使用 ADOBE® FLEX® 4.6
和 ADOBE® FLASH® BUILDER™ 4.6
开发移动设备应用程序

Adobe®

法律声明

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

目录

第 1 章 : 入门

移动设备应用程序入门 1

移动设备、桌面和浏览器应用程序开发的差异 3

第 2 章 : 开发环境

在 Flash Builder 中创建 Android 应用程序 8

在 Flash Builder 中创建 iOS 应用程序 9

在 Flash Builder 中创建 BlackBerry Tablet OS 应用程序 10

创建 ActionScript 移动设备项目 10

使用本机扩展 11

设置移动设备项目首选参数 12

连接 Google Android 设备 15

使用 Flash Builder 的 Apple iOS 开发过程 17

第 3 章 : 用户界面和布局

移动设备应用程序的布局 21

在移动设备应用程序中处理用户输入 27

定义移动设备应用程序和启动屏幕 29

定义移动设备应用程序中的视图 33

定义移动设备应用程序中的选项卡 41

在移动设备应用程序中创建多个窗格 44

定义移动设备应用程序中的导航控件、标题控件和操作控件 52

在移动设备应用程序中使用滚动条 56

定义移动设备应用程序中的菜单 61

为移动设备应用程序中耗时的活动显示忙碌指示符 65

将切换开关添加到移动设备应用程序中 66

将 callout 容器添加到移动设备应用程序中 68

定义移动设备应用程序中的过渡效果 79

在移动设备应用程序中选择日期和时间 83

在移动设备应用程序中使用微调框列表 93

第 4 章 : 应用程序设计和工作流

在移动设备应用程序中启用持久化机制 105

在一个移动设备应用程序中支持多个屏幕大小和 DPI 值 108

第 5 章 : 文本

在移动设备应用程序中使用文本 120

移动设备应用程序中用户与文本的交互 127

在移动设备应用程序中使用软键盘 128

在移动设备应用程序中嵌入字体 139

第 6 章 : 外观设计

移动设备外观设计的基础知识 141
为移动设备应用程序创建外观 145
应用自定义移动设备外观 151

第 7 章 : 测试和调试

管理启动配置 153
在桌面上测试和调试移动设备应用程序 153
在设备上测试和调试移动设备应用程序 154

第 8 章 : 在设备上安装

在 Google Android 设备上安装应用程序 158
在 Apple iOS 设备上安装应用程序 158

第 9 章 : 打包和导出

将移动设备应用程序打包并导出到在线商店 160
导出用于发行的 Android APK 包 160
导出用于发行的 Apple iOS 包 161
使用命令行进行创建、测试和部署 162

第 1 章：入门

移动设备应用程序入门

Adobe Flex 将 Flex 和 Adobe Flash Builder 引入到智能手机和平板电脑中。现在，利用 Adobe AIR，可以像在桌面平台上一样在 Flex 中轻松而高质量地开发移动设备应用程序。

许多现有的 Flex 组件已扩展到移动设备上，其中包括增加了对触摸滚动的支持。 Flex 还包含一组新组件，可用来轻松构建采用手机和平板电脑标准设计模式的应用程序。

Flash Builder 也进行了更新，增加了许多新功能，用以支持针对移动设备开发应用程序。使用 Flash Builder，您可以在桌面上或直接在移动设备上开发、测试和调试应用程序。

Adobe 开发人员 Mark Doherty 发布了有关为桌面计算机、移动设备和平板电脑构建应用程序的视频。

Adobe 开发人员 James Ward 发布了有关使用 Flex 构建移动设备应用程序的视频。

Adobe Community 专家 Joseph Labrecque 发表了关于 Mobile Flex Demonstration 的视频。

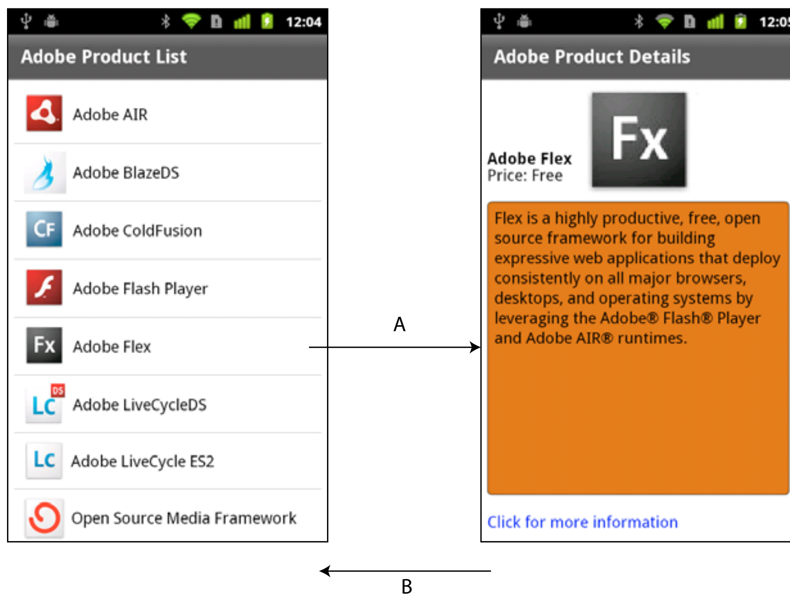
Flash 开发人员 Fabio Biondi 发表了文章：使用 Flash Builder 为 Android 设备创建基于 AIR 的 YouTube 播放器。

设计移动设备应用程序

由于移动设备上使用较小的屏幕，因此移动设备应用程序通常采用不同于基于浏览器的应用程序的设计模式。开发移动设备应用程序时，通常会将内容细分为可以在移动设备设备上显示的一系列视图。

每个视图都包含侧重于单个任务或含有单组信息的组件。用户在视图中点击组件时，通常可以从一个视图 “ 下钻 ” 到或切换到另一个视图。然后，用户可以使用设备的后退按钮返回到上一个视图，或者构建指向应用程序的导航路径。

在以下示例中，应用程序的初始视图显示了一个产品列表：



A. 选择一个列表项以更改应用程序中的视图。 B. 使用设备的后退按钮返回到上一视图。

用户选择列表中的某产品以获得更多信息。选择后会将视图切换为产品的详细说明。

如果要针对移动设备、Web 和桌面平台设计应用程序，通常需要为这些平台设计单独的用户界面。但是，应用程序可以在所有平台之间共享任何基础模型和数据访问代码。

针对手机和平板电脑构建应用程序

针对平板电脑开发应用程序时，不需要像开发手机程序时那样担心屏幕大小。您不必围绕小视图构建平板电脑应用程序。可以使用标准 Spark Application 容器以及受支持的移动设备组件和外观来构建应用程序。

注：您可以基于 Spark Application 容器为移动电话创建应用程序。但是，通常会改为使用 ViewNavigatorApplication 和 TabbedViewNavigatorApplication 容器。

在 Flash Builder 中针对平板电脑创建移动设备项目的步骤与针对移动设备创建项目的步骤一样。平板电脑和移动设备应用程序要求使用相同的移动设备主题，从而使用针对移动设备应用程序优化的组件和外观。

Flash Builder 中的作者移动设备应用程序

Flash Builder 为移动设备开发引入了一个高效的设计、构建和调试工作流。Flash Builder 中的移动设备功能旨在使开发基于 ActionScript 或 Flex 的移动设备应用程序与开发桌面或 Web 应用程序一样轻松。

Flash Builder 提供了两种可选的测试和调试方案。您可以使用 AIR Debug Launcher (ADL) 在桌面上启动和调试应用程序。要实现更好的控制，可以直接在移动设备上启动和调试应用程序。无论是哪种情况，您都可以使用 Flash Builder 调试各种功能，包括使用 “ 变量 ” 和 “ 表达式 ” 面板设置断点和检查应用程序的状态。

在应用程序做好部署的准备后，请使用 “ 导出发行版 ” 过程，这与桌面和 Web 应用程序的部署过程一样。主要区别在于，在您导出移动设备项目的发行版时， Flash Builder 会将版本打包为本机安装程序，而不是 .air 文件。例如，在 Android 上，Flash Builder 会生成一个与本机 Android 应用程序包看起来一样的 .apk 文件。通过这个本机安装程序，可以在每个平台上像处理本机应用程序那样分发基于 AIR 的应用程序。

在 AIR 中部署移动设备应用程序

使用 Adobe AIR 为移动设备部署在 Flex 中构建的移动设备应用程序。要部署移动设备应用程序的设备必须支持 AIR。

您的应用程序可以充分利用 AIR 与移动设备平台的集成。例如，移动设备应用程序可以处理硬件后退按钮和菜单按钮，以及访问本地存储。您还可以使用由 AIR 为移动设备提供的所有功能。这些功能包括地理定位、加速度传感器和相机集成。

在移动设备上，运行 Flex 中内置的应用程序之前不必安装 AIR。用户第一次运行 Flex 中内置的应用程序时，会被提示下载 AIR。

要熟悉 AIR 并了解 AIR 功能的更多信息，请参阅以下内容：

- 关于 Adobe AIR
- AIR 应用程序的调用和终止
- 处理 AIR 运行时和操作系统信息
- 使用 AIR 本机窗口
- 在 AIR 中使用本地 SQL 数据库

开发移动设备应用程序时，不能对 AIR 使用以下 Flex 组件：WindowedApplication 和 Window。可以改为使用 ViewNavigatorApplication 和 TabbedViewNavigatorApplication 容器。在针对平板电脑开发移动设备应用程序时，也可以使用 Spark Application 容器。

有关更多信息，请参阅使用 Flex AIR 组件和第 29 页的 “ 定义移动设备应用程序和启动屏幕 ”。

在应用程序中使用移动设备主题

主题用于定义应用程序可视组件的观感。主题可以为应用程序定义像配色方案或常用字体一样简单的元素，或者也可以彻底改变应用程序所使用的所有组件的外观。

只有在当前主题中包含 CSS 样式时，才可以对 Flex 组件设置这些样式。要确定当前主题是否支持 CSS 样式，请在用于 Adobe Flash Platform 的 ActionScript 3.0 参考中查看样式的条目。

Flex 支持三个主要的主题：移动设备、 Spark 和 Halo。移动设备主题定义了在创建移动设备应用程序时 Flex 组件的默认外观。为了使某些 Flex 组件与移动设备主题兼容， Adobe 为这些组件创建了新的外观。因此，一些组件含有特定于某主题的外观。

使用 Flex 构建的应用程序可用在不同的移动设备上，这些设备可以有不同的屏幕大小和分辨率。 Flex 为移动设备组件提供与 DPI 无关的外观，从而简化了生成与分辨率无关的应用程序的过程。有关移动设备外观的更多信息，请参阅第 141 页的 “ 移动设备外观设计的基础知识 ”。

有关样式和主题的更多信息，请参阅 Styles and themes 和第 141 页的 “ 移动设备样式 ”。

社区资源

阅读 Flex 和 Flash Builder 中新增功能的相关内容：

- 介绍 Adobe Flex 4.5 SDK，作者为 Adobe 产品经理 Deepa Subramaniam
- Adobe 产品设计人员 Narciso Jaramillo 发表了文章：使用 Adobe Flex SDK 和 Flash Builder 进行移动开发。
- Flex 4.6 SDK 的新增功能（作者为 Adobe 产品经理 Jacob Surber）以及 Flash Builder 4.6 的新增功能（作者为 Adobe 产品经理 Adam Lehman）。

Flex 开发人员中心包含许多资源，可以帮助您使用 Flex 构建移动设备应用程序：

- 入门文章、链接和教程
- 在 Flex 中构建的实际应用程序的示例
- Flex 手册，包含常见编码问题的解答
- Flex 社区及其它 Flex 网站的链接

另一个资源是 Adobe TV，包含 Adob 工程师、产品宣传人员以及客户所提交的关于在 Flex 中开发应用程序的视频。其中一个可用的视频是 Build your first mobile application in Flash Builder。

移动设备、桌面和浏览器应用程序开发的差异

使用 Flex 可以针对以下部署环境开发应用程序：

浏览器 将应用程序部署为 SWF 文件，以便在运行于浏览器内的 Flash Player 中使用。

桌面 针对桌面计算机 （例如 Windows 计算机或 Macintosh）部署独立的 AIR 应用程序。

移动设备 针对移动设备 （例如移动设备或平板电脑）部署独立的 AIR 应用程序。

Flash Player 运行时与 AIR 运行时类似。在两个运行时中可以执行的操作大部分都相同。除允许在浏览器外部署独立的应用程序之外， AIR 还提供与主机平台的紧密集成。通过这种集成可以实现许多功能，例如访问设备的文件系统、创建和处理本地 SQL 数据库，等等。

设计和开发移动设备应用程序的注意事项

移动触摸屏设备的应用程序与桌面和浏览器应用程序有以下不同之处：

- 为了通过触控输入实现简便的操作，移动设备组件通常具有比桌面或浏览器应用程序中更大的点击区域。
- 在触摸屏设备上，滚动等操作的交互模式不同。
- 由于屏幕区域有限，因此移动设备应用程序通常设计为在屏幕上同时只显示少量的用户界面。
- 用户界面的设计必须考虑不同设备之间屏幕分辨率的差异。
- 相较于桌面设备，手机和平板电脑的 CPU 和 GPU 性能更为有限。
- 由于移动设备上可用内存有限，因此应用程序必须注意节约内存。
- 移动设备应用程序可以随时 （例如在接听电话或短信时）退出和重新启动。

因此，构建适用于移动设备的应用程序不仅仅是将桌面应用程序缩小为不同的屏幕大小。通过 Flex，您可以针对每个外观因素分别创建适当的用户界面，同时在移动设备、浏览器和桌面项目之间共享基础模型和数据访问代码。

在移动设备应用程序中使用 Spark 和 MX 组件的限制

在 Flex 中创建移动设备应用程序时，将使用 Spark 组件集。Sprak 组件在 spark.components.* 包中定义。但由于性能原因，或者由于并非所有 Spark 组件都具有移动设备主题外观，移动设备应用程序并不支持整个 Spark 组件集。

除 MX 图表控件和 MX Spacer 控件外，移动设备应用程序不支持 mx.* 包中定义的 MX 组件集。

下表列出了在移动设备应用程序中可以使用、不可以使用或需要谨慎使用的组件：

组件	组件	能否用在移动设备程序中?	说明
Spark ActionBar Spark BusyIndicator Spark Callout Spark CalloutButton Spark DateSpinner Spark SpinnerList Spark SpinnerListContainer	Spark TabbedViewNavigator Spark TabbedViewNavigatorApplication Spark ToggleSwitch Spark View Spark ViewMenu Spark ViewNavigator Spark ViewNavigatorApplication	是	这些新组件支持移动设备应用程序。
Spark Button Spark CheckBox Spark DataGroup Spark Group/HGroup/VGroup/TileGroup Spark Image/BitmapImage Spark Label	Spark List Spark RadioButton/RadioButtonGroup Spark SkinnableContainer Spark Scroller Spark TextArea Spark TextInput	是	这些组件大部分都具有移动设备主题外观。尽管 Label、Image 和 BitmapImage 不具有移动设备外观，但也可以使用。 某些 Spark 布局容器 （例如 Group 及其子类）不具有外观。因此，可以在移动设备应用程序中使用这些组件。
其他 Spark Skinnable 组件		建议不要使用	除上面列出的组件外，建议不要使用其它 Skinnable Spark 组件，因为这些组件不具有适用于移动设备主题的外观。如果组件不具有移动设备主题外观，您可以为应用程序创建一个外观。
Spark DataGrid	Spark RichEditableText Spark RichText	建议不要使用	出于性能考虑，建议不要使用这些组件。尽管这些组件可以用在移动设备应用程序中，但这样做会影响性能。 对于 DataGrid 控件，性能取决于呈示的数据量。对于 RichEditableText 和 RichText 控件，性能取决于文本量及应用程序中的控件数量。

组件	组件	能否用在移动设备程序中?	说明
除 Spacer 和图表以外的 MX 组件		否	移动设备应用程序不支持 MX Button、CheckBox、List、DataGrid 等 MX 组件。这些组件对应于 mx.controls.* 和 mx.containers.* 包中定义的 Flex 3 组件。
MX Spacer		是	Spacer 不使用外观，因此可以用在移动设备应用程序中。
MX 图表组件		是，但存在性能隐患	可以在移动设备应用程序中使用 AreaChart 和 BarChart 等 MX 图表控件。 MX 图表控件位于 mx.charts.* 包中。 但这会导致移动设备无法达到最佳性能，具体取决于图表数据的规模和类型。 默认情况下， Flash Builder 的移动设备项目库路径中不包含 MX 组件。要在应用程序中使用 MX 图表组件，请在库路径中添加 mx.swc 和 charts.swc。

移动设备应用程序不支持以下 Flex 功能:

- 不支持拖放操作
- 不支持 ToolTip 控件
- 不支持 RSL

移动设备应用程序在性能方面的注意事项

由于移动设备在性能方面的局限性，移动设备应用程序的某些开发环节与浏览器和桌面应用程序的开发不同。下面是一些性能方面的注意事项:

- 以 **ActionScript** 编写项呈示器

  开发移动设备应用程序时，您希望使列表滚动具有尽可能高的性能。以 ActionScript 编写项呈示器可以获得最高的性能。尽管您可以使用 MXML 编写项呈示器，但这样可能会降低应用程序的性能。

  Flex 提供两个项呈示器，且已优化为适合在移动设备应用程序中使用：spark.components.LabelItemRenderer 和 spark.components.IconItemRenderer。有关这些项呈示器的更多信息，请参阅 Using a mobile item renderer with a Spark list-based control。

  有关以 ActionScript 创建自定义项呈示器的更多信息，请参阅 Custom Spark item renderers。有关移动设备和桌面项呈示器之间不同点的更多信息，请参阅 Differences between mobile and desktop item renderers。

- 使用 **ActionScript** 和已编译的 **FXG** 图形或位图开发自定义外观

  Flex 附带的移动设备外观采用 ActionScript 编写，带有已编译的 FXG 图形，可提供最高的性能。您可以使用 MXML 编写外观，但应用程序的性能可能会有所降低，具体取决于使用 MXML 外观的组件数量。要获得最高的性能，请以 ActionScript 编写外观并使用已编译的 FXG 图形。有关更多信息，请参阅 Spark Skinning 和 FXG and MXML graphics。

- 利用使用 **StageText** 的文本输入组件

  添加 TextInput 和 TextArea 之类的文本输入组件时，请使用默认值。这些控件将 StageText 用做文本输入的基础机制，这种机制会挂接到本机文本输入类。这提高了性能，并允许您使用自动纠正、自动大写、文本限制和自定义软键盘等本机功能。

使用 StageText 也有一些缺点，如不能滚动控件所在的视图等。另外，也不能使用嵌入字体，不能对基于 StageText 的控件使用自定义大小调整。如有必要，可以使用基于 TextField 类的文本输入控件。

有关更多信息，请参阅第 120 页的 " 在移动设备应用程序中使用文本 "。

- 在移动设备应用程序中谨慎使用 **MX** 图表组件

  可以在移动设备应用程序中使用 AreaChart 和 BarChart 等 MX 图表控件。但它们会影响性能，具体取决于图表数据的规模和类型。

博客 Nahuel Foronda 编写了有关 ActionScript 中的 Mobile ItemRenderer 的一系列文章。

博客 Rich Tretola 为移动设备应用程序编写了一个有关使用 ItemRenderer 创建列表的手册条目。

第 2 章：开发环境

在 Flash Builder 中创建 Android 应用程序

这是在 Google Android 平台上创建 Flex 移动设备应用程序的常规工作流。该工作流假设您已对移动设备应用程序进行了设计。有关更多信息，请参阅第 1 页的“设计移动设备应用程序”。

Adobe 宣讲师 Mike Jones 通过提供 10 tips when developing for multiple devices 分享他在开发多平台游戏模式时汲取的一些经验。

AIR 要求

Flex 移动设备项目和 ActionScript 移动设备项目要求使用 AIR 2.6 或更高版本。可以在支持 AIR 2.6 或更高版本 AIR 的物理设备上运行移动设备项目。

只能在运行 Android 2.2 或更高版本的受支持 Android 设备上安装 AIR 2.6 或更高版本。有关支持的 Android 设备的完整列表，请参阅 Certified Devices。另外，查看移动系统要求中有关在 Android 设备上运行 Adobe AIR 的最低系统要求。

注： 如果没有支持 AIR 2.6 或更高版本 AIR 的设备，可以使用 Flash Builder 在桌面上启动和调试移动设备应用程序。

每个版本的 Flex SDK 中都包含所需版本的 Adobe AIR。如果在设备上从早期版本的 Flex SDK 安装了移动设备应用程序，请从该设备上卸载 AIR。当您在设备上运行或调试移动设备应用程序时， Flash Builder 将安装正确版本的 AIR。

创建应用程序

1 在 Flash Builder 中，选择“文件”>“新建”>“Flex 移动设备项目”。

   Flex 移动设备项目是 AIR 项目的特殊类型。请遵循新建项目向导中的提示，就像 Flash Builder 中其它任何 AIR 项目一样。有关更多信息，请参阅 Flex 移动设备项目。

   要设置特定于 Android 的移动设备首选参数，请参阅第 12 页的“设置移动设备项目首选参数”。

   当您创建 Flex 移动设备项目时， Flash Builder 生成项目的以下文件：

   - ***ProjectName*.mxml**

     项目的默认应用程序文件。

     默认情况下， Flash Builder 使用项目名称来命名该文件。如果项目名称中包含非法 ActionScript 字符，则 Flash Builder 将该文件命名为 Main.mxml。此 MXML 文件中包含项目的基本 Spark 应用程序标签。基本 Spark 应用程序标签可以是 ViewNavigatorApplication 或 TabbedViewNavigatorApplication。

     通常，除了在所有视图中显示的 ActionBar 内容外，不需将其它内容直接添加到默认应用程序文件中。要将内容添加至 ActionBar，请设置 navigatorContent、 titleContent 或 actionContent 属性。

   - ***ProjectName*HomeView.mxml**

     代表项目的初始化视图的文件。 Flash Builder 在视图包中放置文件。 ***ProjectName*.mxml** 中 ViewNavigatorApplication 标签的 firstView 属性将该文件指定为应用程序的默认打开视图。

   有关定义视图的更多信息，请参阅第 33 页的“定义移动设备应用程序中的视图”。

   您也可以创建完全以 ActionScript 编写的移动设备项目。请参阅第 10 页的“创建 ActionScript 移动设备项目”。

2 （可选）将内容添加至主应用程序文件的 ActionBar 中。

ActionBar 显示应用于应用程序或应用程序当前视图的内容和功能。在此添加要在应用程序的所有视图中显示的内容。请参阅第 52 页的 " 定义移动设备应用程序中的导航控件、标题控件和操作控件 "。

3 布置应用程序的初始视图的内容。

在设计模式或源代码模式下，使用 Flash Builder 将组件添加至视图中。

仅使用 Flex 支持用于移动设备开发的组件。在设计模式和源代码模式下，Flash Builder 将指导您如何使用支持的组件。请参阅第 21 页的 " 用户界面和布局 "。

在视图中，将内容添加至仅在该视图中可见的 ActionBar 中。

4 （可选）添加应用程序中要包括的任何其它视图。

在 Flash Builder 包资源管理器中，从项目中视图包的上下文菜单中，选择 " 新建 MXML 组件 "。新建 MXML 组件向导将指导您如何创建视图。

有关视图的更多信息，请参阅第 33 页的 " 定义移动设备应用程序中的视图 "。

5 （可选）为 List 组件添加针对移动设备优化的项呈示器。

Adobe 提供 IconItemRenderer，这是一个与移动设备应用程序结合使用的、基于 ActionScript 的项呈示器。请参阅 Using a mobile item renderer with a Spark list-based control。

6 配置启动配置以运行和调试应用程序。

可以在桌面或设备上运行或调试应用程序。

需要启动配置才可从 Flash Builder 运行或调试应用程序。首次运行或调试移动设备应用程序时， Flash Builder 将提示您配置启动配置。

在设备上运行或调试移动设备应用程序时， Flash Builder 将在设备上安装应用程序。

请参阅第 153 页的 " 测试和调试 "。

7 将应用程序导出为安装程序包。

使用 " 导出发行版 " 创建可以安装在移动设备设备上的包。 Flash Builder 根据选择以用于进行导出的平台创建包。请参阅第 160 页的 " 导出用于发行的 Android APK 包 "。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了以下对您有帮助的视频教程：

•Create a Flex mobile application with multiple views

• Create a Flex mobile application using a Spark-based List control

在 Flash Builder 中创建 iOS 应用程序

以下是用于为 Apple iOS 平台创建移动设备应用程序的常规工作流程。

1 在开始创建应用程序之前，请确保遵循第 17 页的 " 使用 Flash Builder 的 Apple iOS 开发过程 " 中的步骤。

2 在 Flash Builder 中，选择 " 文件 ">" 新建 ">"Flex 移动设备项目 "。

将 Apple iOS 选作目标平台，然后设定移动设备项目设置。

按照新建项目向导中的提示操作，就像完成 Flash Builder 中任何其它项目构建向导一样。有关更多信息，请参阅第 8 页的 " 创建应用程序 "。

您也可以创建完全以 ActionScript 编写的移动设备项目。有关更多信息，请参阅创建 ActionScript 移动设备项目。

3 配置启动配置以运行和调试应用程序。可以在桌面或已连接的设备上运行或调试应用程序。

有关更多信息，请参阅第 156 页的 " 在 Apple iOS 设备上调试应用程序 "。

4 将应用程序导出到 Apple App Store 或在设备上部署 iOS 包应用程序 (IPA)。

有关更多信息，请参阅第 161 页的 " 导出用于发行的 Apple iOS 包 " 和第 158 页的 " 在 Apple iOS 设备上安装应用程序 "。

更多帮助主题

Beginning a Mobile Application （视频）

在 Flash Builder 中创建 BlackBerry Tablet OS 应用程序

Flash Builder 包括一款来自 Research In Motion (RIM) 的插件，供您创建并打包适用于 BlackBerry® Tablet OS 的 Flex 和 ActionScript 应用程序。

创建应用程序

下面是创建 BlackBerry Tablet OS 应用程序的常规工作流程。

1 在开始创建移动设备应用程序之前，首先安装 BlackBerry Tablet OS SDK for AIR，可从 BlackBerry Tablet OS 应用程序开发网站获取。

BlackBerry Tablet OS SDK for AIR 提供了创建基于 AIR 的 Flex 和 ActionScript 应用程序所需的 API。

有关安装 BlackBerry Tablet OS SDK 的更多信息，请参阅 BlackBerry Tablet OS 快速入门指南。

2 要创建基于 Flex 的 AIR 应用程序，请在 Flash Builder 中选择 " 文件 ">" 新建 ">"Flex 移动设备项目 "。

请遵循新建项目向导中的提示，就像 Flash Builder 中其它任何 AIR 项目一样。确保选择 BlackBerry Tablet OS 作为目标平台。

有关更多信息，请参阅 Flex 移动设备项目。

3 要创建基于 ActionScript 的 AIR 应用程序，请在 Flash Builder 中选择 " 文件 ">" 新建 ">"ActionScript 移动设备项目 "。

请遵循新建项目向导中的提示，就像 Flash Builder 中其它任何 AIR 项目一样。确保选择 BlackBerry Tablet OS 作为目标平台。

有关更多信息，请参阅创建 ActionScript 移动设备项目。

签名、打包和部署应用程序

有关签名、打包和部署应用程序的信息，请参阅 RIM 提供的 BlackBerry Tablet OS SDK for Adobe AIR Development Guide。

您可以从 Adobe Developer Connection 找到 Adobe 和 RIM 提供的有关 BlackBerry Tablet OS 开发的其它资源。

创建 ActionScript 移动设备项目

使用 Flash Builder 创建 ActionScript 移动设备应用程序。您所创建的应用程序基于 Adobe AIR API。

1 选择 " 文件 ">" 新建 ">"ActionScript 移动设备项目 "。

2 输入项目名称和位置。默认位置为当前的工作空间。

3 使用支持移动设备应用程序开发的默认 Flex 4.6 SDK。

单击 " 下一步 "。

4 选择应用程序的目标平台，然后为每个平台指定移动设备项目设置。

有关移动设备项目设置的更多信息，请参阅第 12 页的 " 设置移动设备项目首选参数 "。

5 单击 " 完成 "，或者单击 " 下一步 " 以指定其它配置选项和构建路径。

有关项目配置选项和构建路径的更多信息，请参阅构建路径、本机扩展和其它项目配置选项。

使用本机扩展

通过本机扩展，可以将本机平台功能引入到移动设备应用程序中。

本机扩展包含 ActionScript 类和本机代码。通过本机代码实现，可以访问使用纯 ActionScript 类不能访问的特定设备功能。例如，访问设备的振动功能。

可以将本机代码实现定义为在 AIR 运行时外部执行的代码。在扩展中定义特定于平台的 ActionScript 类和本机代码实现。ActionScript 扩展类使用 ActionScript 类 ExtensionContext 访问数据和与本机代码交换数据。

扩展特定于设备的硬件平台。您可以创建特定于平台的多个扩展，也可以创建适用于多个平台的单个扩展。例如，您可以创建一个同时适用于 Android 和 iOS 平台的本机扩展。以下移动设备支持本机扩展：

- 运行 Android 2.2 或更高版本的 Android 设备
- 运行 iOS 4.0 或更高版本的 iOS 设备

有关创建跨平台本机扩展的详细信息，请参阅针对 Adobe AIR 开发本机扩展。

有关本机扩展示例的集合 （由 Adobe 和社区贡献），请参阅 Native extensions for Adobe AIR。

将本机扩展打包

要向应用程序开发人员提供本机扩展，您需要通过以下步骤将所有必需的文件打包到一个 ActionScript 本机扩展 (ANE) 中：

1 将扩展的 ActionScript 库构建到 SWC 文件中。

2 构建扩展的本机库。如果扩展必须支持多个平台，则为每个目标平台构建一个库。

3 为扩展创建签名证书。如果扩展没有签名， Flash Builder 会在您将扩展添加到项目中时显示警告。

4 创建扩展描述符文件。

5 包含用于扩展的任何外部资源，例如图像。

6 使用 Air Developer Tool 创建扩展包。有关更多信息，请参阅 AIR 文档。

有关打包 ActionScript 扩展的详细信息，请参阅针对 Adobe AIR 开发本机扩展。

将 本机扩展添加到项目中

您可以在项目构建路径中包含 ActionScript 本机扩展 (ANE) 文件，其方式与包含 SWC 文件相同。

1 在 Flash Builder 中，当您创建 Flex 移动设备项目时，请在 " 构建路径 " 设置页中选择 " 本机扩展 " 选项卡。

还可以通过选择 "Flex 构建路径 " 从 " 项目属性 " 对话框中添加扩展。

2 浏览至 ANE 文件或包含要添加到项目中的 ANE 文件的文件夹。添加 ANE 文件时，默认情况下扩展 ID 添加到项目的应用程序描述符文件 （项目名称 -app.xml）中。

在以下情况下， Flash Builder 为添加的扩展显示一个错误符号：

- 扩展的 AIR 运行时版本高于应用程序的运行时版本。
- 扩展并不包含应用程序所适用的所有选定平台。

注：您可以创建适用于多个平台的 ActionScript 本机扩展。要在开发计算机上使用 AIR 模拟器测试包含该 ANE 文件的应用程序，请确保 ANE 文件支持该计算机的平台。例如，要在 Windows 上使用 AIR 模拟器测试应用程序，应确保 ANE 文件支持 Windows。

在应用程序包中包含 ActionScript 本机扩展

使用导出发行版功能导出移动设备应用程序时，默认情况下项目中使用的扩展包含在应用程序包内。

要更改默认选择，请执行以下步骤：

1 在 " 导出发行版 " 对话框中，在 " 包设置 " 下，选择 " 本机扩展 " 选项卡。

2 将列出项目中引用的 ActionScript 本机扩展文件，并指明项目中是否使用了 ANE 文件。

   如果在项目中使用了 ANE 文件，默认情况下，在应用程序包中已选中该文件。

   如果项目中包含 ANE 文件但未使用，则编译器无法识别该 ANE 文件。因此，应用程序包中也不会包含该文件。要在应用程序包中包含 ANE 文件，请执行以下操作：

   a 在 " 项目属性 " 对话框中，选择 "Flex 构建打包 " 和所需的平台。

   b 选择要在应用程序包中包含的扩展。

支持 iOS5 本机扩展

要打包使用 iOS5 SDK 功能的本机扩展， AIR Developer Tool (ADT) 需要知道 iOS5 SDK 的位置。

在 Mac OS 上， Flash Builder 允许您使用 " 包设置 " 对话框选择 iOS5 SDK 的位置。选择 iOS SDK 位置之后，所选位置将通过 -platformsdk ADT 命令传递。

注：此功能目前在 Windows 上不受支持。

有关更多信息，请参阅针对 Adobe AIR 开发本机扩展。

设置移动设备项目首选参数

设置设备配置

Flash Builder 使用设备配置在 " 设计视图 " 中显示设备屏幕大小预览，或者在桌面上使用 AIR Debug Launcher (ADL) 启动应用程序。请参阅第 153 页的 " 配置桌面预览的设备信息 "。

要设置设备配置，请打开 " 首选项 " 并选择 "Flash Builder">" 设备配置 "。

Flash Builder 提供多种默认设备配置。您可以添加、编辑或删除其它设备配置。您无法修改 Flash Builder 提供的默认配置。

单击 " 恢复默认值 " 按钮可以恢复默认设备配置，但不会删除任何已经添加的配置。此外，如果添加的设备配置与某个默认值同名，则 Flash Builder 将使用默认设置重写添加的配置。

设备配置包含以下属性：

属性	说明
设备名称	设备的唯一名称。
平台	设备平台。从受支持平台列表中选择平台。
全屏大小	设备屏幕的宽度和高度。
可用屏幕大小	设备上应用程序的标准大小。此大小即为应用程序在以非全屏模式启动后的预期大小 （考虑系统镶边，例如状态栏）。
每英寸像素数	设备屏幕上的每英寸像素数。

选择目标平台

Flash Builder 支持基于应用程序类型的目标平台。

要选择平台，请打开 " 首选项 " 并选择 "Flash Builder">" 目标平台 "。

对于所有第三方插件，请参阅相关文档。

选择应用程序模板

创建移动设备应用程序时，可以选择以下应用程序模板：

空白 使用 Spark Application 标签作为基本应用程序元素。

如果要创建不使用标准视图导航的自定义应用程序，请使用此选项。

基于视图的应用程序 使用 Spark ViewNavigatorApplication 标签作为基本应用程序元素，来创建具有单一视图的应用程序。

可以指定初始视图的名称。

选项卡式应用程序 使用 Spark TabbedViewNavigatorApplication 标签作为基本应用程序元素，来创建基于选项卡的应用程序。

要添加选项卡，请输入选项卡的名称，然后单击 " 添加 "。可以通过单击 " 向上 " 和 " 向下 " 来更改选项卡的顺序。要从应用程序中删除选项卡，请选择选项卡并单击 " 删除 "。

视图名称是后面追加 "View" 的选项卡名称。例如，如果将选项卡命名为 "FirstTab"，则 Flash Builder 将生成名为 "FirstTabView" 的视图。

对于每个创建的选项卡，都会在 "view" 包中生成一个新的 MXML 文件。

注：在 "Flex 移动设备项目 " 向导中不能配置该包的名称。

MXML 文件按以下规则生成：

- 如果选项卡名称是有效的 ActionScript 类名称，Flash Builder 在生成 MXML 文件时将使用后面追加 "View" 的选项卡名称。
- 如果选项卡名称不是有效的类名称， Flash Builder 则从选项卡名称中删除无效字符并插入有效起始字符。如果修改后的名称不允许使用，则 Flash Builder 将 MXML 文件名更改为 "ViewN"，其中 N 代表视图位置，从 N=1 开始。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了关于使用选项卡式应用程序模板的视频教程。

选择移动设备应用程序权限

创建移动设备应用程序时，可以指定或更改目标平台的默认权限。权限在编译时指定，运行期间不能更改。

首先选择目标平台，然后根据需要针对每个平台设置权限。以后可以在应用程序描述符 XML 文件中编辑权限。

第三方插件可以为 Flex 和 ActionScript 项目提供额外的平台支持。有关各平台特定的权限，请参阅设备的相关文档。

针对 Google Android 平台的权限

对于 Google Android 平台，可以设置以下权限：

INTERNET 允许进行网络请求和远程调试

默认情况下，将选择 INTERNET 权限。如果取消选择此权限，则无法调试设备上的应用程序。

WRITE_EXTERNAL_STORAGE 允许写入外部设备

选择此权限可允许应用程序写入设备上的外部内存卡。

READ_PHONE_STATE 来电期间设置静音

选择此权限可允许应用程序将来电的音频设为静音。例如，如果应用程序在后台播放音频，可以选择此权限。

ACCESS_FINE_LOCATION 允许访问 GPS 位置

选择此权限可允许应用程序使用 Geolocation 类访问 GPS 数据。

DISABLE_KEYGUARD 和 WAKE_LOCK 在设备上禁止休眠模式

选择此权限可防止设备在使用 SystemIdleMode 类设置的情况下进入休眠状态。

CAMERA 允许访问摄像头

选择此权限可允许应用程序访问摄像机。

RECORD_AUDIO 允许访问麦克风

选择此权限可允许应用程序访问麦克风。

ACCESS_NETWORK_STATE 和 ACCESS_WIFI_STATE 允许访问与设备关联的网络接口相关信息

选择此权限可允许应用程序使用 NetworkInfo 类访问网络信息。

有关设置移动设备应用程序属性的更多信息，请参阅 Adobe AIR 文档。

针对 Apple iOS 平台的权限

Apple iOS 平台会进行运行时权限验证，而不使用预定义权限。也就是说，如果应用程序要访问 Apple iOS 平台中需要用户权限的特定功能，将会出现弹出窗口要求验证权限。

选择平台设置

通过平台设置，可以选择目标设备系列。根据所选平台，可以选择目标设备或目标设备系列。可以选择特定设备或者平台支持的所有设备。

第三方插件可以为 Flex 和 ActionScript 项目提供额外的平台支持。有关各平台特定的设置，请参阅设备的相关文档。

针对 Google Android 平台的平台设置

没有针对 Google Android 平台的设置。

针对 Apple iOS 平台的平台设置

对于 Flex 移动设备项目或 ActionScript 移动设备项目，可以指定适用于 Apple iOS 平台的以下目标设备：

iPhone/iPod Touch 使用此目标系列的应用程序仅与 Apple App Store 中的 iPhone 和 iPod Touch 设备兼容。

iPad 使用此目标系列的应用程序仅与 Apple App Store 中的 iPad 设备兼容。

全部 使用此目标系列的应用程序与 Apple App Store 中的 iPhone、 iPod Touch 和 iPad 设备都兼容。此选项为默认设置。

选择应用程序设置

自动重定向 在用户旋转设备时，旋转应用程序。如果未启用此设置，则应用程序始终以固定方向显示。

全屏 在设备上以全屏模式显示应用程序。如果启用此设置，应用程序上方将不会出现设备的状态栏。应用程序将填满整个屏幕。

如果应用程序要在具有不同屏幕密度的多种设备上运行，请选择 " 对不同屏幕密度自动缩放应用程序 "。如果选择此选项，将自动缩放应用程序并在必要时处理设备的密度变更。请参阅第 15 页的 " 设置应用程序缩放 "。

设置应用程序缩放

使用移动设备应用程序缩放功能可以构建一个与屏幕大小和密度不同的多种设备都兼容的移动设备应用程序。

移动设备屏幕具有不同的屏幕密度，或称 DPI（每英寸点数）。根据目标设备的屏幕密度，可以将 DPI 值指定为 160、240 或 320。如果启用自动缩放， Flex 将根据每个设备的屏幕密度来优化应用程序的显示方式。

例如，假设将目标 DPI 值指定为 160，并启用自动缩放。在 DPI 值为 320 的设备上运行应用程序时，Flex 将按缩放因子 2 来自动缩放应用程序。也就是说， Flex 将所有内容放大为 200%。

要指定目标 DPI 值，请在主应用程序文件中将其设置为 <s:ViewNavigatorApplication> 标签或 <s:TabbedViewNavigatorApplication> 标签的 applicationDPI 属性：

```
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.HomeView"
    applicationDPI="160">
```

如果选择不自动缩放应用程序，则必须根据需要手动处理屏幕布局的密度变化。但 Flex 可以根据每种设备的密度调整外观。

有关创建与密度无关的移动设备应用程序的更多信息，请参阅第 108 页的 " 在一个移动设备应用程序中支持多个屏幕大小和 DPI 值 "。

连接 Google Android 设备

可以将 Google Android 设备连接到开发计算机，用以在 Android 设备上预览或调试应用程序。

支持的 Android 设备

Flex 移动设备项目和 ActionScript 移动设备项目要求使用 AIR 2.6 或更高版本的 AIR。您只能在支持 AIR 2.6 或更高版本的物理设备上运行或调试移动设备项目。

您可以在运行 Android 2.2 或更高版本的受支持 Android 设备上安装 AIR 2.6。有关受支持设备的列表，请参阅 http://www.adobe.com/flashplatform/certified_devices/。另外，查看移动系统要求中有关在 Android 设备上运行 Adobe AIR 的最低系统要求。

配置 Android 设备

要运行或调试 Android 设备上的 Flex 移动设备应用程序，请按如下指定启用 USB 调试：

1 在设备上，执行以下步骤以确保启用 USB 调试：

 a 轻敲 " 主页 " 按钮以显示主屏幕。

 b 转到 " 设置 "，然后选择 " 应用程序 ">" 开发 "。

 c 启用 USB 调试。

2 使用 USB 电缆将设备连接到计算机。

3 将屏幕顶部的通知区域向下拉。您将看到 “USB 已连接 ” 或 “USB 连接 ”。

a 轻敲 “USB 已连接 ” 或 “USB 连接 ”。

b 如果出现一组包含 “ 仅充电 ” 模式的选项，则选择 “ 仅充电 ” 并轻敲 “ 确定 ”。

c 如果您查看按钮以关闭大容量存贮模式，请单击按钮以关闭大容量存贮。

4 （仅限 Windows）为设备安装合适的 USB 驱动程序。请参阅第 16 页的 “ 安装 Android 设备的 USB 设备驱动程序 (Windows)”。

5 将屏幕顶部的通知区域向下拉。

如果 “USB 调试 ” 不作为条目出现，则如以上步骤 3 所述检查 USB 模式。确认 USB 模式未设置为 PC 模式。

注： 调试时需要其它配置。请参阅第 154 页的 “ 在设备上测试和调试移动设备应用程序 ”。

安装 Android 设备的 USB 设备驱动程序 (Windows)

设备驱动程序和配置

Windows 平台需要使用 USB 驱动程序的安装才可将 Android 设备连接到您的开发计算机。 Flash Builder 提供多个 Android 设备的设备驱动程序和配置。

这些设备驱动程序配置列在 android_winusb.inf 中。安装设备驱动程序时，Windows 设备管理器访问该文件。Flash Builder 在以下位置安装了 android_winusb.inf：

```
<Adobe Flash Builder 4.6 Home>\utilities\drivers\android\android_winusb.inf
```

有关受支持设备的完整列表，请参阅 Certified devices。对于未列出的 Android 设备，可以使用 USB 驱动程序更新 android_winusb.inf。请参阅第 16 页的 “ 添加 Android USB 设备驱动程序配置 ”。

安装 **USB** 设备驱动程序

1 将 Android 设备连接至计算机的 USB 端口。

2 转到以下位置：

<Flash Builder>/utilities/drivers/android/

使用 “Windows 发现新硬件 ” 向导或 “Windows 设备管理器 ” 来安装 USB 驱动程序。

重要说明：如果 Windows 仍无法识别您的设备，则需要安装设备制造商提供的适用 USB 驱动程序。OEM USB Drivers 中列出了几家设备制造商的 Web 站点链接，您可以从这些站点下载设备适用的 USB 驱动程序。

添加 Android USB 设备驱动程序配置

如果您的受支持 Android 设备未在第 16 页的 “ 安装 Android 设备的 USB 设备驱动程序 (Windows)” 中列出，请更新 android_winusb.inf 文件以包含该设备。

1 将设备插入到计算机的 USB 端口中。 Windows 通知您无法找到驱动程序。

2 使用 “Windows 设备管理器 ” 打开设备属性的 “ 详细信息 ” 选项卡。

3 选择 “ 硬件 ID” 属性以查看硬件 ID。

4 打开文本编辑器中的 android_winusb.inf。在以下位置找到 android_winusb.inf：

```
<Adobe Flash Builder 4.6 Home>\utilities\drivers\android\android_winusb.inf
```

5 记录文件中适用于您所用体系结构的列表：[Google.NTx86] 或 [Google.NTamd64]。该列表包含描述性的注释，以及带有硬件 ID 的一个或多个行，如下所示：

```
. . .
[Google.NTx86]
; HTC Dream
%CompositeAdbInterface% = USB_Install, USB\VID_0BB4&PID_0C02&MI_01
. . .
```

6 复制并粘贴注释和硬件列表。对于您要添加的设备驱动程序，请如下所示编辑列表：

a 对于注释，请指定设备名称。

b 使用上述第 3 步中确定的硬件 ID 替换硬件 ID。

例如：

```
. . .
[Google.NTx86]
; NEW ANDROID DEVICE
%CompositeAdbInterface%     = USB_Install, NEW HARDWARE ID
. . .
```

7 使用 "Windows 设备管理器 " 安装设备，如以上第 16 页的 " 安装 Android 设备的 USB 设备驱动程序 (Windows)" 中所述。

在安装期间， Windows 会显示一条警告，提示驱动程序来自未知的发行商。但是，驱动程序允许 Flash Builder 访问您的设备。

使用 Flash Builder 的 Apple iOS 开发过程

在使用 Flash Builder 开发 iOS 应用程序之前，必须了解 iOS 开发过程和如何从 Apple 获得必需的证书。

iOS 开发和部署过程概述

下表提供了 iOS 开发过程步骤的快速列表、如何获得必需的证书以及每个步骤的先决条件。

有关其中每个步骤的详细信息，请参阅第 18 页的 " 构建、调试或部署 iOS 应用程序前的准备工作 "。

步骤编号	步骤	位置	先决条件
1.	加入 Apple 开发者计划。	Apple Developer 站点	无
2.	注册 iOS 设备的唯一设备标识符 (UDID)。	iOS 配置门户	Apple 开发者 ID （第 1 步）
3.	生成证书签名请求 (CSR) 文件 (*.certSigningRequest)。	• 在 Mac OS 中，使用 Keychain Access 程序 • 在 Windows 中，使用 OpenSSL	无
4.	生成 iOS 开发者 / 分发证书 (*.cer)。	iOS 配置门户	• Apple 开发者 ID （第 1 步） • CSR 文件 （第 3 步）
5.	将 iOS 开发者 / 分发证书转换为 P12 格式。	• 在 Mac OS 中，使用 Keychain Access 程序 • 在 Windows 中，使用 OpenSSL	• Apple 开发者 ID （第 1 步） • iOS 开发者 / 分发证书 （第 4 步）
6.	生成应用程序 ID。	iOS 配置门户	Apple 开发者 ID （第 1 步）

步骤编号	步骤	位置	先决条件
7.	生成配置概要文件 (*.mobileprovision)	iOS 配置门户	• Apple 开发者 ID （第 1 步） • iOS 设备的 UDID （第 2 步） • 应用程序 ID （第 6 步）
8.	构建应用程序。	Flash Builder	• Apple 开发者 ID （第 1 步） • P12 开发者 / 分发证书 （第 5 步） • 应用程序 ID （第 6 步）
9.	部署应用程序。	iTunes	• 配置概要文件 （第 7 步） • 应用程序包 （第 8 步）

构建、调试或部署 iOS 应用程序前的准备工作

在使用 Flash Builder 构建 iOS 应用程序并将该应用程序部署在 iOS 设备上或提交到 Apple App Store 之前，请执行以下步骤：

1 加入 Apple iOS 开发者计划。

您可以使用现有的 Apple ID 登录或创建一个 Apple ID。 Apple 开发者注册向导将指导您完成必要的步骤。

2 注册设备的唯一设备标识符 (UDID)。

仅在您将应用程序部署到 iOS 设备而不是 Apple App Store 中时，该步骤才适用。如果您要在多个 iOS 设备上部署应用程序，请注册每个设备的 UDID。

获得 **iOS** 设备的 **UDID**

a 将 iOS 设备连接到开发计算机并启动 iTunes。连接的 iOS 设备将显示在 iTunes 中的 " 设备 " 部分下。

b 单击设备名称显示 iOS 设备的摘要。

c 在 " 摘要 " 选项卡中，单击 " 序列号 " 可显示 iOS 设备的包含 40 个字符的 UDID。

您可以使用键盘快捷键 Ctrl+C (Windows) 或 Cmd+C (Mac) 从 iTunes 复制该 UDID。

注册设备的 **UDID**

使用 Apple ID 登录 iOS 配置门户并注册该设备的 UDID。

3 生成证书签名请求 (CSR) 文件 (*.certSigningRequest)。

将生成 CSR 以获得 iOS 开发者 / 分发证书。可以通过使用 Mac 上的 Keychain Access 或 Windows 上的 OpenSSL 生成 CSR。生成 CSR 时，只需提供用户名和电子邮件地址；无需提供有关应用程序或设备的任何信息。

生成 CSR 会创建公钥、私钥以及 *.certSigningRequest 文件。公钥包含在 CSR 中，私钥用于为请求签名。

有关生成 CSR 的更多信息，请参阅生成证书签名请求。

4 根据需要生成 iOS 开发者证书或 iOS 分发证书 (*.cer)。

注：要将应用程序部署到设备中，需要开发者证书。要将应用程序部署到 Apple App Store 中，需要分发证书。

生成 **iOS** 开发者证书

a 使用 Apple ID 登录 iOS 配置门户，然后选择 " 开发 " 选项卡。

b 单击 " 请求证书 "，然后浏览至您在计算机上生成并保存 （第 3 步）的 CSR 文件。

c 选中 CSR 文件并单击 " 提交 "。

d 在 " 证书 " 页面中，单击 " 下载 "。

e 保存已下载文件 (*.developer_identity.cer)。

生成 **iOS** 分发证书

f 使用 Apple ID 登录 iOS 配置门户，然后选择 " 分发 " 选项卡

g 单击 " 请求证书 "，然后浏览至您在计算机上生成并保存 （第 3 步）的 CSR 文件。

h 选中 CSR 文件并单击 " 提交 "。

i 在 " 证书 " 页面中，单击 " 下载 "。

j 保存已下载文件 (*.distribution_identity.cer)。

5 将 iOS 开发者证书或 iOS 分发证书转换为 P12 文件格式 (*.p12)。

将 iOS 开发者证书或 iOS 分发证书转换为 P12 格式以便 Flash Builder 可以为 iOS 应用程序进行数字签名。转换为 P12 格式将使 iOS 开发者 / 分发证书与关联的私钥组合在一个文件中。

注 如果您使用 AIR Debug Launcher (ADL) 测试桌面上的应用程序，则无需将 iOS 开发者 / 分发证书转换为 P12 格式。

使用 Mac 上的 Keychain Access 或 Windows 上的 OpenSSL 生成个人信息交换 (*.p12) 文件。有关更多信息，请参阅将开发人员证书转换为 P12 文件。

6 通过执行下列操作生成应用程序 ID：

a 使用 Apple ID 登录 iOS 配置门户。

b 转至 " 应用程序 ID" 页面，然后单击 " 新建应用程序 ID"。

c 在 " 管理 " 选项卡中，输入应用程序的说明，生成新的捆绑种子 ID，然后输入捆绑标识符。

每个应用程序都有一个唯一的应用程序 ID，您可以在应用程序描述符 XML 文件中指定该 ID。应用程序 ID 包含一个 Apple 提供的 10 字符 " 捆绑种子 ID" 和一个您指定的 " 捆绑标识符 " 后缀。您指定的 " 捆绑标识符 " 必须与应用程序描述符文件中的应用程序 ID 匹配。例如，如果应用程序 ID 是 com.myDomain.*，则应用程序描述符文件中的 ID 必须以 com.myDomain 开始。

重要说明：通配符捆绑标识符有助于开发和测试 iOS 应用程序，但不能用于将应用程序部署到 Apple App Store。

7 生成开发者配置概要文件或分发配置概要文件 (*.mobileprovision)。

注：要将应用程序部署到设备中，需要开发者配置概要文件。要将应用程序部署到 Apple App Store 中，需要分发配置概要文件。使用分发配置概要文件为应用程序签名。

生成开发者配置概要文件

a 使用 Apple ID 登录 iOS 配置门户。

b 转至 " 证书 ">" 配置 "，然后单击 " 新建概要文件 "。

c 输入概要文件名称，选择 iOS 开发者证书、应用程序 ID 以及要在其中安装应用程序的 UDID。

d 单击 " 提交 "。

e 下载生成的开发者配置概要文件 (*.mobileprovision) 并将其保存在计算机上。

生成分发配置概要文件

f 使用 Apple ID 登录 iOS 配置门户。

g 转至 " 证书 ">" 配置 "，然后单击 " 新建概要文件 "。

h 输入概要文件名称，选择 iOS 分发证书和应用程序 ID。如果您要在部署之前测试应用程序，请指定要在其中执行测试的设备的 UDID。

i 单击 " 提交 "。

j 下载生成的配置概要文件 (*.mobileprovision) 并将其保存在计算机上。

更多帮助主题

第 9 页的 " 在 Flash Builder 中创建 iOS 应用程序 "

在测试、调试或安装 iOS 应用程序时选择的文件

要运行、调试或安装在 iOS 设备上测试的应用程序，请在 " 运行 / 调试配置 " 对话框中选择以下文件：

- P12 格式的 iOS 开发者证书 （第 5 步）
- 包含应用程序 ID 的应用程序描述符 XML 文件 （第 6 步）
- 开发者配置概要文件 （第 7 步）

有关更多信息，请参阅第 156 页的 " 在 Apple iOS 设备上调试应用程序 " 和第 158 页的 " 在 Apple iOS 设备上安装应用程序 "。

将应用程序部署到 Apple App Store 时选择的文件

要将应用程序部署到 Apple App Store 中，请在 " 导出发行版 " 对话框中选择 " 包类型 " 作为 Apple App Store 的最终发行包，然后选择以下文件：

- P12 格式的 iOS 分发证书 （第 5 步）
- 包含应用程序 ID 的应用程序描述符 XML 文件 （第 6 步）

  注：您不能在将应用程序提交到 Apple App Store 时使用通配符应用程序 ID。
- 分发配置概要文件 （第 7 步）

有关更多信息，请参阅第 161 页的 " 导出用于发行的 Apple iOS 包 "。

第 3 章：用户界面和布局

移动设备应用程序的布局

使用视图和区域布置移动设备应用程序

移动设备应用程序由一个或多个屏幕或视图组成。例如，移动设备应用程序可能有三个视图：

1 用于添加联系人信息的主视图

2 包含现有联系人列表的联系人视图

3 用于搜索联系人列表的搜索视图

简单移动设备应用程序

下图显示的是在 Flex 中构建的一个简单移动设备应用程序的主屏幕：

A. ActionBar 控件 **B.** 内容区域

上图显示了移动设备应用程序的主区域：

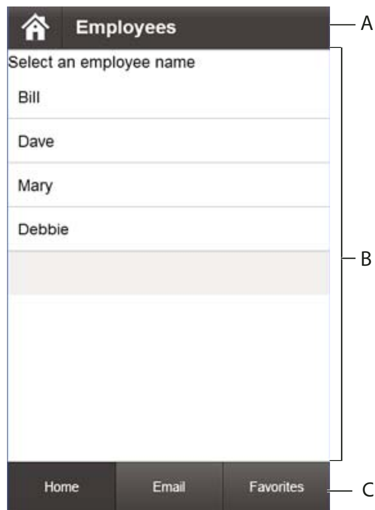
ActionBar 控件 通过 ActionBar 控件，可以显示应用程序当前状态的上下文信息。此信息包括标题区域、应用程序导航控件的区域，以及操作执行控件的区域。可以在 ActionBar 控件中添加应用于整个应用程序的全局内容，也可以添加特定于单个视图的项目。

内容区域 内容区域显示组成应用程序的各屏幕或视图。用户通过应用程序内置组件和移动设备的输入控件来导航应用程序视图。

具有多个部分的移动设备应用程序

在更为复杂的应用程序中，可以定义多个应用程序区域或部分。例如，应用程序可以分为联系人部分、电子邮件部分、收藏夹部分和其他部分。应用程序的每个部分包含一个或多个视图。各部分之间可以共享视图，这样便无需多次定义同一个视图。

下图显示了一个移动设备应用程序，其应用程序窗口的底部带有一个选项卡栏：



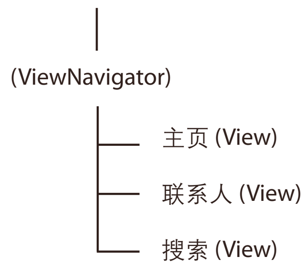
A. ActionBar 控件 **B.** 内容区域 **C.** 选项卡栏

Flex 使用 ButtonBarBase 控件来实现选项卡栏。选项卡栏的每个按钮对应不同的部分。选择选项卡栏中的按钮可以切换当前部分。

应用程序的每个部分定义各自的 ActionBar。因此，选项卡栏全局适用于整个应用程序，而 ActionBar 特定于每个部分。

简单移动设备应用程序的布局

下图显示的是简单移动设备应用程序的体系结构：



图中显示的应用程序由四个文件组成。移动设备应用程序包含一个主应用程序文件和对应于各视图的文件 （每个视图一个文件）。其中没有专用于 ViewNavigator 的单独文件；该控件由 ViewNavigatorApplication 容器创建。

注：尽管此图显示了应用程序的体系结构，但这不代表运行时的应用程序。在运行时，只有一个视图处于活动状态并驻留在内存中。有关更多信息，请参阅第 25 页的 " 在移动设备应用程序的视图间导航 "。

移动设备应用程序中使用的类

使用以下类定义移动设备应用程序：

类	说明
ViewNavigatorApplication	定义主应用程序文件。 ViewNavigatorApplication 容器不接受任何子代。
ViewNavigator	控制应用程序视图之间的导航。 ViewNavigator 还可以创建 ActionBar 控件。 ViewNavigatorApplication 容器自动为整个应用程序创建一个 ViewNavigator 容器。使用 ViewNavigator 容器的方法可以在不同的视图之间切换。
View	定义应用程序的视图，其中每个视图在单独的 MXML 或 ActionScript 文件中定义。View 容器的实例代表应用程序的各个视图。应在单独的 MXML 或 ActionScript 文件中定义每个视图。

使用 ViewNavigatorApplication 容器可以定义主应用程序文件，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSectionSimple.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.HomeView">
</s:ViewNavigatorApplication>
```

ViewNavigatorApplication 容器自动创建一个用于定义 ActionBar 的 ViewNavigator 对象。可以使用 ViewNavigator 在应用程序的视图间导航。

在移动设备应用程序中添加 **View** 容器

每个移动设备应用程序至少带有一个视图。尽管主应用程序文件将创建 ViewNavigator，但不会定义应用程序中使用的任何视图。

应用程序中的每个视图对应于 ActionScript 或 MXML 文件中定义的一个 View 容器。每个 View 中包含一个 data 属性，用于指定与视图相关联的数据。用户导航应用程序时， View 可以使用 data 属性来彼此传递信息。

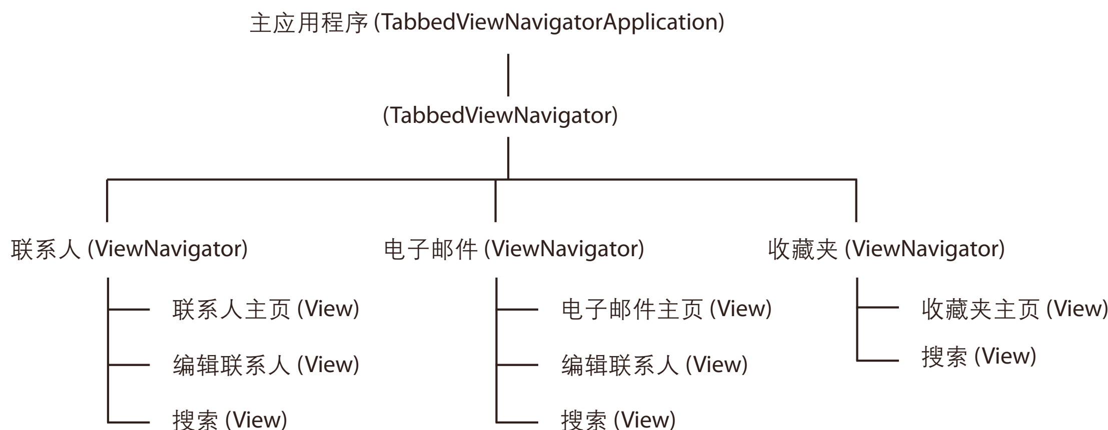
使用 ViewNavigatorApplication.firstView 属性可以指定用于定义应用程序第一个视图的文件。在上面的应用程序中， firstView 属性指定了 views.HomeView。以下示例显示的是定义该视图的 HomeView.mxml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\HomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Home">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>
    <s:Label text="The home screen"/>
</s:View>
```

博客 David Hassoun 发表了关于 ViewNavigator 基础知识的文章。

具有多个部分的移动设备应用程序的布局

移动设备应用程序可以将相关的视图收集到应用程序的不同部分中。例如，下图显示的是具有三个部分的移动设备应用程序的组织结构。



任何部分都可以使用任何 View。也就是说，视图并不从属于特定的部分。各部分只用于定义视图集合的排列和导航方法。在上图中，应用程序的每个部分中都包含 Search 视图。

在运行时，只有一个视图处于活动状态并驻留在内存中。有关更多信息，请参阅第 25 页的 " 在移动设备应用程序的视图间导航 "。

具有多个部分的移动设备应用程序中使用的类

下表列出用于创建具有多个部分的移动设备应用程序的类：

类	说明
TabbedViewNavigatorApplication	定义主应用程序文件。TabbedViewNavigatorApplication 容器唯一允许的子代是 ViewNavigator。应当为应用程序的每个部分定义一个 ViewNavigator。
TabbedViewNavigator	控制应用程序各组成部分之间的导航。 TabbedViewNavigatorApplication 容器自动为整个应用程序创建一个 TabbedViewNavigator 容器。TabbedViewNavigator 容器用于创建选项卡栏，通过选项卡栏可以在各部分之间实现导航。
ViewNavigator	为每个部分定义一个 ViewNavigator 容器。ViewNavigator 控制着构成部分的各视图之间的导航。它还会为部分创建 ActionBar 控件。
View	定义应用程序的视图。View 容器的实例代表应用程序的各个视图。应在单独的 MXML 或 ActionScript 文件中定义每个视图。

具有多个部分的应用程序包含一个主应用程序文件和一个定义每个视图的文件。使用 TabbedViewNavigatorApplication 容器可以定义主应用程序文件，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMultipleSectionsSimple.mxml -->
<s:TabbedViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <s:ViewNavigator label="Contacts" firstView="views.ContactsHome"/>
    <s:ViewNavigator label="Email" firstView="views.EmailHome"/>
    <s:ViewNavigator label="Favorites" firstView="views.FavoritesHome"/>
</s:TabbedViewNavigatorApplication>
```

在具有多个部分的应用程序中使用 **ViewNavigator**

TabbedViewNavigatorApplication 容器唯一允许的子代是 ViewNavigator。应用程序的每个部分对应于不同的 ViewNavigator 容器。

使用 ViewNavigator 容器可以在每个部分的各视图之间导航，它还会为该部分定义 ActionBar 控件。使用 ViewNavigator.firstView 属性可以指定用于定义该部分第一个视图的文件。

在具有多个部分的应用程序中使用 **TabbedViewNavigator**

TabbedViewNavigatorApplication 容器自动创建一个 TabbedViewNavigator 类型的容器。然后，TabbedViewNavigator 容器在应用程序的底部创建选项卡栏。您无需在应用程序中添加逻辑即可在各部分之间导航。

在移动设备应用程序的视图间导航

View 对象的堆栈可以控制移动设备应用程序中的导航。堆栈中最上方的 View 对象定义当前可见的视图。

ViewNavigator 容器负责维护该堆栈。要切换视图，请将新的 View 对象推送到堆栈中，或将当前 View 对象从堆栈中弹出。从堆栈中弹出当前可见的 View 对象时，会破坏该 View 对象，并使用户返回到堆栈中的上一个视图。

在具有多个部分的应用程序中，可以使用选项卡栏在各部分之间进行导航。由于每个部分由不同的 ViewNavigator 定义，因此切换各部分就相当于切换当前的 ViewNavigator 和堆栈。新 ViewNavigator 堆栈最上方的 View 对象将成为当前视图。

为节约内存，默认情况下 ViewNavigator 会确保每次只有一个视图驻留在内存中。但它会保留堆栈中上一个视图的数据。因此，当用户导航回上一个视图时，可以使用相应的数据重新实例化该视图。

注：View 容器会定义 destructionPolicy 属性。如果该属性设置为 auto （默认值）， ViewNavigator 会在视图处于非活动状态时将其破坏。如果该属性设置为 none，视图将在内存中缓存。

博客 Mark Lochrie 发表了关于 ViewNavigator 的文章。

ViewNavigator 导航方法

可以使用 ViewNavigator 类的以下方法来实现导航控制：

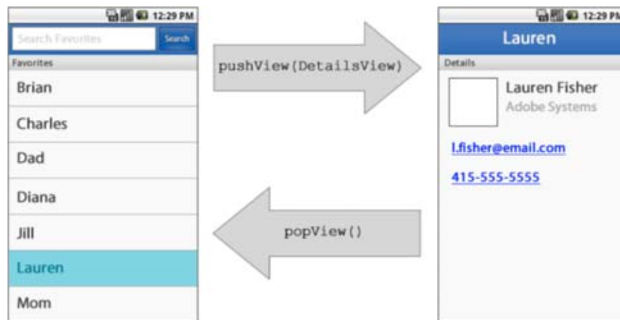
pushView() 将 View 对象推送到堆栈中。作为参数传递给 pushView() 的 View 将成为当前视图。

popView() 从导航堆栈中弹出当前 View 对象，并破坏该 View 对象。堆栈中的上一个 View 对象将成为当前视图。

popToFirstView() 从堆栈中弹出所有 View 对象并将这些对象破坏，但堆栈中的第一个 View 对象除外。堆栈中的第一个 View 对象将成为当前视图。

popAll() 清空 ViewNavigator 的堆栈，并破坏所有 View 对象。应用程序将显示空白视图。

下图中显示了两个视图。要切换当前视图，请使用 ViewNavigator.pushView() 方法将代表新视图的 View 对象推送到堆栈中。pushView() 方法将使 ViewNavigator 将显示画面切换为新的 View 对象。



通过推送和弹出 View 对象来切换视图。

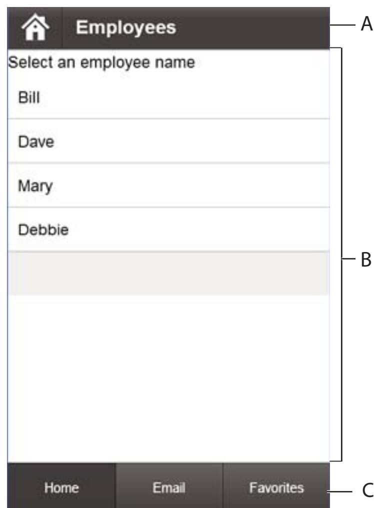
可以使用 ViewNavigator.popView() 方法从堆栈中删除当前 View 对象。 ViewNavigator 将显示画面恢复为堆栈中的上一个 View 对象。

注：移动设备应用程序中的大多数导航操作由移动设备本身进行控制。例如，在 Flex 中构建的移动设备应用程序将自动处理移动设备上的后退按钮。因此，无需在应用程序中添加对后退按钮的支持。当用户在移动设备上按下后退按钮时， Flex 将自动调用 popView() 方法来恢复上一个视图。

博客 David Hassoun 发表了关于在视图中管理数据的文章。

为具有多个部分的应用程序创建导航

在下图中，各个视图安排在多个部分中。每个部分由不同的 ViewNavigator 容器定义。每个部分包含一个或多个视图：



A. ActionBar **B.** 内容区域 **C.** 选项卡栏

要切换当前部分中的视图 （对应于当前的 ViewNavigator），请使用 pushView() 和 popView() 方法。

要切换当前部分，请使用选项卡栏。切换部分时，将切换到新部分的 ViewNavigator 容器。显示画面将改为显示当前位于新 ViewNavigator 堆栈最上方的 View 对象。

也还可以使用 TabbedViewNavigator.selectedIndex 属性以程序方式切换部分。此属性包含选定视图导航器的从 0 开始的索引。

在移动设备应用程序中处理用户输入

在移动设备应用程序中，用户输入的处理方法与桌面或浏览器应用程序不同。在针对 AIR 构建的桌面应用程序或针对 Flash Player 构建的浏览器应用程序中，主要的输入设备是鼠标和键盘。而对于移动设备而言，主要的输入设备是触摸屏。移动设备往往会配备某种类型的键盘，某些设备还包括五向输入法 （左、右、上、下及选择）。

mx.core.UIComponent 类可以定义 interactionMode 样式属性，用于为应用程序中所使用的输入类型配置组件。对于 Halo 和 Spark 主题，默认值为 mouse，表示主要的输入设备是鼠标。对于移动设备主题，默认值为 touch，表示主要的输入设备是触摸屏。

手机应用程序中的硬件按键支持

由 ViewNavigatorApplication 或 TabbedViewNavigatorApplication 容器定义的应用程序将对设备的后退和菜单硬件按键做出响应。当用户按下后退键时，应用程序将导航到上一个视图。如果没有上一个视图，将退出应用程序并显示设备的主屏幕。

当用户按下后退按钮时，应用程序的活动视图会收到 backKeyPressed 事件。可以通过在 backKeyPressed 事件的事件处理函数中调用 preventDefault() 来取消后退键的操作。

当用户按下菜单按钮时，将显示当前视图的 ViewMenu 容器 （如果已经定义）。 ViewMenu 容器定义 View 容器底部的菜单。每个 View 容器定义该视图专用的菜单。

当用户按下菜单键时，当前 View 容器将分派 menuKeyPressed 事件。要取消菜单按钮的操作，并阻止显示 ViewMenu，应在 menuKeyPressed 事件的事件处理函数中调用 preventDefault() 方法。

有关更多信息，请参阅第 61 页的 “ 定义移动设备应用程序中的菜单 ”。

处理移动设备应用程序中的硬件键盘事件

在 Flex 的内置移动设备应用程序中，可以检测到用户何时按下了移动设备上的硬件键。例如，在 Android 设备上，可以检测到用户何时按下了 “ 主页 ” 按钮、 “ 后退 ” 按钮或 “ 菜单 ” 按钮。

要检测用户何时按下了硬件键，为 KEY_UP 或 KEY_DOWN 事件创建一个事件处理函数。通常，按照 Application、ViewNavigatorApplication 或 TabbedViewNavigatorApplication 容器的定义将事件处理函数附加到应用程序对象。

Stage 对象定义应用程序的绘制区域。每个应用程序都有一个 Stage 对象。因此，应用程序容器实际上是 Stage 对象的子容器。

Stage.focus 属性指定当前具有键盘焦点的组件或包含 null （如果组件没有焦点）。具有键盘焦点的组件是在用户与键盘交互时接收事件通知的组件。因此，如果 Stage.focus 设置为应用程序对象，将调用该应用程序对象的事件处理函数。

在移动设备中，您的应用程序可以被其它应用程序中断。例如，移动设备可以在应用程序运行时接电话或用户可以切换到其它应用程序。用户切换回应用程序时， Stage.focus 属性设置为 null。因此，指定给应用程序对象的事件处理函数不对键盘做出响应。

由于 Stage.focus 属性在移动设备应用程序中可以为 null，因此将侦听 Stage 对象自身的键盘事件以确保应用程序可识别该事件。以下示例将键盘事件处理函数指定给 Stage 对象：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkHWEventHandler.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.SparkHWEventhandlerHomeView"
    applicationComplete="appCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            // Add the hardware key event handlers to the stage.
            protected function appCompleteHandler(event:FlexEvent):void {
                stage.addEventListener("keyDown", handleButtons, false,1);
                stage.addEventListener("keyUp", handleButtons, false, 1);
            }

            // Event handler to handle hardware keyboard keys.
            protected function handleButtons(event:KeyboardEvent):void
            {
                if (event.keyCode == Keyboard.HOME) {
                    // Handle Home button.
                }
                else if (event.keyCode == Keyboard.BACK) {
                    // Hanlde back button.
                }
            }
        ]]>
    </fx:Script>
</s:ViewNavigatorApplication>
```

在移动设备应用程序中处理鼠标和触控事件

AIR 生成不同的事件来表示不同类型的输入。这些事件包括以下类型:

鼠标事件 通过鼠标或触摸屏产生的用户交互而生成的事件。鼠标事件包括 mouseOver、 mouseDown 和 mouseUp。

触控事件 因检测到用户与设备的接触 (例如手指触碰触摸屏) 而在设备上产生的事件。触控事件包括 touchTap、touchOver 和 touchMove。当用户通过触摸屏与设备进行交互时，用户通常使用手指或触摸笔触摸屏幕。

手势事件 由多点触控交互 (例如同时用两根手指按压触摸屏) 产生的事件。手势事件包括 gesturePan、 gestureRotate 和 gestureZoom。例如，在某些设备上，可以使用捏合手势来缩小图片。

内置的鼠标事件支持功能

Flex 框架和 Flex 组件集内置了鼠标事件的支持功能，但不支持触控或手势事件。例如，用户使用触摸屏与移动设备应用程序中的 Flex 组件进行交互。组件将响应 mouseDown 和 mouseOver 等鼠标事件，但不响应触控事件和手势事件。

例如，用户按下触摸屏以选择 Flex Button 控件。 Button 控件使用 mouseUp 和 mouseDown 事件来指示用户与控件进行了交互。 Scroller 控件使用 mouseMove 和 mouseUp 事件来指示用户正在滚动显示画面。

Adobe 开发人员 Evangelist Paul Trani 在 Touch Events and Gesture on Mobile 中说明了如何处理触控和手势事件。

控制由 AIR 生成的事件

flash.ui.Multitouch.inputMode 属性用于控制由 AIR 和 Flash Player 生成的事件。flash.ui.Multitouch.inputMode 属性的值可以是以下值之一:

- MultitouchInputMode.NONE AIR 分派鼠标事件，但不分派触控和手势事件。
- MultitouchInputMode.TOUCH_POINT AIR 分派鼠标和触控事件，但不分派手势事件。在此模式中， Flex 框架接收的鼠标事件与 MultitouchInputMode.NONE 模式相同。

- MultitouchInputMode.GESTURE AIR 分派鼠标和手势事件，但不分派触控事件。在此模式中，Flex 框架接收的鼠标事件与 MultitouchInputMode.NONE 模式相同。

如列表中所示，无论如何设置 flash.ui.Multitouch.inputMode 属性的值， AIR 始终都会分派鼠标事件。因此， Flex 组件始终可以对用户通过触摸屏所做的交互进行响应。

Flex 允许您在应用程序中针对 flash.ui.Multitouch.inputMode 属性使用任何值。因此，尽管 Flex 组件不响应触控和手势事件，但您可以在应用程序中添加功能来响应所有事件。例如，可以在 Button 控件中添加事件处理函数来处理 touchTap、touchOver 和 touchMove 事件等触控事件。

《ActionScript 3.0 开发者指南》中概括介绍了在不同设备上处理用户输入的方式，以及触控、多点触控和手势输入的使用。有关更多信息，请参阅：

- 用户交互的基础知识
- 触控、多点触控和手势输入

定义移动设备应用程序和启动屏幕

创建移动设备应用程序容器

移动设备应用程序中的第一个标签通常是以下标签之一：

- <s:ViewNavigatorApplication> 标签用于定义只有一个部分的移动设备应用程序。
- <s:TabbedViewNavigatorApplication> 标签用于定义有多个部分的移动设备应用程序。

开发用于平板电脑的应用程序时，屏幕大小限制并不像在手机应用程序中那样重要。因此，对于平板电脑，不需要以小视图来构建应用程序。可以使用标准 Spark Application 容器以及受支持的移动设备组件和外观来构建应用程序。

注：在开发任何移动设备应用程序时 （即使是用于移动设备的程序），都可以使用 Spark Application 容器。但是， Spark Application 容器不支持视图导航、数据持久化机制、设备的后退和菜单按钮。有关更多信息，请参阅关于 Application 容器的第 29 页的 “ 移动设备应用程序容器与 Spark Application 容器的区别 ”。

移动设备应用程序容器具有以下默认的特性：

特性	Spark ViewNavigatorApplication 和 TabbedViewNavigatorApplication 容器
默认大小	100% 高， 100% 宽，占整个可用屏幕空间。
子代布局	由组成应用程序视图的各 View 容器定义。
默认内边距	0 像素。
滚动条	无。如果在应用程序容器的外观中添加滚动栏，用户可以滚动整个应用程序。包括应用程序的 ActionBar 和选项卡栏区域。您通常并不希望滚动视图的这些区域。因此，将滚动条添加到应用程序的各 View 容器中，而不是添加到应用程序容器的外观中。

移动设备应用程序容器与 Spark Application 容器的区别

Spark 移动设备应用程序容器的大部分功能与 Spark Application 容器相同。例如，可以对移动设备应用程序容器应用样式，其方法与 Spark Application 容器的样式应用方法相同。

Spark 移动设备应用程序容器有一些与 Spark Application 容器不同的特征：

- 支持持久化

  支持在磁盘中存储和加载数据。通过持久化机制，用户可以中断移动设备应用程序的运行，例如接听电话，然后在通话结束时恢复应用程序的状态。

- 支持视图导航

  ViewNavigatorApplication 容器自动创建一个 ViewNavigator 容器，以控制视图之间的导航。

  TabbedViewNavigatorApplication 容器自动创建一个 TabbedViewNavigator 容器，以控制部分之间的导航。

- 支持设备的后退和菜单按钮

  当用户按下后退按钮时，应用程序将导航回到堆栈中的上一个视图。当用户按下菜单按钮时，将显示当前视图的 ViewMenu 容器 （如果已经定义）。

有关 Spark 应用程序容器的更多信息，请参阅 About the Application container。

处理应用程序级别的事件

NativeApplication 类代表一个 AIR 应用程序。它负责提供应用程序信息和应用程序级功能，并分派应用程序级事件。可以使用静态属性 NativeApplication.nativeApplication 来访问与移动设备应用程序对应的 NativeApplication 类的实例。

例如， NativeApplication 类定义了可以在移动设备应用程序中处理的 invoke 和 exiting 事件。下面的示例引用该 NativeApplication 类，来定义 exiting 事件的事件处理函数：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkNativeApplicationEvent.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            protected function creationCompleteHandler(event:FlexEvent):void {
                // Reference NativeApplication to assign the event handler.
                NativeApplication.nativeApplication.addEventListener(Event.EXITING, myExiting);
            }

            protected function myExiting(event:Event):void {
                // Handle exiting event.
            }
        ]]>
    </fx:Script>

</s:ViewNavigatorApplication>
```

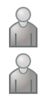
请注意，需使用 ViewNavigatorApplication.navigator 属性来访问 ViewNavigator。

在应用程序中添加启动屏幕

Spark Application 容器是 ViewNavigatorApplication 和 TabbedViewNavigatorApplication 容器的基类。用于 Spark 主题时， Spark Application 容器支持应用程序预加载器，以显示应用程序 SWF 文件的下载和初始化进度。

用于 Mobile 主题时，则可以显示启动屏幕。启动屏幕在应用程序启动期间显示。

注：要在桌面应用程序中使用启动屏幕，请将 Application.preloader 属性设置为 spark.preloaders.SplashScreen。同时将 frameworks\libs\mobile\mobilecomponents.swc 添加到应用程序的库路径中。



博客 Joseph Labrecque 张贴了关于适用于 Flex 中的 Android 启动屏幕的 AIR 的视频。

博客 Brent Arnold 编写了关于将启动屏幕添加到 Android 应用程序的视频。

从图像文件中添加启动屏幕

可以直接从图像文件中加载启动屏幕。要配置启动屏幕，请使用应用程序类的 splashScreenImage、 splashScreenScaleMode 和 splashScreenMinimumDisplayTime 属性。

例如，以下示例从使用信箱格式的 JPG 文件中加载启动屏幕。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileSplashScreen.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    splashScreenImage="@Embed('assets/logo.jpg')"
    splashScreenScaleMode="letterbox">

</s:ViewNavigatorApplication>
```

从自定义组件中添加启动屏幕

上一部分中的示例使用 JPG 文件定义启动屏幕。该机制的缺点是：无论运行应用程序的移动设备具有什么功能，应用程序都使用相同的图像。

移动设备具有不同的屏幕分辨率和大小。您可以定义自定义组件，而不是将单个图像用作启动屏幕。此组件决定着移动设备的功能并将合适的图像用作启动屏幕。

使用 SplashScreenImage 类定义自定义组件，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\myComponents\MySplashScreen.mxml -->
<s:SplashScreenImage xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">

    <!-- Default splashscreen image. -->
    <s:SplashScreenImageSource
        source="@Embed('../assets/logoDefault.jpg')"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Portrait.jpg')"
        dpi="240"
        aspectRatio="portrait"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Landscape.jpg')"
        dpi="240"
        aspectRatio="landscape"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo160.jpg')"
        dpi="160"
        aspectRatio="portrait"
        minResolution="960"/>
</s:SplashScreenImage>
```

在组件定义内，使用 SplashScreenImageSource 类定义每个启动屏幕图像。SplashScreenImageSource.source 属性指定图像文件。 SplashScreenImageSource dpi、 aspectRatio 和 minResolution 属性定义显示图像所需的移动设备功能。

例如，第一个 SplashScreenImageSource 定义仅指定图像的 source 属性。由于没有用于 dpi、aspectRatio 和 minResolution 属性的设置，此图像可以在任何设备上使用。因此，它可以定义在没有其它图像与设备功能相符时显示的默认图像。

第二和第三个 SplashScreenImageSource 定义指定在纵向或横向模式下用于 240 DPI 设备的图像。

最后一个 SplashScreenImageSource 定义指定在纵向模式下用于 160 DPI 设备、最低分辨率为 960 像素的图像。minResolution 属性的值与 Stage.stageWidth 和 Stage.stageHeight 属性值中的较大值进行对照。这两个值中的较大值必须等于或大于 minResolution 属性。

以下移动设备应用程序使用此组件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileSplashComp.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    splashScreenImage="myComponents.MySplashScreen">
</s:ViewNavigatorApplication>
```

SplashScreenImage 类自动确定与设备功能最匹配的图像。此匹配操作基于每个 SplashScreenImageSource 定义的 dpi、aspectRatio 和 minResolution 属性。

决定最佳匹配项的步骤如下所示：

1 决定所有与移动设备设置匹配的所有 SplashScreenImageSource 定义。匹配发生在以下情况：

  a SplashScreenImageSource 定义没有对此设置进行显式定义。例如，dpi 属性的设置不与任何设备的 DPI 设置。

  b 对于 dpi 或 aspectRatio 属性，此属性必须与移动设备的相应设置完全匹配。

  c 对于 minResolution 属性，当 Stage.stageWidth 和 Stage.stageHeight 属性值中的较大值等于或大于 minResolution 时，此属性与设备上的设置匹配。

2 如果有多个 SplashScreenImageSource 定义与设备匹配，则：

  a 选择显式设置数最大的定义。例如，与仅指定 dpi 属性的 SplashScreenImageSource 定义相比，同时指定 dpi 和 aspectRatio 属性的 SplashScreenImageSource 定义是更佳匹配项。

  b 如果仍存在多个匹配项，请选择 minResolution 值最大的匹配项。

  c 如果仍存在多个匹配项，请选择组件中定义的第一个匹配项。

显式选择启动屏幕图像

SplashScreenImage.getImageClass() 方法决定与移动设备功能最匹配的 SplashScreenImageSource 定义。您可以重写此方法以添加自己的自定义逻辑，如下例所示。

在该示例中，添加用于 iOS 启动屏幕的 SplashScreenImageSource 定义。在重写 getImageClass() 方法的主体中，首先确定应用程序是否在 iOS 中运行。如果是这样，显示特定于 iOS 的图像。

如果应用程序未在 iOS 中运行，则调用 super.getImageClass() 方法。此方法使用默认实现决定要显示的 SplashScreenImageSource 实例：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\myComponents\MyIOSSplashScreen.mxml -->
<s:SplashScreenImage xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <fx:Script>
        <![CDATA[
            // Override getImageClass() to return an image for iOS.
          override public function getImageClass(aspectRatio:String, dpi:Number, resolution:Number):Class {
                // Is the application running on iOS?
                if (Capabilities.version.indexOf("IOS") == 0)
                    return iosImage.source;

                return super.getImageClass(aspectRatio, dpi, resolution);
            }
        ]]>
    </fx:Script>
    <!-- Default splashscreen image. -->
    <s:SplashScreenImageSource
        source="@Embed('../assets/logoDefault.jpg')"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Portrait.jpg')"
        dpi="240"
        aspectRatio="portrait"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo240Landscape.jpg')"
        dpi="240"
        aspectRatio="landscape"/>

    <s:SplashScreenImageSource
        source="@Embed('../assets/logo160.jpg')"
        dpi="160"
        aspectRatio="portrait"
        minResolution="960"/>
    <!-- iOS splashscreen image. -->
    <s:SplashScreenImageSource id="iosImage"
        source="@Embed('../assets/logoIOS.jpg')"/>
</s:SplashScreenImage>
```

定义移动设备应用程序中的视图

移动设备应用程序通常会定义多个屏幕或视图。用户在应用程序中导航时，会在不同的视图之间切换。

应当为应用程序用户呈现直观的导航。也就是说，当用户从一个视图移动到另一个视图时，用户希望可以导航回上一个视图。应用程序可以定义 “ 主页 ” 按钮或其它顶级导航辅助功能，使用户可以从应用程序中的任意位置移动到特定位置。

要定义移动设备应用程序的视图，请使用 View 容器。要控制移动设备应用程序各视图之间的导航，请使用 ViewNavigator 容器。

使用 pushView() 切换视图

可以使用 ViewNavigator.pushView() 方法，将新视图推送到堆栈中。可以使用 ViewNavigatorApplication.navigator 属性访问 ViewNavigator。通过推送视图，可以将应用程序显示画面切换为新的视图。

pushView() 方法的语法如下 :

```
pushView(viewClass:Class,
    data:Object = null,
    context:Object = null,
    transition:spark.transitions:ViewTransitionBase = null):void
```

其中：

- viewClass 指定视图的类名称。该类通常对应于定义该视图的 MXML 文件。
- data 指定传递给视图的任何数据。此对象将写入新视图的 View.data 属性中。
- context 指定写入 ViewNavigator.context 属性中的任意对象。创建新视图后，该视图可以引用此属性并根据此值执行操作。例如，视图可能会根据 context 值来以不同的方式显示数据。
- transition 指定在切换到新视图时将执行的过渡效果。有关视图过渡效果的信息，请参阅第 79 页的 “ 定义移动设备应用程序中的过渡效果 ”。

使用 **data** 参数传递单个对象

可以使用 data 参数传递单个对象，其中包含新视图所需要的任何数据。然后，视图可以使用 View.data 属性来访问该对象，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employee View">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

在此示例中，EmployeeView 在 EmployeeView.mxml 文件中定义。该视图使用 data 属性来从传递给它的对象中访问员工姓名及 ID。

在 View 对象发生 add 事件时， View.data 属性必须有效。有关 View 容器生命周期的更多信息，请参阅 第 40 页的 “Spark ViewNavigator 和 View 容器的生命周期 ”。

将数据传递给应用程序中的第一个视图

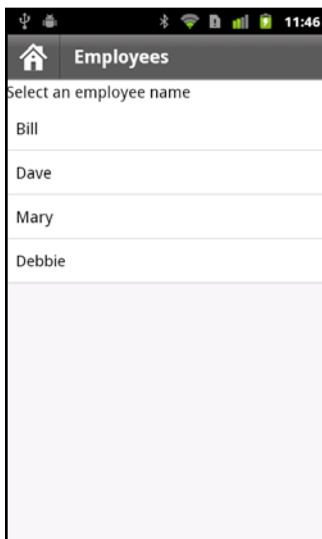
ViewNavigatorApplication.firstView 和 ViewNavigator.firstView 属性用于定义应用程序中的第一个视图。要将数据传递给第一个视图，请使用 ViewNavigatorApplication.firstViewData 属性或 ViewNavigator.firstViewData 属性。

将数据传递给视图

在下面的示例中，将使用 ViewNavigatorApplication 容器定义移动设备应用程序。 ViewNavigatorApplication 容器将自动创建 ViewNavigator 类的一个实例，可以使用该实例在应用程序所定义的视图之间导航。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSection.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
        firstView="views.EmployeeMainView">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
            click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

在此示例中，将在 ActionBar 控件的导航区域中定义 " 主页 " 按钮。选择 " 主页 " 按钮时，将从堆栈中弹出所有视图，从而返回到第一个视图。下图中显示了此应用程序：



EmployeeMainView.mxml 文件定义应用程序的第一个视图，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeMainView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employees">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;
            protected function myList_changeHandler(event:IndexChangeEvent):void {
                navigator.pushView(views.EmployeeView,myList.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select an employee name"/>
    <s:List id="myList"
        width="100%" height="100%"
        labelField="firstName"
        change="myList_changeHandler(event)">
        <s:ArrayCollection>
            <fx:Object firstName="Bill" lastName="Smith" companyID="11233"/>
            <fx:Object firstName="Dave" lastName="Jones" companyID="13455"/>
            <fx:Object firstName="Mary" lastName="Davis" companyID="11543"/>
            <fx:Object firstName="Debbie" lastName="Cooper" companyID="14266"/>
        </s:ArrayCollection>
    </s:List>
</s:View>
```

此视图定义一个 List 控件，供用户选择员工姓名。选择姓名时， change 事件的事件处理函数将另一个视图的实例推送到名为 EmployeeView 的堆栈中。推送 EmployeeView 实例时，将使应用程序切换到 EmployeeView 视图。

在此示例中， pushView() 方法带有两个参数：新视图以及用于定义传递给新视图的数据的对象。在此示例中，将传递与 List 控件中当前选定项目对应的数据对象。

下例中显示的是 EmployeeView 的定义：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employee View">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

EmployeeView 显示来自 List 控件数据提供程序的三个字段。 EmployeeView 使用 View.data 属性来访问传递的数据。

博客 Steve Mathews 编写了一个关于在视图之间传递数据的手册条目。

从视图中返回数据

ViewNavigator.popView() 方法将控制权从当前视图返回给堆栈中的上一个视图。执行 popView() 方法时，将破坏当前视图并恢复堆栈中的上一个视图。恢复上一个 View 时，将从堆栈中重置其 data 属性。

有关视图生命周期（包括创建期间所分派的事件）的完整说明，请参阅 第 40 页的 “Spark ViewNavigator 和 View 容器的生命周期 ”。

恢复新视图时，将使用视图停用时的初始 data 对象。因此，通常不使用初始 data 对象来将旧视图的数据传递给新视图。而是重写旧视图的 createReturnObject() 方法。 createReturnObject() 方法将返回一个对象。

返回对象类型

createReturnObject() 方法所返回的对象将写入 ViewNavigator.poppedViewReturnedObject 属性中。poppedViewReturnedObject 属性的数据类型为 ViewReturnObject。

ViewReturnObject 定义两个属性：context 和 object。object 属性包含由 createReturnObject() 方法返回的对象。context 属性中包含在使用 pushView() 将视图推送到导航堆栈中时被传递给视图的 context 参数值。

在新视图接收 add 事件前，必须在视图中设置 poppedViewReturnedObject 属性。如果 poppedViewReturnedObject.object 属性值为 null，将不返回任何数据。

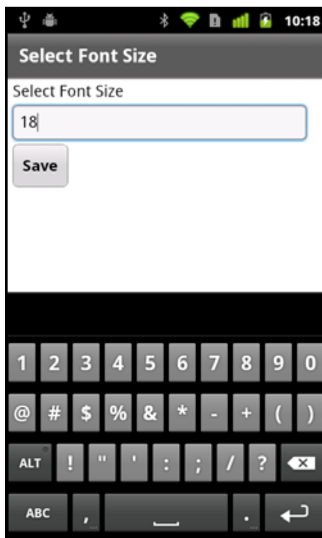
示例：将数据传递给视图

在下面的示例 SelectFont.mxml 中，显示了一个用于设置字体大小的视图。 createReturnObject() 方法的重写语句将返回数字值。从上一个视图传递的 data 属性的 fontSize 字段将设置 TextInput 控件的初始值：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SelectFont.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Select Font Size"
    add="addHandler(event);">
    <s:layout>
        <s:VerticalLayout paddingTop="10"
            paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            // Define return Number object.
            protected var fontSize:Number;

            // Initialize the return object with the passed in font size.
            // If you do not set a value,
            // return this value for the font size.
            protected function addHandler(event:FlexEvent):void {
                fontSize = data.fontSize;
            }
```

```
            // Save the value of the specified font.
            protected function changeHandler(event:Event):void {
                fontSize=Number(ns.text);
                navigator.popView();
            }

            // Override createReturnObject() to return the new font size.
            override public function createReturnObject():Object {
                return fontSize;
            }
        ]]>
    </fx:Script>

    <s:Label text="Select Font Size"/>
    <!-- Set the initlial value of the TextInput to the passed fontSize -->
    <s:TextInput id="ns"
        text="{data.fontSize}"/>
    <s:Button label="Save" click="changeHandler(event);"/>
</s:View>
```

下图显示的是使用 SelectFont.mxml 定义的视图：



下例中的视图 MainFontView.mxml 使用在 SetFont.mxml 中所定义的视图。 MainFontView.mxml 视图定义以下内容：

- ActionBar 中的一个 Button 控件，用于切换到 SetFont.mxml 所定义的视图。
- add 事件的一个事件处理函数，该处理函数首先确定 View.data 属性值是否为 null。如果是 null，事件处理函数将 data.fontSize 字段添加到 View.data 属性中。

  如果 data 属性值不是 null，则事件处理函数将字体大小设置为 data.fontSize 字段中的值。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\MainFontView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Font Size"
    add="addHandler(event);">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;

            // Change to the SelectFont view, and pass the current data property.
            // The data property contains the fontSize field with the current font size.
            protected function clickHandler(event:MouseEvent):void {
                navigator.pushView(views.SelectFont, data);
            }
            // Set the font size in the event handler for the add event.
            protected function addHandler(event:FlexEvent):void {
                // If the data property is null,
                // initialize it and create the data.fontSize field.
                if (data == null) {
                   data = new Object();
                   data.fontSize = getStyle('fontSize');
                   return;
                }

                // Otherwise, set data.fontSize to the retured value,
                // and set the font size.
                data.fontSize = navigator.poppedViewReturnedObject.object;
                setStyle('fontSize', data.fontSize);
            }
        ]]>
    </fx:Script>

    <s:actionContent>
        <s:Button label="Set Font&gt;"
            click="clickHandler(event);"/>
    </s:actionContent>

    <s:Label text="Text to size."/>
</s:View>
```

针对纵向和横向配置应用程序

当移动设备设备的方向发生改变时，设备将自动设置应用程序的方向。为了针对不同的方向配置应用程序， Flex 定义了分别对应于纵向和横向的两种视图状态：portrait 和 landscape。可以使用这些视图状态来基于方向设置应用程序的特性。

在下例中，将使用视图状态，基于当前方向控制 Group 容器的 layout 属性：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SearchViewStates.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Search">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <s:states>
        <s:State name="portrait"/>
        <s:State name="landscape"/>
    </s:states>

    <s:Group>
        <s:layout>
            <s:VerticalLayout/>
        </s:layout>
        <s:layout.landscape>
            <s:HorizontalLayout/>
        </s:layout.landscape>
        <s:TextInput text="Enter search text" textAlpha="0.5"/>
        <s:Button label="Search"/>
    </s:Group>
    <s:TextArea text="search results" textAlpha="0.5"/>
</s:View>
```

此示例定义了一个搜索视图。 Group 容器控制着输入搜索文本和搜索按钮的布局。在纵向模式下， Group 容器使用垂直布局。当布局改为横向模式时， Group 容器将使用水平布局。

定义自定义外观以支持布局模式

可以为移动设备应用程序定义自定义的外观类。如果该外观支持纵向和横向布局，则必须处理 portrait 和 landscape 视图状态。

可以将应用程序配置为，当用户旋转设备时，应用程序不更改布局方向。为此，请编辑应用程序的 XML 文件 （以 -app.xml 结尾的文件），设置以下属性：

- 要禁止应用程序更改布局方向，请将 <autoOrients> 属性设置为 false。
- 要设置方向，请将 <aspectRatio> 属性设置为 portrait 或 landscape。

设置 Spark ViewNavigator 容器的覆盖模式

默认情况下，移动设备应用程序的选项卡栏和 ActionBar 控件所定义的区域不能供应用程序视图使用。也就是说，内容在移动设备上不能全屏显示。

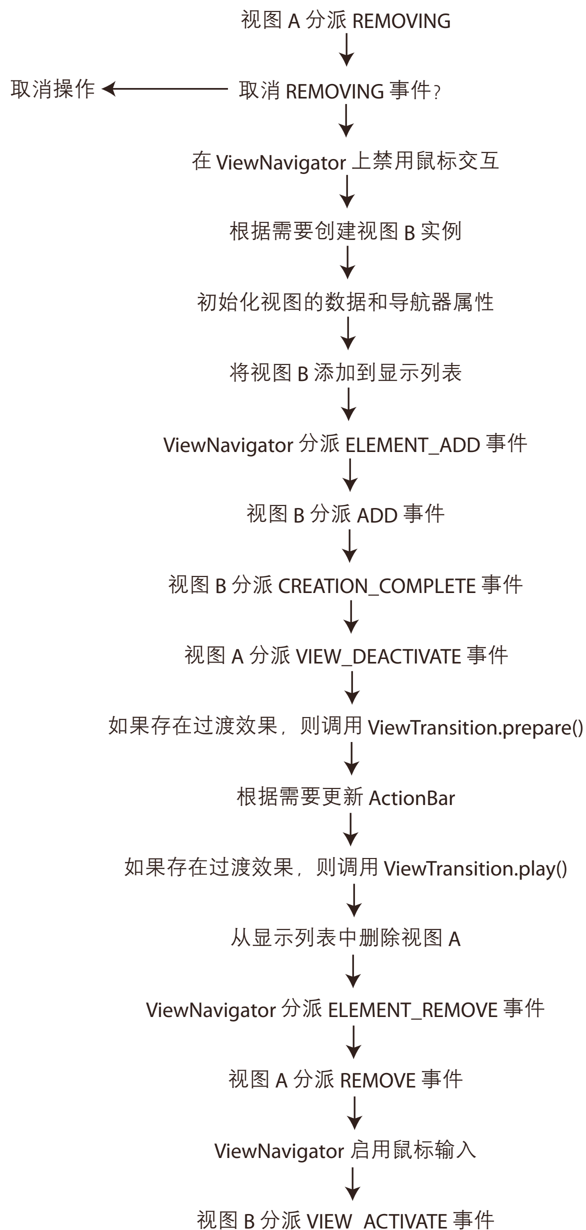
但您可以使用 ViewNavigator.overlayControls 属性来更改这些组件的默认布局。如果将 overlayControls 属性设置为 true，应用程序的内容区域将横跨屏幕的整个宽度和高度。ActionBar 控件和选项卡栏将悬浮在内容区域之上，其 Alpha 值使其呈半透明状态。

ViewNavigator 容器的外观类 spark.skins.mobile.ViewNavigatorSkin 定义视图状态，以处理 overlayControls 属性的不同值。当 overlayControls 属性为 true 时，将在当前状态名称中追加 “AndOverlay”。例如，默认情况下 ViewNavigator 的外观处于 “portrait” 状态。当 overlayControls 属性为 true 时，导航器外观状态更改为 “portraitAndOverlay”。

Spark ViewNavigator 和 View 容器的生命周期

从移动设备应用程序中的一个视图切换到另一个视图时，Flex 将执行一系列操作。在视图切换过程的各个点上，Flex 都会分派事件。在此过程中，可以监视这些事件，以执行相应操作。例如，可以使用 removing 事件来取消视图的切换。

下图说明了从当前视图（视图 A）切换到另一个视图（视图 B）的过程：



定义移动设备应用程序中的选项卡

定义应用程序的各部分

使用 TabbedViewNavigatorApplication 容器可以定义带有多个部分的移动设备应用程序。TabbedViewNavigatorApplication 容器将自动创建 TabbedViewNavigator 容器。TabbedViewNavigator 容器将创建一个选项卡栏，用以支持在应用程序各部分之间进行导航。

每个 ViewNavigator 容器定义应用程序的不同部分。使用 TabbedViewNavigatorApplication 容器的 navigators 属性可以指定 ViewNavigator 容器。

在下例中，将定义三个部分，分别对应三个 ViewNavigator 标签。每个 ViewNavigator 定义在切换到该部分时所显示的第一个视图：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMultipleSections.mxml -->
<s:TabbedViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">

    <s:navigators>
        <s:ViewNavigator label="Employees" firstView="views.EmployeeMainView"/>
        <s:ViewNavigator label="Contacts" firstView="views.ContactsMainView"/>
        <s:ViewNavigator label="Search" firstView="views.SearchView"/>
    </s:navigators>

</s:TabbedViewNavigatorApplication>
```

注：无需在 MXML 中指定 navigators 子标签，因为它是 TabbedViewNavigator 的默认属性。

每个 ViewNavigator 各自维护独立的导航堆栈。因此， ViewNavigator 方法 （例如 pushView() 和 popView()）适用于当前的活动部分。移动设备上的后退按钮将控件返回到当前 ViewNavigator 堆栈中的上一个视图。切换视图并不会改变当前部分。

无需在应用程序中添加任何特定逻辑即可实现部分之间的导航。 TabbedViewNavigator 容器会在应用程序底部自动创建一个选项卡栏，用以控制在各部分之间的导航。

您可以添加当前部分的编程控件，但这不是必需的。要以编程方式切换部分，请将 TabbedViewNavigator.selectedIndex 属性设置为所需部分的索引。部分的索引从 0 开始：应用程序的第一个部分使用索引 0，第二个部分使用索引 1，依此类推。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了关于使用 ViewNavigator 导航堆栈的视频。

Adobe 开发人员 Holly Schinsky 在 Flex Mobile Development - Passing Data Between Tabs 文章中介绍了在移动设备应用程序的标签之间传递数据的方式。

观看 video2brain 上有关 TabbedViewNavigator 容器的视频 Creating a Tabbed View Navigator Application。

处理部分切换事件

切换部分时， TabbedViewNavigator 容器将分派以下事件：

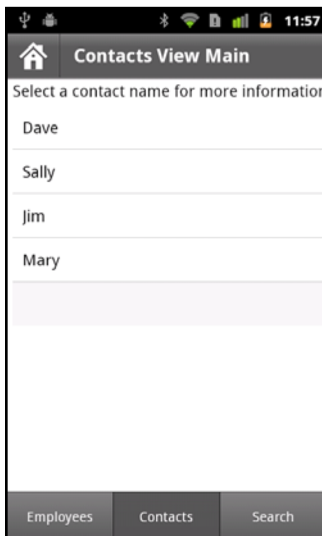
- changing 事件在切换部分之前分派。要阻止在部分之间切换，请在 changing 事件的事件处理函数中调用 preventDefault() 方法。
- change 事件在切换部分之后分派。

针对多个部分配置 ActionBar

ActionBar 控件与 ViewNavigator 相关联。因此，可以在定义各部分的 ViewNavigator 时为各部分配置 ActionBar。在下例中，将为定义应用程序三个不同部分的每个 ViewNavigator 容器分别配置 ActionBar：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMultipleSectionsAB.mxml -->
<s:TabbedViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first section in the application.
                tabbedNavigator.selectedIndex = 0;
                // Switch to the first view in the section.
                ViewNavigator(tabbedNavigator.selectedNavigator).popToFirstView();
            }
        ]]>
    </fx:Script>

    <s:navigators>
        <s:ViewNavigator label="Employees" firstView="views.EmployeeMainView">
            <s:navigationContent>
                <s:Button icon="@Embed(source='assets/Home.png')"
                    click="button1_clickHandler(event)"/>
            </s:navigationContent>
        </s:ViewNavigator>
        <s:ViewNavigator label="Contacts" firstView="views.ContactsMainView">
            <s:navigationContent>
                <s:Button icon="@Embed(source='assets/Home.png')"
                    click="button1_clickHandler(event)"/>
            </s:navigationContent>
        </s:ViewNavigator>
        <s:ViewNavigator label="Search" firstView="views.SearchView">
            <s:navigationContent>
                <s:Button icon="@Embed(source='assets/Home.png')"
                    click="button1_clickHandler(event)"/>
            </s:navigationContent>
        </s:ViewNavigator>
    </s:navigators>

</s:TabbedViewNavigatorApplication>
```

下图显示了该应用程序，此时在选项卡栏中选择了“联系人”选项卡：



或者，可以在应用程序的每个视图中定义 ActionBar。这样，每个视图都使用相同的 ActionBar 内容，而不论它用在应用程序的什么位置。

控制选项卡栏

在视图中隐藏选项卡栏

可以通过将 View.tabBarVisible 属性设置为 false，来隐藏任何视图中的选项卡栏。默认情况下，tabBarVisible 属性的值为 true，表示显示选项卡栏。

您也可以使用 TabbedViewNavigator.hideTabBar() 和 TabbedViewNavigator.showTabBar() 方法来控制可见性。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了关于隐藏选项卡栏的视频。

将效果应用到 TabbedViewNavigator 容器的选项卡栏

默认情况下，选项卡栏使用滚动效果作为其显示和隐藏效果。在切换当前选定的选项卡时，选项卡栏不使用任何效果。

可以通过重写 TabbedViewNavigator.createTabBarHideEffect() 和 TabbedViewNavigator.createTabBarShowEffect() 方法来更改显示和隐藏选项卡栏时的默认效果。在隐藏选项卡栏后，请记得将选项卡栏的 visible 和 includeInLayout 属性设置为 false。

在移动设备应用程序中创建多个窗格

SplitViewNavigator 是可更换外观的容器，可以在移动设备的同一屏幕上显示两个或更多的子视图导航器。每个视图导航器都显示在 SplitViewNavigator 容器管理的单独窗格中。

SplitViewNavigator 容器的子代可以是用于扩展 ViewNavigatorBase 的任何组件。因此，您可以使用 ViewNavigator 和 TabbedViewNavigator 容器作为其子代。

注：由于同时显示多个窗格所需的屏幕空间的因素， Adobe 建议您在平板电脑中仅使用 SplitViewNavigator。

默认情况下， SplitViewNavigator 水平排列窗格以与其子代对应。您可以指定是使用垂直布局还是定义自定义布局。

创建 SplitViewNavigator 容器

平板电脑设备的通用界面模式是主 / 细节模式。该模式将屏幕分为两个主要内容区域：主窗格和细节窗格。通常，用户与主窗格交互以控制细节窗格中的内容显示。

每个窗格对应 SplitViewNavigator 的一个子代，其中子代为 ViewNavigator 或 TabbedViewNavigator 容器。由于其子代是视图导航器，因此每个窗格的视图导航器都包含各自的视图堆栈和操作栏。

下图显示应用程序中的 SplitViewNavigator 容器，其中包含一个主窗格和一个细节窗格：

在该示例中，左边的主窗格包含用于显示一组 Adobe 产品的 Spark List 控件。右边的细节窗格显示有关主窗格中的当前选定产品的附加信息。

该示例的主应用程序文件如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSplitVNSimple.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <s:SplitViewNavigator width="100%" height="100%">
        <s:ViewNavigator width="256" height="100%"
            firstView="views.MasterCategory"/>
        <s:ViewNavigator width="100%" height="100%"
            firstView="views.DetailView"/>
    </s:SplitViewNavigator>
</s:Application>
```

SplitViewNavigator 可以是 Application 或 TabbedViewNavigatorApplication 容器的子代。在该示例中，SplitViewNavigator 是 Application 容器的唯一子代。请注意，SplitViewNavigator 指定的高度和宽度为 100%，占满整个设备屏幕区域。

在该示例中，SplitViewNavigator 的子代是 ViewNavigator 容器。第一个 ViewNavigator 定义主窗格，第二个定义细节窗格。

注：SplitViewNavigator 可以拥有两个以上的子代。因此，您可以创建具有 3 个、 4 个或更多窗格的 SplitViewNavigator。

访问 SplitViewNavigator 容器的窗格和视图

SplitViewNavigator 容器定义您用于访问其子代的方法和属性。例如，使用 SplitViewNavigator.numViewNavigators 属性确定 SplitViewNavigator 的子视图导航器数。

使用 SplitViewNavigator.getViewNavigatorAt() 方法基于子代索引访问 SplitViewNavigator 的子代。在上面的示例中，主窗格的 ViewNavigator 使用索引 0，细节窗格的 ViewNavigator 使用索引 1。

注：SplitViewNavigator 容器继承 getElementAt() 和 getElementIndex() 方法。不要对 SplitViewNavigator 使用这些方法，而应使用 getViewNavigatorAt()。

从对单个窗格的 ViewNavigator 的引用， SplitViewNavigator 可以访问该窗格的各个视图。

通过使用子代的 parentNavigator 属性从子代访问 SplitViewNavigator 容器。例如，ViewNavigator.parentNavigator 包含对父 SplitViewNavigator 容器的引用。

View 容器通过使用 View.navigator 属性访问其父视图导航器。因此，视图可以使用 View.navigator.parentNavigator 访问 SplitViewNavigator。

在上面的示例中，将主窗格的 ViewNavigator 指定为它的第一个视图 MasterCategory。该视图是在 MasterCategory.mxml 文件中定义的，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\MasterCategory.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Master">

    <fx:Script>
        <![CDATA[
            import spark.components.SplitViewNavigator;
            import spark.components.ViewNavigator;
            import spark.events.IndexChangeEvent;

            protected function myList_changeHandler(event:IndexChangeEvent):void {
                // Create a reference to the SplitViewNavigator.
                var splitNavigator:SplitViewNavigator = navigator.parentNavigator as SplitViewNavigator;
                // Create a reference to the ViewNavigator for the Detail frame.
                var detailNavigator:ViewNavigator = splitNavigator.getViewNavigatorAt(1) as ViewNavigator;
                // Change the view of the Detail frame based on the selected List item.
                detailNavigator.pushView(DetailView, myList.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:List width="100%" height="100%" id="myList"
            change="myList_changeHandler(event);">
        <s:dataProvider>
            <s:ArrayCollection>
                <fx:Object Product="Adobe AIR" Price="11.99"
                           Image="@Embed(source='../assets/air_icon_sm.jpg')"
                           Description="Try AIR." Link="air"/>
                <fx:Object Product="Adobe ColdFusion" Price="11.99"
                           Image="@Embed(source='../assets/coldfusion_icon_sm.jpg')"
                           Description="Try ColdFusion." Link="coldfusion"/>
                <fx:Object Product="Adobe Flash Player" Price="11.99"
                           Image="@Embed(source='../assets/flashplayer_icon_sm.jpg')"
                           Description="Try Flash." Link="flashplayer"/>
                <fx:Object Product="Adobe Flex" Price="Free"
                           Image="@Embed(source='../assets/flex_icon_sm.jpg')"
                           Description="Try Flex." Link="flex.html"/>
                <fx:Object Product="Adobe LiveCycleDS" Price="11.99"
                           Image="@Embed(source='../assets/livecycleds_icon_sm.jpg')"
                           Description="Try LiveCycle DS." Link="livcycle"/>
                <fx:Object Product="Adobe LiveCycle ES2" Price="11.99"
                           Image="@Embed(source='../assets/livecyclees_icon_sm.jpg')"
                           Description="Try LiveCycle ES." Link="livcycle"/>
            </s:ArrayCollection>
        </s:dataProvider>
        <s:itemRenderer>
            <fx:Component>
                <s:IconItemRenderer
                    labelField="Product"
                    iconField="Image"/>
            </fx:Component>
        </s:itemRenderer>
    </s:List>
</s:View>
```

MasterCategory.mxml 定义包含 Adobe 产品相关信息的单个 List 控件。 List 控件使用自定义项呈示器显示每个产品的标签和图标。有关定义项呈示器的更多信息，请参阅对基于 Spark 列表的控件使用移动设备项呈示器。

主窗格中的 List 控件使用 change 事件更新细节窗格，以便对用户操作做出响应。事件处理函数首先获取对 SplitViewNavigator 容器的引用。从该引用中，它获取对细节框的 ViewNavigator 的引用。

最后，事件处理函数调用细节框的 ViewNavigator 中的 push() 方法。 push() 方法接受两个参数、推送到 ViewNavigator 堆栈的视图以及包含选定 List 项相关信息的对象。

更新 SplitViewNavigator 容器的细节窗格

上面示例中的细节窗格显示有关主窗格的 List 控件中选定项的相关信息。细节窗格命名为 DetailView 并在 DetailView.mxml 文件中定义，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\DetailView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Detail">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[

            // Use navigateToURL() to open a link to the product page.
            protected function label1_clickHandler(event:MouseEvent):void {
                var destinationURL:String = "http://www.adobe.com/products/" + data.Link;
                navigateToURL(new URLRequest(destinationURL));
            }
        ]]>
    </fx:Script>


    <s:VGroup width="461" height="670">
        <s:Image source="{data.Image}"
            height="176" width="169"
            horizontalCenter="0" top="0"/>
        <s:Label text="{data.Product}"
            fontSize="24" fontWeight="bold"
            top="100" left="0"/>
        <s:Label text="Price: {data.Price}"
            top="125" left="0"/>
        <s:TextArea y="174"
            width="100%" height="20%"
            contentBackgroundColor="0xCC6600"
            text="{data.Description}"/>
        <s:Label text="Click for more information"
            color="#0000FF"
            click="label1_clickHandler(event)"/>
    </s:VGroup>
</s:View>
```

主窗格将对象传递给与 List 控件中的选定项对应的 DetailView.mxml 文件。细节窗格使用 View.data 属性访问这些数据。然后，细节窗格显示产品的图像和产品相关信息，并创建指向包含更多产品相关信息的页面的超链接。

有关将数据传递给 View 容器的更多信息，请参阅第 34 页的 “ 将数据传递给视图 ”。

基于设备方向显示窗格

为平板电脑开发应用程序时，可以基于平板电脑方向使用不同的布局。例如，在横向模式下，平板电脑具有可轻松显示多个窗格的宽屏区域。在纵向布局中，如果屏幕较窄，由于屏幕宽度减少，可以选择隐藏窗格。

SplitViewNavigator 容器定义 autoHideFirstViewNavigator 属性，该属性可用于控制针对不同方向的第一个窗格的可见性。默认情况下， autoHideFirstViewNavigator 为 false，这样无论在什么方向，该容器都可以显示第一个窗格。

如果将 autoHideFirstViewNavigator 设置为 true，该容器在横向模式下显示第一个窗格，在纵向模式下隐藏第一个窗格。通过将相关视图导航器的 visible 属性设置为 false， SplitViewNavigator 容器将隐藏第一个窗格。

在隐藏第一个窗格的纵向模式下，可以使用 SplitViewNavigator.showFirstViewNavigatorInPopUp() 方法将其打开。调用时，该方法将打开 Callout 容器中的第一个窗格。 callout 容器是显示在应用程序顶部的弹出容器，如下图所示：

该示例将标记为 " 显示导航器 " 的按钮添加到 SplitViewNavigator 细节窗格的操作栏中。如果容器的第一个窗格处于隐藏状态，则用户可以选择该按钮打开主窗格。

注：为 SplitViewNavigator 创建自定义外观以打开 SkinnablePopUpContainer 或 SkinnablePopUpContainer 子类的第一个窗格。

Callout 已经打开时将忽略 showFirstViewNavigatorInPopUp() 方法。设备重定向到横向模式时， callout 会自动关闭并重新显示第一个窗格。

单击 Callout 容器以外的位置可以将其关闭。您还可以通过调用 SplitViewNavigator.hideViewNavigatorPopUp() 方法将其关闭。

有关 Callout 容器的更多信息，请参阅第 68 页的 " 将 callout 容器添加到移动设备应用程序中 "。

将操作栏添加到 Callout 容器中显示的窗格

将 SplitViewNavigator 的 autoHideFirstViewNavigator 属性设置为 true 的主应用程序文件如下所示。该示例在设备处于纵向模式时使用视图状态将按钮添加到细节窗格的操作栏中：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSVNOrient.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    resize="resizeHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.ResizeEvent;

            // Update the state based on the orientation of the device.
            protected function resizeHandler(event:ResizeEvent):void {
                currentState = aspectRatio;
            }
        ]]>
    </fx:Script>

    <s:states>
        <s:State name="portrait"/>
        <s:State name="landscape"/>
    </s:states>

    <s:SplitViewNavigator id="splitNavigator"
            width="100%" height="100%"
            autoHideFirstViewNavigator="true">

        <s:ViewNavigator width="256" height="100%"
            firstView="views.MasterCategoryOrient"/>
        <s:ViewNavigator width="100%" height="100%"
            firstView="views.DetailView">
            <s:actionContent.portrait>
                <s:Button id="navigatorButton"
                    label="Show Navigator"
                    click="splitNavigator.showFirstViewNavigatorInPopUp(navigatorButton);"/>
            </s:actionContent.portrait>
        </s:ViewNavigator>
    </s:SplitViewNavigator>
</s:Application>
```

应用程序在 Application 容器中添加用于 resize 事件的事件处理函数。 Flex 在平板电脑的方向改变时分派 resize 事件。在 resize 事件的事件处理函数中，基于当前方向设置应用程序的视图状态。有关视图状态的更多信息，请参阅视图状态。

细节窗格的视图导航器使用当前状态控制 Button 控件在操作栏中的外观。在横向模式下，该按钮由于主窗格可见而隐藏。

在纵向模式下，如果主窗格隐藏，则 Button 控件出现在细节窗格的操作栏中。然后，用户可以选择 Button 控件打开包含主窗格的 Callout。

将对 Button 控件的引用作为参数传递到 showFirstViewNavigatorInPopUp() 方法。该参数指定 Callout 容器的主机，这意味着相对于 Button 控件的位置确定 Callout 的位置。

关闭 SplitViewNavigator callout 以响应用户操作

单击 Callout 容器以外的位置可以将其关闭。但是默认情况下，单击 Callout 容器内的位置不会将其关闭。

该示例在用户通过调用 SplitViewnavigator.hideViewNavigatorPopUp() 方法选择 List 项时关闭 Callout。在 List 控件的 change 事件的事件处理函数中调用该方法，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\MasterCategoryOrient.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Master">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.components.SplitViewNavigator;
            import spark.components.ViewNavigator;
            import spark.events.IndexChangeEvent;

            protected function myList_changeHandler(event:IndexChangeEvent):void {
                // Create a reference to the SplitViewNavigator.
                var splitNavigator:SplitViewNavigator = navigator.parentNavigator as SplitViewNavigator;
                // Create a reference to the ViewNavigator for the Detail frame.
                var detailNavigator:ViewNavigator = splitNavigator.getViewNavigatorAt(1) as ViewNavigator;
                // Change the view of the Detail frame based on the selected List item.
                detailNavigator.pushView(DetailView, myList.selectedItem);

                // If the Master is open in a callout, close it.
                // Otherwise, this method does nothing.
                splitNavigator.hideViewNavigatorPopUp();
            }
        ]]>
    </fx:Script>

    <s:List width="100%" height="100%" id="myList"
            change="myList_changeHandler(event);">
        <s:dataProvider>
            <s:ArrayCollection>
                <fx:Object Product="Adobe AIR" Price="11.99"
                           Image="@Embed(source='../assets/air_icon_sm.jpg')"
                           Description="Try AIR." Link="air"/>
                <fx:Object Product="Adobe ColdFusion" Price="11.99"
                           Image="@Embed(source='../assets/coldfusion_icon_sm.jpg')"
                           Description="Try ColdFusion." Link="coldfusion"/>
                <fx:Object Product="Adobe Flash Player" Price="11.99"
                           Image="@Embed(source='../assets/flashplayer_icon_sm.jpg')"
                           Description="Try Flash." Link="flashplayer"/>
                <fx:Object Product="Adobe Flex" Price="Free"
                           Image="@Embed(source='../assets/flex_icon_sm.jpg')"
                           Description="Try Flex." Link="flex.html"/>
                <fx:Object Product="Adobe LiveCycleDS" Price="11.99"
                           Image="@Embed(source='../assets/livecycleds_icon_sm.jpg')"
                           Description="Try LiveCycle DS." Link="livcycle"/>
                <fx:Object Product="Adobe LiveCycle ES2" Price="11.99"
                           Image="@Embed(source='../assets/livecyclees_icon_sm.jpg')"
                           Description="Try LiveCycle ES." Link="livcycle"/>
            </s:ArrayCollection>
        </s:dataProvider>
        <s:itemRenderer>
            <fx:Component>
                <s:IconItemRenderer
                    labelField="Product"
                    iconField="Image"/>
            </fx:Component>
        </s:itemRenderer>
    </s:List>
</s:View>
```

实现 SplitViewNavigator 容器的持久化

移动设备设备应用程序的运行经常被其它操作（例如短信、电话或其它移动设备应用程序）中断。通常，当重新启动被打断的应用程序时，用户希望应用程序恢复先前的状态。通过持久化机制，设备可以将应用程序恢复为先前的状态。有关更多信息，请参阅第 105 页的“在移动设备应用程序中启用持久化机制”。

SplitViewNavigator 实现它从 ViewNavigatorBase 基类继承的 loadViewData() 和 saveViewData() 方法。因此，SplitViewNavigator 可以为它的每个子导航器序列化和反序列化导航堆栈以及视图数据。

但是，您必须在应用程序中断时手动调用这些方法，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSplitVNPersist.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    initialize="initializeHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.managers.PersistenceManager;

            // Create an instance of the PersistenceManager.
            public var persistenceManager:PersistenceManager;

            // Event handler to initialize SplitViewNavigator.
            protected function initializeHandler(event:FlexEvent):void {

                // Register the event handler for the deactivate event.
                NativeApplication.nativeApplication.addEventListener(Event.DEACTIVATE, onDeactivate);

                persistenceManager = new PersistenceManager();
                persistenceManager.load();

                var data:Object = persistenceManager.getProperty("navigatorState");
                if (data)
                    splitNavigator.loadViewData(data);
            }

            // Event handler to save SplitViewNavigator on application deactivate event.
            protected function onDeactivate(event:Event):void {
                persistenceManager.setProperty("navigatorState", splitNavigator.saveViewData());
                persistenceManager.save();
            }
        ]]>
    </fx:Script>

    <s:SplitViewNavigator id="splitNavigator" width="100%" height="100%">
        <s:ViewNavigator width="256" height="100%"
            firstView="views.MasterCategory"/>
        <s:ViewNavigator width="100%" height="100%"
            firstView="views.DetailView"/>
    </s:SplitViewNavigator>
</s:Application>
```

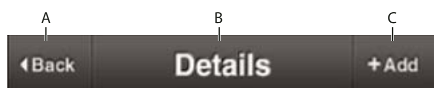
定义移动设备应用程序中的导航控件、标题控件和操作控件

配置 ActionBar 控件

ViewNavigator 容器可以定义 ActionBar 控件。 ActionBar 控件为标题控件、导航控件和操作控件提供了标准的区域。通过该控件，可以定义可在应用程序任何位置或特定视图中访问的全局控件。例如，可以使用 ActionBar 控件添加主页按钮、搜索按钮或其他选项。

对于仅有一个部分 （即只有一个 ViewNavigator 容器）的移动设备应用程序而言，所有视图共享同一个操作栏。对于带有多个部分 （即带有多个 ViewNavigator 容器）的移动设备应用程序而言，每个部分都会定义各自的操作栏。

使用 ActionBar 控件定义操作栏区域。 ActionBar 控件可以定义三种不同的区域，如下图所示：



A. 导航区域 **B.** 标题区域 **C.** 操作区域

ActionBar 区域

- 导航区域

  包含可用于在该部分进行导航的组件。例如，可以在导航区域中定义主页按钮。

  可以使用 navigationContent 属性定义导航区域中所显示的组件。可以使用 navigationLayout 属性定义导航区域的布局。

- 标题区域

  包含字符串 （标题文本）或组件。如果指定组件，则不能指定标题字符串。

  可以使用 title 属性指定在标题区域中所显示的字符串。可以使用 titleContent 属性定义在标题区域中所显示的组件。可以使用 titleLayout 属性定义标题区域的布局。如果为 titleContent 属性指定一个值，则 ActionBar 外观将忽略 title 属性。

- 操作区域

  包含多个组件，用于定义用户可在视图中执行的操作。例如，可以在操作区域中定义搜索或刷新按钮。

  可以使用 actionContent 属性定义在操作区域中所显示的组件。可以使用 actionLayout 属性定义操作区域的布局。

尽管 Adobe 建议您按上文所述方法使用导航、标题和操作区域，但对于放置在这些区域中的组件并没有任何限制。

在 **ViewNavigatorApplication**、 **ViewNavigator** 或 **View** 容器中设置 **ActionBar** 属性

可以在 ViewNavigatorApplication 容器、 ViewNavigator 容器或各 View 容器中设置用于定义 ActionBar 控件内容的属性。 View 容器的优先级最高，其次是 ViewNavigator，再其次是 ViewNavigatorApplication 容器。因此，在 ViewNavigatorApplication 容器中设置的属性将应用于整个应用程序，但您可以在 ViewNavigator 或 View 容器中重写这些属性。

注：ActionBar 控件与 ViewNavigator 相关联，因此专用于移动设备应用程序的单个部分。因此，无法从 TabbedViewNavigatorApplication 和 TabbedViewNavigator 容器定义 ActionBar。

示例：在应用程序级别自定义 Spark ActionBar 控件

下面的示例展示了一个移动设备应用程序的主应用程序文件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkActionBarSimple.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.MobileViewHome">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Perform a refresh
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button label="Home" click="navigator.popToFirstView();"/>
    </s:navigationContent>

    <s:actionContent>
        <s:Button label="Refresh" click="button1_clickHandler(event);"/>
    </s:actionContent>
</s:ViewNavigatorApplication>
```

此示例在 ActionBar 控件的导航内容区域中定义 " 主页 " 按钮，在操作内容区域定义 " 刷新 " 按钮。

下面的示例定义了 MobileViewHome View 容器，该容器定义应用程序的第一个视图。 View 容器定义了标题字符串 "Home View"，但不重写 ActionBar 控件的导航内容或操作内容区域。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\MobileViewHome.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Home View">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <s:Label text="Home View"/>
    <s:Button label="Submit"/>
</s:View>
```

示例：在 View 容器中自定义 ActionBar 控件

本示例使用带有一个部分的主应用程序文件，用以在 ViewNavigatorApplication 容器的导航区域中定义 " 主页 " 按钮，并在操作区域中定义 " 搜索 " 按钮：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkActionBarOverride.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.MobileViewHomeOverride">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                navigator.popToFirstView();
            }
            protected function button2_clickHandler(event:MouseEvent):void {
                // Handle search
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
            click="button1_clickHandler(event);"/>
    </s:navigationContent>

    <s:actionContent>
        <s:Button icon="@Embed(source='assets/Search.png')"
            click="button2_clickHandler(event);"/>
    </s:actionContent>
</s:ViewNavigatorApplication>
```

此应用程序的第一个视图是 MobileViewHomeOverride 视图。 MobileViewHomeOverride 视图定义 Button 控件，用以导航到定义 “Search” 页面的另一个 View 容器：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\MobileViewHomeOverride.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Home View">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[

            // Navigate to the Search view.
            protected function button1_clickHandler(event:MouseEvent):void {
                navigator.pushView(SearchViewOverride);
            }
        ]]>
    </fx:Script>

    <s:Label text="Home View"/>
    <s:Button label="Search" click="button1_clickHandler(event)"/>
</s:View>
```

定义 “Search” 页面的 View 容器将重写 ActionBar 控件的标题区域和操作区域，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SearchViewOverride.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark">
    <s:layout>
        <s:VerticalLayout paddingTop="10"
            paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Perform a search.
            }
        ]]>
    </fx:Script>

    <!-- Override the title to insert a TextInput control. -->
    <s:titleContent>
        <s:TextInput text="Enter search text ..." textAlpha="0.5"
            width="250"/>
    </s:titleContent>

    <!-- Override the action area to insert a Search button. -->
    <s:actionContent>
        <s:Button label="Search" click="button1_clickHandler(event);"/>
    </s:actionContent>

    <s:Label text="Search View"/>
    <s:TextArea text="Search results appear here ..."
        height="75%"/>
</s:View>
```

下图显示了此视图的 ActionBar 控件：

由于 “Search” 视图不重写 ActionBar 控件的导航区域，因此导航区域仍显示 “Home” 按钮。

隐藏 ActionBar 控件

可以通过将 View.actionBarVisible 属性设置为 false，来隐藏任何视图中的 ActionBar 控件。默认情况下，actionBarVisible 属性的值为 true，表示显示 ActionBar 控件。

使用 ViewNavigator.hideActionbar() 方法可以对 ViewNavigator 所控制的所有视图隐藏 ActionBar 控件，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSectionNoAB.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.HomeView"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            protected function creationCompleteHandler(event:FlexEvent):void {
                // Access the ViewNavigator using the ViewNavigatorApplication.navigator property.
                navigator.hideActionBar();
            }
        ]]>
    </fx:Script>

</s:ViewNavigatorApplication>
```

您可以定义隐藏 ActionBar 或使 ActionBar 可见时 ActionBar 的自定义效果。默认情况下，ActionBar 在显示或隐藏时使用 Animate 效果。可以通过重写 ViewNavigator.createActionBarHideEffect() 和 ViewNavigator.createActionBarShowEffect() 方法来更改默认效果。在应用 ActionBar 隐藏效果后，将其 visible 和 includeInLayout 属性设置为 false，使视图布局中不再包括该控件。

在移动设备应用程序中使用滚动条

在移动设备应用程序中使用滚动条的注意事项

通常，如果屏幕的可见区域无法显示全部内容，应用程序将显示滚动条。使用 Scroller 控件可以在应用程序中添加滚动条。其他组件 （例如 Spark List 控件）支持滚动，因此您无需使用 Scroller 组件。有关更多信息，请参阅 Scrolling Spark containers。

滚动条的点击区域是指屏幕上放置鼠标以执行滚动操作的区域。在基于桌面或浏览器的应用程序中，点击区域是滚动条的可见区域。在移动设备应用程序中，即使屏幕的可见区域无法显示全部内容，也会隐藏滚动栏。隐藏滚动栏可使应用程序使用屏幕的全屏宽度和高度。

移动设备应用程序必须区分用户的意图，用户是要与控件交互 （例如选择 Button 控件）还是要进行滚动。移动设备应用程序中有关滚动条的一个注意事项是：Flex 组件经常更改其外观以响应用户交互。

例如，当用户按 Button 控件时，按钮的外观将会更改，以指示该按钮已被选定。用户释放按钮时，按钮将其外观更改回未选定的状态。但是，在进行滚动时，如果用户触摸屏幕上 Button 所在的位置，您并不希望按钮的外观发生变化。

Adobe 工程师 Steven Shongrunden 在 Saving scroll position between views in a mobile Flex Application 中介绍了一个处理滚动条的示例。

滚动术语

以下术语用于介绍移动设备应用程序中的滚动：

内容 对于可滚动组件（例如 Group 容器或 List 控件），指整个组件区域。根据屏幕大小和应用程序布局，可能会仅显示部分内容。

视口 当前可见的组件内容区域中的一部分。

拖动 用户触摸可滚动区域，然后移动手指以便内容沿手势移动时发生的触摸手势。

速度 拖动手势的移动速率和方向。以沿 X 和 Y 轴每毫秒的像素为单位。

抛开 用户在拖动手势达到特定速度后抬起手指时的拖动手势，此时可滚动内容仍继续移动。

跳动 拖动或抛开手势可以将可滚动组件的视口移至组件内容之外。然后，视口显示空白区域。当您释放手指或抛开速度达到零时，视口将跳跃回其静止点，并且视口有内容填充。移动随着视口接近静止点而减慢，因此能平稳停止。

移动设备应用程序中的滚动模式

可滚动组件（例如 List 和 Scroller）基于组件的 pageScrollingEnabled 和 scrollSnappingMode 属性设置支持不同类型的滚动。这些属性仅在 interactionMode 样式设置为 touch 时有效。

下表介绍了滚动模式：

pageScrollingEnabled	scrollSnappingMode	模式
false （默认值）	无 （默认值）	默认情况下，滚动以像素为基础。最终的滚动位置基于拖动或抛开手势位于任何像素位置。例如，您滚动 List 控件。当您抬起手指时滚动即结束，即使仅部分 List 项可见也是如此。
false	leadingEdge、 center、trailingEdge	滚动以像素为基础，但是内容基于 scrollSnappingMode 的值与最终位置对齐。 例如，可以通过将 scrollSnappingMode 设置为值 leadingEdge 来垂直滚动 List。List 控件与最终滚动位置对齐，其中顶部列表元素对齐到列表顶部。
true	none	滚动以页面为基础。可滚动组件的视口大小决定着页面大小。无论采用什么手势，一次只能滚动一个页面。 至少滚动组件可见区域的 50% 才能使该页面滚动到下一页。如果您滚动的区域少于 50%，则该组件仍保留在当前页面。或者，如果滚动速度足够高，会显示下一页。如果滚动速度不够高，组件仍保留在当前页面。 如果内容大小不是视口大小的整数倍，则会向最后一页中添加另外的内边距以使其完全适合视口的大小。
true	leadingEdge、 center、trailingEdge	滚动以页面为基础，但是组件基于 scrollSnappingMode 的值与最终位置对齐。为确保遵守对齐模式，滚动距离始终都不能完全等于页面大小。

移动设备应用程序中的滚动示例

在以下示例中，使用 Scroller 组件将 Group 容器封装在移动设备应用程序中。Group 容器将包含较大图像的 Image 控件作为其子代。通过将 Group 容器封装在 Scroller 中，您可以滚动图像：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SparkMobilePixelScrollerHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
    <s:Scroller width="200" height="200">
        <s:Group>
            <s:Image width="300" height="400"
                source="@Embed(source='../assets/logo.jpg')"/>
        </s:Group>
    </s:Scroller>
</s:View>
```

请注意，在该示例中，将忽略 pageScrollingEnabled 和 scrollSnappingMode 属性的任何设置。因此，该示例使用默认的像素滚动模式，并且您可以滚动到图像中的任何像素位置。

下一示例中显示的 List 控件将设置 pageScrollingEnabled 和 scrollSnappingMode 属性：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SparkMobilePageScrollHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Adobe Product List">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;

            protected function myList_changeHandler(event:IndexChangeEvent):void {
                navigator.pushView(views.ProductPricelView,myList.selectedItem);
            }

        ]]>
    </fx:Script>

    <s:List id="myList" labelField="Product"
        height="200" width="100%"
        borderVisible="true"
        scrollSnappingMode="leadingEdge"
        pageScrollingEnabled="true"
        change="myList_changeHandler(event);">
        <s:dataProvider>
            <s:ArrayCollection>
                <fx:Object Product="Adobe AIR" Price="11.99"/>
                <fx:Object Product="Adobe BlazeDS" Price="11.99"/>
                <fx:Object Product="Adobe ColdFusion" Price="11.99"/>
                <fx:Object Product="Adobe Flash Player" Price="11.99"/>
                <fx:Object Product="Adobe Flex" Price="Free"/>
                <fx:Object Product="Adobe LiveCycleDS" Price="11.99"/>
                <fx:Object Product="Adobe LiveCycle ES2" Price="11.99"/>
                <fx:Object Product="Open Source Media Framework"/>
                <fx:Object Product="Adobe Photoshop" Price="11.99"/>
                <fx:Object Product="Adobe Illustrator" Price="11.99"/>
                <fx:Object Product="Adobe Reader" Price="11.99"/>
                <fx:Object Product="Adobe Acrobat" Price="11.99"/>
                <fx:Object Product="Adobe InDesign" Price="Free"/>
                <fx:Object Product="Adobe Connect" Price="11.99"/>
                <fx:Object Product="Adobe Dreamweaver" Price="11.99"/>
                <fx:Object Product="Open Framemaker"/>
            </s:ArrayCollection>
        </s:dataProvider>
    </s:List>
</s:View>
```

该示例使用具有 leadingEdge 对齐设置的页面滚动。因此，当您滚动 List 时，该 List 一次可以滚动一页。页面更改时，控件与最终滚动位置对齐，其中顶部列表元素对齐到列表顶部。

与 StageText 相关的滚动注意事项

StageText 允许您在移动设备应用程序中使用本机文本输入，但不允许使用标准文本字段控件。但是，可滚动的容器容纳不下使用 StageText 的文本输入控件，如 TextInput 或 TextArea 控件。因此，要在可滚动容器中使用文本输入控件，请重新设计控件外观，以使其不使用 StageText。

对于没有使用 StageText 的 TextInput 和 TextArea 控件， Flex 附带了一些外观。请在可滚动容器中对这些控件使用下列外观：

- spark.skins.mobile.TextInputSkin 外观，用于不使用 StageText 的 TextInput。
- spark.skins.mobile.TextAreaSkin 外观，用于不使用 StageText 的 TextArea。

下例显示了一个在可滚动容器中使用 TextInput 和 TextArea 控件的 View 容器：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileStageTextScrollHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <!-- Create CSS class selectors that reference the skins
         that do not rely on StageText. -->
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        .myTextInputStyle {
            skinClass: ClassReference("spark.skins.mobile.TextInputSkin");
        }
        .myTextAreaStyle {
            skinClass: ClassReference("spark.skins.mobile.TextAreaSkin");
        }
    </fx:Style>

    <!-- Apply the class selectors to the TextInput and TextArea controls. -->
    <s:Scroller width="100%" height="100%">
        <s:VGroup height="250" width="100%"
                  paddingTop="10" paddingLeft="5" paddingRight="10">
            <s:HGroup verticalAlign="middle">
                <s:Label text="Text Input 1: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
                <s:Label text="Text Input 2: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
```

```
                <s:Label text="Text Input 3: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
                <s:Label text="Text Input 4: "
                    fontWeight="bold"/>
                <s:TextInput width="225"
                    styleName="myTextInputStyle"/>
            </s:HGroup>
            <s:HGroup verticalAlign="middle">
                <s:Label text="TextArea 1: "
                    fontWeight="bold"/>
                <s:TextArea width="225" height="100"
                    styleName="myTextAreaStyle "/>
            </s:HGroup>
        </s:VGroup>
    </s:Scroller>
</s:View>
```

事件和滚动条

Flex 组件通过事件来指示已发生了用户交互。然后，组件可以更改其外观或执行其它操作，来对用户交互做出响应。

应用程序开发人员依赖事件来处理用户交互。例如，通常使用 Button 控件的 click 事件来运行事件处理函数，以对用户选择按钮做出响应。当用户选择 List 中的某一项时，使用 List 控件的 change 事件来运行事件处理函数。

Flex 滚动机制依赖于 mouseDown 事件。这意味着滚动机制会侦听 mouseDown 事件以确定是否要启动滚动操作。

将用户手势解释为滚动操作

应用程序在一个可滚动容器中包含多个 Button 控件：

1 使用手指按下 Button 控件。按钮将分派 mouseDown 事件。

2 Flex 将对用户交互的响应延迟预定义的时间段。通过延迟期，可以确保用户正在选择按钮而不是尝试滚动屏幕。

   如果在延迟期间，您移动手指的量多于预定义量，则 Flex 将该手势解释为滚动操作。手指必须移动约 0.08 英寸的距离，该手势才会被解释为滚动。此距离大约相当于 252 DPI 设备上的 20 个像素。

   由于您在延迟期结束前移动了手指，所以 Button 控件不会识别该交互。该按钮不会分派事件或更改其外观。

3 延迟期结束后， Button 控件将识别用户交互。该按钮将更改其外观，以指示其已被选定。

   使用控件的 touchDelay 属性配置延迟的持续时间。默认值为 100 毫秒。如果将 touchDelay 属性设置为 0，则无延迟，且滚动会立即启动。

4 延迟期到期且 Flex 已分派鼠标事件后，移动手指的距离可以大于 20 像素。 Button 控件将恢复为正常状态，并启动滚动操作。

   在这种情况下，由于延迟期结束，因此按钮会更改其外观。但是，一旦移动手指的距离大于 20 像素，即使延迟期到期后，Flex 也会将手势解释为滚动操作。

注：Flex 组件除支持鼠标事件外，还支持许多不同类型的事件。使用组件时，您确定应用程序对这些事件的反应方式。对于 mouseDown 事件，用户的预期行为不明确。用户可能要与组件交互，或者要进行滚动。由于存在此不明确性，因此 Adobe 建议侦听 click 或 mouseUp 事件，而不是 mouseDown 事件。

处理滚动事件外观和项呈示器中的文本控件

要在滚动操作开始时发出信号，分派 mouseDown 事件的组件将分派冒泡的 touchInteractionStarting 事件。如果未取消该事件，组件将分派冒泡的 touchInteractionStart 事件。

当组件检测到 touchInteractionStart 事件时，不得尝试响应用户手势。例如，当 Button 控件检测到 touchInteractionStart 事件时，将关闭基于初始 mouseDown 事件而设置的任何可视指示符。

如果组件不希望启动滚动，组件可以在 touchInteractionStarting 事件的事件处理函数中调用 preventDefault() 方法。

完成滚动操作之后，分派 mouseDown 事件的组件将分派冒泡的 touchInteractionEnd 组件。

基于初始触摸点的滚动行为

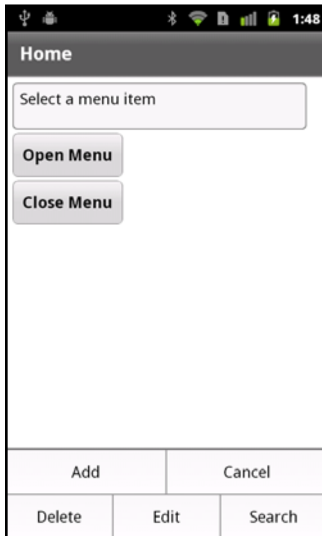
下表说明了根据初始触摸点位置处理滚动的方式：

选定项	行为
空白空间、 不可编辑的文本、 不可选择的文本	所有组件都不识别该手势。直到用户移动触摸点的距离大于 20 个像素时， Scroller 才会启动滚动。
List 控件中的项	延迟期结束后，选定项的项呈示器将显示更改为选定状态。但是，只要用户移动的距离大于 20 像素，项就会将其外观更改为正常状态并启动滚动。
Button、 CheckBox、 RadioButton、 DropDownList	延迟期结束后，将显示其 mouseDown 状态。但是，如果用户移动触摸点的距离大于 20 个像素，控件则将其外观更改为正常状态并启动滚动。
List 项呈示器中的 Button 组件	项呈示器从不加亮。 Button 或 Scroller 处理手势，与正常的 Button 情况相同。

定义移动设备应用程序中的菜单

ViewMenu 容器可以定义移动设备应用程序中位于 View 容器底部的菜单。每个 View 容器定义该视图专用的菜单。

下图显示的是某个应用程序中的 ViewMenu 容器：



ViewMenu 容器定义具有单个层次结构菜单按钮的菜单。即，您无法创建具有子菜单的菜单。

ViewMenu 容器的子代定义为 ViewMenuItem 控件。每个 ViewMenuItem 控件代表菜单中的单个按钮。

用户与 ViewMenu 容器的交互

可以使用移动设备设备上的硬件菜单键来打开菜单。也可以通过编程将其打开。

选择某个菜单按钮将关闭整个菜单。用户选择菜单按钮时， ViewMenuItem 控件会分派 click 事件。

菜单处于打开状态时，按设备的后退或菜单按钮可以关闭菜单。如果在菜单之外的任意位置按屏幕，菜单也会关闭。

插入标记是当前具有焦点的菜单按钮。使用设备的五向控件或箭头键可更改插入标记。按设备的输入键或五向控件可选择插入标记项并关闭菜单。

在移动设备应用程序中创建菜单

使用 View.viewMenuItems 属性为视图定义菜单。View.viewMenuItems 属性采用 ViewMenuItem 控件的矢量，如以下示例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\ViewMenuHome.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Home">

    <fx:Script>
        <![CDATA[
            // The event listener for the click event.
            private function itemClickInfo(event:MouseEvent):void {
                switch (event.currentTarget.label) {
                    case "Add" :
                        myTA.text = "Add selected";
                        break;
                    case "Cancel" :
                        myTA.text = "Cancel selected";
                        break;
                    case "Delete" :
                        myTA.text = "Delete selected";
                        break;
                    case "Edit" :
                        myTA.text = "Edit selected";
                        break;
                    case "Search" :
                        myTA.text = "Search selected";
                        break;
                    default :
                        myTA.text = "Error";
                }
            }
        ]]>
    </fx:Script>

    <s:viewMenuItems>
        <s:ViewMenuItem label="Add" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Cancel" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Delete" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Edit" click="itemClickInfo(event);"/>
        <s:ViewMenuItem label="Search" click="itemClickInfo(event);"/>
    </s:viewMenuItems>

    <s:VGroup paddingTop="10" paddingLeft="10">
        <s:TextArea id="myTA" text="Select a menu item"/>
        <s:Button label="Open Menu"
            click="mx.core.FlexGlobals.topLevelApplication.viewMenuOpen=true;"/>
        <s:Button label="Close Menu"
            click="mx.core.FlexGlobals.topLevelApplication.viewMenuOpen=false;"/>
    </s:VGroup>
</s:View>
```

在该示例中，使用 View.viewMenuItems 属性添加了五个菜单项，其中每个菜单项均由一个 ViewMenuItem 控件来表示。每个 ViewMenuItem 控件使用 label 属性指定出现在该项的菜单中的文本。

请注意，您没有明确定义 ViewMenu 容器。 View 容器自动创建 ViewMenu 容器的实例，以保存 ViewMenuItem 控件。

使用 **ViewMenuItem** 控件的 icon 样式

ViewMenuItem 控件定义您可以用于包含图像的 icon 样式属性。可以将 icon 样式与 label 属性配合使用，也可以不配合使用。

处理 **ViewMenuItem** 控件的 click 事件

每个 ViewMenuItem 控件也为 click 事件定义一个事件处理函数。用户选择项目时， ViewMenuItem 控件分派 click 事件。在此示例中，所有菜单项使用同一个事件处理函数。但是，可以选择为每个 click 事件定义一个单独的事件处理函数。

以编程方式打开 **ViewMenuItem** 控件

可以使用设备上的硬件菜单键来打开菜单。该应用程序也定义了两个 Button 控件，用于以编程方式打开和关闭菜单。

要以编程方式打开菜单，请将应用程序容器的 viewMenuOpen 属性设置为 true。要关闭菜单，请将该属性设置为 false。viewMenuOpen 属性在 ViewNavigatorApplicationBase 类、 ViewNavigatorApplication 的基类和 TabbedViewNavigatorApplication 容器中进行定义。

将外观应用到 ViewMenu 和 ViewMenuItem 组件

使用外观来控制 ViewMenu 和 ViewMenuItem 组件的外观。默认 ViewMenu 外观类是 spark.skins.mobile.ViewMenuSkin。默认 ViewMenuItem 外观类是 spark.skins.mobile.ViewMenuItemSkin。

博客 Daniel Demmel 介绍了如何设置 ViewMenu 控件的外观以使其像姜饼黑色。

外观类使用正常、已关闭和已禁用等外观状态来控制外观的外观。外观也定义过渡，以控制视图状态更改时菜单的外观。

有关更多信息，请参阅第 141 页的 " 移动设备外观设计的基础知识 "。

设置 ViewMenu 容器的布局

ViewMenuLayout 类定义视图菜单的布局。菜单可具有多行，具体取决于菜单项的数量。

ViewMenuItem 布局规则

ViewMenuLayout 类的 requestedMaxColumnCount 属性定义一行中所含菜单项的最大数量。默认情况下，requestedMaxColumnCount 属性设置为三。

以下规则定义 ViewMenuLayout 类执行布局的方式：

- 如果定义三个或更少的菜单项，其中 requestedMaxColumnCount 属性的默认值为三，则菜单项会显示在单行中。每个菜单项的大小相同。

  如果定义四个或更多的菜单项 （即，定义的菜单项比 requestedMaxColumnCount 属性指定的菜单项多），则 ViewMenu 容器创建多行。

- 如果菜单项的数量可被 requestedMaxColumnCount 属性整除，则每行包含相同数量的菜单项。每个菜单项的大小相同。

  例如，将 requestedMaxColumnCount 属性的默认值设置为 3，并且定义 6 个菜单项。菜单将显示两行，且每行包含三个菜单项。

- 如果菜单项的数量不能被 requestedMaxColumnCount 属性整除，则行可以包含不同数量的菜单项。菜单项的大小取决于行中包含的菜单项数。

  例如，将 requestedMaxColumnCount 属性的默认值设置为 3，并且定义 8 个菜单项。菜单将显示三行。第一行包含两个菜单项。第二行和第三行各包含三个菜单项。

创建自定义 **ViewMenuItem** 布局

ViewMenuLayout 类包含用于修改菜单项数与每行中的默认菜单项数之间的差距的属性。您也可以通过创建自己的布局类来为菜单创建自定义布局。

默认情况下， spark.skins.mobile.ViewMenuSkin 类为 ViewMenu 容器定义外观。要将自定义的 ViewMenuLayout 类应用于 ViewMenu 容器，请为 ViewMenu 容器定义新的外观类。

默认 ViewMenuSkin 类包含一个名为 contentGroup 的 Group 容器的定义，如下例所示：

```
...
    <s:Group id="contentGroup" left="0" right="0" top="3" bottom="2"
        minWidth="0" minHeight="0">
        <s:layout>
            <s:ViewMenuLayout horizontalGap="2" verticalGap="2" id="contentGroupLayout"
                requestedMaxColumnCount="3"
                requestedMaxColumnCount.landscapeGroup="6"/>
        </s:layout>
    </s:Group>
...
```

您的外观类也必须定义名为 contentGroup 的容器。该容器使用 layout 属性指定您的自定义布局类。

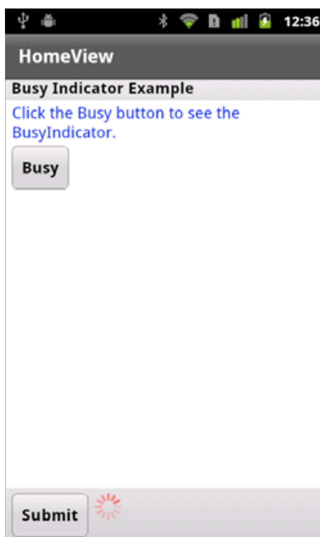
之后，可以在应用程序中应用自定义外观类，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\ViewMenuSkin.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.ViewMenuHome">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|ViewMenu {
            skinClass: ClassReference("skins.MyVMSkin");
        }
    </fx:Style>
</s:ViewNavigatorApplication>
```

为移动设备应用程序中耗时的活动显示忙碌指示符

Spark BusyIndicator 控件显示一个含有 12 根轮辐的旋转微调框。可以使用 BusyIndicator 控件提供可视指示，指示正在执行耗时操作。

下图中，在 “ 提交 ” 按钮旁显示了 Spark Panel 容器控件栏区域中的 BusyIndicator 控件：



BusyIndicator 控件应在耗时操作执行期间可见。操作完成时，应隐藏该控件。

例如，可以在事件处理函数中创建 BusyIndicator 控件的实例，该事件处理函数可以是用于启动耗时进程的事件处理函数。在该事件处理函数中，调用 addElement() 方法以将该控件添加到容器中。进程完成后，调用 removeElement() 以将 BusyIndicator 控件从容器中删除。

还可以使用控件的 visible 属性将其显示和隐藏。在以下示例中，您将 BusyIndicator 控件添加至 View 容器中 Spark Panel 容器的控制栏区域：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\SimpleBusyIndicatorHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">

    <s:Panel id="panel" title="Busy Indicator Example"
        width="100%" height="100%">
        <s:controlBarContent>
            <s:Button label="Submit" />
            <s:BusyIndicator id="bi"
                visible="false"
                symbolColor="red"/>
        </s:controlBarContent>

        <s:VGroup left="10" right="10" top="10" bottom="10">
            <s:Label width="100%" color="blue"
                text="Click the Busy button to see the BusyIndicator."/>
            <s:Button label="Busy"
                click="{bi.visible = !bi.visible}" />
        </s:VGroup>
    </s:Panel>
</s:View>
```

在该示例中， BusyIndicator 控件的 visible 属性最初设置为 false 以将其隐藏。单击 “ 忙 ” 按钮，将 visible 属性设置为 true 以显示控件。

BusyIndicator 控件仅在可见时旋转。因此，当将 visible 属性设置为 false 时，控件不需要任何处理循环。

注：将 visible 属性设置为 false 会隐藏控件，但控件仍包含在其父容器的布局中。要从布局中排除控件，请将 visible 和 includeInLayout 属性设置为 false。

Spark BusyIndicator 控件不支持设置外观。但是，可以使用样式设置微调框的颜色和旋转间隔。在上面的示例中，您使用 symbolColor 属性设置指示符的颜色。

将切换开关添加到移动设备应用程序中

Spark ToggleSwitch 控件定义一个简单二进制开关。该控件包含缩略图和滑动缩略图的轨迹。

ToggleSwitch 控件与 ToggleButton 和 CheckBox 控件类似。所有这些控件都允许您在选定值和未选定值之间进行选择。

下图显示了应用程序中的 ToggleSwitch 开关：

ToggleSwitch 控件包含两个位置：选定位置和未选定位置。缩略图位于左边时，控件在未选定位置。缩略图位于右边时，控件在选定位置。在图中，开关位于未选定位置。

单击控件中的任意位置可切换其位置。还可以沿轨迹滑动缩略图以更改位置。释放缩略图时，它将移至最接近缩略图位置的位置（选定位置或未选定位置）。

默认情况下，标签 OFF 对应于未选定位置，标签 ON 对应于选定位置。

创建 ToggleSwitch 控件

用于定义上图中显示的 ToggleSwitch 控件的 View 容器如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\ToggleSwitchSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingTop="10" paddingLeft="5"/>
    </s:layout>
    <s:ToggleSwitch id="ts"
        slideDuration="1000"/>
    <s:Form>
        <s:FormItem label="Toggle Label: ">
            <s:Label text="{ts.selected ? 'ON' : 'OFF'}"/>
        </s:FormItem>
        <s:FormItem label="Toggle Position: ">
            <s:Label text="{ts.thumbPosition}"/>
        </s:FormItem>
    </s:Form>
</s:View>
```

在该示例中，基于缩略图位置在第一个 Label 控件中显示 ON 或 OFF。第二个 Label 控件将当前缩略图位置显示为 0.0（未选定）与 1.0 （选定）之间的值。

该示例还将 slideDuration 样式设置为 1000。该样式确定缩略图在选定位置与未选定位置之间滑动时的动画持续时间（毫秒）。

主应用程序文件如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\ToggleSwitchSimple.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.ToggleSwitchSimpleHomeView">
</s:ViewNavigatorApplication>
```

更改 ToggleSwitch 控件的默认 Callout

在上一示例中， ToggleSwitch 控件使用选定标签和未选定标签的默认值：OFF （未选定）和 ON （选定）。要自定义控件的标签或其它可视特性，将外观类定义为 spark.skins.mobile.ToggleSwitchSkin 的子类或创建您自己的外观类。

以下外观类将标签更改为 Yes 和 No：

```
package skins
// components\mobile\skins\MyToggleSwitchSkin.as
{
    import spark.skins.mobile.ToggleSwitchSkin;

    public class MyToggleSwitchSkin extends ToggleSwitchSkin
    {
        public function MyToggleSwitchSkin()
        {
            super();
            // Set properties to define the labels
            // for the selected and unselected positions.
            selectedLabel="Yes";
            unselectedLabel="No";
        }
    }
}
```

以下 View 容器使用该外观类：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\ToggleSwitchSkinHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingTop="10" paddingLeft="5"/>
    </s:layout>

    <s:ToggleSwitch id="ts"
        slideDuration="1000"
        skinClass="skins.MyToggleSwitchSkin"/>

    <s:Form>
        <s:FormItem label="Toggle Label: ">
            <s:Label text="{ts.selected ? 'Yes' : 'No'}"/>
        </s:FormItem>
        <s:FormItem label="Toggle Position: ">
            <s:Label text="{ts.thumbPosition}"/>
        </s:FormItem>
    </s:Form>
</s:View>
```

将 callout 容器添加到移动设备应用程序中

在移动设备应用程序中，callout 是在应用程序顶部弹出的容器。该容器可以容纳一个或多个组件，并且支持不同类型的布局。

callout 容器可以是模态或非模态容器。模态容器在其关闭之前接受所有的键盘和鼠标输入。非模态容器允许应用程序中的其它组件在该容器处于打开状态时接受输入。

Flex 提供了两个可用于将 callout 容器添加到移动设备应用程序中的组件：CalloutButton 和 Callout。

使用 CalloutButton 控件创建 callout 容器

CalloutButton 控件提供了一种创建 callout 容器的简单方式。通过该组件，您可以定义显示在 callout 中的组件和设置容器布局。

在移动设备应用程序中选择 CalloutButton 控件时，该控件将打开 callout 容器。 Flex 会自动绘制一个从 callout 容器指向 CalloutButton 控件的箭头，如下图所示：

以下示例说明了用于创建上图中显示的 CalloutButton 的移动设备应用程序：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutButtonSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009" 
    xmlns:s="library://ns.adobe.com/flex/spark" 
    title="HomeView">
    <s:layout>
        <s:VerticalLayout 
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <s:Label text="Select the button to open the callout"/>

    <s:CalloutButton id="myCB"
        horizontalPosition="end" 
        verticalPosition="after" 
        label="Open callout">
        <s:calloutLayout>
            <s:HorizontalLayout/>
        </s:calloutLayout>

        <!-- Define buttons that appear in the callout. -->
        <s:Button label="OK"
            click="myCB.closeDropDown();"/>
        <s:Button label="Cancel"
            click="myCB.closeDropDown();"/>
    </s:CalloutButton>
</s:View>
```

CalloutButton 控件定义显示在 callout 容器内的两个 Button 控件。CalloutButton 控件还指定将 HorizontalLayout 用作 callout 容器的布局。默认情况下，该容器使用 BasicLayout。

使用 **CalloutButton** 控件打开和关闭 **Callout** 容器

当用户选择 CalloutButton 控件或您调用 CalloutButton.openDropDown() 方法时，callout 容器将打开。horizontalPosition 和 verticalPosition 属性确定 callout 容器相对于 CalloutButton 控件的位置。有关示例，请参阅第 77 页的 “ 确定 callout 容器的大小和位置 ”。

CalloutButton 打开的 callout 容器始终都是非模态的。这意味着应用程序中的其它组件可以在 callout 处于打开状态时接受输入。使用 Callout 容器创建模态 callout。

在您单击 callout 容器以外的位置或调用 CalloutButton.closeDropDown() 方法之前，callout 容器一直处于打开状态。在该示例中，对于 callout 容器中两个 Button 控件的 click 事件，您在事件处理函数中调用了 closeDropDown() 方法。

使用 Callout 容器创建 callout

CalloutButton 控件将 callout 容器以及打开和关闭 callout 必需的所有逻辑封装在一个控件中。则 CalloutButton 控件成为 callout 容器的主机。

您还可以在移动设备应用程序中使用 Callout 容器。 Callout 容器的好处是它不与单个主机关联，因此可以重用于应用程序中的任何位置。

通常，为了响应事件，使用 Callout.open() 和 Callout.close() 方法打开 Callout 容器。调用 open() 方法时，可以传递可选参数以指定调用容器为模态容器。默认情况下，调用容器为非模态容器。

调用容器的位置相对于主机组件。 horizontalPosition 和 verticalPosition 属性确定容器相对于主机的位置。有关示例，请参阅第 77 页的 " 确定 callout 容器的大小和位置 "。

由于它为弹出窗口，因此不能创建一个 Callout 容器作为应用程序的正常 MXML 布局代码的一部分。应在 MXML 文件中将 Callout 容器定义为自定义 MXML 组件。

在以下示例中，在应用程序的 comps 目录的 MyCallout.mxml 文件中定义 Callout 容器：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\comps\MyCallout.mxml -->
<s:Callout xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    horizontalPosition="start"
    verticalPosition="after">

    <s:VGroup
        paddingTop="10" paddingLeft="5" paddingRight="10">

        <s:HGroup verticalAlign="middle">
            <s:Label text="First Name: "
                fontWeight="bold"/>
            <s:TextInput width="225"/>
        </s:HGroup>

        <s:HGroup verticalAlign="middle">
            <s:Label text="Last Name: "
                fontWeight="bold"/>
            <s:TextInput width="225"/>
        </s:HGroup>

        <s:HGroup>
            <s:Button label="OK" click="close();"/>
            <s:Button label="Cancel" click="close();"/>
        </s:HGroup>
    </s:VGroup>
</s:Callout>
```

MyCallout.mxml 定义一个简单的弹出窗口以允许用户输入名和姓。请注意，按钮调用 close() 方法关闭 callout 以响应 click 事件。

以下示例说明了用于打开 MyCallout.mxml 的 View 容器以响应 click 事件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import comps.MyCallout;

            // Event handler to open the Callout component.
            protected function button1_clickHandler(event:MouseEvent):void {
                var myCallout:MyCallout = new MyCallout();
                // Open as a modal callout.
                myCallout.open(calloutB, true);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select the button to open the callout"/>
    <s:Button id="calloutB"
        label="Open Callout container"
        click="button1_clickHandler(event);"/>
</s:View>
```

首先，将 MyCallout.mxml 组件导入至应用程序。为了响应 click 事件，名为 calloutB 的按钮创建了一个 MyCallout.mxml 实例，然后调用 open() 方法。

open() 方法指定两个参数。第一个参数指定 calloutB 是 callout 的主机组件。因此， callout 在应用程序中将自身定位在相对于 calloutB 位置的位置。第二个参数为 true 以创建模态 callout。

定义内联 callout 容器

您无需在单独文件中定义 Callout 容器。以下示例使用 <fx:Declaration> 标签将其定义为 View 容器的内联组件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutInlineHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            // Event handler to open the Callout component.
            protected function button1_clickHandler(event:MouseEvent):void {
                var myCallout:MyCallout = new MyCallout();
                // Open as a modal callout.
                myCallout.open(calloutB, true);
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <fx:Component className="MyCallout">
```

```
            <s:Callout
                horizontalPosition="end"
                verticalPosition="after">
                <s:VGroup
                    paddingTop="10" paddingLeft="5" paddingRight="10">
                    <s:HGroup verticalAlign="middle">
                        <s:Label text="First Name: "
                                 fontWeight="bold"/>
                        <s:TextInput width="225"/>
                    </s:HGroup>
                    <s:HGroup verticalAlign="middle">
                        <s:Label text="Last Name: "
                                 fontWeight="bold"/>
                        <s:TextInput width="225"/>
                    </s:HGroup>
                    <s:HGroup>
                        <s:Button label="OK" click="close();"/>
                        <s:Button label="Cancel" click="close();"/>
                    </s:HGroup>
                </s:VGroup>
            </s:Callout>
        </fx:Component>
    </fx:Declarations>

    <s:Label text="Select the button to open the callout"/>
    <s:Button id="calloutB"
        label="Open Callout container"
        click="button1_clickHandler(event);"/>
</s:View>
```

从 Callout 容器中传回数据

使用 Callout 容器的 close() 方法将数据传回主应用程序。 close() 方法具有如下签名:

```
public function close(commit:Boolean = false, data:*):void
```

其中:

- 如果应用程序应提交返回的数据，则 commit 包含 true。
- data 指定返回的数据。

调用 close() 方法会分派 close 事件。与 close 事件关联的事件对象是类型为 spark.events.PopUpEvent 的对象。PopUpEvent 类定义两个属性：commit 和 data，这两个属性包含 close() 方法的相应参数值。在 close 事件的事件处理函数中使用这些属性，以便检查从 callout 返回的任何数据。

callout 容器是 SkinnablePopUpContainer 类的子类，它使用相同的机制将数据传回主应用程序中。有关从 SkinnablePopUpContainer 容器传回数据的示例，请参阅从 Spark SkinnablePopUpContainer 容器传回数据。

以下示例将修改如上所示的 Callout 组件以返回名和姓值:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\comps\MyCalloutPassBack.mxml -->
<s:Callout xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    horizontalPosition="start"
    verticalPosition="after">

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;

            public var retData:String = new String();

            // Event handler for the click event of the OK button.
            protected function clickHandler(event:MouseEvent):void {
                //Create the return data.
                retData = firstName.text + " " + lastName.text;
                // Close the Callout.
                // Set the commit argument to true to indicate that the
                // data argument contains a valid value.
                close(true, retData);
            }

        ]]>
    </fx:Script>

    <s:VGroup
        paddingTop="10" paddingLeft="5" paddingRight="10">
        <s:HGroup verticalAlign="middle">
            <s:Label text="First Name: "
                     fontWeight="bold"/>
            <s:TextInput id="firstName" width="225"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Last Name: "
                     fontWeight="bold"/>
            <s:TextInput id="lastName" width="225"/>
        </s:HGroup>
        <s:HGroup>
            <s:Button label="OK" click="clickHandler(event);"/>
            <s:Button label="Cancel" click="close();"/>
        </s:HGroup>
    </s:VGroup>
</s:Callout>
```

在该示例中，创建 String 返回名和姓，以对用户选择 “ 确定 ” 按钮做出响应。

View 容器然后使用 Callout 中的 close 事件以显示返回的数据：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutPassBackDataHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import comps.MyCalloutPassBack;
            import spark.events.PopUpEvent;

            public var myCallout:MyCalloutPassBack = new MyCalloutPassBack();

            // Event handler to open the Callout component.
            protected function clickHandler(event:MouseEvent):void {
                // Add an event handler for the close event to check for
                // any returned data.
                myCallout.addEventListener('close', closeHandler);
                // Open as a modal callout.
                myCallout.open(calloutB, true);
            }

            // Handle the close event from the Callout.
            protected function closeHandler(event:PopUpEvent):void {
                // If commit is false, no data is returned.
                if (!event.commit)
                    return;

                // Write the returned Data to the TextArea control.
                myTA.text = String(event.data);

                // Remove the event handler.
                myCallout.removeEventListener('close', closeHandler);
            }

        ]]>
    </fx:Script>

    <s:Label text="Select the button to open the callout"/>
    <s:Button id="calloutB"
        label="Open Callout container"
        click="clickHandler(event);"/>
    <s:TextArea id="myTA"/>
</s:View>
```

将 ViewNavigator 添加到 Callout 中

您可以在 Callout 容器中使用 ViewNavigator。通过 ViewNavigator 可以向 callout 中添加操作栏和多个视图。

例如，以下 View 打开了在文件 MyCalloutPassBackVN 中定义的 Callout 容器：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutPassBackDataHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import comps.MyCalloutPassBackVN;
            import spark.events.PopUpEvent;

            public var myCallout:MyCalloutPassBackVN = new MyCalloutPassBackVN();

            // Event handler to open the Callout component.
            protected function clickHandler(event:MouseEvent):void {
                myCallout.addEventListener('close', closeHandler);
                myCallout.open(calloutB, true);
            }

            // Handle the close event from the Callout.
            protected function closeHandler(event:PopUpEvent):void {
                if (!event.commit)
                    return;

                myTA.text = String(event.data);
                myCallout.removeEventListener('close', closeHandler);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select the Open button to open the callout"/>
    <s:TextArea id="myTA"/>
    <s:actionContent>
        <s:Button id="calloutB" label="Open"
            click="clickHandler(event);"/>
    </s:actionContent>
</s:View>
```

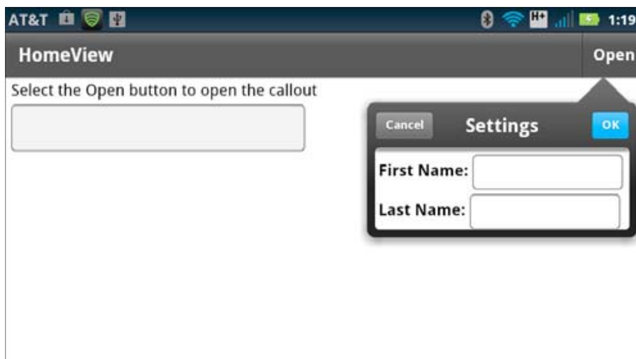
MyCalloutPassBackVN.mxml 文件定义用于容纳 ViewNavigator 容器的 Callout 容器:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\comps\MyCalloutVN.mxml -->
<s:Callout xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    contentBackgroundAppearance="none"
    horizontalPosition="start"
    verticalPosition="after">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexMouseEvent;
            import views.SettingsView;

            protected function done_clickHandler(event:MouseEvent):void {
                // Create an instance of SettingsView, and
                // initialize it as a copy of the current View of the ViewNavigator.
                var settings:SettingsView = (viewNav.activeView as SettingsView);

                // Create the String to represent the returned data.
                var retData:String = new String();
                // Initialze the String from the current View.
                retData = settings.firstName.text + " " + settings.lastName.text;
                // Close the Callout and return thhe data.
                this.close(true, retData);
            }
        ]]>
    </fx:Script>

    <s:ViewNavigator id="viewNav" width="100%" height="100%" firstView="views.SettingsView">
        <s:navigationContent>
            <s:Button label="Cancel" click="close(false)"/>
        </s:navigationContent>
        <s:actionContent>
            <s:Button id="done" label="OK" emphasized="true" click="done_clickHandler(event);"/>
        </s:actionContent>
    </s:ViewNavigator>
</s:Callout>
```

在 MyCalloutPassBackVN.mxml 中，指定 ViewNavigator 的第一个视图是 SettingsView。 SettingsView 定义针对用户名字和姓氏的 TextInput 控件。用户选择 " 确定 " 按钮时，将关闭 Callout 并将任何返回数据传回 MyCalloutPassBackVN。

注：当 ViewNavigator 出现在 Callout 容器中时， ActionBar 会具有透明的背景颜色。在此例中，您将 Callout 容器上的 contentBackgroundAppearance 设置为 none。此设置会防止透明 ActionBar 区域中出现 Callout 的默认白色 contentBackgroundColor。

下图显示在 Callout 处于打开状态下的应用程序：



SettingsView.mxml 如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\SettingsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Settings">

    <s:VGroup
        paddingTop="10" paddingLeft="5" paddingRight="10">
        <s:HGroup verticalAlign="middle">
            <s:Label text="First Name: "
                fontWeight="bold"/>
            <s:TextInput id="firstName" width="225"/>
        </s:HGroup>
        <s:HGroup verticalAlign="middle">
            <s:Label text="Last Name: "
                fontWeight="bold"/>
            <s:TextInput id="lastName" width="225"/>
        </s:HGroup>
    </s:VGroup>
</s:View>
```

注：由 Callout 容器中的 ViewNavigator 定义的 ActionBar 具有透明背景。默认情况下，从一个 View 到另一个 View 的过渡效果可以正确显示在 Callout 中。但是，如果您指定非默认的过渡效果 （例如 CrossFadeViewTransition 或 ZoomViewTransition），两个视图的 ActionBar 区域可能会重叠。要解决该问题，为使用非透明背景的 ActionBar 和 Callout 创建一个自定义外观类。

确定 callout 容器的大小和位置

CalloutButton 控件和 Callout 容器使用以下两个属性指定 callout 容器相对于其主机的位置：horizontalPosition 和 verticalPosition。这两个属性可以具有下列值：“before”、“start”、“middle”、“end”、“after” 和 “auto” （默认值）。

例如，按照如下所示的方式设置这两个属性：

```
horizontalPosition="before"
verticalPosition="after"
```

callout 容器将在主机组件的左下方打开。如果按照如下所示的方式设置这两个属性：

```
horizontalPosition="middle"
verticalPosition="middle"
```

callout 容器将在主机组件的顶部打开， callout 的中心与主机组件的中心对齐。

绘制一个从 **Callout** 指向主机的箭头

对于除 horizontalPosition 与 verticalPosition 属性的五种组合之外的所有位置， callout 都会绘制一个指向主机的箭头。当 Callout 位于主机中部上方居中以及角落时，不会显示箭头。以下组合不显示箭头：

```
// Centered
horizontalPosition="middle"
verticalPosition="middle"

// Upper-left corner
horizontalPosition="before"
verticalPosition="before"

// Lower-left corner
horizontalPosition="before"
verticalPosition="after"

// Upper-right corner
horizontalPosition="after"
verticalPosition="before"

// Lower-right corner
horizontalPosition="after"
verticalPosition="after"
```

对于 Callout 容器，horizontalPosition 和 verticalPosition 属性还确定 Callout.arrowDirection 只读属性的值。callout 容器相对于主机的位置确定 arrowDirection 属性的值。可能的值包括 "up"、 "left" 和其它。

Callout.arrow 外观部件使用 arrowDirection 属性的值基于 callout 位置绘制箭头。

管理 callout 容器的内存

使用 callout 容器时的一个注意事项是如何管理 callout 使用的内存。例如，如果希望减少应用程序使用的内存，则在每次打开应用程序时创建 callout 的实例。然后，在应用程序关闭时破坏 callout。但是，确保删除 callout （尤其是事件处理函数）的所有引用，否则 callout 不会被破坏。

或者，如果 callout 容器相对比较小，则可以在应用程序中多次重用相同的 callout。在该配置中，应用程序创建单个 callout 实例。然后，它重用该实例， callout 在使用的间隔保留在内存中。该配置减少了在应用程序中的执行时间，因为应用程序无需在每次打开时重新创建 callout。

使用 **CalloutButton** 控件管理内存

要配置 CalloutButton 控件使用的 callout，请设置 CalloutButton.calloutDestructionPolicy 属性。"auto" 的值配置在 callout 关闭时将其破坏的控件。 "never" 的值配置将 callout 缓存在内存中的控件。

使用 **Callout** 容器管理内存

Callout 容器不定义 calloutDestructionPolicy 属性，而是按照您在应用程序中创建 callout 容器实例的方式控制其内存使用。在以下示例中，将在每次打开 callout 容器时创建它的实例：

```
protected function button1_clickHandler(event:MouseEvent):void {
    // Create a new instance of the callout container every time you open it.
    var myCallout:MyCallout = new MyCallout();
    myCallout.open(calloutB, true);
}
```

或者，也可以定义在每次打开 callout 容器时重用的单个 callout 容器实例：

```
// Create a single instance of the callout container.
public var myCallout:MyCallout = new MyCallout();

protected function button1_clickHandler(event:MouseEvent):void {
    myCallout.open(calloutB, true);
}
```

定义移动设备应用程序中的过渡效果

Spark 视图过渡效果用于定义屏幕上发生 View 容器切换时的效果。在切换视图时，将通过应用动画来表现过渡效果。使用过渡效果可以使移动设备应用程序的界面更为引人注目。

默认情况下，现有 View 容器滑出屏幕，而新的视图滑进屏幕。或者，您也可以自定义切换效果。例如，应用程序在某个 View 容器中定义一个仅包含少量字段的表单，而在随后的 View 容器中显示更多字段。您可以在切换视图时采用翻转或淡化的过渡效果，而不采用滑动效果。

Flex 支持以下视图过渡类，可以在切换 View 容器时使用这些类：

过渡效果	说明
CrossFadeViewTransition	在现有视图与新视图之间采用交叉淡化过渡。现有视图淡出，同时新视图淡入。
FlipViewTransition	在现有视图与新视图之间采用翻转过渡。可以定义翻转方向和类型。
SlideViewTransition	在现有视图与新视图之间采用滑动过渡。现有视图滑出，同时新视图滑入。可以控制滑动方向和类型。此过渡效果是 Flex 使用的默认视图过渡效果。
ZoomViewTransition	在现有视图与新视图之间采用缩放过渡。可以缩小现有视图，或者放大新视图。

注：移动设备应用程序中的视图过渡效果与标准 Flex 过渡效果不相关。标准 Flex 过渡效果用于定义在状态变化期间所表现的效果。 ViewNavigator 容器的导航操作将触发视图过渡效果。视图过渡效果不能在 MXML 中定义，并且与状态无关。

应用过渡效果

切换活动 View 容器时，可以应用过渡效果。由于视图过渡发生在切换 View 容器时，因此可以通过 ViewNavigator 容器来控制过渡效果。

例如，可以使用 ViewNavigator 类的以下方法来切换当前视图：

- pushView()
- popView()
- popToFirstView()
- popAll()
- replaceView()

这些方法都接受一个可选参数，该参数用于定义切换视图时所执行的过渡效果。

也可以使用设备上的硬件导航键 （例如后退按钮）来切换当前视图。使用硬件键切换视图时， ViewNavigator 使用 ViewNavigator.defaultPopTransition 和 ViewNavigator.defaultPushTransition 属性所定义的默认过渡效果。默认情况下，这些属性指定使用 SlideViewTransition 类。

以下示例显示的主应用程序文件将 defaultPopTransition 和 defaultPushTransition 属性初始化为使用 FlipViewTransition：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkViewTrans.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainViewTrans"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.transitions.FlipViewTransition;

            // Define a flip transition.
            public var flipTrans:FlipViewTransition = new FlipViewTransition();

            // Set the default push and pop transitions of the navigator
            // to use the flip transition.
            protected function creationCompleteHandler(event:FlexEvent):void {
                navigator.defaultPopTransition = flipTrans;
                navigator.defaultPushTransition = flipTrans;
            }

            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                // Use the default pop view transition defined by
                // the ViewNavigator.defaultPopTransition property.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
            click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

第一个视图 EmployeeMainViewTrans.mxml 定义一个 CrossFadeViewTransition。当切换到 EmployeeView 时，它将 CrossFadeViewTransition 作为参数传递给 pushView() 方法：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeMainViewTrans.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employees">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;
            import spark.transitions.CrossFadeViewTransition;

            // Define two transitions: a cross fade and a flip.
            public var xFadeTrans:CrossFadeViewTransition = new CrossFadeViewTransition();

            // Use the cross fade transition on a push(),
            // with a duration of 100 ms.
            protected function myList_changeHandler(event:IndexChangeEvent):void {
                xFadeTrans.duration = 1000;
                navigator.pushView(views.EmployeeView, myList.selectedItem, null, xFadeTrans);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select an employee name"/>
    <s:List id="myList"
        width="100%" height="100%"
        labelField="firstName"
        change="myList_changeHandler(event);">
        <s:ArrayCollection>
            <fx:Object firstName="Bill" lastName="Smith" companyID="11233"/>
            <fx:Object firstName="Dave" lastName="Jones" companyID="13455"/>
            <fx:Object firstName="Mary" lastName="Davis" companyID="11543"/>
            <fx:Object firstName="Debbie" lastName="Cooper" companyID="14266"/>
        </s:ArrayCollection>
    </s:List>
</s:View>
```

EmployeeView 在 EmployeeView.mxml 文件中定义，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employee View">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

将过渡效果应用于 ActionBar 控件

默认情况下，视图切换的过渡效果中不包括 ActionBar。无论指定何种过渡效果，在切换视图时 ActionBar 控件都使用滑动过渡。要在切换视图时的过渡效果中包含 ActionBar，请将过渡类的 transitionControlsWithContent 属性设置为 true。

结合使用缓动类与过渡效果

过渡效果分两个阶段执行：加速阶段和减速阶段。可以使用缓动类来更改过渡效果的加速和减速属性。通过缓动，可以产生更加实际的加速和减速速率。也可以使用缓动类来创建跳跃效果，或控制其他类型的运动。

Flex 在 spark.effects.easing 包中提供 Spark 缓动类。此数据包中包括最常用缓动类型的类，其中包括 Bounce、 Linear 和 Sine 缓动。有关这些类的用法的更多信息，请参阅 Using Spark easing classes。

在下面的示例中，对上一部分所定义的应用程序进行了修改。此版本在 FlipViewTransition 中添加了 Bounce 缓动类：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkViewTransEasier.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainViewTransEaser"
    creationComplete="creationCompleteHandler(event);">

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.transitions.FlipViewTransition;

            // Define a flip transition.
            public var flipTrans:FlipViewTransition = new FlipViewTransition();

            // Set the default push and pop transitions of the navigator
            // to use the flip transition.
            // Specify the Bounce class as the easer for the flip.
            protected function creationCompleteHandler(event:FlexEvent):void {
                flipTrans.easer = bounceEasing;
                flipTrans.duration = 1000;
                navigator.defaultPopTransition = flipTrans;
                navigator.defaultPushTransition = flipTrans;
            }

            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                // Use the default pop view transition defined by
                // the ViewNavigator.defaultPopTransition property.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

    <fx:Declarations>
        <s:Bounce id="bounceEasing"/>
    </fx:Declarations>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
                  click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

要查看跳跃效果，请确保在设备上使用 “ 返回 ” 按钮。

视图过渡效果的生命周期

视图过渡效果的执行过程分两个主要阶段：准备阶段和执行阶段。

过渡效果使用三个方法定义准备阶段。这三个方法的调用顺序如下:

1 captureStartValues()

当 ViewNavigator 创建了新视图但尚未验证新视图或更新 ActionBar 控件和选项卡栏的内容时，将调用此方法。使用此方法可以捕获在过渡中发挥作用的各组件的起始值。

2 captureEndValues()

当新视图彻底验证完毕并且 ActionBar 控件和选项卡栏反映出新视图的状态时，将调用此方法。过渡可以使用此方法来捕获所需要的来自新视图的任何值。

3 prepareForPlay()

调用此方法时，过渡效果将初始化用于以动画效果呈现过渡组件的效果实例。

当 ViewNavigator 调用过渡的 play() 方法时，执行阶段开始。此时，新视图已创建并验证完毕， ActionBar 控件和选项卡栏已完成初始化。过渡将分派 start 事件，并通过调用效果的 play() 方法来调用准备阶段所创建的任何效果实例。

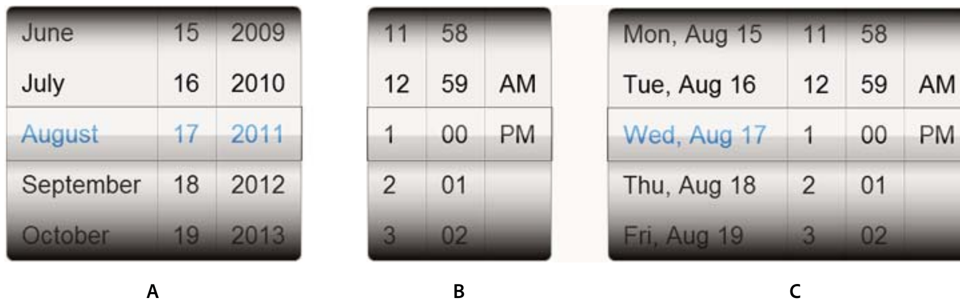
完成视图过渡后，过渡将分派 end 事件。过渡效果的基类 ViewTransitionBase 定义 transitionComplete() 方法，您可以调用该方法来分派 end 事件。重要的是，在分派完成事件前，过渡效果将清理所有临时对象，并删除所创建的侦听器。

调用 transitionComplete() 方法后， ViewNavigator 将完成视图切换过程并将过渡效果重置为未初始化的状态。

在移动设备应用程序中选择日期和时间

DateSpinner 控件允许用户在移动设备应用程序中选择日期和时间。它使用包含一系列相邻滚轮的常用移动设备界面，其中每个滚轮显示日期和 / 或时间的不同部分。

您可以使用三种基本类型的 DateSpinner 控件。下图显示了三种类型的 DateSpinner 控件:



A. 日期。 B. 时间。 C. 日期和时间

下表介绍了 DateSpinner 类型:

类型	常量 （等效字符串）	说明
日期	DateSelectorDisplayMode.DATE (“date”)	显示月、日和年。例如: `|| June || 11 || 2011 ||` 日期是默认类型。如果不设置 DateSpinner 控件的 displayMode 属性， Flex 将其设置为日期。 当前日期用 accentColor 样式属性定义的颜色高亮显示。 支持的最早日期是 1601 年 1 月 1 日。支持的最晚日期是 9999 年 12 月 31 日。

类型	常量 （等效字符串）	说明
时间	DateSelectorDisplayMode.TIME (“time”)	显示小时和分钟。对于使用 12 小时制的区域设置，还会显示 AM/PM 指示符。例如： `|| 2 || 57 || PM ||` 当前时间不高亮显示。 不能在 DateSpinner 控件中显示秒。 不能在 12 小时制与 24 小时制格式之间切换。 DateSpinner 使用当前区域设置的典型格式。
日期和时间	DateSelectorDisplayMode.DATE_AND_TIME (“dateAndTime”)	显示日、小时和分钟。对于使用 12 小时制的区域设置，还会显示 AM/PM 指示符。例如： `|| Mon Jun 13 || 2 || 57 || PM ||` 当前日期用 accentColor 样式属性定义的颜色高亮显示。当前时间不高亮显示。 不能在 DateSpinner 控件中显示秒。 月名称以缩短格式显示。不显示年。

DateSpinner 控件由多个 SpinnerList 控件组成。每个 SpinnerList 在 DateSpinner 控件中为特定位置显示一个有效值列表。例如，显示日期的 DateSpinner 控件包含三个 SpinnerList：一个用于日期，一个用于月，一个用于年。仅显示时间的 DateSpinner 包含两个或三个 SpinnerList：一个用于小时，一个用于分钟，（可选）一个用于 AM/PM （如果时间是以 12 小时为增量表示）。

更改 DateSpinner 控件的类型

您可以通过设置控件中 displayMode 属性的值选择 DateSpinner 的类型。您可以将 displayMode 属性设置为 DateSelectionDisplayMode 类或其等效的字符串定义的常量。

您可以通过以下示例切换不同的 DateSpinner 类型：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerTypes.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Types">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <s:ComboBox id="modeList" selectedIndex="0" change="ds1.displayMode=modeList.selectedItem.value">
        <s:ArrayList>
            <fx:Object value="date" label="Date"/>
            <fx:Object value="time" label="Time"/>
            <fx:Object value="dateAndTime" label="Date and Time"/>
        </s:ArrayList>
    </s:ComboBox>
    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>

    <s:Label text="{ds1.selectedDate}"/>

</s:View>
```

用户与 DateSpinner 控件交互时，微调框与列表中最近的项对齐。静止时，微调框绝不会在所选内容之间。

将 DateSpinner 控件选择绑定到其它控件

您可以将 DateSpinner 控件的 selectedDate 属性绑定到移动设备应用程序中的其它控件。selectedDate 属性是指向 Date 对象的指针，因此可以通过该方式访问 Date 对象的方法。

以下示例将日、月和年绑定到 Label 控件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerBinding.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Binding" creationComplete="initAC()">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
            import mx.collections.ArrayCollection;

            [Bindable]
            private var dayArrayC:ArrayCollection = new ArrayCollection();

            [Bindable]
            private var selectedDateProperty:Date;

            private function initAC():void {
                dayArrayC.addItem("Sunday");
                dayArrayC.addItem("Monday");
                dayArrayC.addItem("Tuesday");
                dayArrayC.addItem("Wednesday");
                dayArrayC.addItem("Thursday");
                dayArrayC.addItem("Friday");
                dayArrayC.addItem("Saturday");
            }

        ]]>
    </fx:Script>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>
    <fx:Binding source="ds1.selectedDate" destination="selectedDateProperty"/>

    <s:Label id="label1" text="Day: {dayArrayC.getItemAt(ds1.selectedDate.day)}"/>
    <s:Label id="label2" text="Day of month: {selectedDateProperty.getDate()}"/>
    <s:Label id="label3" text="Month: {ds1.selectedDate.getMonth() + 1}"/>
    <s:Label id="label4" text="Year: {selectedDateProperty.getFullYear()}"/>

</s:View>
```

在 DateSpinner 控件中以编程方式选择日期

通过将新的 Date 对象指定给 selectedDate 属性的值，您可以在 DateSpinner 控件中以编程方式更改日期。

以下示例提示您输入日、月和年。单击按钮时， DateSpinner 更改为新日期：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerProgrammaticSelection.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Programmatic Selection"
        creationComplete="init()">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.events.FlexEvent;
            import spark.components.calendarClasses.DateSelectorDisplayMode;

            private function init():void {
                // change event is dispatched when DateSpinner changes from user interaction
                ds1.addEventListener("change", spinnerEventHandler);
                // valueCommit event is dispatched when DateSpinner programmatically changes
                ds1.addEventListener("valueCommit", spinnerEventHandler);
            }

            private function b1_clickHandler(e:Event):void {
                ds1.selectedDate = new Date(ti3.text,ti1.text,ti2.text);
            }

            protected function spinnerEventHandler(event:Event):void {
                eventLabel.text = event.type;
            }
        ]]>
    </fx:Script>

    <s:TextInput id="ti1" prompt="Enter a Month"/>
    <s:TextInput id="ti2" prompt="Enter a Day"/>
    <s:TextInput id="ti3" prompt="Enter a Year"/>
    <s:Button id="b1" label="Go!" click="b1_clickHandler(event)"/>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.DATE}"/>

    <s:Label id="eventLabel"/>

</s:View>
```

以编程方式更改日期时，DateSpinner 控件同时分派 change 和 valueCommit 事件。通过用户交互更改日期时，DateSpinner 控件分派 change 事件。

以编程方式更改所选日期时，所选值对齐到视图中，无需通过中间值以动画效果呈现。

限制 DateSpinner 控件中的日期范围

可以通过 minDate 和 maxDate 属性限制用户可以在 DateSpinner 控件中选择的日期。这些属性接受 Date 对象。任何早于 minDate 属性的日期和晚于 maxDate 属性的日期在 DateSpinner 控件中都不可访问。此外，“date” 模式下不显示无效的年份，“dateAndTime” 模式下不显示无效的日期。

以下示例将创建两个具有不同的可用日期范围的 DateSpinner 控件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/MinMaxDates.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Min/Max Dates">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Min date today, max date October 31, 2012. -->
    <s:Label text="{dateSpinner1.selectedDate}"/>
    <s:DateSpinner id="dateSpinner1"
        displayMode="{DateSelectorDisplayMode.DATE}"
        minDate="{new Date()}"
        maxDate="{new Date(2012,9,31)}"/>
    <!-- Min date 3 days ago, max date 7 days from now. -->
    <s:Label text="{dateSpinner2.selectedDate}"/>
    <s:DateSpinner id="dateSpinner2"
        displayMode="{DateSelectorDisplayMode.DATE}"
        minDate="{new Date(new Date().getTime() - 1000*60*60*24*3)}"
        maxDate="{new Date(new Date().getTime() + 1000*60*60*24*7)}"/>
</s:View>
```

只能设置一个最小日期和一个最大日期。不能设置日期数组或多个选择范围。

此外，还可以使用 minDate 和 maxDate 属性将 DateSpinner 限制在 “time” 模式下。以下示例限制在 8 AM 和 2 PM 之间选择时间：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/MinMaxTime.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Min/Max Time">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Limit time selection to between 8am and 2pm -->
    <s:DateSpinner id="dateSpinner1"
                   displayMode="{DateSelectorDisplayMode.TIME}"
                   minDate="{new Date(0,0,0,8,0)}"
                   maxDate="{new Date(0,0,0,14,0)}"/>

</s:View>
```

响应 DateSpinner 控件的事件

DateSpinner 控件在用户更改日期时分派 change 事件。如以下示例所示，该 change 事件的 target 属性包含到 DateSpinner 的引用，您可以使用该引用访问所选日期：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerChangeEvent.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Change Event">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;

            private var dayArray:Array = new Array(
                "Sunday","Monday","Tuesday",
                "Wednesday","Thursday","Friday","Saturday");

            private function ds1_changeHandler(e:Event):void {
                // Optionally cast the DateSpinner's selectedDate as a Date
                var d:Date = new Date(e.currentTarget.selectedDate);
                ta1.text = "You selected:";
                ta1.text += "\n Day of Week: " + dayArray[d.day];
                ta1.text += "\n Year: " + d.fullYear;
                // Month is 0-based in ActionScript, so add 1:
                ta1.text += "\n Month: " + int(d.month + 1);
                ta1.text += "\n Day: " + d.date;
            }
        ]]>
    </fx:Script>

    <s:DateSpinner id="ds1"
        displayMode="{DateSelectorDisplayMode.DATE}"
        change="ds1_changeHandler(event)"/>

    <s:TextArea id="ta1" height="200" width="350"/>
</s:View>
```

仅在所有的微调框都已阻止在用户交互中执行微调时才会分派 change 事件 （并更新 selectedDate 属性的值）。

要捕获以编程方式完成的日期更改，请侦听 value_commit 事件。

更改 DateSpinner 控件的分钟间隔

您可以通过使用 minuteStepSize 属性更改 DateSpinner 控件显示的分钟间隔。该属性仅适用于将 displayMode 设置为 “time” 或 “dateAndTime” 的 DateSpinner 控件。例如，如果将 minuteStepSize 属性设置为 10，DateSpinner 控件会在分钟微调框中显示值 0、 10、 20、 30、 40 和 50。

通过以下示例，您可以设置 minuteStepSize 属性的值。分钟微调框会相应进行更新。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerMinuteInterval.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Minute Interval">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>
    <s:Label text="Select an interval:"/>

    <s:ComboBox id="intervalList" selectedIndex="0"
        change="ds1.minuteStepSize=intervalList.selectedItem.value">
        <s:ArrayList>
            <fx:Object value="1" label="1"/>
            <fx:Object value="2" label="2"/>
            <fx:Object value="3" label="3"/>
            <fx:Object value="4" label="4"/>
            <fx:Object value="5" label="5"/>
            <fx:Object value="6" label="6"/>
            <fx:Object value="10" label="10"/>
            <fx:Object value="12" label="12"/>
            <fx:Object value="15" label="15"/>
            <fx:Object value="20" label="20"/>
            <fx:Object value="30" label="30"/>
        </s:ArrayList>
    </s:ComboBox>

    <s:DateSpinner id="ds1" displayMode="{DateSelectorDisplayMode.TIME}"/>
</s:View>
```

minuteStepSize 属性的有效值必须能被 60 整除。如果您指定的值不能被 60 整除 （例如 25），则 minuteStepSize 属性的值默认为 1。

如果指定分钟间隔并且当前的时间不属于分钟微调框中的值，则 DateSpinner 控件会将当前所选内容向下舍入到最接近的间隔。例如，如果时间是 10:29，minuteStepSize 是 15，则 DateSpinner 舍入为 10:15，在此假定 10:15 的值不违反 minDate 设置。

自定义 DateSpinner 控件的外观

DateSpinner 控件支持大多数的文本样式，例如 fontSize、color 和 letterSpacing。此外，它还添加名为 accentColor 的新样式属性。该样式更改微调框列表中当前日期或时间的颜色。以下示例中将该颜色设置为红色：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/views/DateSpinnerStyles.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="DateSpinner Styles">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.components.calendarClasses.DateSelectorDisplayMode;
        ]]>
    </fx:Script>

    <!-- Acceptable style formats are color_name (e.g., 'red') or
        hex colors (e.g., '0xFF0000') -->
    <s:DateSpinner id="dateSpinner1" accentColor="0xFF0000"
        displayMode="{DateSelectorDisplayMode.DATE}"/>
</s:View>
```

DateSpinner 控件不支持 textAlign 属性。文本对齐由该控件设置。

要自定义 DateSpinner 控件外观的其它方面，可以为控件创建自定义外观或通过 CSS 修改某些子组件。

DateSpinnerSkin 类控制 DateSpinner 控件的大小调整。 DateSpinner 控件内的每个微调框就是一个具有各自 SpinnerListSkin 的 SpinnerList 对象。单个 DateSpinner 控件中的所有微调框都是单个 SpinnerListContainer 的子代，其中后者具有自己的外观 SpinnerListContainerSkin。

可以显式设置 DateSpinner 控件的 height 属性。如果设置 width 属性，该控件会将自身在大小调整为所请求宽度的区域中居中。

要在 DateSpinner 控件内修改微调框的设置，还可以使用 SpinnerList、 SpinnerListContainer 和 SpinnerListItemRenderer CSS 类型选择器。例如， SpinnerList 类型选择器控制微调框中的内边距属性。

以下示例将更改微调框中的内边距：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/DateSpinnerExamples2.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.CustomSpinnerListSkinExample">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        s|SpinnerListItemRenderer {
            paddingTop: 7;
            paddingBottom: 7;
            paddingLeft: 5;
            paddingRight: 5;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

在移动设备应用程序中，类型选择器必须位于顶级应用程序文件中，而不在应用程序的子视图中。如果您尝试在视图内的样式块中设置 SpinnerListItemRenderer 类型选择器， Flex 将引发编译器警告。

您可以扩展 SpinnerListContainerSkin 类以进一步自定义 DateSpinner 控件中微调框的外观。以下示例中将自定义外观应用于 SpinnerListContainer：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/DateSpinnerExamples3.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                            xmlns:s="library://ns.adobe.com/flex/spark"
                            firstView="views.CustomSpinnerListSkinExample">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        /* Change SpinnerListContainer for all DateSpinner controls */
        s|SpinnerListContainer {
            skinClass: ClassReference("customSkins.CustomSpinnerListContainerSkin");
        }

        /* Change padding for all DateSpinner controls */
        s|SpinnerListItemRenderer {
            paddingTop: 7;
            paddingBottom: 7;
            paddingLeft: 5;
            paddingRight: 5;
            fontSize: 12;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

以下 CustomSpinnerListContainerSkin 类将降低 " 选择指示器 " 的大小，因此它能更确切地反映微调框行中的字体和内边距的新大小。

```
// datespinner/customSkins/CustomSpinnerListContainerSkin.as
package customSkins {
    import mx.core.DPIClassification;

    import spark.skins.mobile.SpinnerListContainerSkin;
    import spark.skins.mobile.supportClasses.MobileSkin;
    import spark.skins.mobile160.assets.SpinnerListContainerBackground;
    import spark.skins.mobile160.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile160.assets.SpinnerListContainerShadow;
    import spark.skins.mobile240.assets.SpinnerListContainerBackground;
    import spark.skins.mobile240.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile240.assets.SpinnerListContainerShadow;
    import spark.skins.mobile320.assets.SpinnerListContainerBackground;
    import spark.skins.mobile320.assets.SpinnerListContainerSelectionIndicator;
    import spark.skins.mobile320.assets.SpinnerListContainerShadow;

    public class CustomSpinnerListContainerSkin extends SpinnerListContainerSkin
    {
        public function CustomSpinnerListContainerSkin() {
            super();

            switch (applicationDPI)
            {
                case DPIClassification.DPI_320:
                {
                    borderClass = spark.skins.mobile320.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile320.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile320.assets.SpinnerListContainerShadow;

                    cornerRadius = 10;
                    borderThickness = 2;
                    selectionIndicatorHeight = 80; // was 120
                    break;
                }
```

```
                case DPIClassification.DPI_240:
                {
                    borderClass = spark.skins.mobile240.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile240.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile240.assets.SpinnerListContainerShadow;

                    cornerRadius = 8;
                    borderThickness = 1;
                    selectionIndicatorHeight = 60; // was 90
                    break;
                }
                default: // default DPI_160
                {
                    borderClass = spark.skins.mobile160.assets.SpinnerListContainerBackground;
                    selectionIndicatorClass =
spark.skins.mobile160.assets.SpinnerListContainerSelectionIndicator;
                    shadowClass = spark.skins.mobile160.assets.SpinnerListContainerShadow;

                    cornerRadius = 5;
                    borderThickness = 1;
                    selectionIndicatorHeight = 40; // was 60

                    break;
                }
            }
        }
    }
}
```

有关制作移动设备组件外观的更多信息，请参阅第 141 页的 “ 移动设备外观设计的基础知识 ”。

对 DateSpinner 控件使用本地化日期和时间

DateSpinner 控件支持运行应用程序的设备支持的所有区域设置。如果您将区域设置设定为 ja-JP，则 DateSpinner 会发生相应更改，以日本区域设置标准显示日期。

可以直接在 DateSpinner 控件上设置 locale 属性，也可以在容器上设置，例如 Application。DateSpinner 控件将继承该属性的值。默认区域设置是运行应用程序的设备中的区域设置，除非您通过 locale 属性将其重写。

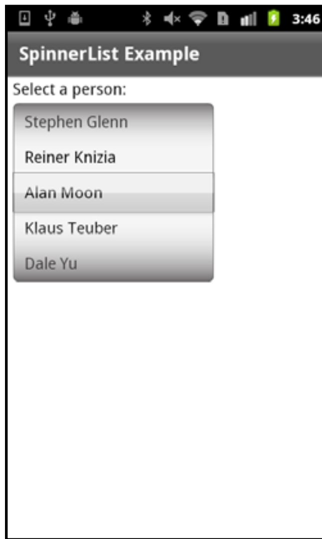
以下示例将默认区域设置设定为 “ja-JP”。您可以选择区域设置以更改 DateSpinner 的格式：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- datespinner/LocalizedDateSpinner.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Localized DateSpinner" locale="ja_JP">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            private function localeChangeHandler():void {
                ds1.setStyle('locale',localeSelector.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:ComboBox id="localeSelector" change="localeChangeHandler()">
        <s:ArrayList>
            <fx:String>en-US</fx:String>
            <fx:String>en-UK</fx:String>
            <fx:String>es-AR</fx:String>
            <fx:String>he-IL</fx:String>
            <fx:String>ko-KR</fx:String>
            <fx:String>ja-JP</fx:String>
            <fx:String>vi-VN</fx:String>
            <fx:String>zh-CN</fx:String>
            <fx:String>zh-TW</fx:String>
        </s:ArrayList>
    </s:ComboBox>
    <s:DateSpinner id="ds1" displayMode="dateAndTime"/>

</s:View>
```

在移动设备应用程序中使用微调框列表

SpinnerList 组件是专用 List，在移动设备应用程序中通常用于数据选择。默认情况下，用户滚动浏览列表项时，这些项会在用户到达列表末尾时封装。 SpinnerList 控件通常用作移动设备应用程序中的 Numeric Stepper 组件。

下图显示了 SpinnerList 控件在移动设备应用程序中的典型外观：



SpinnerList 控件

SpinnerList 控件的行为方式就像旋转的圆柱形滚筒一样。用户可以通过使用向上或向下抛开来旋转列表，将其向上或向下拖动，以及单击列表中的项。

通常将 SpinnerList 控件封装在 SpinnerListContainer 控件中。该类提供 SpinnerList 的大多数主色并定义布局。主色包含选择指示符的边框、阴影和外观。

SpinnerList 数据以列表形式存储。这些数据使用 SpinnerListItemRenderer 呈示在微调框中。您可以重写项呈示器以自定义列表项的外观或内容。

DateSpinner 控件是一组带有自定义项呈示器的 SpinnerList 控件的示例。

如果您不禁用整个 SpinnerList 控件，则当前不能禁用该控件中的项。该限制不适用于 DateSpinner 控件，该控件提供用于设置禁用日期范围的附加逻辑。

定义微调框列表的数据

要定义 SpinnerList 控件的数据，可以执行以下操作之一：

- 定义 SpinnerList 控件中 dataProvider 属性的内联数据。
- 将数据定义为 <s:SpinnerList> 标签的子标签。
- 定义 ActionScript 或 MXML 中的数据并将其绑定到 SpinnerList 控件。该数据可以来自外部服务、如 XML 文件之类的嵌入资源或任何其它数据源。
- 通过 Flash Builder 服务向导将 SpinnerList 控件绑定到数据服务操作。有关使用 Flash Builder 构建以数据为中心的应用程序的更多信息，请参阅连接到数据服务。

SpinnerList 控件可以接受作为数据提供程序实现 IList 接口的任何类。这些类包含 ArrayCollection、 ArrayList、NumericDataProvider 和 XMLListCollection 类。

如果您在实例化 SpinnerList 控件时没有为该控件定义数据提供程序，则 SpinnerList 中将显示一个空行。添加数据提供程序之后， SpinnerList 会调整大小以在列表中显示设置为默认值的五个项。

以下示例为 <s:SpinnerList> 标签的子标签中的 SpinnerList 控件定义数据：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListComplexDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Complex Data Provider">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <s:Label text="Select a person:"/>

    <s:SpinnerListContainer>
        <s:SpinnerList id="peopleList" width="200" labelField="name">
            <s:ArrayList>
                <fx:Object name="Friedeman Friese" companyID="14266"/>
                <fx:Object name="Stephen Glenn" companyID="14266"/>
                <fx:Object name="Reiner Knizia" companyID="11233"/>
                <fx:Object name="Alan Moon" companyID="11543"/>
                <fx:Object name="Klaus Teuber" companyID="13455"/>
                <fx:Object name="Dale Yu" companyID="14266"/>
            </s:ArrayList>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:Label text="Selected ID: {peopleList.selectedItem.companyID}"/>

</s:View>
```

以下示例定义 <s:SpinnerList> 标签中的 SpinnerList 数据：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListInlineDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Inline Data Provider">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <!-- Create data provider inline. -->
        <s:SpinnerList id="smallList" dataProvider="{new ArrayList([1,5,10,15,30])}"
                       wrapElements="false" typicalItem="44"/>
    </s:SpinnerListContainer>

    <s:Label text="Selected Item: {smallList.selectedItem}"/>

</s:View>
```

以下示例定义 ActionScript 中的 SpinnerList 数据:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListBasicDataProvider.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Basic Data Provider"
        creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            public var daysOfWeek:ArrayList;

            private function initApp():void {
                daysOfWeek = new ArrayList(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]);
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="daysList" width="100" dataProvider="{daysOfWeek}"/>
    </s:SpinnerListContainer>

    <s:Label text="Selected Day: {daysList.selectedItem}"/>

</s:View>
```

如果将复杂对象用作 ActionScript 中的数据，指定 labelField 属性以便 SpinnerList 显示正确的标签，如下例所示:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListComplexASDP.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Complex Data Provider in AS" creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            private var myAC:ArrayList;
```

```
            private function initApp():void {
                myAC = new ArrayList([
                    {name:"Alan Moon",id:42},
                    {name:"Friedeman Friese",id:44},
                    {name:"Dale Yu",id:45},
                    {name:"Stephen Glenn",id:47},
                    {name:"Reiner Knizia",id:48},
                    {name:"Klaus Teuber",id:49}
                ]);
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="peopleList" dataProvider="{myAC}"
                       width="200"
                       labelField="name"/>
    </s:SpinnerListContainer>
    <s:Label text="Selected ID: {peopleList.selectedItem.id}"/>
</s:View>
```

您还可以使用便捷类 NumericDataProvider 为 SpinnerList 控件提供数字数据。可以通过该类轻松定义一组包含最小值、最大值和步长的数字数据。

以下示例将 NumericDataProvider 类用作 SpinnerList 控件的数据源：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/MinMaxSpinnerList.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Min/Max SpinnerLists"
        backgroundColor="0x000000">

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer top="10" left="10">
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="23" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="59" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="59" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
        <s:SpinnerList typicalItem="100"
                       dataProvider="{new ArrayList(['AM','PM'])}"
                       wrapElements="false"/>
    </s:SpinnerListContainer>
</s:View>
```

stepSize 属性的值可以是负数。在这种情况下，最大值是显示的第一个值。微调框首先显示最大值，然后逐渐降低到最小值。

选择微调框列表中的项

SpinnerList 控件仅支持一次选择一项。默认情况下，所选项始终位于组件的中心并显示在选择指示符下方。不旋转时，SpinnerList 必须始终选中一项。不能选择禁用项或没有项的行。

要获得 SpinnerList 控件中的当前选定项，您可以访问控件的 selectedIndex 或 selectedItem 属性。

要设置 SpinnerList 控件中的当前选定项，您可以设置 selectedIndex 或 selectedItem 属性的值。通常在 <s:SpinnerList> 标签中设置这些属性，这样在创建 SpinnerList 时即选定该项。

如果未在 SpinnerList 中显式设置 selectedIndex 或 selectedItem 属性的值，默认的选定项就是列表中的第一项。

您可以使用 selectedIndex 或 selectedItem 属性以编程方式更改微调框中的选定项。设置其中的一个属性时，控件将与该项对齐，而不以动画效果呈现 （或 " 旋转 "）该项的微调框。

以下示例将 SpinnerList 控件用作倒计时器。 Timer 对象通过每秒更改一次 selectedIndex 属性的值来更改微调框中的选定项:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListCountdownTimer.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Countdown Timer"
        creationComplete="initApp()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            private var myTimer:Timer;

            private function initApp():void {
                myTimer = new Timer(1000, 0); // 1 second
                myTimer.addEventListener(TimerEvent.TIMER, changeSpinner);
                myTimer.start();
            }
            private function changeSpinner(e:Event):void {
                secList.selectedIndex = secList.selectedIndex - 1;
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer left="50" top="50">
        <s:SpinnerList id="secList" width="100" selectedIndex="60">
            <s:dataProvider>
                <s:NumericDataProvider minimum="0" maximum="60" stepSize="1"/>
            </s:dataProvider>
        </s:SpinnerList>
    </s:SpinnerListContainer>
</s:View>
```

用户与微调框列表的交互和事件

SpinnerList 控件中的选定项发生更改时，该控件会分派 change 和 valueCommit 事件。这通常是对用户交互 （例如滑动）所做的反应。如果用户通过触摸项将其选定，该控件会分派 click 事件以及 change 和 valueCommit 事件。

选定项以编程方式更改时， SpinnerList 控件仅分派 valueCommit 事件。

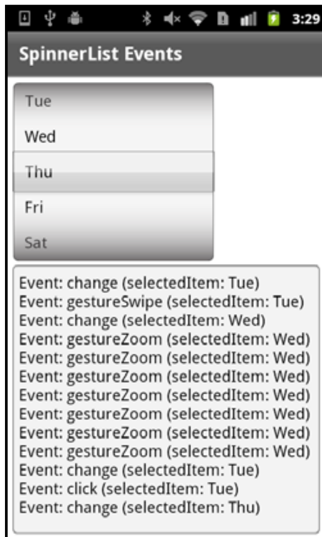
SpinnerList 控件旋转时，不针对它传递的每个项分派事件。该控件仅当它停留在新项上时分派事件，例如 change 或 valueCommit。

SpinnerList 控件首次通过数据提供程序实例化时，它同时分派 change 和 valueCommit 事件。

以下示例显示了使用 SpinnerList 控件时分派的常见事件:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListEvents.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="SpinnerList Events"
        creationComplete="initApp()">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"
                          paddingRight="10" paddingBottom="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;

            [Bindable]
            public var daysOfWeek:ArrayList;

            private function initApp():void {
                daysOfWeek = new ArrayList(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]);
            }

            private function eventHandler(e:Event):void {
               ta1.text += "Event: " + e.type + " (selectedItem: " + e.currentTarget.selectedItem + ")\n";
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="daysList" width="300"
            dataProvider="{daysOfWeek}"
            change="eventHandler(event)"
            gestureSwipe="eventHandler(event)"
            click="eventHandler(event)"
            gestureZoom="eventHandler(event)"
        />
    </s:SpinnerListContainer>
    <s:TextArea id="ta1" width="100%" height="100%"/>
</s:View>
```

下图显示与 SpinnerList 控件交互后的输出：



SpinnerList 控件事件

设置微调框列表中的封装

默认情况下，如果 SpinnerList 控件的数据提供程序中的项数少于微调框中显示的项数，则微调框不封装，而是停止在列表中的最后一项。否则，微调框将在用户经过最后一项时封装到列表的开头。

列表中显示的默认项数是 5。如果您要更改项数，请创建自定义外观。有关更多信息，请参阅第 102 页的 “ 为微调框列表创建自定义外观 ”。

wrapElements 属性的值确定是否在到达列表中的最后一项后 SpinnerList 控件再次从第一项开始。如果 wrapElements 设置为 false，则无论列表中的项数和显示项数为多少，微调框都会在到达列表末尾时停止。

如果 wrapElements 属性设置为 true，则仅当列表中包含的项至少比可显示的项数多一个时，微调框才会再次从第一项开始。例如，如果 SpinnerList 显示 5 项，但是列表中仅有 4 项，则无论 wrapElements 属性的设置为何，列表都不封装。

可以通过将 wrapElements 属性设置为 true 或 false 重写 SpinnerList 的默认封装行为。

可以通过以下示例切换 wrapElements 属性的值：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListWrapElements.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Wrap Elements">
    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="smallList" typicalItem="45"
                       dataProvider="{new ArrayList([1,5,6,10,15,30])}"
                       wrapElements="{cb1.selected}"/>
    </s:SpinnerListContainer>

    <!-- By default, cause the elements to be wrapped by setting this to true -->
    <s:CheckBox id="cb1" label="Wrap Elements" selected="true"/>

</s:View>
```

通常，如果列表中的项多于列表一次显示的项，用户希望封装列表。如果列表中的项少于微调框可以显示的项，则用户通常希望不封装列表。

设置微调框列表的样式

SpinnerList 控件支持对 Spark 移动设备主题通用的所有文本样式。这些样式包含 fontSize、 fontWeight、 color、 textDecoration 和对齐属性。您可以在 MXML 或 CSS 中的控件上直接设置这些样式属性。如果这些属性是在父容器中设置的，则 SpinnerList 也会继承这些属性。

您还可以通过修改 SpinnerListItemRenderer 样式属性定义 SpinnerList 的内边距属性。

以下示例设置 SpinnerList 类型选择器的与文本相关的样式属性和 SpinnerListItemRenderer 类型选择器的内边距属性：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/SpinnerListExamples2.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                            xmlns:s="library://ns.adobe.com/flex/spark"
                            firstView="views.SpinnerListStyles">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|SpinnerList {
            textAlign: right;
            fontSize: 13;
            fontWeight: bold;
            color: red;
        }
        s|SpinnerListItemRenderer {
            paddingTop: 5;
            paddingBottom: 5;
            paddingRight: 5;
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

在移动设备应用程序中，如果使用类型选择器，则在顶级应用程序文件中定义 <fx:Style> 块。否则，编译器会引发警告并且不应用样式。

SpinnerList 控件不支持 accentColor、 backgroundAlpha、 backgroundColor 或 chromeColor 样式属性。

为微调框列表创建自定义外观

可以为 SpinnerList 控件或 SpinnerListContainer 控件创建自定义外观。为此，通常复制 SpinnerListSkin 或 SpinnerListContainerSkin 源作为自定义外观类的基础。

通常创建自定义 SpinnerList 外观以便修改 SpinnerList 控件或其容器的以下方面：

- 更改当前选定项 (selectionIndicator) 的框大小或形状。可以通过创建自定义 SpinnerListContainerSkin 类完成上述任务。
- 定义每行的高度 (rowHeight)。可以通过创建自定义 SpinnerListSkin 类完成上述任务。
- 定义显示行数 (requestedRowCount)。可以通过创建自定义 SpinnerListSkin 类完成上述任务。
- 定义容器外观 （例如角半径和边框宽度）。可以通过创建自定义 SpinnerListContainerSkin 类完成上述任务。

有关自定义 SpinnerListSkin 和 SpinnerListContainerSkin 的示例，请参阅第 89 页的 “ 自定义 DateSpinner 控件的外观 ”。

在微调框列表中使用图像

通过将 IconItemRenderer 定义为 SpinnerList 的项呈示器，您可以在 SpinnerList 控件中使用图像而不是文本标签。

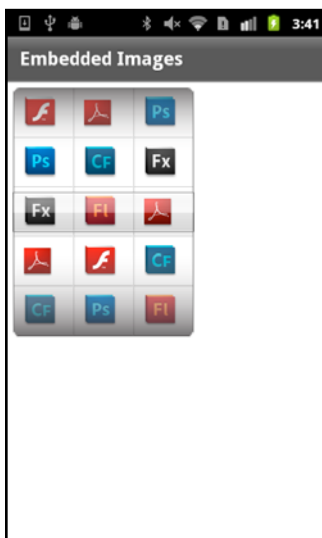
要在 IconItemRenderer 对象中使用图像，可以嵌入图像或在运行时加载图像。对于移动设备用户，嵌入图像可能更为适用，这样可以最大限度地减少数据网络使用量。

以下示例在 SpinnerList 控件中使用嵌入图像：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_spinnerlist/views/SpinnerListEmbeddedImage.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Embedded Images">

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10"/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            import mx.collections.ArrayList;
            [Embed(source="../../assets/product_icons/flex_50x50.gif")]
            [Bindable]
            public var icon0:Class;
            [Embed(source="../../assets/product_icons/acrobat_reader_50x50.gif")]
            [Bindable]
            public var icon1:Class;
            [Embed(source="../../assets/product_icons/coldfusion_50x50.gif")]
            [Bindable]
            public var icon2:Class;
            [Embed(source="../../assets/product_icons/flash_50x50.gif")]
            [Bindable]
            public var icon3:Class;
            [Embed(source="../../assets/product_icons/flash_player_50x50.gif")]
            [Bindable]
            public var icon4:Class;
            [Embed(source="../../assets/product_icons/photoshop_50x50.gif")]
            [Bindable]
            public var icon5:Class;

            // Return an ArrayList of icons for each spinner
            private function getIconList():ArrayList {
                var a:ArrayList = new ArrayList();
                a.addItem({icon:icon0});
                a.addItem({icon:icon1});
                a.addItem({icon:icon2});
                a.addItem({icon:icon3});
                a.addItem({icon:icon4});
                a.addItem({icon:icon5});
                return a;
            }
        ]]>
    </fx:Script>

    <s:SpinnerListContainer>
        <s:SpinnerList id="productList1" width="90" dataProvider="{getIconList()}" selectedIndex="0">
            <s:itemRenderer>
                <fx:Component>
```

```
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
        <s:SpinnerList id="productList2" width="90" dataProvider="{getIconList()}" selectedIndex="2">
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
        <s:SpinnerList id="productList3" width="90" dataProvider="{getIconList()}" selectedIndex="1">
            <s:itemRenderer>
                <fx:Component>
                    <s:IconItemRenderer labelField="" iconField="icon"/>
                </fx:Component>
            </s:itemRenderer>
        </s:SpinnerList>
    </s:SpinnerListContainer>
</s:View>
```

下图显示该应用程序在移动设备中的外观：



包含嵌入图像的 SpinnerList 控件

第 4 章：应用程序设计和工作流

在移动设备应用程序中启用持久化机制

移动设备设备应用程序的运行经常被其它操作 （例如短信、电话或其它移动设备应用程序）中断。通常，当重新启动被打断的应用程序时，用户希望应用程序恢复先前的状态。通过持久化机制，设备可以将应用程序恢复为先前的状态。

Flex 框架为移动设备应用程序提供两种持久化机制。内存持久化机制用于在用户进行应用程序导航时保存视图数据。会话持久化机制用于在用户退出应用程序后重新启动时恢复数据。由于移动操作系统随时都可能退出应用程序 （例如当内存不足时），因此会话持久化机制在移动设备应用程序中非常重要。

博客 Steve Mathews 编写了一个有关 Flex 移动设备应用程序中的简单数据持久性的手册条目。

博客 Holly Schinsky 发表了关于持久化和数据处理的文章 Flex Mobile Data Handling。

内存持久化

View 容器通过使用 View.data 属性来支持内存持久化机制。当切换所选部分或者将新视图推送到 ViewNavigator 堆栈而导致现有视图被破坏时，将自动保存现有视图的 data 属性。当控制权返回给该视图并重新实例化和激活该视图时，将恢复该视图的 data 属性。因此，通过内存持久化机制可以在运行时维护视图的状态信息。

会话持久化

会话持久化机制在应用程序执行各种操作时保留应用程序状态信息。 ViewNavigatorApplication 和 TabbedViewNavigatorApplication 容器通过定义 persistNavigatorState 属性来实现会话持久化机制。将 persistNavigatorState 设置为 true 以启用会话持久化机制。默认情况下， persistNavigatorState 设置为 false。

如果启用，会话持久化机制将使用名为 FxAppCache 的本地共享对象将应用程序的状态写入磁盘。应用程序还可以使用 spark.managers.PersistenceManager 的方法将其它信息写入本地共享对象。

ViewNavigator 会话持久化

在退出应用程序时， ViewNavigator 容器将其视图堆栈的状态保存到磁盘，以支持会话持久化机制。同时会保存当前视图的 data 属性。

当重新启动应用程序时，将重新初始化 ViewNavigator 的堆栈，且用户将看到与退出应用程序时相同的视图与内容。由于堆栈中包含每个视图的 data 属性的副本，因此在激活视图时，可以重新创建堆栈中的上一个视图。

TabbedViewNavigator 会话持久化

对于 TabbedViewNavigator 容器，在退出应用程序时，会话持久化机制将保存选项卡栏上当前选定的选项卡。选项卡对应于 ViewNavigator 和定义该选项卡的视图堆栈。同时保存的还有当前视图的 data 属性。因此，重新启动应用程序时，活动选项卡和相关联的 ViewNavigator 设置为退出应用程序时的状态。

注：对于使用 TabbedViewNavigatorApplication 容器定义的应用程序，仅保存当前 ViewNavigator 的堆栈。因此，重新启动应用程序时，仅恢复当前 ViewNavigator 的状态。

会话持久化机制数据表示

Flex 采用的持久化机制没有加密或保护措施。因此，其它程序或用户可以解读持久化数据的存储格式。不要使用此机制持久保存用户凭据等敏感信息。可以选择自行编写可提供更好保护的持久化管理器。有关更多信息，请参阅第 108 页的 “ 自定义持久化机制 ”。

使用会话持久化机制

在以下示例中，将应用程序的 persistNavigatorState 属性设置为 true，以启用会话持久化机制：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkSingleSectionPersist.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.EmployeeMainView"
    persistNavigatorState="true">

    <fx:Script>
        <![CDATA[
            protected function button1_clickHandler(event:MouseEvent):void {
                // Switch to the first view in the section.
                navigator.popToFirstView();
            }
        ]]>
    </fx:Script>

    <s:navigationContent>
        <s:Button icon="@Embed(source='assets/Home.png')"
                  click="button1_clickHandler(event)"/>
    </s:navigationContent>
</s:ViewNavigatorApplication>
```

此应用程序使用 EmployeeMainView.mxml 作为其第一个视图。 EmployeeMainView.mxml 定义 List 控件，您可以使用该控件选择用户名：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeMainView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employees">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import spark.events.IndexChangeEvent;
            protected function myList_changeHandler(event:IndexChangeEvent):void {
                navigator.pushView(views.EmployeeView,myList.selectedItem);
            }
        ]]>
    </fx:Script>

    <s:Label text="Select an employee name"/>
    <s:List id="myList"
        width="100%" height="100%"
        labelField="firstName"
        change="myList_changeHandler(event)">
        <s:ArrayCollection>
            <fx:Object firstName="Bill" lastName="Smith" companyID="11233"/>
            <fx:Object firstName="Dave" lastName="Jones" companyID="13455"/>
            <fx:Object firstName="Mary" lastName="Davis" companyID="11543"/>
            <fx:Object firstName="Debbie" lastName="Cooper" companyID="14266"/>
        </s:ArrayCollection>
    </s:List>
</s:View>
```

要查看会话持久化机制，请打开应用程序，然后在 List 控件中选择 “Dave”，以导航至 EmployeeView.mxml 视图：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\EmployeeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="Employee View">
    <s:layout>
        <s:VerticalLayout paddingTop="10"/>
    </s:layout>

    <s:VGroup>
        <s:Label text="{data.firstName}"/>
        <s:Label text="{data.lastName}"/>
        <s:Label text="{data.companyID}"/>
    </s:VGroup>
</s:View>
```

EmployeeView.mxml 视图中将显示有关 “Dave” 的数据。然后，退出应用程序。重新启动应用程序时，您将再次看到 EmployeeView.mxml 视图中显示与退出应用程序时相同的数据。

访问本地共享对象中的数据

本地共享对象中的信息以键 : 值对的格式保存。 PeristenceManager 的方法 （如 setPropery() 和 getProperty()）根据键在本地共享对象中访问相关联的值。

您可以使用 setProperty() 方法将自己的键 : 值对写入本地共享对象。 setProperty() 方法具有如下签名：

```
setProperty(key:String, value:Object):void
```

使用 getProperty() 方法可以访问特定键的值。 getProperty() 方法具有如下签名：

```
getProperty(key:String):Object
```

如果 persistNavigatorState 属性为 true，则在退出应用程序时，持久化管理器自动将两个键 : 值对保存到本地共享对象：

- applicationVersion

  应用程序版本，如 application.xml 文件中所述。

- navigatorState

  导航器的视图状态，对应于当前 ViewNavigator 的堆栈。

执行手动持久化

当 persistNavigatorState 属性为 true 时， Flex 将自动执行会话持久化机制。当 persistNavigatorState 属性为 false 时，也仍可以持久保存应用程序数据。在此情况下，请使用 PeristenceManager 的方法来实现自己的持久化机制。

使用 setProperty() 和 getProperty() 方法，在本地共享对象中写入和读取信息。调用 load() 方法以初始化 PeristenceManager。调用 save() 方法以将任何数据写入磁盘。

注：如果 persistNavigatorState 属性为 false，Flex 将不会在退出应用程序时自动保存当前 ViewNavigator 的视图堆栈，也不会在启动应用程序时恢复视图堆栈。

处理持久化事件

可以使用移动设备应用程序容器的以下事件来开发自定义的持久化机制：

- navigatorStateSaving
- navigatorStateLoading

您可以通过在 navigatorStateSaving 事件的处理函数中调用 preventDefault() 方法，来取消将应用程序状态保存至磁盘的操作。在 navigatorStateLoading 事件的处理函数中调用 preventDefault() 方法可以取消在重新启动应用程序时加载状态。

自定义持久化机制

如果启用会话持久化机制，打开应用程序时，将显示退出该应用程序时所显示的视图。必须在视图的 data 属性或共享对象等其它位置中存储足够多的信息，才能完全恢复应用程序状态。

例如，恢复的视图可能需要根据视图的 data 属性来进行计算。然后，应用程序必须在重新启动应用程序时进行识别，并执行必要的计算。一种方法是在退出和重新启动应用程序时重写 View 的 serializeData() 和 deserializePersistedData() 方法，以执行自己的操作。

会话持久化机制支持的内置数据类型

持久化机制自动支持所有内置数据类型，包括：Number、String、Vector、Object、uint、int 和 Boolean。持久化机制会自动保存这些数据类型。

会话持久化机制支持的自定义类

许多应用程序使用自定义类来定义数据。如果自定义类中包含由内置数据类型定义的属性，则持久化机制可以自动保存和加载这些类。但是，必须首先调用 flash.net.registerClassAlias() 方法，向持久化机制注册这些类。通常，您可以在应用程序的 preinitialize 事件中调用此方法，然后初始化持久化存储或将任意数据保存到持久化存储中。

如果定义复杂的类 （即使用非内置数据类型的数据类型的类），则必须将数据转换为受支持的类型，如 String。此外，如果类定义了任何私有变量，则不会自动持久保存这些变量。要在持久化机制中支持复杂的类，该类必须实现 flash.utils.IExternalizable 接口。此接口要求类实现 writeExternal() 和 readExternal() 方法以保存和恢复类的实例。

在一个移动设备应用程序中支持多个屏幕大小和 DPI 值

支持多个屏幕大小和 DPI 值的指导原则

要部署独立于平台的应用程序，应了解不同的输出设备。设备可以具有不同的屏幕大小或分辨率以及不同的 DPI 值或密度。

Flex 工程师 Jason SJ 在他的博客中介绍了两种创建与分辨率无关的移动设备应用程序的方法。

术语

分辨率是像素高度乘以像素宽度得到的数值：即设备支持的像素总数。

DPI 是每平方英寸的点数：即设备屏幕上的像素密度。术语 DPI 和 PPI （每英寸像素数）可以互换使用。

Flex 对 **DPI** 的支持

以下 Flex 功能简化了生成与分辨率和 DPI 无关的应用程序的过程。

外观 移动设备组件与 DPI 有关的外观。默认移动设备外观无需额外编写代码，即可根据大多数设备的分辨率进行正常缩放。

applicationDPI 该属性用于定义自定义外观的设计尺寸。假设将该属性设置为某 DPI 值，当用户在具有不同 DPI 值的设备上运行应用程序时， Flex 会根据所使用设备的 DPI 缩放应用程序中的所有内容。

无论是否具有 DPI 缩放功能，默认移动设备外观都与 DPI 无关。因此，如果不使用具有静态大小或自定义外观的组件，则通常无需设置 applicationDPI 属性。

动态布局

动态布局可以帮助您适应各种不同的分辨率。例如，如果将控件的宽度设置为 100%，将始终填满屏幕的宽度，无论屏幕分辨率是 480x854 还是 480x800。

设置 **applicationDPI** 属性

设置与密度无关的应用程序时，可以在根应用程序标签上设置目标 DPI。（对于移动设备应用程序，根标签为 <s:ViewNavigatorApplication>、 <s:TabbedViewNavigatorApplication> 或 <s:Applica ion>。）

可以将 applicationDPI 属性的值设置为 160、 240 或 320，取决于目标设备的近似分辨率。例如：

```
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.DensityView1"
    applicationDPI="320">
```

设置 applicationDPI 属性时，可以对照运行时目标设备的实际分辨率 (runtimeDPI) 来有效定义应用程序的缩放比例。例如，如果将 applicationDPI 属性设置为 160，而目标设备的 runtimeDPI 为 160，则缩放比例因子为 1 （不缩放）。如果将 applicationDPI 属性设置为 240，则缩放比例因子为 1.5 （Flex 将所有内容放大到 150%）。设置为 320 时，缩放比例因子为 2，则 Flex 将所有内容放大到 200%。

在某些情况下，非整数缩放会因为插值处理而造成失真，例如线条模糊。

禁用 **DPI** 缩放

要对应用程序禁用 DPI 缩放，请不要设置 applicationDPI 属性的值。

了解 **applicationDPI** 和 **runtimeDPI**

下表介绍 Application 类的两个属性，在不同分辨率下处理应用程序时，这两个属性不可或缺：

属性	说明
applicationDPI	应用程序的目标密度或 DPI。 当您指定该属性的值时， Flex 会对根应用程序应用缩放比例因子。这将使针对某一 DPI 值设计的应用程序进行缩放，以便在具有不同 DPI 值的设备上正常显示。 将此属性的值与 runtimeDPI 属性的值进行比较，可以计算出缩放比例因子。此缩放比例因子应用于整个应用程序，包括预加载器、弹出窗口和所有显示的组件。 如果不指定此属性的值，此属性将返回与 runtimeDPI 属性相同的值。 此属性不能在 ActionScript 中设置，而只能在 MXML 中设置。在运行时，不能更改此属性的值。
runtimeDPI	当前正在运行应用程序的设备的密度或 DPI 值。 返回 Capabilities.screenDPI 属性的值，舍入到 DPIClassification 类所定义的一个常量。 此属性为只读属性。

创建与分辨率和 **DPI** 无关的应用程序

与分辨率和 DPI 无关的应用程序具有以下特征：

图像 矢量图可以平滑缩放至与目标设备实际分辨率相符的大小。而位图则无法像矢量图那样缩放。在这些情况下，您可以使用 MultiDPIBitmapSource 类，根据设备分辨率，以不同的分辨率加载位图。

文本 文本的字体大小 （而非文本本身）缩放至与分辨率相符的大小。

布局 使用动态布局，以确保应用程序在缩放后能够正常显示。通常应尽量避免使用基于约束的布局，在这种布局中使用绝对值来指定像素边界。如果使用约束，应通过 applicationDPI 属性的值来进行缩放。

缩放 不要对应用程序对象使用 scaleX 和 scal Y 属性。设置 applicationDPI 属性时， Flex 会进行缩放。

样式 您可以使用样式表自定义目标设备的操作系统和应用程序 DPI 设置的样式属性。

外观 移动设备主题中的 Flex 外观使用应用程序 DPI 值来确定运行时所使用的资源。由 FXG 文件定义的所有可视外观资源都适用于目标设备。

应用程序大小 不要显式设置应用程序的高度和宽度。同样，在计算自定义组件或弹出窗口的大小时，不要使用 stageWidth 和 stageHeight 属性。而应使用 SystemManager.screen 属性。

确定运行时 DPI

应用程序在启动时，会从 Capabilities.screenDPI Flash Player 属性中获取 runtimeDPI 属性的值。此属性映射到 DPIClassification 类所定义的一个常量。例如，以 232 DPI 运行的 Droid 映射到 240 运行时 DPI 值。设备 DPI 值并不总是与 DPIClassification 常量 （160、 240 或 320）完全一致。而是根据目标值的范围，映射到这些分类。

映射关系如下所示：

DPIClassification 常量	160 DPI	240 DPI	320 DPI
实际设备 DPI	<200	>=200 并 <280	>=280

您可以自定义这些映射关系来重写默认行为，或者调整错误报告了自己的 DPI 值的设备。有关更多信息，请参阅第 117 页的 "重写默认 DPI"。

选择自动缩放或非自动缩放

选择使用自动缩放 （通过设置 applicationDPI 属性值）是获得方便和确保像素精确的视觉保真度之间的一种折衷方法。如果将 applicationDPI 属性设置为自动缩放应用程序， Flex 会使用针对 applicationDPI 的外观。 Flex 将根据设备的实际密度缩小或放大外观。应用程序和布局位置中的其它资源也将进行缩放。

如果要使用自动缩放，并要创建针对单个 DPI 值的外观或资源，通常需要做到以下几点：

- 创建针对所指定的 applicationDPI 的单组外观和视图 / 组件布局。
- 创建在外观或应用程序中的其它位置所使用的任何位图资源的多个版本，然后使用 MultiDPIBitmapSource 类进行指定。自动缩放时，外观中的矢量资源和文本不需要考虑密度。
- 不要在样式表中使用 @media 规则，因为应用程序仅考虑单个目标 DPI 值。
- 在不同密度的设备上测试应用程序，以确保缩放后的应用程序的外观在每个设备上都可接受。特别是要检查按非整数因子进行缩放的设备。例如，如果 applicationDPI 为 160，请在 DPI-240 设备上测试应用程序。

如果选择不使用自动缩放 （不设置 applicationDPI 属性），将获得 applicationDPI 值。可使用该属性确定设备的实际 DPI 等级，并通过在运行时执行以下操作调整应用程序：

- 针对每个运行时 DPI 等级创建多组外观和布局，或创建可根据不同密度动态调整的单组外观和布局。（内置的 Flex 外观采用第二种方法，即每个外观类检查 applicationDPI 属性并对自身进行相应的设置。）
- 在样式表中使用 @media 规则，以根据设备的 DPI 等级过滤 CSS 规则。通常，应针对每个 DPI 值自定义字体大小和内边距值。
- 在不同密度的设备上测试应用程序，以确保外观和布局正确调整。

根据 DPI 选择样式

Flex 支持根据 CSS 中的目标操作系统和应用程序 DPI 值来应用样式。您可以在样式表中使用 @media 规则来应用样式。 @media 规则是 CSS 规范的一部分； Flex 扩展了此规则，将额外的属性 application-dpi 和 os-platform 包括进来。通过这些属性，可以根据应用程序 DPI 和运行应用程序的平台来选择性地应用样式。

下面的示例将 Spark Button 控件的默认 fontSize 样式属性设置为 12。如果设备使用 240 DPI，并在 Android 操作系统中运行，则 fontSize 属性为 10。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/MediaQueryValuesMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="320">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        @namespace mx "library://ns.adobe.com/flex/mx";
        s|Button {
            fontSize: 12;
        }
        @media (os-platform: "Android") and (application-dpi: 240) {
            s|Button {
                fontSize: 10;
            }
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

application-dpi 属性的值

将 application-dpi CSS 属性与在根应用程序上设置的 applicationDPI 样式属性值进行对照。以下是 application-dpi CSS 属性的有效值：

- 160
- 240
- 320

application-dpi 支持的每个值对应于 DPIClassification 类中的一个常量。

os-platform 属性的值

将 os-platform CSS 属性与 Flash Player 的 flash.system.Capabilities.version 属性值进行匹配。以下是 os-platform CSS 属性的有效值：

- Android
- iOS
- Macintosh
- Linux
- QNX
- Windows

匹配时不区分大小写。

如果所有条目均不匹配，则 Flex 会将前三个字符与受支持平台列表进行对照，来查找次要匹配项。

application-dpi 和 **os-platform** 属性的默认值

如果您未显式定义包含 application-dpi 或 os-platform 属性的表达式，则假定这些表达式都匹配。

@media 规则中的运算符

@media 规则支持常用的运算符 “and” 和 “not”。同时还支持以逗号分隔的列表。如果以逗号分隔表达式，意味着各项之间是 “or” 的关系。

使用 “not” 运算符时， “not” 必须是表达式中的第一个关键字。该运算符将对整个表达式求反，而不仅仅是对 “not” 后面的属性求反。由于存在 bug SDK-29191， “not” 运算符后面必须跟有一个介质类型 （例如 “all”），然后是一个或多个表达式。

下面的示例显示了其中一些常用运算符的使用方法：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/MediaQueryValuesMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="320">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        @namespace mx "library://ns.adobe.com/flex/mx";
        /* Every os-platform @ 160dpi */
        @media (application-dpi: 160) {
            s|Button {
                fontSize: 10;
            }
        }
        /* IOS only @ 240dpi */
        @media (application-dpi: 240) and (os-platform: "IOS") {
            s|Button {
                fontSize: 11;
            }
        }
        /* IOS at 160dpi or Android @ 160dpi */
        @media (os-platform: "IOS") and (application-dpi:160), (os-platform: "ANDROID") and (application-
dpi: 160) {
            s|Button {
                fontSize: 13;
            }
        }
        /* Every os-platform except Android @ 240dpi */
        @media not all and (application-dpi: 240) and (os-platform: "Android") {
            s|Button {
                fontSize: 12;
            }
        }
        /* Every os-platform except IOS @ any DPI */
        @media not all and (os-platform: "IOS") {
            s|Button {
                fontSize: 14;
            }
        }
    </fx:Style>

</s:ViewNavigatorApplication>
```

根据 DPI 选择位图资源

位图图像资源通常只能在其设计分辨率下呈现出最佳效果。在设计适用于多种分辨率的应用程序时，这点局限性将带来许多难题。此问题的解决方法是创建多个位图，每个位图对应一个分辨率，然后根据应用程序的 runtimeDPI 属性值加载相应的位图。

Spark BitmapImage 和 Image 组件具有 Object 类型的 source 属性。通过该属性可以传递一个用于定义要使用的资源的类。在此情况下，应传递 MultiDPIBitmapSource 类，以根据 runtimeDPI 属性的值映射不同的源。

下面的示例将根据 DPI 加载不同的图像：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/MultiSourceView3.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Image with MultiDPIBitmapSource">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;

            private function doSomething():void {
                /* The MultiDPIBitmapSource's source data. */
                myTA.text =
myImage.source.getSource(FlexGlobals.topLevelApplication.applicationDPI).toString();
            }


        ]]>
    </fx:Script>
    <s:Image id="myImage">
        <s:source>
            <s:MultiDPIBitmapSource
                    source160dpi="assets/low-res/bulldog.jpg"
                    source240dpi="assets/med-res/bulldog.jpg"
                    source320dpi="assets/high-res/bulldog.jpg"/>
        </s:source>
    </s:Image>
    <s:Button id="myButton" label="Click Me" click="doSomething()"/>
    <s:TextArea id="myTA" width="100%"/>
</s:View>
```

在桌面应用程序中将 BitmapImage 和 Image 类与 MultiDPIBitmapSource 配合使用时，将对映射源使用 source160dpi 属性。

Button 控件的 icon 属性也以类作为参数。因此，也可以使用 MultiDPIBitmapSource 对象作为 Button 图标的源。可以将图标源定义为内联形式，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/MultiSourceView2.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Icons Inline">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;

            private function doSomething():void {
                /* The MultiDPIBitmapSource's source data. */
                myTA.text =
dogButton.getStyle("icon").getSource(FlexGlobals.topLevelApplication.applicationDPI).toString();
            }
        ]]>
    </fx:Script>
    <s:Button id="dogButton" click="doSomething()">
        <s:icon>
            <s:MultiDPIBitmapSource id="dogIcons"
                    source160dpi="@Embed('../../assets/low-res/bulldog.jpg')"
                    source240dpi="@Embed('../../assets/med-res/bulldog.jpg')"
                    source320dpi="@Embed('../../assets/high-res/bulldog.jpg')"/>
        </s:icon>
    </s:Button>
    <s:TextArea id="myTA" width="100%"/>
</s:View>
```

如下例所示，还可以通过在 <fx:Declarations> 块中声明图标并通过数据绑定分配源来定义图标：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/MultiSourceView1.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:mx="library://ns.adobe.com/flex/mx"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Icons in Declarations">
    <fx:Declarations>
        <s:MultiDPIBitmapSource id="dogIcons"
                source160dpi="@Embed('../../assets/low-res/bulldog.jpg')"
                source240dpi="@Embed('../../assets/med-res/bulldog.jpg')"
                source320dpi="@Embed('../../assets/high-res/bulldog.jpg')"/>
    </fx:Declarations>
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;

            private function doSomething():void {
                /* The MultiDPIBitmapSource's source data. */
                myTA.text = dogIcons.getSource(FlexGlobals.topLevelApplication.applicationDPI).toString();
            }
        ]]>
    </fx:Script>
    <s:Button id="dogButton" icon="{dogIcons}" click="doSomething()"/>
    <s:TextArea id="myTA" width="100%"/>
</s:View>
```

如果 runtimeDPI 属性映射到 source*XXX*dpi 属性，而该属性值为 null 或空字符串 ("")，则 Flash Player 使用紧邻的更高密度的属性作为源。如果该值仍是 null 或空，则使用紧邻的更低密度。如果该值仍为 null 或空，Flex 会分配 null 作为源且不显示任何图像。换句话说，您无法显式指定对特定 DPI 不显示图像。

根据 DPI 选择外观资源

默认移动设备外观的构造函数中的逻辑根据 applicationDPI 属性值选择资源。这些类将选择与目标 DPI 值最匹配的资源。在设计无论是否发生 DPI 缩放都能正常显示的自定义外观时，应使用 applicationDPI 属性，而不要使用 runtimeDPI 属性。

例如，spark.skins.mobile.ButtonSkin 类使用 switch/case 语句选择针对特定 DPI 值而设计的 FXG 资源，类似于下面的示例：

```
switch (applicationDPI) {
    case DPIClassification.DPI_320: {
        upBorderSkin = spark.skins.mobile320.assets.Button_up;
        downBorderSkin = spark.skins.mobile320.assets.Button_down;
        ...
        break;
    }
    case DPIClassification.DPI_240: {
        upBorderSkin = spark.skins.mobile240.assets.Button_up;
        downBorderSkin = spark.skins.mobile240.assets.Button_down;
        ...
        break;
    }
}
```

除了有条件地选择 FXG 资源外，移动设备外观类还会设置其它样式属性的值，例如布局间隙和布局内边距。这些设置取决于目标设备的 DPI。

未设置 **applicationDPI**

如果未设置 applicationDPI 属性，则外观默认使用 runtimeDPI 属性。此机制可保证无论是否发生 DPI 缩放，值基于 applicationDPI 属性而非 runtimeDPI 属性的外观都将使用相应的资源。

创建自定义外观时，可以选择忽略 applicationDPI 设置。这样做的结果是，外观仍根据目标设备的 DPI 进行缩放，但如果资源不是专为该 DPI 值而设计的，则外观可能无法显示出最佳效果。

在 **CSS** 中使用 **applicationDPI**

可以在 CSS @media 选择器中使用 applicationDPI 属性值来自定义移动设备或平板电脑应用程序所用的样式，而无需创建自定义外观。有关更多信息，请参阅第 110 页的 " 根据 DPI 选择样式 "。

手动确定缩放比例因子和当前 DPI

要手动指示移动设备或平板电脑应用程序根据目标设备的 DPI 值选择资源，可以在运行时计算缩放比例因子。为此，请将 runtimeDPI 属性值除以 applicationDPI 样式属性值。

```
import mx.core.FlexGlobals;
var curDensity:Number = FlexGlobals.topLevelApplication.runtimeDPI;
var curAppDPI:Number = FlexGlobals.topLevelApplication.applicationDPI;
var currentScalingFactor:Number = curDensity / curAppDPI;
```

可以使用算得的缩放比例因子来手动选择资源。下面的示例定义了对应于每个 DPI 值的位图资源自定义位置。然后从该自定义位置加载图像：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/DensityMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark"
     firstView="views.DensityView1"
     applicationDPI="240" initialize="initApp()">

    <fx:Script>
        <![CDATA[
                [Bindable]
                public var densityDependentDir:String;
                [Bindable]
                public var curDensity:Number;
                [Bindable]
                public var appDPI:Number;
                [Bindable]
                public var curScaleFactor:Number;

                public function initApp():void {
                    curDensity = runtimeDPI;
                    appDPI = applicationDPI;
                    curScaleFactor  =  appDPI / curDensity;
                    switch (curScaleFactor) {
                        case 1: {
                            densityDependentDir = "../../assets/low-res/";
                            break;
                        }
                        case 1.5: {
                            densityDependentDir = "../../assets/med-res/";
                            break;
                        }
                        case 2: {
                            densityDependentDir = "../../assets/high-res/";
                            break;
                        }
                    }
                }
        ]]>
    </fx:Script>

</s:ViewNavigatorApplication>
```

使用缩放比例因子的视图如下所示:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/DensityView1.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Home"
        creationComplete="initView()">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import mx.core.FlexGlobals;
            [Bindable]
            public var imagePath:String;
            private function initView():void {
                label0.text = "App DPI:" + FlexGlobals.topLevelApplication.appDPI;
                label1.text = "Cur Density:" + FlexGlobals.topLevelApplication.curDensity;
                label2.text = "Scale Factor:" + FlexGlobals.topLevelApplication.curScaleFactor;
                imagePath = FlexGlobals.topLevelApplication.densityDependentDir + "bulldog.jpg";

                ta1.text = myImage.source.toString();
            }
        ]]>
    </fx:Script>

    <s:Image id="myImage" source="{imagePath}"/>
    <s:Label id="label0"/>
    <s:Label id="label1"/>
    <s:Label id="label2"/>
    <s:TextArea id="ta1" width="100%"/>
</s:View>
```

重写默认 DPI

设置应用程序 DPI 值后，应用程序将根据所在设备报告的 DPI 值来进行缩放。在某些情况下，设备会报告错误的 DPI 值，或者您希望重写默认 DPI 选择方法以使用自定义缩放方法。

可以通过重写默认 DPI 映射来重写应用程序的默认缩放行为。例如，如果某个设备将 160 DPI 错误地报告为 240 DPI，您可以创建自定义映射来查找此设备，并将设备划分到 160 DPI 类别中。

要重写特定设备的 DPI 值，请将 Application 类的 runtimeDPIProvider 属性指向 RuntimeDPIProvider 类的子类。在子类中，重写 runtimeDPI getter 并添加用于提供自定义 DPI 映射的逻辑。不要为框架中的其它类添加依赖项，例如 UIComponent。该子类只能调用到 Player API 中。

以下示例为其 Capabilities.os 属性与 “Mac 10.6.5” 匹配的设备设置自定义 DPI 映射。

```
package {
import flash.system.Capabilities;
import mx.core.DPIClassification;
import mx.core.RuntimeDPIProvider;
public class DPITestClass extends RuntimeDPIProvider {
    public function DPITestClass() {
    }

    override public function get runtimeDPI():Number {
        // Arbitrary mapping for Mac OS.
        if (Capabilities.os == "Mac OS 10.6.5")
            return DPIClassification.DPI_320;

        if (Capabilities.screenDPI < 200)
            return DPIClassification.DPI_160;

        if (Capabilities.screenDPI <= 280)
            return DPIClassification.DPI_240;

        return DPIClassification.DPI_320;
    }
}
}
```

下面的应用程序使用 DPITestClass 确定用于缩放的运行时 DPI 值。它指向 ViewNavigatorApplication 类的 runtimeDPIProvider 属性 :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/DPIMappingOverrideMain.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
     xmlns:s="library://ns.adobe.com/flex/spark"
     firstView="views.DPIMappingView"
     applicationDPI="160"
     runtimeDPIProvider="DPITestClass">

</s:ViewNavigatorApplication>
```

下面是关于 RuntimeDPIProvider 类的子类的另一个示例。在此示例中，自定义类将检查设备的 X 和 Y 分辨率来确定设备所报告的 DPI 值是否正确:

```
package
{
import flash.system.Capabilities;
import mx.core.DPIClassification;
import mx.core.RuntimeDPIProvider;
public class SpecialCaseMapping extends RuntimeDPIProvider {
    public function SpecialCaseMapping() {
    }

    override public function get runtimeDPI():Number {
        /* A tablet reporting an incorrect DPI of 240. We could use
            Capabilities.manufacturer to check the tablet's OS as well. */
        if (Capabilities.screenDPI == 240 &&
            Capabilities.screenResolutionY == 1024 &&
            Capabilities.screenResolutionX == 600) {
            return DPIClassification.DPI_160;
        }

        if (Capabilities.screenDPI < 200)
            return DPIClassification.DPI_160;

        if (Capabilities.screenDPI <= 280)
            return DPIClassification.DPI_240;

        return DPIClassification.DPI_320;
    }
}
}
```

第 5 章：文本

在移动设备应用程序中使用文本

在移动设备应用程序中使用文本的指导原则

有些 Spark 文本控件已经过优化，适合在移动设备应用程序中使用。可能的情况下，请使用以下文本控件：

- Spark TextArea
- Spark TextInput
- Spark Label

这些允许用户交互的文本控件（TextArea 和 TextInput）使用 StageText 类作为基础输入机制。StageText 挂接到基础 OS 的本机文本控件。因此，这些文本控件像本机控件一样，而不像典型的 Flex 控件那样运行。

在设备上使用 StageText 的优点是可获得以下功能（如果设备支持这些功能）：

- 本机性能以及软键盘的观感
- 自动完成
- 自动纠正
- 触控式文本选择
- 可自定义的软键盘
- 按键限制

Adobe 宣讲师 Christian Cantrell 介绍了使用基于 StageText 的控件的优缺点。

基于 TextField 版的 TextArea 和 TextInput 控件也供可用。您可以使用这些版本来嵌入字体，或使用在基于 StageText 版上不可用的某些其它功能。

移动设备文本控件的外观

创建移动设备应用程序时，Flex 将自动应用移动设备主题。因此，Spark TextInput 和 TextArea 控件默认情况下使用下列基于 StageText 的移动设备外观：

- StageTextAreaSkin
- StageTextInputSkin

StageTextAreaSkin 和 StageTextInputSkin 类已针对移动设备应用程序进行了优化，并且基于 StageTextSkinBase 类。它们充当本机文本输入类的包装器。但是，它们不支持非基于 TextField 外观的下列功能：

基于 TextField 的控件支持	基于 TextField 的控件不支持
滚动表单 文本测量 剪切 嵌入字体 小数 Alpha 值 Flash 文本引擎 (FTE) 访问低级别的键盘事件，如 keyUp 和 keyDown	文本布局框架 (TLF) 双向性和镜像 压缩字体格式 (CFF) 使用 RichEditableText 呈现文本 HTML 文本

其中一些限制可通过使用基于 TextField 的版本来解决。要使用基于 TextField 版的文本输入控件，请将其外观类指向 TextField 版的 TextInputSkin 和 TextAreaSkin；例如：

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin" text="TextField-based Skin"/>
```

Spark Label 控件不使用外观，但也不使用 TLF。

移动设备应用程序中的 TLF

通常情况下，移动设备应用程序中应避免出现使用文本布局框架 (TLF) 的文本控件。TextArea 和 TextInput 控件的移动设备外观已针对移动设备应用程序进行优化，不像其桌面应用程序和基于 Web 的应用程序一样使用 TLF。TLF 在应用程序中用于提供一组丰富的控件，以控制文本呈示。

在移动设备应用程序中应避免使用以下文本控件，因为它们使用 TLF 并且其外观未针对移动设备应用程序进行优化：

- Spark RichText
- Spark RichEditableText

使用软键盘进行输入

当用户将焦点放在可接受输入的文本控件上时，无键盘的移动设备将显示软键盘。您可以在某种程度上控制软键盘的可用按键和其它属性。例如，您可以启用自动纠正和自动大写功能，可以从几种预定义的键盘布局中进行选择。

有关更多信息，请参阅第 128 页的“在移动设备应用程序中使用软键盘”。

使用文本输入控件进行滚动

TextInput 和 TextArea 控件的默认移动设备外观不支持滚动表单。也就是说，您不能在要求控件进行滚动的表单或视图中显示这些控件。如果您这样做，控件的执行方式会导致出现可视失真。

要在滚动容器中使用文本输入控件，请使用基于 TextField 的外观，而不是基于 StageText 的外观。有关更多信息，请参阅第 58 页的“与 StageText 相关的滚动注意事项”。

使用文本输入控件进行过渡

为了使动画更加平滑，每当播放动画时，运行时都会将 StageText 控件替换为从这些控件捕获的位图。这会导致在过渡动画的最初会有一个轻微的延迟，并在开始和结束动画时造成一些视觉效果。

产生轻微延迟的原因是需要时间来捕获组件中文本的位图表示形式。当基于 StageText 的控件的面积增大且数量增多时，延迟也相应地增加。要降低延迟，请避免以动画形式表现大面积的基于 StageText 的组件或数量众多的这种组件。

使用文本输入控件的弹出窗口

当显示弹出窗口时，最顶层弹出窗口以外的基于 StageText 的文本输入将被替换为位图表示形式。因此：

- 使用模态弹出窗口，而不要使用非模态弹出窗口。当显示模态弹出窗口时，弹出窗口以外的组件将不再能够与用户交互。在这种情况下，使用位图替换 StageText 不太明显。
- 基于 StageText 的组件应当仅在实施 IFocusManagerContainer 接口的弹出窗口内使用。例如，使用 SkinnableContainer 或其派生类型之一作为弹出窗口的基础。
- 当较低层中的文本组件应保持活动状态时，使用 Callout 容器。如果文本组件中包含 Callout，该文本组件将保持活动状态并将 Callout 的箭头设置为指向该文本组件。这样可以自然地提示用户该文本组件仍然可以使用。
- 避免同时使用多个弹出窗口。如果同时有多个弹出窗口可见，仅可以与最顶层的弹出框口进行交互。但是，如果这些弹出窗口没有发生重叠，则用户将无法确定哪一个弹出窗口位于最顶层。
- 当非模态弹出窗口与文本控件重叠时，调整弹出窗口的位置以达到几乎完全重叠。如果用户看不到文本，他们将不太可能认为仍然可以与文本控件进行交互。

在移动设备应用程序中嵌入字体

您不能在使用 StageText 的文本输入控件中使用嵌入字体。应对文本输入控件使用基于 TextField 的外观。也不能对 Label 控件使用嵌入字体，因为该控件使用 FTE，要求使用基于 CFF 的字体。 CFF 字体并不十分适合于移动设备应用程序。

有关更多信息，请参阅第 139 页的 “ 在移动设备应用程序中嵌入字体 ”。

StageText 控件类层次结构

StageText 类 （TextInput 和 TextArea）具有复杂的类层次结构。基础类自身具有下列层次结构：

```
    UIComponent
        |
SkinnableComponent
        |
SkinnableTextBase
        |
TextInput/TextArea
```

与所有 Spark 类一样，外观类有自己的层次结构：

```
    UIComponent
        |
    MobileSkin    StyleableStageText
        |               |
StageTextSkinBase: textDisplay
        |
StageTextInputSkin/StageTextAreaSkin
```

在基础外观类中， textDisplay 属性以 StyleableStageText 对象的形式提供了到本机文本输入的挂接。此类也负责定义基于 StageText 的文本输入控件上的可用样式。

在移动设备应用程序中使用 Label 控件

Spark Label 控件最适用于不可编辑、不可选择的单文本行。

下例在移动设备应用程序中使用了一个简单的 Label 控件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleLabel.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Simple Label">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="This is a simple Label control."/>

</s:View>
```

Label 控件使用 FTE，它的性能不如已针对移动设备应用程序进行优化的文本控件 （例如 TextInput 和 TextArea）。但是 Label 控件不使用 TLF，因此性能通常优于实现 TLF 的控件，例如 RichText 和 RichEditableText。

通常，在移动设备应用程序中，应谨慎使用 Spark Label 控件。在外观或项呈示器中不要使用 Spark Label 控件。在创建基于 ActionScript 的项呈示器时，请使用 StyleableTextField 类来呈示文本。对于基于 MXML 的组件，您仍然可以使用 Label。

在移动设备应用程序中嵌入字体时请不要使用 Label 控件，因为 Label 控件使用 CFF。应改为使用基于 TextField 版的 TextArea 控件。有关更多信息，请参阅第 139 页的 “ 在移动设备应用程序中嵌入字体 ”。

在移动设备应用程序中使用 TextArea 控件

Spark TextArea 控件是允许用户输入和编辑多行文本的文本输入控件。它针对移动设备应用程序进行了优化。

TextArea 控件的默认行为是使用提供了到基础 OS 本机方法的挂接的软键盘。因此，它支持自动纠正、自动完成和软键盘自定义等功能。

下例在移动设备应用程序中使用了一个 TextArea 控件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleTextArea.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Simple TextArea">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <fx:Script>
        <![CDATA[
            // Note the use of \n to add line feeds/carriage returns
            // and \" to add quotation marks.
            [Bindable]
           public var myText:String ="\"Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam
nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.\"\n\n\"Ut wisi enim ad minim
veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.\"";
        ]]>
    </fx:Script>

    <!-- Basic TextArea control with multiple lines of text. -->
    <s:TextArea id="myTA" height="75%" text="{myText}"
                paddingLeft="20" paddingTop="20"
                paddingRight="20" paddingBottom="20"/>

</s:View>
```

在移动设备应用程序中， TextArea 控件默认为其外观使用 StageTextAreaSkin 类。此外观使用 StyleableStageText 类而不是 RichEditableText 类来呈示文本。因此，TextArea 控件不支持 TLF。它仅支持非移动设备外观 TextArea 控件上的一部分可用样式。

如果您需要一个非交互式多行文本块，请将 TextArea 控件的 editable 属性设置为 false。（运行时不会考虑 selectable 属性。）您也可以通过将 borderVisible 属性设置为 false 来删除边框。您可以通过设置 contentBackgroundColor 和 contentBackgroundAlpha 属性来更改背景颜色。

下例创建了一个与应用程序背景融合在一起的非交互式文本块：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/BlockOfText.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Block of Text">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s:HGroup>
        <s:Image source="@Embed(source='../../assets/myImage.jpg')" width="30%"/>
        <!-- Create a multi-line block of text. -->
        <s:TextArea width="65%"
                editable="false"
                borderVisible="false"
                contentBackgroundColor="0xFFFFFF"
                contentBackgroundAlpha="0"
                height="400"
                text="Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat."/>
    </s:HGroup>

</s:View>
```

因为 TextArea 控件不支持 TLF，所以无法使用 textFlow、 content 或 selectionHighlighting 属性。此外，也无法使用以下方法：

- getFormatOfRange()
- setFormatOfRange()

在移动设备应用程序中使用 TextInput 控件

Spark TextInput 控件是允许用户输入和编辑单行文本的文本输入控件。它针对移动设备应用程序进行了优化。

TextInput 控件的默认行为是使用提供了到基础 OS 本机方法的挂接的软键盘。因此，它支持自动纠正、自动完成和软键盘自定义等功能。

下例显示了在移动设备应用程序中具有提示文本和自定义焦点外框的 TextInput 控件：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/SimpleTextInput.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Simple TextInput">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout
            paddingTop="20"
            paddingLeft="20"
            paddingRight="20"/>
    </s:layout>

    <s:TextInput
        prompt="Enter text here"
        focusColor="green"
        focusThickness="5"
        focusAlpha=".1"/>
    <s:TextInput
        prompt="Enter text here, too"
        focusColor="red"
        focusThickness="5"
        focusAlpha=".1"/>
</s:View>
```

在移动设备应用程序中，TextInput 控件默认为其外观使用 StageTextInputSkin 类。此外观使用 StyleableStageText 类而不是 RichEditableText 类来呈示文本。因此，TextInput 控件不支持 TLF。它仅支持非移动设备外观 TextInput 控件上的一部分可用样式。

在移动设备应用程序中使用 RichText 和 RichEditableText 控件

在移动设备应用程序中，应尽量避免使用 RichText 和 RichEditableText 控件。这些控件不具有移动设备外观，并且未针对移动设备应用程序进行优化。如果使用这些控件，则需要使用 TLF，这样做会产生非常高昂的计算成本。

MX 文本控件

不能在移动设备应用程序中使用 MX 文本控件，例如 MX Text 和 MX Label。请改用相当于 Spark 的控件。

在移动设备应用程序中设置文本输入控件的样式

TextInput 和 TextArea 控件仅支持移动设备主题中的一部分样式。 StyleableStageText 类定义了这些样式。

以下是移动设备应用程序中 TextInput 和 TextArea 支持的仅有的几个样式：

- color
- contentBackgroundAlpha
- contentBackgroundColor
- 焦点外框样式：focusAlpha、 focusBlendMode、 focusColor 和 focusThickness
- fontFamily
- fontStyle
- fontSize
- fontWeight

- locale
- 内边距样式：paddingBottom、 paddingLeft、 paddingRight 和 paddingTop
- showPromptWhenFocused
- textAlign

在移动设备应用程序中， Label 控件支持这些样式以及 textDecoration 样式。

使用 **fontFamily** 样式

在默认的移动设备外观中， fontFamily 属性不支持逗号分隔的字体列表。您只能指定一个字体，运行时会尝试将此字体映射到设备上的现有字体。

例如，如果您指定 “Arial”，设备会以 Arial 字体呈现文本 （如果该字体可用）。如果该字体不可用，运行时会做出最佳预测，确定使用什么类型的字体来代替它。如果您指定了运行时无法识别的字体名称，设备会以默认字体呈现文本。移动设备上的默认设置通常是 sans-serif 字体。

您可以指定 _sans、 _serif 或 _typewriter，以始终在移动设备上相应地获得 sans-serif、 serif 或代码字体。

下例显示了根据 fontFamily 样式的值呈现文本的方式：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/FontFamilyExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="The fontFamily style">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>
    <s:TextInput prompt="This is _sans" fontSize="14" fontFamily="_sans"/>
    <s:TextInput prompt="This is _serif" fontSize="14" fontFamily="_serif"/>
    <s:TextInput prompt="This is _typewriter" fontSize="14" fontFamily="_typewriter"/>
    <s:TextInput prompt="This is Arial" fontSize="14" fontFamily="arial"/>
    <s:TextInput prompt="This is Times" fontSize="14" fontFamily="times"/>
    <s:TextInput prompt="This is Times New Roman" fontSize="14" fontFamily="Times New Roman"/>
    <!-- Try a gibberish font name to see what the device's default font is: -->
    <s:TextInput prompt="This is bugblatter" fontSize="14" fontFamily="bugblatter"/>
</s:View>
```

在文本输入控件上创建自定义移动设备外观

通过使用 MXML 和 ActionScript，您可以在移动设备应用程序中控制文本输入控件的一些可视外观和行为。例如，可以设置 TextArea 和 TextInputFor 控件上的边框颜色或切换边框的外观。在某些情况下，您必须创建一个自定义外观以更改文本控件特定部分的外观。

StageTextAreaSkin 和 StageTextInputSkin 类定义了移动设备主题中默认的 TextInput 和 TextArea 外观。这些外观通过 StageTextSkinBase 类确定了其大部分布局和主色逻辑。

要创建自定义外观，请创建定义新外观的自定义 StageTextSkinBase 类。然后创建扩展此自定义类的自定义 StageTextAreaSkin 或 StageTextInputSkin 类。

有关为移动设备主题创建自定义外观的更多信息，请参阅第 141 页的 “ 移动设备外观设计的基础知识 ”。

限制文本输入控件中的按键

当为文本输入控件使用基于 StageText 的外观时，可以通过使用 TextInput 或 TextArea 控件的 restrict 属性来限制所允许的字符。

restrict 属性的默认值是 null，表示默认情况下用户可以输入任何字符。

restrict 属性包含一系列允许使用的字符。对于范围，请使用连字符 (-)。不要以空格、逗号或其它字符分隔范围，除非您希望将该字符包含在定义中。下例显示了几个使用限制语法的示例：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/RestrictStrings.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Examples of restrict">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="20" paddingLeft="20" paddingRight="20"/>
    </s:layout>

    <s:TextInput prompt="Alpha-numeric only" restrict="a-zA-Z0-9"/>
    <s:TextInput prompt="Numbers only" restrict="0-9"/>
    <s:TextInput prompt="All chars, only uppercase alpha" restrict="^a-z"/>
    <!-- ASCII chars 32 (space) through 126 (tilde) only: -->
    <s:TextInput prompt="" restrict="\u0020-\u007E"/>
    <s:TextInput prompt="All chars but not the caret or hyphen" restrict="^\^\-"/>
</s:View>
```

restrict 属性的字符串值从左向右读；脱字符 (^) 后的所有字符都是禁用字符。例如：

```
ta1.restrict = "A-Z^Q"; // All uppercase alpha characters, but exclude Q
```

如果字符串中的第一个字符是脱字符 (^)，则表示允许使用除脱字符后的字符外的所有字符。例如：

```
ta1.restrict = "^a-z"; // All characters, but exclude lowercase alpha
```

可以使用反斜线字符来转义特殊字符。例如，要限制使用脱字符：

```
ta1.restrict = "^\^"; // All characters, but exclude the caret
```

您可以使用 \u 来输入 ASCII 键码；例如：

```
ta1.restrict = "\u0020-\u007E"; // ASCII chars 32 through 126 only
```

移动设备应用程序中用户与文本的交互

可以针对文本控件使用各种手势，例如滑动手势。下面的示例将侦听滑动事件，并告知您滑动的方向：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/TextAreaEventsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="TextArea swipe event"
        viewActivate="view1_viewActivateHandler()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            import flash.events.TransformGestureEvent;
            import mx.events.FlexEvent;

            protected function swipeHandler(event:TransformGestureEvent):void {
                // event.offsetX shows the horizontal direction of the swipe (1 is right, -1 is left)
                swipeEvent.text = event.type + " " + event.offsetX;
                if (swipeText.text.length == 0) {
                    swipeText.text = "Swipe again to make text go away."
                }
                else {
                    swipeText.text = "";
                }
            }

            protected function view1_viewActivateHandler():void {
                swipeText.addEventListener(TransformGestureEvent.GESTURE_SWIPE,swipeHandler);
            }

        ]]>
    </fx:Script>
    <s:VGroup>
        <s:TextArea id="swipeText" height="379"
                    editable="false" selectable="false"
                    text="Swipe to make text go away."/>
        <s:TextInput id="swipeEvent" />
    </s:VGroup>

</s:View>
```

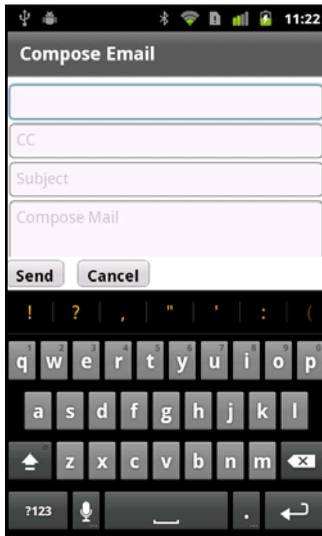
触摸加拖动的手势用于选择文本，但前提是文本控件可以选择或编辑。在某些情况下，您可能并不希望当用户在文本控件上执行触摸加拖动手势或滑动手势时文本被选择。在这种情况下，请将 selectable 和 editable 属性设置为 false，或者在滑动事件处理程序中通过调用 selectRange(0,0) 方法重置选项。

如果文本位于 Scroller 中，则只有在文本组件外部做出该手势时， Scroller 才会滚动。

在移动设备应用程序中使用软键盘

许多设备都不包括硬件键盘。在必要时，这些设备会使用在屏幕中打开的键盘。软键盘又称做屏幕键盘或虚拟键盘，当用户输入信息或取消操作后，软键盘将会关闭。

下图显示了一个使用了软键盘的应用程序：



根据启用软键盘的组件，软键盘有不同的功能集：

- 本机功能：用于默认文本输入控件 TextArea 和 TextInput 的键盘，它挂接到本机接口，具有自动纠正、自动完成和自定义键盘布局等功能。对完整功能集的支持已内置到基于 StageText 默认外观类的文本输入控件中。并非所有设备都支持所有本机功能。
- 有限功能：用于 TextArea 和 TextInput 之外的任何其它控件的键盘，或者在控件使用基于 TextField 的外观时，用于 TextArea 和 TextInput 的键盘。有限功能集不支持本机 OS 功能，如自动纠正、自动完成和自定义键盘布局。

由于该键盘会占用一部分屏幕， Flex 必须确保应用程序在缩小的屏幕区域中仍能正常工作。例如，用户选择 TextInput 控件时会打开软键盘。键盘打开后， Flex 自动根据可用屏幕区域调整应用程序的大小。 Flex 随后可以调整所选 TextInput 控件的位置，使其在键盘之上可见。

博客 Peter Elst 发表了关于在 Flex 移动设备应用程序中控制软键盘的文章。

在移动设备 Flex 应用程序中打开软键盘

可使用三种方法在移动设备应用程序中打开软键盘：

- 将焦点放在具有 TextInput 或 TextArea 之类文本输入控件的控件上
- 将控件的 needsSoftKeyboard 属性设置为 true，并将焦点放在该控件上
- 对控件调用 requestSoftKeyboard() 方法 （不在 iOS 上）

键盘将保持打开状态，直至发生以下操作之一：

- 用户将焦点移动到不接收文本输入的控件。当用户手动指向另一个文本输入控件，或者用户按键盘上的返回键，应用程序随之将焦点移到另一个控件时就会发生这种情况。

  如果焦点移动到其他文本输入控件，或移动到 needsSoftKeyboard 属性为 true 的控件，则键盘保持打开状态。
- 用户按下设备上的后退按钮以取消输入。
- 您可通过编程将焦点更改到非交互式控件或将 stage.focus 设置为 null。

为用户提供文本输入控件

如果为用户提供了文本输入控件，当用户将焦点放在该控件上时软键盘就会出现，除非将 editable 属性设置为 false。

TextInput 和 TextArea 控件的默认行为是使用 StageText 类来呈现文本。因此，为这些控件显示的键盘支持自动纠正、自动大写和键盘类型等本机功能。并非所有功能在所有设备上都受支持。

如果将外观类更改为对文本输入控件使用基于 TextField 的外观，这些功能将受到限制。键盘本身相同，但不支持本机功能。

设置 needsSoftKeyboard 属性

可以配置非输入控件，以为其打开软键盘，例如 Button 或 ButtonBar 控件。要在文本输入控件外的其它控件获得焦点时打开键盘，请将该控件的 needsSoftKeyboard 属性设置为 true。所有 Flex 组件都从 InteractiveObject 类继承此属性。

为 TextInput 和 TextArea 之外的任何控件打开的键盘都不支持自动大写、自动纠正和自定义键盘类型等本机功能。

注：当文本输入控件获得焦点时，始终会打开键盘。文本输入控件将忽略 needsSoftKeyboard 属性，设置该属性不会影响文本输入控件。

调用 requestSoftKeyboard() 方法

要以编程方式打开软键盘，可以调用 requestSoftKeyboard() 方法。对于调用此方法的对象，必须也将其 needsSoftKeyboard 属性设置为 true。此方法会将焦点更改到调用此方法的对象上，如果设备没有硬件键盘，则会打开软键盘。

InteractiveObject 类定义了 requestSoftKeyboard() 方法。因此，您可以在作为 InteractiveObject 子类的任何组件上调用此方法。

如果在 TextArea 或 TextInput 控件上调用 requestSoftKeyboard() 方法，则自动纠正和自动大写等本机键盘功能将是受支持的（如果设备支持这些功能）。

requestSoftKeyboard() 方法不适用于 iOS 设备。

使用软键盘的本机功能

StageTextInputSkin 和 StageTextAreaSkin 类定义了移动设备应用程序中 TextInput 和 TextArea 控件的外观。这些外观使用 StageText 类来呈现文本，并挂接到软键盘的本机功能中。这些功能包括：

- 自动纠正
- 自动大写
- 自定义返回键标签
- 自定义键盘类型

文本输入控件也可以使用基于 TextField 的外观来呈现文本。这些外观不支持本机功能。但是，它们为基础文本控件提供了附加功能，如支持滚动表单、嵌入字体、访问 keyUp 和 keyDown 事件、剪切、文本测量和小数 Alpha 值。

要使用基于 TextField 的外观，请将文本输入控件的 skinClass 属性设置为指向 TextInputSkin 和 TextAreaSkin 类。例如：

```
<s:TextInput skinClass="spark.skins.mobile.TextInputSkin"/>
<s:TextArea skinClass="spark.skins.mobile.TextAreaSkin"/>
```

在 Flex 移动设备应用程序中对软键盘使用自动纠正功能

自动纠正是 OS 试图纠正拼写错误并对用户输入应用预测性键入的一种行为。根据设备类型，此行为可能以文本之上的气泡提示框、软键盘扩展或其它某种方式实现。

您可以通过将文本输入控件的 autoCorrect 属性设置为 true 来在移动设备应用程序中对软键盘使用自动纠正功能。这是默认值。

在下例中您可以打开和关闭自动纠正功能。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/AutoCorrectionExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Auto-Correction">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextInput prompt="Enter your text" autoCorrect="{myCB.selected}"/>
    <s:CheckBox id="myCB" label="Enable auto-correct" enabled="true"/>

</s:View>
```

并非所有设备都支持自动纠正功能。如果您在不支持此功能的设备上启用或禁用 autoCorrect 属性，运行时会忽略此值，并使用设备的默认行为。

在 Flex 移动设备应用程序中对软键盘使用自动大写功能

自动大写是一种在用户输入文本时指示文本输入控件将某些字或字母设置为大写的设置。例如，您可以将所有字母设为大写，也可以自动地只将每个句子的第一个字设为大写。这样用户在移动设备应用程序中输入文本时就不必担心大写设置了，这十分方便。

您可通过在文本输入控件上设置 autoCapitalize 属性值来在软键盘中使用自动大写功能。可能值有 none、 word、 sentence 和 all。 AutoCapitalize 类定义了可能的值。默认值为 none。

在下例中您可以为自动大写功能选择不同的值。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/AutoCapitalizeExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Auto-Capitalization">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a capitalization setting:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="capTypeList" width="300" labelField="name" fontSize="12">
            <s:ArrayCollection>
                <fx:Object name="All" value="all"/>
                <fx:Object name="None" value="none"/>
                <fx:Object name="Sentence" value="sentence"/>
                <fx:Object name="Word" value="word"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput autoCapitalize="{capTypeList.selectedItem.value}"/>

</s:View>
```

并非所有设备都支持自动大写功能。如果您在不支持此功能的设备上设置 autoCapitalize 属性的值，运行时会忽略此值，并使用设备的默认值。

在 Flex 移动设备应用程序中更改软键盘类型

SoftKeyboardType 类定义了移动设备应用程序的软键盘类型。您可在文本输入控件上使用 softKeyboardType 属性选择键盘类型。

大多数键盘 （如 email 和 contact）之间只有细微差别。例如，除以 "@" 代替了话筒外， email 键盘与 contact 键盘为用户提供的按键完全相同。 url 键盘使用 "/" 符号。 number 键盘与众不同。此键盘外观类似一个计算器屏幕，焦点在数字和运算符上。

下例显示了可用的不同类型的软键盘：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/KeyboardTypes.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Keyboard Types">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a keyboard type:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="keyboardTypeList" width="300" labelField="name">
            <s:ArrayCollection>
                <fx:Object name="Contact" value="contact"/>
                <fx:Object name="Default" value="default"/>
                <fx:Object name="Email" value="email"/>
                <fx:Object name="Number" value="number"/>
                <fx:Object name="Punctuation" value="punctuation"/>
                <fx:Object name="URL" value="url"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput softKeyboardType="{keyboardTypeList.selectedItem.value}" text=""/>

</s:View>
```

并非所有软键盘类型在所有设备上都受支持。如果指定了不受支持的类型，运行时会忽略此值，而使用设备的默认值。

在 Flex 移动设备应用程序中更改软键盘上的返回键标签

软键盘弹出，用户输入文本时，必须有一种方法让用户表示他们已完成了输入，并且希望移到下一个字段或提交所输入的数据。在软键盘上，通常使用 " 返回键 " 完成此操作。此按键不会在文本输入中输入任何字符，只是通知文本输入控件用户已完成了文本输入。

ReturnKeyLabel 类为返回键定义了可能的标签。可能的值有 default、done、go、next 和 search。您可以在文本输入控件中使用 returnKeyLabel 属性指定返回键标签。

在下例中您可以选择不同的返回键标签：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/ReturnKeyLabels.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Return Key Labels">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:Label text="Select a return key label:"/>
    <s:SpinnerListContainer>
        <s:SpinnerList id="returnKeyLabelList" width="300" labelField="name">
            <s:ArrayCollection>
                <fx:Object name="Default" value="default"/>
                <fx:Object name="Done" value="done"/>
                <fx:Object name="Go" value="go"/>
                <fx:Object name="Next" value="next"/>
                <fx:Object name="Search" value="search"/>
            </s:ArrayCollection>
        </s:SpinnerList>
    </s:SpinnerListContainer>

    <s:TextInput returnKeyLabel="{returnKeyLabelList.selectedItem.value}" text=""/>

</s:View>
```

在事件或交互方面，不同的返回键类型之间没有差别。更改 returnKeyLabel 属性只会更改按键的标签。

并非所有设备都支持返回键标签的设置。如果您在不支持此功能的设备上设置 returnKeyLabel 属性的值，运行时会忽略此值，而使用设备的默认值。

在移动设备应用程序中使用与软键盘相关的事件

在移动设备上与软键盘进行交互不同于在桌面或基于 Web 的应用程序中与键盘进行交互。下表列出了与使用软键盘相关的事件：

事件	何时分派
enter	用户按返回键时。
keyDown 和 keyUp	对于基于 StageText 的外观，仅在按下和释放某些按键时。对于基于 TextField 的外观，在按下和释放所有按键时。 不会为所有设备上的所有按键分派这些事件。不要依赖这些方法来捕获软键盘的按键输入，除非对启用键盘的控件使用了基于 TestField 的外观。
softKeyboardActivating	打开键盘之前。
softKeyboardActivate	打开键盘之后。
softKeyboardDeactivate	关闭键盘之后。

要确定用户何时在软键盘上完成了工作，可以在文本输入控件上侦听 FlexEvent.ENTER 事件。按下返回键时控件就会分派此事件。通过侦听 enter 事件，您可以执行验证、更改焦点，或对最近输入的文本执行其它操作。

在某些情况下，不会分派 enter 事件。在 Android 设备上当对视图中的最后一个文本输入控件使用软键盘时，此限制就会出现。要解决此问题，请在视图的最后一个文本输入控件上将 returnKeyLabel 属性设置为 go、 next 或 search。

在下例中，当用户在软键盘上按 “Next” 键时，焦点将从一个字段更改到下一个字段。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/UseNextLikeTab.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Change Focus">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout paddingTop="10" paddingLeft="10" paddingRight="10"/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private function changeField(ti:TextInput):void {
                // Before changing focus to a new control, set the stage's focus to null:
                stage.focus = null;

                // Set focus on the TextInput that was passed in:
                ti.setFocus();
            }
        ]]>
    </fx:Script>

    <s:HGroup>
        <s:Label text="1:" paddingTop="15"/>
        <s:TextInput id="ti1" prompt="First Name"
                     width="80%"
                     returnKeyLabel="next"
                     enter="changeField(ti2)"/>
    </s:HGroup>
    <s:HGroup>
        <s:Label text="2:" paddingTop="15"/>
        <s:TextInput id="ti2" prompt="Middle Initial"
                     width="80%"
                     returnKeyLabel="next"
                     enter="changeField(ti3)"/>
    </s:HGroup>
    <s:HGroup>
        <s:Label text="3:" paddingTop="15"/>
        <s:TextInput id="ti3" prompt="Last Name"
                     width="80%"
                     returnKeyLabel="next"
                     enter="changeField(ti1)"/>
    </s:HGroup>

</s:View>
```

当用户与 TextInput 和 TextArea 控件的默认软键盘交互时，仅对一小部分按键分派 keyUp 和 keyDown 事件。要捕获所有键的各个按键行为，请使用 change 事件。只要文本输入控件的内容有更改，就会分派 change 事件。这样做的缺点是您无法访问按下的按键的属性，并且必须编写自己的按键逻辑。

下例显示了最后一个按下的按键的字符代码：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/CompareMobileKeyPresses.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Keyboard Events">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private var storedValueOfText:String = null;

            // Compare the new text against the stored value to see what key was pressed.
            private function compareKey(e:Event):void {
                var key:String = "";
                if (storedValueOfText == null) {
                 key = ti1.text.charCodeAt(0).toString(); // Capture the first key pressed.
                } else {
                    // Compare the stored value against the current value and extract the difference.
                    for (var i:int = 0; i<ti1.text.length; i++) {
                        if (ti1.text.charAt(i) == storedValueOfText.charAt(i)) {
                            // Do nothing if they're equal.
                        } else {
                            key = ti1.text.charCodeAt(i).toString();
                        }
                    }
                }
                ti2.text = "The '" + key + "' key was pressed.";
                storedValueOfText = ti1.text;
            }
        ]]>
    </fx:Script>

    <s:TextInput id="ti1" change="compareKey(event)"/>
    <s:TextInput id="ti2" editable="false"/>
</s:View>
```

如果光标位于文本输入字段的末尾，此示例仅识别最后一个按下的按键。

当用户与基于 TextField 控件的软键盘进行交互时，像 keyUp 和 keyDown 这样的事件将适用于所有按键。下例使用 keyUp 处理函数来获取当前按键，并基于按键代码对 Label 控件应用样式。因为 requestSoftKeyboard() 方法会为 Label 控件而不是 TextInput 或 TextArea 控件启用键盘，应用程序将使用基于 TextField 的键盘。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/views/RequestSoftKeyboardExample.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="requestSoftKeyboard()">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            private function handleButtonClick():void {
                myLabel.requestSoftKeyboard();
            }
            /* Ok to use keyUp handler on limited screen keyboard. */
            private function handleKeys(event:KeyboardEvent):void {
                var c:int;
                switch(event.keyCode) {
                    case(82): // 82 = "r"
                        c = 0xFF0000;
                        break;
                    case(71): // 71 = "g"
                        c = 0x00FF00;
                        break;
                    case(66): // 66 = "b"
                        c = 0x0000FF;
                        break;
                }
                event.currentTarget.setStyle("color",c);
            }
        ]]>
    </fx:Script>
    <s:Label id="myLabel" text="This is a label." needsSoftKeyboard="true" keyUp="handleKeys(event)"/>
    <s:Button id="b1" label="Click Me" click="handleButtonClick()"/>

</s:View>
```

要以编程方式关闭软键盘，请将 stage.focus 设为 null。要关闭软键盘并将焦点设置到另一个控件，请将 stage.focus 设为 null，然后将焦点设置在目标控件上。您也可以调用另一个控件的 requestSoftKeyboard() 方法来更改焦点，在另一个控件上打开软键盘。

可通过事件处理函数访问软键盘的某些属性。要访问软键盘的大小和位置，请使用 flash.display.Stage 类的 softKeyboardRect 属性，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_keyboard/views/SoftKeyboardEvents.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Soft Keyboard Events">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <s:TextInput prompt="Enter your text"
             softKeyboardActivate="ta1.text+=stage.softKeyboardRect + '\n'"
             softKeyboardDeactivate="ta1.text+=stage.softKeyboardRect + '\n'"
             softKeyboardActivating="ta1.text+=stage.softKeyboardRect + '\n'"/>
    <s:TextArea id="ta1" width="100%" height="100%" editable="false"/>

</s:View>
```

配置应用程序以支持软键盘

为支持软键盘，当键盘打开时，应用程序应当可以执行以下操作：

- 根据剩余可用屏幕空间来调整应用程序大小，以免键盘会与应用程序发生重叠。
- 滚动获得焦点的文本输入控件的父容器，以确保该控件可见。

配置系统以支持软键盘

在全屏模式下运行的应用程序不支持软键盘。因此，请确保 app.xml 文件中的 <fullScreen> 属性设置为默认值 false。

请确保应用程序的呈示模式设置为 CPU 模式。呈示模式由应用程序 app.xml 描述符文件中的 <renderMode> 属性控制。请确保 <renderMode> 属性设置为默认值 cpu，而不是 gpu。

注： 默认情况下， app.xml 文件中不包括 <renderMode> 属性。要更改其设置，请将其添加为 <initialWindow> 属性中的一个条目。如果 app.xml 文件中未包含此设置，则其默认值为 cpu。

打开软键盘时滚动父容器

Application 容器的 resizeForSoftKeyboard 属性决定着应用程序的大小调整行为。如果 resizeForSoftKeyboard 属性是默认值 false，则键盘可以显示在应用程序的顶部。如果值为 true，则会调整应用程序大小，以减去键盘大小。

要支持滚动，文本输入控件必须使用基于 TextField 的外观，而不是基于 StageText 的外观。可通过相应地将 TextInput 或 TextArea 控件的 skinClass 属性指向 TextInputSkin 或 TextAreaSkin 类来达到此目的。

要进行滚动，请将任何文本输入控件的父容器封装在 Scroller 组件中。当打开键盘的组件获得焦点时， Scroller 自动将该组件滚动到视图中。该组件也可以是 Scroller 组件的多个嵌套容器的子代。

父容器必须是 GroupBase 或 SkinnableContainer 类或其子类。获得焦点的组件必须实现 IVisualElement 接口，并且必须可以获得焦点。

通过将父容器封装在 Scroller 组件中，可以在键盘打开的情况下滚动该容器。例如，某个容器中包含多个文本输入控件。然后，您滚动到每个文本输入控件以输入数据。

当关闭键盘时，父容器可能小于可用屏幕空间。如果容器小于可用屏幕空间，Scroller 会将滚动位置恢复为 0，即容器的顶部。

下面的示例中显示的是带有多个 TextInput 控件和一个 Scroller 组件的 View 容器。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\views\SparkMobileKeyboardHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Compose Email">

    <s:Scroller width="100%" top="10" bottom="50">
        <s:VGroup paddingTop="3" paddingLeft="5" paddingRight="5" paddingBottom="3">
            <!-- Use TextField-based skins so that scrolling works. -->
            <s:TextInput prompt="To" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextInput prompt="CC" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextInput prompt="Subject" width="100%" skinClass="spark.skins.mobile.TextInputSkin"/>
            <s:TextArea height="400" width="100%" prompt="Compose Mail"
skinClass="spark.skins.mobile.TextAreaSkin"/>
        </s:VGroup>
    </s:Scroller>

    <s:HGroup width="100%" gap="20"
              bottom="5" horizontalAlign="left">
        <s:Button label="Send" height="40"/>
        <s:Button label="Cancel" height="40"/>
    </s:HGroup>

</s:View>
```

VGroup 容器是 TextInput 控件的父容器。Scroller 将 VGroup 封装起来，每个 TextInput 控件在获得焦点时，都将显示在键盘上方。

有关 Scroller 组件的更多信息，请参阅 Scrolling Spark containers。

打开软键盘时调整应用程序大小

如果 Application 容器的 resizeForSoftKeyboard 属性为 true，则应用程序会调整自身大小以在键盘打开时适合可用的屏幕区域。当关闭键盘时，应用程序恢复原有大小。

下例中显示的是一个应用程序的主应用程序文件，该应用程序通过将 resizeForSoftKeyboard 属性设置为 true 来允许调整应用程序大小。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- containers\mobile\SparkMobileKeyboard.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    firstView="views.SparkMobileKeyboardHomeView"
    resizeForSoftKeyboard="true">

</s:ViewNavigatorApplication>
```

要允许调整应用程序大小，请确保将应用程序 app.xml 描述符文件中 <softKeyboardBehavior> 属性设置为 none。<softKeyboardBehavior> 属性的默认值为 none。此默认值将 AIR 配置为移动整个 Stage，来使获得焦点的文本组件可见。

尽管软键盘事件的可靠性足以保证大多时候的自动调整行为可以达到预期效果，但仍应尽量避免将关键的 UI 元素放在软键盘事件失败时软键盘可能遮盖到的位置。例如，请避免在视图的下半部分放置 “ 确定 ”、 “ 登录 ” 或 “ 提交 ” 按钮。

当基于 StageText 的控件成为动画或参与动画时，运行时会临时将其替换为位图，以便文本与其他项目同步移动。如果该控件具有焦点，这将导致控件临时丢失焦点。在某些情况下，软键盘会在不隐藏软键盘的情况下隐藏或触发 softKeyboardDeactivate 事件。

尤其是当以编程方式将弹出窗口内的基于 StageText 的组件设为焦点，而由于软键盘导致窗口大小动态变化时，该问题特别明显。要避免此情况，请尽量在不会因软键盘而调整大小的弹出窗口内使用基于 StageText 的组件。

如果必须在可调整大小的弹出窗口中使用基于 StageText 的组件，请尝试首先显示软键盘并等待弹出窗口的动画完成，然后再以编程方式将基于 StageText 的组件设为焦点。作为替代方法，可以避免以编程方式在弹出窗口内设置焦点。

配置用于软键盘的弹出窗口

Callout 容器作为弹出窗口显示在移动设备应用程序的顶部。 Callout 容器可以包含一个或多个接受键盘输入的组件。有关 Callout 容器的更多信息，请参阅第 70 页的 " 使用 Callout 容器创建 callout"。

使用 SkinnablePopUpContainer 容器 （Callout 的父类）的属性配置弹出窗口与键盘的交互：

moveForSoftKeyboard 如果为默认值 true，弹出窗口将在键盘打开时移至键盘上方。

resizeForSoftKeyboard 如果为默认值 true，弹出窗口将在键盘打开时调整大小以适合键盘上方的可用空间。

如果 SkinnablePopUpContainer 的 moveForSoftKeyboard 或 resizeForSoftKeyboard 属性设置为 true，每当软键盘的可见性发生变化时，该容器都可能移动或发生大小调整。这两种自动行为都可能导致其他组件移动或发生大小变化。

避免这种情况的最简单方法是不将 resizeForSoftKeyboard 设置为 true。如果应用程序不会因软键盘而调整大小，则软键盘不会导致组件从用户手指下的位置移开。但是，这会导致软键盘遮盖一些应用程序的 UI。

如果应用程序需要自动调整大小，请置入交互式组件，这样，当在对其操作时，软键盘可见性的变化便不会导致其从用户手指的位置移开。要做到这一点，需要以下技术：

- 不要为按钮、复选框或其他较小目标组件设置下约束，也不要将这些组件放置在设置了百分比高度的其他组件下方。
- 尝试将布局设计为当显示或隐藏键盘时最底层组件的顶部保持固定。
- 对于内置功能不能实现隐藏软键盘的平台，请提供一个固定的 UI 元素来实现该操作。这可以是专用于隐藏软键盘的按钮，也可以仅仅是足够大的固定空白边缘 （可以轻敲该边缘以将焦点从文本组件移开）。
- 不要垂直排列可能会移动的组件。这么做可能导致在软键盘的可见性发生变化时手势定位到错误的目标。

在移动设备应用程序中嵌入字体

可以嵌入要在移动设备应用程序中使用的字体，但有一些限制条件。

由于 Label 控件使用 FTE （因此使用 CFF 字体），在移动设备应用程序中嵌入字体时应使用具有基于 TextField 外观的 TextArea 或 TextInput 控件。不能嵌入具有基于 StageText 外观的字体。通常情况下，在移动设备应用程序中应避免使用 FTE。

在您的 CSS 中，将 embedAsCFF 设置为 false，并应用基于 TextField 的外观，如下例所示：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/Main.mxml -->
<s:ViewNavigatorApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                     xmlns:s="library://ns.adobe.com/flex/spark"
                     firstView="views.EmbeddingFontsView">
    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";

        @font-face {
            src: url("../assets/MyriadWebPro.ttf");
            fontFamily: myFontFamily;
            embedAsCFF: false;
        }
        .customStyle {
            fontFamily: myFontFamily;
            fontSize: 24;
            skinClass: ClassReference("spark.skins.mobile.TextAreaSkin");
        }
    </fx:Style>
</s:ViewNavigatorApplication>
```

EmbeddingFontView 视图的 TextArea 控件应用类型选择器：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/EmbeddingFontsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Embedded Fonts">
    <s:layout>
        <s:VerticalLayout paddingTop="20" paddingLeft="20" paddingRight="20"/>
    </s:layout>
    <s:TextArea id="ta1"
                width="100%"
                styleName="customStyle"
                text="This is a TextArea control that uses an embedded font."/>
    <s:TextArea id="ta2"
                width="100%"
                text="This TextArea control does not use an embedded font."/>
</s:View>
```

如果使用类选择器（例如 s|TextArea）应用样式（或嵌入字体），请在主应用程序文件中定义类选择器。不能在移动设备应用程序的视图中定义类选择器。

有关更多信息，请参阅嵌入字体。

第 6 章：外观设计

移动设备外观设计的基础知识

比较桌面外观和移动设备外观

与桌面外观相比，移动设备外观更加轻量化。因此，它们具有许多不同之处；例如：

- 移动设备外观以 ActionScript 编写。在移动设备上，完全以 ActionScript 编写的外观可以提供最佳的性能。
- 移动设备外观扩展了 spark.skins.mobile.supportClasses.MobileSkin 类。该类扩展了 UIComponent，而 SparkSkin 类则扩展了 Skin 类。
- 移动设备外观使用已编译的 FXG 或简单的 ActionScript 图形作为图形资源，用以提高性能。而在桌面应用程序的外观中，通常使用 MXML 图形来组成大多数绘图。
- 移动设备外观无需声明任何外观状态。由于手机外观以 ActionScript 编写，因此必须通过过程来实现状态。
- 移动设备外观不支持状态转换。
- 移动设备外观通过手动布置。移动设备外观不扩展 Group，所以不支持 Spark 布局。因此，需要以 ActionScript 手动放置其子代。
- 移动设备外观并不支持所有样式。移动设备主题会忽略某些样式，具体取决于移动设备外观的性能或其它不同点。

除了与性能相关的差异外， Flash Builder 也以不同的方式使用某些移动设备外观文件。这个在库项目中使用的移动设备主题尤为突出。博客 Jeffry Houser 说明了如何修正此问题。

移动设备主机组件

移动设备外观通常会声明一个公共的 hostComponent 属性。该属性不是必需的，但建议使用。hostComponent 属性必须与使用外观的组件属于同一类型。例如， ActionBarSkin 将 hostComponent 声明为 ActionBar 类型：

```
public var hostComponent:ActionBar;
```

Flex 会在组件首次加载外观时设置 hostComponent 属性。

与桌面外观一样，您可以使用主机组件来访问外观所连组件的属性和方法。例如，您可以从外观类中访问主机组件的公共属性或者将事件侦听器添加到主机组件中。

移动设备样式

移动设备外观所支持的样式属性仅仅是桌面外观所支持样式属性中的一部分。这组样式由移动设备主题定义。

下表定义了使用移动设备主题时可供组件使用的样式属性：

样式属性	提供支持的组件	继承 / 非继承
accentColor	Button、 ActionBar、 ButtonBar	继承
backgroundAlpha	ActionBar	非继承
backgroundColor	Application	非继承
borderAlpha	List	非继承
borderColor	List	非继承

样式属性	提供支持的组件	继承 / 非继承
borderVisible	List	非继承
chromeColor	ActionBar、 Button、 ButtonBar、 CheckBox、 HSlider、 RadioButton	继承
color	所有带文本的组件	继承
contentBackgroundAlpha	TextArea、 TextInput	继承
contentBackgroundColor	TextArea、 TextInput	继承
focusAlpha	所有可获得焦点的组件	非继承
focusBlendMode	所有可获得焦点的组件	非继承
focusColor	所有可获得焦点的组件	继承
focusThickness	所有可获得焦点的组件	非继承
locale	所有组件	继承
paddingBottom	TextArea、 TextInput	非继承
paddingLeft	TextArea、 TextInput	非继承
paddingRight	TextArea、 TextInput	非继承
paddingTop	TextArea、 TextInput	非继承
selectionColor	ViewMenuItem	继承

基于文本的组件也支持标准文本样式，如 fontFamily、 fontSize 和 fontWeight。

要了解移动设备主题是否支持某个样式属性，请打开 ActionScript 语言参考中有关组件的描述。许多样式限制的原因在于基于文本的移动设备组件不使用 TLF （文本布局框架），而是，移动设备外观使用更轻型的组件来替换基于 TLF 的文本控件。有关更多信息，请参阅第 120 页的 “ 在移动设备应用程序中使用文本 ”。

移动设备主题不支持 rollOverColor、 cornerRadius 和 dropShadowVisible 样式属性。

Flex 工程师 Jason SJ 在他的博客中介绍了移动设备应用程序的样式和主题。

移动设备外观部件

就外观部件而言，移动设备外观必须遵循与桌面外观相同的外观约定。如果组件必须具备某个外观部件，则移动设备外观必须声明相应类型的公共属性。

异常

并非所有外观部件都是必须具备的。例如， Spark Button 有可选的 iconDisplay 和 labelDisplay 外观部件。因此，移动设备 ButtonSkin 类可以声明 BitmapImage 类型的 iconDisplay 属性。也可以声明 StyleableTextField 类型的 labelDisplay 属性。

labelDisplay 部件不设置 id 属性，因为该部件所使用的所有样式都会继承文本样式。此外， StyleableTextField 不是 UIComponent，因此不具有 id 属性。 iconDisplay 部件不支持样式，因此同样不设置 id 属性。

使用高级 **CSS** 设置样式

如果希望使用高级 CSS id 选择器设置外观部件的样式，则外观必须也设置外观部件的 id 属性。例如， ActionBar 的 titleDisplay 外观部件设置 id 属性，以便可以使用高级 CSS 设置其样式；例如：

```
@namespace s "library://ns.adobe.com/flex/spark";
s|ActionBar #titleDisplay {
    color:red;
}
```

移动设备主题

移动设备主题决定着移动设备应用程序所支持的样式。可供移动设备主题使用的样式仅仅是 Sprak 主题所用样式中的一部分（外加少量的其它样式）。有关移动设备主题所支持的样式的完整列表，请参阅第 141 页的“ 移动设备样式 ”。

移动设备应用程序的默认主题

移动设备应用程序的主题在 themes/Mobile/mobile.swc 文件中定义。该文件定义了移动设备应用程序的全局样式，以及各个移动设备组件的默认设置。此主题文件中的移动设备外观在 spark.skins.mobile.* 包中定义。此包中包括 MobileSkin 基类。

默认情况下， mobile.swc 主题文件包括在 Flash Builder 移动设备项目中，但该 SWC 文件不会显示在包资源管理器中。

在 Flash Builder 中创建新的移动设备项目时，将默认应用此主题。

更改主题

要更改主题，请使用 theme 编译器参数来指定新的主题；例如：

```
-theme+=myThemes/NewMobileTheme.swc
```

有关主题的更多信息，请参阅 About themes。

Flex 工程师 Jason SJ 在他的博客中介绍了如何在移动设备应用程序中创建和覆盖主题。

移动设备外观状态

MobileSkin 类可以重写 UIComponent 类的状态机制，但不采用桌面应用程序的视图状态实现方式。因此，移动设备外观仅声明由外观实现的主机组件外观状态。它们仅根据状态名称来通过过程更改状态。相反，桌面外观则必须声明所有状态，而无论这些状态是否使用。桌面外观还使用 mx.states.* 包中的类来更改状态。

大多数移动设备外观实现的状态数量都少于桌面外观。例如， spark.skins.mobile.ButtonSkin 类实现 up、 down 和 disabled 状态。 spark.skins.spark.ButtonSkin 可实现所有这些状态以及 over 状态。移动设备外观不会定义 over 状态的行为，因为触控设备通常不会使用该状态。

commitCurrentState() 方法

移动设备外观类在 commitCurrentState() 方法中定义其状态行为。可以通过在自定义外观类中编辑 commitCurrentState() 方法，在移动设备外观中添加行为来支持其它状态。

currentState 属性

外观的表现取决于 currentState 属性的值。例如，在移动设备 ButtonSkin 类中，currentState 属性的值决定着使用哪个 FXG 类作为边框类：

```
if (currentState == "down")
    return downBorderSkin;
else
    return upBorderSkin;
```

有关 currentState 属性的更多信息，请参阅 Create and apply view states。

移动设备图形

移动设备外观通常使用已编译的 FXG 作为图形资源。而在桌面应用程序的外观中，通常使用 MXML 图形来组成大多数绘图。

嵌入式位图图形

可以在类中使用嵌入式位图图形，这些图形通常能够正常工作。但是，这些位图有时无法在多种屏幕密度之间正常缩放。为了实现更好的缩放效果，应创建多个不同的资源，每个资源对应一种屏幕密度。

默认移动设备主题中的图形

在默认移动设备主题中，移动设备外观所使用的 FXG 图形都针对目标设备的 DPI 进行了优化。外观将根据根应用程序的 applicationDPI 属性值来加载图形。例如，在 DPI 为 320 的设备上加载 CheckBox 控件时，CheckBoxSkin 类为 upIconClass 属性使用 spark.skins.mobile320.assets.CheckBox_up.fxg 图形。而在 DPI 为 160 的设备上，则使用 spark.skins.mobile160.assets.CheckBox_up.fxg 图形。

以下桌面示例显示了不同 DPI 时 CheckBox 外观所使用的不同图形：

```
<?xml version="1.0"?>
<!-- mobile_skins/ShowCheckBoxSkins.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:skins160="spark.skins.mobile160.assets.*"
    xmlns:skins240="spark.skins.mobile240.assets.*"
    xmlns:skins320="spark.skins.mobile320.assets.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <!--
    NOTE: You must add the mobile theme directory to source path
    to compile this example.

    For example:
    mxmlc -source-path+=\frameworks\projects\mobiletheme\src\ ShowCheckBoxSkins.mxml
    -->
    <s:Label text="160 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins160:CheckBox_down/>
        <skins160:CheckBox_downSymbol/>
        <skins160:CheckBox_downSymbolSelected/>
        <skins160:CheckBox_up/>
        <skins160:CheckBox_upSymbol/>
        <skins160:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="240 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins240:CheckBox_down/>
        <skins240:CheckBox_downSymbol/>
        <skins240:CheckBox_downSymbolSelected/>
        <skins240:CheckBox_up/>
        <skins240:CheckBox_upSymbol/>
        <skins240:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="320 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins320:CheckBox_down/>
        <skins320:CheckBox_downSymbol/>
        <skins320:CheckBox_downSymbolSelected/>
        <skins320:CheckBox_up/>
        <skins320:CheckBox_upSymbol/>
        <skins320:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <s:Label text="down, downSymbol, downSymbolSelected, up, upSymbol, upSymbolSelected"/>
</s:Application>
```

有关移动设备应用程序中的分辨率和 DPI 的更多信息，请参阅第 108 页的 “ 在一个移动设备应用程序中支持多个屏幕大小和 DPI 值 ”。

在 ActionScript 外观中，也可以使用缓存为位图的矢量图形。这样做唯一的缺点是，无法使用需要重新绘制像素的过渡效果，例如 Alpha 过渡。有关更多信息，请参阅 www.adobe.com/cn/devnet/air/flex/articles/writing_multiscreen_air_apps.html。

为移动设备应用程序创建外观

自定义移动设备外观时，可以创建自定义的移动设备外观类。在某些情况下，也可以编辑移动设备外观类所使用的资源。

编辑移动设备外观类时，可以更改基于状态的交互，实现对新样式的支持，或者在外观中添加或删除子组件。通常可以从现有外观的源代码入手，将其保存为新类。

也可以编辑移动设备外观所使用的资源来更改外观的可视属性，如大小、颜色、渐变和背景。在这种情况下，也可以编辑外观所使用的 FXG 资源。移动设备外观所使用的源 *.fxg 文件位于 spark/skins/mobile/assets 目录下。

并非移动设备外观的所有可视属性都在 *.fxg 文件中定义。例如，Button 外观的背景色由 ButtonSkin 类中的 chromeColor 样式属性定义，而不是在 FXG 资源中定义。在这种情况下，需要编辑外观类来更改背景颜色。

创建移动设备外观类

创建自定义移动设备外观类时，最简单的方法是使用现有移动设备外观类作为基础。然后更改该类，并将其用作自定义外观。

要创建自定义外观类，请执行以下操作：

1 在项目中创建目录 （例如 customSkins）。此目录的名称是自定义外观的包名称。尽管不需要创建包，但建议您将自定义外观放在单独的包中。

2 在新目录中创建自定义外观类。可根据需要为新类命名，例如 CustomButtonSkin.as。

3 复制新类所基于的外观类的内容。例如，如果正在使用 ButtonSkin 作为基类，请将 spark.skins.mobile.ButtonSkin 文件的内容复制到新的自定义外观类中。

4 编辑新类。例如，至少对 CustomButtonSkin 类进行以下更改：

- 更改包的位置：

```
package customSkins
//was: package spark.skins.mobile
```

- 在类声明中更改类的名称。它是对新外观所基于的类的扩展 （而不是基本外观类）：

```
public class CustomButtonSkin extends ButtonSkin
// was: public class ButtonSkin extends ButtonSkinBase
```

- 更改构造函数中的类名称：

```
public function CustomButtonSkin()
//was: public function ButtonSkin()
```

5 更改自定义外观类。例如，添加对其它状态或新的子组件的支持。此外，外观类本身也定义了一些图形资源，因此可以更改某些资源。

为使外观类更易于理解，通常可以从自定义外观中删除任何不会被重写的方法。

以下自定义外观类扩展了 ButtonSkin，并使用自定义逻辑来替换 drawBackground() 方法。它使用径向渐变替换线性渐变来完成背景填充。

```
package customSkins {
    import mx.utils.ColorUtil;
    import spark.skins.mobile.ButtonSkin;
    import flash.display.GradientType;
    import spark.skins.mobile.supportClasses.MobileSkin;
    import flash.geom.Matrix;
    public class CustomButtonSkin extends ButtonSkin {

        public function CustomButtonSkin() {
            super();
        }
        private static var colorMatrix:Matrix = new Matrix();
        private static const CHROME_COLOR_ALPHAS:Array = [1, 1];
        private static const CHROME_COLOR_RATIOS:Array = [0, 127.5];

        override protected function drawBackground(unscaledWidth:Number, unscaledHeight:Number):void {
            super.drawBackground(unscaledWidth, unscaledHeight);

            var chromeColor:uint = getStyle("chromeColor");
            /*
            if (currentState == "down") {
                graphics.beginFill(chromeColor);
            } else {
            */
            var colors:Array = [];
            colorMatrix.createGradientBox(unscaledWidth, unscaledHeight, Math.PI / 2, 0, 0);
            colors[0] = ColorUtil.adjustBrightness2(chromeColor, 70);
            colors[1] = chromeColor;
            graphics.beginGradientFill(GradientType.RADIAL, colors, CHROME_COLOR_ALPHAS,
CHROME_COLOR_RATIOS, colorMatrix);
            // }
            graphics.drawRoundRect(layoutBorderSize, layoutBorderSize,
                unscaledWidth - (layoutBorderSize * 2),
                unscaledHeight - (layoutBorderSize * 2),
                layoutCornerEllipseSize, layoutCornerEllipseSize);
            graphics.endFill();
        }

    }
}
```

6 在应用程序中，通过第 151 页的“应用自定义移动设备外观”中介绍的一种方法应用自定义外观。下面的示例使用组件标签上的 skinClass 属性来应用 customSkins.CustomButtonSkin 外观。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/CustomButtonSkinView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
    <fx:Declarations>
        <!-- Place non-visual elements (e.g., services, value objects) here -->
    </fx:Declarations>

    <s:Button label="Click Me" skinClass="customSkins.CustomButtonSkin"/>

</s:View>
```

移动设备外观的生命周期方法

创建自定义外观类时，应熟悉以下 UIComponent 方法。这些继承的、受保护的方法将定义外观的子代和成员，并帮助外观与显示列表中的其它组件进行交互。

- createChildren() — 创建外观所需的任何子图形或文本对象。

- commitProperties() — 必要时将组件数据复制到外观中。
- measure() — 尽可能有效地测量外观，并将结果存储在外观的 measuredWidth 和 measuredHeight 属性中。
- updateDisplayList() — 设置图形和文本的位置和大小。执行任何所需的 ActionScript 绘制。此方法对外观调用 drawBackground() 和 layoutContents() 方法。

有关这些方法使用方法的更多信息，请参阅 Implementing the component。

常用的移动设备外观自定义方法

许多移动设备外观都实现以下方法：

- layoutContents() — 确定外观子代 （例如投影和标签）的位置。移动设备外观类不支持 HorizontalLayout 和 VerticalLayout 等 Spark 布局。可以在 layoutContents() 等方法中手动布置外观的子代。
- drawBackground() — 呈示外观的背景。通常情况下，其用途包括绘制 chromeColor、 backgroundColor 或 contentBackgroundColor 样式，具体取决于外观的形状。它还可以用于着色，例如使用 applyColorTransform() 方法。
- commitCurrentState() — 定义移动设备外观的状态行为。可以通过编辑此方法，来添加或删除支持的状态，或者更改现有状态的行为。当状态改变时调用此方法。大多数外观类都会重写此方法。有关更多信息，请参阅第 143 页的 “移动设备外观状态 ”。

创建自定义 FXG 资源

移动设备外观的大多数可视资源都使用 FXG 进行定义。FXG 是一种声明性语法，用于定义静态图形。可以使用图形工具 （如 Adobe Fireworks、Adobe Illustrator 或 Adobe Catalyst）导出 FXG 文档。然后可以在移动设备外观中使用该 FXG 文档。也可以在文本编辑器中创建 FXT 文档，但从头编写复杂的图形非常困难。

移动设备外观通常使用 FXG 文件来定义外观的状态。例如， CheckBoxSkin 类使用以下 FXG 文件来定义其方框和复选标记符号的外观：

- CheckBox_down.fxg
- CheckBox_downSymbol.fxg
- CheckBox_downSymbolSelected.fxg
- CheckBox_up.fxg
- CheckBox_upSymbol.fxg
- CheckBox_upSymbolSelected.fxg

如果在图形编辑器中打开这些文件，将显示如下内容：

复选框状态 （down、 downSymbol、 downSymbolSelected、 up、 upSymbol 和 upSymbolSelected）

适用于多种分辨率的 **FXG** 文件

大多数移动设备外观都有三组 FXG 图形文件，每一组对应一个默认目标分辨率。例如，所有六个 CheckBoxSkin 类的不同版本位于 spark/skins/mobile160、 spark/skins/mobile240 和 spark/skins/mobile320 目录中。

创建自定义外观时，可以执行以下操作之一：

- 使用一个默认外观作为基础 （分辨率通常为 160 DPI）。添加外观缩放逻辑，以通过设置 Application 对象的 applicationDPI 属性来根据运行应用程序的设备缩放自定义外观。
- 创建三种版本的自定义外观 （160、 240 和 320 DPI）以优化显示效果。

某些移动设备外观为其图像资源使用一组 FXG 文件，而不具有特定于 DPI 的图形。这些资源存储在 spark/skins/mobile/assets 目录下。例如，ViewMenuItem 外观和 TabbedViewNavigator 按钮栏外观不具有特定于 DPI 的版本，因此其所有 FXG 资源都存储在此目录下。

自定义 **FXG** 文件

可以打开现有的 FXG 文件并对其进行自定义，或者在图形编辑器（例如 Adobe Illustrator）中创建并导出 FXG 文件。编辑 FXG 文件后，将其应用到外观类。

要通过修改 FXG 文件来创建自定义外观，请执行以下操作：

1 创建自定义外观类并将其放在 customSkins 目录下，如第 145 页的 “创建移动设备外观类” 中所述。

2 在 customSkins 目录下创建一个子目录，例如 assets。创建子目录是可选步骤，但有助于组织 FXG 文件和外观类。

3 在 assets 目录下创建一个文件，并将现有 FXG 文件的内容复制到该文件中。例如，创建名为 CustomCheckBox_upSymbol.fxg 的文件。将 spark/skins/mobile160/assets/CheckBox_upSymbol.fxg 内容复制到新建的 CustomCheckBox_upSymbol.fxg 文件中。

4 更改新的 FXG 文件。例如，使用以渐变项填充的 “X” 号替换复选标记的绘制逻辑：

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- mobile_skins/customSkins/assets/CustomCheckBox_upSymbol.fxg -->
<Graphic xmlns="http://ns.adobe.com/fxg/2008" version="2.0"
    viewWidth="32" viewHeight="32">
    <!-- Main Outer Border -->
    <Rect x="1" y="1" height="30" width="30" radiusX="2" radiusY="2">
        <stroke>
            <SolidColorStroke weight="1" color="#282828"/>
        </stroke>
    </Rect>
    <!-- Replace check mark with an "x" -->
    <Group x="2" y="2">
        <Line xFrom="3" yFrom="3" xTo="25" yTo="25">
            <stroke>
                <LinearGradientStroke caps="none" weight="8" joints="miter" miterLimit="4">
                    <GradientEntry color="#FF0033"/>
                    <GradientEntry color="#0066FF"/>
                </LinearGradientStroke>
            </stroke>
        </Line>
        <Line xFrom="25" yFrom="3" xTo="3" yTo="25">
            <stroke>
            <stroke>
                <LinearGradientStroke caps="none" weight="8" joints="miter" miterLimit="4">
                    <GradientEntry color="#FF0033"/>
                    <GradientEntry color="#0066FF"/>
                </LinearGradientStroke>
            </stroke>
            </stroke>
        </Line>
    </Group>
</Graphic>
```

5 在自定义外观类中，导入新 FXG 类，并将其应用到某个属性。例如，在 CustomCheckBox 类中：

   1 导入新的 FXG 文件：

   ```
   //import spark.skins.mobile.assets.CheckBox_upSymbol;
   import customSkins.assets.CustomCheckBox_upSymbol;
   ```

   2 将新资源添加到自定义外观类中。例如，更改 upSymbolIconClass 属性的值以指向新的 FXG 资源：

   ```
   upSymbolIconClass = CustomCheckBox_upSymbol;
   ```

完整的自定义外观类如下所示：

```
// mobile_skins/customSkins/CustomCheckBoxSkin.as
package customSkins {
    import spark.skins.mobile.CheckBoxSkin;
    import customSkins.assets.CustomCheckBox_upSymbol;

    public class CustomCheckBoxSkin extends CheckBoxSkin {
        public function CustomCheckBoxSkin() {
            super();
            upSymbolIconClass = CustomCheckBox_upSymbol; // was CheckBox_upSymbol
        }
    }
}
```

有关处理和优化外观 FXG 资源的信息，请参阅 Optimizing FXG。

在应用程序中查看 FXG 文件

由于 FXG 文件以 XML 编写，因此很难直观看到最终产品的效果。可以编写一个 Flex 应用程序，通过将 FXG 文件添加为组件并封装在 Spark 容器中，来导入和呈示 FXG 文件。

要将 FXG 文件作为组件添加到应用程序中，请将源文件的位置添加到应用程序的源路径中。例如，要在基于 Web 的应用程序中显示移动设备 FXG 资源，请在源路径中添加移动设备主题。然后，编译器即可找到该 FXG 文件。

下面的桌面示例呈示 CheckBox 组件的各种 FXG 资源 （如果您在移动设备应用程序中使用该组件）。编译此示例时，将 frameworks\projects\mobiletheme\src\ 目录添加到编译器的 source-path 参数中。

```
<?xml version="1.0"?>
<!-- mobile_skins/ShowCheckBoxSkins.mxml -->
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:skins160="spark.skins.mobile160.assets.*"
    xmlns:skins240="spark.skins.mobile240.assets.*"
    xmlns:skins320="spark.skins.mobile320.assets.*">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <!--
    NOTE: You must add the mobile theme directory to source path
    to compile this example.

    For example:
    mxmlc -source-path+=\frameworks\projects\mobiletheme\src\ ShowCheckBoxSkins.mxml
    -->
    <s:Label text="160 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins160:CheckBox_down/>
        <skins160:CheckBox_downSymbol/>
        <skins160:CheckBox_downSymbolSelected/>
        <skins160:CheckBox_up/>
        <skins160:CheckBox_upSymbol/>
        <skins160:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="240 DPI" fontSize="24" fontWeight="bold"/>
```

```
    <s:HGroup>
        <skins240:CheckBox_down/>
        <skins240:CheckBox_downSymbol/>
        <skins240:CheckBox_downSymbolSelected/>
        <skins240:CheckBox_up/>
        <skins240:CheckBox_upSymbol/>
        <skins240:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <mx:Spacer height="30"/>
    <s:Label text="320 DPI" fontSize="24" fontWeight="bold"/>
    <s:HGroup>
        <skins320:CheckBox_down/>
        <skins320:CheckBox_downSymbol/>
        <skins320:CheckBox_downSymbolSelected/>
        <skins320:CheckBox_up/>
        <skins320:CheckBox_upSymbol/>
        <skins320:CheckBox_upSymbolSelected/>
    </s:HGroup>
    <s:Label text="down, downSymbol, downSymbolSelected, up, upSymbol, upSymbolSelected"/>
</s:Application>
```

在自定义移动设备外观中使用文本

要在移动设备外观中呈示文本，请使用 StyleableStageText 或 StyleableTextField 类。这些文本类已针对移动设备应用程序进行了优化。

StyleableStageText 允许 TextInput 和 TextArea 控件访问本机文本输入。它扩展了 UIComponent 类，实现了 IEditableText 和 ISoftKeyboardHintClient 接口。

当您不需要访问本机输入时， StyleableTextField 也可由 TextInput 和 TextArea 控件使用。它也可由非输入文本控件 （如 ActionBar 和 Button）使用。它扩展了 TextField 类，并可以实现 ISimpleStyleClient 和 IEditableText 接口。

有关在移动设备应用程序中使用文本控件的更多信息，请参阅第 120 页的 " 在移动设备应用程序中使用文本 "。

移动设备外观中的 **TLF**

出于性能方面的考虑，在移动设备外观中应尽量避免出现使用 TLF 的类。在某些情况下 （例如对于 Spark Label 组件），可以出现使用 FTE 的类。

移动设备外观中的 **htmlText**

不能在移动设备应用程序中使用 htmlText 属性。

针对文本的手势

触摸加拖动的手势始终会选择文本 （如果该文本可以选择或编辑）。如果文本位于 Scroller 中，则只有在文本组件之外做出该手势时， Scroller 才会滚动。只有在文本可以编辑或选择时，这些手势才会生效。

使文本可供编辑和选择

要使文本可以编辑或选择，请将 editable 和 selectable 属性设置为 true。

```
textDisplay.editable = true;
textDisplay.selectable = true;
```

双方向性

在 StyleableStageText 或 StyleableTextField 类中，文本不支持双方向性。

应用自定义移动设备外观

可以将自定义外观应用到移动设备组件，其方法与将自定义外观应用到桌面应用程序中的组件相同。

以 **ActionScript** 应用外观

```
// Call the setStyle() method:
myButton.setStyle("skinClass", "MyButtonSkin");
```

以 **MXML** 应用外观

```
<!-- Set the skinClass property: -->
<s:Button skinClass="MyButtonSkin"/>
```

以 **CSS** 应用外观

```
// Use type selectors for mobile skins, but only in the root document:
s|Button {
    skinClass: ClassReference("MyButtonSkin");
}
```

或者

```
// Use class selectors for mobile skins in any document:
.myStyleClass {
    skinClass: ClassReference("MyButtonSkin");
}
```

应用自定义移动设备外观的示例

下面的示例演示了全部三种将自定义移动设备外观应用到移动设备组件的方法：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_skins/views/ApplyingMobileSkinsView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="Home">
    <s:layout>
        <s:VerticalLayout/>
    </s:layout>

    <fx:Script>
        <![CDATA[
            import customSkins.CustomButtonSkin;
            private function changeSkin():void {
                b3.setStyle("skinClass", customSkins.CustomButtonSkin);
            }
        ]]>
    </fx:Script>

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        .customButtonStyle {
            skinClass: ClassReference("customSkins.CustomButtonSkin");
        }
    </fx:Style>

    <s:Button id="b1" label="Click Me" skinClass="customSkins.CustomButtonSkin"/>
    <s:Button id="b2" label="Click Me" styleName="customButtonStyle"/>
    <s:Button id="b3" label="Click Me" click="changeSkin()"/>

</s:View>
```

使用 CSS 类型选择器应用自定义外观时，应在根移动设备应用程序文件中设置选择器。不能在移动设备视图中设置类型选择器，自定义组件也是如此。仍可以使用类选择器在移动设备应用程序的任何视图或文档中以 ActionScript、MXML 或 CSS 设置样式。

第 7 章：测试和调试

管理启动配置

在您运行或调试移动设备应用程序时， Flash Builder 将使用启动配置。您可以指定是在桌面上启动应用程序，还是在连接到计算机的设备上启动应用程序。

要创建启动配置，请执行以下步骤：

1 选择 " 运行 ">" 运行配置 " 以打开 " 运行配置 " 对话框。

要打开 " 调试配置 " 对话框，请选择 " 运行 ">" 调试配置 "。请参阅第 154 页的 " 在设备上测试和调试移动设备应用程序 "。

也可以在 Flash Builder 工具栏上 " 运行 " 按钮或 " 调试 " 按钮的下拉列表中访问 " 运行配置 " 或 " 调试配置 "。

2 展开 " 移动设备应用程序 " 节点。在对话框工具栏中单击 " 新建启动配置 " 按钮。

3 在下拉列表中指定目标平台。

4 指定启动方法：

- 桌面上

  根据您已指定的设备配置，使用 AIR Debug Launcher (ADL) 在桌面上运行或调试应用程序。该启动方法不是在设备上运行应用程序的实际模拟。但是，该方法确实可以用于查看应用程序布局并与应用程序进行交互。请参阅第 154 页的 " 使用 ADL 预览应用程序 "。

  单击 " 配置 " 可编辑设备配置。请参阅第 153 页的 " 配置桌面预览的设备信息 "。

- 设备上

  在设备上安装并运行应用程序。

  对于 Google Android 平台，Flash Builder 将应用程序安装到设备上并启动应用程序。Flash Builder 访问连接到计算机 USB 端口的设备。有关更多信息，请参阅第 154 页的 " 在设备上测试和调试移动设备应用程序 "。

  Windows 需要使用 USB 驱动程序来将 Android 设备连接到计算机。有关更多信息，请参阅第 16 页的 " 安装 Android 设备的 USB 设备驱动程序 (Windows)"。

5 指定是否要在每次启动时清除应用程序数据 （如果适用）。

在桌面上测试和调试移动设备应用程序

进行初始测试或调试时，或者在没有移动设备的情况下，Flash Builder 允许您在桌面上使用 AIR Debug Launcher (ADL) 测试和调试应用程序。

在首次测试或调试移动设备应用程序前，应定义启动配置。应指定目标平台，并指定 " 桌面上 " 作为启动方法。请参阅第 153 页的 " 管理启动配置 "。

配置桌面预览的设备信息

设备配置的属性决定着应用程序在 ADL 和 Flash Builder 设计模式下的显示方式。

第 12 页的 " 设置设备配置 " 中列出了支持的配置。设备配置不影响应用程序在设备上的外观。

屏幕密度

您可以在开发桌面上预览应用程序，或者以 Flash Builder 设计模式查看应用程序的布局。 Flash Builder 使用的屏幕密度为 240 DPI。应用程序在预览期间的外观有时不同于在支持不同像素密度的设备上的外观。

使用 ADL 预览应用程序

在桌面上预览应用程序时， Flash Builder 使用 ADL 启动应用程序。 ADL 对 " 设备 " 菜单提供相对应的快捷方式来模拟设备上的按钮。

例如，要模拟设备上的后退按钮，请选择 " 设备 ">" 后退 "。选择 " 设备 ">" 逆时针旋转 " 或 " 设备 ">" 顺时针旋转 " 来模拟设备的旋转。如果未选择自动定向，则禁用旋转选项。

在列表中拖动可以模拟在设备上的滚动列表操作。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了关于使用 ADL 在桌面上预览移动设备应用程序的视频教程。

在设备上测试和调试移动设备应用程序

您可以使用 Flash Builder 在开发桌面或设备上测试或调试移动设备应用程序。

可以基于所定义的启动配置测试和调试应用程序。 Flash Builder 在运行应用程序和调试应用程序时共享相同的启动配置。使用 Flash Builder 在设备上调试应用程序时， Flash Builder 会在该设备上安装调试版本的应用程序。

注： 如果将发行版导出到设备，将安装应用程序的非调试版本。非调试版本不适用于调试。

有关更多信息，请参阅管理启动配置。

在 Google Android 设备上调试应用程序

在 Android 设备上进行调试需要使用 Android 2.2 或更高版本。

您可以在以下任一情形中进行调试：

通过 USB 调试 要通过 USB 连接调试应用程序，您需要经由 USB 端口将设备连接到主机。通过 USB 进行调试时， Flash Builder 始终会将应用程序打包，然后在调试开始之前，在设备上安装和启动软件包。请确保在整个调试会话期间，您的设备与主机的 USB 端口相连。

通过网络调试 通过网络调试应用程序时，设备和主机必须位于同一网络中。设备和主机可以通过 WiFi、以太网或蓝牙连接到网络。

通过网络调试时，可使用 Flash Builder 调试已安装在所连设备上的应用程序，而无需重新安装应用程序。只有在打包期间以及在设备上安装应用程序的过程中，才需要通过 USB 端口将设备连接到主机。调试期间，可以将设备从 USB 端口中拔出。但请确保在整个调试会话期间，设备和主机之间保持网络连接。

调试应用程序前的准备工作

开始通过 USB 或网络进行调试之前，请执行以下步骤：

1 (Windows) 确保安装了合适的 USB 驱动程序。

   在 Windows 上，安装 Android USB 驱动程序。有关更多信息，请参阅 Android SDK 内部版本随附的文档。有关更多信息，请参阅第 16 页的 " 安装 Android 设备的 USB 设备驱动程序 (Windows)"。

2 确保在设备上启用了 USB 调试。

   在设备设置中，转至 " 应用程序 ">" 开发 "，然后启用 USB 调试。

检查连接的设备

您在设备上运行或调试移动设备应用程序时， Flash Builder 会检查连接的设备。如果 Flash Builder 找到一台连接的联机设备，则 Flash Builder 将部署和启动应用程序。否则，针对以下情况， Flash Builder 将启动 " 选择设备 " 对话框：

- 找不到连接的设备
- 找到一台连接的脱机设备或者其操作系统版本不受支持
- 找到多台连接的设备

如果找到多台设备， " 选择设备 " 对话框将列出设备及其状态 （联机或脱机）。选择要启动的设备。

" 选择设备 " 对话框将列出操作系统版本和 AIR 版本。如果设备上未安装 AIR，则 Flash Builder 会自动进行安装。

配置网络调试

只要在通过网络调试应用程序时，才需要执行以下步骤。

通过网络调试应用程序前的准备工作

通过网络调试应用程序之前，请执行以下步骤：

1 在 Windows 上，打开端口 7935 （Flash Player 调试器端口）和端口 7 （echo/ping 端口）。

   有关详细说明，请参阅此 Microsoft TechNet 文章。

   在 Windows Vista 中，取消选择 "Windows 防火墙中的无线网络连接 ">" 更改设置 ">" 高级 "。

2 在您的设备中，配置 " 设置 ">" 无线和网络 " 中的无线设置。

选择主网络接口

您的主机可以同时连接到多个网络接口。但是只能选择一个主要的网络接口用于调试。可通过在 Android APK 包文件中添加主机地址来选择该接口。

1 在 Flash Builder 中，打开 " 首选参数 "。

2 选择 "Flash Builder">" 目标平台 "。

   对话框将列出主机上所有可用的网络接口。

3 选择要嵌入到 Android APK 包中的网络接口。

确保可以从设备访问选定的网络接口。如果设备建立连接后无法访问选定的网络接口， Flash Builder 会显示一个对话框，要求您输入主机的 IP 地址。

调试应用程序

1 通过 USB 端口或通过网络连接来连接设备。

2 选择 " 运行 ">" 调试配置 " 以配置要调试的启动配置。

   - 对于 " 启动方法 "，请选择 " 设备上 "。
   - 选择 " 通过 USB 进行调试 " 或 " 通过网络进行调试 "。

     首次通过网络调试应用程序时，可以通过 USB 将应用程序安装到设备上。为此，请选择 " 通过 USB 将应用程序安装到设备 "，然后通过 USB 端口将设备连接到主机。

     安装了应用程序后，如果在后续的调试会话中，不希望通过 USB 进行连接，请取消选择 " 通过 USB 将应用程序安装到设备 "。

   - （可选）每次启动时清除应用程序数据。

如果要对每次调试会话保持应用程序的状态，请选择此选项。仅当在您的应用程序中将 sessionCachingEnabled 设置为 True 时，此选项才适用。

3 选择 " 调试 " 以开始调试会话。

调试器启动并等待应用程序启动。调试器建立与设备的连接时，调试会话启动。

尝试通过网络在设备上进行调试时，应用程序有时会显示一个对话框，要求您输入 IP 地址。该对话框将指明调试器无法连接。确保设备已正确连接到网络，并且可通过该网络访问运行 Flash Builder 的计算机。

注： 在公司、酒店或其它来宾网络上，有时候设备无法连接到计算机，即使这两者都在同一网络中。

如果您通过网络进行调试，并且应用程序之前就已安装在设备上，则输入主机的 IP 地址即可开始调试。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了关于通过 USB 调试 Android 设备的应用程序的视频教程。

更多帮助主题

Debug and Package Apps for Devices （视频）

在 Apple iOS 设备上调试应用程序

要在 Apple iOS 设备上调试应用程序，请在 iOS 设备上手动部署和安装调试 iOS 包（IPA 文件）。Apple iOS 平台不支持自动部署。

重要说明：在 iOS 设备上调试应用程序之前，确保执行第 18 页的 " 构建、调试或部署 iOS 应用程序前的准备工作 " 中所述的步骤。

1 将 Apple iOS 设备连接到开发计算机。

2 在 iOS 设备上启动 iTunes。

注：您需要使用 iTunes 将应用程序安装到 iOS 设备上，并获得 iOS 设备的设备 ID。

3 在 Flash Builder 中，选择 " 运行 ">" 调试配置 "。

4 在 " 调试配置 " 对话框中，执行以下步骤：

a 选择要调试的应用程序。

b 选择 "Apple iOS" 作为目标平台。

c 选择 " 设备上 " 作为启动方法。

d 选择以下任一打包方法：

标准 使用此方法可以打包具有发行版质量的应用程序，该版本可以在 Apple iOS 设备上运行。使用此方法的应用程序性能与最终发行包相似，可以提交至 Apple App Store。

但是，使用此方法创建调试 iOS (IPA) 文件需要几分钟时间。

快速 使用此方法可快速创建 IPA 文件，然后在设备上运行并调试文件。该方法适用于测试应用程序。使用此方法的应用程序性能不具有发行版质量，不适于提交至 Apple App Store。

e 单击 " 配置 " 选择相应的代码签名证书、配置文件和包内容。

f 单击 " 配置网络调试 "，选择希望添加到调试 iOS 包中的网络接口。

注：您的主机可以同时连接到多个网络接口。但是只能选择一个主要的网络接口用于调试。

g 单击 " 调试 "。 Flash Builder 会显示一个对话框，要求您输入密码。输入 P12 证书密码。

Flash Builder 会生成调试 IPA 文件并将其置于 bin-debug 文件夹内。

5 在 iOS 设备上，执行以下步骤：

   1 （可选）在 iTunes 中，选择 " 文件 ">" 添加到库 "，然后浏览至您从 Apple 获得的移动设备配置概要文件（文件扩展名为 .mobileprovision）。

   2 在 iTunes 中，选择 " 文件 ">" 添加到库 "，然后浏览至您在第 4 步中生成的调试 IPA 文件。

   3 通过选择 " 文件 ">" 同步 " 将您的 iOS 设备与 iTunes 同步。

4 Flash Builder 会尝试连接到调试 IPA 文件中指定的主机地址。如果应用程序无法连接到主机地址， Flash Builder 会显示一个对话框，要求您输入主机的 IP 地址。

注：如果自生成上一个调试 IPA 包后您未更改代码或资源， Flash Builder 会跳过打包步骤而调试应用程序。也就是说，您可以在设备上启动安装的应用程序并单击 " 调试 " 以连接到 Flash Builder 调试器。这样，您便可以反复进行调试，而无需每次都将应用程序打包。

第 8 章：在设备上安装

在 Google Android 设备上安装应用程序

在项目的开发、测试和部署阶段，可以将应用程序直接安装在设备上。

您可以使用 Flash Builder 直接在 Android 设备上安装应用程序。在未安装 Adobe AIR 的设备上安装包时， Flash Builder 会自动安装 AIR。

1 将 Google Android 设备连接到开发计算机。

   Flash Builder 访问连接到计算机 USB 端口的设备。确保已配置了必需的 USB 设备驱动程序。请参阅第 15 页的 " 连接 Google Android 设备 "。

2 在 Flash Builder 中，选择 " 运行 ">" 运行配置 "。在 " 运行配置 " 对话框中，选择要部署的移动设备应用程序。

3 选择 " 设备上 " 作为启动配置方法。

4 （可选）指定是否要在每次启动时清除应用程序数据。

5 单击 " 应用 "。

Flash Builder 将在 Android 设备上安装和启动应用程序。如果将包安装在未安装 Adobe AIR 的设备上，Flash Builder 会自动安装 AIR。

Flex 方面的 Adobe 认证专家 Brent Arnold 创建了关于在 Android 设备上设置和运行应用程序的视频教程。

在 Apple iOS 设备上安装应用程序

在 iOS 设备上，需要手动安装应用程序 （IPA 文件），因为 Apple iOS 平台不支持自动部署。

重要说明：在 iOS 设备上安装应用程序之前，您需要具有 Apple iOS 开发证书（P12 格式）以及开发版本的配置概要文件。请确保遵循第 18 页的 " 构建、调试或部署 iOS 应用程序前的准备工作 " 中所述的步骤。

1 将 Apple iOS 设备连接到开发计算机。

2 在开发计算机上启动 iTunes。

   注：您需要使用 iTunes 将应用程序安装到 iOS 设备上，并获得 iOS 设备的唯一设备标识符 (UDID)。

3 在 Flash Builder 中，选择 " 运行 ">" 运行配置 "。

4 在 " 运行配置 " 对话框中，执行以下步骤：

   a 选择要安装的应用程序。

   b 选择 "Apple iOS" 作为目标平台。

   c 选择 " 设备上 " 作为启动方法。

   d 选择以下任一打包方法：

      标准 使用此方法可以打包具有发行版质量的应用程序，该版本可以在 Apple iOS 设备上运行。

      标准打包方法会在打包之前将应用程序的 SWF 文件的字节码转换为 ARM 指令。由于在打包之前增加了此转换步骤，因此，使用此方法创建应用程序 (IPA) 文件需要几分钟才能完成。与快速方法相比，标准方法花费的时间更长。但是，使用标准方法的应用程序性能具有发行版质量，可以提交至 Apple App Store。

      快速 使用此方法可快速创建 IPA 文件。

快速打包方法省略了字节码转换过程，仅将应用程序 SWF 文件和资源与预编译的 AIR 运行时捆绑在一起。与标准方法相比，快速打包方法速度更快。但是，使用快速方法的应用程序性能不具有发行版质量，因此不可以提交至 Apple App Store。

注：标准打包方法与快速打包方法之间不存在运行时或功能上的差别。

e 单击 " 配置 " 选择相应的代码签名证书、配置文件和包内容。

f 单击 " 运行 "。 Flash Builder 会显示一个对话框，要求您输入密码。输入 P12 证书密码。

Flash Builder 会生成 IPA 文件并将其置于 bin-debug 文件夹内。

5 在开发计算机上，执行以下步骤：

1 在 iTunes 中，选择 " 文件 ">" 添加到库 "，然后浏览至您从 Apple 获得的移动设备配置概要文件 （文件扩展名为 .mobileprovision）。

您也可以将移动设备配置概要文件拖放到 iTunes 中。

2 在 iTunes 中，选择 " 文件 ">" 添加到库 "，然后浏览至您在第 4 步中生成的 IPA 文件。

您也可以将 IPA 文件拖放到 iTunes 中。

3 通过选择 " 文件 ">" 同步 " 将您的 iOS 设备与 iTunes 同步。

应用程序将部署在 iOS 设备上，您可以进行启动。

第 9 章：打包和导出

将移动设备应用程序打包并导出到在线商店

使用 Flash Builder 的“导出发行版”功能可以打包并导出发行版的移动设备应用程序。发行版通常是要上传到在线商店（如 Android Market、Amazon Appstore 或 Apple App store）的应用程序的最终版本。

在导出应用程序时，可以选择在某个设备上安装应用程序。如果在导出期间设备已连接到计算机，Flash Builder 将在设备上安装应用程序。您也可以选择导出特定于平台的应用程序包，以供以后在设备上安装。产生的软件包可以进行部署和安装，其方法与本机应用程序相同。

有关将 Android 应用程序导出到 Android Market 或 Amazon App Store 的详细信息，请参阅第 160 页的“导出用于发行的 Android APK 包”。

有关将 iOS 应用程序导出到 Amazon App Store 的详细信息，请参阅第 161 页的“导出用于发行的 Apple iOS 包”。

导出具有运行时绑定的应用程序

当您使用“导出发行版”功能导出移动设备应用程序时，您可以选择在应用程序包内嵌入 AIR 运行时。

然后，即使是在尚未安装 AIR 的设备上，用户也可以运行应用程序。根据要向其导出包的平台，您可以使用运行时绑定或共享运行时。

导出用于发行的 Android APK 包

导出移动设备应用程序前，可以自定义 Android 权限。可以在应用程序描述符文件中手动自定义相关设置。这些设置在 bin-debug/app_name-app.xml 文件的 <android> 块中。有关更多信息，请参阅设置 AIR 应用程序属性。

如果导出应用程序以供以后安装在设备上，请使用设备操作系统提供商所提供的工具来安装应用程序包。

1 在 Flash Builder 中，选择“项目”>“导出发行版”。

2 选择要导出的项目和应用程序。

3 选择目标平台和项目导出位置。

4 导出特定于平台的应用程序包并为其进行签名。

   您可以为每个目标平台打包带有数字签名的应用程序，或打包为带有数字签名的 AIR 桌面应用程序。

   也可以将应用程序导出为中间 AIRI 文件，可以在以后为其进行签名。如果选择此选项，请稍后使用 AIR adt 命令行工具，将 AIRI 打包为 APK 文件。然后使用特定于平台的工具在设备上安装 APK 文件（例如，对于 Android SDK，请使用 adb）。有关使用命令行工具将应用程序打包的信息，请参阅第 162 页的“使用命令行进行创建、测试和部署”。

5 在“打包设置”页面上，可以选择数字证书、包内容和任何本机扩展。

   部署 如果还想在设备上安装应用程序，请单击“部署”页并选择“在所有已连接的设备中安装并启动应用程序”。请确保已将一个或多个设备连接到计算机的 USB 端口。

   - 导出具有运行时绑定的应用程序

     如果您要在导出应用程序包时将 AIR 运行时嵌入 APK 文件内，请选择该选项。然后，即使是在尚未安装 AIR 的设备上，用户也可以运行应用程序。

   - 使用共享运行时导出应用程序

如果不希望在导出应用程序包时将 AIR 运行时嵌入 APK 文件内，请选择该选项。如果用户设备上尚未安装 AIR，您可以选择或指定为应用程序包下载 Adobe AIR 的 URL。

默认 URL 指向 Android Market。但是，您可以覆盖默认 URL，并选择指向 Amazon Appstore 上某位置的 URL，或者输入您自己的 URL。

数字签名 单击 " 数字签名 " 选项卡可以创建或浏览代表应用程序发布者身份的数字证书。也可以为所选证书指定密码。

如果创建证书，则该证书为自签名证书。可以从证书提供商处获得商业签名证书。请参阅对 AIR 应用程序进行数字签名。

包内容 （可选）单击 " 包内容 " 选项卡，以指定包中所包含的文件。

本机扩展 （可选）选择您要在应用程序包中包含的本机扩展。

有关本机扩展的更多信息，请参阅第 11 页的 " 使用本机扩展 "。

6 单击 " 完成 "。

Flash Builder 在第一个面板 （默认为项目的最顶级）所指定的目录中创建 ***ApplicationName***.apk。如果在导出期间已将设备连接到计算机， Flash Builder 将在设备上安装应用程序。

导出用于发行的 Apple iOS 包

您可以创建和导出用于临时分发或用于提交到 Apple App Store 的 iOS 包。

重要说明：导出 iOS 包之前，确保您已从 Apple 获得必需的证书和分发配置概要文件。为此，请执行第 18 页的 " 构建、调试或部署 iOS 应用程序前的准备工作 " 中所述的步骤。

1 在 Flash Builder 中，选择 " 项目 ">" 导出发行版 "。

2 选择 Apple iOS 作为目标平台，以导出和签名 IPA 包。

单击 " 下一步 "。

3 选择从 Apple 获得的 P12 证书和分发配置概要文件。

4 在 " 打包设置 " 页面上，可以选择配置证书、数字证书、包内容和任何本机扩展。

部署 导出 iOS 包时，默认情况下 AIR 运行时嵌入在 IPA 文件中。

数字签名 选择从 Apple 获得的 P12 证书和分发配置概要文件。

可以选择以下包类型之一：

- 限制分发的临时分发，适用于应用程序的限制分发
- **Apple App Store** 的最终发行包，将应用程序提交至 Apple App Store

包内容 （可选）单击 " 包内容 " 选项卡，以指定包中所包含的文件。

本机扩展 （可选）选择您要在应用程序包中包含的本机扩展。

如果本机扩展使用了 iOS5 SDK 功能，请选择 iOS SDK 的位置。有关更多信息，请参阅第 12 页的 " 支持 iOS5 本机扩展 "。

5 单击 " 完成 "。

Flash Builder 将验证软件包设置的配置，然后编译应用程序。打包完成后，您可以将 IPA 文件安装在连接的 Apple iOS 设备上或提交至 Apple App Store。

要使用 AIR Developer Tool (ADT) 打包 IPA 文件，请参阅 " 构建 AIR 应用程序 " 中的 iOS 包。

使用命令行进行创建、测试和部署

不使用 Flash Builder 也可以创建手机应用程序。可以改用 mxmlc、 adl 和 adt 命令行工具。

以下是使用命令行工具开发移动设备应用程序并将其安装到设备中的常规过程。后面将更详细地描述每个步骤：

1 使用 mxmlc 工具编译应用程序。

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

该步骤需要您传递设置为 “airmobile” 的 configname 参数。

2 使用 adl 工具在 AIR Debug Launcher (ADL) 中测试应用程序。

```
adl MyMobileApp-app.xml -profile mobileDevice
```

该步骤需要您创建应用程序描述符文件并将其作为参数传递至 adl 工具。还需要指定 mobileDevice 概要文件。

3 使用 adt 工具将应用程序打包。

```
adt -package -target apk SIGN_OPTIONS MyMobileApp.apk MyMobileApp-app.xml MyMobileApp.swf
```

该步骤需要您首先创建证书。

4 在移动设备上安装应用程序。要在 Android 设备上安装应用程序，请使用 adb 工具。

```
adb install -r MyMobileApp.apk
```

该步骤需要首先将移动设备通过 USB 连接至计算机。

5 将移动设备应用程序部署到联机存储。

使用 mxmlc 编译手机应用程序

您可以使用 mxmlc 命令行编译器编译手机应用程序。要使用 mxmlc，请将值 airmobile 传递给 configname 参数；例如：

```
mxmlc +configname=airmobile MyMobileApp.mxml
```

通过传递 +configname=airmobile，将指示编译器使用 airmobile-config.xml 文件。该文件在 sdk/frameworks 目录中。该文件执行以下任务：

- 应用 mobile.swc 主题。
- 更改以下库路径：
  - 从库路径中删除 libs/air。手机应用程序不支持 Window 和 WindowedApplication 类。
  - 从库路径中删除 libs/mx。手机应用程序不支持 MX 组件 （不是图表）。
  - 将 libs/mobile 添加至库路径。
- 删除 ns.adobe.com/flex/mx 和 www.adobe.com/2006/mxml 命名空间。手机应用程序不支持 MX 组件 （不是图表）。
- 禁用辅助功能。
- 删除 RSL 条目；手机应用程序不支持 RSL。

mxmlc 编译器生成 SWF 文件。

使用 adl 测试手机应用程序

可以使用 AIR Debug Launcher (ADL) 来测试手机应用程序。通过 ADL，无需先将应用程序打包并安装到设备上，即可运行和测试应用程序。

使用 **adl** 工具进行调试

ADL 会在标准输出中输出 trace 语句和运行时错误，但不支持断点或其它调试功能。对于复杂的调试问题，可以使用 Flash Builder 等集成开发环境。

启动 **adl** 工具

要从命令行启动 adl 工具，请传递手机应用程序的应用程序描述符文件并将 profile 参数设置为 mobileDevice，如下例所示：

```
adl MyMobileApp-app.xml -profile mobileDevice
```

mobileDevice 概要文件定义移动设备上所安装应用程序的一组功能。有关 mobileDevice 概要文件的特定信息，请参阅不同概要文件的功能。

创建应用程序描述符

如果未使用 Flash Builder 来编译应用程序，请手动创建应用程序描述符文件。您可以使用 /sdk/samples/descriptor-sample.xml 文件作为基础文件。通常，至少需要进行以下更改：

- 将 <initialWindow><content> 元素指向您的手机应用程序 SWF 文件的名称：

```
<initialWindow>
    <content>MyMobileApp.swf</content>
    ...
</initialWindow>
```

- 更改应用程序的标题，因为它将显示在移动设备上该应用程序的图标下。要更改标题，请编辑 <name><text> 元素：

```
<name>
    <text xml:lang="en">MyMobileApp by Nick Danger</text>
</name>
```

- 将 <android> 块添加到应用程序的 Android 特定的权限中。您通常可以使用以下权限，具体取决于设备所使用的服务：

```
<application>
    ...
    <android>
        <manifestAdditions>
            <![CDATA[<manifest>
                <uses-permission android:name="android.permission.INTERNET"/>
            </manifest>]]>
        </manifestAdditions>
    </android>
</application>
```

您也可以使用描述符文件设置应用程序的高度和宽度、图标文件的位置、版本控制信息以及有关安装位置的其它详细信息。

有关创建和编辑应用程序描述符文件的更多信息，请参阅 AIR 应用程序描述符文件。

使用 adt 将手机应用程序打包

可以使用 AIR Developer Tool (ADT) 通过命令行将手机应用程序打包。 adt 工具可以创建能部署到手机 Android 设备的 APK 文件。

创建证书

在创建 APK 文件前，请创建一个证书。出于开发目的，您可以使用自签证书。您可以使用 adt 工具创建自签证书，如以下示例所述：

```
adt -certificate -cn SelfSign -ou QE -o "Example" -c US 2048-RSA newcert.p12 password
```

adt 工具在当前目录中创建 newcert.p12 文件。当您打包您的应用程序时，您将该证书传递至 adt。请不要为生产应用程序使用自签名证书。这些证书仅为用户提供有限担保。有关使用权威证书颁发机构所颁发的证书为 AIR 安装文件进行签名的信息，请参阅对 AIR 应用程序进行签名。

创建包文件

要为 Android 创建 APK 文件，请将应用程序的相关详细信息（包括证书）传递给 adt，如下例所示：

```
adt -package -target apk -storetype pkcs12 -keystore newcert.p12 -keypass password MyMobileApp.apk
MyMobileApp-app.xml MyMobileApp.swf
```

adt 工具将输出 appname.apk 文件。

针对 **iOS** 打包

要针对 iOS 将手机应用程序打包，必须从 Apple 获取开发人员证书和配置文件。这需要您加入 Apple 的开发者计划。有关更多信息，请参阅第 18 页的 " 构建、调试或部署 iOS 应用程序前的准备工作 "。

Flex evangelist Piotr Walczyszy 解释了如何使用 ADT 和 Ant 将应用程序打包（适用于 iOS 设备）。

Blogger Valentin Simonov 提供了有关如何在 iOS 上发布应用程序的其它信息。

使用 abd 在设备上安装移动设备应用程序

可以使用 Android Debug Bridge (adb) 在运行 Android 的移动设备上安装应用程序（APK 文件）。adb 工具是 Android SDK 的一部分。

将设备连接到计算机

在运行 abd 以便在移动设备上安装 APK 文件之前，需要将设备连接到计算机。在 Windows 和 Linux 系统中，要连接设备，需要使用 USB 驱动程序。

有关为设备安装 USB 驱动程序的信息，请参阅使用硬件设备。

在连接的设备上安装应用程序

将设备连接到计算机后，可以将应用程序安装到设备中。要使用 adb 工具安装应用程序，请使用 install 选项并传递 APK 文件的名称，如下例所示：

```
adb install -r MyMobileApp.apk
```

如果您之前已安装应用程序，则使用 -r 选项将其覆盖。否则，每次要将更新版本的应用程序安装到移动设备中时，都必须先卸载该应用程序。

更多帮助主题

Android Debug Bridge

将应用程序部署到联机存储

您可以将应用程序部署到 Android Market、 Amazon Appstor 或 Apple App Store 等联机应用程序存储。

Lee Brimlow 介绍了如何将 Android 应用程序的新 AIR 部署到 Android Market。

Christian Cantrell 解释了如何将应用程序部署到 Amazon Appstore for Android。