

闲云无衣 <http://www.blogjava.net/xvridan>

Flex 快速入门

无衣

摘自 <http://www.adobe.com/cn/>
(版权归原作者所有)

Flex 快速入门: 使用 MXML 和 ActionScript 进行编码

Adobe® 将 Flex 实施为 ActionScript 类库。该类库包含组件 (容器和控件)、管理器类、数据服务类和所有其他功能的类。您通过将 MXML 和 ActionScript 语言与该类库一起使用来开发应用程序。

MXML

MXML 是用于为 Adobe® Flex™ 应用程序进行用户界面组件布局的 XML 语言。您还使用 MXML 来显式定义应用程序的非可视方面, 例如访问服务器端数据源和用户界面组件与数据源之间的数据绑定。

例如, 您通过使用下面的 MXML 语句, 使用 `<mx:Button>` 标签来创建 Button 控件的实例:

```
<mx:Button id="myButton" label="I'm a button!"/>
```

您设置 `id` 属性以赋予 Button 实例一个唯一的名称, 以后可以使用该名称引用到它。 `label` 属性设置在 Button 实例上显示的标签的文本。

下面的示例显示创建显示 Button 控件的 Flex 应用程序所需的完整代码:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    horizontalAlign="center" verticalAlign="center"
>

    <mx:Button id="myButton" label="I'm a button!" />
</mx:Application>
```

在编写 Flex 应用程序之后, 您必须使用 Flex 编译器来编译它。 Flex 编译器是称为 `mxmclc` 的一个很小的可执行文件, 处于 Flex 2 安装文件夹下的 Flex SDK 2.0\bin 文件夹中。

提示: 确保 Flex 2 installation folder\Flex SDK 2.0\bin 文件夹处于您的系统的路径中。 让 Flex 编译器处于您的路径中, 使您不管当前处于哪个文件夹中, 都可以从命令行调用它。

说明

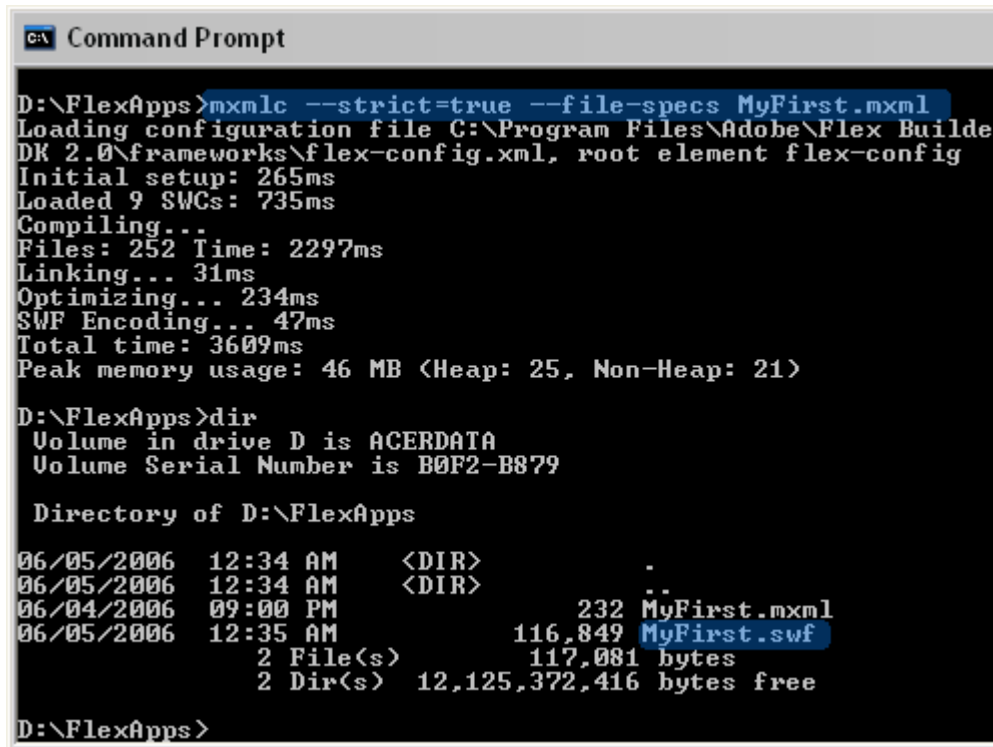
1. 在您喜爱的文本编辑器 (如, 记事本) 中创建一个新文件并将它另存为 `MyFirst.mxml`。
2. 从前面的示例中将代码输入到 `MyFirst.mxml` 中并保存您的文件。
3. 通过选择 “开始” > “所有程序” > “附件” > “命令提示符”, 打开命令窗口。
4. 将您的当前目录更改为包含您在步骤 1 中保存的 Flex 应用程序的文件夹。
5. 键入下面的命令来调用 Flex 编译器:

```
mxmclc --strict=true --file-specs MyFirst.mxml
```

以双短划线开头的命令字符串中的项目被称为编译器选项, 它们被用于定义 Flex 编译

器的行为。在前面的示例中, 您将 `--strict` 选项设置为 `true` 以强制编译器进入 `Strict` 模式。在 `Strict` 模式下, 编译器对您的代码具有较高的期望。例如, 它期望您以静态方式键入变量。您使用 `--file-specs` 选项来指定被编译的 `MXML` 文件。

6. 在 Windows 资源管理器中双击 `SWF` 文件或在命令行中输入其名称, 在独立的 Adobe Flash Player 9 中打开它。



```
C:\> Command Prompt

D:\FlexApps>mxmmlc --strict=true --file-specs MyFirst.mxml
Loading configuration file C:\Program Files\Adobe\Flex Builde
DK 2.0\frameworks\flex-config.xml, root element flex-config
Initial setup: 265ms
Loaded 9 SWCs: 735ms
Compiling...
Files: 252 Time: 2297ms
Linking... 31ms
Optimizing... 234ms
SWF Encoding... 47ms
Total time: 3609ms
Peak memory usage: 46 MB <Heap: 25, Non-Heap: 21>

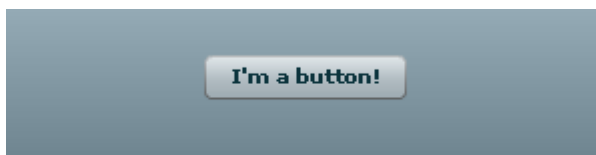
D:\FlexApps>dir
Volume in drive D is ACERDATA
Volume Serial Number is B0F2-B879

Directory of D:\FlexApps

06/05/2006  12:34 AM    <DIR>          .
06/05/2006  12:34 AM    <DIR>          ..
06/04/2006  09:00 PM             232 MyFirst.mxml
06/05/2006  12:35 AM        116,849 MyFirst.swf
               2 File(s)             117,081 bytes
               2 Dir(s)  12,125,372,416 bytes free

D:\FlexApps>
```

此示例会产生下列 `SWF` 文件:



提示: 您还可以使用 Adobe Flex Builder 2 创建和编译 Flex 应用程序, Adobe Flex Builder 2 是包含可视设计视图的用于 Flex 开发的集成开发环境 (IDE)。有关 Flex Builder 2 的详细信息, 请参阅使用 Flex Builder 2。

ActionScript

`MXML` 标签与 `ActionScript` 类或类的属性相对应。当您编译 Flex 应用程序时, Flex 会解析 `MXML` 标签并生成相应的 `ActionScript` 类。接着它将这些 `ActionScript` 类编译成存储在 `SWF` 文件中的 `SWF` 字节码。

提示: 若要查看 Flex 生成的中间 `ActionScript` 文件, 请将 `--keep-generated-actionscript` 选项添加到 `mxmmlc` 命令中。

继续上面的例子, Flex 提供定义 Flex Button 控件的 `ActionScript Button` 类。

注意: 在前面的示例中, `<mx:Button>` 标签中的 `mx` 前缀是一个名称空间。它是通过使用 `Application` 标签中的唯一 URL 声明的。 `mx` 前缀将 `mx` 名称空间中的每个组件映射到其完全合格的类名称。这就是 Flex 编译器可以找到与 `mx` 名称空间中的 MXML 标签相对应的 `ActionScript` 类的方式。

下面的示例阐述如何通过使用 `ActionScript` 创建 `Button` 控件。该结果与该 MXML 版本是相同的。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/GettingStartedActionScript/index.html"

    creationComplete="creationCompleteHandler();"
    width="300" height="80"
>

    <mx:Script>

        <![CDATA[
            import mx.controls.Button;
            import mx.events.FlexEvent;

            private var myButton:Button;

            private function creationCompleteHandler():void

            {
                // Create a Button instance and set its label
                myButton = new Button();
                myButton.label = "I'm a button!";

                // Get notified once button component has been created and
processed for layout

                myButton.addEventListener (FlexEvent.CREATION_COMPLETE,
buttonCreationCompleteHandler);

                // Add the Button instance to the DisplayList
                addChild (myButton);
            }

            private function buttonCreationCompleteHandler ( evt:FlexEvent ):
void

            {
                // Center the button
                myButton.x = parent.width/2 - myButton.width/2;
                myButton.y = parent.height/2 - myButton.height/2;
            }

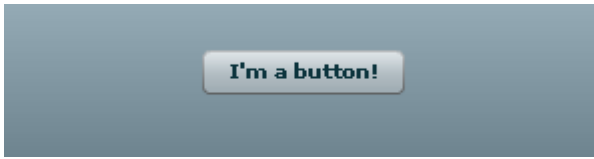
        ]]>
    </mx:Script>
</mx:Application>
```

通过 `ActionScript` 创建 Flex 组件时, 必须导入组件的类。您还必须通过使用 `addChild()` 方法使组件可见, 将组件添加到应用程序的 `DisplayList` 中。通过将此示例的长度和复杂性与其等价的 MXML 版本相比较, 您可以看到 MXML 的简单的基于标签的声明性语法是如何

闲云无衣 <http://www.blogjava.net/xvridan>

使您免于编写许多 `ActionScript` 代码行来进行组件布局的。

此示例会产生下列 `SWF` 文件:



注意: 此示例阐述线上 `ActionScript` 与 `Script` 标签的使用, 这是在 `Flex` 应用程序中包含 `ActionScript` 的一个可能的方法。 其他方法有: 将脚本块分隔到外部 `ActionScript` 文件中, 或使用外部 `ActionScript` 类。

Flex 快速入门: 创建您的第一个应用程序

MXML 文件是普通的 XML 文件, 所以可以选择多种开发环境。简单的文件编辑器、专用的 XML 编辑器或者支持文本编辑的集成开发环境 (IDE) 均可用于 MXML 代码编写。Flex 提供专用的 IDE, 称为 Adobe® Flex™ Builder™ 2, 您可以使用它来开发应用程序。

MXML 文件的第一行是 XML 声明。此行必须成为每个 MXML 文件的第一行。

下一行是 `<mx:Application>` 标签, 它定义始终是 Flex 应用程序的根标签的 `Application` 容器。

`<mx:Panel>` 标签定义包含一个标题栏、一个标题、一条状态消息、一个边框和其子级的一个内容区域的 `Panel` 容器。其 `title` 属性被设置为 "My Application"。

`<mx:Label>` 标签代表一个 `Label` 控件, 一个用于显示文本的非常简单的用户界面组件。其 `text` 属性被设置为 "Hello, World!"。

`<mx:Label>` 标签的 `fontWeight` 和 `fontSize` 属性改变使用的字体的样式。还可以使用 CSS 来设置组件的样式。有关详细信息, 请参阅 Flex 2 开发人员指南*中的“使用层叠样式表 (CSS)”。

注意: 可以通过使用命令行编译器 `mxmmlc` 或 `Flex Builder 2` 编译 Flex 应用程序。有关使用 `mxmmlc` 编译应用程序的说明, 请参阅使用 MXML 和 `ActionScript` 进行编码教程。

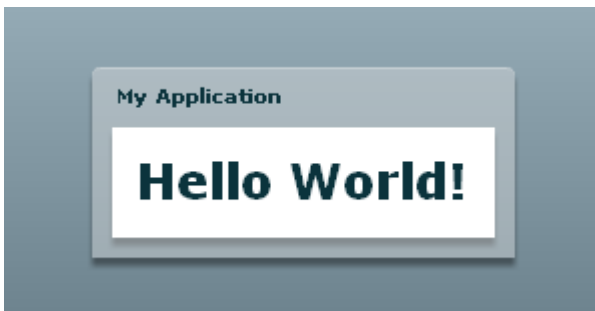
示例

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/HelloWorld/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="300" height="160"
>
    <mx:Panel
        paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10"
        title="My Application"
    >

        <mx:Label text="Hello World!" fontWeight="bold" fontSize="24"/>
    </mx:Panel>
</mx:Application>
```

结果



Flex 快速入门: 入门处理事件

Adobe® Flex™ 应用程序是事件驱动的。事件让程序员知道用户何时与界面组件交互, 以及在组件的外观或生命周期中何时发生重要的变化, 如组件的创建或破坏或调整其大小。

当组件的实例发出某个事件时, 会通知您注册为该事件的监听器的对象。您在 `ActionScript` 中定义事件监听器 (也称为事件处理程序) 来处理事件。您在组件的 `MXML` 声明中或者在 `ActionScript` 中注册事件的事件监听器。

接收事件通知有三种方式:

- 在 `MXML` 中注册事件处理程序
- 在 `MXML` 定义中创建线上事件处理程序
- 通过 `ActionScript` 注册事件监听器

在 `MXML` 中注册事件处理程序

获得事件通知的第一个和使用最广泛的方法是在 `MXML` 中定义事件发生时调用的事件处理程序。

在此示例中, 您为 `Button` 控件的 `click` 事件定义一个事件处理程序。用户单击 `Button` 控件时, 事件处理程序会将 `Label` 控件的 `text` 属性设置为 “Hello, World!”。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="300" height="200"
    horizontalAlign="center" verticalAlign="middle"
    viewSourceURL="src/HandlingEventsEventHandler/index.html"
>
    <mx:Script>
        <![CDATA[
            import flash.events.MouseEvent;

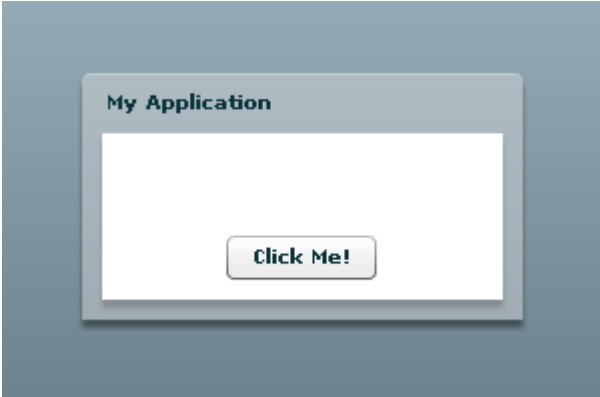
            private function clickHandler ( event:MouseEvent ):void
            {
                myLabel.text = "Hello, World!";
            }
        ]]>
    </mx:Script>

    <mx:Panel
        title="My Application" horizontalAlign="center"
        paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10"
    >

        <mx:Label id="myLabel" width="180" fontWeight="bold" fontSize="24"/>
        <mx:Button id="myButton" label="Click Me!" click="clickHandler(event);"
/>
```

```
</mx:Panel>
</mx:Application>
```

结果



在 MXML 定义中创建线上事件处理程序

有时, 对事件作出响应最容易的方法是在组件的 MXML 定义中完全地定义事件处理程序。这也称为使用线上事件处理程序。

在下面的示例中, 您设置 `<mx:Button>` 标签的 `click` 属性, 这样它会直接设置 `Label` 控件的 `text` 属性, 而不会调用事件处理程序方法。

提示: 使用线上事件处理程序可能很快并产生很少代码, 但它们也会导致代码很难阅读、维护和缩放。不在一个线上事件处理程序中包含多个 `ActionScript` 语句是一个很好的经验。如果您必须包含更加复杂的逻辑, 请将它置于 `ActionScript` 帮助器方法中或置于 `ActionScript` 事件处理程序中。

示例

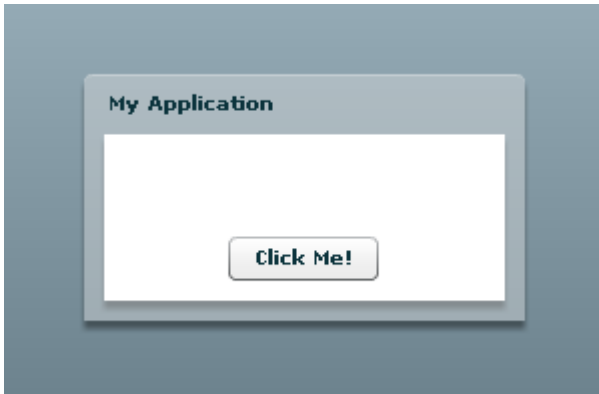
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/HandlingEventsInlineMethod/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="300" height="200"

>
    <mx:Panel
        title="My Application" horizontalAlign="center"
        paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10"
    >

        <mx:Label id="myLabel" width="180" fontWeight="bold" fontSize="24"/>
        <mx:Button id="myButton" label="Click Me!" click="myLabel.text = 'Hello,
World!'" />

    </mx:Panel>
</mx:Application>
```


结果



通过 ActionScript 注册事件监听器

您还可以通过使用 ActionScript, 通过注册事件处理程序来对事件作出响应。

在此示例中, 您通过使用 ActionScript 中的 `addEventListener()` 方法来注册事件处理程序。您将 `addEventListener()` 方法置于 Application 容器的 `creationComplete` 事件的事件处理程序中。

提示: Application 表单的 `creationComplete` 事件在 Application 表单及其子级被初始化之后, 在启动应用程序时发生。`creationComplete` 事件的事件处理程序为您提供一个运行 ActionScript 代码以注册事件监听器的方便位置。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/HandlingEventsActionScript/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="300" height="200"
    creationComplete="creationCompleteHandler(event);"
>
    <mx:Script>
        <![CDATA[
            import flash.events.MouseEvent;
            import mx.events.FlexEvent;

            private function creationCompleteHandler(event:FlexEvent):void

            {
                // Listen for the click event on the Button control
                myButton.addEventListener(MouseEvent.CLICK, clickHandler);
            }

            private function clickHandler ( event:Event ):void

            {
                myLabel.text = "Hello, World!";
            }
        ]]>
```

[闲云无衣 http://www.blogjava.net/xvridan](http://www.blogjava.net/xvridan)

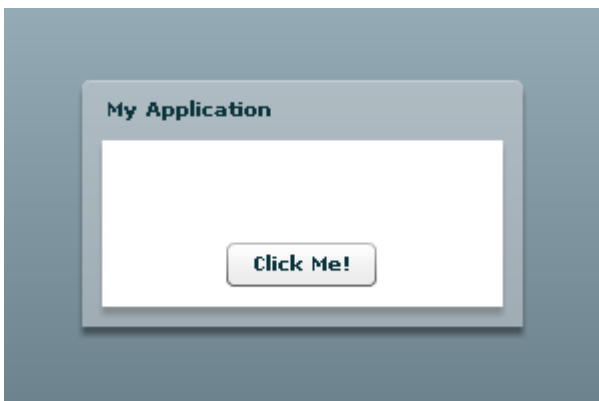
```
</mx:Script>

<mx:Panel
    title="My Application" horizontalAlign="center"
    paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10"
>

    <mx:Label id="myLabel" width="180" fontWeight="bold" fontSize="24"/>
    <mx:Button id="myButton" label="Click Me!" />

</mx:Panel>
</mx:Application>
```

结果



Flex 快速入门: Flex 组件的定位和布局

大多数 Flex 容器使用预定义的规则集来自动定位您在其内定义的所有子组件。如果您使用 Canvas 容器, 或者 Application 或 Panel 容器, 其 layout 属性被设置为 "absolute", 则可以为其子级指定绝对位置, 或者使用基于限制的布局。

在 Flex 应用程序中定位组件的方法有三种:

- 使用自动定位
- 使用绝对定位
- 使用基于限制的布局

使用自动定位

对于大多数容器, Flex 会根据容器的布局规则 (如布局方向、容器填充和容器的子级之间的间隙) 自动定位容器子级。

对于使用自动定位的容器, 直接设置其子组件的 x 或 y 属性或调用 move() 方法没有任何效果, 或仅有一个临时效果, 因为布局计算将组件的位置设置为一个计算的结果, 而不是指定的值。

可以通过指定容器属性控制布局的各个方面。下列属性是最常见的:

- layout: 可能的值有 "horizontal"、 "vertical" 或 "absolute"。当设置为 "horizontal" 时, 容器会将其子级布局在一行内。 当设置为 "vertical" 时, 容器会将其子级布局在一列内。 有关 "absolute" 设置的信息, 请参阅绝对定位和基于限制的布局上的部分。
- horizontalAlign: 可能的值有 "left"、 "center" 或 "right"。
- verticalAlign: 可能的值有 "top"、 "middle" 或 "bottom"。

此示例覆盖 Application 容器的 horizontalAlign 属性的 "left" 的默认设置和 Application 容器的 verticalAlign 属性的 "top" 的默认设置以分别指定 "center" 和 "middle"。

此示例覆盖 Panel 容器的 layout 属性的 "vertical" 的默认设置, 从而以水平方式显示 Label 和 Button 控件。 Panel 容器的 padding 属性定义容器的空白区 (以像素计)。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/LayoutAutomatic/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="380" height="150"

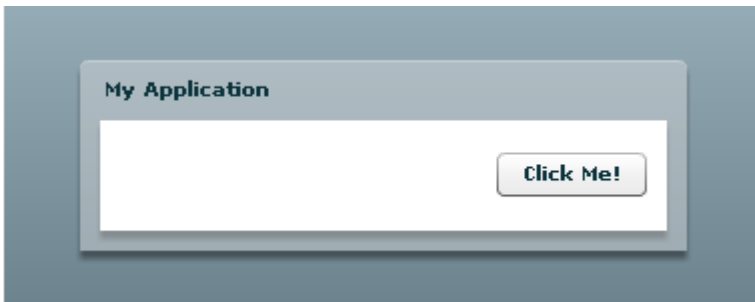
>
    <mx:Panel
        title="My Application"
        paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10"
        layout="horizontal" verticalAlign="middle"
```

```
>

<mx:Label id="myLabel" width="180" fontWeight="bold" fontSize="24"/>
<mx:Button
    id="myButton" label="Click Me!"
    click="myLabel.text = 'Hello, World!';"
/>

</mx:Panel>
</mx:Application>
```

结果



使用绝对定位

有三个容器支持绝对定位:

- 如果您将 `layout` 属性指定为 `"absolute"`, 则 `Application` 和 `Panel` 控件使用绝对定位。
- `Canvas` 容器始终使用绝对定位。

使用绝对定位, 你通过使用其 `x` 和 `y` 属性来指定子控件的位置, 或者指定基于限制的布局; 否则, `Flex` 会将该子级置于父容器的位置 `0,0` 处。当您指定 `x` 和 `y` 坐标时, 仅当您更改这些属性值时, `Flex` 才会重新定位控件。

此示例将 `Panel` 容器的 `layout` 属性设置为 `"absolute"`。接着, 它会设置 `Label` 和 `Button` 控件的 `x` 和 `y` 属性, 从而这两个控件会重叠。

提示: 使用绝对定位是使 `Flex` 控件重叠的唯一方法。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/LayoutAbsolute/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="280" height="170"
>
    <mx:Panel
        title="My Application" layout="absolute"
        width="220" height="90"
    >
        <mx:Label
            id="myLabel"
```

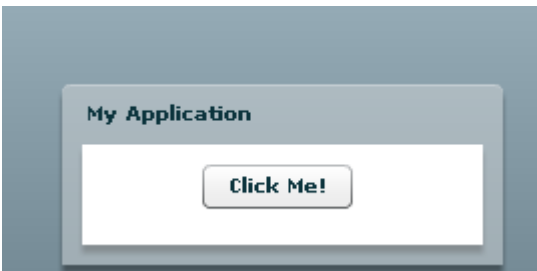
闲云无衣 <http://www.blogjava.net/xvridan>

```
x="10" y="10" width="180"
fontWeight="bold" fontSize="24"
/>

<mx:Button
    id="myButton"
    x="60" y="10"
    label="Click Me!"
    click="myLabel.text = &apos;Hello, World!&apos;;"
/>

</mx:Panel>
</mx:Application>
```

结果



使用基于限制的布局

您可以通过使用基于限制的布局同时管理子组件大小和定位子组件, 在该布局中您锚定组件的侧边或中心以相对于组件的容器进行定位。

您可以使用基于限制的布局来确定支持绝对定位的任何容器的即时子级的位置和大小。

您通过使用子组件的 `top`、`bottom`、`left`、`right`、`horizontalCenter` 或 `verticalCenter` 样式属性来指定限制。

锚定组件的边缘

您可以将组件的一个或多个边缘锚定在其容器的相应边缘的某个像素偏移处。 当容器调整大小时, 锚定的子级边缘保持与父级边缘的距离相同。

`top`、`bottom`、`left` 和 `right` 样式属性指定组件侧边与相应的容器侧边之间的距离 (以像素计)。

下面的示例中的 `Button` 控件具有锚定的底边和右边, 与它所在的 `Panel` 容器的边相距 10 个像素。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/LayoutConstraintsBottomRight/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="360" height="200"
/>
```

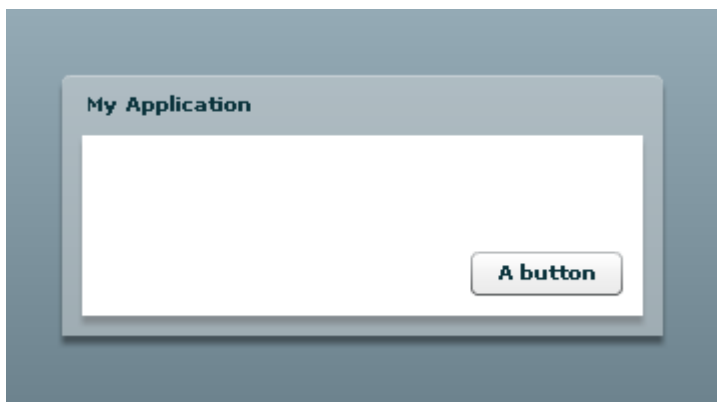
闲云无衣 <http://www.blogjava.net/xvridan>

```
<mx:Panel
    title="My Application" layout="absolute"
    width="300" height="130"
>
    <mx:Button
        id="myButton"
        label="A button"
        bottom="10"
        right="10"

        />

</mx:Panel>
</mx:Application>
```

结果



拉伸组件

如果在一个方向中锚定两个边,如顶边和底边,则在调整容器大小时,也会调整组件大小。下面的示例中的 **Button** 控件四个边都已被锚定,与它所在的 **Panel** 容器的边相距 10 个像素。

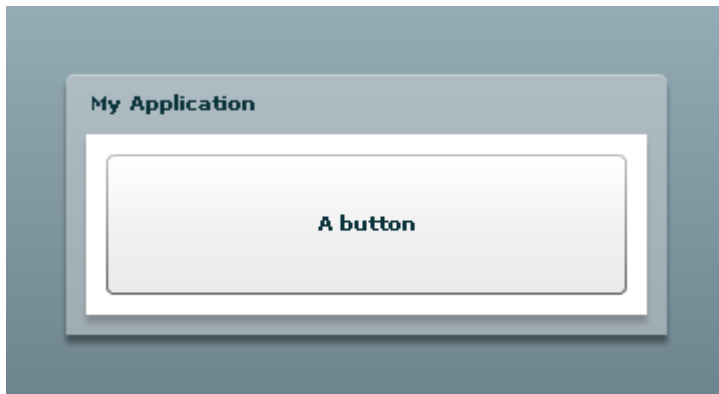
示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/LayoutConstraintsEdges/index.html"
    horizontalAlign="center" verticalAlign="middle"
    width="360" height="200"
>
    <mx:Panel
        title="My Application" layout="absolute"
        width="300" height="130"
    >
        <mx:Button
            id="myButton"
            label="A button"
            top="10"
            bottom="10"
            left="10"
            right="10"
        />
    </mx:Panel>
</mx:Application>
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
        />  
    </mx:Panel>  
</mx:Application>
```

结果



锚定组件的中心

您还可以将某个子组件的水平或垂直中心 (或全部两者) 锚定在容器中心的某个像素偏移处。除非您同时使用基于百分比的大小调整, 否则该子级不会在指定的尺寸内调整大小。

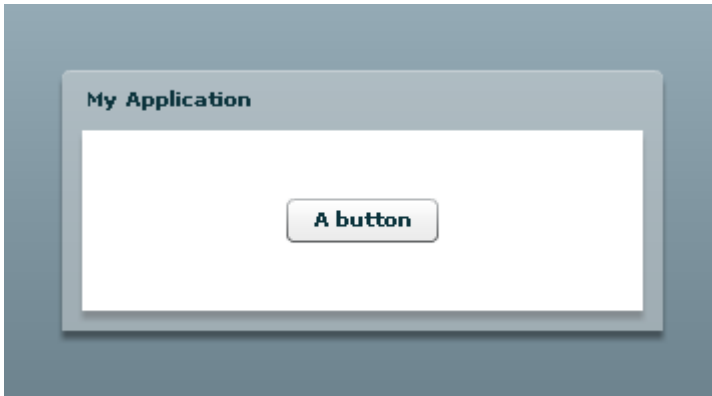
`horizontalCenter` 和 `verticalCenter` 样式指定在指定的方向上组件的中心点与容器的中心之间的距离; 一个负数会从中心向左或向上移动组件。

`horizontalCenter` 和 `verticalCenter` 样式指定从容器中心的偏移 (以像素计), 应将控件置于此处。下面的示例中的 `Button` 控件将两个样式都设置为 0 (零) 以完美地将它在 `Panel` 容器中居中。

示例

```
<?xml version="1.0" encoding="utf-8"?>  
  
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    viewSourceURL="src/LayoutConstraintsCenter/index.html"  
    horizontalAlign="center" verticalAlign="middle"  
    width="360" height="200"  
>  
    <mx:Panel  
        title="My Application" layout="absolute"  
        width="300" height="130"  
    >  
        <mx:Button  
            id="myButton"  
            label="A button"  
            verticalCenter="0"  
            horizontalCenter="0"  
        />  
    </mx:Panel>  
</mx:Application>
```

结果

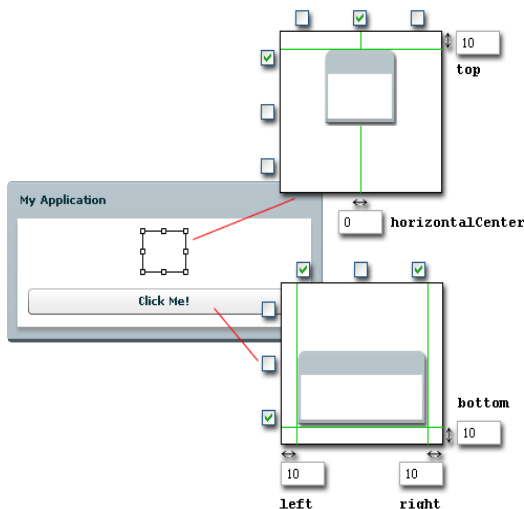


一个更加复杂的示例

下面的这个示例使用基于限制的布局来居中 Label 控件并使 Button 控件拉伸至几乎 Panel 的完全长度。将 Label 控件的 `top` 属性设置为 10 以限制它看起来与 Panel 的顶部的距离为 10 像素。将其 `horizontalCenter` 属性设置为 0 以完美地将它在 Panel 中居中。

将 Button 控件的 `left` 和 `right` 属性设置为 10 以使它拉伸至距离 Panel 的每一边都在 10 像素内。将其底边属性设置为 10 以限制其底边距离 Panel 的下边为 10 个像素。

下面的图说明您以可视方式设置的属性的效果。可视布局控件是 Flex Builder 2 的设计视图的组成部分。



提示: 不要使用 `verticalCenter` 样式属性指定 `top` 或 `bottom` 样式属性, `verticalCenter` 值会覆盖其他属性。类似地, 不要使用 `horizontalCenter` 样式属性指定 `left` 或 `right` 样式属性。

由基于限制的布局确定的大小会覆盖任何显式或基于百分比的大小规范。例如, 如果您指定 `left` 和 `right` 样式属性, 产生的基于限制的宽度会覆盖由 `width` 或 `percentWidth` 属性设置的任何宽度。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/LayoutConstraints/index.html"
    horizontalAlign="center" verticalAlign="middle">
```


闲云无衣 <http://www.blogjava.net/xvridan>

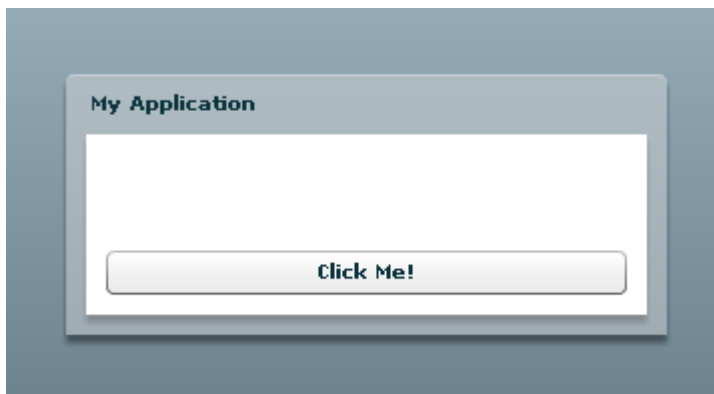
```
width="360" height="200"

>
<mx:Panel
    title="My Application" layout="absolute"
    width="300" height="130"
>
    <mx:Label
        id="myLabel"
        fontWeight="bold"
        fontSize="24"
        top="10"
        horizontalCenter="0"
    />

    <mx:Button
        id="myButton"
        label="Click Me!"
        click="myLabel.text = &apos;Hello, World!&apos;;"
        bottom="10"
        height="22"
        left="10"
        right="10"
    />

</mx:Panel>
</mx:Application>
```

结果



Flex 快速入门: 嵌入应用程序资源

可以在 Adobe® Flex™ 应用程序中嵌入各种类型的资源。嵌入的资源被编译到 Flex 应用程序的 SWF 文件中。它们不是在运行时加载的, 您并非必须使用您的应用程序部署原始资源文件。

提示: 嵌入资源的另一种方法是在运行时加载它们。在运行时加载的资源必须使用您的应用程序进行部署, 因为它们没有被编译到您的应用程序中。这具有保持 Flex 应用程序的文件大小更小和缩短其初始加载时间的优点。

可以嵌入具有 PNG、JPEG 和 GIF 文件格式的图像, SWF 文件, 具有 MP3 文件格式的声音文件, SVG 文件和字体。下列主题描述如何嵌入这些资源:

- 图像 (多个实例)
- 图像 (单一实例)
- 使用 scale-9 拉伸的图像
- 用于与 CSS 和外观一起使用的图像
- SWF 文件
- SWF 库资源
- 声音文件
- SVG 文件
- 字体

嵌入图像 (多个实例)

可以在 Flex 应用程序中嵌入具有 PNG、JPEG 或 GIF 文件格式的图像并创建它的一个或多个实例。

此示例使用 [Embed] 元数据标签在您的应用程序中嵌入图像并将它与可绑定的 ActionScript 类相关联。接着它将 Image 控件的 source 属性绑定到 Logo 类。可以将 Logo 类绑定到采用某个图像的任何组件属性, 比如 Button 控件的 icon 属性。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/EmbeddingImages/index.html"
    layout="horizontal" width="350" height="250"
>

    <mx:Script>
        <![CDATA[
            [Embed(source="assets/logo.png")]
            [Bindable]
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
        public var Logo:Class;
    ]]>
</mx:Script>

<mx:Image id="myLogo" source="{Logo}"/>

<mx:Image id="myLogo2" source="{Logo}"/>

</mx:Application>
```

结果



嵌入图像 (单一实例)

可以使用线上 `@Embed` 指令在 Flex 应用程序中嵌入只希望显示一次的图像。

此示例将一个图像组件添加到一个应用程序中并使用其 `source` 属性中的 `@Embed` 指令。若要在另一个 `Image` 控件中使用这一相同的图像, 则还必须将它嵌入到该组件中。 如果希望显示嵌入的图像的多个实例, 请使用 `[Embed]` 元数据标签。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/EmbeddingAnImage/index.html"
    width="200" height="240"
>
    <mx:Image id="myLogo" source="@Embed('assets/logo.png')"/>

</mx:Application>
```

结果

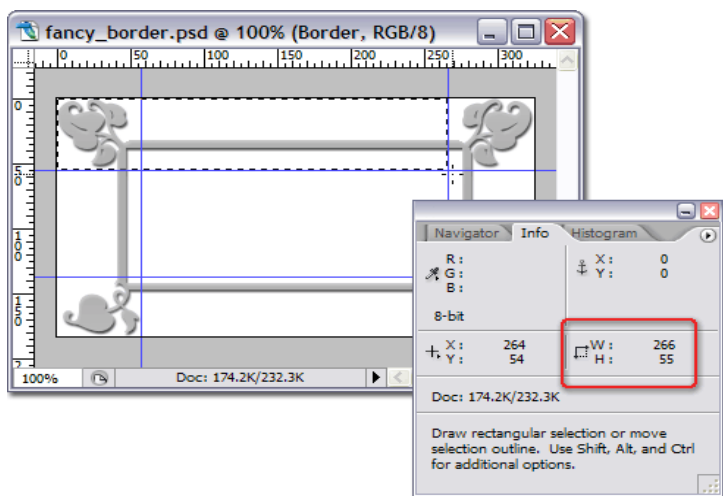


嵌入使用 scale-9 伸缩的图像

您可以将某个图像文件嵌入到您的 Flex 应用程序并以类似组件的方式智能地缩放它。通过使用 scale-9 功能, 您的图像的四个角根本没有缩放, 水平边框仅在水平方向上缩放, 而垂直边框仅在垂直方向上缩放。例如, 这对于以下情况很有用: 创建使用圆角的框, 或在您希望在缩放组件时保持边框清晰的位置进行组件样式调整。

此示例使用 Embed 元数据标签的 scaleGridTop、scaleGridBottom、scaleGridLeft 和 scaleGridRight 网格线位置属性来创建您的 scale-9 网格。

提示: 获得网格线位置的值的一种比较容易的方法是将指南与 Adobe® Photoshop® 中的 Rectangular Marquee 工具和 Info 选项板一起使用。



提示: 旋转嵌入的 scale-9 图像的实例会关闭该图像的 scale-9 以在将来进行任意大小转换。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/EmbeddingImagesScale9/index.html"
    layout="vertical" width="400" height="480"

>

    <mx:Script>
        <![CDATA[
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
[Embed (
    source="assets/fancy_border.png",
    scaleGridTop="55", scaleGridBottom="137",
    scaleGridLeft="57", scaleGridRight="266"

)]

[Bindable]
public var FancyBorderImage:Class;
]]>

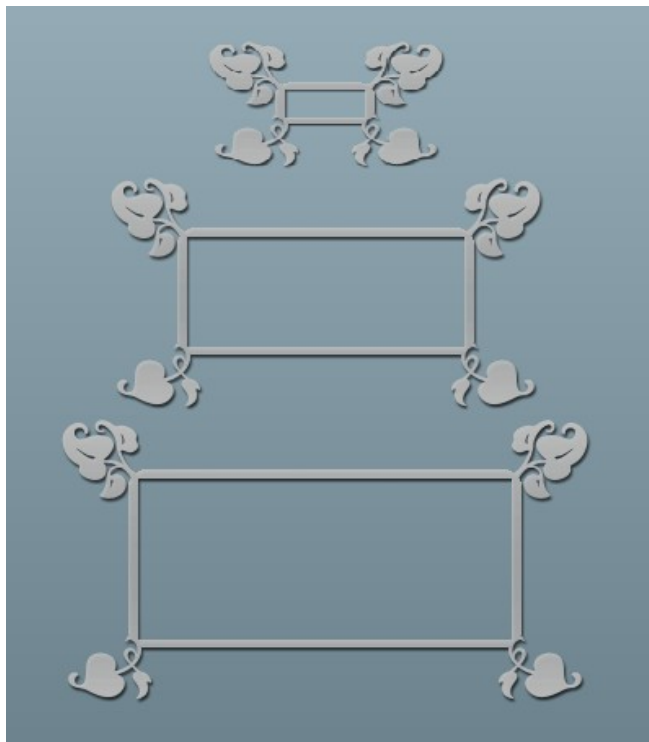
</mx:Script>

<mx:Image source="{FancyBorderImage}" width="146" height="82"/>

<mx:Image source="{FancyBorderImage}" width="266" height="150"/>
<mx:Image source="{FancyBorderImage}" width="325" height="183"/>

</mx:Application>
```

结果



使用 CSS 为外观嵌入图像

您可以在 Flex 应用程序中嵌入图像并将它用于设置组件的外观。

您可以定义一个 CSS 类型选择器为给出类型的所有组件设置外观。在此示例中, 您为 Button 控件创建了一个类型选择器。类型选择器通过使用 Embed 指定这些图像用作您的组件的外观属性。您还可以定义一个类选择器来创建可作为样式应用到特定组件的自定义 CSS 类。

提示: 类选择器创建可用来设置单独的组件的样式的已命名样式类, 方法是通过使用

`styleName` 属性将类选择器分配给该组件。 类型选择器为给出类型的所有组件设置样式, 如下面的示例所示。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/EmbeddingImagesCSS/index.html"
    layout="horizontal" width="270" height="100"
    horizontalAlign="center" verticalAlign="middle"

>

    <mx:Style>
        Button
        {
            upSkin: Embed("assets/box_closed.png");

            overSkin: Embed("assets/box.png");
            downSkin: Embed("assets/box_new.png");

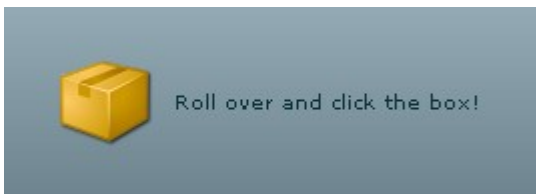
        }
    </mx:Style>

    <mx:Button/>

    <mx:Text text="Roll over and click the box!"/>

</mx:Application>
```

结果



嵌入 SWF 文件

嵌入 SWF 文件与嵌入图像几乎是相同的。 差别在于您可以将嵌入的 SWF 文件的实例当作 `MovieClip` 类的实例处理。(它们实际上是 `MovieClipAsset` 类的子类, `MovieClipAsset` 类是 `MovieClip` 类的子类。)

注意: 您无法直接访问嵌入的 SWF 文件的属性或方法。 但是, 您可以使用 `LocalConnection` 以允许它们进行通信。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/EmbeddingSwfFiles/index.html"
```

闲云无衣 <http://www.blogjava.net/xvridan>

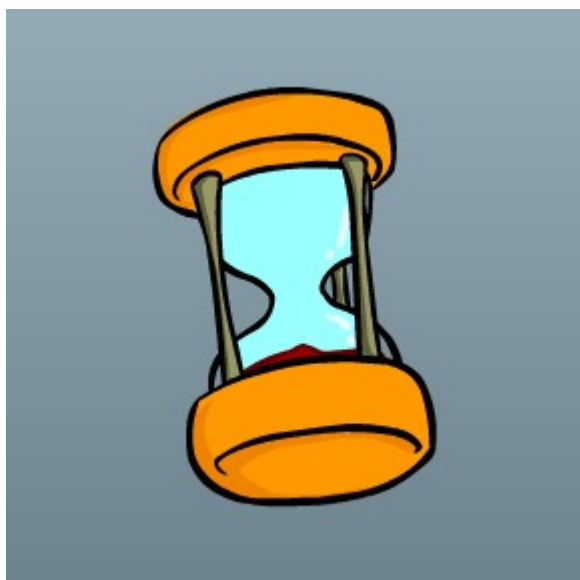
```
layout="horizontal" width="290" height="290"
horizontalAlign="center" verticalAlign="middle"
>

<mx:Script>
<![CDATA[
    [Embed(source="assets/hourglass.swf")]

    [Bindable]
    public var Hourglass:Class;
    ]]>
</mx:Script>

<mx:Image id="hourglass" source="{Hourglass}"/>
</mx:Application>
```

结果



嵌入 SWF 库资源

您 可以在应用程序中嵌入来自现有 SWF 库中的特定符号。Flash 定义三种类型的符号: Button、MovieClip 和 Graphic。您可以在 Flex 应用程序中嵌入 Button 和 MovieClip 符号, 但您不能嵌入 Graphic 符号, 因为无法为 ActionScript 导出 Graphic 符号。

此示例使用 [Embed] 元数据标签的 source 属性来指定包含您的库的 SWF 文件, 并使用 [Embed] 元数据标签的 symbol 属性来指定您要在该库中嵌入的符号的链接 ID。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/EmbeddingSwfLibraryAssets/index.html"
    layout="horizontal" width="450" height="240"
    horizontalAlign="center" verticalAlign="bottom"
>
```

```
<mx:Script>
<![CDATA[
    [Embed(source="assets/library.swf", symbol="BadApple")]

    [Bindable]
    public var BadApple:Class;

    [Embed(source="assets/library.swf", symbol="Pumpkin")]

    [Bindable]
    public var Pumpkin:Class;

    ]]>
</mx:Script>

<mx:Image id="badApple" source="{BadApple}" width="150" height="151.8"/>

<mx:Image id="pumpkin" source="{Pumpkin}" width="150" height="131.7"/>

</mx:Application>
```

结果



嵌入声音文件

您可以在 Flex 应用程序中通过使用 [Embed] 元数据标签嵌入 MP3 文件并播放它。

注意: 记住嵌入的声音文件会成为您的应用程序 (最终的 SWF 文件) 的一部分, 而 MP3 文件会很大, 从而会使您的应用程序变得很大并会对应用程序的下载速度产生负面影响。

此实例将该 MP3 的一个新实例创建为一个 SoundAsset。它使用 SoundAsset 类的 play() 方法来播放 MP3 文件的实例, 并存储返回的 SoundChannel 对象, 从而您可以稍后调用 SoundChannel 对象的 stop() 方法以结束播放。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
horizontalAlign="center" verticalAlign="center"
viewSourceURL="srcEmbeddingSoundFiles/index.html">
```



```
<mx:Script>
    <![CDATA[
        import mx.core.SoundAsset;
        import flash.media.*;

        [Embed(source="assets/pie-yan-knee.mp3")]

        [Bindable]
        public var Song:Class;

        public var mySong:SoundAsset = new Song() as SoundAsset;
        public var channel:SoundChannel;

        public function playSound():void
        {
            // Make sure we don't get multiple songs playing at the same
time        stopSound();

            // Play the song on the channel
            channel = mySong.play();
        }

        public function stopSound():void
        {
            // Stop the channel, but only if it exists
            if ( channel != null ) channel.stop();
        }

    ]]>
</mx:Script>

<mx:HBox>
    <mx:Button label="play" click="playSound();" />

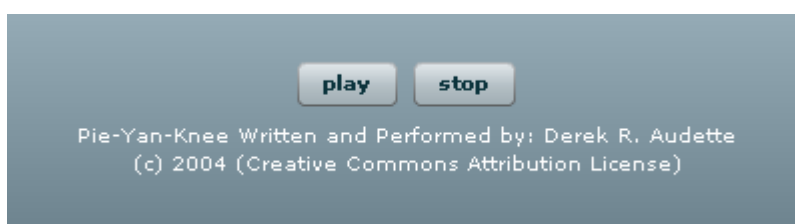
    <mx:Button label="stop" click="stopSound();" />
</mx:HBox>

<mx:Text width="348" textAlign="center" color="#ffffff">

    <mx:htmlText>
        <![CDATA[<a href="http://derekaudette.ottawaarts.com/music.php">Pie-
Yan-Knee Written and Performed by: Derek R. Audette © 2004 (Creative Commons
Attribution License)</a>]]>
    </mx:htmlText>
</mx:Text>

</mx:Application>
```

结果



嵌入 SVG 文件

可以将 SCG 文件嵌入到 Flex 应用程序中。

嵌入 SVG 文件与嵌入图像几乎是相同的。差别在于您可以将嵌入的 SVG 文件的实例当作 `Sprite` 类的实例处理。(它们实际上是 `SpriteAsset` 类的实例, `SpriteAsset` 类是 `Sprite` 类的子类。)嵌入的 SVG 文件还保留它们的矢量属性, 且在被缩放或转换时不会显示像素。

注意: 不能在运行时导入 SVG 文件; 仅可以在编译时在 Flex 应用程序中嵌入它们。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    viewSourceURL="srcEmbeddingSvgFiles/index.html"
    width="600" height="470"

>
    <mx:Script>
        <![CDATA[
            [Embed(source="assets/frog.svg")]

            [Bindable]
            public var SvgFrog:Class;
        ]]>
    </mx:Script>

    <mx:Image id="smallFrog" source="{SvgFrog}" width="128" height="130"/>

    <mx:Image id="largeFrog" source="{SvgFrog}"/>
</mx:Application>
```

结果



提示: SVG 青蛙图形是 Architetto Francesco Rollandin 创建的, 他慷慨地将它发布到 Open Clip Art Library 的公共区域中。您可以在网站找到其他可免费使用和试验的 SVG 文件。

嵌入字体

您希望在 Flex 应用程序中嵌入一种字体并将它用作基于文本的组件的样式。

下面的示例创建引用嵌入的字体的 `font-family` 名称的一个类选择器。接着它会创建一个 `Text` 控件并将其样式设置为该类选择器。

提示: 您在嵌入字体以节省文件大小时仅可以从字体添加某些字符, 方法是指定您的 `@font-face` 声明的 `unicode-range` 属性。

详细信息: 有关使用字体的详细信息, 请参阅 *Flex 2 开发人员指南* 中的“使用字体”。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="horizontal"
    horizontalAlign="center"
    verticalAlign="center"
    viewSourceURL="src/EmbeddingFonts/index.html"
>

    <mx:Style>
        @font-face
        {
            font-family: Copacetix;

            src: url("assets/copacetix.ttf");
            unicode-range:
                U+0020-U+0040, /* Punctuation, Numbers */

                U+0041-U+005A, /* Upper-Case A-Z */
                U+005B-U+0060, /* Punctuation and Symbols */
                U+0061-U+007A, /* Lower-Case a-z */
                U+007B-U+007E; /* Punctuation and Symbols */

        }

        .MyTextStyle
        {
            font-family: Copacetix;
            font-size: 24pt;
        }
    </mx:Style>

    <mx:Text styleName="MyTextStyle" text="Embedded fonts rock!" width="100%"/>

</mx:Application>
```

结果

闲云无衣 <http://www.blogjava.net/xvridan>

Embedded fonts rock!

Copacetix.com 的 Copacetix TrueType 免费字体。 在 Creative Commons Attribution-ShareAlike 2.0 许可证下得到许可。

Flex 快速入门: 构建简单的用户界面

使用控件

在 Adobe® Flex™ 模型 - 视图设计模式下, 用户界面组件代表视图。MXML 语言支持两种用户界面组件类型: 控件和容器。容器是包含控件和其他容器的屏幕的矩形区域。控件是表单元素, 如按钮、文本字段和列表框。

存在许多类型的 Flex 控件时, 此 QuickStart 描述三种最常见的控件: 基于文本的控件、基于按钮的控件和基于列表的控件。

- 使用基于文本的控件
- 使用基于按钮的控件
- 使用基于列表的控件

使用基于文本的控件

基于文本的控件用于显示文本和/或接收来自用户的文本输入。

在 Flex 中提供的基于文本的控件有 Label、Text、TextArea、TextInput 和 RichTextEditor 控件。TextInput 和 TextArea 组件既可以显示文本又可以接受用户输入, 而 Label 和 Text 控件仅用于显示文本。

Text 和 TextArea 控件可以显示跨多行的文本, 而 Label 和 TextInput 控件用于显示单行文本。

使用 RichTextEditor 控件可以输入文本、编辑文本和设置文本格式。用户通过使用位于 RichTextEditor 控件底部的子控件, 应用文本格式和 URL 链接。

所有基于文本的组件都有一个 text 属性, 可用来设置要显示的文本。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/ControlsTextBased/index.html"
    layout="horizontal" width="380" height="320"
    horizontalAlign="center" verticalAlign="middle"
>
    <mx:Panel
        title="Leave a comment"
        paddingBottom="10" paddingTop="10"
        paddingLeft="10" paddingRight="10"
    >
        <mx:Text
            text="Fill out this form to leave us a comment:"
            width="100%"
        />
        <mx:Label text="Name:" />
        <mx:TextInput id="userName" />
    </mx:Panel>
</mx:Application>
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
<mx:Label text="Email:" />
<mx:TextInput id="userEmail" />
<mx:Label text="Comment:" />
<mx:TextArea id="userComment" width="100%" />
</mx:Panel>
</mx:Application>
```

结果



使用基于按钮的控件

组件的 **Button** 系列包括 **Button**、**LinkButton**、**CheckBox**、**RadioButton** 和 **PopupButton** 控件。

Button 控件是一个常用的矩形按钮。**Button** 控件看起来就像被按下一样,在其面上有一个文本标签、一个图标或全部两者。可以选择为几个 **Button** 状态的每一个指定图形外观。可以创建一个普通 **Button** 控件或一个切换 **Button** 控件。只要在选中之后按下鼠标按钮,普通 **Button** 控件就会保持其被按下的状态。切换 **Button** 控件直到您又一次选中它之后,才会保持被按下的状态。

当用户选中控件时,按钮通常使用事件监听器来执行操作。当用户在 **Button** 控件上单击鼠标,且 **Button** 控件被启用时,它会发出一个 **click** 事件和一个 **buttonDown** 事件。

LinkButton 控件创建一个支持可选图标的单行超文本链接。它根本上是一个没有边框的 **Button** 控件。可以使用 **LinkButton** 控件在 Web 浏览器中打开 URL。

CheckBox 控件是一个常用的图形控件,它可以包含一个复选标记或者未被选中 (空)。在需要收集一组并不相互排斥的 **true** 或 **false** 值的地方,可以使用 **CheckBox** 控件。可以给 **CheckBox** 控件添加文本标签,并将文本标签置于复选框的左侧、右侧、顶部或底部。Flex 会裁剪 **CheckBox** 控件的标签以适合控件的边界。

使用 **RadioButton** 控件,用户可以在一组相互排斥的选项内作单一选择。**RadioButtonGroup** 控件由具有相同组名的两个或更多 **RadioButton** 控件组成。该组可以指由 `<mx:RadioButtonGroup>` 标签创建的组。用户一次仅选择该组的一个成员。选择某个未选中的组成员会取消选中该组内当前选中的 **RadioButton** 控件。

PopUpButton 控件给 Button 控件添加一个灵活的弹出控件界面。它包含一个主按钮和一个辅助按钮,这个辅助按钮也称为弹出按钮,当用户单击该弹出按钮时,它会弹出任何 UIComponent 对象。PopUpButton 控件的一个常见的用途是让弹出按钮打开 List 控件或 Menu 控件,这两个控件更改主按钮的功能和标签。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/ControlsButtonBased/index.html"
    layout="absolute" width="460" height="400"
>
    <mx:Script>
        <![CDATA[
            import flash.events.MouseEvent;
            import mx.controls.Alert;

            private const NL:String = "\r";

            private function submitButtonClickHandler (event:MouseEvent):void
            {
                var userDetails:String = "You submitted the following details:"
+ NL + NL;
                userDetails += "Name: " + userName.text + NL;
                userDetails += "Email: " + userEmail.text + NL;
                userDetails += "Hide email? " + (hideEmail.selected ? "Yes" :
"No") + NL + NL;
                userDetails += "Comment:" + NL + NL + userComment.text;

                Alert.show (userDetails);
            }

            private function emailButtonClickHandler (event:MouseEvent):void
            {
                var msg:String = "You can use the navigateToURL() method to open
a URL"
                msg += " using a call similar to the following:\r\r";
                msg += "navigateToURL (new URLRequest
('mailto:comments@somewhere.com'))";

                Alert.show (msg);
            }
        ]]>
    </mx:Script>

    <mx:Panel
        title="Leave a comment"
        left="10" top="10" right="10" bottom="10"
        layout="absolute"
    >
        <mx:Text
            text="Fill out this form to leave us a comment:"
            width="250" x="10" y="10" fontWeight="bold"
        />
        <mx:Label text="Name:" x="10" y="38"/>
        <mx:TextInput id="userName" y="36" right="10" left="90"/>
        <mx:Label text="Email:" x="10" y="68"/>
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
<mx:TextInput id="userEmail" y="66" right="10" left="90"/>
<mx:Label text="Comment:" x="10" y="129"/>
<mx:TextArea id="userComment" left="10" right="10" height="109"
bottom="40"/>

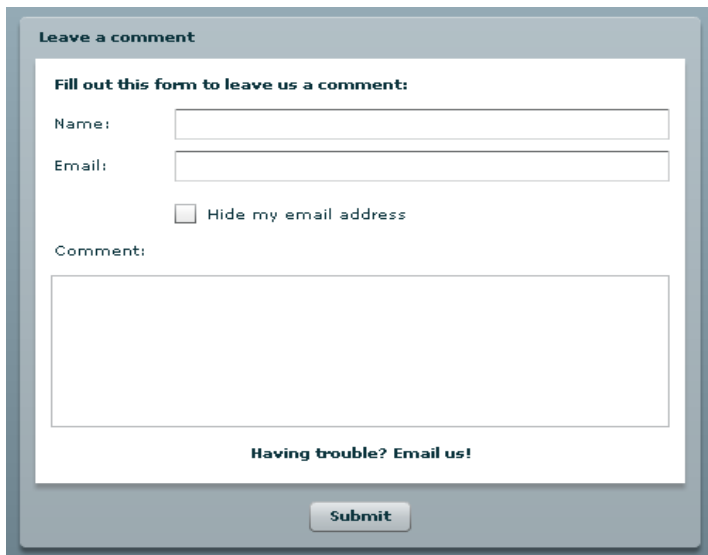
<mx:CheckBox
    id="hideEmail"
    y="103" left="90"
    label="Hide my email address"
    selected="true"
/>

<mx:LinkButton
    id="emailButton"
    y="272" horizontalCenter="0"
    label="Having trouble? Email us!"
    click="emailButtonClickHandler(event);"
/>

<mx:ControlBar x="120" y="258" horizontalAlign="center">
    <mx:Button
        id="submitButton" label="Submit"
        click="submitButtonClickHandler(event);"
    />
</mx:ControlBar>

</mx:Panel>
</mx:Application>
```

结果



The screenshot shows a web form titled "Leave a comment". Inside the form, there is a heading "Fill out this form to leave us a comment:". Below this, there are three input fields: "Name:", "Email:", and a checkbox labeled "Hide my email address". Below the checkbox is a large text area labeled "Comment:". At the bottom of the form, there is a link that says "Having trouble? Email us!" and a "Submit" button.

使用基于列表的控件

基于列表的控件是在其继承层次结构内的某些点上扩展 `ListBase` 类的那些控件。它们包括 `ComboBox`、`List`、`HorizontalList`、`DataGrid`、`Tile`、`Menu` 和 `Tree` 控件。

几个 Flex 框架组件 (包括所有基于列表的控件) 表示来自某个数据提供程序的数据, 数据提供程序是一个包含控件所需数据的对象。例如, 一个 `Tree` 控件的数据提供程序确定树的结构和分配给每个树节点的相关联的数据, 而一个 `ComboBox` 控件的数据提供程序确定控件的下

拉列表中的项目。

注意: 许多标准控件 (包括 `ColorPicker` 和 `MenuBar` 控件) 也从数据提供程序获取数据。显示应用程序数据的控件有时被称为数据提供程序控件。有关使用数据提供程序的详细信息, 请参阅 *Flex 2 开发人员指南* 中的“使用数据提供程序和集合”。

您 可以使用两种方法设置组件的数据提供程序: 第一种方法是通过将 `Array` 或 `Collection` 定义为取得数据提供程序的控件的子标签, 在线上在 `MXML` 中定义数据提供程序。这种方法具有实施快速的优点, 适合与静态数据一起使用及用于原型设计。第二种方法是使用数据绑定将控件绑定到使用 `ActionScript` 定义的现有 `Array` 或 `Collection`。这后一种方法用于显示由服务器端调用引起的数据及用于绑定到在 `ActionScript` 中创建的数据结构。这种方法也是这两种方法中较有可伸缩性的。

下面的示例说明这两种方法。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/ControlsListBased/index.html"
    layout="horizontal" width="460" height="360"
>

    <mx:Script>
        <![CDATA[
            import flash.events.MouseEvent;
            import mx.controls.Alert;
            import mx.collections.ArrayCollection;

            private const NL:String = "\r";

            // A data provider created by using ActionScript
            [Bindable]
            private var subscriptions:ArrayCollection =
                new ArrayCollection
            (
                [
                    {data:0, label:"Print"},
                    {data:1, label:"Website"},
                    {data:2, label:"RSS (text)"},
                    {data:3, label:"Podcast"}
                ]
            );

            private function submitButtonClickHandler(event:MouseEvent):void
            {
                var userDetails:String = "You submitted the following details:"
+ NL + NL;
                userDetails += "Name: " + userName.text + NL;
                userDetails += "Email: " + userEmail.text + NL;
                userDetails += "Site rating: " + userRating.selectedItem.label +
+ NL + NL;
                userDetails += "Subscriptions:";

                var selectedSubscriptionItems:Array =
userSubscriptions.selectedItems;
                for ( var i:String in selectedSubscriptionItems)
```

```
        {
            // Display the selected subscriptions, separated by commas
            userDetails += selectedSubscriptionItems[i].label + ", ";
        }
        // Remove the last comma and space from the string we're
displaying        userDetails = userDetails.substring(0, userDetails.length-2);

        Alert.show ( userDetails );
    }

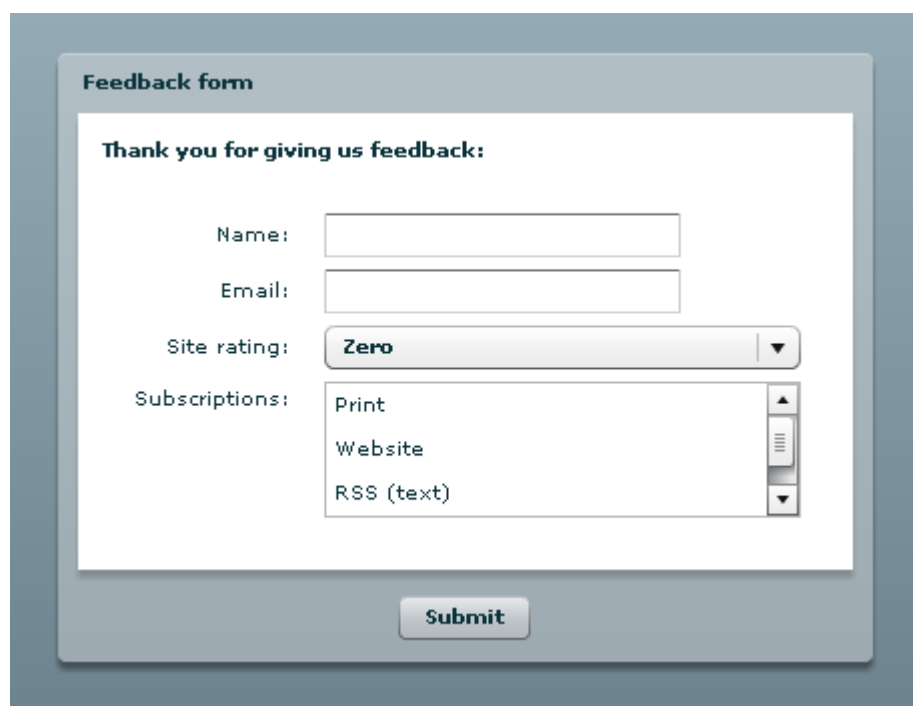
    ]]>
</mx:Script>

<mx:Panel
    title="Feedback form" width="99%"
    paddingLeft="10" paddingTop="10" paddingRight="10" paddingBottom="10"
    layout="vertical"
>
    <mx:Text
        text="Thank you for giving us feedback:"
        width="100%" fontWeight="bold"
    />
    <mx:Form width="100%">
        <mx:FormItem label="Name:" width="100%">
            <mx:TextInput id="userName" />
        </mx:FormItem>
        <mx:FormItem label="Email:" width="100%">
            <mx:TextInput id="userEmail" />
        </mx:FormItem>
        <mx:FormItem label="Site rating:" width="100%">
            <mx:ComboBox id="userRating" width="100%">
                <!-- An inline data provider -->
                <mx:Array>
                    <mx:Object data="0" label="Zero" />
                    <mx:Object data="1" label="One" />
                    <mx:Object data="2" label="Two" />
                    <mx:Object data="3" label="Three" />
                    <mx:Object data="4" label="Four" />
                </mx:Array>
            </mx:ComboBox>
        </mx:FormItem>
        <mx:FormItem label="Subscriptions:" width="100%">
            <mx>List
                id="userSubscriptions" rowCount="3"
                allowMultipleSelection="true" width="100%"
                dataProvider="{subscriptions}"
            />
        </mx:FormItem>
    </mx:Form>

    <mx:ControlBar x="120" y="258" horizontalAlign="center">
        <mx:Button
            id="submitButton" label="Submit"
            click="submitButtonClickHandler(event);"
        />
    </mx:ControlBar>

</mx:Panel>
</mx:Application>
```

结果



A screenshot of a web-based feedback form titled "Feedback form". The form is set against a light blue background. It begins with a thank-you message: "Thank you for giving us feedback:". Below this, there are four input fields: "Name:" (a text box), "Email:" (a text box), "Site rating:" (a dropdown menu currently showing "Zero"), and "Subscriptions:" (a list box containing "Print", "Website", and "RSS (text)". To the right of the list box is a vertical scrollbar. At the bottom center of the form is a "Submit" button.

Feedback form

Thank you for giving us feedback:

Name:

Email:

Site rating: **Zero** ▼

Subscriptions:
Website
RSS (text)

Submit

使用容器

容器定义 Adobe® Flash® Player 的绘图表面的一个矩形区域。在容器内, 可以定义希望出现在容器内的组件, 包括控件和容器。在容器内定义的组件称为容器的子级。Adobe Flex 提供范围广泛的容器, 从简单的框到面板和表单, 到在子容器之间提供内置导航 的元素 (如折叠式导航器或选项卡式导航器)。

Container 类是所有 Flex 容器类的基本类。 扩展 Container 类的容器添加它们自己的功能以进行子组件布局。








在 Flex 应用程序的根部是称为 Application 容器的单一容器, 它代表整个 Flash Player 绘图表面。此 Application 容器容纳所有其他容器和组件。

提示: 不同的容器支持不同的布局规则, 包括 automatic 和 absolute 定位。关于这一点的详细信息, 请参阅 Flex 组件的定位和布局。

- 使用布局容器
- 使用导航器

使用布局容器

使用布局容器可进行用户界面布局。下面的表格描述下面的示例使用的布局容器:

	名称	描述
	Panel	Panel 容器显示一个标题栏、一个标题、一个边框及其子级。默认情况下, Panel 容器会对子组件进行垂直布局, 并且可以通过将布局属性设置为 "absolute" 或 "horizontal" 来覆盖此设置。
	HDividedBox	HDividedBox 容器对子组件进行水平布局, 除了在子级之间插入一个可调整的分割线之外, 它与 HBox 容器很相似。
	VDividedBox	VDividedBox 容器对子组件进行垂直布局, 而且也在子级之间插入一个可调整的分割线。
	Tile	Tile 容器以多行或多列的形式排列其子级。
	Form	Form 容器以标准的表单格式排列其子级。
	ApplicationControlBar	ApplicationControlBar 容器容纳提供全局导航和应用程序命令的组件, 并可以停靠在 Application 容器的上边缘。
	ControlBar	ControlBar 容器将控件置于 Panel 或 TitleWindow 容器的下边缘。

此外, 该示例使用 Spacer 控件 (它不是一个容器) 来帮助进行界面的布局。

提示: Spacer 控件是用于在自动定位的容器内准确定位元素的一个不可见控件。 在此示例中, Spacer 控件是 ApplicationControlBar 容器中唯一基于百分比的组件。 Flex 调整 Spacer 控

件的大小以占据容器中其他组件不需要的所有可用空间。通过扩展 `Spacer` 控件, `Flex` 将 `Button` 控件推到该容器的右边缘。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="525" height="400"
    viewSourceURL="src/LayoutContainers/index.html"
>
    <mx:ApplicationControlBar dock="true">
        <mx:Label
            text="ApplicationControlBar"
            fontFamily="Verdana" fontWeight="bold" fontSize="12"
        />
        <mx:Spacer width="100%"/>
        <mx:Button label="Log out"/>
    </mx:ApplicationControlBar>

    <mx:Panel
        layout="horizontal" title="Panel"
        width="100%" height="100%"
    >
        <mx:HDividedBox width="100%" height="100%">
            <mx:Panel
                width="100%" height="100%"
                headerHeight="0" borderStyle="solid" shadowDistance="0"
            >
                <mx:Label text="Panel (without header)" fontWeight="bold" />
                <mx:Form>
                    <mx:FormHeading label="First form" />
                    <mx:FormItem label="Name:">
                        <mx:TextInput width="100"/>
                    </mx:FormItem>
                    <mx:FormItem label="Email:">
                        <mx:TextInput width="100"/>
                    </mx:FormItem>
                </mx:Form>
            </mx:Panel>

            <mx:Panel
                width="100%" height="100%"
                headerHeight="0" borderStyle="solid" shadowDistance="0"
            >
                <mx:Label text="Panel (without header)" fontWeight="bold" />
                <mx:Form>
                    <mx:FormHeading label="Second form" />
                    <mx:FormItem label="Name:">
                        <mx:TextInput width="100"/>
                    </mx:FormItem>
                    <mx:FormItem label="Email:">
                        <mx:TextInput width="100"/>
                    </mx:FormItem>
                </mx:Form>
            </mx:Panel>
        </mx:HDividedBox>

        <mx:ControlBar horizontalAlign="center">
            <mx:Label text="ControlBar in Panel" fontWeight="bold"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

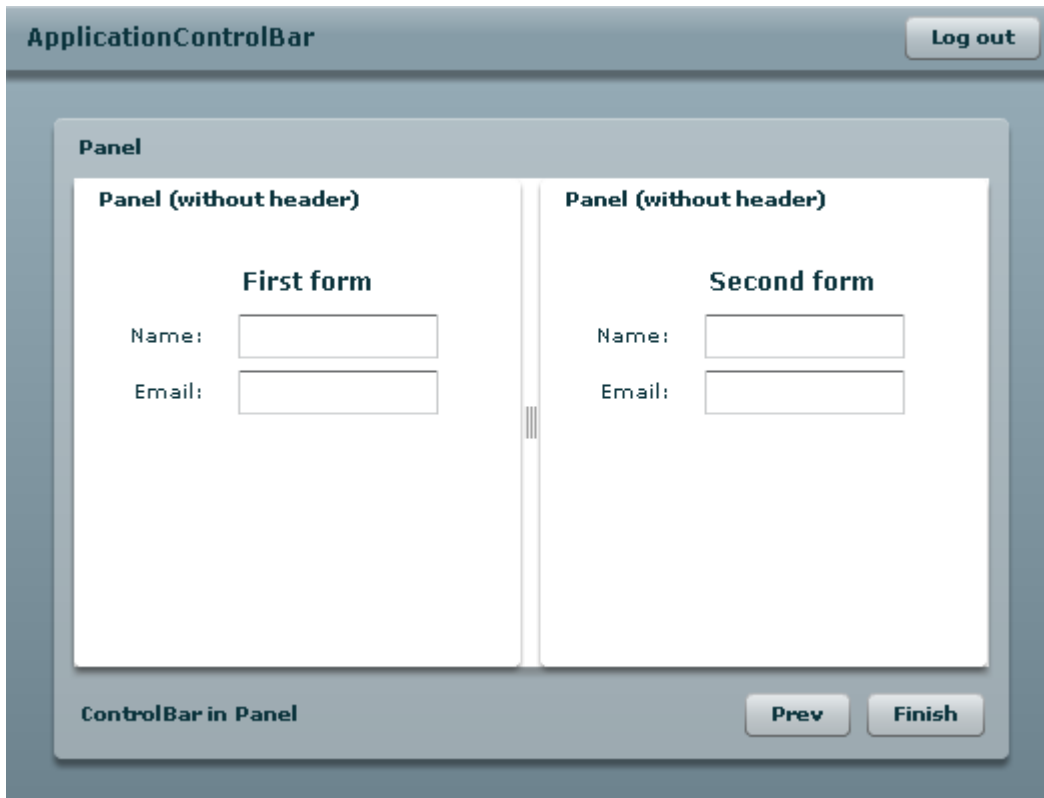
闲云无衣 <http://www.blogjava.net/xvridan>

```
<mx:Spacer width="100%"/>
<mx:Button label="Prev"/>
<mx:Button label="Finish"/>
</mx:ControlBar>
```

```
</mx:Panel>
```

```
</mx:Application>
```

结果






使用导航容器

导航器容器控制子级是其他容器的多个子级之间的用户移动或导航。

导航器容器的直接子级必须是容器, 要么是布局容器, 要么是导航器容器。 无法在导航器内直接嵌套控件; 控件必须是导航器容器的子容器的子级。

下面的表格描述下面的示例使用的导航器容器:

	名称	描述
	Accordion	Accordion 容器定义一个子面板序列,但一次仅显示一个面板。若要导航容器,用户会单击与他们需要访问的子面板相对应的导航按钮。使用 Accordion 容器,用户可以按任何顺序访问子面板以在表单中前后移动。
	TabNavigator	TabNavigator 容器创建和管理一组选项卡,使用它们可在其子级中间导航。TabNavigator 容器的子级是其他容器。TabNavigator 容器为每个子级创建一个选项卡。当用户选中某个选项卡时,TabNavigator 容器会显示相关联的子级。
	ViewStack	ViewStack 导航器容器由彼此堆叠在一起的子容器的一个集合组成,一次只有一个容器是可见的或活动的。ViewStack 容器不为用户定义切换当前活动容器的内置机制;您必须使用 LinkBar、TabBar、ButtonBar 或 ToggleButtonBar 控件或自己在 ActionScript 中构建逻辑让用户来更改当前活动的子级。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    width="550" height="550"
    viewSourceURL="src/NavigationContainers/index.html"
>
    <mx:Panel
        layout="absolute"
        left="10" top="10" right="10" bottom="10" title="Navigators"
    >
        <mx:Accordion width="47.5%" height="200" top="36" left="10">
            <mx:Canvas label="Navigation button 1" width="100%" height="100%">
                <mx:Label x="10" y="10" text="Contents 1"/>
            </mx:Canvas>
            <mx:Canvas label="Navigation button 2" width="100%" height="100%">
                <mx:Label x="10" y="10" text="Contents 2"/>
            </mx:Canvas>
            <mx:Canvas label="Navigation button 3" width="100%" height="100%">
                <mx:Label x="10" y="10" text="Contents 3"/>
            </mx:Canvas>
        </mx:Accordion>

        <mx:TabNavigator right="10" width="47.5%" height="200" y="36">
            <mx:Canvas label="Tab 1" width="100%" height="100%">
                <mx:Label x="10" y="10" text="Contents 1"/>
            </mx:Canvas>
            <mx:Canvas label="Tab 2" width="100%" height="100%">
                <mx:Label x="10" y="10" text="Contents 2"/>
            </mx:Canvas>
            <mx:Canvas label="Tab 3" width="100%" height="100%">
                <mx:Label x="10" y="10" text="Contents 3"/>
            </mx:Canvas>
        </mx:TabNavigator>

        <mx:ViewStack
            id="myViewStack"
            width="47.5%" height="200" right="10" bottom="10"
            borderColor="#000000" borderStyle="solid"
```

```
>
    <mx:Canvas label="View 1" width="100%" height="100%">
        <mx:Label x="10" y="10" text="Contents 1"/>
    </mx:Canvas>
    <mx:Canvas label="View 2" width="100%" height="100%">
        <mx:Label x="10" y="10" text="Contents 2"/>
    </mx:Canvas>
    <mx:Canvas label="View 3" width="100%" height="100%">
        <mx:Label x="10" y="10" text="Contents 3"/>
    </mx:Canvas>
</mx:ViewStack>

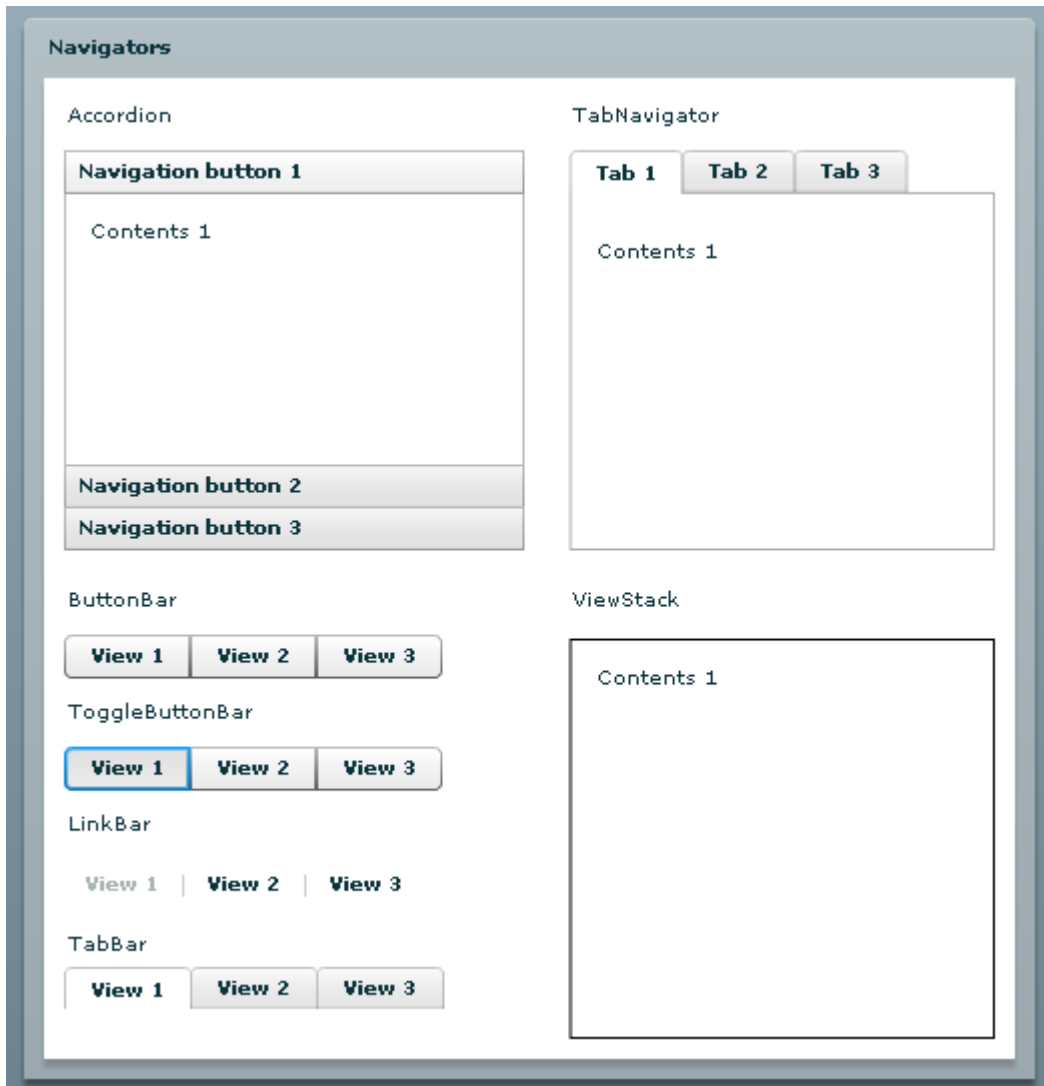
<!-- Labels for the various controls -->
<mx:Label x="10" y="252" text="ButtonBar"/>
<mx:Label x="10" y="10" text="Accordion"/>
<mx:Label x="262.5" y="10" text="TabNavigator"/>
<mx:Label x="262.5" y="252" text="ViewStack"/>
<mx:Label x="10" y="308" text="ToggleButtonBar"/>
<mx:Label x="10" y="364" text="LinkBar"/>
<mx:Label x="10" y="424" text="TabBar"/>

<!--
    All these navigators use the same dataProvider,
    namely, the myViewStack ViewStack instance.
    Changing the selected view in one will affect
    all the others also.
-->
<mx:ButtonBar x="10" y="278" dataProvider="{myViewStack}" />
<mx:ToggleButtonBar x="10" y="334" dataProvider="{myViewStack}" />
<mx:LinkBar x="10" y="390" dataProvider="{myViewStack}" />
<mx:TabBar x="10" y="444" dataProvider="{myViewStack}" />

</mx:Panel>

</mx:Application>
```


结果



更多信息

有关使用布局容器的详细信息, 请参阅 Flex 2 开发人员指南*中的“容器简介”、“使用 Application 容器”、“使用 Layout 容器”和“使用 Navigator 容器”。

设置组件的样式

样式对于定义 Adobe® Flex™ 应用程序的外观和感觉 (外观) 很有用。您可以使用它们来更改单一组件的外观, 或在所有组件上应用它们。

在 Flex 中应用样式有许多方法。某些样式提供更多粒度控制并能以编程方式被执行。其他样式不像那么灵活, 但可能需要较少的计算。在 Flex 中, 可以使用以下几种方法将样式应用到控件:

- 使用本地样式定义
- 使用外部样式表
- 使用线上样式
- 使用 `setStyle()` 方法

Flex 不支持使用层叠样式表 (CSS) 来控制组件的所有可视方面。属性, 比如 `x`, `y`, `width` 和 `height` 是 `UIComponent` 类的属性, 而不是其样式, 因此, 无法在 CSS 中进行设置。您还必须知道您的主题支持哪些属性。Flex 中的默认主题并不支持所有样式属性。

主题是定义 Flex 应用程序的外观和感觉的一组样式。主题可以将一些事项定义得像应用程序的配色方案或常见字体一样简单, 或者它可以成为应用程序所使用的所有组件的完全重新设置的外观。主题通常采取 SWC 文件的形式。但是, 主题也可以是 CSS 文件和嵌入式图形资源, 如来自 SWF 文件的符号。

有关主题中支持的样式属性的详细信息, 请参阅 Flex 2 开发人员指南*中的“关于支持的样式”。

若要确定特定可视组件所支持的样式, 请参阅 Adobe Flex 2 语言参考中该组件的样式部分。

使用本地样式定义

可以使用 `<mx:Style>` 标签在 MXML 文件中创建本地样式定义。此标签包含符合 CSS 2.0 语法的样式表定义。这些定义应用到当前文档和当前文档的所有子级。

下面的示例创建两个类选择器, `.solidBorder` 和 `.solidBorderPaddedVertically`, 并通过使用它们的 `styleName` 属性来使用它们设置两个 `VBox` 容器的样式。如此示例所示, 使用类选择器使您能够为相同组件的多个实例提供不同的样式。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="150" height="140"
    viewSourceURL="src/StylesStyleTag/index.html"
>
    <mx:Style>
        .solidBorder { border-style: solid; }

        .solidBorderPaddedVertically
        {
            padding-top: 12;
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
padding-bottom: 12;
border-style: solid;
}
</mx:Style>
<mx:VBox styleName="solidBorder">
    <mx:Button label="Submit"/>
</mx:VBox>
<mx:VBox
    styleName="solidBorderPaddedVertically"
>
    <mx:Button label="Submit"/>
</mx:VBox>
</mx:Application>
```

提示: 如果您希望某个特定组件的所有实例共享相同的样式, 则可以使用 CSS 类型选择器。例如, 通过使用下列类型选择器, 应用程序中的所有 VBox 实例将有一个实心边框。

```
VBox { border-style: solid; }
```

结果



使用外部样式表

Flex 支持外部 CSS 样式表。若要将样式表应用到当前文档及其子文档, 请使用 `<mx:Style>` 标签的 `source` 属性。

注意: 您应尝试限制在应用程序中使用的样式表的数量, 并在应用程序中仅设置处于顶层文档的样式表 (顶层文档是包含 `<mx:Application>` 标签的文档)。如果您在子文档中设置一个样式表, 则会发生异常结果。

下面的示例在称为 `external.css` 的外部 CSS 文件中定义两个 CSS 类选择器。您可以通过在 `<mx:Style>` 标签的来源属性中指定其路径和文件名, 在 Flex 应用程序中使用外部 CSS 文件。

示例:

外部 CSS 文件

```
/* An external CSS file */
.solidBorder
{
    borderStyle: "solid";
}

.solidBorderPaddedVertically
{
    borderStyle: "solid";
}
```

```
paddingTop: 12px;  
paddingBottom: 12px;  
}
```

MXML 文件

```
<?xml version="1.0" encoding="utf-8"?>  
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    width="150" height="140"  
    viewSourceURL="src/StylesExternal/index.html"  
>  
    <mx:Style source="styles/external.css" />  
  
    <mx:VBox styleName="solidBorder">  
        <mx:Button label="Submit"/>  
    </mx:VBox>  
  
    <mx:VBox styleName="solidBorderPaddedVertically">  
        <mx:Button label="Submit"/>  
    </mx:VBox>  
</mx:Application>
```

结果



使用线上样式

可以将样式属性设置为 MXML 标签中该组件的属性。除了通过使用 `setStyle()` 标签定义的运行时样式更改之外, 线上样式定义优先于任何其他样式定义。例如, 您可以通过使用 `<mx:VBox>` 标签的 `paddingTop` 和 `paddingBottom` 属性在 `Box` 容器的边框与其内容之间设置填充。类似地, 您可以使用 `borderStyle` 属性来定义组件的边框的可视外观。

示例

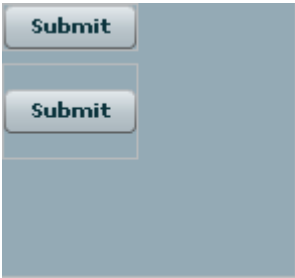
```
<?xml version="1.0" encoding="utf-8"?>  
<mx:Application  
    xmlns:mx="http://www.adobe.com/2006/mxml"  
    width="150" height="140"  
    viewSourceURL="src/StylesTagAttributes/index.html"  
>  
    <mx:VBox id="myVBox1" borderStyle="solid">  
        <mx:Button label="Submit"/>  
    </mx:VBox>  
</mx:Application>
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
id="myVBox2" borderStyle="solid"
paddingTop="12" paddingBottom="12"
>
    <mx:Button label="Submit"/>

</mx:VBox>
</mx:Application>
```

结果



使用 setStyle() 方法

使用 `setStyle()` 方法操纵 `ActionScript` 中的控件实例的样式属性。使用此方法应用样式与使用样式表应用样式相比,需要在客户端上有更大数量的处理功率,但对如何应用样式提供更多粒度控制。

`setStyle()` 方法采用两个参数: 样式名称和样式值。

提示: 第一次实例化对象和设置样式时,应尝试应用样式表而不是使用 `setStyle()` 方法,因为据估算,此方法很昂贵。仅当要在运行时过程中更改对象的样式时,才应使用此方法。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    width="150" height="140"
    viewSourceURL="src/StylesSetStyle/index.html"
>
    <mx:Script>
        <![CDATA[
            private function initVBox():void
            {
                myVBox2.setStyle("paddingTop", 12);
                myVBox2.setStyle("paddingBottom", 12);
            }
        ]]>
    </mx:Script>

    <mx:VBox borderStyle="solid">
        <mx:Button label="Submit"/>

    </mx:VBox>
    <mx:VBox
        id="myVBox2" borderStyle="solid"
        paddingTop="12" paddingBottom="12"
        initialize="initVBox();"
    >
```

[闲云无衣 http://www.blogjava.net/xvridan](http://www.blogjava.net/xvridan)

```
>  
<mx:Button label="Submit"/>  
</mx:VBox>  
</mx:Application>
```

结果



添加效果

效果是在较短时间内发生的对组件的更改。 效果的例子有: 淡化组件、调整组件大小和移动组件。 一种效果与一个触发器相结合才能形成一个行为, 如组件上的鼠标单击、组件获得焦点或组件变成可见的。 在 MXML 中, 您将效果应用为控件或容器的属性。 Adobe® Flex™ 提供具有默认属性的一组内置效果。

作为对某些用户或编程操作的响应, 行为使您可以将动画、动作和声音添加到应用程序中。 例如, 您可使用行为在获得焦点时弹出对话框, 或是在用户输入无效的值时发出声音。

Flex 触发器属性是作为层叠样式表 (CSS) 样式被实施的。 在 Adobe Flex 2 语言参考中, 触发器被列出在标题“效果”的下面。

若要创建行为, 您定义一个具有唯一 ID 的特定效果并将它绑定到触发器。 例如, 下面的代码创建两个缩放效果: 一个用于轻微缩小组件, 一个用于将组件还原至其原始大小。 这些效果通过使用它们的唯一 ID 被分配到“按钮”组件上的 `mouseDownEffect` 和 `mouseUpEffect` 触发器上。

注意如何将 `Panel` 容器的 `autoLayout` 属性设置为 `"false"`。 这样是为了阻止在按钮改变大小时面板改变大小。

示例

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" width="170" height="140"
>
    <!-- Define effects -->
    <mx:Zoom id="shrink" duration="100" zoomHeightTo=".9" zoomWidthTo=".9" />

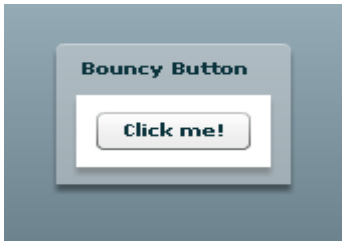
    <mx:Zoom id="revert" duration="50" zoomHeightTo="1" zoomWidthTo="1" />

    <mx:Panel
        title="Bouncy Button" paddingTop="10" paddingBottom="10"
        paddingLeft="10" paddingRight="10" autoLayout="false"
        left="41" top="24" right="42">

        <!-- Assign effects to target -->
        <mx:Button
            id="bouncyButton" label="Click me!"
            mouseDownEffect="{shrink}" mouseUpEffect="{revert}"
        />

    </mx:Panel>
</mx:Application>
```

结果



使用效果方法和事件

您可以调用效果上的方法来改变它们播放的方式。例如,可以通过调用效果的 `pause()` 方法来暂停效果,并通过使用其 `resume()` 方法来继续该效果。可以通过调用效果的 `end()` 方法来结束该效果。

当效果开始和效果结束时,它也会发出 `startEffect` 和 `endEffect` 事件。您可以监听这些事件并响应您的事件状态中的更改。

下面的示例使用“移动”效果的方法和事件来创建一个简单的游戏。该游戏的目标是使直升飞机尽可能接近靶而又不撞到它。靠得越近,赢得的点数越多。

该游戏使用取自 **SWF** 文件库的电影剪辑符号来表示靶、直升飞机和爆炸动画。有关这如何工作的详细信息,请参阅嵌入资源。

示例

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" width="525" height="450"
    viewSourceURL="src/EffectsChopperGame/index.html"
>
    <mx:Script>

        <![CDATA[
            // Easing equations have to be explicitly imported.
            import mx.effects.easing.Quadratic;

            // Embed movie clip symbols from the library of a Flash SWF.

            [Embed(source="assets/library.swf", symbol="DartBoard")]

            public var DartBoard:Class;

            [Embed(source="assets/library.swf", symbol="Helicopter")]

            public var Helicopter:Class;

            [Embed(source="assets/library.swf", symbol="Explosion")]

            public var Explosion:Class;

            // Event handler for the effectEnd event.
            private function endEffectHandler():void
            {

                helicopter.visible = false;
                explosion.visible = true;
```


[闲云无衣 http://www.blogjava.net/xvridan](http://www.blogjava.net/xvridan)

```
        score.text = "0";
        pauseButton.enabled = resumeButton.enabled =
            selfDestructButton.enabled = false;
    }

    // Event handler for the "Play Again!" button.
    private function playAgainHandler():void
    {

        // Call resume() to make sure effect is not

        // in paused state before it ends or calling
        // pause() on it later will not work.
        flyChopper.resume();

        // End the effect
        flyChopper.end();

        // Reset assets to their original states.
        helicopter.visible = true;
        explosion.visible = false;
        startButton.enabled = pauseButton.enabled =
            resumeButton.enabled = selfDestructButton.enabled = true;
    }

    private function pauseChopperHandler():void
    {
        // Pause the Move effect on the helicopter.
        flyChopper.pause();

        // Calculate player's score based on the inverse of the

        // distance between the helicopter and the dart board.
        score.text = String(Math.round(1/(helicopter.x -
dartBoard.x)*10000));

        pauseButton.enabled = false;
        resumeButton.enabled = true;
    }

    private function resumeChopperHandler():void
    {
        flyChopper.resume();
        resumeButton.enabled = false; pauseButton.enabled = true;
    }

    ]]>
</mx:Script>

<!-- Create a Move effect with a random duration between .5 and 1.5 seconds
-->
<mx:Move
    id="flyChopper" target="{helicopter}"
    xBy="-290" easingFunction="mx.effects.easing.Quadratic.easeIn"

    duration="{Math.round(Math.random()*1500+500)}"
    effectEnd="endEffectHandler();"
/>
```

```
<mx:Panel
    title="Effects Chopper Game" width="100%" height="100%"
    paddingTop="10" paddingLeft="10" paddingRight="10"
    paddingBottom="10" horizontalAlign="right"
>

<!-- The game canvas -->
<mx:Canvas width="100%" height="100%">

    <mx:Image
        id="dartBoard" width="100" height="150.2"
        source="{DartBoard}" x="10" y="20"

    />

    <!-- Hide the explosion animation initially. -->
    <mx:Image
        id="explosion" source="{Explosion}"
        y="50" x="0" added="explosion.visible = false;"

    />

    <mx:Image
        id="helicopter" width="80" height="48.5"
        source="{Helicopter}" right="0" y="67"

    />

</mx:Canvas>

<!-- Instructions -->
<mx:Text
    width="100%" color="#ff0000"
    text="Pause the helicopter as close as possible to the dartboard
without hitting it."
    textAlign="center" fontWeight="bold"

/>

<mx:HBox>
    <mx:Label text="Score:" fontSize="18"/>
    <mx:Label id="score" text="0" fontSize="18"/>
</mx:HBox>

<mx:ControlBar horizontalAlign="right">
    <mx:Button
        id="startButton" label="Start"
        click="flyChopper.play(); startButton.enabled=false;"

    />

    <mx:Button id="pauseButton" label="Pause"
click="pauseChopperHandler();" />

    <mx:Button id="resumeButton" label="Resume"
click="resumeChopperHandler();" />
</mx:ControlBar>
</mx:Panel>
```

闲云无衣 <http://www.blogjava.net/xvridan>

```
<mx:Button
    id="selfDestructButton" label="Self destruct!"
    click="flyChopper.resume(); flyChopper.end();"

/>
<mx:Button label="Play again!" click="playAgainHandler();" />

</mx:ControlBar>

</mx:Panel>
</mx:Application>
```

提示: 如果调用某个效果的 `end()` 方法时, 该效果被暂停, 则当您再次显示相同的效果时, 调用其 `pause()` 方法将不会有效果。若要解决此问题, 请在调用其 `end()` 方法之前调用效果上的 `resume()` 方法。换句话说, 在首先继续暂停的效果之前, 不要结束它。

结果



创建状态

在许多 Rich Internet Application 中, 界面会根据用户正在执行的任务而变化。 当用户在图像上滚动鼠标时图像会发生变化, 这样的图像就是一个简单的示例。 许多复杂的示例包括其内容会根据用户在某个任务中的进度而变化的用户界面, 如从浏览视图更改到详细信息视图。 这些界面可以使用平滑的打开-关闭效果在视图之间过渡。

视图状态使您可以很容易地实施这样的行为, 并可以简化在其他方面复杂的事件处理代码的内容。

若要了解如何定义视图状态之间的过渡, 请参阅定义状态过渡。

简单地说, 视图状态定义组件的某个特定视图。 例如, 产品缩略图可以有两个视图状态: 包含次要信息的基本状态和包含附加信息的富状态。 相似地, 应用程序可以有与不同应用程序状况相对应的多个视图状态, 如登录状态、概述状态或搜索结果状态。

下面的示例使用视图状态很容易地实现登录和注册表单。 在此示例中, 初始视图状态提示用户登录, 并会根据需要包含让他们注册的链接。 如果用户选择“需要注册”链接, 则该表单会改变视图状态以显示注册信息。 当用户单击“返回登录”链接时, 视图状态会变回到登录表单。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" verticalAlign="middle"
    width="340" height="250"
>
    <!-- The states property of the Application class
         defines the view states. -->
    <mx:states>

        <!--
            The "Register" state is based on the base state.
            All states are based on the base state by default
            so you can leave out the basedOn property.
        -->
        <mx:State name="Register" basedOn="">

            <!-- Add a FormItem component to the form. -->
            <mx:AddChild
                relativeTo="{loginForm}"
                position="lastChild"
                creationPolicy="all"
            >

                <mx:FormItem id="confirm" label="Confirm:">
                    <mx:TextInput/>
                </mx:FormItem>
            </mx:AddChild>

            <!-- Set properties on the Panel container and Button control. -->
            <mx:SetProperty target="{loginPanel}"
                name="title" value="Register"/>
        </mx:State>
    </mx:states>
</mx:Application>
```

```
<mx:SetProperty target="{loginButton}"
name="label" value="Register"/>

<!-- Remove the existing LinkButton control. -->

<mx:RemoveChild target="{registerLink}"/>

<!--
    Add a new LinkButton control to change
    view state back to the login form.
-->
<mx:AddChild relativeTo="{spacer1}" position="before">

    <mx:LinkButton label="Return to Login" click="currentState=''"/>
</mx:AddChild>

</mx:State>

</mx:states>

<mx:Panel
    title="Login" id="loginPanel"
    horizontalScrollPolicy="off" verticalScrollPolicy="off"
>

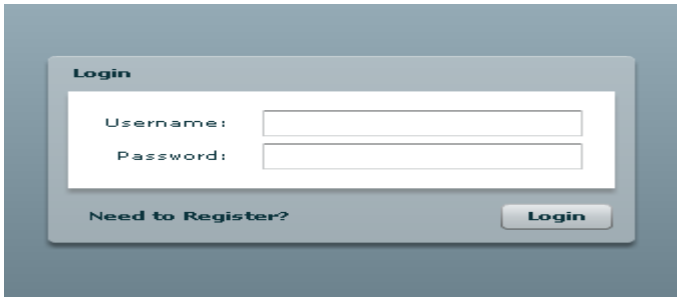
<mx:Form id="loginForm">
    <mx:FormItem label="Username:">
        <mx:TextInput/>
    </mx:FormItem>
    <mx:FormItem label="Password:">
        <mx:TextInput/>
    </mx:FormItem>
</mx:Form>

<mx:ControlBar>
    <!--
        Use the LinkButton control to change to
        the Register view state.
    -->
    <mx:LinkButton
        label="Need to Register?" id="registerLink"
        click="currentState='Register'"
    />

    <mx:Spacer width="100%" id="spacer1"/>
    <mx:Button label="Login" id="loginButton"/>
</mx:ControlBar>
</mx:Panel>
</mx:Application>
```

结果

[闲云无衣 http://www.blogjava.net/xvridan](http://www.blogjava.net/xvridan)



Login

Username:

Password:

[Need to Register?](#)

定义状态过渡

通常在响应用户操作时,视图状态使您可以改变应用程序的内容和外观。改变视图状态时,Adobe® Flex® 会同时执行对应用程序的所有可视更改。由于对视图状态的所有更改会同时发生,用户会看到应用程序从一种状态跳到另一种状态。

而您可能希望定义一个从一种状态到下一种状态的平滑的可视更改,在其中更改是在一段时间上发生的。过渡定义如何使对视图状态的更改看起来像是在屏幕上发生的一样。过渡是当视图状态更改发生时播放的组合到一起的一种或多种效果。

过渡不会替换效果;即,您仍可以将单一效果应用到一个组件,并通过使用一个效果触发器或者 `playEffect()` 方法来调用该效果。

您使用 `<mx:Transition>` 标签来创建过渡并自定义它,方法是通过使用其 `fromState`、`toState` 和 `effect` 属性。`fromState` 属性指定当应用该过渡时您要更改的视图状态,`toState` 属性指定您要更改为的视图状态,而 `effect` 属性是对要播放的 `Effect` 对象的引用。

在过渡中,可以通过使用 `<mx:Parallel>` 和 `<mx:Sequence>` 标签引发并行或按顺序播放的效果。

在下面的示例中,您定义了一个任何时候状态发生更改都会使用的过渡。此过渡是由一种并行效果组成的。并行效果也是一种复合效果,因为它包含其他效果。在此示例中,并行效果包含一个调整大小效果和一个顺序效果。并行效果的子效果全部在同时运行。

执行调整大小效果会花费 500 毫秒,且该效果使用一种简易的功能使得在调整大小时该面板会弹出。顺序效果也是一种复合效果。与并行效果不同,顺序效果的子事件按它们被添加的顺序,一次运行一个。下面的示例中的顺序效果包含两个模糊效果。模糊效果会模糊其目标组件并可以用于创建速度感或表示焦点的增益或损失。

示例

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" verticalAlign="middle"
    width="340" height="250"
    viewSourceURL="src/DefiningStateTransitions/index.html"
>
    <mx:Script>
        <![CDATA[
            // You need to import easing classes even if
            // you're going to use them only in MXML.
            import mx.effects.easing.Bounce;
        ]]>
    </mx:Script>

    <!--
        Use the transitions property (array) of
        the Application class to store your transitions.
    -->
    <mx:transitions>
        <!--
            The "*" indicates that the transition should be applied
            to any changes in the view state. You can set either
```

```
property to "" to refer to the base view state.
-->
<mx:Transition fromState="*" toState="*">
    <!-- Parallel effects execute in unison -->
    <mx:Parallel targets="{loginPanel, registerLink, loginButton,
confirm]}">
        <mx:Resize duration="500" easingFunction="Bounce.easeOut"/>
        <!--
            Sequence effects execute in turn. The effects
            in this sequence will only affect the confirm FormItem.
        -->
        <mx:Sequence target="{confirm}">
            <mx:Blur duration="200" blurYFrom="1.0" blurYTo="20.0" />
            <mx:Blur duration="200" blurYFrom="20.0" blurYTo="1" />
        </mx:Sequence>
    </mx:Parallel>
</mx:Transition>
</mx:transitions>

<!-- The states property of the Application class
defines the view states. -->
<mx:states>
<!--
    The "Register" state is based on the base state.
    All states are based on the base state by default
    so you can leave out the basedOn property.
-->
<mx:State name="Register" basedOn="">

    <!-- Add a FormItem component to the form. -->

    <mx:AddChild
        relativeTo="{loginForm}"
        position="lastChild"
        creationPolicy="all"
    >
        <mx:FormItem id="confirm" label="Confirm:">

            <mx:TextInput/>
        </mx:FormItem>
    </mx:AddChild>

    <!-- Set properties on the Panel container and Button control. -->
    <mx:SetProperty target="{loginPanel}"
name="title" value="Register"/>

    <mx:SetProperty target="{loginButton}"
name="label" value="Register"/>

    <!-- Remove the existing LinkButton control. -->

    <mx:RemoveChild target="{registerLink}"/>

    <!--
        Add a new LinkButton control to change
        view state back to the login form.
    -->
    <mx:AddChild relativeTo="{spacer1}" position="before">
```


[闲云无衣 http://www.blogjava.net/xvridan](http://www.blogjava.net/xvridan)

```
<mx:LinkButton label="Return to Login" click="currentState=''"/>
</mx:AddChild>

</mx:State>

</mx:states>

<mx:Panel
    title="Login" id="loginPanel"
    horizontalScrollPolicy="off" verticalScrollPolicy="off"
>

<mx:Form id="loginForm">
    <mx:FormItem label="Username:">
        <mx:TextInput/>

    </mx:FormItem>
    <mx:FormItem label="Password:">
        <mx:TextInput/>
    </mx:FormItem>
</mx:Form>

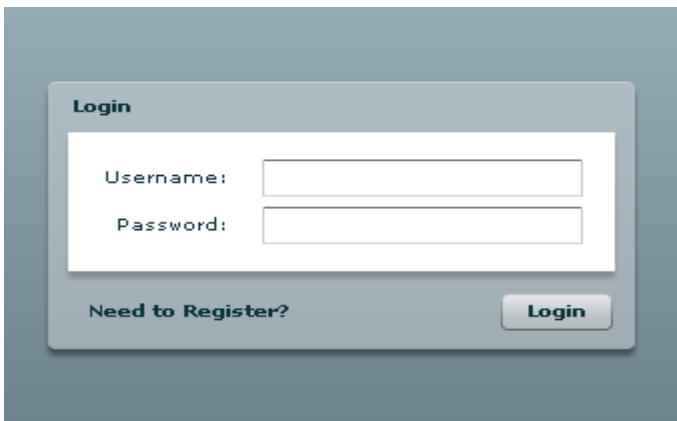
<mx:ControlBar>
    <!--
        Use the LinkButton control to change to
        the Register view state.
    -->
    <mx:LinkButton
        label="Need to Register?" id="registerLink"
        click="currentState='Register'"
    />

    <mx:Spacer width="100%" id="spacer1"/>
    <mx:Button label="Login" id="loginButton"/>

</mx:ControlBar>

</mx:Panel>
</mx:Application>
```

结果



使用工具提示

Adobe® Flex™ ToolTip 使您能够为您的用户提供有帮助的信息。当用户在图形组件上移动鼠标指针时, 会弹出包含文本信息的工具提示。您可以使用工具提示来指导用户完成使用应用程序或自定义它们来提供其他功能。

扩展 `UIComponent` 类 (该类实现 `IToolTipManagerClient` 界面) 的每个可视 Flex 组件都支持 `toolTip` 属性。您将 `toolTip` 属性的值设置为一个文本字符串, 并且, 当鼠标指针悬停在该组件上时, 会显示该文本字符串。

尽管长消息很难读取, 但对工具提示文本的大小不存在任何限制。当工具提示文本达到工具提示框的宽度时, 文本会自动换至下一行。可以在工具提示文本中添加换行符。在 `ActionScript` 中, 您使用 `\n` 转义的新行字符。在 `MXML` 标签中, 您使用 `` XML 实体。

可以通过使用层叠样式表 (CSS) 语法或 `mx.styles.StyleManager` 类更改工具提示文本和工具提示框的外观。对工具提示样式的更改适用于当前应用程序中的所有工具提示。

示例

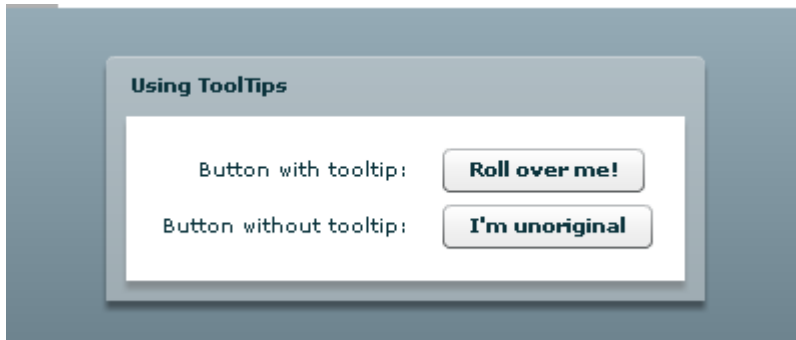
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/UsingTooltips/index.html"
    width="400" height="170"
>
    <mx:Style>
        ToolTip
        {
            fontFamily: "Arial";
            fontSize: 19;
            fontStyle: "italic";
            color: #FFFFFF;
            backgroundColor: #33CC99;
        }
    </mx:Style>

    <mx:Panel
        title="Using ToolTips"
        toolTip="Child components without &#13; ToolTips will inherit this."
    >
        <mx:Form>
            <mx:FormItem label="Button with tooltip:">
                <mx:Button label="Roll over me!" toolTip="You can click me too!"
            />
            </mx:FormItem>

            <mx:FormItem label="Button without tooltip:">
                <mx:Button label="I'm unoriginal" />
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>

</mx:Application>
```

结果



控制光标

使用 Adobe® Flex™ 光标管理器可以控制 Flex 应用程序中的光标图像。例如, 如果应用程序执行的处理需要用户等待, 直到处理完成为止, 则可以将光标更改为某个自定义的光标图像, 比如沙漏, 以使它反映该等待期。

您还可以更改光标以向用户提供反馈, 指示用户可以执行的操作。例如, 您可以使用一个光标图像来指示用户输入被启用, 而使用另一个光标图像来指示输入被禁用。

`CursorManager` 类控制一个光标优先顺序列表, 在其中具有最高优先级的光标当前是可见的。如果光标列表包含具有相同优先级的多个光标, 则光标管理器会显示最近创建的光标。

使用默认的忙光标

Flex 定义了一个默认的忙光标, 可用来指示应用程序正在处理, 且在应用程序对用户输入作出响应之前, 用户应等待, 直到处理完成。默认的忙光标是一个动画时钟。

可以使用以下几种方式来控制忙光标:

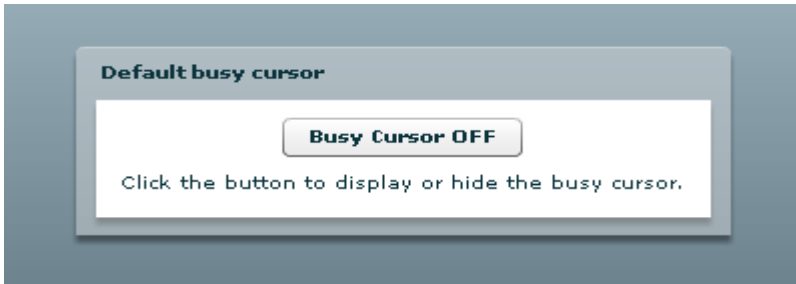
- 可以使用 `CursorManager` 方法来设置和删除忙光标。
- 可以使用 `SWFLoader`、`WebService`、`HttpService` 和 `RemoteObject` 类的 `showBusyCursor` 属性自动显示忙光标。

下面的示例使用 `CursorManager` 类的静态 `setBusyCursor()` 和 `removeBusyCursor()` 方法, 根据切换按钮的状态显示和隐藏默认的 Flex 忙光标。

示例

```
<?xml version="1.0" encoding="utf-8"?> <mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
viewSourceURL="src/CursorDefaultBusy/index.html" width="400" height="160" >
<mx:Script> <![CDATA[ import mx.controls.Button; import
mx.managers.CursorManager; import flash.events.*; private const
ON_MESSAGE:String = "Busy Cursor ON"; private const OFF_MESSAGE:String = "Busy
Cursor OFF"; private function busyCursorButtonHandler(event:MouseEvent):void
{ var toggleButton:Button = event.target as Button; if (toggleButton.selected) {
CursorManager.setBusyCursor(); toggleButton.label = ON_MESSAGE; } else
{ CursorManager.removeBusyCursor(); toggleButton.label = OFF_MESSAGE; } } ]]>
</mx:Script> <mx:Panel paddingBottom="10" paddingTop="10" paddingLeft="10"
paddingRight="10" horizontalAlign="center" verticalAlign="middle" title="Default
busy cursor" > <!--Toggle button turns default busy cursor on and off. -->
<mx:Button label="{OFF_MESSAGE}" toggle="true"
click="busyCursorButtonHandler(event);" /> <mx:Text text="Click the button to
display or hide the busy cursor."/> </mx:Panel> </mx:Application>
```

结果



使用自定义光标

可以使用 JPEG、GIF、PNG 或 SVG 图像, Sprite 对象或 SWF 文件作为光标图像。

若要使用光标管理器, 您将 `mx.managers.CursorManager` 类导入到应用程序中, 然后引用其属性和方法。

下面的示例嵌入一个在 Adobe Flash 中创建的沙漏的 SWF 动画, 并将它用作一个自定义光标。

在该示例中, 创建自定义光标的方法是, 调用 `CursorManager` 类的 `setCursor()` 静态方法, 然后将它传送给对您希望用作自定义光标的嵌入资源的类的引用。 可以通过调用 `CursorManager` 类的 `removeCursor()` 静态方法并将它传送给 `CursorManager` 类的 `currentCursorID` 静态属性来删除活动的自定义光标。

示例

```
<?xml version="1.0" encoding="utf-8"?> <mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
viewSourceURL="src/CursorCustom/index.html" width="400" height="160" >
<mx:Script> <![CDATA[ import mx.controls.Button; import
mx.managers.CursorManager; import flash.events.*; // Embed the SWF that will be
used as // the custom cursor. [Embed(source="assets/hourglass.swf")] public
var HourGlassAnimation:Class; private const ON_MESSAGE:String = "Custom Cursor
ON"; private const OFF_MESSAGE:String = "Custom Cursor OFF"; private function
busyCursorButtonHandler(event:MouseEvent):void { var toggleButton:Button =
event.target as Button; if (toggleButton.selected) { // The setCursor() method
returns a numeric ID for // the cursor being set. You can store and use this
// ID later in a removeCursor() call, or, you can // use the static
currentCursorID property of the // CursorManager class to achieve the same
result. CursorManager.setCursor(HourGlassAnimation); toggleButton.label =
ON_MESSAGE; } else { CursorManager.removeCursor(CursorManager.currentCursorID);
toggleButton.label = OFF_MESSAGE; } } ]]> </mx:Script> <mx:Panel
paddingBottom="10" paddingTop="10" paddingLeft="10" paddingRight="10"
horizontalAlign="center" verticalAlign="middle" title="自定义光标" > <!--Toggle
button turns the custom cursor on and off.--> <mx:Button label="{OFF_MESSAGE}"
toggle="true" click="busyCursorButtonHandler(event);" /> <mx:Text text="Click
the button to display or hide the custom cursor."/> </mx:Panel>
</mx:Application>
```

结果

闲云无衣 <http://www.blogjava.net/xvridan>

