渐进式视频时，该方法的调用方必须与 FLV 文件位于同一沙箱中，或者 FLV 文件所在的服务器上必须存在一个策略文件，用于授予对调用 SWF 文件所在域的访问权限。通过将 NetStream 对象的 checkPolicyFile 属性设置为 true，可以请求下载该策略文件。

加载数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 和 AIR 内容可以与服务器交换数据。加载数据是一种与加载媒体不同的操作，因为加载的信息以程序对象的形式显示，而不是以媒体的形式显示。通常，内容可能从内容源自的同一个域加载数据。但是，内容通常需要策略文件以便从其他域加载数据 （请参阅第 901 页的 " 网站控制 （策略文件） "）。

注：AIR 应用程序沙箱中运行的内容无法由远程域提供 （除非开发人员特意将远程内容导入应用程序沙箱），因此该内容无法参与策略文件防范的攻击类型。应用程序沙箱中的 AIR 内容加载数据时不受策略文件限制。但是，其他沙箱中的 AIR 内容受此处说明的限制的约束。

使用 URLLoader 和 URLStream

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

可以加载诸如 XML 文件或文本文件等数据。 URLLoader 和 URLStream 类的 load() 方法受到 URL 策略文件权限的控制。

如果您使用 load() 方法从域加载内容而不是调用此方法的代码的内容，则运行时会检查加载资源的服务器上是否存在 URL 策略文件。如果存在策略文件，并且该策略文件授予对加载内容的域的访问权限，则可以加载数据。

连接到套接字

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

默认情况下，运行时查找端口 843 提供的套接字策略文件。和 URL 策略文件一样，此文件称为 " 主策略文件 "。

在 Flash Player 6 中首次引入策略文件时，并不支持套接字策略文件。与套接字服务器的连接由默认位置中的一个策略文件授权，该默认位置是套接字服务器所在主机的 HTTP 服务器的端口 80。Flash Player 9 仍支持此功能，但 Flash Player 10 不再支持。在 Flash Player 10 中，只有套接字策略文件可授权套接字连接。

与 URL 策略文件类似，套接字策略文件支持元策略语句 （用于指定可提供策略文件的端口）。但是，套接字策略文件的默认元策略不是 "master-only" 而是 "all"。即，除非主策略文件指定更具限制性的设置，否则 Flash Player 假定主机上的任意套接字均可提供套接字策略文件。

默认情况下禁止访问套接字和 XML 套接字连接，即使所要连接的套接字与 SWF 文件位于同一个域中也是如此。可以从下列任意位置提供套接字策略文件，从而允许套接字级别访问：

- 端口 843 （主策略文件的位置）
- 与主套接字连接相同的端口
- 主套接字连接端口之外的端口

默认情况下， Flash Player 在端口 843 和主套接字连接所在的端口上查找套接字策略文件。如果要从其他端口提供套接字策略文件， SWF 文件必须调用 Security.loadPolicyFile()。

套接字策略文件具有与 URL 策略文件相同的语法，只是前者还必须指定要对哪些端口授予访问权限。如果套接字策略文件来自低于 1024 的端口号，则它可以对任何端口授予访问权限；如果策略文件来自 1024 或更高的端口，则它只能对 1024 端口和更高的端口授予访问权限。允许的端口在 <allow-access-from> 标签中的 to-ports 属性中指定。单个端口号、端口范围和通配符都是允许值。

下面是套接字策略文件的示例：

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- Policy file for xmlsocket://socks.mysite.com -->
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="507" />
    <allow-access-from domain="*.example.com" to-ports="507,516" />
    <allow-access-from domain="*.example.org" to-ports="516-523" />
    <allow-access-from domain="adobe.com" to-ports="507,516-523" />
    <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

要从端口 843 或主套接字连接所在的端口上检索套接字策略文件，请调用 Socket.connect() 或 XMLSocket.connect() 方法。Flash Player 首先在端口 843 上检查是否有主策略文件。如果找到主策略文件，Flash Player 将检查该文件是否含有在目标端口上禁止套接字策略文件的元策略语句。如果未禁止访问， Flash Player 首先在主策略文件中查找适当的 allow-access-from 语句。如果找不到主策略文件， Flash Player 将从主套接字连接所在的端口上查找套接字策略文件。

要从其他位置检索套接字策略文件，请首先调用 Security.loadPolicyFile() 方法，调用时使用特殊的 "xmlsocket" 语法，如下所示：

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

先调用 Security.loadPolicyFile() 方法，然后再调用 Socket.connect() 或 XMLSocket.connect() 方法。 Flash Player 随后将一直等待完成策略文件请求，之后再决定是否允许主连接。但是，如果主策略文件指定目标位置不能提供策略文件，则调用 loadPolicyFile() 无效，即使该位置有策略文件。

如果要实现套接字服务器，并且需要提供套接字策略文件，则应决定是使用接受主连接的同一端口提供策略文件，还是使用不同的端口来提供策略文件。无论是哪种情况，服务器都必须等到客户端的第一次传输收到之后才能发送响应。

当 Flash Player 请求策略文件时，它始终会在建立连接后传输以下字符串：

```
<policy-file-request/>
```

服务器收到此字符串后，即会传输该策略文件。来自 Flash Player 的请求始终由 null 字节终止，来自服务器的响应也必须由 null 字节终止。

程序对于策略文件请求和主连接并不会使用同一连接，因此请在传输策略文件后关闭连接。如果不关闭连接， Flash Player 将关闭策略文件连接，之后重新连接以建立主连接。

保护数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

要在数据遍历 Internet 时保护其免于被窃听和更改，您可以在作为数据来源的服务器上使用传输层安全性 (TLS) 或套接字层安全性 (SSL)。然后，可以使用 HTTPS 协议连接到该服务器。

在为 AIR 2 或更高版本创建的应用程序中，还可以保护 TCP 套接字通信。通过 SecureSocket 类，您可以启动到使用 TLS 版本 1 或 SSL 版本 4 的套接字服务器的套接字连接。

发送数据

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当代码向服务器或资源发送数据时，会执行数据发送操作。对于来自网络域的内容，始终允许发送数据。仅当本地 SWF 文件位于受信任的本地沙箱、只能与远程内容交互的沙箱或 AIR 应用程序沙箱时，它才可以向网络地址发送数据。有关详细信息，请参阅第 896 页的 " 本地沙箱 "。

可以使用 flash.net.sendToURL() 函数向 URL 发送数据。还可以使用其他方法向 URL 发送请求。这些方法包括 Loader.load() 和 Sound.load() 等加载方法以及 URLLoader.load() 和 URLStream.load() 等数据加载方法。

上载和下载文件

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

FileReference.upload() 方法可以将用户选择的文件上载到远程服务器。必须先调用 FileReference.browse() 或 FileReferenceList.browse() 方法，然后再调用 FileReference.upload() 方法。

仅在响应鼠标事件或键盘事件时才可以调用用于启动 FileReference.browse() 或 FileReferenceList.browse() 方法的代码。如果在其他情况下调用，Flash Player 10 和更高版本会引发异常。但是，从 AIR 应用程序沙箱调用这些方法不需要用户启动的事件。

调用 FileReference.download() 方法可打开一个对话框，用户可以在该对话框中从远程服务器下载文件。

注：如果服务器要求用户身份验证，则只有在浏览器中运行的 SWF 文件 （即使用浏览器插件或 ActiveX 控件的文件）才可以提供对话框来提示用户输入用户名和密码以进行身份验证，并且只适用于下载。 Flash Player 不允许上载到需要用户身份验证的服务器。

如果执行调用的 SWF 文件位于只能与本地文件系统内容交互的沙箱中，则不允许执行上载和下载操作。

默认情况下， SWF 文件不会在自身所在服务器之外的服务器上执行上载或下载操作。如果其他服务器提供策略文件向执行调用的 SWF 文件所在的域授予访问权限，则执行调用的 SWF 文件可以在其他服务器上执行上载或下载操作。

从导入到安全域的 SWF 文件加载嵌入内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当加载 SWF 文件时，可以设置用于加载文件的 Loader 对象的 load() 方法中的 context 参数。此参数是一个 LoaderContext 对象。将此 LoaderContext 对象的 securityDomain 属性设置为 Security.currentDomain 时，Flash Player 将在被加载 SWF 文件所在的服务器上检查是否存在 URL 策略文件。如果存在策略文件，并且该文件向执行加载的 SWF 文件所在的域授予访问权限，则可以作为导入媒体加载 SWF 文件。这样，执行加载的文件可以获得对 SWF 文件的库中对象的访问权限。

SWF 文件访问其他安全沙箱中被加载 SWF 文件的类的另一种方法是：使被加载的 SWF 文件调用 Security.allowDomain() 方法，以向执行调用的 SWF 文件所在的域授予访问权限。可以将对 Security.allowDomain() 方法的调用添加到被加载 SWF 文件的主类的构造函数方法中，然后使执行加载的 SWF 文件添加事件侦听器，以便响应由 Loader 对象的 contentLoaderInfo 属性调度的 init 事件。当调度此事件时，被加载的 SWF 文件已经调用构造函数方法中的 Security.allowDomain() 方法，因此被加载 SWF 文件中的类可用于执行加载的 SWF 文件。正在加载的 SWF 文件可以从已加载的 SWF 文件检索类，方法是调用 Loader.contentLoaderInfo.applicationDomain.getDefinition() 或 Loader.contentLoaderInfo.applicationDomain.getQualifiedDefinitionNames()（Flash Player 11.3 和更高版本、AIR 3.3 和更高版本）。

使用旧内容

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

在 Flash Player 6 中，用于某些 Flash Player 设置的域基于 SWF 文件所在的域的末尾部分。这些设置包括对摄像头和麦克风访问权限、存储配额及永久共享对象存储的设置。

如果 SWF 文件所在的域包含的段数超过两个 （如 www.example.com），则会去除该域的第一段 (www)，并使用该域的剩余部分。因此，在 Flash Player 6 中，www.example.com 和 store.example.com 都使用 example.com 作为这些设置的域。同样，www.example.co.uk 和 store.example.co.uk 都使用 example.co.uk 作为这些设置的域。这样会导致出现问题，使得来自不相关域 （如 example1.co.uk 和 example2.co.uk）的 SWF 文件可以访问相同的共享对象。

在 Flash Player 7 及更高版本中，默认情况下是根据 SWF 文件的精确域来选择播放器设置。例如，来自 www.example.com 的 SWF 文件将使用 www.example.com 的播放器设置，来自 store.example.com 的 SWF 文件将使用 store.example.com 的不同播放器设置。

在使用 ActionScript 3.0 编写的 SWF 文件中，当 Security.exactSettings 设置为 true （默认值）时， Flash Player 将针对精确域使用播放器设置。当它设置为 false 时，Flash Player 使用在 Flash Player 6 中使用的域设置。如果将 exactSettings 更改为使用其他值而不是默认值，则必须在要求 Flash Player 选择播放器设置的任何事件 （例如，使用摄像头或麦克风，或者检索永久共享对象）发生之前执行此操作。

如果发布了版本 6 的 SWF 文件并通过该版本创建了永久共享对象，则要从使用 ActionScript 3.0 编写的 SWF 中检索这些永久共享对象，必须先将 Security.exactSettings 设置为 false，然后再调用 SharedObject.getLocal()。

设置 LocalConnection 权限

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

借助 LocalConnection 类，您可以在一个 Flash Player 或 AIR 应用程序与另一个 Flash Player 或 AIR 应用程序之间发送消息。LocalConnection 对象仅可以在运行在同一台客户端计算机上的 Flash Player 或 AIR 内容之间进行通信，但是它们可以在不同的应用程序中运行，例如，在浏览器中运行的 SWF 文件、在放映文件中运行的 SWF 文件以及 AIR 应用程序都可以使用 LocalConnection 类进行通信。

对于每个 LocalConnection 通信，都存在发送方和侦听器。默认情况下， Flash Player 允许在同一域中运行的代码之间进行 LocalConnection 通信。对于在不同的沙箱中运行的代码，侦听器必须允许使用 LocalConnection.allowDomain() 方法授予发送方权限。作为参数传递到 LocalConnection.allowDomain() 方法的字符串可以包含以下任意项：确切域名、 IP 地址和通配符 *。

allowDomain() 方法的格式已更改，与其在 ActionScript 1.0 和 2.0 中的格式不同。在这两个早期版本中，allowDomain() 是可以实现的回调方法。在 ActionScript 3.0 中， allowDomain() 则是调用的 LocalConnection 类的内置方法。由于此更改，allowDomain() 的用法与 Security.allowDomain() 基本相同。

SWF 文件可以使用 LocalConnection 类的 domain 属性确定其所在的域。

控制外出 URL 访问

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

通过使用下列 API 可实现外出脚本访问和外出 URL 访问 （通过使用 HTTP URL、 mailto: 等）：

- flash.system.fscommand() 函数
- ExternalInterface.call() 方法
- flash.net.navigateToURL() 函数

对于从本地文件系统加载的内容，仅当代码和包含的网页 （如果存在）位于受信任的本地沙箱或 AIR 应用程序安全沙箱时，才能成功调用这些方法。如果内容位于只能与远程内容交互的沙箱或只能与本地文件系统内容交互的沙箱中，则对这些方法的调用将失败。

对于非本地加载的内容，所有这些 API 都可以与其嵌入的网页进行通信，具体取决于下述 AllowScriptAccess 参数的值。flash.net.navigateToURL() 函数还有额外的功能，可以与任何打开的浏览器窗口或框架通信，而不只是与嵌入 SWF 文件的网页通信。有关此功能的详细信息，请参阅第 920 页的 “ 使用 navigateToURL() 函数 ”。

HTML 代码中用于加载 SWF 文件的 AllowScriptAccess 参数控制能否从 SWF 文件内执行外出 URL 访问。此参数在 PARAM 或 EMBED 标签内设置。如果不设置 AllowScriptAccess 的值，则仅当 SWF 文件与 HTML 页来自同一个域时才能通信。

AllowScriptAccess 参数可以有 "always"、 "sameDomain" 和 "never" 这三个可能值中的一个。

- 当 AllowScriptAccess 为 "always" 时， SWF 文件可以与其嵌入到的 HTML 页进行通信，即使该 SWF 文件来自不同于 HTML 页的域也可以。
- 当 AllowScriptAccess 为 "sameDomain" 时，仅当 SWF 文件与其嵌入到的 HTML 页来自相同的域时，该 SWF 文件才能与该 HTML 页进行通信。此值是 AllowScriptAccess 的默认值。使用此设置，或者不设置 AllowScriptAccess 的值，可以防止一个域中的 SWF 文件访问另一个域的 HTML 页内的脚本。
- 当 AllowScriptAccess 为 "never" 时，SWF 文件将无法与任何 HTML 页进行通信。在 Adobe Flash CS4 Professional 版本之后，已不再使用此值。如果没有在自己的域中提供不受信任的 SWF 文件，则不建议也不应使用该值。如果确实需要使用不受信任的 SWF 文件，则 Adobe 建议您创建一个不同的子域，并将所有不受信任的内容置于其中。

下面的示例说明如何在 HTML 页中设置 AllowScriptAccess 标签以允许对其他域进行外出 URL 访问：

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
codebase='http://download.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0' height='100%'
width='100%'>
<param name='AllowScriptAccess' value='always'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer' src='MyMovie.swf'
height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

使用 navigateToURL() 函数

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

除上述 allowScriptAccess 参数所指定的安全设置外， navigateToURL() 函数还有一个可选的第二参数：target。 target 参数可用于指定将 URL 请求发送到的 HTML 窗口或框架的名称。此类请求还有其他一些安全限制，具体的限制取决于是否将 navigateToURL() 用作脚本访问或非脚本访问语句。

对于脚本访问语句，如 navigateToURL("javascript: alert('Hello from Flash Player.')")，适用以下规则。

- 如果 SWF 文件是本地受信任的文件，则请求成功。
- 如果目标是 SWF 文件嵌入到的 HTML 页，则适用上述 allowScriptAccess 规则。
- 如果目标容纳的内容是加载自 SWF 文件所在的域，则请求成功。
- 如果目标容纳的内容是加载自与 SWF 文件不同的域，并且上述两种条件均不满足，则请求失败。

对于非脚本访问语句 （如 HTTP、 HTTPS 和 mailto:），如果满足以下所有条件，则请求失败：

- 目标具有下面任一特殊关键字："_top" 或 "_parent"。
- SWF 文件位于其他域的网页中。
- 嵌入 SWF 文件时 allowScriptAccess 的值不是 "always"。

详细信息

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

有关外出 URL 访问的详细信息，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考中的下列条目：

- flash.system.fscommand() 函数
- ExternalInterface 类的 call() 方法
- flash.net.navigateToURL() 函数

共享对象

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

Flash Player 提供使用 " 共享对象 " 的功能，这些对象是永久位于 SWF 文件外部的 ActionScript 对象，它们或者位于用户的本地文件系统中，或者位于远程 RTMP 服务器上。共享对象与 Flash Player 中的其他媒体相似，也划分到安全沙箱中。但是，共享对象的沙箱模型稍有不同，因为共享对象不是可以跨域边界访问的资源，而是始终从共享对象存储区获得的资源，该存储

库特定于调用 SharedObject 类的方法的每个 SWF 文件的域。通常，共享对象存储区比 SWF 文件所在的域更精确：默认情况下，每个 SWF 文件使用特定于其整个源 URL 的共享对象存储区。有关共享对象的详细信息，请参阅第 602 页的 " 共享对象 "。

SWF 文件可以使用 SharedObject.getLocal() 和 SharedObject.getRemote() 方法的 localPath 参数，以便使用仅与其部分 URL 关联的共享对象存储区。这样，SWF 文件可以允许与其他 URL 的其他 SWF 文件共享。即使将 '/' 作为 localPath 参数传递，仍然会指定特定于其自身所在域的共享对象存储区。

用户可通过使用 "Flash Player 设置 " 对话框或 " 设置管理器 " 来限制共享对象访问。默认情况下，可以将共享对象创建为每个域最多可以保存 100 KB 的数据。管理用户和用户还可限制写入文件系统的能力。有关详细信息，请参阅第 899 页的 " 管理员控制 " 和第 900 页的 " 用户控制 "。

可以通过为 SharedObject.getLocal() 方法或 SharedObject.getRemote() 方法的 secure 参数指定 true 来指定共享对象是安全的。请注意有关 secure 参数的以下说明：

- 如果此参数设置为 true，则 Flash Player 将创建一个新的安全共享对象或获取一个对现有安全共享对象的引用。只可由通过 HTTPS 传递的 SWF 文件对此安全共享对象进行读取或写入，而该 HTTPS 调用 SharedObject.getLocal() 并将 secure 参数设置为 true。
- 如果此参数设置为 false，则 Flash Player 将创建一个新的共享对象或获取一个对现有共享对象的引用，可以通过非 HTTPS 连接传递的 SWF 文件对此共享对象或现有共享对象的引用进行读取或写入。

如果执行调用的 SWF 文件不是来自 HTTPS URL，则为 SharedObject.getLocal() 方法或 SharedObject.getRemote() 方法的 secure 参数指定 true 会导致 SecurityError 异常。

对共享对象存储区的选择基于 SWF 文件的源 URL。即使在导入加载和动态加载这两种情况下也是如此，这时 SWF 文件不是源自简单的 URL。导入加载是指在 LoaderContext.securityDomain 属性设置为 SecurityDomain.currentDomain 时加载 SWF 文件。在这种情况下，被加载的 SWF 文件将具有一个伪 URL，该 URL 以执行加载的 SWF 文件所在的域开头，后跟实际的源 URL。动态加载是指使用 Loader.loadBytes() 方法加载 SWF 文件。在这种情况下，被加载的 SWF 文件将具有一个伪 URL，该 URL 以执行加载的 SWF 文件的完整 URL 开头，后跟一个整数 ID。在导入加载和动态加载这两种情况下，都可以使用 LoaderInfo.url 属性检查 SWF 文件的伪 URL。选择共享对象存储区时，可将该伪 URL 完全视为真实的 URL。可以指定使用部分或全部伪 URL 的共享对象 localPath 参数。

用户和管理员可以选择禁止使用 " 第三方共享对象 "。当在 Web 浏览器中执行的任何 SWF 文件的源 URL 与浏览器地址栏中显示的 URL 属于不同的域时，该 SWF 文件可以使用这样的共享对象。用户和管理员可以出于隐私原因选择禁止使用第三方共享对象，从而可以避免跨域跟踪。为避开这种限制，可能希望确保使用共享对象的任何 SWF 文件都只在 HTML 页面结构内部进行加载，这样可以确保 SWF 文件位于浏览器的地址栏中所示的同一个域中。试图使用第三方 SWF 文件中的共享对象时，如果第三方共享对象禁止使用，则 SharedObject.getLocal() 和 SharedObject.getRemote() 方法将返回 null。有关详细信息，请访问 www.adobe.com/cn/products/flashplayer/articles/thirdpartylso。

摄像头、麦克风、剪贴板、鼠标和键盘访问

Flash Player 9 和更高版本， **Adobe AIR 1.0** 和更高版本

当 SWF 文件试图使用 Camera.get() 或 Microphone.get() 方法访问用户的摄像头或麦克风时，Flash Player 将显示一个 " 隐私 " 对话框，用户可以在该对话框中允许或拒绝对其摄像头或麦克风的访问。用户和管理用户还可以通过 mms.cfg 文件、" 设置 UI" 和 " 设置管理器 " 中的控制基于站点或全局禁用摄像头访问 （请参阅第 899 页的 " 管理员控制 " 和第 900 页的 " 用户控制 "）。用户加以限制后，Camera.get() 和 Microphone.get() 方法均返回 null 值。可以使用 Capabilities.avHardwareDisable 属性确定是已经通过管理方式禁止了 (true) 对摄像头和麦克风的访问，还是允许 (false) 对摄像头和麦克风的访问。

System.setClipboard() 方法允许 SWF 文件用纯文本字符串替换剪贴板内容。这不会带来任何安全性风险。为避免因剪切或复制到剪贴板的密码和其他敏感数据所带来的风险，并未提供相应的 getClipboard() 方法。

运行于 Flash Player 中的应用程序只能监视在其焦点以内发生的键盘和鼠标事件。运行于 Flash Player 中的内容不能检测其他应用程序中的键盘或鼠标事件。

AIR 安全性

Adobe AIR 1.0 和更高版本

AIR 安全性基础知识

Adobe AIR 1.0 和更高版本

AIR 应用程序运行的安全性限制与本机应用程序一样。一般来说，AIR 应用程序像本机应用程序一样，对操作系统功能有广泛的访问权限，例如读取和写入文件、启动应用程序、绘制到屏幕以及与网络通信。适用于本机应用程序的操作系统限制 （例如特定于用户的权限）同样适用于 AIR 应用程序。

虽然 Adobe® AIR® 安全模型是由 Adobe® Flash® Player 安全模型发展而来的，但是其安全协定与适用于浏览器中的内容的安全协定不同。此协定为开发人员提供了一种自由访问更广泛的功能以便获得丰富体验的安全方式，而这种方式并不适合基于浏览器的应用程序。

AIR 应用程序是采用编译过的字节码 （SWF 内容）或解释过的脚本 （JavaScript、 HTML）编写的，以便运行时提供内存管理。这样可以最大程度地减少与内存管理 （如缓冲区溢出和内存损坏）有关的漏洞对 AIR 应用程序产生影响的可能性。下面是一些影响用本机代码编写的桌面应用程序的最常见漏洞。

安装和更新

Adobe AIR 1.0 和更高版本

AIR 应用程序通过具有 air 扩展名的 AIR 安装程序文件分发，或者通过具有本机平台的文件格式和扩展名的本机安装程序分发。例如， Windows 的本机安装程序格式是 EXE 文件，对于 Android，则本机格式是 APK 文件。

当安装 Adobe AIR 后，打开 AIR 安装程序文件时，AIR 运行时会管理安装进程。当使用本机安装程序时，操作系统管理安装进程。

注：开发人员可以在使用 AIR 文件格式时，指定版本、应用程序名称和发行商来源，但初始应用程序安装工作流本身无法修改。此限制对用户非常有利，因为所有 AIR 应用程序共享由运行时管理的安全、简单且一致的安装过程。如果有必要对应用程序进行自定义，则可以在首次执行应用程序时进行自定义。

运行时安装位置

Adobe AIR 1.0 和更高版本

使用 AIR 文件格式的 AIR 应用程序首先要求在用户的计算机上安装运行时，就像 SWF 文件首先要求安装 Flash Player 浏览器插件一样。

运行时将安装到桌面计算机上的以下位置：

- Mac OS：/Library/Frameworks/
- Windows：C:\Program Files\Common Files\Adobe AIR
- Linux：/opt/Adobe AIR/

在 Mac OS 中，若要安装某一应用程序的更新版本，用户必须具有足够的系统权限才能将新版本安装到应用程序目录中。在 Windows 和 Linux 中，用户必须具有管理权限。

注：在 iOS 上，不单独安装 AIR 运行时；每个 AIR 应用程序都是自包含应用程序。

可以通过两种方式安装运行时：使用无缝安装功能 （直接从 Web 浏览器安装）或通过手动安装。打包成本机安装程序的 AIR 应用程序还可以安装 AIR 运行时，作为其正常应用程序安装流程的一部分。（以此方式分发 AIR 运行时需要与 Adobe 签订再分发协议。）

无缝安装 （运行时和应用程序）

Adobe AIR 1.0 和更高版本

借助无缝安装功能，开发人员可以让没有 Adobe AIR 安装经验的用户体验简单化的安装过程。通过无缝安装方法，开发人员可以创建用于提供应用程序安装的 SWF 文件。用户单击该 SWF 文件安装应用程序时，该 SWF 文件将尝试检测运行时。如果检测不到运行时，运行时会自行安装并且会立即激活，同时开始安装开发人员的应用程序。

手动安装

Adobe AIR 1.0 和更高版本

用户也可以在打开 AIR 文件之前手动下载并安装运行时。开发人员随后可以通过不同的方式 （例如通过电子邮件或网站上的 HTML 链接）分发 AIR 文件。打开 AIR 文件后，运行时便开始处理应用程序安装过程。

应用程序安装流程

Adobe AIR 1.0 和更高版本

AIR 安全模型允许用户决定是否要安装 AIR 应用程序。 AIR 安装体验在本机应用程序安装技术的基础上提供了以下几个方面的改进，使用户可以更容易地做出信任安装的决定：

- 即使通过 Web 浏览器中的链接安装 AIR 应用程序，运行时也会对所有操作系统提供一致的安装体验。大多数本机应用程序安装体验根据浏览器或其他应用程序提供安全信息 （如果提供了安全信息）。
- AIR 应用程序安装体验可以确定应用程序的源以及有关应用程序可用权限的信息 （如果用户允许继续安装）。
- 运行时会管理 AIR 应用程序的安装过程。 AIR 应用程序无法控制运行时使用的安装过程。

通常，用户不应安装来自其不信任源或无法验证源的任何桌面应用程序。与其他可安装应用程序一样，对本机应用程序执行的安全验证也适用于 AIR 应用程序。

应用程序安装目标

Adobe AIR 1.0 和更高版本

可以选择以下两种方式之一设置安装目录：

1. 用户在安装过程中自定义目标。应用程序将安装到用户指定的任意位置。
2. 如果用户未更改安装目标，则应用程序将安装到运行时确定的默认路径下：
   - Mac OS：~/Applications/
   - Windows XP 及更低版本：C:\Program Files\
   - Windows Vista：~/Apps/
   - Linux：/opt/

如果开发人员在应用程序描述符文件中指定了 installFolder 设置，则应用程序将安装到此目录的子路径下。

AIR 文件系统

Adobe AIR 1.0 和更高版本

AIR 应用程序的安装过程会将开发人员在 AIR 安装程序文件中包括的所有文件复制到用户的本地计算机上。安装的应用程序由以下内容组成：

- Windows：包含 AIR 安装程序文件中的所有文件的目录。在安装 AIR 应用程序的过程中，运行时还会创建一个 exe 文件。
- Linux：包含 AIR 安装程序文件中所含所有文件的目录。在安装 AIR 应用程序的过程中，运行时还会创建一个 bin 文件。
- Mac OS：包含 AIR 安装程序文件的所有内容的 app 文件。可以使用 Finder 中的 " 显示包内容 " 选项检查该文件。运行时会在 AIR 应用程序的安装过程中创建该 app 文件。

AIR 应用程序的运行方式如下：

- Windows：运行安装文件夹中的 .exe 文件或对应于此文件的快捷方式 （如 " 开始 " 菜单或桌面上的快捷方式）。
- Linux：启动安装文件夹中的 .bin 文件、从 " 应用程序 " 菜单中选择该应用程序，或者从别名或桌面快捷方式运行。
- Mac OS：运行 .app 文件或指向该文件的别名。

应用程序文件系统还包括与应用程序功能相关的子目录。例如，写入加密本地存储的信息保存到以应用程序的应用程序标识符命名的目录的子目录中。

AIR 应用程序存储

Adobe AIR 1.0 和更高版本

AIR 应用程序具有写入用户硬盘驱动器上的任意位置的权限；但是，鼓励开发人员使用 app-storage:/ 路径作为与其应用程序相关的本地存储。从应用程序写入 app-storage:/ 的文件位于标准位置中，具体取决于用户的操作系统：

- 在 Mac OS 上：应用程序的存储目录随 AIR 版本不同而不同：
  - **AIR 3.2** 和早期版本 - <appData>/<appId>/Local Store/，其中 <appData> 表示用户的 " 首选参数文件夹 "，通常为：/Users/<user>/Library/Preferences
  - **AIR 3.3** 和更高版本 - <path>/Library/Application Support/<appID>/Local Store，其中 <path> 为 /Users/<user>/Library/Containers/<bundle-id>/Data （沙箱环境）或 /Users/<user> （在沙箱环境外运行时）
- 在 Windows 中，应用程序的存储目录为 <appData>\<appId>\Local Store\，其中 <appData> 为用户的 CSIDL_APPDATA" 特殊文件夹 "，通常为 C:\Documents and Settings\<user>\Application Data
- 在 Linux 中为 <appData>/<appID>/Local Store/，其中 <appData> 为 /home/<user>/.appdata

可以通过 air.File.applicationStorageDirectory 属性访问应用程序存储目录。可以使用 File 类的 resolvePath() 方法访问目录中的内容。有关详细信息，请参阅第 559 页的 " 使用文件系统 "。

更新 Adobe AIR

Adobe AIR 1.0 和更高版本

如果用户安装的 AIR 应用程序需要运行时的更新版本，则运行时会自动安装所需的运行时更新。

若要更新运行时，用户必须具有计算机的管理权限。

更新 AIR 应用程序

Adobe AIR 1.0 和更高版本

开发和部署软件更新是本机代码应用程序面临的最大安全挑战之一。 AIR API 提供了一种改进此问题的机制：可以在启动时调用 Updater.update() 方法来检查 AIR 文件的远程位置。如果存在适当的更新，则会下载并安装 AIR 文件，然后重新启动该应用程序。开发人员可以使用此类提供新功能和响应潜在安全漏洞。

Updater 类仅用于更新作为 AIR 文件分发的应用程序。作为本机应用程序分发的应用程序必须使用本机操作系统的更新组件（如果有）。

注：开发人员可以通过设置应用程序描述符文件的 versionNumber 属性指定应用程序的版本。

卸载 AIR 应用程序

Adobe AIR 1.0 和更高版本

删除 AIR 应用程序的同时也将删除应用程序目录中的所有文件。然而，它不删除应用程序可能写入应用程序目录外的所有文件。删除 AIR 应用程序不会撤消 AIR 应用程序对该应用程序目录外部的文件所做的更改。

针对管理员的 Windows 注册表设置

Adobe AIR 1.0 和更高版本

在 Windows 中，管理员可以通过配置计算机来阻止 （或允许）安装 AIR 应用程序和更新运行时。这些设置包含在 Windows 注册表的 HKLM\Software\Policies\Adobe\AIR 项中。这些设置包括以下内容：

注册表设置	说明
AppInstallDisabled	指定是否允许安装和卸载 AIR 应用程序。设置为 0 表示 “ 允许 ”，设置为 1 表示 “ 禁止 ”。
UntrustedAppInstallDisabled	指定允许安装不受信任的 AIR 应用程序 （没有可信证书的应用程序）。设置为 0 表示 “ 允许 ”，设置为 1 表示 “ 禁止 ”。
UpdateDisabled	指定是否允许更新运行时，该操作可以作为后台任务执行，也可以作为显式安装的一部分执行。设置为 0 表示 “ 允许 ”，设置为 1 表示 “ 禁止 ”。

Adobe AIR 中的 HTML 安全性

Adobe AIR 1.0 和更高版本

本主题介绍 AIR HTML 安全体系结构，以及如何使用 iframe、帧与沙箱桥设置基于 HTML 的应用程序和安全地将 HTML 内容集成到基于 SWF 的应用程序。

运行时会强制执行规则，并提供克服 HTML 和 JavaScript 中的潜在安全漏洞的机制。不论您的应用程序为主要采用 JavaScript 编写，还是您将 HTML 和 JavaScript 内容加载到基于 SWF 的应用程序，强制执行的规则都相同。应用程序沙箱和非应用程序安全沙箱中的内容具有不同的权限。将内容加载到 iframe 或 frame 中时，运行时会提供一种安全的沙箱桥机制，该机制允许 frame 或 iframe 中的内容能够与应用程序安全沙箱中的内容进行安全通信。

AIR SDK 为呈现 HTML 内容提供三个类。

HTMLLoader 类提供 JavaScript 代码和 AIR API 之间的紧密集成。

StageWebView 类是一个 HTML 呈现类，与主机 AIR 应用程序仅有非常有限的集成。StageWebView 类加载的内容从不放置在应用程序安全沙箱中，因而无法访问主机 AIR 应用程序中的数据或调用其中的函数。在桌面平台上， StageWebView 类根据 Webkit（HTMLLoader 类也使用它）使用内置 AIR HTML 引擎。在移动平台上，StageWebView 类使用操作系统提供的 HTML 控件。因此，在移动平台上， StageWebView 类与系统 Web 浏览器有着相同的安全性注意事项和漏洞。

TextField 类可以显示 HTML 文本字符串。 JavaScript 不可以执行，但文本可以包括链接和外部加载的图像。

有关详细信息，请参阅第 841 页的 " 避免与安全相关的 JavaScript 错误 "。

配置基于 HTML 的应用程序概述

Adobe AIR 1.0 和更高版本

Frame 和 iframe 提供了一种用于组织 AIR 中的 HTML 内容的便利结构。Frame 提供了一种用于维护数据永久性以及安全使用远程内容的方式。

由于 AIR 中的 HTML 保持其基于页面的正常组织，因此 HTML 环境在 HTML 内容的顶框架 " 导航 " 到其他页面时会完全刷新。您可以使用 frame 和 iframe 来维护 AIR 中的数据永久性，方法与维护在浏览器上运行的 Web 应用程序中的数据永久性基本相同。定义顶框架中的主应用程序对象，只要不允许框架导航到新页面，这些对象就会永久保留。使用子级 frame 或 iframe 加载并显示应用程序的临时部分。（除 frame 外，还可以使用多种方式维护数据永久性。其中包括 cookie、本地共享对象、本地文件存储、加密文件存储以及本地数据库存储。）

由于 AIR 中的 HTML 在可执行代码与数据之间保持正常的模糊界限，因此 AIR 将 HTML 环境的顶部框架中的内容放入应用程序沙箱中。在页面的 load 事件之后， AIR 限制 eval() 等任何可以将文本的字符串转换为可执行对象的操作。即使应用程序未加载远程内容，也会强制实施此限制。若要允许 HTML 内容执行这些受限制的操作，必须使用 frame 或 iframe 将内容放入非应用程序沙箱中。 （使用某些依赖 eval() 函数的 JavaScript 应用程序框架时，可能必须运行沙箱子帧中的内容。）有关应用程序沙箱中的 JavaScript 限制的完整列表，请参阅第 927 页的 " 对不同沙箱中的内容的代码限制 "。

由于 AIR 中的 HTML 能够加载远程潜在不安全内容，因此 AIR 会强制实施同源策略，以防止一个域中的内容与另一个域中的内容进行交互。若要允许应用程序内容与其他域中的内容进行交互，可以设置一个桥，将其用作父级和子级 frame 之间的接口。

设置父子沙箱关系

Adobe AIR 1.0 和更高版本

AIR 会将 sandboxRoot 和 documentRoot 属性添加到 HTML frame 和 iframe 元素中。使用这些属性，您可以将应用程序内容视为其他域中的内容：

属性	说明
sandboxRoot	用于确定在其中放置 frame 内容的沙箱和域的 URL。必须使用 file:、 http: 或 https: URL 方案。
documentRoot	从中加载 frame 内容的 URL。必须使用 file:、 app: 或 app-storage: URL 方案。

以下示例对要在远程沙箱中运行的应用程序的 sandbox 子目录中安装的内容以及 www.example.com 域进行了映射：

```
<iframe
    src="ui.html"
    sandboxRoot="http://www.example.com/local/"
    documentRoot="app:/sandbox/">
</iframe>
```

在不同沙箱或域的父级和子级 frame 之间设置桥

Adobe AIR 1.0 和更高版本

AIR 将 childSandboxBridge 和 parentSandboxBridge 属性添加到任何子级 frame 的 window 对象中。使用这些属性，您可以定义用作父级和子级 frame 之间的接口的桥。每个桥都指向一个方向：

childSandboxBridge — childSandboxBridge 属性允许子级 frame 向父级 frame 中的内容公开接口。若要公开接口，需将 childSandbox 属性设置为子级 frame 中的函数或对象。然后，可以从父级 frame 中的内容访问该对象或函数。以下示例显示了子级 frame 中运行的脚本如何向其父级 frame 公开包含函数和属性的对象：

```
var interface = {};
interface.calculatePrice = function(){
    return .45 + 1.20;
}
interface.storeID = "abc"
window.childSandboxBridge = interface;
```

如果此子级内容位于分配的 ID 为 "child" 的 iframe 中，则可以通过读取 frame 的 childSandboxBridge 属性从父级内容来访问接口：

```
var childInterface = document.getElementById("child").childSandboxBridge;
air.trace(childInterface.calculatePrice()); //traces "1.65"
air.trace(childInterface.storeID)); //traces "abc"
```

parentSandboxBridge — parentSandboxBridge 属性允许父级 frame 向子级 frame 中的内容公开接口。若要公开接口，需将子级 frame 的 parentSandbox 属性设置为父级 frame 中的函数或对象。然后，可以从子级 frame 中的内容访问该对象或函数。以下示例显示了父级 frame 中运行的脚本如何向其子级 frame 公开包含 save 函数的对象：

```
var interface = {};
interface.save = function(text){
    var saveFile = air.File("app-storage:/save.txt");
    //write text to file
}
document.getElementById("child").parentSandboxBridge = interface;
```

使用此接口，子级 frame 中的内容可以将文本保存到名为 save.txt 的文件。但对文件系统不具备任何其他访问权限。通常，应用程序内容向其他沙箱公开的接口应越窄越好。子级内容可以调用 save 函数，如下所示：

```
var textToSave = "A string.";
window.parentSandboxBridge.save(textToSave);
```

如果子级内容尝试设置 parentSandboxBridge 对象的属性，则运行时会引发 SecurityError 异常。如果父级内容尝试设置 childSandboxBridge 对象的属性，则运行时会引发 SecurityError 异常。

对不同沙箱中的内容的代码限制

Adobe AIR 1.0 和更高版本

在 第 925 页的 “Adobe AIR 中的 HTML 安全性 ” 主题的简介中已介绍过，运行时强制执行规则，并提供克服 HTML 和 JavaScript 中可能的安全漏洞的机制。本主题列出了这些限制。如果代码尝试调用这些受限制的 API，则运行时将发出错误：“Adobe AIR runtime security violation for JavaScript code in the application security sandbox”（应用程序安全沙箱中存在针对 JavaScript 代码的 Adobe AIR 运行时安全侵犯）。

有关详细信息，请参阅第 841 页的 “ 避免与安全相关的 JavaScript 错误 ”。

使用 JavaScript eval() 函数及类似技术的限制

Adobe AIR 1.0 和更高版本

对于应用程序安全沙箱中的 HTML 内容，加载代码后 （即在调度 body 元素的 onload 事件以及 onload 处理函数完成执行后），使用可将字符串动态转换为可执行代码的 API 时存在一些限制。这是为了阻止应用程序从非应用程序源 （例如潜在不安全网络域）意外插入 （及执行）代码。

例如，如果应用程序使用远程源中的字符串数据来写入 DOM 元素的 innerHTML 属性，则字符串中包括的可执行 (JavaScript) 代码可能会执行不安全操作。但是，在加载内容时将远程字符串插入 DOM 不存在风险。

使用 JavaScript eval() 函数时存在一个限制。在加载应用程序沙箱中的代码且处理 onload 事件处理函数之后，只能通过有限的方式使用 eval() 函数。以下规则适用于在从应用程序安全沙箱中加载代码之后 使用 eval() 函数：

- 允许表达式中包含文本。例如：

```
eval("null");
eval("3 + .14");
eval("'foo'");
```

- 允许使用对象文本，如下所示：

```
{ prop1: val1, prop2: val2 }
```

- 禁止 使用 setter/getter 对象文本，如下所示：

```
{ get prop1() { ... }, set prop1(v) { ... } }
```

- 允许使用数组文本，如下所示：

```
[ val1, val2, val3 ]
```

- 禁止 表达式中包含属性读取，如下所示：

```
a.b.c
```

- 禁止 调用函数。
- 禁止 禁止对函数进行定义。
- 禁止 设置任何属性。
- 禁止 使用函数文本。

但是，加载代码时，在 onload 事件之前和执行 onload 事件处理函数过程中，这些限制不适用于应用程序安全沙箱中的内容。

例如，加载代码后，以下代码会导致运行时引发异常：

```
eval("alert(44)");
eval("myFunction(44)");
eval("NativeApplication.applicationID");
```

如果应用程序沙箱中允许使用代码，则动态生成的代码 （例如在调用 eval() 函数时生成的代码）将导致安全风险。例如，应用程序可能意外执行了从网络域中加载的字符串，而该字符串可能包含恶意代码。例如，这些代码可能会删除或修改用户计算机上的文件。也可能会将本地文件的内容报告给某个不受信任的网络域。

生成动态代码的方式如下所示：

- 调用 eval() 函数。
- 使用 innerHTML 属性或 DOM 函数插入加载应用程序目录外部的脚本的 script 标签。
- 使用 innerHTML 属性或 DOM 函数插入具有内联代码的 script 标签 （而不是通过 src 属性加载脚本）。
- 设置 script 标签的 src 属性可以加载应用程序目录外部的 JavaScript 文件。
- 使用 javascript URL 方案 （如 href="javascript:alert('Test')" 所示）。
- 使用 setInterval() 或 setTimout() 函数，其中，第一个参数 （用于定义要异步运行的函数）为要求值的字符串而不是函数名 （如 setTimeout('x = 4', 1000) 所示）。
- 调用 document.write() 或 document.writeln()。

加载内容时，应用程序安全沙箱中的代码只能使用这些方法。

这些限制不会 阻止将 eval() 和 JSON 对象文本一起使用。这样便可以在 JSON JavaScript 库中使用应用程序内容。但是会限制您使用重载的 JSON 代码 （通过事件处理函数）。

对于其他 Ajax 框架和 JavaScript 代码库，需检查框架或库中的代码是否在限制动态生成的代码时起作用。如果不起作用，则需包括在非应用程序安全沙箱中使用框架或库的所有内容。有关详细信息，请参阅 第 897 页的 " 对 AIR 内的 JavaScript 的限制 " 和第 934 页的 " 通过脚本访问应用程序和非应用程序内容 "。 Adobe 维护一个已知的支持应用程序安全沙箱的 Ajax 框架列表，其网址为 http://www.adobe.com/cn/products/air/develop/ajax/features/。

与应用程序安全沙箱中的内容不同，非应用程序安全沙箱中的 JavaScript 内容随时都可以 调用 eval() 函数来执行动态生成的代码。

访问 AIR API 的限制 （针对非应用程序沙箱）

Adobe AIR 1.0 和更高版本

非应用程序沙箱中的 JavaScript 代码无法访问 window.runtime 对象，也无法执行 AIR API。如果非应用程序安全沙箱中的内容调用以下代码，则应用程序会引发 TypeError 异常：

```
try {
    window.runtime.flash.system.NativeApplication.nativeApplication.exit();
}
catch (e)
{
    alert(e);
}
```

异常类型为 TypeError （未定义的值），由于非应用程序沙箱中的内容无法识别 window.runtime 对象，因此将其认为是未定义的值。

可以使用脚本桥将运行时功能公开给非应用程序沙箱中的内容。有关详细信息，请参阅第 934 页的 " 通过脚本访问应用程序和非应用程序内容 "。

使用 XMLHttpRequest 调用的限制

Adobe AIR 1.0 和更高版本

应用程序安全沙箱中的 HTML 内容无法使用同步 XMLHttpRequest 方法，在加载 HTML 内容和执行 onLoad 事件期间从应用程序沙箱外部加载数据。

默认情况下，不允许非应用程序安全沙箱中的 HTML 内容使用 JavaScript XMLHttpRequest 对象从非调用请求的域加载数据。 frame 或 iframe 标签可以包括 allowcrosscomainxhr 属性。如果将此属性设置为任何非空值，则会允许帧或 iframe 中的内容使用 JavaScript XMLHttpRequest 对象从域 （注意不是调用请求的代码的域）加载数据：

```
<iframe id="UI"
    src="http://example.com/ui.html"
    sandboxRoot="http://example.com/"
    allowcrossDomainxhr="true"
    documentRoot="app:/">
</iframe>
```

有关详细信息，请参阅第 930 页的 " 通过脚本访问不同域中的内容 "。

加载 CSS、 frame、 iframe 和 img 元素的限制 （针对非应用程序沙箱中的内容）

Adobe AIR 1.0 和更高版本

远程 （网络）安全沙箱中的 HTML 内容只能从远程沙箱 （网络 URL）加载 CSS、 frame、 iframe 和 img 内容。

只能与本地文件系统内容交互的沙箱、只能与远程内容交互的沙箱或受信任的本地沙箱中的 HTML 内容只能从本地沙箱 （而不是应用程序或远程沙箱）加载 CSS、 frame、 iframe 和 img 内容。

调用 JavaScript window.open() 方法的限制

Adobe AIR 1.0 和更高版本

如果通过调用 JavaScript window.open() 方法创建的窗口中显示了非应用程序安全沙箱中的内容，则窗口的标题将以主 （启动）窗口的标题开头，后跟一个冒号字符。无法使用代码将窗口的标题部分从屏幕上删除。

非应用程序安全沙箱中的内容只能成功调用 JavaScript window.open() 方法来响应用户与鼠标或键盘交互而触发的事件。这会阻止非应用程序内容创建可能被欺骗使用的窗口 （例如用于仿冒攻击）。此外，鼠标或键盘事件的事件处理函数无法将 window.open() 方法设置为在延迟 （例如调用 setTimeout() 函数）后执行。

远程 （网络）沙箱中的内容只能使用 window.open() 方法打开远程网络沙箱中的内容。无法使用 window.open() 方法打开应用程序或本地沙箱中的内容。

只能与本地文件系统内容交互的沙箱、只能与远程内容交互的沙箱或受信任的本地沙箱中的内容 （请参阅第 895 页的 " 安全沙箱 "）只可使用 window.open() 方法打开本地沙箱中的内容。无法使用 window.open() 打开应用程序或远程沙箱中的内容。

调用受限代码时出现的错误

Adobe AIR 1.0 和更高版本

如果由于这些安全限制而限制在沙箱中使用所调用的代码，则运行时将发出 JavaScript 错误："Adobe AIR runtime security violation for JavaScript code in the application security sandbox" （应用程序安全沙箱中存在针对 JavaScript 代码的 Adobe AIR 运行时安全侵犯）。

有关详细信息，请参阅第 841 页的 " 避免与安全相关的 JavaScript 错误 "。

从字符串中加载 HTML 内容时对沙箱的保护

Adobe AIR 1.0 和更高版本

通过 HTMLLoader 类的 loadString() 方法，可以在运行时创建 HTML 内容。但是，如果从不安全的 Internet 来源加载数据，则用作 HTML 内容的数据可能已损坏。由于此原因，默认情况下，使用 loadString() 方法创建的 HTML 不放置在应用程序沙箱中，并且无权访问 AIR API。但是，将 HTMLLoader 对象的 placeLoadStringContentInApplicationSandbox 属性设置为 true，即可将使用 loadString() 方法创建的 HTML 放置在应用程序沙箱中。有关详细信息，请参阅第 840 页的 " 从字符串加载 HTML 内容 "。

通过脚本访问不同域中的内容

Adobe AIR 1.0 和更高版本

AIR 应用程序在安装时会被授予特殊权限。不要将相同权限泄露给其他内容 （包括不属于应用程序的远程文件和本地文件），这一点很重要。

关于 AIR 沙箱桥

Adobe AIR 1.0 和更高版本

通常，一个域中的内容无法调用其他域中的脚本。为了保护 AIR 应用程序不会意外泄露获得权限的信息或控制，对应用程序安全沙箱中的内容 （随应用程序一起安装的内容）施加了以下限制：

- 无法通过调用 Security.allowDomain() 方法允许应用程序安全沙箱中的代码访问其他沙箱。从应用程序安全沙箱中调用此方法将引发错误。
- 禁止通过设置 LoaderContext.securityDomain 或 LoaderContext.applicationDomain 属性将非应用程序内容导入应用程序沙箱。

但是仍存在这样一些情况：主 AIR 应用程序要求远程域中的内容对主 AIR 应用程序中的脚本具有受控访问权限，反之亦然。为此，运行时提供了沙箱桥机制，沙箱桥充当两个沙箱之间的通道。沙箱桥可以在远程安全沙箱和应用程序安全沙箱之间提供显式交互。

沙箱桥公开了以下两个对象，已加载和要加载的脚本都可以访问这两个对象：

- parentSandboxBridge 对象允许要加载的内容将属性和函数公开给已加载的内容中的脚本。
- childSandboxBridge 对象允许已加载的内容将属性和函数公开给要加载的内容中的脚本。

通过沙箱桥公开的对象按值而不是按引用进行传递。所有数据都会序列化。这意味着由桥的一端公开的对象无法由另一端设置，并且公开的对象为无类型对象。此外，只能公开简单对象和函数；不能公开复杂对象。

如果子级内容试图设置 parentSandboxBridge 对象的一个对象，则运行时会引发 SecurityError 异常。同样，如果父级内容试图设置 childSandboxBridge 对象的一个对象，则运行时也会引发 SecurityError 异常。

沙箱桥示例 (SWF)

Adobe AIR 1.0 和更高版本

假设 AIR 音乐商店应用程序需要允许远程 SWF 文件广播专辑价格，但不需要远程 SWF 文件透露该价格是否为销售价格。为此，StoreAPI 类提供了一种获取价格的方法，但屏蔽了销售价格。随后会将此 StoreAPI 类的实例分配给 Loader 对象（该对象加载远程 SWF）的 LoaderInfo 对象的 parentSandboxBridge 属性。

以下是 AIR 音乐商店的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" title="Music Store"
creationComplete="initApp()">
    <mx:Script>
        import flash.display.Loader;
        import flash.net.URLRequest;

        private var child:Loader;
        private var isSale:Boolean = false;

        private function initApp():void {
            var request:URLRequest =
                    new URLRequest("http://[www.yourdomain.com]/PriceQuoter.swf")

            child = new Loader();
            child.contentLoaderInfo.parentSandboxBridge = new StoreAPI(this);
            child.load(request);
            container.addChild(child);
        }
        public function getRegularAlbumPrice():String {
            return "$11.99";
        }
        public function getSaleAlbumPrice():String {
            return "$9.99";
        }
        public function getAlbumPrice():String {
            if(isSale) {
                return getSaleAlbumPrice();
            }
            else {
                return getRegularAlbumPrice();
            }
        }
    </mx:Script>
    <mx:UIComponent id="container" />
</mx:WindowedApplication>
```

StoreAPI 对象将调用主应用程序来检索常规专辑价格，但在调用 getSaleAlbumPrice() 方法时将返回 “Not available”。以下代码定义了 StoreAPI 类：

```
public class StoreAPI
{
    private static var musicStore:Object;

    public function StoreAPI(musicStore:Object)
    {
        this.musicStore = musicStore;
    }

    public function getRegularAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }

    public function getSaleAlbumPrice():String {
        return "Not available";
    }

    public function getAlbumPrice():String {
        return musicStore.getRegularAlbumPrice();
    }
}
```

以下代码表示的是 PriceQuoter SWF 文件示例，该文件报告了商店价格，但不能报告销售价格。

```
package
{
    import flash.display.Sprite;
    import flash.system.Security;
    import flash.text.*;

    public class PriceQuoter extends Sprite
    {
        private var storeRequester:Object;

        public function PriceQuoter() {
            trace("Initializing child SWF");
            trace("Child sandbox: " + Security.sandboxType);
            storeRequester = loaderInfo.parentSandboxBridge;

            var tf:TextField = new TextField();
            tf.autoSize = TextFieldAutoSize.LEFT;
            addChild(tf);

            tf.appendText("Store price of album is: " + storeRequester.getAlbumPrice());
            tf.appendText("\n");
            tf.appendText("Sale price of album is: " + storeRequester.getSaleAlbumPrice());
        }
    }
}
```

沙箱桥示例 (HTML)

Adobe AIR 1.0 和更高版本

在 HTML 内容中，会将 parentSandboxBridge 和 childSandboxBridge 属性添加到子级文档的 JavaScript window 对象中。有关如何在 HTML 内容中设置桥函数的示例，请参阅第 856 页的 “ 设置沙箱桥接口 ”。

限制 API 公开

Adobe AIR 1.0 和更高版本

公开沙箱桥时，公开限制沙箱桥滥用程度的高级别 API 非常重要。请注意，调用桥实施的内容可能会被破坏 （例如，通过插入代码）。因此 （举例来说），通过桥公开 readFile(path:String) 方法 （读取任意文件的内容）易于受到滥用。最好公开未使用路径且读取特定文件的 readApplicationSetting() API。一旦应用程序部分受到损坏，语义方法越多，就越能限制应用程序产生的破坏。

更多帮助主题

第 855 页的 " 跨脚本访问不同安全沙箱中的内容 "

第 897 页的 "AIR 应用程序沙箱 "

写入磁盘

Adobe AIR 1.0 和更高版本

在 Web 浏览器中运行的应用程序只能与用户的本地文件系统进行有限的交互。 Web 浏览器会实施安全策略，用于确保用户的计算机不会由于加载 Web 内容而被破坏。例如，通过 Flash Player 在浏览器中运行的 SWF 文件无法直接与用户计算机中的文件进行交互。可以将共享对象和 Cookie 写入用户的计算机，以便维护用户首选项和其他数据，但文件系统交互将受到此限制。由于 AIR 应用程序安装在本地，因此它们具有不同的安全协议，其中包括在本地文件系统间进行读取和写入的功能。

这一灵活性要求开发人员担负较高的责任。意外的应用程序不安全因素不仅会危害应用程序的功能，而且会危害用户计算机的完整性。为此，开发人员应阅读第 935 页的 " 开发人员的最佳安全做法 "。

AIR 开发人员可以使用多个 URL 方案协议来访问文件并将文件写入本地文件系统：

URL 方案	说明
app:/	应用程序目录的别名。从此路径访问的文件被分配到应用程序沙箱中，并由运行时授予完全权限。
app-storage:/	本地存储目录的别名，由运行时进行标准化。从此路径访问的文件被分配到非应用程序沙箱中。
file:///	表示用户硬盘的根目录的别名。如果从此路径访问的文件位于应用程序目录中，则该文件会被分配到应用程序沙箱中，否则将被分配到非应用程序沙箱中。

注：AIR 应用程序无法使用 app: URL 方案修改内容。此外，由于管理员设置，只可以读取应用程序目录。

除非用户计算机存在管理员限制，否则 AIR 应用程序有权写入用户硬盘上的任意位置。建议开发人员使用 app-storage:/ 路径作为与其应用程序相关的本地存储。从应用程序写入 app-storage:/ 的文件将放在标准位置中：

- 在 Mac OS 中：应用程序的存储目录为 <appData>/<appId>/Local Store/，其中 <appData> 表示用户的首选文件夹。通常为 /Users/<user>/Library/Preferences
- 在 Windows 中：应用程序的存储目录为 <appData>\<appId>\Local Store\，其中 <appData> 表示用户的 CSIDL_APPDATA 特殊文件夹。通常为 C:\Documents and Settings\<userName>\Application Data
- 在 Linux 中为 <appData>/<appID>/Local Store/，其中 <appData> 为 /home/<user>/.appdata

如果应用程序设计用于与用户文件系统中的现有文件进行交互，请确保阅读第 935 页的 " 开发人员的最佳安全做法 "。

安全使用不受信任的内容

Adobe AIR 1.0 和更高版本

未分配给应用程序沙箱的内容可以为应用程序提供其他脚本功能，但前提是满足运行时的安全条件。本主题介绍 AIR 安全协议以及非应用程序内容。

Security.allowDomain()

Adobe AIR 1.0 和更高版本

AIR 应用程序限制脚本访问非应用程序内容比 Flash Player 浏览器插件限制脚本访问不受信任内容更严格。例如，在浏览器的 Flash Player 中，当分配给 local-trusted 沙箱的 SWF 文件调用 System.allowDomain() 方法时，将向从指定域中加载的任何 SWF 授予脚本访问权限。不允许对 AIR 应用程序中的 application 内容使用类似方法，因为这将准予对用户文件系统中的非应用程序文件进行不合理访问。远程文件无法直接访问应用程序沙箱，不管是否调用了 Security.allowDomain() 方法。

通过脚本访问应用程序和非应用程序内容

Adobe AIR 1.0 和更高版本

通过脚本访问应用程序和非应用程序内容的 AIR 应用程序具有更复杂的安全安排。只允许不在应用程序沙箱中的文件使用沙箱桥来访问应用程序沙箱中的文件的属性和方法。沙箱桥充当应用程序内容与非应用程序内容之间的通道，在两个文件之间提供显式交互。如果使用正确，沙箱桥会提供额外的安全层，从而限制非应用程序内容访问属于应用程序内容的对象引用。

通过示例可以更好地说明沙箱桥的优点。假设 AIR 音乐商店应用程序需要为希望创建自己的 SWF 文件的广告商提供 API，商店应用程序可以使用这些文件进行通信。该商店需要为广告商提供在商店中查找艺术家和光盘的方法，另外出于安全原因，还需要将某些方法和属性与第三方 SWF 文件进行隔离。

沙箱桥可以提供此功能。默认情况下，在运行时从外部加载到 AIR 应用程序的内容无法访问主应用程序中的任何方法或属性。通过自定义沙箱桥，开发人员可以在不公开这些方法或属性的情况下为远程内容提供服务。将沙箱桥视为受信任内容和不受信任内容之间的通道，在加载方和被加载方内容之间提供通信而不公开对象引用。

有关如何安全使用沙箱桥的详细信息，请参阅第 930 页的 “ 通过脚本访问不同域中的内容 ”。

防止动态生成不安全 SWF 内容

Adobe AIR 1.0 和更高版本

Loader.loadBytes() 方法为应用程序提供了一种从字节数组生成 SWF 内容的方式。但是，在加载内容时对从远程源加载的数据进行注入攻击可能会导致严重损坏。将数据加载到应用程序沙箱时尤其如此，生成的 SWF 内容可以在该应用程序沙箱中访问完整 AIR API。

存在一些合理的用法，可以使用 loadBytes() 方法而不生成可执行 SWF 代码。例如，可以使用 loadBytes() 方法生成图像数据，从而控制图像的显示时间。还存在一些确实依赖执行代码的合理用法，例如动态创建 SWF 内容以便播放音频。在 AIR 中，默认情况下 loadBytes() 方法不 允许加载 SWF 内容；只允许加载图像内容。在 AIR 中，loadBytes() 方法的 loaderContext 属性具有 allowLoadBytesCodeExecution 属性，可以将该属性设置为 true，以便明确允许应用程序使用 loadBytes() 来加载可执行 SWF 内容。下列代码说明如何使用此功能：

```
var loader:Loader = new Loader();
var loaderContext:LoaderContext = new LoaderContext();
loaderContext.allowLoadBytesCodeExecution = true;
loader.loadBytes(bytes, loaderContext);
```

如果调用 loadBytes() 来加载 SWF 内容且 LoaderContext 对象的 allowLoadBytesCodeExecution 属性设置为 false （默认值），则 Loader 对象将引发 SecurityError 异常。

注：在未来的 Adobe AIR 版本中，此 API 可能会更改。如果发生更改，您可能需要重新编译使用 LoaderContext 类的 allowLoadBytesCodeExecution 属性的内容。

开发人员的最佳安全做法

Adobe AIR 1.0 和更高版本

虽然 AIR 应用程序是使用 Web 技术构建的，但开发人员应知道这些应用程序并非在浏览器安全沙箱中运行，这一点很重要。这意味着，可以构建会对本地系统有意或无意产生损害的 AIR 应用程序。AIR 会尝试最大程度降低此风险，但仍存在一些可能引入漏洞的方式。本主题介绍了重要的潜在不安全因素。

将文件导入应用程序安全沙箱的风险

Adobe AIR 1.0 和更高版本

位于应用程序目录中的文件会被分配到应用程序沙箱中，并具有运行时的完全权限。建议将写入本地文件系统的应用程序写入 app-storage:/ 中。此目录与用户计算机上的应用程序文件位于不同的位置，因此这些文件不会分配到应用程序沙箱中，并且安全风险的程度会降低。建议开发人员考虑以下问题：

- 仅在必要时才在 AIR 文件 （位于安装的应用程序中）中包含文件。
- 仅在脚本文件的行为被完全理解和信任时才在 AIR 文件 （位于安装的应用程序中）中包含该脚本文件。
- 不要向应用程序目录中写入内容或修改其中的内容。运行时会阻止应用程序通过引发 SecurityError 异常，写入或修改使用 app:/ URL 方案的文件和目录。
- 不要将网络源中的数据用作可能引起代码异常的 AIR API 的方法的参数。其中包括使用 Loader.loadBytes() 方法和 JavaScript eval() 函数。

使用外部源确定路径的风险

Adobe AIR 1.0 和更高版本

使用外部数据或内容可能会破坏 AIR 应用程序。因此，使用网络或文件系统中的数据时应特别小心。信任最终由开发人员及其构建的网络连接进行保障，但加载外部数据本身就具有风险，不应在敏感操作中使用此输入。建议开发人员不要执行以下操作：

- 使用网络源中的数据确定文件名
- 使用网络源中的数据构建应用程序用来发送私人信息的 URL

使用、存储或传输不安全凭据的风险

Adobe AIR 1.0 和更高版本

将用户凭据存储在用户的本地文件系统中将引入可能破坏这些凭据的风险。建议开发人员考虑以下问题：

- 如果凭据必须存储在本地，请在写入本地文件系统时对凭据进行加密。运行时通过 EncryptedLocalStore 类提供了对每个安装的应用程序都唯一的加密存储。有关详细信息，请参阅第 610 页的 “ 加密的本地存储区 ”。
- 除非网络源可信并且使用 HTTPS: 或传输层安全 (TLS) 协议进行传输，否则不要将未加密的用户凭据传输到网络源。
- 永远不要在创建凭据时指定默认密码，应让用户自己创建密码。保留默认值不变的用户将其凭据暴露在已了解默认密码的攻击者面前。

降级攻击的风险

Adobe AIR 1.0 和更高版本

在安装应用程序过程中，运行时会检查以确保应用程序的版本不是当前安装的版本。如果应用程序已经安装，则运行时会比较版本字符串与已安装的版本。如果此字符串不同，则用户可以选择升级安装。运行时不保证新安装的版本比旧版本新，仅保证版本不同。攻击者可能会向用户分发旧版本以避开安全漏洞。因此，建议开发人员在运行应用程序时检查版本。最好让应用程序检查网络中是否存在所需更新。这样，即使攻击者让用户运行旧版本，该旧版本也会识别出需要更新。此外，为应用程序使用明确的版本控制方案将使欺骗用户安装降级版本变得更加困难。

代码签名

Adobe AIR 1.0 和更高版本

所有 AIR 安装程序文件都需要进行代码签名。代码签名是一种加密过程，用于确认指定的软件源是否正确。可使用由外部证书颁发机构 (CA) 颁发的证书或您自己创建的自签名证书对 AIR 应用程序进行签名。强烈建议从已知的 CA 获得商业证书，这样的证书可以保证用户安装的是您提供的应用程序，而不是赝品。但是，可以使用 SDK 中的 adt 或者使用 Flash、Flash Builder 或使用 adt 生成证书的其他应用程序来创建自签名的证书。自签名证书不保证安装的应用程序为正版，它只能用于测试即将公开发行的应用程序。

Android 设备上的安全性

Adobe AIR 2.5 和更高版本

在 Android 上，和在所有计算设备上一样， AIR 符合本机安全模型。同时， AIR 保持自己的安全性规则，旨在使开发人员轻松编写安全的、与 Internet 连接的应用程序。

因为 Android 上的 AIR 应用程序使用 Android 软件包格式，安装属于 Android 安全性模型。不使用 AIR 应用程序安装程序。

Android 安全性模型有三个主要方面：

- 权限
- 应用程序签名
- 应用程序用户 ID

Android 权限

Android 的许多功能由操作系统权限机制保护。为了使用某种保护的功能， AIR 应用程序描述符必须声明应用程序要求必要的权限。用户尝试安装应用程序时， Android 操作系统会在继续安装之前对用户显示所有请求的权限。

大多数 AIR 应用程序需要在应用程序描述符中指定 Android 权限。默认情况下，不包括任何权限。通过 AIR 运行时公开的受保护的 Android 功能需要下列权限：

ACCESS_COARSE_LOCATION 允许应用程序通过 Geolocation 类访问 WIFI 和移动电话网络位置数据。

ACCESS_FINE_LOCATION 允许应用程序通过 Geolocation 类访问 GPS 数据。

ACCESS_NETWORK_STATE 和 ACCESS_WIFI_STATE 允许应用程序通过 NetworkInfo 类访问网络信息。

CAMERA 允许应用程序访问摄像头。

INTERNET 允许应用程序提出网络请求。也允许远程调试。

READ_PHONE_STATE 允许 AIR 运行时在有来电时静音。

RECORD_AUDIO 允许应用程序访问麦克风。

WAKE_LOCK 和 DISABLE_KEYGUARD 允许应用程序使用 SystemIdleMode 类设置阻止设备休眠。

WRITE_EXTERNAL_STORAGE 允许应用程序写入设备上的外部存储卡。

应用程序签名

为 Android 平台创建的所有应用程序包都必须进行签名。由于 Android 上的 AIR 应用程序都以本机 Android APK 格式打包，因此根据 Android 约定而非 AIR 约定对其进行签名。尽管 Android 和 AIR 使用代码签名的方式相似，但存在显著区别:

- 在 Android 上，签名将验证开发人员是否持有私钥，但不用于验证开发人员的身份。
- 对于提交给 Android 市场的应用程序，证书必须至少在 25 年内有效。
- Android 不支持将包签名迁移到其他证书。如果更新通过其他证书进行签名，则用户必须卸载原始应用程序后才能安装更新的应用程序。
- 两个使用相同证书签名的应用程序可以指定一个共享的 ID，以便允许它们访问彼此的缓存和数据文件。（AIR 没有提供此类共享。）

应用程序用户 **ID**

Android 使用 Linux 内核。为每个安装的应用程序分配了 Linux 类型的用户 ID，该用户 ID 确定其进行文件访问等操作的权限。通过文件系统权限提供保护，防止对应用程序、应用程序存储和临时目录中的文件进行非授权访问。写入外部存储器 （即 SD 卡）的文件在 SD 卡作为大容量存储设备安装到计算机上时，可以被其他应用程序或用户读取、修改和删除。

通过 Internet 请求接收的 Cookie 不在 AIR 应用程序之间共享。

背景图像隐私

当用户将某个应用程序切换到后台时，一些 Android 版本会捕获它在最近应用程序列表中使用缩览图的屏幕快照。此屏幕快照存储在设备内存中，攻击者可以通过该设备的物理控件访问该快照。

如果您的应用程序显示敏感信息，则您应保护此类信息，防止被背景屏幕快照捕获。由 NativeApplication 对象调度的 deactivate 事件表示应用程序将要切换到背景。使用此事件可以清除或隐藏所有敏感信息。

更多帮助主题

Android：安全性和权限

Android 上的加密数据

Android 上的 AIR 应用程序可以使用内置 SQL 数据库中提供的加密选项保存加密数据。为提供最佳安全性，请将加密密钥基于用户在运行应用程序时输入的密码。本地存储的加密密钥或密码很难或者不可能对访问应用程序文件的攻击者 “ 隐藏 ”。如果攻击者可以检索密钥，那么除了 Android 系统提供的基于用户 ID 的文件系统安全外，加密数据不会提供任何附加保护。

EncryptedLocalStore 类可用于保存数据，但在 Android 设备上不对该数据进行加密。而 Android 安全模型则利用应用程序用户 ID 保护数据不被其他应用程序访问。使用共享用户 ID 并使用相同代码签名证书签名的应用程序使用相同的加密本地存储区。

重要说明：在根手机上，使用根权限运行的任何应用程序都可以访问任何其他应用程序的文件。因此，使用加密本地存储区存储的数据在根设备上并不安全。

在 iOS 设备上的安全性

在 iOS 上， AIR 符合本机安全模型。同时， AIR 保持自己的安全性规则，旨在使开发人员轻松编写安全的、与 Internet 连接的应用程序。

因为 iOS 上的 AIR 应用程序使用 iOS 软件包格式，安装属于 iOS 安全性模型。不使用 AIR 应用程序安装程序。此外，在 iOS 设备上不使用单独的 AIR 运行时。所有 AIR 应用程序都包含运行所需的全部代码。

应用程序签名

必须对所有针对 iOS 平台创建的应用程序包进行签名。因为 iOS 上的 AIR 应用程序都打包为本机 iOS IPA 格式，所以它们是根据 iOS 要求 （而不是 AIR 要求）进行签名的。虽然 iOS 和 AIR 以类似的方式使用代码签名，但也存在明显的差异：

- 在 iOS 上，用于对应用程序进行签名的证书必须由 Apple 颁发；无法使用来自其他证书颁发机构的证书。
- 在 iOS 上， Apple 颁发的分发证书的有效期一般为一年。

背景图像隐私

当用户将某个应用程序切换到 iOS 上的背景时，操作系统会捕获一个屏幕快照，它将使用该屏幕快照令过渡产生动画效果。此屏幕快照存储在设备内存中，攻击者可以通过该设备的物理控件访问该快照。

如果您的应用程序显示敏感信息，则您应保护此类信息，防止被背景屏幕快照捕获。由 NativeApplication 对象调度的 deactivate 事件表示应用程序将要切换到背景。使用此事件可以清除或隐藏所有敏感信息。

第 65 章：如何使用 ActionScript 示例

运行 ActionScript 3.0 代码示例是了解特定类和方法工作原理的最佳途径之一。您可以采用不同的方式使用示例，具体取决于您当前正在使用的设备或设定为目标的设备。

运行 Flash Professional 或 Flash Builder 的计算机 有关如何使用这些开发环境来运行 ActionScript 3.0 示例的信息，请参阅第 940 页的 " 在 Flash Professional 中运行 ActionScript 3.0 示例 " 或第 942 页的 " 在 Flash Builder 中运行 ActionScript 3.0 示例 "。使用 trace 语句和其他调试工具以增进您对代码示例工作原理的理解。

移动设备 可以在支持 Flash Player 10.1 和更高版本的移动设备上运行 ActionScript 3.0 代码示例。请参阅第 943 页的 " 在移动设备上运行 ActionScript 3.0 示例 "。也可以使用 Flash Professional 或 Flash Builder 在您的计算机上运行这些示例。

电视设备 虽然不能在电视设备上运行这些示例，但是仍然可以通过在您的计算机上运行这些示例来了解它们。有关针对电视设备开发应用程序的信息，请参阅 Adobe Developer Connection 网站上的 Flash Platform for TV。

示例类型

ActionScript 3.0 代码示例的类型包括：

- 代码片断示例 （位于 ActionScript 3.0 文档集中）
- 基于类的示例 （主要位于 ActionScript 3.0 语言参考中）
- 包含多个源文件的实用示例 （从 www.adobe.com/go/learn_programmingAS3samples_flash_cn 上下载源 ZIP 文件）

代码片断示例

代码片断示例类似于：

```
var x:int = 5;
trace(x); // 5
```

代码片断仅包含说明一种想法的代码。它们通常不包含包语句或类语句。

基于类的示例

许多示例用于显示完整的 ActionScript 类的源代码。基于类的示例类似于：

```
package {
    public class Example1 {
        public function Example1():void {
            var x:int = 5;
            trace(x); //5
        }
    }
}
```

基于类的示例的代码包括一个包语句、一个类声明和一个构造函数。

包含多个源文件的实用示例

《ActionScript 3.0 开发人员指南》中的许多主题都以实用示例来结束，这些示例显示了如何在实用的实际上下文中使用某些 ActionScript 功能。这些示例通常包含多个文件，包括：

- 一个或多个 ActionScript 源文件
- 用于 Flash Professional 的 .FLA 文件

- 用于 Flash Builder 的一个或多个 MXML 文件
- 示例应用程序使用的数据文件、图像文件、声音文件或其他资源 （可选）。

实用示例通常以 ZIP 归档文件的形式提供。

开发人员指南在 ZIP 中提供的示例列表

Flash Professional CS5 和 Flex 4 的 ZIP 文件（从 www.adobe.com/go/learn_programmingAS3samples_flash_cn 上下载）包含以下示例：

- AlarmClock （第 118 页的 " 事件处理示例：闹钟 "）
- AlgorithmicVisualGenerator （第 194 页的 " 绘制 API 示例：算法可视化生成器 "）
- ASCIIArt （第 17 页的 " 字符串示例：ASCII 图表 "）
- CapabilitiesExplorer （第 747 页的 " 功能示例：检测系统功能 "）
- CustomErrors （第 57 页的 " 处理错误示例：CustomErrors 应用程序 "）
- DisplayObjectTransformer （第 181 页的 " 几何形状示例：对显示对象应用矩阵转换 "）
- FilterWorkbench （第 244 页的 " 筛选显示对象示例：Filter Workbench"）
- GlobalStockTicker （第 815 页的 " 示例：国际化股票报价应用程序 "）
- IntrovertIM_HTML （第 727 页的 " 外部 API 示例：在 ActionScript 和 Web 浏览器中的 JavaScript 之间进行通信 "）
- NewsLayout （第 325 页的 "TextField 示例：报纸风格的文本格式设置 "）
- PlayList （第 39 页的 " 数组示例：播放列表 "）
- PodcastPlayer （第 394 页的 " 声音示例：Podcast Player"）
- ProjectionDragger （第 299 页的 " 示例：透视投影 "）
- ReorderByZ （第 303 页的 " 使用 Matrix3D 对象重新排序显示 "）
- RSSViewer （第 96 页的 " 在 ActionScript 中使用 XML 的示例：从 Internet 加载 RSS 数据 "）
- RuntimeAssetsExplorer （第 277 页的 " 影片剪辑示例：RuntimeAssetsExplorer"）
- SimpleClock （第 5 页的 " 日期和时间示例：简单模拟时钟 "）
- SpinningMoon （第 213 页的 " 位图示例：带动画效果的旋转的月亮 "）
- SpriteArranger （第 169 页的 " 显示对象示例：SpriteArranger"）
- TelnetSocket （第 685 页的 "TCP 套接字示例：构建 Telnet 客户端 "）
- VideoJukebox （第 429 页的 " 视频示例：视频自动唱片点唱机 "）
- WikiEditor （第 76 页的 " 正则表达式示例：Wiki 解析程序 "）
- WordSearch （第 492 页的 " 鼠标输入示例：WordSearch"）

Flash 开发人员中心和 Flex 开发人员中心的许多快速入门文章中也提供了实用示例。

在 Flash Professional 中运行 ActionScript 3.0 示例

要使用 Flash Professional 运行示例，请使用下列步骤之一 （具体取决于示例类型）。

在 **Flash Professional** 中运行代码片断示例

要在 Flash Professional 中运行代码片断示例，请执行以下操作：

1 选择 “ 文件 ”>“ 新建 ”。

2 在 “ 新建文档 ” 对话框中，选择 “Flash 文档 ”，然后单击 “ 确定 ”。

将显示一个新的 Flash 窗口。

3 在 “ 时间轴 ” 面板中，单击第一个图层的第一帧。

4 在 “ 动作 ” 面板中，键入或粘贴代码片断示例。

5 选择 “ 文件 ”>“ 保存 ”。为文件指定一个名称，然后单击 “ 确定 ”。

6 要测试该示例，请选择 “ 控制 ”>“ 测试影片 ”。

在 **Flash Professional** 中运行基于类的示例

要在 Flash Professional 中运行基于类的示例，请执行以下操作：

1 选择 “ 文件 ”>“ 新建 ”。

2 在 “ 新建文档 ” 对话框中，选择 “ActionScript 文件 ”，然后单击 “ 确定 ”。此时将显示一个新的编辑器窗口。

3 复制基于类的示例代码并将其粘贴到该编辑器窗口。

如果该类是程序的主文档类，则其必须扩展 MovieClip 类：

```
import flash.display.MovieClip;
public class Example1 extends MovieClip{
//...
}
```

此外，请确保示例中引用的所有类使用 import 语句进行了声明。

4 选择 “ 文件 ”>“ 保存 ”。为文件指定一个与示例中的类相同的名称 （例如 ContextMenuExample.as）。

注：某些基于类的示例 （如 flashx.textLayout.container.ContainerController 类示例）在包声明 (package flashx.textLayout.container.examples {) 中包含多个级别。对于这些示例，将文件保存在与包声明 (flashx/textLayout/container/examples) 匹配的子文件中，或删除包名称（以便 ActionScript 仅以 package { 开头），则可以从任意位置测试该文件。

5 选择 “ 文件 ”>“ 新建 ”。

6 在 “ 新建文档 ” 对话框中，选择 “Flash 文档 (ActionScript 3.0)”，然后单击 “ 确定 ”。将显示一个新的 Flash 窗口。

7 在 “ 属性 ” 面板中的 “ 文档类 ” 字段中，输入示例类的名称，此名称应与刚刚保存的 ActionScript 源文件的名称相匹配（例如 ContextMenuExample）。

8 选择 “ 文件 ”>“ 保存 ”。为 FLA 文件指定一个与示例中的类相同的名称 （例如 ContextMenuExample.fla）。

9 要测试该示例，请选择 “ 控制 ”>“ 测试影片 ”。

在 **Flash Professional** 中运行实用示例

实用示例通常以 ZIP 归档文件的形式提供。要使用 Flash Professional 运行实用示例，请执行以下操作：

1 将归档文件解压缩到所选的文件夹中。

2 在 Flash Professional 中，选择 “ 文件 ”>“ 打开 ”。

3 浏览到将归档文件解压缩到的文件夹中。在该文件夹中选择 FLA 文件并单击 “ 打开 ”。

4 要测试该示例，请选择 “ 控制 ”>“ 测试影片 ”。

在 Flash Builder 中运行 ActionScript 3.0 示例

要使用 Flash Builder 运行示例，请使用下列步骤之一 （具体取决于示例类型）。

在 **Flash Builder** 中运行代码片断示例

要在 Flash Builder 中运行代码片断示例，请执行以下操作：

1 创建一个新的 Flex 项目（选择 “ 文件 ”>“ 新建 ”>“Flex 项目 ”），或在现有的 Flex 项目中创建一个新的 MXML 应用程序（选择 “ 文件 ”>“ 新建 ”>“MXML 应用程序 ”）。为该项目或应用程序指定一个描述性名称 （例如 ContextMenuExample）。

2 在生成的 MXML 文件中，添加一个 <mx:Script> 标签。

3 将代码片断示例的内容粘贴到 <mx:Script> 和 </mx:Script> 标签之间。保存 MXML 文件。

4 要运行该示例，请为主 MXML 文件选择 “ 运行 ”>“ 运行 ” 菜单选项 （例如 “ 运行 ”>“ 运行 ContextMenuExample”）。

在 **Flash Builder** 中运行基于类的示例

要在 Flash Builder 中运行基于类的示例，请执行以下操作：

1 选择 “ 文件 ”>“ 新建 ”>“ActionScript 项目 ”。

2 将主类的名称（例如 ContextMenuExample）输入到 “ 项目名称 ” 字段中。为其他字段使用默认值（或根据特定环境对这些默认值进行更改）。单击 “ 完成 ” 创建项目和主 ActionScript 文件。

3 擦除任何从 ActionScript 文件生成的内容。将示例代码 （包括包语句和 import 语句）粘贴到 ActionScript 文件并保存该文件。

   注：某些基于类的示例 （如 flashx.textLayout.container.ContainerController 类示例）在包声明 (package flashx.textLayout.container.examples {) 中包含多个级别。对于这些示例，将文件保存在与包声明 (flashx/textLayout/container/examples) 匹配的子文件中，或删除包名称（以便 ActionScript 仅以 package { 开头），则可以从任意位置测试该文件。

4 要运行该示例，请为主 ActionScript 类名称选择 “ 运行 ”>“ 运行 ” 菜单选项 （例如 “ 运行 ”>“ 运行 ContextMenuExample”）。

在 **Flash Builder** 中运行实用示例

实用示例通常以 ZIP 归档文件的形式提供。要使用 Flash Builder 运行实用示例，请执行以下操作：

1 将归档文件解压缩到所选的文件夹中。为该文件夹指定一个描述性名称 （例如 ContextMenuExample）。

2 在 Flash Builder 中，选择“文件”>“新建 Flex 项目”。在“项目位置”部分，单击“浏览”并选择包含示例文件的文件夹。在“项目名称 ” 字段中输入该文件夹的名称 （例如 ContextMenuExample）。为其他字段使用默认值 （或根据特定环境对这些默认值进行更改）。单击 “ 下一步 ” 继续。

3 在 “ 输出 ” 面板中单击 “ 下一步 ” 接受默认值。

4 在“源路径”面板中单击“主应用程序文件”字段旁的“浏览”按钮。从示例文件夹中选择主 MXML 示例文件。单击“完成”创建项目文件。

5 要运行该示例，请为主 MXML 文件选择 “ 运行 ”>“ 运行 ” 菜单选项 （例如 “ 运行 ”>“ 运行 ContextMenuExample”）。

在移动设备上运行 ActionScript 3.0 示例

您可以在支持 Flash Player 10.1 的移动设备上运行 ActionScript 3.0 代码示例。不过，通常，运行代码示例是为了了解特定类和方法是如何运作的。但在该案例中，是在非移动设备 （例如桌面计算机）上运行示例的。在桌面计算机上，您可以使用 Flash Professional 或 Flash Builder 中的 trace 语句和其他调试工具来提高您对代码示例的理解。

如果希望在移动设备上运行示例，则可以将文件复制到该设备上或复制到 Web 服务器上。要将文件复制到该设备上并在浏览器中运行示例，请执行下列操作：

1 按照第 940 页的 " 在 Flash Professional 中运行 ActionScript 3.0 示例 " 或第 942 页的 " 在 Flash Builder 中运行 ActionScript 3.0 示例 " 中的说明创建 SWF 文件。在 Flash Professional 中，应在选择 " 控制 ">" 测试影片 " 时创建 SWF 文件。在 Flash Builder 中，应在运行、调试或构建 Flash Builder 项目时创建 SWF 文件。

2 将 SWF 文件复制到移动设备上的一个目录中。使用随设备提供的软件复制文件。

3 在移动设备上的浏览器的地址栏中，输入 SWF 文件的 file:// URL。例如，输入 file:://applications/myExample.swf。

要将文件复制到 Web 服务器上并在设备的浏览器中运行示例，请执行下列操作：

1 创建一个 SWF 文件和一个 HTML 文件。首先，请按第 940 页的 " 在 Flash Professional 中运行 ActionScript 3.0 示例 " 或第 942 页的 " 在 Flash Builder 中运行 ActionScript 3.0 示例 " 中的说明操作。在 Flash Professional 中，选择 " 控制 ">" 测试影片 " 仅会创建 SWF 文件。要同时创建这两个文件，首先在 " 发布设置 " 对话框中的 " 格式 " 选项卡上同时选中 Flash 和 HTML。然后选择 " 文件 ">" 发布 " 以同时创建 HTML 和 SWF 文件。在 Flash Builder 中，应在运行、调试或构建 Flash Builder 项目时创建 SWF 文件和 HTML 文件。

2 将 SWF 文件和 HTML 文件复制到 Web 服务器中的一个目录下。

3 在移动设备上的浏览器的地址栏中，输入 HTML 文件的 HTTP 地址。例如，输入 http://www.myWebServer/examples/myExample.html。

在移动设备上运行示例之前，请考虑下面的每个问题。

舞台大小

在移动设备上运行示例时的舞台要比在使用非移动设备时的舞台小得多。许多示例不要求特定的舞台大小。创建 SWF 文件时，应指定适合设备的舞台大小。例如，指定为 176 x 208 像素。

《ActionScript 3.0 开发指南》中的实用示例旨在阐明不同的 ActionScript 3.0 概念和类。这些示例的用户界面设计得很美观，而且在桌面计算机或便携式计算机上运作良好。虽然这些示例可以在移动设备上运行，但其舞台大小和用户界面设计不太适合小屏幕。 Adobe 建议您在计算机上运行这些实用示例来了解 ActionScript，然后在您的移动应用程序中使用相关的代码片断。

文本字段而不是 **trace** 语句

在移动设备上运行示例时，无法显示示例的 trace 语句的输出。要查看输出，请创建 TextField 类的一个实例。然后，将 trace 语句的文本附加到该文本字段的 text 属性中。

您可以使用以下函数设置用于跟踪的文本字段：

```
function createTracingTextField(x:Number, y:Number,
                                width:Number, height:Number):TextField {

    var tracingTF:TextField = new TextField();
    tracingTF.x = x;
    tracingTF.y = y;
    tracingTF.width = width;
    tracingTF.height = height;

    // A border lets you more easily see the area the text field covers.
    tracingTF.border = true;
    // Left justifying means that the right side of the text field is automatically
    // resized if a line of text is wider than the width of the text field.
    // The bottom is also automatically resized if the number of lines of text
    // exceed the length of the text field.
    tracingTF.autoSize = TextFieldAutoSize.LEFT;

    // Use a text size that works well on the device.
    var myFormat:TextFormat = new TextFormat();
    myFormat.size = 18;
    tracingTF.defaultTextFormat = myFormat;

    addChild(tracingTF);
    return tracingTF;
}
```

例如，将此函数作为私有函数添加到文档类中。然后，在文档类的其他方法中，跟踪数据，代码类似以下内容：

```
var traceField:TextField = createTracingTextField(10, 10, 150, 150);
// Use the newline character "\n" to force the text to the next line.
traceField.appendText("data to trace\n");
traceField.appendText("more data to trace\n");
// Use the following line to clear the text field.
traceField.appendText("");
```

appendText() 方法只接受一个值作为参数。该值是字符串 （String 实例或字符串文本）。要输出非字符串变量的值，首先将该值转换为字符串。最简单的方法是调用对象的 toString() 方法：

```
var albumYear:int = 1999;
traceField.appendText("albumYear = ");
traceField.appendText(albumYear.toString());
```

文本大小

许多示例使用文本字段来帮助阐明概念。有时，调整文本字段中文本的大小可以提高移动设备上的可读性。例如，如果示例使用名为 myTextField 的文本字段实例，则可以使用下列代码更改其文本的大小：

```
// Use a text size that works well on the device.
var myFormat:TextFormat = new TextFormat();
myFormat.size = 18;
myTextField.defaultTextFormat = myFormat
```

捕获用户输入

移动操作系统和浏览器将捕获 SWF 内容没有接收到的一些用户输入事件。具体的行为要取决于操作系统和浏览器，但在移动设备上运行示例时，可能会导致意外行为。有关详细信息，请参阅第 478 页的 “KeyboardEvent 优先级 ”。

此外，许多示例的用户界面是针对桌面计算机或便携式计算机设计的。例如，《ActionScript 3.0 开发人员指南》中的大多数实用示例适合通过桌面观看。因此，有时在移动设备的屏幕上看不见整个舞台。能否显示浏览器中的内容具体还要取决于浏览器。而且，这些示例的设计初衷不是用来捕捉和处理滚动或平移事件。因此，有些示例的用户界面不适合在小型屏幕上运行。Adobe 建议您在计算机上运行这些示例来了解 ActionScript，然后在您的移动应用程序中使用相关的代码片断。

有关详细信息，请参阅第 148 页的 “ 平移和滚动显示对象 ”。

处理焦点

一些示例要求您将焦点置于一区域。通过将焦点置于一区域，您可以完成某些任务，例如，输入文本或选择按钮。要指定区域焦点，请使用移动设备的指针设备，例如笔针或您的手指。或者，使用移动设备的导航键指定区域焦点。要选择有焦点的按钮，请使用移动设备的 Select 键，就像使用计算机上的 Enter 一样。在某些设备上，敲两次按钮才能将其选中。

有关焦点的详细信息，请参阅第 473 页的 " 管理焦点 "。

处理鼠标事件

许多示例侦听鼠标事件。例如，在计算机上，当用户将鼠标移动到某个显示对象上方或在某个显示对象上单击鼠标按钮时会发生这些鼠标事件。在移动设备上，使用指针设备（如笔针或手指）的事件称为触摸事件。Flash Player 10.1 已将触摸事件映射到鼠标事件。这种映射可以确保在 Flash Player 10.1 之前开发的 SWF 内容可以继续正常使用。因此，当使用指针设备选择或拖动显示对象时示例可正常工作。

性能

移动设备的处理能力没有桌面设备强。一些使 CPU 高负载的示例在移动设备上的执行速度可能会很慢。例如，第 194 页的 " 绘制 API 示例：算法可视化生成器 " 中的示例每输入一个帧便会执行大量计算和绘图。在计算机上运行此示例可以阐示多种绘图 API。但是，由于移动设备的性能限制，此示例不适合在一些移动设备上运行。

有关移动设备上的性能的详细信息，请参阅优化 Flash Platform 的性能。

最佳做法

这些示例不考虑开发适用于移动设备的应用程序的最佳做法。您需要特别注意移动设备在内存和处理能力方面的限制。同样，小型屏幕的用户界面的要求也与桌面显示不同。有关开发用于移动设备的应用程序的详细信息，请参阅优化 Flash Platform 的性能。

第 66 章：本地数据库中的 SQL 支持

Adobe AIR 包括一个 SQL 数据库引擎，该引擎使用开放源代码 SQLite 数据库系统，支持具有许多标准 SQL 功能的本地 SQL 数据库。运行时未指定在文件系统上存储数据库数据的方式或位置。每个数据库都完全存储在单个文件中。开发人员可指定数据库文件在文件系统中的存储位置，单个 AIR 应用程序可访问一个或多个单独的数据库 （即单独的数据库文件）。本文档概述了 SQL 语法和支持 Adobe AIR 本地 SQL 数据库的数据类型。本文档并不用作综合的 SQL 参考，而仅介绍有关 Adobe AIR 支持的 SQL 方言的详细信息。运行时支持大多数符合 SQL-92 标准的 SQL 方言。由于可以通过众多的参考资料、网站、书籍和培训材料来学习 SQL，因此本文档并不用作综合的 SQL 参考或教程。相反，本文档特别侧重于 AIR 支持的 SQL 语法，以及 SQL-92 和支持的 SQL 方言之间的差异。

SQL 语句定义约定

在本文档的语句定义中，使用了以下约定：

- 文本大小写
  - UPPER CASE — 文本 SQL 关键字以全大写形式书写。
  - lower case — 占位符项或子句名称以全小写形式书写。
- 定义字符
  - ::= - 指示一个子句或语句定义。
- 分组和替代字符
  - | - 管道字符在备选选项之间使用，可读作 “ 或 ”。
  - [] - 中括号中的项是可选项；括号中可包含单个项或一组替代项。
  - () - 括住一组替代项 （一组由管道字符分隔的项）的圆括号指定一组必需项，即作为单个必需项的可能值的一组项。
- 数量表示符
  - + - 圆括号中的项后面的加号字符指示前面的项可出现 1 次或更多次。
  - * - 中括号中的项后面的星号字符指示前面的 （在括号中的）项可出现 0 次或更多次
- 文本字符
  - * - 在列名中或函数名称后面的圆括号之间使用的星号，表示文本星号字符，而不是 “0 个或更多个 ” 数量表示符。
  - . 句点字符表示文本句点。
  - , - 逗号字符表示文本逗号。
  - () - 括住单个子句或项的一对圆括号指示圆括号是必需的文本圆括号字符。
  - 其他字符 - 除非另行说明，否则其他字符均表示相应的文本字符。

支持的 SQL 语法

本部分介绍 Adobe AIR SQL 数据库引擎支持的 SQL 语法。下面所列的各项分别用于说明不同的语句和子句类型、表达式、内置函数和运算符。本部分包含以下主题：

- 常规 SQL 语法
- 数据操作语句 （SELECT、 INSERT、 UPDATE 和 DELETE）
- 数据定义语句 （用于表、索引、视图和触发器的 CREATE、 ALTER 和 DROP 语句）

- 特殊的语句和子句
- 内置函数 （聚合函数、标量函数和日期 / 时间格式函数）
- 运算符
- 参数
- 不支持的 SQL 功能
- 其他 SQL 功能

常规 SQL 语法

除了各种语句和表达式的特定语法外，以下是 SQL 语法的一般规则：

区分大小写 SQL 语句 （包括对象名称）不区分大小写。但是， SQL 语句经常以大写形式的 SQL 关键字编写，本文档使用了该约定。尽管 SQL 语法不区分大小写，但是 SQL 中的文本值区分大小写，而且比较和排序操作可区分大小写，如为列或操作定义的排序规则序列所指定的。有关详细信息，请参阅 COLLATE。

空白 必须使用空白字符 （如空格、制表符、换行符等）来分隔 SQL 语句中的各个单词。但是，单词和符号之间的空白是可选的。 SQL 语句中空白字符的类型和数量并不重要。可使用空白 （如缩进和换行符）来设置 SQL 语句的格式以便于阅读，而不影响语句的含义。

数据操作语句

数据操作语句是最常用的 SQL 语句。这些语句用于从数据库表检索、添加、修改和删除数据。支持以下数据操作语句：SELECT、 INSERT、 UPDATE 和 DELETE。

SELECT

SELECT 语句用于查询数据库。 SELECT 的结果是零行或多行数据，其中每行都具有固定的列数。结果中的列数由 result 列名称或 SELECT 和可选 FROM 关键字之间的表达式列表指定。

```
sql-statement   ::=  SELECT [ALL | DISTINCT] result
                     [FROM table-list]
                     [WHERE expr]
                     [GROUP BY expr-list]
                     [HAVING expr]
                     [compound-op select-statement]*
                     [ORDER BY sort-expr-list]
                     [LIMIT integer [( OFFSET | , ) integer]]
result          ::=  result-column [, result-column]*
result-column   ::=  * | table-name . * | expr [[AS] string]
table-list      ::=  table [ join-op table join-args ]*
table           ::=  table-name [AS alias] |
                     ( select ) [AS alias]
join-op         ::=  , | [NATURAL] [LEFT | RIGHT | FULL] [OUTER | INNER | CROSS] JOIN
join-args       ::=  [ON expr] [USING ( id-list )]
compound-op     ::=  UNION | UNION ALL | INTERSECT | EXCEPT
sort-expr-list  ::=  expr [sort-order] [, expr [sort-order]]*
sort-order      ::=  [COLLATE collation-name] [ASC | DESC]
collation-name  ::=  BINARY | NOCASE
```

任意的表达式都可用作结果。如果结果表达式是 *，则以所有表的所有列替换该表达式。如果表达式是表名后跟 .*，则结果是该表中的所有列。

DISTINCT 关键字可导致返回结果行的子集，其中每个结果行都不同。各 NULL 值不被视为彼此不同。默认行为是返回所有结果行，这可通过关键字 ALL 明确指定。

对在 FROM 关键字后指定的一个或多个表执行查询。如果多个表名由逗号分隔，则查询将使用各个表的交叉联接。 JOIN 语法还可用于指定如何联接表。支持的唯一一个外部联接类型是 LEFT OUTER JOIN。join-args 中的 ON 子句表达式必须解析为布尔值。括号中的子查询可用作 FROM 子句中的表。可省略整个 FROM 子句，在此情况下结果是由 result 表达式列表的值组成的单个行。

WHERE 子句用于限制查询所检索的行数。WHERE 子句表达式必须解析为布尔值。WHERE 子句筛选是在任何分组之前执行的，因此 WHERE 子句表达式不能包括聚合函数。

GROUP BY 子句导致将结果的一行或多行合并到输出的单个行中。当结果包含聚合函数时， GROUP BY 子句尤其有用。GROUP BY 子句中的表达式不必是出现在 SELECT 表达式列表中的表达式。

HAVING 子句与 WHERE 类似，因为它限制语句返回的行数。但是， HAVING 子句在发生由 GROUP BY 子句指定的任何分组后应用。因此，HAVING 表达式可能引用包括聚合函数的值。不要求 HAVING 子句表达式出现在 SELECT 列表中。与 WHERE 表达式一样， HAVING 表达式必须解析为布尔值。

ORDER BY 子句导致对输出行进行排序。ORDER BY 子句的 sort-expr-list 参数是用作排序关键字的表达式列表。对于简单的 SELECT，这些表达式不必是结果的一部分，但是在复合 SELECT （使用 compound-op 运算符之一的 SELECT）中，每个排序表达式都必须与结果列之一完全匹配。每个排序表达式可能 （可选）后跟 sort-order 子句，该子句包含 COLLATE 关键字以及用于对文本进行排序的排序规则函数的名称和 / 或用于指定排序顺序 （升序或降序）的关键字 ASC 或 DESC。可省略排序顺序，在此情况下将使用默认值 （升序）。有关 COLLATE 子句和排序规则函数的定义，请参阅 COLLATE。

LIMIT 子句为结果中返回的行数设定上限。负的 LIMIT 指示无上限。LIMIT 后面的可选 OFFSET 指定要在结果集开头跳过的行数。在复合 SELECT 查询中，LIMIT 子句可能仅出现在最终 SELECT 语句之后，并且限制应用于整个查询。请注意，如果在 LIMIT 子句中使用 OFFSET 关键字，则限制是第一个整数，偏移量是第二个整数。如果使用逗号而不是 OFFSET 关键字，则偏移量是第一个数，限制是第二个数。此表面上的矛盾是有意的 — 这最大限度提高了与早期 SQL 数据库系统的兼容性。

复合 SELECT 是由运算符 UNION、UNION ALL、INTERSECT 或 EXCEPT 之一所连接的两个或更多个简单 SELECT 语句构成的。在复合 SELECT 中，所有用于构成的 SELECT 语句都必须指定相同数量的结果列。在最终 SELECT 语句之后只能有一个 ORDER BY 子句（如果指定了单个 LIMIT 子句，还要在这类子句之前）。UNION 和 UNION ALL 运算符将前面和后面的 SELECT 语句的结果组合到单个表中。两者的差异在于，在 UNION 中，所有结果行都不重复，但是在 UNION ALL 中，则可能存在重复行。 INTERSECT 运算符求出前面和后面的 SELECT 语句的结果的交集。在删除后面的 SELECT 的结果后，EXCEPT 求出前面的 SELECT 的结果。将三个或更多 SELECT 语句连接为复合语句时，它们从第一个到最后一个进行组合。

有关允许的表达式的定义，请参阅表达式。

从 AIR 2.5 开始，为了将 BLOB 数据转换为 ActionScript ByteArray 对象而进行读取时，支持 SQL CAST 运算符。例如，以下代码读取不以 AMF 格式存储的原始数据并将该数据存储在 ByteArray 对象中：

```
stmt.text = "SELECT CAST(data AS ByteArray) AS data FROM pictures;";
stmt.execute();
var result:SQLResult = stmt.getResult();
var bytes:ByteArray = result.data[0].data;
```

INSERT

INSERT 语句有两种基本形式，它用于用数据填充表。

```
sql-statement  ::=  INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-list)] VALUES
(value-list) |
                    INSERT [OR conflict-algorithm] INTO [database-name.] table-name [(column-list)] select-
statement
                     REPLACE INTO [database-name.] table-name [(column-list)] VALUES (value-list) |
                     REPLACE INTO [database-name.] table-name [(column-list)] select-statement
```

第一种形式 （使用 VALUES 关键字）在现有表中创建单个新行。如果未指定 column-list，则值的数量必须等于表中的列数。如果指定了 column-list，则值的数量必须与指定列的数量匹配。将用创建表时定义的默认值填充未出现在列的列表中的表列；如果未定义默认值，则使用 NULL 填充。

INSERT 语句的第二种形式从 SELECT 语句提取其数据。如果未指定 column-list，则 SELECT 的结果中的列数必须与表中的列数完全匹配，或者它必须与在 column-list 中指定的列数完全匹配。在表中为 SELECT 结果的每一行创建一个新项。SELECT 可能是简单的或复合的。有关允许的 SELECT 语句的定义，请参阅 SELECT。

可选的 conflict-algorithm 允许指定要在此命令过程中使用的替代约束冲突解决算法。有关冲突算法的解释和定义，请参阅第 954 页的 " 特殊的语句和子句 "。

该语句的两种 REPLACE INTO 形式等效于将标准的 INSERT [OR conflict-algorithm] 形式与 REPLACE 冲突算法一起使用 （即 INSERT OR REPLACE... 形式）。

该语句的两种 REPLACE INTO 形式等效于将标准的 INSERT [OR conflict-algorithm] 形式与 REPLACE 冲突算法一起使用 （即 INSERT OR REPLACE... 形式）。

UPDATE

update 命令可更改表中的现有记录。

```
sql-statement  ::=  UPDATE [database-name.] table-name SET column1=value1, column2=value2,... [WHERE expr]
```

该命令由 UPDATE 关键字后跟要更新记录的表的名称组成。在 SET 关键字之后，采用逗号分隔的列表形式提供列名称以及要将该列更改为的值。 WHERE 子句表达式提供要更改其记录的一个或多个行。

DELETE

delete 命令用于从表中删除记录。

```
sql-statement  ::=  DELETE FROM [database-name.] table-name [WHERE expr]
```

该命令由 DELETE FROM 关键字后跟要从其删除记录的表的名称组成。

如果没有 WHERE 子句，则删除表的所有行。如果提供了 WHERE 子句，则仅删除与表达式匹配的那些行。 WHERE 子句表达式必须解析为布尔值。有关允许的表达式的定义，请参阅表达式。

数据定义语句

数据定义语句用于创建、修改和删除数据库对象，如表、视图、索引和触发器。支持以下数据定义语句：

- 表：
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- 索引：
  - CREATE INDEX
  - DROP INDEX
- 视图：
  - CREATE VIEWS
  - DROP VIEWS
- 触发器：
  - CREATE TRIGGERS
  - DROP TRIGGERS

CREATE TABLE

CREATE TABLE 语句由关键字 CREATE TABLE 后跟新表的名称以及列定义和约束的列表（在括号中）组成。表名可以是标识符或字符串。

```
sql-statement       ::=  CREATE [TEMP | TEMPORARY] TABLE [IF NOT EXISTS] [database-name.] table-name
                         ( column-def [, column-def]* [, constraint]* )
sql-statement       ::=  CREATE [TEMP | TEMPORARY] TABLE [database-name.] table-name AS select-statement
column-def          ::=  name [type] [[CONSTRAINT name] column-constraint]*
type                ::=  typename | typename ( number ) | typename ( number , number )
column-constraint   ::=  NOT NULL [ conflict-clause ] |
                         PRIMARY KEY [sort-order] [ conflict-clause ] [AUTOINCREMENT] |
                         UNIQUE [conflict-clause] |
                         CHECK ( expr ) |
                         DEFAULT default-value |
                         COLLATE collation-name
constraint          ::=  PRIMARY KEY ( column-list ) [conflict-clause] |
                         UNIQUE ( column-list ) [conflict-clause] |
                         CHECK ( expr )
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-algorithm  ::=  ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
default-value       ::=  NULL | string | number | CURRENT_TIME | CURRENT_DATE | CURRENT_TIMESTAMP
sort-order          ::=  ASC | DESC
collation-name      ::=  BINARY | NOCASE
column-list         ::=  column-name [, column-name]*
```

每个列定义都是列的名称后跟该列的数据类型，然后是一个或多个可选的列约束。列的数据类型限制可在该列中存储的数据。如果尝试在具有不同数据类型的列中存储某个值，则运行时会将该值转换为相应的类型 （如果可能），或者引发错误。有关其他信息，请参阅数据类型支持部分。

NOT NULL 列约束指示列不能包含 NULL 值。

UNIQUE 约束导致在指定的一个或多个列上创建索引。此索引必须包含唯一键 — 没有任何两行可以包含重复值或者指定的一个或多个列的值的组合。CREATE TABLE 语句可具有多个 UNIQUE 约束（包括列定义中有 UNIQUE 约束的多个列）和 / 或多个表级 UNIQUE 约束。

CHECK 约束定义计算结果必须为 true 的表达式，以便插入或更新行的数据。 CHECK 表达式必须解析为布尔值。

列定义中的 COLLATE 子句指定在比较列的文本项时要使用的文本排序规则函数。默认情况下，使用 BINARY 排序规则函数。有关 COLLATE 子句和排序规则函数的详细信息，请参阅 COLLATE。

DEFAULT 约束指定执行 INSERT 时要使用的默认值。该值可能是 NULL、字符串常量或数字。默认值还可能是与大小写无关的特殊关键字 CURRENT_TIME、 CURRENT_DATE 或 CURRENT_TIMESTAMP 之一。如果该值是 NULL、字符串常量或数字，则只要 INSERT 语句不指定列的值，则按字面将它插入到列中。如果该值是 CURRENT_TIME、CURRENT_DATE 或 CURRENT_TIMESTAMP，则将当前的 UTC 日期和 / 或时间插入到列中。对于 CURRENT_TIME，格式为 HH:MM:SS。对于 CURRENT_DATE，格式为 YYYY-MM-DD。CURRENT_TIMESTAMP 的格式为 YYYY-MM-DD HH:MM:SS。

指定 PRIMARY KEY 通常仅在对应的一个或多个列上创建 UNIQUE 索引。但是，如果 PRIMARY KEY 约束在具有数据类型 INTEGER （或它的一个同义词，如 int）的单个列上，则数据库会将该列用作表的实际主键。这意味着该列只能保存唯一整数值。（注意：在许多 SQLite 实现中，只有列类型 INTEGER 会导致列用作内部主键，但在 Adobe AIR 中， INTEGER 的同义词，如 int，也会指定该行为。）

如果表没有 INTEGER PRIMARY KEY 列，则在插入行时将自动生成整数键。使用特殊名称 ROWID、 OID 或 _ROWID_ 之一，始终可以访问行的主键。可使用这些名称，而不管它是明确声明的 INTEGER PRIMARY KEY 还是内部生成的值。但是，如果表具有显式 INTEGER PRIMARY KEY，则结果数据中的列名称是实际列名称，而不是特殊名称。

INTEGER PRIMARY KEY 列还可包括关键字 AUTOINCREMENT。如果使用 AUTOINCREMENT 关键字，则在执行未指定列的显式值的 INSERT 语句时，数据库会在 INTEGER PRIMARY KEY 列中自动生成并插入按顺序递增的整数键。

CREATE TABLE 语句中只能有一个 PRIMARY KEY 约束。它可以是一个列的定义的一部分或一个单表级 PRIMARY KEY 约束。主键列隐式为 NOT NULL。

许多约束后面的可选 conflict-clause 允许为该约束指定替代的默认约束冲突解决算法。默认值为 ABORT。同一表中的不同约束可能具有不同的默认冲突解决算法。如果 INSERT 或 UPDATE 语句指定不同的冲突解决算法，则使用该算法，替代在 CREATE TABLE 语句中指定的算法。请参阅第 954 页的 " 特殊的语句和子句 " 的 ON CONFLICT 部分，了解更多信息。

其他约束 （如 FOREIGN KEY 约束）不会导致错误，但运行时会忽略它们。

如果在 CREATE 和 TABLE 之间出现 TEMP 或 TEMPORARY 关键字，则创建的表仅在同一数据库连接（SQLConnection 实例）内才是可见的。关闭数据库连接时会自动删除它。在临时表上创建的任何索引也是临时的。临时的表和索引存储在与主数据库文件不同的单独文件中。

如果指定可选的 database-name 前缀，则在指定的数据库 （通过使用指定的数据库名称调用 attach() 方法连接到 SQLConnection 实例的数据库）中创建表。除非 database-name 前缀是 temp，否则同时指定 database-name 前缀和 TEMP 关键字是错误的。如果未指定数据库名称，而且不存在 TEMP 关键字，则在主数据库（使用 open() 或 openAsync() 方法连接到 SQLConnection 实例的数据库）中创建表。

对表中的列数或约束数没有任何限制。对行中的数据量也没有任何限制。

CREATE TABLE AS 形式将表定义为查询的结果集。表列的名称是结果中列的名称。

如果存在可选的 IF NOT EXISTS 子句，而且同名的其他表已存在，则数据库将忽略 CREATE TABLE 命令。

可使用 DROP TABLE 语句删除表，并可使用 ALTER TABLE 语句进行有限的更改。

ALTER TABLE

ALTER TABLE 命令允许用户重命名现有表或向其添加新列。无法从表中删除列。

```
sql-statement ::= ALTER TABLE [database-name.] table-name alteration
alteration    ::= RENAME TO new-table-name
alteration    ::= ADD [COLUMN] column-def
```

RENAME TO 语法用于将由 [database-name.] table-name 标识的表重命名为 new-table-name。此命令不能用于在附加的数据库之间移动表，而只能用于重命名同一数据库中的表。

如果要重命名的表具有触发器或索引，则在重命名表后，这些触发器或索引仍然附加到表。但是，如果存在任何视图定义或由引用要重命名的表的触发器执行的语句，则不会自动修改它们以使用新表名。如果重命名的表具有关联的视图或触发器，则必须手动删除再重新创建使用新表名的触发器或视图定义。

ADD [COLUMN] 语法用于向现有表添加新列。新列始终追加到现有列的列表的结尾。 column-def 子句可采用 CREATE TABLE 语句中允许的任何形式，但有以下限制：

- 列不能具有 PRIMARY KEY 或 UNIQUE 约束。
- 列不能具有默认值 CURRENT_TIME、 CURRENT_DATE 或 CURRENT_TIMESTAMP。
- 如果指定了 NOT NULL 约束，则列必须具有除 NULL 之外的默认值。

ALTER TABLE 语句的执行时间不受表中数据量的影响。

DROP TABLE

DROP TABLE 语句删除使用 CREATE TABLE 语句添加的表。具有指定 table-name 的表就是所删除的表。它将从数据库和磁盘文件中完全删除。无法恢复表。与表关联的所有索引也将被删除。

```
sql-statement  ::=  DROP TABLE [IF EXISTS] [database-name.] table-name
```

默认情况下， DROP TABLE 语句不减小数据库文件的大小。将保留数据库中的空白空间，并在后续 INSERT 操作中使用。若要删除数据库中的可用空间，请使用 SQLConnection.clean() 方法。如果在最初创建数据库时 autoClean 参数设置为 true，则将自动释放空间。

可选的 IF EXISTS 子句抑制表不存在时通常会导致的错误。

CREATE INDEX

CREATE INDEX 命令包含关键字 CREATE INDEX，后跟新索引的名称、关键字 ON、之前创建的要编制索引的表的名称，以及表中其值用于索引键的列的名称的带括号列表。

```
sql-statement  ::=  CREATE [UNIQUE] INDEX [IF NOT EXISTS] [database-name.] index-name
                    ON table-name ( column-name [, column-name]* )
column-name    ::=  name [COLLATE collation-name] [ASC | DESC]
```

每个列名都可后跟 ASC 或 DESC 关键字以指示排序顺序，但运行时会忽略指定的排序顺序。排序始终按升序执行。

每个列名后面的 COLLATE 子句定义用于该列中文本值的排序规则序列。默认的排序规则序列是在 CREATE TABLE 语句中为该列定义的排序规则序列。如果未指定排序规则序列，则使用 BINARY 排序规则序列。有关 COLLATE 子句和排序规则函数的定义，请参阅 COLLATE。

对于可附加到单个表的索引数没有任何限制。对于索引中的列数也没有限制。

DROP INDEX

drop index 语句删除使用 CREATE INDEX 语句添加的语句。指定的索引将从数据库文件中完全删除。恢复索引的唯一方法是，重新输入相应的 CREATE INDEX 命令。

```
sql-statement ::= DROP INDEX [IF EXISTS] [database-name.] index-name
```

默认情况下， DROP INDEX 语句不减小数据库文件的大小。将保留数据库中的空白空间，并在后续 INSERT 操作中使用。若要删除数据库中的可用空间，请使用 SQLConnection.clean() 方法。如果在最初创建数据库时 autoClean 参数设置为 true，则将自动释放空间。

CREATE VIEW

CREATE VIEW 命令为预定义的 SELECT 语句分配一个名称。之后，此新名称可用于其他 SELECT 语句的 FROM 子句中来代替表名。视图通常用于简化查询，方法是将一组复杂的 （和频繁使用的）数据组合到可在其他操作中使用的结构中。

```
sql-statement ::= CREATE [TEMP | TEMPORARY] VIEW [IF NOT EXISTS] [database-name.] view-name AS select-
statement
```

如果在 CREATE 和 VIEW 之间出现 TEMP 或 TEMPORARY 关键字，则创建的视图仅对已打开数据库的 SQLConnection 实例是可见的，并且在关闭数据库时会自动删除该视图。

如果指定了 [database-name]，则使用指定的 name 参数在指定的数据库 （使用 attach() 方法连接到 SQLConnection 实例的数据库）中创建视图。除非 [database-name] 是 temp，否则同时指定 [database-name] 和 TEMP 关键字是错误的。如果未指定数据库名称，而且不存在 TEMP 关键字，则在主数据库（使用 open() 或 openAsync() 方法连接到 SQLConnection 实例的数据库）中创建视图。

视图是只读的。除非至少定义一个关联类型 （INSTEAD OF DELETE、 INSTEAD OF INSERT、 INSTEAD OF UPDATE）的触发器，否则不能对视图使用 DELETE、INSERT 或 UPDATE 语句。有关为视图创建触发器的信息，请参阅 CREATE TRIGGER。

使用 DROP VIEW 语句可从数据库中删除视图。

DROP VIEW

DROP VIEW 语句删除由 CREATE VIEW 语句创建的视图。

```
sql-statement ::= DROP VIEW [IF EXISTS] view-name
```

指定的 view-name 是要删除的视图的名称。将从数据库中删除它，但不会修改基础表中的数据。

CREATE TRIGGER

create trigger 语句用于向数据库架构添加触发器。触发器是在发生指定的数据库事件 (database-event) 时自动执行的数据库操作 (trigger-action)。

```
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-name
                     [BEFORE | AFTER] database-event
                     ON table-name
                     trigger-action
sql-statement   ::=  CREATE [TEMP | TEMPORARY] TRIGGER [IF NOT EXISTS] [database-name.] trigger-name
                     INSTEAD OF database-event
                     ON view-name
                     trigger-action
database-event  ::=  DELETE |
                     INSERT |
                     UPDATE |
                     UPDATE OF column-list
trigger-action  ::=  [FOR EACH ROW] [WHEN expr]
                     BEGIN
                       trigger-step ;
                       [ trigger-step ; ]*
                     END
trigger-step    ::=  update-statement |
                     insert-statement |
                     delete-statement |
                     select-statement
column-list     ::=  column-name [, column-name]*
```

触发器被指定为只要出现以下条件就会激发：发生特定数据库表的 DELETE、 INSERT 或 UPDATE，或者更新了表的一个或多个指定列的 UPDATE。除非使用了 TEMP 或 TEMPORARY 关键字，否则触发器是永久性的。在这种情况下，当关闭 SQLConnection 实例的主数据库连接时，将删除触发器。如果未指定时间 （BEFORE 或 AFTER），则触发器默认为 BEFORE。

仅支持 FOR EACH ROW 触发器，因此 FOR EACH ROW 文本是可选的。对于 FOR EACH ROW 触发器，如果 WHEN 子句表达式的计算结果为 true，则对导致触发器激发的语句所插入、更新或删除的每个数据库行执行 trigger-step 语句。

如果提供了 WHEN 子句，则仅对 WHEN 子句为 true 的行执行指定为触发器步骤的 SQL 语句。如果未提供 WHEN 子句，则对所有行执行 SQL 语句。

在触发器体 （trigger-action 子句）中，受影响表的更改之前值和更改之后值使用特殊的表名 OLD 和 NEW 提供。 OLD 和 NEW 表的结构与创建触发器的表的结构匹配。 OLD 表包含由触发语句修改或删除的任何行，处于其在触发语句操作之前的状态。 NEW 表包含由触发语句修改或创建的任何行，处于其在触发语句操作之后的状态。 WHEN 子句和 trigger-step 语句都可访问使用 NEW.column-name 和 OLD.column-name 形式的引用插入、删除或更新的行的值，其中 column-name 是与触发器关联的表中的列的名称。 OLD 和 NEW 表引用的可用性取决于触发器所处理的 database-event 类型：

- INSERT – NEW 引用有效
- UPDATE – NEW 和 OLD 引用有效
- DELETE – OLD 引用有效

指定的时间（BEFORE、AFTER 或 INSTEAD OF）确定何时执行与插入、修改或删除关联行有关的 trigger-step 语句。可将 ON CONFLICT 子句指定为 trigger-step 中 UPDATE 或 INSERT 语句的一部分。但是，如果将 ON CONFLICT 子句指定为导致触发器激发的语句的一部分，则改用该冲突处理策略。

除了表触发器外，还可在视图上创建 INSTEAD OF 触发器。如果在某个视图上定义了一个或多个 INSTEAD OF INSERT、INSTEAD OF DELETE 或 INSTEAD OF UPDATE 触发器，则在该视图上执行关联类型的语句（INSERT、DELETE 或 UPDATE）不会被视为错误。在这种情况下，在视图上执行 INSERT、DELETE 或 UPDATE 会导致关联触发器激发。由于触发器是 INSTEAD OF 触发器，因此导致触发器激发的语句不会修改视图的基础表。但是，触发器可用于对基础表执行修改操作。

在具有 INTEGER PRIMARY KEY 列的表上创建触发器时，请牢记一个重要问题。如果 BEFORE 触发器修改要由导致触发器激发的语句更新的行的 INTEGER PRIMARY KEY 列，则不会发生更新。解决方法是创建具有 PRIMARY KEY 列而不是 INTEGER PRIMARY KEY 列的表。

可使用 DROP TRIGGER 语句删除触发器。删除表或视图时，也会自动删除与该表或视图关联的所有触发器。

RAISE() 函数

特殊的 SQL 函数 RAISE() 可用于触发器的 trigger-step 语句中。此函数的语法如下：

```
raise-function  ::=  RAISE ( ABORT, error-message ) |
                     RAISE ( FAIL, error-message ) |
                     RAISE ( ROLLBACK, error-message ) |
                     RAISE ( IGNORE )
```

在触发器执行期间调用前三种形式之一时，会执行指定的 ON CONFLICT 处理操作 （ABORT、 FAIL 或 ROLLBACK），而且当前语句的执行结束。 ROLLBACK 被认为是语句执行失败，因此执行其 execute() 方法的 SQLStatement 实例将调度 error (SQLErrorEvent.ERROR) 事件。被调度的事件对象的 error 属性中的 SQLError 对象将其 details 属性设置为在 RAISE() 函数中指定的 error-message。

调用 RAISE(IGNORE) 时，将放弃当前触发器的剩余部分、导致触发器执行的语句和执行的任何后续触发器。不回滚数据库更改。如果导致触发器执行的语句本身是触发器的一部分，则在下一步开始时该触发器程序将继续执行。有关冲突解决算法的详细信息，请参阅 ON CONFLICT （冲突算法）部分。

DROP TRIGGER

DROP TRIGGER 语句删除由 CREATE TRIGGER 语句创建的触发器。

```
sql-statement  ::=  DROP TRIGGER [IF EXISTS] [database-name.] trigger-name
```

将从数据库中删除触发器。请注意，在删除其关联表时，会自动删除触发器。

特殊的语句和子句

本部分介绍几个对运行时提供的 SQL 进行扩展的子句，以及可在许多语句、注释和表达式中使用的两个语言元素。

COLLATE

COLLATE 子句在 SELECT、CREATE TABLE 和 CREATE INDEX 语句中使用，指定对值进行比较或排序时使用的比较算法。

```
sql-statement   ::=  COLLATE collation-name
collation-name  ::=  BINARY | NOCASE
```

列的默认排序规则类型是 BINARY。将 BINARY 排序规则用于 TEXT 存储类的值时，通过比较内存中表示值的字节来执行二进制排序规则，而不考虑文本编码。

NOCASE 排序规则序列仅应用于 TEXT 存储类的值。在使用时， NOCASE 排序规则执行不区分大小写的比较。

排序规则序列不用于 NULL、 BLOB、 INTEGER 或 REAL 类型的存储类。

若要将除 BINARY 之外的排序规则类型用于列，必须将 COLLATE 子句指定为 CREATE TABLE 语句中列定义的一部分。每当对两个 TEXT 值进行比较时，都将按照以下规则使用排序规则序列来确定比较的结果：

- 对于二进制比较运算符 （=、 <、 >、 <= 和 >=），如果任一操作数为列，则列的默认排序规则类型确定用于比较的排序规则序列。如果两个操作数都是列，则左操作数的排序规则类型确定所用的排序规则序列。如果操作数都不是列，则使用 BINARY 排序规则序列。
- BETWEEN...AND 运算符等效于使用两个包含 >= 和 <= 运算符的表达式。例如，表达式 x BETWEEN y AND z 等效于 x >= y AND x <= z。因此， BETWEEN...AND 运算符按照上述规则来确定排序规则序列。
- IN 运算符的行为与 = 运算符类似，用于确定要使用的排序规则序列。例如，如果 x 是列，则用于表达式 x IN (y, z) 的排序规则序列是 x 的默认排序规则类型。否则，使用 BINARY 排序规则。
- 可为属于 SELECT 语句的 ORDER BY 子句明确分配要用于排序操作的排序规则序列。在这种情况下，始终使用显式排序规则序列。否则，如果由 ORDER BY 子句排序的表达式是一个列，则使用列的默认排序规则类型确定排序顺序。如果表达式不是一个列，则使用 BINARY 排序规则序列。

EXPLAIN

EXPLAIN 命令修饰符是 SQL 的非标准扩展。

```
sql-statement  ::=  EXPLAIN sql-statement
```

如果 EXPLAIN 关键字出现在任何其他 SQL 语句之前，则结果报告它用于执行命令的虚拟机指令序列，而不是实际执行命令，就像不存在 EXPLAIN 关键字一样。 EXPLAIN 功能是一种高级功能，允许开发人员更改 SQL 语句文本以尝试优化性能或调试看起来工作不正常的语句。

ON CONFLICT （冲突算法）

ON CONFLICT 子句不是单独的 SQL 命令。它是可出现在许多其他 SQL 命令中的非标准子句。

```
conflict-clause     ::=  ON CONFLICT conflict-algorithm
conflict-clause     ::=  OR conflict-algorithm
conflict-algorithm  ::=  ROLLBACK |
                         ABORT |
                         FAIL |
                         IGNORE |
                         REPLACE
```

ON CONFLICT 子句的第一种形式 （使用关键字 ON CONFLICT）用于 CREATE TABLE 语句中。对于 INSERT 或 UPDATE 语句，使用第二种形式（将 ON CONFLICT 替换为 OR）以使语法看起来更自然。例如，语句变为 INSERT OR IGNORE，而不再是 INSERT ON CONFLICT IGNORE。虽然关键字是不同的，但是子句的含义在任一形式中都是相同的。

ON CONFLICT 子句指定用于解决约束冲突的算法。五种算法为 ROLLBACK、 ABORT、 FAIL、 IGNORE 和 REPLACE。默认算法为 ABORT。以下是对这五种冲突算法的说明：

ROLLBACK 出现约束冲突时，会立即发生 ROLLBACK，结束当前的事务。命令将中止， SQLStatement 实例调度 error 事件。如果没有事务处于活动状态 （在每个命令上创建的隐含事务除外），则此算法的作用与 ABORT 相同。

ABORT 出现约束冲突时，命令将撤消它之前可能已进行的任何更改， SQLStatement 实例会调度 error 事件。不执行 ROLLBACK，因此将保留以前命令在事务中进行的更改。 ABORT 是默认行为。

FAIL 出现约束冲突时，命令将中止， SQLStatement 会调度 error 事件。但是，将保留而不撤消在遇到约束违反之前语句对数据库进行的任何更改。例如，如果 UPDATE 语句在它尝试更新的第 100 行上遇到约束冲突，则保留对前 99 行的更改，但不会更改第 100 行和之后的行。

IGNORE 出现约束冲突时，不插入或更改包含约束冲突的一行。除了忽略此行外，命令通常会继续正常执行。通常会继续正常地插入或更新包含约束违反的行之前和之后的其他行。不返回错误。

REPLACE 出现 UNIQUE 约束冲突时，在插入或更新当前行之前，删除导致约束冲突的预先存在的行。因此，插入或更新会始终进行，而且命令通常会继续正常执行。不返回错误。如果出现 NOT NULL 约束冲突，则将 NULL 值替换为该列的默认值。如果列没有默认值，则使用 ABORT 算法。如果出现 CHECK 约束冲突，则使用 IGNORE 算法。此冲突解决策略删除行以便满足约束时，它不会调用这些行上的删除触发器。

在 INSERT 或 UPDATE 语句的 OR 子句中指定的算法将覆盖在 CREATE TABLE 语句中指定的任何算法。如果未在 CREATE TABLE 语句或者执行 INSERT 或 UPDATE 语句中指定算法，则将使用 ABORT 算法。

REINDEX

REINDEX 命令用于删除并重新创建一个或多个索引。在排序规则序列的定义更改时，此命令很有用。

```
sql-statement  ::=  REINDEX collation-name
sql-statement  ::=  REINDEX [database-name .] ( table-name | index-name )
```

在第一种形式中，将重新创建使用指定排序规则序列的所有附加数据库中的所有索引。在第二种形式中，指定 table-name 时，将重新生成与表关联的所有索引。如果提供了 index-name，则仅删除并重新创建指定的索引。

COMMENTS

注释不是 SQL 命令，但它们可出现在 SQL 查询中。运行时将它们视为空白。它们可从能找到空白的任何位置开始，包括跨多行的表达式的内部。

```
comment             ::=  single-line-comment |
                         block-comment
single-line-comment ::=  -- single-line
block-comment       ::=  /* multiple-lines or block [*/]
```

单行注释由两个短划线指示。单行注释仅扩展到当前行的结尾。

块注释可跨任何数目的行，或者嵌入到单行中。如果没有终止分隔符，则块注释可扩展到输入的结尾。此情况并不被视为错误。新的 SQL 语句可在块注释结束后的行上开始。块注释可嵌入到能出现空白的任何位置中，包括表达式内部和其他 SQL 语句的中间。块注释不嵌套。忽略块注释内的单行注释。

EXPRESSIONS

表达式是其他 SQL 块中的子命令。以下介绍 SQL 语句中表达式的有效语法：

```
expr            ::=  expr binary-op expr |
                     expr [NOT] like-op expr [ESCAPE expr] |
                     unary-op expr |
                     ( expr ) |
                     column-name |
                     table-name.column-name |
                     database-name.table-name.column-name |
                     literal-value |
                     parameter |
                     function-name( expr-list | * ) |
                     expr ISNULL |
                     expr NOTNULL |
                     expr [NOT] BETWEEN expr AND expr |
                     expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name |
                     [EXISTS] ( select-statement ) |
                     CASE [expr] ( WHEN expr THEN expr )+ [ELSE expr] END |
                     CAST ( expr AS type ) |
                     expr COLLATE collation-name
like-op         ::=  LIKE | GLOB
binary-op       ::=  see Operators
unary-op        ::=  see Operators
parameter       ::=  :param-name | @param-name | ?
value-list      ::=  literal-value [, literal-value]*
literal-value   ::=  literal-string | literal-number | literal-boolean | literal-blob | literal-null
literal-string  ::=  'string value'
literal-number  ::=  integer | number
literal-boolean  ::=  true | false
literal-blob  ::=  X'string of hexadecimal data'
literal-null  ::=  NULL
```

表达式是值和可解析为单个值的运算符的任何组合。根据表达式解析为布尔值 （true 或 false）还是解析为非布尔值，可将表达式分为两种常规类型。

在几种常见情况下，包括在 WHERE 子句、HAVING 子句、JOIN 子句中的 ON 表达式以及 CHECK 表达式中，表达式必须解析为布尔值。以下类型的表达式满足此条件：

- ISNULL
- NOTNULL
- IN ()
- EXISTS ()

- LIKE
- GLOB
- 某些函数
- 某些运算符 （特别是比较运算符）

字面值

文本数字值是以整数或浮点数形式书写的。支持科学记数法。 .（句点）字符始终用作小数点。

字符串文本是通过用单引号 ' 将字符串括起来指示的。若要在字符串中加入单引号，请在一行中连续放置两个单引号，例如 ''。

布尔文本由值 true 或 false 指示。文本布尔值用于 Boolean 列数据类型。

BLOB 文本是包含十六进制数据且前面有单个 x 或 X 字符的字符串文本，例如 X'53514697465'。

字面值也可以是标记 NULL。

列名

列名可以是在 CREATE TABLE 语句中定义的任何名称或以下特殊标识符之一：ROWID、 OID 或 _ROWID_。这些特殊标识符都描述与每个表的每一行关联的唯一随机整数键 （“ 行键 ”）。如果 CREATE TABLE 语句未定义同名的真正列，则特殊标识符仅引用行键。行键充当只读列。行键可在能使用常规列的任何位置使用，但您不能在 UPDATE 或 INSERT 语句中更改行键的值。 SELECT * FROM table 语句在其结果集中不包括行键。

SELECT 语句

SELECT 语句可在表达式中作为 IN 运算符的右操作数，作为纯量 （单个结果值）或者作为 EXISTS 运算符的操作数。用作纯量或 IN 运算符的操作数时， SELECT 在其结果中只能具有单个列。允许使用复合 SELECT 语句 （通过诸如 UNION 或 EXCEPT 之类的关键字连接）。在使用 EXISTS 运算符时，忽略 SELECT 的结果集中的列。如果存在一个或多个行，则表达式返回 TRUE ；如果结果集为空，则返回 FALSE。如果 SELECT 表达式中没有项引用包含查询中的值，则该表达式将在任何其他处理之前计算一次，并在必要时重用结果。如果 SELECT 表达式不包含外部查询 （称为相关子查询）中的变量，则 SELECT 在每次需要时都重新进行计算。

当 SELECT 是 IN 运算符的右操作数时，如果左操作数的结果等于 SELECT 语句结果集中的任一值，则 IN 运算符返回 TRUE。可在 IN 运算符前面加上 NOT 关键字以反转测试的意义。

如果 SELECT 出现在表达式内但不是 IN 运算符的右操作数，则 SELECT 结果的第一行将成为表达式中使用的值。如果 SELECT 生成多个结果行，则忽略第一行后的所有行。如果 SELECT 未生成行，则 SELECT 的值为 NULL。

CAST 表达式

CAST 表达式将指定的值的数据类型更改为提供的类型。指定的类型可以是对 CREATE TABLE 语句的列定义中的类型有效的任何非空类型名称。有关详细信息，请参阅数据类型支持。

其他表达式元素

以下部分介绍可在表达式中使用的其他 SQL 元素：

- 内置函数：聚合函数，标量函数，日期和时间格式函数
- 运算符
- 参数

内置函数

内置函数分为以下三种主要类别：

- 聚合函数

- 标量函数
- 日期和时间格式函数

除了这些函数外，还有一个用于提供执行触发器时的错误通知的特殊函数 RAISE()。此函数只能在 CREATE TRIGGER 语句体使用。有关 RAISE() 函数的信息，请参阅 CREATE TRIGGER > RAISE()。

与 SQL 中的所有关键字一样，函数名称不区分大小写。

聚合函数

聚合函数对多行中的值执行操作。这些函数主要在 SELECT 语句中与 GROUP BY 子句一起使用。

AVG(X)	返回一个组中所有非 NULL X 的平均值。看起来不像数字的字符串和 BLOB 值被解释为 0。AVG() 的结果始终是浮点值，即使所有输入都是整数。
COUNT(X) COUNT(*)	第一种形式返回组中不是 NULL 的 X 次数的计数。第二种形式 （使用 * 参数）返回组中的总行数。
MAX(X)	返回组中所有值的最大值。通常的排序顺序用于确定最大值。
MIN(X)	返回组中所有值的最小非 NULL 值。通常的排序顺序用于确定最小值。如果组中的所有值都是 NULL，则返回 NULL。
SUM(X)	返回组中所有非 NULL 值的数字和。如果所有值都是 NULL，则 SUM() 返回 NULL， TOTAL() 返回 0.0。 TOTAL() 的结果始终是浮点值。如果所有非 NULL 输入都是整数，则 SUM() 的结果是整数值。如果 SUM() 的任一输入不是整数且不是 NULL，则 SUM() 返回浮点值。此值可能是真实和的近似值。
TOTAL(X)	

在采用单个参数的上述任一聚合函数中，可在该参数前面加上关键字 DISTINCT。在这种情况下，在将重复的元素传递到聚合函数之前，会对其进行筛选。例如，函数调用 COUNT(DISTINCT x) 返回列 X 的非重复值数目，而不是列 x 中非 NULL 值的总数。

标量函数

标量函数一次对一行上的值进行运算。

ABS(X)	返回参数 X 的绝对值。
COALESCE(X, Y, ...)	返回第一个非 NULL 参数的副本。如果所有参数都为 NULL，则返回 NULL。必须至少有两个参数。
GLOB(X, Y)	此函数用于实现 X GLOB Y 语法。
IFNULL(X, Y)	返回第一个非 NULL 参数的副本。如果两个参数都为 NULL，则返回 NULL。此函数的行为与 COALESCE() 相同。
HEX(X)	该参数被解释为 BLOB 存储类型的值。结果是该值内容的十六进制呈现。
LAST_INSERT_ROW ID()	返回通过当前的 SQLConnection 插入到数据库的最后一行的行标识符 （生成的主键）。此值与 SQLConnection.lastInsertRowID 属性返回的值相同。
LENGTH(X)	返回 X 的字符串长度 （以字符计）。
LIKE(X, Y [, Z])	此函数用于实现 SQL 的 X LIKE Y [ESCAPE Z] 语法。如果存在可选的 ESCAPE 子句，则使用三个参数调用该函数。否则，仅使用两个参数调用它。
LOWER(X)	返回字符串 X 的副本，其中所有字符都已转换为小写形式。
LTRIM(X) LTRIM(X, Y)	返回通过删除 X 左侧的空格而形成的字符串。如果指定了 Y 参数，则函数将从 X 的左侧删除 Y 中的任何字符。
MAX(X, Y, ...)	返回具有最大值的参数。除了数字外，参数还可能是字符串。最大值由定义的排序顺序确定。请注意，当 MAX() 具有 2 个或更多参数时，它是简单函数，而它只有一个参数时则是聚合函数。
MIN(X, Y, ...)	返回具有最小值的参数。除了数字外，参数还可能是字符串。最小值由定义的排序顺序确定。请注意， MIN() 在具有 2 个或更多参数时是简单函数，在只有一个参数时则是聚合函数。
NULLIF(X, Y)	如果参数是不同的，则返回第一个参数，否则返回 NULL。
QUOTE(X)	此例程返回一个字符串，该字符串是其适合于包含到另一 SQL 语句中的参数的值。字符串括在单引号中，并根据需要对内部引号转义。 BLOB 存储类作为十六进制文本进行编码。在编写触发器以实现撤消 / 重做功能时，此函数是很有用的。
RANDOM(*)	返回一个介于 -9223372036854775808 和 9223372036854775807 之间的伪随机整数。此随机值不是强加密的。
RANDOMBLOB(N)	返回一个包含伪随机字节的 N 字节 BLOB。 N 应该是一个正整数。此随机值不是强加密的。如果 N 的值为负，则返回单个字节。

ROUND(X) ROUND(X, Y)	将数值 X 舍入到小数点右侧的 Y 位。如果省略 Y 参数，则使用 0。
RTRIM(X) RTRIM(X, Y)	返回通过删除 X 右侧的空格而形成的字符串。如果指定了 Y 参数，则函数将从 X 的右侧删除 Y 中的任何字符。
SUBSTR(X, Y, Z)	返回输入字符串 X 中以第 Y 个字符开头且长度为 Z 个字符的子字符串。 X 的最左侧字符是索引位置 1。如果 Y 为负，则通过从右 （而不是从左）计数来查找子字符串的第一个字符。
TRIM(X) TRIM(X, Y)	返回通过删除 X 右侧的空格而形成的字符串。如果指定了 Y 参数，则函数将从 X 的右侧删除 Y 中的任何字符。
TYPEOF(X)	返回表达式 X 的类型。可能的返回值有 “null”、 “integer”、 “real”、 “text” 和 “blob”。有关数据类型的详细信息，请参阅数据类型支持。
UPPER(X)	返回已转换为全大写字母的输入字符串 X 的副本。
ZEROBLOB(N)	返回包含 0x00 的 N 字节的 BLOB。

日期和时间格式函数

日期和时间格式函数是一组用于创建带格式的日期和时间数据的标量函数。请注意，这些函数对字符串值和数字值进行运算，并返回字符串值和数字值。这些函数并不预定用于 DATE 数据类型。如果对其声明数据类型为 DATE 的列中的数据使用这些函数，则它们不会像预期的那样工作。

DATE(T, ...)	DATE() 函数返回一个包含日期的字符串，日期格式为 YYYY-MM-DD。第一个参数 (T) 指定在时间格式下找到的格式的时间字符串。可在时间字符串后指定任何数量的修饰符。可在修饰符下找到这些修饰符。
TIME(T, ...)	TIME() 函数返回一个包含时间的字符串，时间格式为 HH:MM:SS。第一个参数 (T) 指定在时间格式下找到的格式的时间字符串。可在时间字符串后指定任何数量的修饰符。可在修饰符下找到这些修饰符。
DATETIME(T, ...)	DATETIME() 函数返回一个包含日期和时间的字符串，日期和时间格式为 YYYY-MM-DD HH:MM:SS。第一个参数 (T) 指定在时间格式下找到的格式的时间字符串。可在时间字符串后指定任何数量的修饰符。可在修饰符下找到这些修饰符。
JULIANDAY(T, ...)	JULIANDAY() 函数返回一个数字，指示自格林尼治标准时间公元前 4714 年 11 月 24 日中午至所提供日期的天数。第一个参数 (T) 指定在时间格式下找到的格式的时间字符串。可在时间字符串后指定任何数量的修饰符。可在修饰符下找到这些修饰符。
STRFTIME(F, T, ...)	STRFTIME() 例程返回根据指定为第一个参数 F 的格式字符串设置格式的日期。格式字符串支持以下替换： %d - 一月中的某天 %f - 带小数的秒 SS.SSS %H - 小时 00-24 %j - 一年中的某天 001-366 %J - 儒略历日数 %m - 月份 01-12 %M - 分钟 00-59 %s - 自 1970-01-01 以来的秒数 %S - 秒 00-59 %w - 一周中的某天 0-6 （星期日 = 0） %W - 一年中的某周 00-53 %Y - 年份 0000-9999 %% - % 第二个参数 (T) 指定在时间格式下找到的格式的时间字符串。可在时间字符串后指定任何数量的修饰符。可在修饰符下找到这些修饰符。

时间格式

时间字符串可采用以下任一格式：

YYYY-MM-DD	2007-06-15
YYYY-MM-DD HH:MM	2007-06-15 07:30
YYYY-MM-DD HH:MM:SS	2007-06-15 07:30:59
YYYY-MM-DD HH:MM:SS.SSS	2007-06-15 07:30:59.152
YYYY-MM-DDTHH:MM	2007-06-15T07:30
YYYY-MM-DDTHH:MM:SS	2007-06-15T07:30:59
YYYY-MM-DDTHH:MM:SS.SSS	2007-06-15T07:30:59.152
HH:MM	07:30 （日期是 2000-01-01）
HH:MM:SS	7:30:59 （日期是 2000-01-01）
HH:MM:SS.SSS	07:30:59:152 （日期是 2000-01-01）
目前	当前的日期和时间 （按通用协调时间）。
DDDD.DDDD	儒略历日数是一个浮点数。

这些格式中的字符 T 是分隔日期和时间的文本字符 “T”。仅包括时间的格式假定日期是 2001-01-01。

修饰符

时间字符串可后跟用于更改日期或更改日期解释的零个或更多个修饰符。可用的修饰符如下：

NNN 天	要添加到时间的天数。
NNN 小时	要添加到时间的小时数。
NNN 分钟	要添加到时间的分钟数。
NNN.NNNN 秒	要添加到时间的秒数和毫秒数。
NNN 月	要添加到时间的月数。
NNN 年	要添加到时间的年数。
一月的开始	将时间后移到一月的开始。
一年的开始	将时间后移到一年的开始。
一天的开始	将时间后移到一天的开始。
工作日 N	将时间前移到指定的工作日。（0 = 星期日， 1 = 星期一，依此类推）。
本地时间	将日期转换为本地时间。
utc	将日期转换为通用协调时间。

运算符

SQL 支持大量的运算符 （其中包括大多数编程语言中存在的常见运算符）以及 SQL 独有的几个运算符。

常见运算符

在 SQL 块中允许以下二元运算符，它们按从最高到最低的优先顺序列出：

```
*    /    %
+    -
<< >> & |
< >=   > >=
=    ==   !=   <> IN
AND
OR
```

支持的一元前缀运算符是：

```
 !    ~    NOT
```

可以认为 COLLATE 运算符是一元后缀运算符。 COLLATE 运算符具有最高的优先顺序。它始终比任何一元前缀运算符或任何二元运算符绑定得更紧密。

请注意，等号和不等号运算符各有两个变体。等号可以是 = 或 ==。不等号运算符可以是 != 或 <>。

|| 运算符是字符串串联运算符 — 它将其操作数的两个字符串联接在一起。

运算符 % 输出其左操作数以其右操作数为模的余数。

任何二元运算符的结果都是一个数字值，但给出字符串结果的 || 串联运算符除外。

SQL 运算符

LIKE

LIKE 运算符进行模式匹配比较。

```
expr     ::=  (column-name | expr) LIKE pattern
pattern  ::=  '[ string | % | _ ]'
```

LIKE 运算符右侧的操作数包含模式，而左侧的操作数包含对模式进行匹配的字符串。模式中的百分比符号 (%) 是一个通配符 — 它与字符串中零个或更多个字符的任何序列匹配。模式中的下划线 (_) 与字符串中的任何单个字符匹配。任何其他字符与自身匹配，或与其小写 / 大写形式匹配 （即匹配是以不区分大小写的方式执行的）。（注意：数据库引擎仅识别 7 位拉丁字符的大写 / 小写形式。因此，对于 8 位 iso8859 字符或 UTF-8 字符， LIKE 运算符区分大小写。例如，表达式 'a' LIKE 'A' 为 TRUE，但是 'æ' LIKE 'Æ' 为 FALSE）。可使用 SQLConnection.caseSensitiveLike 属性更改拉丁字符的区分大小写。

如果存在可选的 ESCAPE 子句，则 ESCAPE 关键字后面的表达式的计算结果必须是一个包含单个字符的字符串。可在 LIKE 模式中使用此字符，与文本百分比或下划线字符进行匹配。转义符后跟百分比符号、下划线或转义符本身分别与字符串中的文本百分比符号、下划线或转义符匹配。

GLOB

GLOB 运算符与 LIKE 类似，但是对其通配符使用 Unix 文件名替换语法。与 LIKE 不同， GLOB 区分大小写。

IN

IN 运算符计算其左操作数是否等于其右操作数 （括号中的一组值）中的值之一。

```
in-expr         ::=  expr [NOT] IN ( value-list ) |
                     expr [NOT] IN ( select-statement ) |
                     expr [NOT] IN [database-name.] table-name
value-list      ::=  literal-value [, literal-value]*
```

右操作数可以是一组逗号分隔的字面值，也可以是 SELECT 语句的结果。有关将 SELECT 语句用作 IN 操作符的右侧操作数的说明和限制，请参阅表达式中的 SELECT 语句。

BETWEEN...AND

BETWEEN...AND 运算符等效于使用两个包含 >= 和 <= 运算符的表达式。例如，表达式 x BETWEEN y AND z 等效于 x >= y AND x <= z。

NOT

NOT 运算符是一个求反运算符。可在 GLOB、 LIKE 和 IN 运算符前面加上 NOT 关键字来反转测试的意义 （换句话说，检查一个值是否与指示的模式不匹配）。

参数

参数在表达式中指定占位符，用于在运行时通过将值分配给 SQLStatement.parameters 关联数组来填充的字面值。参数可采用以下三种形式：

?	问号指示一个索引参数。根据参数在语句中的顺序，为其分配数字 （从零开始的）索引值。
:AAAA	冒号后跟标识符名称保存名为 AAAA 的命名参数的位置。命名参数也是根据它们在 SQL 语句中的顺序编号的。若要避免混淆，最好避免混合使用命名参数和编号参数。
@AAAA	“at 符号 ” 等效于冒号。

不支持的 SQL 功能

以下是在 Adobe AIR 中不支持的标准 SQL 元素的列表：

FOREIGN KEY 约束 分析但不强制执行 FOREIGN KEY 约束。

触发器 不支持 FOR EACH STATEMENT 触发器（所有触发器都必须为 FOR EACH ROW）。在表上不支持 INSTEAD OF 触发器 （仅在视图上允许 INSTEAD OF 触发器）。不支持递归触发器 （触发自身的触发器）。

ALTER TABLE 仅支持 ALTER TABLE 命令的 RENAME TABLE 和 ADD COLUMN 变体。其他种类的 ALTER TABLE 操作 （如 DROP COLUMN、 ALTER COLUMN、 ADD CONSTRAINT 等）将被忽略。

嵌套事务 仅允许单个活动事务。

RIGHT 和 FULL OUTER JOIN 不支持 RIGHT OUTER JOIN 或 FULL OUTER JOIN。

可更新的 VIEW 视图是只读的。不能对视图执行 DELETE、INSERT 或 UPDATE 语句。支持在尝试对视图执行 DELETE、INSERT 或 UPDATE 时激发的 INSTEAD OF 触发器，可使用它更新触发器体中的支持表。

GRANT 和 REVOKE 数据库是一个普通的磁盘文件；可应用的访问权限只有基础操作系统的常规文件访问权限。未实现通常在客户端 / 服务器 RDBMS 上找到的 GRANT 和 REVOKE 命令。

在一些 SQLite 实现中支持以下 SQL 元素和 SQLite 功能，但是在 Adobe AIR 中不支持它们。此功能的大部分可通过 SQLConnection 类的方法获取：

与事务相关的 SQL 元素 （BEGIN、 END、 COMMIT、 ROLLBACK） 通过 SQLConnection 类的与事务相关的方法 SQLConnection.begin()、 SQLConnection.commit() 和 SQLConnection.rollback() 可使用此功能。

ANALYZE 此功能可通过 SQLConnection.analyze() 方法获取。

ATTACH 此功能可通过 SQLConnection.attach() 方法获取。

COPY 不支持此语句。

CREATE VIRTUAL TABLE 不支持此语句。

DETACH 此功能可通过 SQLConnection.detach() 方法获取。

PRAGMA 不支持此语句。

VACUUM 此功能可通过 SQLConnection.compact() 方法获取。

无法访问系统表 系统表 （包括 sqlite_master 及其他具有 “sqlite_” 前缀的表）在 SQL 语句中不可用。运行时包括一个架构 API，用于提供面向对象的方法来访问架构数据。有关详细信息，请参阅 SQLConnection.loadSchema() 方法。

正则表达式函数 （MATCH() 和 REGEX()） 这些函数在 SQL 语句中不可用。

以下功能在许多 SQLite 实现和 Adobe AIR 之间是不同的：

索引语句参数 在许多实现中，索引语句参数从 1 开始。但是，在 Adobe AIR 中，索引语句参数是从零开始的 （即，为第一个参数指定索引 0，为第二个参数指定索引 1，依此类推）。

INTEGER PRIMARY KEY 列定义 在许多实现中，只有确切定义为 INTEGER PRIMARY KEY 的列才用作表的实际主键列。在这些实现中，使用通常为 INTEGER 的同义词 （例如 int）的其他数据类型不会导致将列用作内部主键。但是，在 Adobe AIR 中，int 数据类型（和其他 INTEGER 同义词）将视为完全等同于 INTEGER。因此，定义为 int PRIMARY KEY 的列用作表的内部主键。有关更多信息，请参阅 CREATE TABLE 和列关联部分。

其他 SQL 功能

默认情况下 SQLite 不支持以下列关联类型，但 Adobe AIR 却支持 （请注意，与 SQL 中的所有关键字一样，这些数据类型名称不区分大小写）：

Boolean 对应于 Boolean 类。

Date 对应于 Date 类。

int 对应于 int 类 （等效于 INTEGER 列关联）。

Number 对应于 Number 类 （等效于 REAL 列关联）。

Object 对应于 Object 类或可使用 AMF3 序列化和反序列化的任何子类。（这包括大多数类 （其中包括自定义类），但是不包括某些类 （其中包括显示对象以及将显示对象作为属性包括的对象）。）

String 对应于 String 类 （等效于 TEXT 列关联）。

XML 对应于 ActionScript (E4X) XML 类。

XMLList 对应于 ActionScript (E4X) XMLList 类。

在 SQLite 中默认情况下不支持以下字面值，但是在 Adobe AIR 中支持它们：

true 用于表示文本布尔值 true，以处理 BOOLEAN 列。

false 用于表示文本布尔值 false，以处理 BOOLEAN 列。

数据类型支持

与大多数 SQL 数据库不同， Adobe AIR SQL 数据库引擎不要求或强制表列包含某种类型的值。相反，运行时使用两个概念（存储类和列关联）来控制数据类型。本部分介绍存储类和列关联，以及如何在各种情况下解决数据类型差异：

- 第 963 页的 " 存储类 "
- 第 964 页的 " 列关联 "
- 第 966 页的 " 数据类型与比较运算符 "
- 第 966 页的 " 数据类型与数学运算符 "
- 第 966 页的 " 数据类型与排序 "
- 第 966 页的 " 数据类型与分组 "
- 第 966 页的 " 数据类型与复合 SELECT 语句 "

存储类

存储类表示用于在数据库中存储值的实际数据类型。下列存储类由数据库使用：

NULL 值为 NULL 值。

INTEGER 值为带符号的整数。

REAL 值为浮点数值。

TEXT 值为文本字符串 （限制为 256 MB）。

BLOB 值为二进制大型对象 (BLOB)；换句话说，为原始二进制数据 （限制为 256 MB）。

对于作为嵌入在 SQL 语句中的文本或使用参数绑定到准备的 SQL 语句的值提供给数据库的所有值，都在执行 SQL 语句之前为其分配存储类。

如果属于 SQL 语句的文本括在单引号或双引号中，则为其分配存储类 TEXT；如果文本被指定为没有小数点或指数的不带引号数字，则分配 INTEGER；如果文本是带有小数点或指数的不带引号数字，则分配 REAL；如果值为 NULL，则分配 NULL。具有存储类 BLOB 的文本是使用 X'ABCD' 表示法指定的。有关详细信息，请参阅表达式中的字面值。

对于作为使用 SQLStatement.parameters 关联数组的参数提供的值，为其分配与绑定的本机数据类型最紧密匹配的存储类。例如，int 值作为 INTEGER 存储类绑定，为 Number 值分配 REAL 存储类，为 String 值分配 TEXT 存储类，为 ByteArray 对象分配 BLOB 存储类。

列关联

列的关联是存储在该列中的数据的建议类型。一个值存储在列中 （通过 INSERT 或 UPDATE 语句）时，运行时尝试将该值从其数据类型转换为指定的关联。例如，如果将 Date 值 （ActionScript 或 JavaScript Date 实例）插入到一个其关联为 TEXT 的列中，则 Date 值在存储于数据库中之前将转换为 String 表示形式（等效于调用对象的 toString() 方法）。如果该值无法转换为指定的关联，则出现错误，且不执行操作。使用 SELECT 语句从数据库中检索值时，它作为对应于关联的类的实例返回，而不管它在被存储时是否已从不同数据类型转换。

如果一个列接受 NULL 值，则 ActionScript 或 JavaScript 值 null 可用作参数值在该列中存储 NULL。当 NULL 存储类值在 SELECT 语句中检索时，它始终作为 ActionScript 或 JavaScript 值 null 返回，而不管列的关联如何。如果一个列接受 NULL 值，则在尝试将值转换成不可为 Null 的类型（如 Number 或 Boolean）之前，始终检查从该列检索的值以确定它们是否为 null。

为数据库中的每个列分配以下类型关联之一：

- TEXT （或 String）
- NUMERIC
- INTEGER （或 int）
- REAL （或 Number）
- Boolean
- Date
- XML
- XMLLIST
- Object
- NONE

TEXT （或 **String**）

具有 TEXT 或 String 关联的列使用存储类 NULL、 TEXT 或 BLOB 存储所有数据。如果将数字值插入到具有 TEXT 关联的列中，则在存储它之前将它转换为文本形式。

NUMERIC

具有 NUMERIC 关联的列包含使用存储类 NULL、 REAL 或 INTEGER 的值。将文本数据插入到 NUMERIC 列中时，在存储它之前，会尝试将它转换为整数或实数。如果转换成功，则使用 INTEGER 或 REAL 存储类存储值（例如，值 '10.05' 在被存储之前转换为 REAL 存储类）。如果无法执行转换，则出现错误。不会尝试转换 NULL 值。从 NUMERIC 列检索的值作为该值适合的最具体数字类型的实例返回。换句话说，如果该值是一个正整数或 0，则它作为 uint 实例返回。如果它是一个负整数，则它作为 int 实例返回。最后，如果它具有浮点部分 （它不是一个整数），则它作为 Number 实例返回。

INTEGER （或 **int**）

使用 INTEGER 关联的列的行为方式与具有 NUMERIC 关联的列相同，但有一处不同。如果要存储的值是一个没有浮点部分的实数值 （如 Number 实例），或者该值是可转换为没有浮点部分的实数值的文本值，则将它转换为整数并使用 INTEGER 存储类存储它。如果尝试存储具有浮点部分的实数值，则出现错误。

REAL （或 **Number**）

具有 REAL 或 NUMBER 关联的列的行为与具有 NUMERIC 关联的列类似，但是它将整数值强制为浮点表示形式。 REAL 列中的值始终作为 Number 实例从数据库返回。

Boolean

具有 Boolean 关联的列存储 true 或 false 值。 Boolean 列接受作为 ActionScript 或 JavaScript Boolean 实例的值。如果代码尝试存储字符串值，则将长度大于零的字符串视为 true，将空字符串视为 false。如果代码尝试存储数字数据，则任何非零值作为 true 存储，而 0 作为 false 存储。使用 SELECT 语句检索 Boolean 值时，它作为 Boolean 实例返回。非 NULL 值是使用 INTEGER 存储类存储的 （0 表示 false， 1 表示 true），并在检索数据时转换为 Boolean 对象。

Date

具有 Date 关联的列存储日期和时间值。Date 列用于接受作为 ActionScript 或 JavaScript Date 实例的值。如果尝试在 Date 列中存储 String 值，则运行时会尝试将该 String 值转换为罗马儒略历日期。如果转换失败，则出现错误。如果代码尝试存储 Number、int 或 uint 值，则不会尝试验证数据，而假定它是有效的罗马儒略历日期值。使用 SELECT 语句检索的 Date 值将自动转换为 Date 实例。使用 REAL 存储类将 Date 值存储为罗马儒略历日期值，因此排序和比较操作可以如预期的那样进行。

XML 或 XMLList

使用 XML 或 XMLList 关联的列存储 XML 结构。当代码尝试使用 SQLStatement 参数在 XML 列中存储数据时，运行时会尝试使用 ActionScript XML() 或 XMLList() 函数转换和验证值。如果该值无法转换为有效的 XML，则出现错误。如果存储数据的尝试使用 SQL 文本字面值（例如 INSERT INTO (col1) VALUES ('Invalid XML (no closing tag)')），则不分析或验证该值 — 而是假定它的格式正确。如果存储了无效值，则在检索它时，它作为空的 XML 对象返回。 XML 和 XMLList 数据是使用 TEXT 存储类或 NULL 存储类存储的。

Object

具有 Object 关联的列存储 ActionScript 或 JavaScript 复杂对象，其中包括 Object 类实例以及 Object 子类的实例 （如 Array 实例），甚至还包括自定义类实例。 Object 列数据以 AMF3 格式进行序列化，并使用 BLOB 存储类进行存储。在检索一个值时，它从 AMF3 进行反序列化，并在被存储时作为类的实例返回。请注意，某些 ActionScript 类 （尤其是显示对象）无法反序列化为其原始数据类型的实例。在存储自定义类实例之前，必须使用 flash.net.registerClassAlias() 方法 （或者在 Flex 中，通过向类声明添加 [RemoteObject] 元数据）为该类注册一个别名。此外，在检索该数据之前，必须为类注册相同的别名。无法正确进行反序列化的任何数据 （因为类本身无法进行反序列化，或者因为缺少类别名或类别名不匹配），在被存储时作为具有对应于原始实例的属性和值的匿名对象 （Object 类实例）返回。

NONE

对于具有关联 NONE 的列，使用任何存储类都是等效的。它不尝试在插入数据之前将其转换。

确定关联

列的类型关联由在 CREATE TABLE 语句中声明的列类型确定。在确定类型时，可应用以下规则 （不区分大小写）：

- 如果列的数据类型包含字符串 “CHAR”、“CLOB”、“STRI” 或 “TEXT” 中的任一个，则该列具有 TEXT/String 关联。请注意，类型 VARCHAR 包含字符串 “CHAR”，因此为其分配 TEXT 关联。
- 如果列的数据类型包含字符串 “BLOB”，或者未指定数据类型，则该列具有关联 NONE。
- 如果列的数据类型包含字符串 “XMLL”，则该列具有 XMLList 关联。
- 如果数据类型是字符串 “XML”，则列具有 XML 关联。
- 如果数据类型包含字符串 “OBJE”，则列具有 Object 关联。
- 如果数据类型包含字符串 “BOOL”，则列具有 Boolean 关联。
- 如果数据类型包含字符串 “DATE”，则列具有 Date 关联。
- 如果数据类型包含字符串 “INT” （包括 “UINT”），则为其分配 INTEGER/int 关联。
- 如果列的数据类型包含字符串 “REAL”、 “NUMB”、 “FLOA” 或 “DOUB” 中的任一个，则该列具有 REAL/Number 关联。
- 否则，关联为 NUMERIC。
- 如果表使用 CREATE TABLE t AS SELECT... 语句创建的，则所有列都未指定数据类型，将为它们分配关联 NONE。

数据类型与比较运算符

支持以下二元比较运算符 =、<、<=、>= 和 !=，以及测试集合成员身份的运算符 IN 和三元比较运算符 BETWEEN。有关这些运算符的详细信息，请参阅 " 运算符 "。

比较的结果取决于所比较的两个值的存储类。在比较两个值时，可应用以下规则：

- 将具有存储类 NULL 的值视为小于其他任何值 （包括具有存储类 NULL 的其他值）。
- INTEGER 或 REAL 值小于任何 TEXT 或 BLOB 值。对 INTEGER 或 REAL 与其他 INTEGER 或 REAL 进行比较时，将执行数字比较。
- TEXT 值小于 BLOB 值。对两个 TEXT 值进行比较时，将执行二元比较。
- 对两个 BLOB 值进行比较时，结果始终是使用二元比较确定的。

始终将三元运算符 BETWEEN 重新转换为等效的二元表达式。例如，a BETWEEN b AND c 将重新转换为 a >= b AND a <= c，即使这意味着在计算表达式所需的每个比较中将不同的关联应用于 a。

a IN (SELECT b ....) 类型的表达式由先前为二元比较列举的三个规则处理，即以与 a = b 类似的方式。例如，如果 b 是一个列值，a 是一个表达式，则在进行任何比较之前，将 b 的关联应用于 a。将表达式 a IN (x, y, z) 重新转换为 a = +x OR a = +y OR a = +z。IN 运算符右侧的值（此示例中为 x、y 和 z 值）被视为表达式，即使它们碰巧是列值。如果 IN 左侧的值是一个列，则使用该列的关联。如果该值是一个表达式，则不发生转换。

执行比较的方式也会受到使用 COLLATE 子句的影响。有关详细信息，请参阅 COLLATE。

数据类型与数学运算符

对于每个支持的数学运算符（*、/、%、+ 和 -），在计算表达式之前，会将数字关联应用于每个操作数。如果任一操作数无法成功转换为 NUMERIC 存储类，则表达式的计算结果将为 NULL。

如果使用了串联运算符 ||，则在计算表达式之前将每个操作数转换为 TEXT 存储类。如果任一操作数无法转换为 TEXT 存储类，则表达式的结果为 NULL。在两种情况下可能发生无法转换值的此问题：操作数的值为 NULL，或者它是包含非 TEXT 存储类的 BLOB。

数据类型与排序

通过 ORDER BY 子句对值排序时，具有存储类 NULL 的值排在最前面。它们后跟按数字顺序散布的 INTEGER 和 REAL 值，再后跟按二进制顺序或基于指定排序规则 （BINARY 或 NOCASE）的 TEXT 值。最后是按二进制顺序的 BLOB 值。在排序之前，不发生存储类转换。

数据类型与分组

使用 GROUP BY 子句对值分组时，将具有不同存储类的值视为非重复。例外是 INTEGER 和 REAL 值，如果它们在数值上相等，则认为它们相等。 GROUP BY 子句不会导致将关联应用于任何值。

数据类型与复合 SELECT 语句

复合 SELECT 运算符 UNION、 INTERSECT 和 EXCEPT 执行值之间的隐式比较。在执行这些比较之前，可能会将关联应用于每个值。相同的关联 （如果有）将应用于在复合 SELECT 结果集的单个列中可能返回的所有值。所应用的关联是在该位置中具有列值 （而不是其他某种表达式）的第一个构成性 SELECT 语句所返回的列的关联。如果对于给定的复合 SELECT 列，构成性 SELECT 语句都不返回列值，则在对该列中的值进行比较之前，不将关联应用于这些值。

第 67 章：SQL 错误详细消息、 ID 和参数

SQLError 类表示在使用 Adobe AIR 本地 SQL 数据库时可能出现的各种错误。对于任何给定的异常，SQLError 实例都具有一个包含中文错误消息的 details 属性。此外，每条错误消息都具有关联的唯一标识符，该标识符在 SQLError 对象的 detailID 属性中提供。使用 detailID 属性，应用程序可标识特定的 details 错误消息。应用程序可以为最终用户提供用其所在地区的语言表示的替换文本。可在错误消息字符串的合适位置替换 detailArguments 数组的参数值。这对于要直接向使用特定区域设置的最终用户显示错误的 details 属性错误消息的应用程序非常有用。

下表列出了 detailID 值和关联的中文错误消息文本。消息中的占位符文本指示在运行时替换 detailArguments 值的位置。此列表可用作对在 SQL 数据库操作中可能出现的错误消息进行本地化的源。

SQLError detailID	中文错误详细消息和参数
1001	连接已关闭。
1102	必须打开数据库才能执行此操作。
1003	在参数属性中找到了 %s [,| 和 %s] 参数名称，但在指定的 SQL 中未找到。
1004	参数计数不匹配。在指定的 SQL 中找到了 %d 个值，但在参数属性中设置了 %d 个值。应有 %s [,| 和 %s] 的值。
1005	无法打开自动压缩。
1006	无法设置 pageSize 值。
1007	未找到名为 “%s”、类型为 “%s” 的架构对象 （在数据库 “%s” 中）。
1008	未找到名为 “%s” 的架构对象 （在数据库 “%s” 中）。
1009	未找到类型为 “%s” 的架构对象 （在数据库 “%s” 中）。
1010	在数据库 “%s” 中未找到架构对象。
2001	分析器堆栈溢出
2002	函数 “%s” 上的参数过多
2003	“%s” 附近：语法错误
2004	已有另一个表或索引采用此名称：“%s”
2005	在 SQL 中不允许 PRAGMA。
2006	不是可写入目录。
2007	联接类型未知或不受支持：“%s %s %s”
2008	当前不支持 RIGHT OUTER JOIN 和 FULL OUTER JOIN。
2009	NATURAL 联接可能没有 ON 或 USING 子句。
2010	在同一联接中不能同时具有 ON 和 USING 子句。
2011	无法使用列 “%s” 进行联接 – 并非两个表中都有此列。
2012	作为表达式一部分的 SELECT 仅允许单个结果。
2013	没有这样的表：“[%s.]%s”
2014	未指定表。
2015	结果集中的列过多 |“%s” 上的列过多。
2016	%s ORDER|GROUP BY 项数超出范围 – 应介于 1 和 %d 之间
2017	ORDER BY 子句中的项过多。
2018	%s ORDER BY 项数超出范围 – 应介于 1 和 %d 之间。
2019	%r ORDER BY 项与结果集中的任何列都不匹配。
2020	ORDER BY 子句应在 “%s” 之后而不是之前。
2021	LIMIT 子句应在 “%s” 之后而不是之前。
2022	“%s” 左右 SELECT 的结果列数量不同。
2023	HAVING 前需要有 GROUP BY 子句。
2024	在 GROUP BY 子句中不允许聚合函数。
2025	聚合中的 DISTINCT 必须后跟一个表达式。
2026	复合 SELECT 中的项过多。
2027	ORDER|GROUP BY 子句中的项过多
2028	临时触发器可能没有限定名
2030	触发器 “%s” 已存在
2032	无法在视图上创建 BEFORE|AFTER 触发器：“%s”。

2033	无法在表上创建 INSTEAD OF 触发器：“%s”。
2034	没有这样的触发器：“%s”
2035	不支持递归触发器 （“%s”）。
2036	没有这样的列：%s[。 %s[.%s]]
2037	SQL 不允许 VACUUM。
2043	表 “%s”：索引函数返回的计划无效。
2044	一个联接中最多只能有 %d 个表。
2046	无法添加 PRIMARY KEY 列。
2047	无法添加 UNIQUE 列。
2048	无法添加具有默认值 NULL 的 NOT NULL 列。
2049	无法添加具有非常量默认值的列。
2050	无法向视图添加列。
2051	SQL 不允许 ANALYZE。
2052	名称无效：“%s”
2053	SQL 不允许 ATTACH。
2054	%s“%s” 不能引用数据库 “%s” 中的对象
2055	禁止访问 “[%s.]%s.%s”。
2056	未经授权。
2058	没有这样的视图：“[%s.]%s”
2060	临时表的名称必须是非限定的。
2061	表 “%s” 已存在。
2062	已存在此名称的索引：“%s”
2064	列名重复：“%s”
2065	表 “%s” 有多个主键。
2066	仅在 INTEGER PRIMARY KEY 上允许 AUTOINCREMENT
2067	没有这样的排序规则序列：“%s”
2068	视图中不允许有参数。
2069	视图 “%s” 是循环定义的。
2070	不能删除表 “%s”。
2071	使用 DROP VIEW 删除视图 “%s”
2072	使用 DROP TABLE 删除表 “%s”
2073	“%s” 上的外键应该仅引用表 “%s” 的一列。
2074	外键中的列数与引用表中的列数不匹配。
2075	外键定义中的列 “%s” 未知。
2076	不能编制表 “%s” 的索引。
2077	不能编制视图的索引。
2080	指定的 ON CONFLICT 子句相冲突。
2081	没有这样的索引：“%s”
2082	不能删除与 UNIQUE 或 PRIMARY KEY 约束关联的索引。
2083	SQL 不允许 BEGIN。
2084	SQL 不允许 COMMIT。
2085	SQL 不允许 ROLLBACK。
2086	无法打开临时数据库文件以存储临时表。
2087	无法标识要重新编制索引的对象。
2088	不能修改表 “%s”。
2089	“%s” 是视图，无法进行修改。
2090	变量编号必须介于 ?0 和 ?%d< 之间
2092	误用了别名聚合 “%s”
2093	列名不明确：“[%s.[%s.]]%s”
2094	没有这样的函数：“%s”
2095	函数 “%s” 的参数数量不正确
2096	在 CHECK 约束中禁止使用子查询。
2097	在 CHECK 约束中禁止使用参数。
2098	表达式树过大 （最大深度为 %d）
2099	RAISE() 只能在触发器程序中使用
2100	表 “%s” 具有 %d 个列，但提供了 %d 个值
2101	数据库架构已锁定：“%s”

2102	语句过长。
2103	由于存在活动的语句，无法删除 / 修改排序规则序列
2104	附加的数据库过多 — 最多 %d 个
2105	无法在事务内附加数据库。
2106	数据库 “%s” 已在使用中。
2108	附加的数据库必须使用与主数据库相同的文本编码。
2200	内存不足。
2201	无法打开数据库。
2202	无法在事务内分离数据库。
2203	无法分离数据库：“%s”
2204	数据库 “%s” 已锁定。
2205	无法获取数据库上的读取锁定。
2206	[列 | 列]“%s”[,“%s”] 不是 [唯一的 | 不是] 唯一的。
2207	数据库架构格式不正确。
2208	不支持的文件格式。
2209	标记无法识别：“%s”
2300	无法将文本值转换为数值。
2301	无法将字符串值转换为日期。
2302	无法将浮点值转换为整数而不丢失数据。
2303	无法回滚事务 - SQL 语句正在执行。
2304	无法提交事务 - SQL 语句正在执行。
2305	数据库表已锁定：“%s”
2306	只读表。
2307	字符串或 blob 过大。
2309	无法打开索引列以写入。
2400	无法打开类型为 %s 的值。
2401	没有这样的 rowid：%s
2402	此对象名称被保留供内部使用：“%s”
2403	不能更改视图 “%s”。
2404	列 “%s” 的默认值并不固定。
2405	无权使用函数 “%s”
2406	误用了聚合函数 “%s”
2407	误用了聚合：“%s”
2408	没有这样的数据库：“%s”
2409	表 “%s” 没有名为 “%s” 的列
2501	没有这样的模块：“%s”
2508	没有这样的保存点：“%s”
2510	无法回滚 - 没有事务处于活动状态。
2511	无法提交 - 没有事务处于活动状态。

第 68 章 : Adobe 图形汇编语言 (AGAL)

Adobe 图形汇编语言 (AGAL) 是一种着色器语言，用于定义顶点和片段渲染程序。必须将 AGAL 程序上载到本文档中介绍的二进制字节码格式的渲染上下文。

AGAL 字节码格式

AGAL 字节码必须使用 Endian.LITTLE_ENDIAN 格式。

字节码标头

AGAL 字节码必须以一个 7 字节标头开头。

```
A0 01000000 A1 00 -- for a vertex program
A0 01000000 A1 01 -- for a fragment program
```

偏移量 （字节数）	大小 （字节数）	名称	说明
0	1	magic	必须为 0xa0
1	4	version	必须为 1
5	1	shader type ID	必须为 0xa1
6	1	shader type	0 表示顶点程序； 1 表示片段程序

标记

标头后直接跟任意数量的标记。每个标记的大小都为 192 位 （24 字节），且格式始终为：

[opcode][destination][source1][source2 or sampler]

并不是每一个 opcode 都使用所有这些字段。不使用的字段必须设置为 0。

操作代码

[opcode] 字段的大小为 32 位，可具有以下值之一：

名称	Opcode	操作	说明
mov	0x00	移动	将数据从 source1 移动到 destination，按组件
add	0x01	相加	destination = source1 + source2，按组件
sub	0x02	相减	destination = source1 - source2，按组件
mul	0x03	相乘	destination = source1 * source2，按组件
div	0x04	除以	destination = source1 / source2，按组件
rcp	0x05	倒数	destination = 1/source1，按组件
min	0x06	最小值	destination = minimum(source1,source2)，按组件
max	0x07	最大值	destination = maximum(source1,source2)，按组件
frc	0x08	分数	destination = source1 - (float)floor(source1)，按组件

名称	Opcode	操作	说明
sqt	0x09	平方根	destination = sqrt(source1)，按组件
rsq	0x0a	平方根倒数	destination = 1/sqrt(source1)，按组件
pow	0x0b	幂	destination = pow(source1,source2)，按组件
log	0x0c	对数	destination = log_2(source1)，按组件
exp	0x0d	指数	destination = 2^source1，按组件
nrm	0x0e	标准化	destination = normalize(source1)，按组件 （仅生成一个 3 组件结果，目标必须掩码为 .xyz 或更小）
sin	0x0f	正弦	destination = sin(source1)，按组件
cos	0x10	余弦	destination = cos(source1)，按组件
crs	0x11	叉积	destination.x = source1.y * source2.z - source1.z * source2.y destination.y = source1.z * source2.x - source1.x * source2.z destination.z = source1.x * source2.y - source1.y * source2.x （仅生成一个 3 组件结果，目标必须掩码为 .xyz 或更小）
dp3	0x12	点积	destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z
dp4	0x13	点积	destination = source1.x*source2.x + source1.y*source2.y + source1.z*source2.z + source1.w*source2.w
abs	0x14	取绝对值	destination = abs(source1)，按组件
neg	0x15	求反	destination = -source1，按组件
sat	0x16	饱和	destination = maximum(minimum(source1,1),0)，按组件
m33	0x17	矩阵连乘 3x3	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) （仅生成一个 3 组件结果，目标必须掩码为 .xyz 或更小）
m44	0x18	矩阵连乘 4x4	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w) destination.w = (source1.x * source2[3].x) + (source1.y * source2[3].y) + (source1.z * source2[3].z) + (source1.w * source2[3].w)

名称	Opcode	操作	说明
m34	0x19	矩阵连乘 3x4	destination.x = (source1.x * source2[0].x) + (source1.y * source2[0].y) + (source1.z * source2[0].z) + (source1.w * source2[0].w) destination.y = (source1.x * source2[1].x) + (source1.y * source2[1].y) + (source1.z * source2[1].z) + (source1.w * source2[1].w) destination.z = (source1.x * source2[2].x) + (source1.y * source2[2].y) + (source1.z * source2[2].z) + (source1.w * source2[2].w) （仅生成一个 3 组件结果，目标必须掩码为 .xyz 或更小）
kil	0x27	丢弃 （仅适用于片段着色器）	如果单个标量源组件小于零，则将丢弃片段并不会将其绘制到帧缓冲区。（目标寄存器必须全部设置为 0）
tex	0x28	纹理取样 （仅适用于片段着色器）	destination 等于从坐标 source1 上的纹理 source2 进行加载。在这种情况下， source2 必须采用取样器格式。
sge	0x29	大于等于时设置	destination = source1 >= source2 ? 1 : 0，按组件
slt	0x2a	小于时设置	destination = source1 < source2 ? 1 : 0，按组件
seq	0x2c	相等时设置	destination = source1 == source2 ? 1 : 0，按组件
sne	0x2d	不相等时设置	destination = source1 != source2 ? 1 : 0，按组件

在 AGAL2 中引入了以下 opcode：

名称	Opcode	操作	说明
ddx	0x1a	X 偏导数	将 source1 的 X 偏导数加载到目标中。
ddy	1x0b	Y 偏导数	将 source1 的 Y 偏导数加载到目标中。
ife	0x1c	如果相等	如果 source1 等于 source2 则跳转。
ine	0x1d	如果不相等	如果 source1 不等于 source2 则跳转。
ifg	1x0e	如果大于	如果 source1 大于等于 source2 则跳转。
ifl	1x0f	如果小于	如果 source1 小于 source2 则跳转。
els	0x20	else	Else 块
eif	0x21	Endif	关闭 if 或 else 块。

Destination 字段格式

[destination] 字段的大小为 32 位：

```
31.............................0
----TTTT----MMMMNNNNNNNNNNNNNNNN
```

T = 寄存器类型 （4 位）

M = 写入掩码 （4 位）

N = 寄存器编号 （16 位）

- = 未定义，必须为 0

Source 字段格式

[source] 字段的大小为 64 位：

```
63.............................................................0
D-------------QQ----IIII----TTTTSSSSSSSSOOOOOOOONNNNNNNNNNNNNNNN
```

D = Direct=0/Indirect=1 表示直接使用 Q 并忽略 I， 1 位

Q = 索引寄存器组件选择 （2 位）

I = 索引寄存器类型 （4 位）

T = 寄存器类型 （4 位）

S = 重排 （8 位，每个组件 2 位）

O = 间接偏移量 （8 位）

N = 寄存器编号 （16 位）

- = 未定义，必须为 0

Sampler 字段格式

tex opcode 的第二个源字段必须为 [sampler] 格式，大小为 64 位：

```
63.............................................................0
FFFFMMMMWWWWSSSSDDDD--------TTTT--------BBBBBBBBNNNNNNNNNNNNNNNN
```

N = 取样器寄存器编号 （16 位）

B = 纹理详细级别 (LOD) 偏差，带符号整数，以 8 为单位递增。使用的浮点值为 b/8.0 （8 位）

T = 寄存器类型，必须为 5， Sampler （4 位）

F = 滤镜 （0=nearest、 1=linear）（4 位）

M = Mipmap （0=disable、 1=nearest、 2=linear）

W = 环绕 （0=clamp、 1=repeat）

S = 特殊标志位 （必须为 0）

D = 维度 （0=2D、 1=Cube）

程序寄存器

使用的寄存器数取决于使用的 Context3D 配置文件。下表中定义了寄存器数与其用法：

名称 AGAL2 AGAL3 用法	值	AGAL	
每个片段程序的数量 每个顶点程序的数量 每个片段程序的数量 每个顶点程序的数量		每个片段程序的数量	每个顶点程序的数量
Context 3D 配置文件支持 标准 扩展标准		低于标准	
SWF 版本 25 28 及更高版本		低于 25	
属性 NA 8 NA 16 顶点着色器输入；从使用 Context3D.setVertexBufferAt() 指定的顶点缓冲区读取。	0	NA	8

名称 **AGAL2** **AGAL3** 用法	值	**AGAL**	
常数 64 250 200 250 着色器输入；使用 Context3D.setProgramConstants() 系列函数设置。	1	28	128
临时 26 26 26 26 用于计算的临时寄存器，无法从程序外部访问。	2	8	8
输出 1 1 1 1 着色器输出：在顶点程序中，输出的是剪辑空间位置；在片段程序中，输出的是颜色。	3	1	1

名称 **AGAL2** **AGAL3** 用法	值	**AGAL**	
渐变 10 10 10 10 在顶点着色器和片段着色器之间传递插补数据。将顶点程序的渐变寄存器用作片段程序的输入。根据与三角形顶点的距离插补计算所需的值。	4	8	8

名称 AGAL2 AGAL3 用法	值	AGAL	
取样器 16 NA 16 NA 片段着色器输入；从使用 Context3D.setTextureAt() 指定的纹理读取。	5	8	NA
片段寄存器 1 NA 1 NA 只能写入，用于重新写入在顶点着色器中写入的 z 值（即深度值）。	6	NA	NA
标记 1024 2048		200	

可以在此找到最新的 AGAL Mini Assembler。