

主要讲 3 个主题:

- 1.Flex 是什么, 用途和前景;
- 2.Flex Builder 基础;
- 3.MXML 基础;

现在进入第 1 个主题:

首先我们要明白, flex 不是专门用来做网页的, 它主要是用来做 RIAs 的

RIAs = Rich Internet Applications

富英特网应用程序

Flex 现在虽然是 2.0, 以前是 1.0 和 1.5, 但是不要担心 1.0 和 1.5 版本自己没有用过, 因为 2.0 版本才是 Flex 真正的开始, 首先, Flex 2.0 的技术基础很高, 采用 Action Script 3.0 为编程核心, 以 FlashPlayer 9.0 为平台

它与以前版本的区别是

FlashPlayer 9.0 采用全新的虚拟机, 运行速度是 8.0 的 10 倍以上

Flash 8.0 和 Flex 1.0, 1.5, 都是 Action Script 2.0 用的 FlashPlayer 8.0, 在性能上, 跟 Flex 2.0 的 Action Script 3.0 和 FlashPlayer 9.0 相差太远了

我们现在处于 RIAs 浏览器时代, 意思是, 我们用 Flex 做的程序, 必须要用浏览器打开, 以网页的形式发布, 明年, 我们将步入下一个 RIAs 时代——桌面 RIAs 时代, 我们现在所有用 flex 2.0 做的程序, 明年都可以被重新发布成桌面应用程序, 就跟我们现在用的软件是一样了, 到时候, 我们的程序不再被浏览器限制了, flex 能做的程序很多, 比如 QQ 这样的即时聊天软件、论坛、股票软件、网络视频聊天、等等。你能想到的基本都能做出来, Flex 的前景是一片明朗的, 而且越往后越会被广泛采用, 这一切都从 Flex 2.0 开始, 所以, 对于我们来说, 这是个前所未有的好机会。现在学 flex 的人很少很少, 如果你等到他已经大面积普及, 那就没有竞争力了。

好, 现在提问时间, 马上要进入下一主题:

Flex 是纯面向对象语言, C 是面向过程语言, ActionScript 3.0 相对 2.0, 改动太大了, 如果有 2.0 基础, 对 3.0 帮助不大, 所以, 如果你第一次接触的就是 3.0, 也不要紧, 3.0 删除了很多 2.0 的东西, 而且增加了很多新类, 3.0 跟 JAVA 基本没什么区别了, 所以, 有 JAVA 基础的人, 学起来更快。

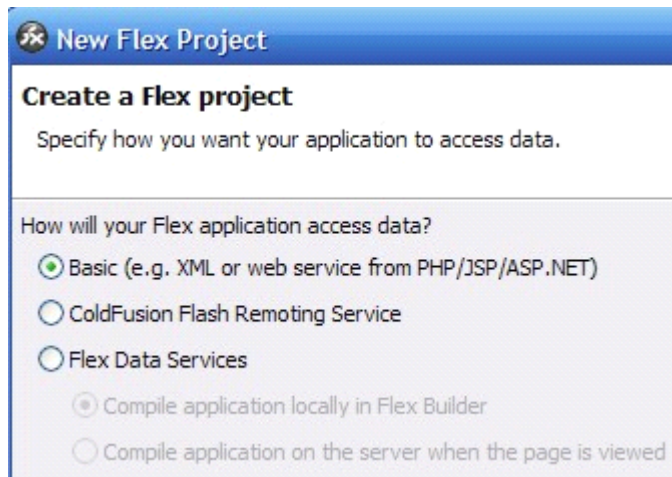
AS3 的语法跟 JAVA 一样, 有库、包、类、接口这些概念, 两者的本质区别是 java 可以做软件, 但局限性很大, 比如 java 界面很差, 很多功能比较古老等等

今天第 2 个主题——Flex Builder 2 基础

1、打开 Flex Builder 2

A. 新建 Flex Project





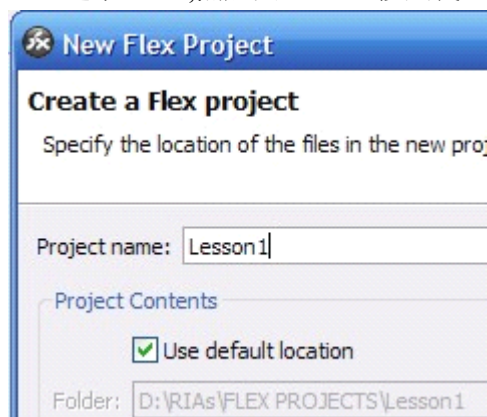
我来讲讲这 3 个选项的用途：

I) Basic 是基本 Flex 项目,你要获取数据的话,必须靠自己写外部的 PHP 或者 JSP 脚本来实现

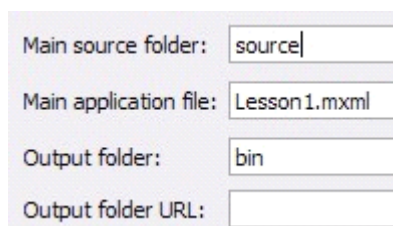
I I) Coldfusion 这个是用 Coldfusion 服务器来给 Flex 程序提供数据,一般都不用这个, Coldfusion 在中国没什么发展

I I I) Flex Data Service 则是 Flex 自己的一套完整的后台系统,他可以动态编译 MXML 文件成 SWF, Flex Data Service 这个以后再讲,现在大致了解一下即可

B. 选中 Basic,然后点 Next, 按照我这个图里来填

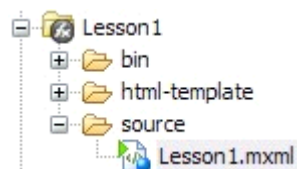


好了就点 Next



4 个输入栏：第一个是填你源代码目录，写 source；第 2 个是你主应用程序文件名，默认；第三个是你应用程序做好以后的输出目录；第 4 个是输出 url，不用写，一般不用它。做好了的应用程序会放到 bin 目录里。

填好以后按 finish 完成



解释一下这个目录结构，从上往下：

首先

Lesson1 是你项目的名称

bin 里面专门放你做好的程序文件

html-template 是放 html 模板的

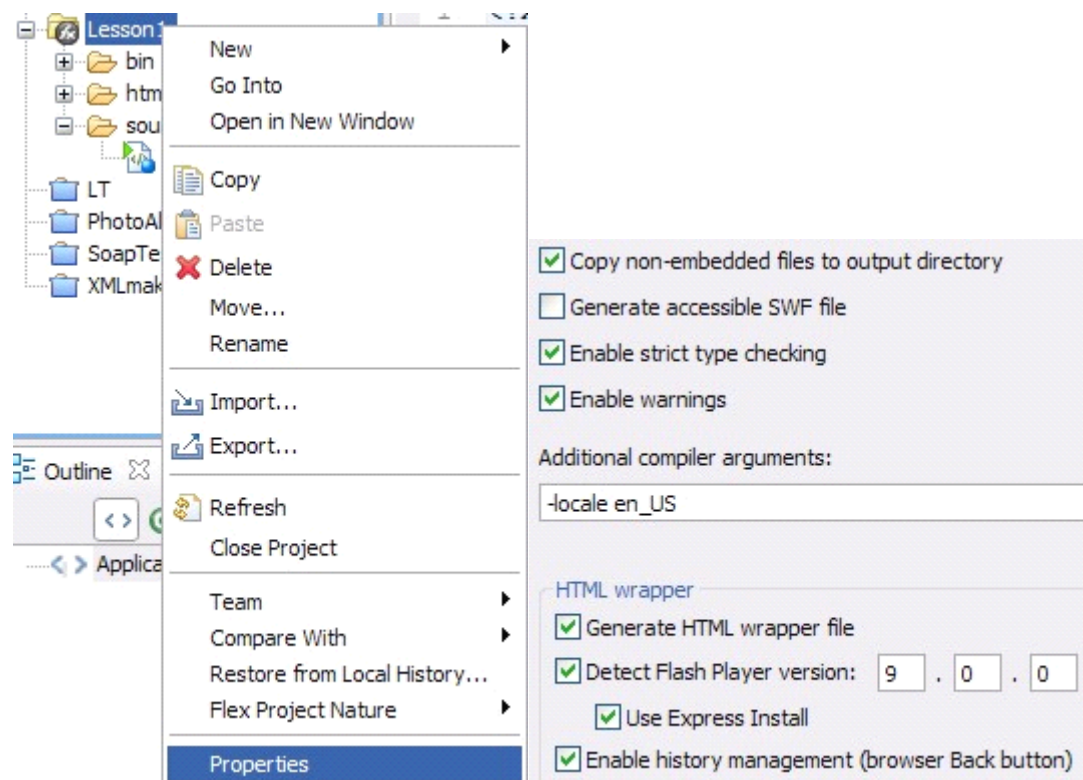
source 专门放源代码

这里我解释一下 html-template 这个目录里东西的用途，这个目录放的全部是 html 模板和 js 脚本，他们的作用是把生成的应用程序 SWF 文件包裹在一个 HTML 文件中，然后用户只要访问这个 HTML 文件,就等于访问了应用程序 SWF 文件。

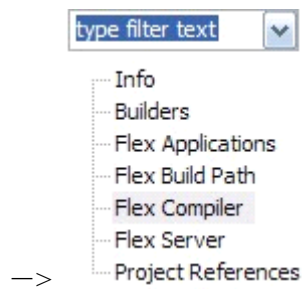
html-template 包含了重要功能：

1. 它里面的代码能自动检测访问者的 Flashplayer 版本,如果版本太低,则自动提示安装
2. 能让 swf 控件免激活。一般的网页里的 swf 都要点一下之后才能访问,也就是要激活
3. 历史记忆功能,可以让你的 Flex 应用程序拥有浏览器中的后退和前进功能。这个是 Flash 做的程序里一直没有的

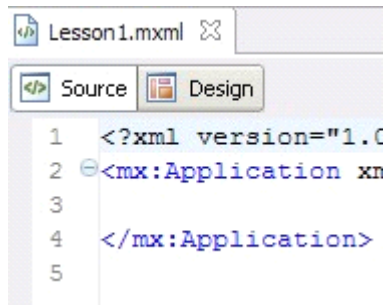
好，现在在 Lesson1 的图标上点右键，选择最下面一项—Properties，点 Flex Compiler



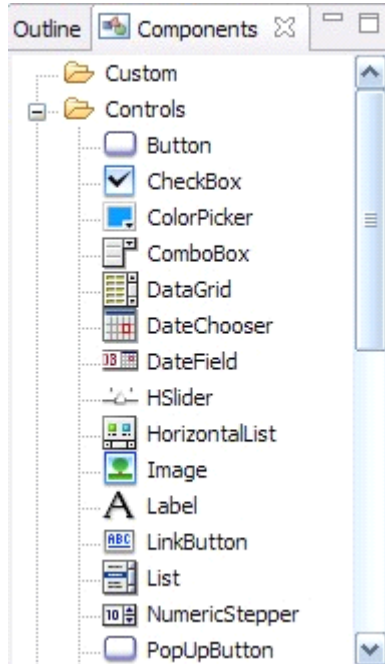
我简单说一下这里的选项作用：看到 9.0.0 吗，这是自动检测 flashplayer 版本的（现在最新版本是 9.0.28）大家可以改成 9.0.28，这样用户访问的时候就会提示更新 flashplayer 版本。注意，版本一定不要低于 9.0.0，因为 flashplayer 8.0 是无法也不可能运行 Flex 程序的。



好了，现在讲主界面



在主要页面，分 Source 和 Design，Source 是专门写代码的，Design 是视图面板，你写的代码可以在 Design 模式下看到实体，现在打开 Design，好，我们看到现在舞台上什么都没有，请看左边 Components 面板，我们可以把 Components,也就是组件面板里的东西拖到舞台上



Components 面板中：

Custom 是放你自定义组件的；

Controls 是放控件的，用户可以直接与之交互；

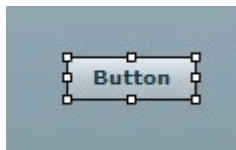
底下还有 Layout，它是放容器的，简单讲,我们应该把所有控件都放到合适的容器里，否则程序做出来就杂乱无章；

Navigators 里面放的是导航组件，比如菜单组件等；

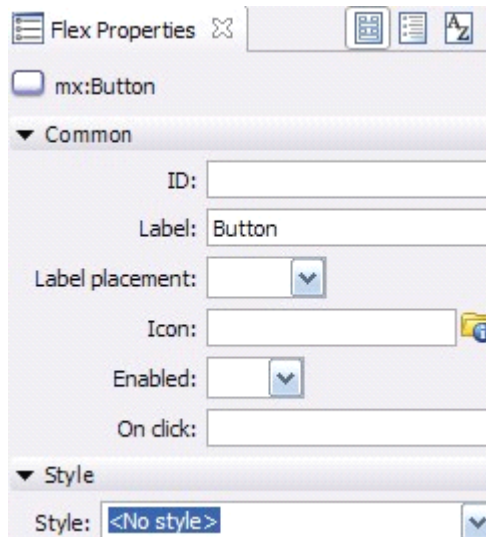
Charts 里面是图表,用他可以做出很多漂亮的统计图；

现在我们先看 Controls:

先拖个 Button 组件到舞台，鼠标直接拖到中间的舞台里



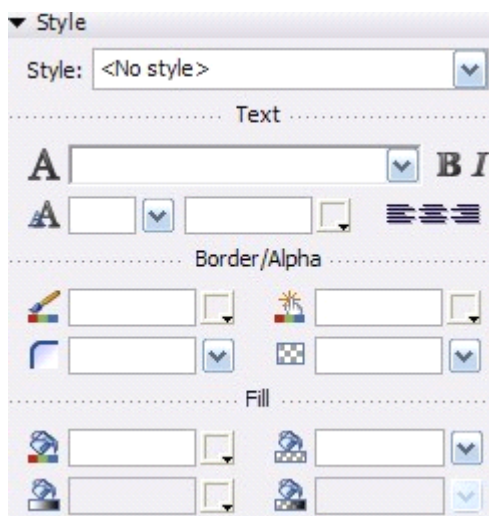
好,现在看右边的 Flex Properties 面板



可以看到这个 Button 组件的常用属性都在上面了, ID 里写这个 Button 组件的引用名称, Label 是他的显示标签, 大家可以把 Label 随便改成其他名字, 你们会看到 Label 更改以后, 舞台上的 Button 标签马上变化了, Icon 是用来改 Button 图标的, Enabled 是指 是否允许使用 Button 组件, 如果写成 false, 组改 Button 组件不可被用户点击

(问: 有个问题, label placement 属性里, 我的理解是选择 label 的放在按钮里的排版靠左等等, 不过我选了没有什么区别。答: 哦, 只有在你先定义了 icon 图标以后, label placement 才会生效)

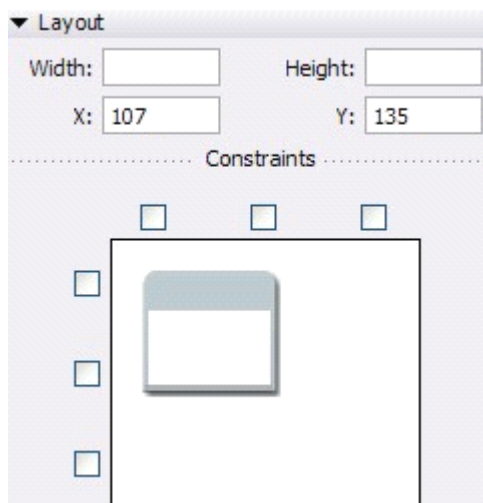
这里是改 Button 的常用样式



比如字体, 字号, 颜色, 透明度, 边线宽度, 背景色等等

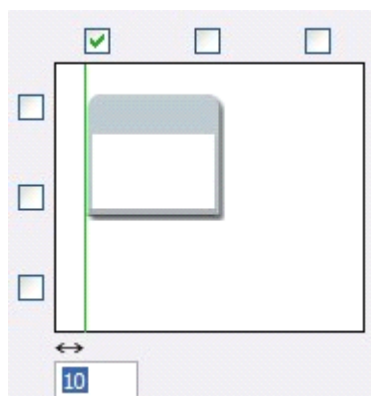
当我们更改这个面板里任何一个属性的时候, 我们观察一下 Source 模式, 可以看到, Source

模式下，代码立即被更新



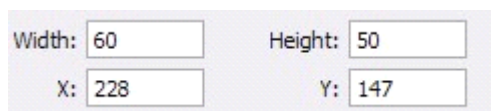
这里面是调整 button 的宽和高，x,y 坐标，以及布局约束（哦，对了，布局约束，永远只能是控件相对于容器边缘，而不能是控件相对于控件边缘）。

这里我简单教大家用布局约束 Constraints，请看图：



如图所示，这样设置的话，Button 组件就会永远相对与舞台边缘距离 10 像素位置。不论用户怎么更改浏览器大小，它都会保持与左边 10 像素距离，如果是这样的话，Button 会一直保持在屏幕左下角的位置距离下边和左边 10 像素

好,现在我来讲讲相对和绝对：

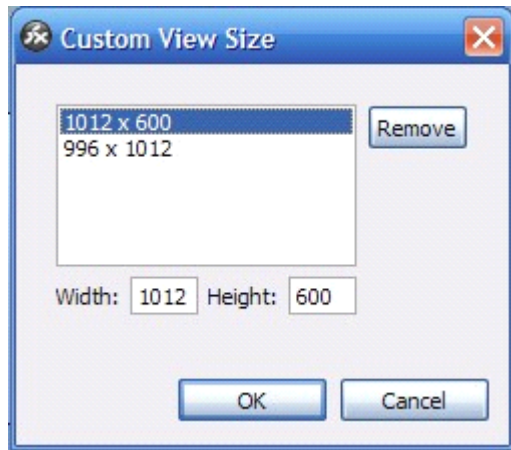


这样的设置,button 宽 60 高 50，这样的数值是绝对的，也就是说,你屏幕不论分辨率多的多大，宽高依然是 60 和 50，很多时候这满足不了我们的需求，所以我们有时把它设置相对值：



改成百分号的，把宽和高改成百分号以后,大小就相对于屏幕的 x%，这样就有很大的可伸缩性。一般对于按钮不需要，主要是对于容器很有用

现在点这里 ，这个很重要



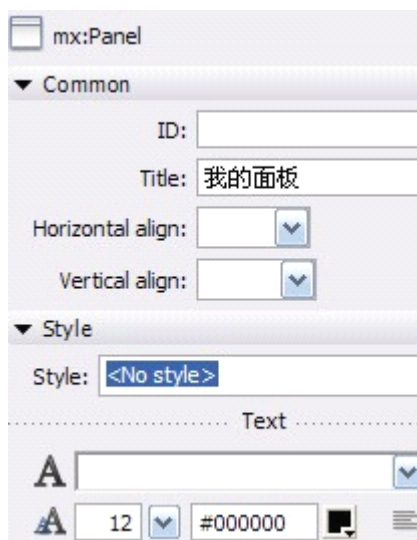
请把里面默认数值删除掉，点 Remove 擅长删除，然后添加我图上的这 2 对数值。1012*600 是在 1024*768 下,IE 7.0 中的最大尺寸，我们以后做程序，都要以这个尺寸为主

（问：不设置不可以吗？答：必须设置，不设置的话,你所有的组件会全部挤到一堆,而且输出结果给你预想的会完全不一样，1012*600 最好）

好，现在请把 button 的宽高删除掉，变成默认大小，现在我们从 layout 中拖个容器出来，拖个 panel 吧，我们把 Button 拖到 Panel 里放着先

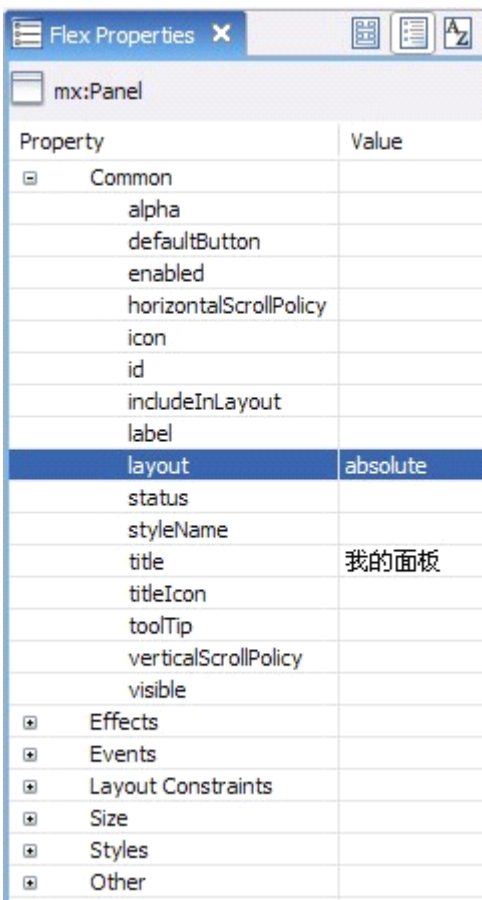


我们把 Panel 简单设置一下



再看看 Design 下的效果

注意，在这里提供的选项并不是一个组件全部的选项，全部选项在这里 Catagory View 里



Common 里是常用选项；Effects 里是效果；Events 里是事件；还有，比较重要的是 Styles 里，Styles 里是一个组件的所有样式：

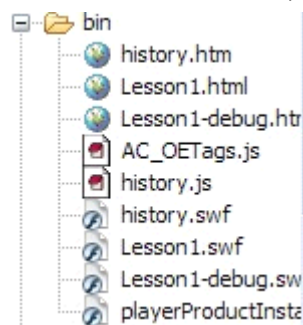
Styles	
backgroundAlpha	
backgroundAttachmer	
backgroundColor	<div><div></div></div> #FFFF80
backgroundDisabledC	
backgroundImage	
backgroundSize	
barColor	
borderAlpha	
borderColor	<div><div></div></div> #FF8080
borderSides	
borderSkin	
borderStyle	
borderThickness	
borderThicknessBotto	5
borderThicknessLeft	5
borderThicknessRight	5
borderThicknessTop	
closeButtonDisabledSl	
closeButtonDownSkin	
closeButtonOverSkin	
closeButtonUpSkin	
color	<div><div></div></div> #000000

可以看看,Style 里改了这些设置以后,Panel 面板样式发生了很多变化:



好，现在看看怎么编译程序：

每次点下保存按钮以后,不仅代码全部被保存,而且直接被编译成 swf 了，不需要手动编译



刚刚保存以后，swf 文件就已经被输出到 bin 目录里了

history.htm + history.swf + history.js 负责程序的后退前进功能

playerProductInstall.swf 负责在线安装 flashplayer

Lesson1.html + Lesson1.swf 是你主程序 最重要的部件

Lesson1-debug.html + Lesson1-debug.swf 是调式程序文件

AC_OETags.js 负责让你的 swf 在网页里显示的时候免激活

当你发布一个程序的时候，最少你得要 Lesson1.swf。

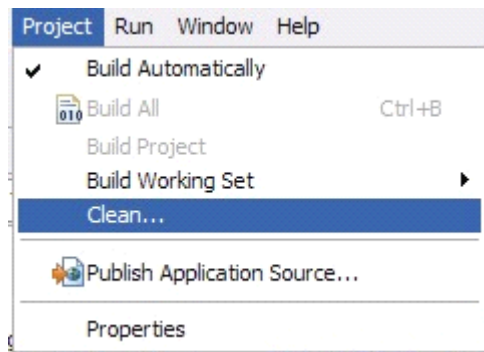
一般的情况下，除了 Lesson1-debug.html + Lesson1-debug.swf 不要以外，其他全部都需要。也就是把 bin 中除了 Lesson1-debug.html + Lesson1-debug.swf 以外，其他所有文件都上传到你的网站某个文件夹里。

好，要测试你的程序：

有很多方法，比如标准方法是点这个按钮

点了之后，浏览器自动打开，即可测试

再告诉大家一个实用的：Project---Clean

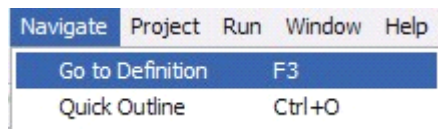


有时候，你导入的外部文件可能会识别错误，这个时候就点 Project---Clean，它非常有用，可以理顺代码和不正确的连接，同时保存和编译程序

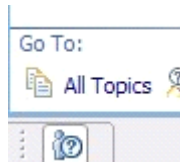
好，再告诉大家 2 个东西就进入下一主题了

```
<mx:Panel x=
    <mx:Butt
```

如图</mx:Panel>，好现在进 source 模式，把光标放到 Panel 上面，这个时候,点这个



(或者直接按 F3)，这样可以看到 Panel 的源代码，Flex 大部分源代码都是公开的。不过，不要试图修改这些代码，flex 会产生错误，这些代表只帮助学习代码，接着，还是把光标放 panel 上点左下角的帮助按钮



这里会出现关于 panel 的动态帮助，非常有用，查语言参考和教程

好了，现在进入今天最后一个主题：MXML 基础
从这里开始就是关键了

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Panel x="74" y="148" width="250" height="200" layout="absolute" title="panel">
        <mx:Button x="10" y="23" label="Button"/>
    </mx:Panel>
</mx:Application>
```

我来简化一下

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Panel title="panel">
        <mx:Button label="Button"/>
    </mx:Panel>
```

</mx:Application>

好，一行一行的说：

1. <?xml version="1.0" encoding="utf-8"?> 是文档类型定义
MXML 也是 xml，所以它同样需要这个定义，这句代表 xml 版本是 1.0，以 utf-8 编码
2. <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
在每个 Flex 应用程序当中 mx:Application 都是顶级容器，一个 MXML 文件里，只能有一个 <mx:Application> 标签
3. xmlns:mx="http://www.adobe.com/2006/mxml" 这个是命名空间，预习了 xml 的人应该都看得懂
4. layout="absolute" 这代表目前是绝对布局，只有在绝对布局下才能用 布局约束

<mx:Panel title="panel">

<mx:Button label="Button"/>

</mx:Panel>

<mx:Application> 中可以有 很多这样的子标签；可以有子容器，子控件等等。现在，Button 就是 Panel 的子控件，Panel 则是 Application 的子容器，每个子组件都可以有 1 个 id 属性，id 用来引用这个组件，这个 id 用来找到并调用这个组件。但是注意，**mx:Application 是不可以有 id 的**，如果你想引用 mx:Application，你就得用 **this** 关键字，this 永远指一个 MXML 的顶级类，在这里 this 就指 mx:Application，因为我们这个文件是 Lesson1.mxml，Lesson1 就是我们这个程序的一个主类。**this 很简单，永远指最上层的那个标签**，如果你新建一个 MXML Component，顶级标签是 HBox 的话，this 就指这个 HBox

先在教大家用几个组件：

请删除舞台里所有组件，拖一个 combobox 到舞台

<mx:ComboBox>

<mx:Object label="第 1 项" data="1"/>

<mx:Object label="第 2 项" data="2"/>

<mx:Object label="第 3 项" data="3"/>

<mx:Object label="第 4 项" data="4"/>

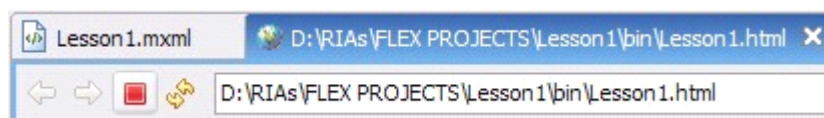
</mx:ComboBox>

在 <mx:ComboBox> 里面加入这些，然后保存，看右下角，如果进度走完了就代表保存完毕了，保存完毕以后，按那个绿色的运行按钮，可以看到 combobox 已经可以用了，可以看到 combobox 已经可以用了。

对了，用运行按钮测试是可以，但是反应比较慢

我告诉大家一个更快的方法：展开 bin，双击 Lesson1.html，这时 flex builder 内部浏览器马上打开了，这是最快的方法。保存的时候，右下角进度条走完了再测试

flex builder 内存浏览器：





现在在 ComboBox 上加 id 属性

```
<mx:ComboBox id="cb1" fontSize="12">
    <mx:Object label="第 1 项" data="1"/>
    <mx:Object label="第 2 项" data="2"/>
    <mx:Object label="第 3 项" data="3"/>
    <mx:Object label="第 4 项" data="4"/>
</mx:ComboBox>
```

接着，到 Design 模式下，拖个 Label 出来

```
<mx:ComboBox id="cb1" fontSize="12">
    <mx:Object label="第 1 项" data="1"/>
    <mx:Object label="第 2 项" data="2"/>
    <mx:Object label="第 3 项" data="3"/>
    <mx:Object label="第 4 项" data="4"/>
</mx:ComboBox>
<mx:Label text="{cb1.selectedItem.label}" y="50" fontSize="12"/>
```

我来解释一下花括号的意思：

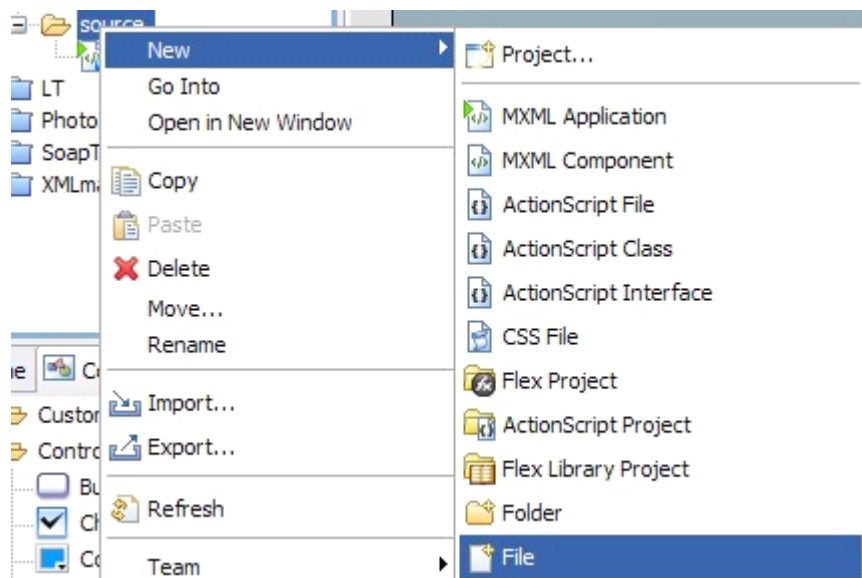
花括号是数据绑定，cb1 代表这个 ComboBox，cb1.selectedItem 代表这个 ComboBox 当前所选中的项目，cb1.selectedItem.label 代表这个 ComboBox 当前所选中的项目的标签

好，现在保存，然后用 flex builder 内部浏览器打开，注意，一定要等保存完。

（问：属性值什么时候要加花括号。答：只有在你需要数据绑定的时候）我想把 2 个值实时捆绑的时候用，然后用内部浏览器打开，看看效果是怎样的：

大家会发现，combobox 的标签跟 label 完全绑定到一起了，这就是 Flex 强大的数据绑定功能，冰山一角，你可以把外部 xml 文件中的数据，绑定到你的组件里。

绑定 xml 文件的例子：



在 source 上右键，new --- file ，然后点 Finish。
source 下新建了一个 data.xml，

打开 data.xml，输入

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

```
<root>
```

```
  <rec label="第 1 项目" data="1"/>
```

```
  <rec label="第 2 项目" data="2"/>
```

```
  <rec label="第 3 项目" data="3"/>
```

```
  <rec label="第 4 项目" data="4"/>
```

```
  <rec label="第 5 项目" data="5"/>
```

```
  <rec label="第 6 项目" data="6"/>
```

```
  <rec label="第 7 项目" data="7"/>
```

```
</root>
```

然后保存。注意，你们最后可能读取乱码，因为有时候会被保存成 gb2312 的，一定要用 utf-8 编码保存，推荐用 editplus，它可以保存成 utf-8 的。

进 lesson1.mxml，source 模式，注意，这次不能再直接拖东西了，要写代码了：

在 combobox 标签上方加入

```
<mx:HTTPService id="https1" url="data.xml" useProxy="false"/>
```

清空 combobox 里面的内容：

即

```
<mx:HTTPService id="https1" url="data.xml" useProxy="false"/>
<mx:ComboBox id="cb1" fontSize="12"/>
<mx:Label text="{cb1.selectedItem.label}" y="50" fontSize="12"/>
```

useProxy="false" 表示不用代理, url 里永远写数据源的地址, 不一定非要是 xml 结尾的, 可以是 wsdl, 也可以是 php, 任何可以返回 xml 的地址

好, 现在在<mx:HTTPService/>标签上方, 加入

```
<mx:Script>
    <![CDATA[
        代码只能写在这里
    ]]>
</mx:Script>
```

<mx:Script> 就是真正写 AS 3 代码的地方了,本来想下次课讲的,今天多讲 1 点算了

Flex 主要分 3 大途径获取数据:

1.RPC Service

2.Data Management Service

3.Messaging Service

2 和 3 都需要安装 Flex Data Service 服务器软件

RPC Service 分 3 个子 Service:

HTTPService

WebService

RemoteObject Service (需要 Flex Data Service)

我们用得最最多的就是 HTTPService 和 WebService 了, 只有他们 2 个不需要 Flex Data Service 支持。

Flex 其实是个多米诺, 他不能无缘无故的运行你的 as 3 代码, 它只能在发生某些事件的时候才能执行代码, 没有发生事件的时候是不可能执行代码的。所以, 我们要让 HTTPService 去主动获取外部 xml 数据源, 并且处理数据的话, 我们必须在一个事件下进行, flex 有 n 多事件, 比如最常用的 creationComplete 事件: 每当一个组件自我创建完毕以后, 就发出 creationComplete 事件, 只有在某个事件发出以后, 我们才能利用这个机会运行 AS 3 代码。基本每个可视组件都有 creationComplete 事件, Application 这个顶级容器也是可视的, 所以它也有 creationComplete 事件

(问: AS 3 代码是在客户端还是服务器端。答: 只有在你装有 Flex Data Service 的时候, AS 3 才有机会在服务器上运行; 否则, AS 3 永远运行在客户端)

as 3 基本都运行在客户端, 服务器端对机器要求高。服务器端对编程的水平要求很高

每个可视组件都有 creationComplete 事件

为了让我们执行 as 3 代码, 我们随便找个有 creationComplete 事件的组件, 比如 Application

```
<mx:Application creationComplete="init()" xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
```

大家看到, 我新加了一个 creationComplete="init()": 意思是, 当 Application 这个顶级容器创建完毕以后, 执行 init() 这个方法,

现在我们在 script 当中写 init() 这个方法:

```
<mx:Script>
    <![CDATA[
        private function init():void {
            https1.send();
        }
    ]]>
</mx:Script>
```

注意, private 代表私有, 不要写成 public 或者别的, 因为那样你的这个方法可以被别人调用我之前说过, 所有 swf 中的 public 代码都能被别人直接调用

这整个代码的意思是

1. 当 Application 这个顶级容器创建完毕以后, 执行 init() 这个方法
 2. 在 init() 这个方法中, 执行 https1.send(); 也就是发送 http 请求
- 这样 flex 程序一创建好, 就立即发送 http 请求给 data.xml, 并且把 data.xml 返回给 flex 程序

现在进入第 2 个事件:

也就是 httpservice 的 result 事件

```
<mx:HTTPService id="https1" result="chuli()" url="data.xml" useProxy="false"/>
result="chuli()"
```

意思是, 当 https1 发送 http 请求给 data.xml, 把 data.xml 数据带回 flex 以后, 就发生了 result 这个结果事件。

我们利用这个事件再执行一个 chuli() 方法, 处理得到的数据
在 script 当中, 加入以下代码

```
private function chuli():void {
    cb1.dataProvider = https1.lastResult.root.rec;
}
```

https1.lastResult, lastResult 是上一次 http 请求的结果, 也就是数据源(data.xml)
data.xml 内容

```
<?xml version="1.0" encoding="utf-8"?>
<root>
    <rec label="第 1 项目" data="1"/>
    <rec label="第 2 项目" data="2"/>
    <rec label="第 3 项目" data="3"/>
    <rec label="第 4 项目" data="4"/>
    <rec label="第 5 项目" data="5"/>
    <rec label="第 6 项目" data="6"/>
    <rec label="第 7 项目" data="7"/>
</root>
```

解释:

```
private function chuli():void {
    cb1.dataProvider = https1.lastResult.root.rec;
```

```
}
```

cb1.dataProvider, dataProvider 代表 cb1 的数据源, 这个数据源由 https1.lastResult.root.rec 提供, 即数据源是

```
<rec label="第 1 项目" data="1"/>
<rec label="第 2 项目" data="2"/>
<rec label="第 3 项目" data="3"/>
<rec label="第 4 项目" data="4"/>
<rec label="第 5 项目" data="5"/>
<rec label="第 6 项目" data="6"/>
<rec label="第 7 项目" data="7"/>
```

好, 现在保存

我把完整代码贴出来:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application creationComplete="init()" xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
    <mx:Script>
        <![CDATA[
            private function init():void {
                https1.send();
            }


            private function chuli():void {
                cb1.dataProvider = https1.lastResult.root.rec;
            }
        ]]>
    </mx:Script>
    <mx:HTTPService id="https1" result="chuli()" url="data.xml" useProxy="false"/>
    <mx:ComboBox id="cb1" fontSize="12"/>
    <mx:Label text="{cb1.selectedItem.label}" y="50" fontSize="12"/>
</mx:Application>
```

把这个全部复制到 lesson1.mxml 里, 覆盖全部原代码, 保存, 然后打开 lesson1.html 测试。如果发现乱码, 则是因为 data.xml 文件被保存成 gb2312 了

(问: 问个问题 as 的代码可以放在一个代码文件里面吗, 然后 mxml 文件导入, 如果可以请以这个例子讲讲。答: as 代码可以放到一个文件里, 但是不能动态导入)

(问: 要如何导入? 答: 在 source 目录下, 新建一个 Action Script File, 取名

myAS.as



```
。 private function init():void {
    https1.send();
}
```

```
private function chuli():void {
    cb1.dataProvider = https1.lastResult.root.rec;
}
```

把这些代码复制到 myAS.as 里面，保存。接着，把 <mx:script> 整个改成 <mx:Script source="myAS.as"/>，保存即可）

Lesson1.mxml 完整代码为

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application creationComplete="init()" xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
    <mx:Script source="myAS.as"/>
    <mx:HTTPService id="https1" result="chuli()" url="data.xml" useProxy="false"/>
    <mx:ComboBox id="cb1" fontSize="12"/>
    <mx:Label text="{cb1.selectedItem.label}" y="50" fontSize="12"/>
</mx:Application>
```

myAS.as 完整代码为

```
private function init():void {
    https1.send();
}

private function chuli():void {
    cb1.dataProvider = https1.lastResult.root.rec;
}
```

这样虽然把 as 和 mxml 分离了,但是每次保存的时候,as 还是被内置到 MXML 里了
 如果不用 Flex Data Service,你不可能动态导入外部 AS 代码

笔记日期: 2006/12/08

今天主要讲一下 Flex 中的 CSS 使用方法,以及 AS 3.0 基础

请把 Lesson1.mxml 所有代码删除,然后替换为以下代码

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

现在打开 Design 模式,从 Components 面板里拖几个 Button 出来，就拖 3 个吧。好，更改每个 button 的 label，改为按钮 1、按钮 2、按钮 3



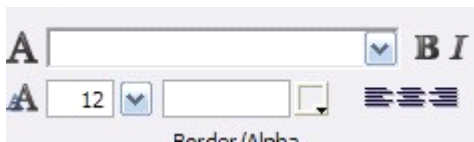
如图所示

我们可以看到, Flex 对中文的显示不是太好



我告诉大家，中文字体大小设置为 12px 最

好。我们现在选中按钮 1，然后在右边的属性面板里，调整它的字体大小：



改成 12，可以看到，按钮 1 现在清晰多了。我们可


以加更多的样式上去，比如字体颜色、效果等等。

（问：请问一下，可以将 `fontSize="12px"` ？答：不行，12 px 只能在 CSS 里定义）

嗯，我们可以看到，当调整“按钮 1”的样式的时候，Source 面板里的代码立即改变了，现在就有个问题——如果按钮很少，就几个，我们可以一个一个的进行样式设计，但是，如果按钮有 1 万个，或者更多呢？这个时候就得用到 CSS，方法如下：

在 Application 标签下新建

```
<mx:Style>
    Button { fontSize: 12px }
</mx:Style>
```

这个代码一写完，我们可以看到，Design 模式下，所有按钮全部变为 12 号字体。如果按钮没有变成 12 号字体，那可能是没有刷新，请点刷新按钮 ，大家现在的按钮都是 12 号了吧。

我先解释一下 px 和 pt 的区别：

又拖一个 button 进去，也是 12 号，自动的。px 是像素单位，它是矢量的，它会随着屏幕分辨率的变化而变化；pt 则是表量单位，即使屏幕分辨率再大，pt 单位的字体不会改变大小，我们尽量不要使用 pt，而应该使用 px。

CSS 里面，所有数值单位，默认都是 px 的，你不需要把 px 加到数字后面：

```
<mx:Style>
    Button { fontSize: 12 }
</mx:Style>
```

这样也是正确的，单位默认 px。

好，现在我具体讲讲 CSS 的用法：

CSS 分 2 大类型：Type Selector 和 Class Selector

```
Button { fontSize: 12 }
```

这个属于 Type Selector, 它应用于所有的 Button 组件

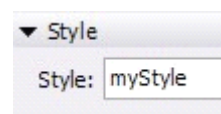
```
.myStyle { fontSize: 12 }
```

这个属于 Class Selector,它是专门用在指定组件上的

请将 CSS 改成这样的

```
<mx:Style>
    Button { fontSize: 12px }
    .myStyle { color: #FF0000; fontSize: 15px }
</mx:Style>
```

现在选中“按钮 2”，然后，在属性面板里更改 Style，如图：



观察 Design 面板的变化，大家看到，按钮 2 的代码也发生了变化

```
<mx:Button x="10" y="51" label="按钮 2" styleName="myStyle"/>
```

注意，所有的组件，如果要应用 Class Selector 到它们身上，都要写在 styleName 属性里，比如 styleName="myStyle"。

现在我讲讲优先级：

如果一个组件，它既有 Type Selector 又有 Class Selector，并且还有自己标签上的样式属性，它们的优先级别是——

自己标签上的样式属性 最高

Class Selector 第 2

Type Selector 第 3

——

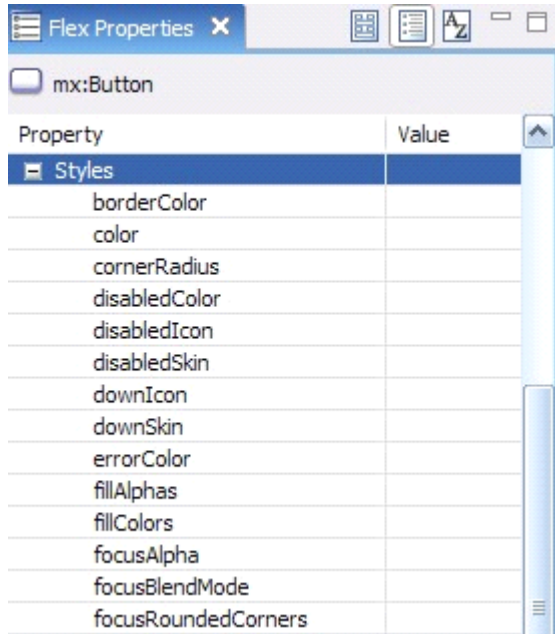
但是按照作用范围划分的话

自己标签上的样式属性 只作用于这一个组件

Class Selector 则作用与所有 styleName 等于这个 Class Selector 的组件上

Type Selector 则作用于整个类型的组件

好，现在讲将你能够写到 CSS 里的所有样式，其实很简单，我们只要打开属性面板里的 Styles，就可以看到所有能够修改的样式，如图：



我的建议是：先在这个面板里定义样式，定义完了之后，在把样式转移到 CSS 里面，我举个例子，教大家转移的方法，选中“按钮 3”， 请看图：

Styles	
borderColor	<input type="text" value="#FFFFFF"/>
color	<input type="text" value="#0000FF"/>
cornerRadius	10
disabledColor	
disabledIcon	
disabledSkin	
downIcon	
downSkin	
errorColor	
fillAlphas	[1, 0]
fillColors	[#ffffff, #ffffff]



我简单设置了一下按钮 3 的样式，现在按钮 3 的样子是这样的一。好，转 Source 模式——

```
<mx:Button x="10" y="93" label=" 按钮 3" color="#0000FF" borderColor="#FFFFFF"
cornerRadius="10" fillColors=["#ffffff, #ffffff]" fillAlphas="[1, 0]"/>
```

我们现在要把按钮 3 的样式转移到 CSS 里面——

我们首先裁剪这个部分

```
color="#0000FF" borderColor="#FFFFFF" cornerRadius="10" fillColors=["#ffffff, #ffffff]"
fillAlphas="[1, 0]"
```

注意是裁剪。然后，在 CSS 里面，这样写：

```
.b3 { color="#0000FF" borderColor="#FFFFFF" cornerRadius="10" fillColors=["#ffffff, #ffffff]"
fillAlphas="[1, 0]" }
```

也就是先粘贴到 CSS 里的 Class Selector b3 里，当然现在这个写法是不对的，我们需要修改成 CSS 格式，其实也很简单，CSS 永远是——

```
{ 样式名: 值; }
```


（问：Tooltip 的圆角和阴影如何修改？答：）

正确的写法应该是

```
.b3 { color: #0000FF; borderColor: #FFFFFF; cornerRadius:10; fillColors: #ffffff, #ffffff;  
fillAlphas: 1, 0 }
```

然后，在 按钮 3 上加入 styleName 属性，即——

```
<mx:Button x="10" y="93" label="按钮 3" styleName="b3"/>
```

这样 CSS 就转移成功了。

如果 b3 这样样式要应用到所有 Button 上，则——

```
.b3 { color: #0000FF; borderColor: #FFFFFF; cornerRadius:10; fillColors: #ffffff, #ffffff;  
fillAlphas: 1, 0 }
```

修改成——

```
Button { color: #0000FF; borderColor: #FFFFFF; cornerRadius:10; fillColors: #ffffff, #ffffff;  
fillAlphas: 1, 0 }
```

也就是把 Class Selector 修改成 Type Selector

所有的 Class Selector 前面全部要加 “.”， 所有的 Type Selector 前面没有点

注意了，Type Selector 的名字只能是组件名字，比如 Button、Label、Panel 等等。前面没有点，Class Selector 可以是任意名字，但是前面必须加点，也就是说：

```
Button { fontSize: 12px }
```

```
.Button { color: #FF0000; fontSize: 15px }
```

第 1 个是 Type，第 2 个是 Class 的。

这是现在的完整代码

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

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
```

```
  <mx:Style>
```

```
    Button { fontSize: 12px }
```

```
    .myStyle { color: #FF0000; fontSize: 15px }
```

```
    .b3 { color: #0000FF; borderColor: #FFFFFF; cornerRadius:10; fillColors: #ffffff, #ffffff;  
fillAlphas: 1, 0 }
```

```
  </mx:Style>
```

```
  <mx:Button x="10" y="10" label="按钮 1" />
```

```
  <mx:Button x="10" y="51" label="按钮 2" styleName="myStyle"/>
```

```
  <mx:Button x="10" y="93" label="按钮 3" styleName="b3"/>
```

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

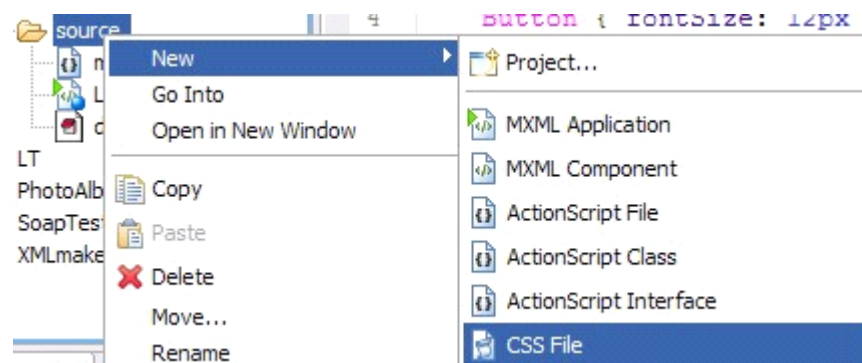
效果如下：



大家用我这个代码，看效果是不是一样的。

现在讲 CSS 最后一个知识点——CSS 的外部载入：

如图：



新建 CSS File：lessonStyle.css

打开 lessonStyle.css

将 Lesson1.mxml 中的内容

Button { fontSize: 12px }

.myStyle { color: #FF0000; fontSize: 15px }

.b3 { color: #0000FF; borderColor: #FFFFFF; cornerRadius:10; fillColors: #ffffff, #ffffff;
fillAlphas: 1,0 }

裁剪之后，粘贴到 lessonStyle.css 里，

好，Lesson1.mxml 中的 Style 标签改成——

<mx:Style source="lessonStyle.css"/>

Lesson1.mxml 完整代码

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

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">

<mx:Style source="lessonStyle.css"/>

<mx:Button x="10" y="10" label="按钮 1" />

<mx:Button x="10" y="51" label="按钮 2" styleName="myStyle"/>

<mx:Button x="10" y="93" label="按钮 3" styleName="b3"/>

</mx:Application>

lessonStyle.css 完整代码

Button { fontSize: 12px }

.myStyle { color: #FF0000; fontSize: 15px }

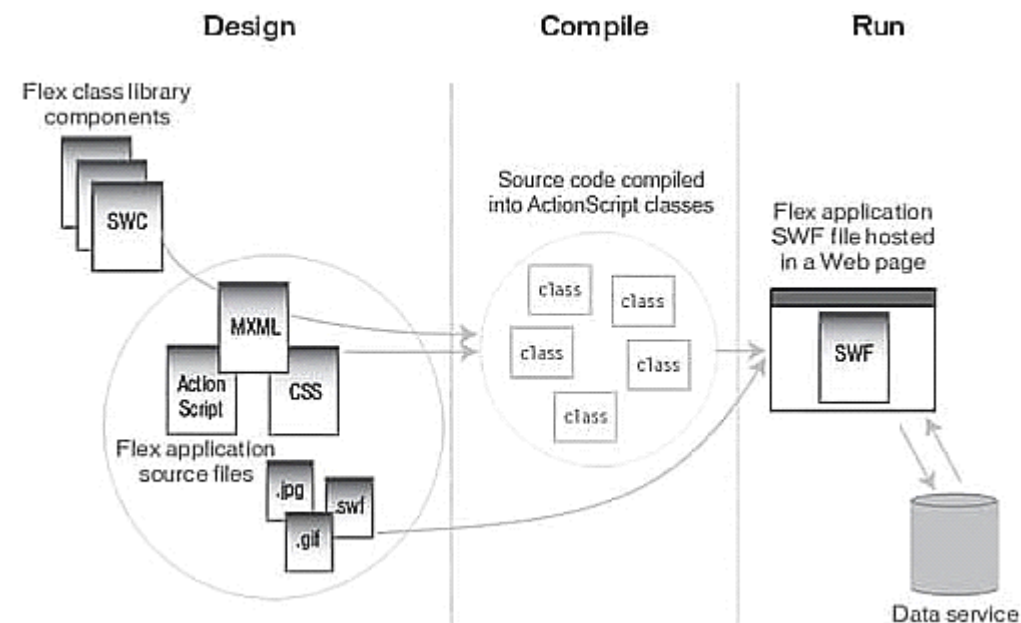
```
.b3 { color: #0000FF; borderColor: #FFFFFF; cornerRadius:10; fillColors: #ffffff, #ffffff; fillAlphas: 1, 0 }
```

然后保存。注意了，Flex 里的 CSS 虽然可以放到外面，但是，当编译成 SWF 以后，CSS 其实是不能实现动态载入的

（问：是不是在发布的时间一定要保存 .css 文件？答：发布的时候，可以把样式单独保存在 CSS 文件里，也可以内置到 MXML 里，都可以，没多大区别）

我来讲讲原理——

给大家看张图：



请记住这张图，很重要，它让大家了解 Flex 的运做方式。

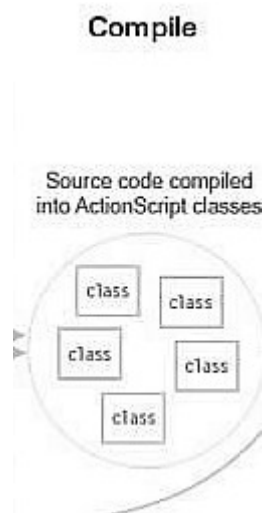
（看不到图的,请自己翻 Flex 帮助的这一章

[How Flex Works](#)

[Building and running Flex applications](#) 翻到那一页的下面就能看到图)

我来解释一下这张图：

开发 Flex 程序的时候，Flex 的 SWC(Flex 的组件包)，MXML、AS、CSS 以及一些内置的素材像 jpg、swf、gif 等等，在你保存项目的时候，全部被编译到一个 SWF 文件里

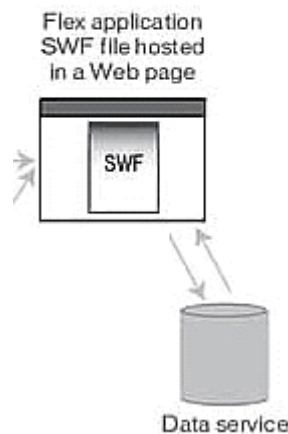


这个是编译过程

主要是把 MXML Application 文件编译成主类(class)，把其他导入的类也一同编译进去，成为副类。

比如我们这里的 Lesson1.mxml 就是一个 MXML Application。编译的时候，Lesson1 是主类，而其他导入的类，比如 Button 类，也被一起编译进去，成为副类。

Run



编译的最终结果是一个 SWF 文件。所以，我要强调的是：不论你的 CSS 是内置的还是外置的，不论你的 AS 是内置的还是外置的，编译成 SWF 以后，都不可能动态导入外部 CSS 和 AS 文件。也就是说编译成 SWF 以后，即使外部 CSS 或者 AS 文件变化了，SWF 文件本身不可能再变化了。外部 AS 和 CSS 文件，唯一的用途是为了便于开发。Flex Data Service 可以动态编译 MXML 和 AS 成为 SWF，但是注意，每次外部的 AS 或者 CSS 文件修改以后，Flex Data Service 要重新编译 1 次程序才能生成新的 SWF，而不是说，有了 Flex Data Service 就可以动态导入 CSS 和 AS 了。

虽然已经编译好的 SWF 是不能动态导入外部 CSS 文件的，但是，Flex 2.01 中多了 2 个重要的新功能，一个是 CSS to SWF、一个是 Module，Flex 2.01 可以把一个 CSS 文件编译成 SWF 文件，然后主程序 SWF 就可以动态导入这个由 CSS 生成的 SWF 文件了，这个叫动态导入样式。as 也可以动态导入。

现在正式讲 AS 3.0 基础

首先，我们要认识变量，所有的变量声明都用 var 关键字，比如

```
var abc:String = "你好";
```

但是在 MXML 里声明变量的时候，一定要注意，也就是注意命名空间问题

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
```

大家可以看到 xmlns:mx="http://www.adobe.com/2006/mxml" 是命名空间定义

如果我们直接这样写的话

```
<mx:Script>
    <![CDATA[
        var abc:String = "你好";
    ]]>
</mx:Script>
```

保存的时候会警告你

方法 `var 'abc'` 没有 `namespace` 属性。它将作为 `'package-private'` namespace 默认值的一部分。`namespace` 即命名空间。在 MXML 里，命名空间你可以简单的理解为一个包—package

还是讲变量先

```
<mx:Script>
    <![CDATA[
        var abc:String = "你好";
    ]]>
</mx:Script>
```

这样写是不好的,我们在 `var` 前面一定要给它一个作用域的限制

比如

```
public var abc:String = "你好";
```

可以用的有 4 个，他们是 `public`, `private`, `internal`, `protected`

注意!!!

`public`, `private`, `internal`, `protected` 这 4 个在 AS 3.0 里面跟 JAVA 是不同的

```
public var abc:String = "你好";
```

证明 `abc` 这个变量任何其他的类都可以调用

```
internal var abc:String = "你好";
```

这个 `abc` 变量只能被跟它在一个包里面的类调用。包简单讲吧，就是一个文件夹。

```
internal var abc:String = "你好";
```

`abc` 变量只能被跟它在一个文件夹里的类调用

`internal` 是默认的，你不写它也可以：

```
internal var abc:String = "你好";
```

```
var abc:String = "你好";
```

这 2 种等效

```
protected var abc:String = "你好";
```

`abc` 只能被自己和自己的子类调用，其他任何类都无法调用

```
private var abc:String = "你好";
```

`abc` 只能在自己的类里面调用，任何其他类无法调用

大家一定要注意，在 Flex 2.01 没有发布以后，在 MXML 里所有变量全部用 `private` 限制

因为，如果你写成 `public` 的，当你的程序编译成 SWF 以后，其他的 SWF 其实是可以直接访问你的 `public` 变量和方法的!!! 这个是非常危险的

好，然后我们谈谈变量类型定义：

和 java 不同，AS 3 变量类型是写在变量的后面的，请看——

```
private var abc:String = "你好";
```

`String` 是一个 AS 3.0 的类(Class)

同理，我们还可以有——

```
private var b1:Button;
```

这样就声明了一个 `Button` 变量。说得更准确一点, `b1` 应该是 `Button` 类的对象, `Button` 的实例。

```
var a : ArrayCollection
```

定义一个数组。

```
<mx:Button id="b1" label="按钮 1" />
```

```
<mx:Button id="b2" label="按钮 2" />
```

```
<mx:Button id="b3" label="按钮 3" />
```

这个相当于

```
private var b1:Button;
```

```
private var b2:Button;
```

```
private var b3:Button;
```

声明了 3 个 `Button` 类的对象

好, 现在讲变量的使用

AS 3.0 当中, 你要修改一个变量必须放到 `function` 里, 比如:

```
private var b1:Button;
```

```
private function dosth():void {
```

```
    b1 = new Button();
```

```
    b1.enabled = false;
```

```
    addChild(b1);
```

```
}
```

在方法外面, 你只能声明变量, 顶多在赋予一个初值, 比如:

```
private var b1:Button = new Button();
```

```
private var b1:Button;
```

```
private var b1:Button = new Button();
```

都是合法的

但是

```
private var b1:Button = new Button();
```

```
b1.enabled = false;
```

这个是绝对不可以的

在方法外面, 永远只能声明变量(如 `private var b1:Button`), 或者声明的同时赋予它一个内存空间(如 `private var b1:Button = new Button()`)。除此以外, 你不能做任何事情。任何事情都要放到方法里

```
private var b1:Button;
```

```
private function dosth():void {
```

```
    b1 = new Button();
```

```
    b1.enabled = false;
```

```
    addChild(b1);
```

```
}
```


这样才是合法的。

（问：那怎么执行这个方法呢？

答：

我说了，Flex 是个多米诺，只有在发生特定事件的时候，才能执行一个方法。这是第一堂课讲的，我举个例子，请将 Lesson1.mxml 改成

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
        <![CDATA[
            private function dosth():void {
                label1.text = "你点了按钮!";
            }
        ]]>
    </mx:Script>
    <mx:Button click="dosth()" label="Button" x="10" y="47"/>
    <mx:Label id="label1" text="按钮" x="10" y="10" />
</mx:Application>
```

我们要达到的目的是，当我点击按钮以后，执行 dosth()方法，然后 label1 上的文字变成了"你点了按钮!"

为了更便于理解，请再更新 1 次代码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
        <![CDATA[
            private function dosth():void {
                label1.text = "你点了按钮!";
            }
        ]]>
    </mx:Script>
    <mx:Label id="label1" text="标签" x="10" y="10" fontSize="12"/>
    <mx:Button click="dosth()" label="按钮" x="10" y="47" fontSize="12"/>
</mx:Application>
```

复制以上代码到 lesson1.mxml,然后保存。接着，打开 bin，然后双击 Lesson1.html



当我们点了按钮以后，大家看看标签变化成了什么。

我们在观察一下 Button 的代码

```
<mx:Button click="dosth()" label="按钮" x="10" y="47" fontSize="12"/>
```

click="dosth()"

click 属性是 Button 类的事件类属性

click="dosth()" 的意思是，当用户点击按钮的时候，就发生了 click 事件，接着执行 dosth() 方法

在 MXML 里，事件全部被简化了。

如果不用 click="dosth()" 的话，我们可以在 as 代码里写出同样效果：

首先，主标签加入 creationComplete="init()"

```
<mx:Application creationComplete="init()" xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
```

然后写 init() 方法

完整代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application creationComplete="init()" xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
    <mx:Script>
        <![CDATA[
            private function init():void {
                b1.addEventListener(MouseEvent.CLICK, dosth);
            }

            private function dosth(e:MouseEvent):void {
                label1.text = "你点了按钮!";
            }
        ]]>
    </mx:Script>
    <mx:Label id="label1" text="标签" x="10" y="10" fontSize="12"/>
    <mx:Button id="b1" label="按钮" x="10" y="47" fontSize="12"/>
</mx:Application>
```

```
b1.addEventListener(MouseEvent.CLICK, dosth);
```

这一句等同于 MXML 里 click="dosth()"

)

变量有 2 种特殊类型：（很有用！可以实现 java 中的类型注入依赖的功能）

（提醒一点，AS3 在定义数组类型时可以像 java1.5 中一样可以指定数组中的类型！以后讲讲吧）

***类型和 function 类型——**

类型代表这个变量可能是任何类型，比如 private var abc:;

当然，这种变量常用在方法的参数里

```
private function init():void
```

这个等同于

```
private var init:function;  
init = function():void {  
}  
}
```

好，再简单讲讲局部变量

```
private var abc:String;  
private function dosth():void {  
    var def:String = new String();  
}
```

abc 是成员变量，整个类里可以用；def 是局部变量，只能在 dosth()方法里用。

在方法内声明的变量前面不需要 private，直接 var def:String = new String();

现在简单讲讲方法的格式：

```
private function dosth():void { }
```

dosth 是方法名

()里的是参数,当前没有参数

void 是返回值,当前没有返回空值

```
private var c:Number;  
private function dosth(a:Number, b:Number):Number {  
    c = a+b;  
    return c;  
}
```

c 是成员变量，a 和 b 是方法参数，a 和 b 是局部变量,前面不需要加 var，它们只存在与 dosth 方法内。

dosth(a:Number, b:Number):Number

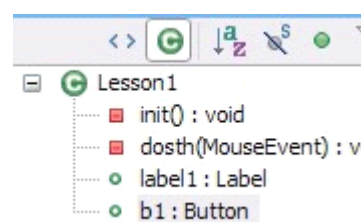
这个方法执行以后，返回 Number 类型的数据

我举个简单例子——

```
<mx:Button id="b1" click="dosth(1,2)"/>
```

当用户点击这个按钮的时候，执行 dosth 方法，同时传递 1 和 2 这两个参数给方法，即 a=1, b=2，这样的话，最终 c = 3，这个方法最终返回结果 3

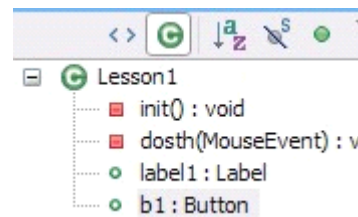
好了，方法也讲完了，然后是类：



所谓的 Class(类)，就是变量和方法的集合。在 MXML 中，每个 MXML 文件的文件名都是主类。比如这里的主类是 Lesson1，即

```
Public Class Lesson1 {
}
```

然后里面写有变量和方法



这张图写成类相当于：

```
public class Lesson1 {
    public var label1:Label;
    public var b1:Button;
    private function init():void {
        b1.addEventListener(MouseEvent.CLICK, dosth);
    }
    private function dosth(e:MouseEvent):void {
        label1.text = "你点了按钮!";
    }
}
```

当然，这写的不完整，但是大概的意思明白就行了：Lesson1 是主类。实际上，Button 和 Label 都是我们导入的类，我没有写，如果写上去应该是：

```
package source {
import mx.controls.Label;
import mx.controls.Button;
public class Lesson1 {
    public var label1:Label;
    public var b1:Button;
    private function init():void {
        b1.addEventListener(MouseEvent.CLICK, dosth);
    }
    private function dosth(e:MouseEvent):void {
        label1.text = "你点了按钮!";
    }
}
}
```

source 是存放 Lesson1 这个类的目录,称为包(package)

(问：可以用 import mx.controls.*?答：可以，不过你这样可能使最终的 swf 变大)

(问：这个 (e:MouseEvent)是不是鼠标事件？)

答：

```
private function init():void {
    b1.addEventListener(MouseEvent.CLICK, dosth);
}
```

b1 按钮添加事件监视器 (addEventListener), 监视鼠标事件 (MouseEvent) 中的点击 (CLICK) 事件, 如果在 b1 按钮上发生了点击事件, 则运行 dosth 这个方法

```
private function dosth(e:MouseEvent):void {  
    label1.text = "你点了按钮!";  
}
```

当 dosth 被调用的时候, MouseEvent 类创建一个对象 e, 你可以把这个 e 放到方法体内。比如:

```
private function dosth(e:MouseEvent):void {  
    label1.text = "你点了按钮!";  
    trace(e.target);  
}
```

(

续问: 这个 e 不用传值吗

续答:

注意, (e:MouseEvent) 在这里必须得写

而 e 是传递到方法体内部的, 你想用 e 也可以, 你不想用 e 也可以

)

(e:MouseEvent) 写成 (event:MouseEvent) 或者 (abcdegf:MouseEvent)

都可以

最常用的是 event 和 e

)

课后问答:

1、问: `<mx:TextInput id="password"/>`
`<mx:TextInput id="username"/>`
`<mx:Button x="10" y="397" label="确认" click="bb();"/>`
bb() 怎么写判断不为空, 在提示啊!!

答: `<mx:TextInput id="password"/>`
`<mx:TextInput id="username"/>`
`<mx:Button x="10" y="397" label="确认" click="bb();"/>`
`<mx:Script>`
 `<![CDATA[`
 `private function bb():void {`
 `//你的代码`
 `}`
 `]]>`
`</mx:Script>`

2、问: 比如有一个 aaa.xml, 现在有一个按钮, 点击进行进入另一个 bbb.xml
就像两个网页一个, 比如 aaa 为第一个网页, bbb 为另一个网页。但是在 flex builder 中是生

成 swf

答：相互通讯，有 2 种方法：

一个是 LocalConnection 传统方法，一个是 ApplicationDomain 高级方法
用 LocalConnection 可以将 AS 3.0 做的 SWF 和 AS 2.0 做的 SWF 互相通讯

(

续问：那我再问一个：mxm1 中有没有 session 这种呢

续答：session 没有的

)

如果你要把 2 个 swf 互相通讯，那就不是这堂课的内容了，第 3 堂课讲 2 个 swf 如何互相通讯，涉及到模块概念，比较深。

```
3、问：private function init():void {  
    b1.addEventListener(MouseEvent.CLICK, dosth);  
}  
private function dosth(e:MouseEvent):void {  
    label1.text = "你点了按钮!";  
}
```

这个 可以用 你第一次讲的 private function dosth():void {
 label1.text = "你点了按钮!";
}

么

答：不行

b1.addEventListener(MouseEvent.CLICK, dosth);
只要有 addEventListener,那么 dosth 就一定得有(e:MouseEvent)
如果是在 MXML 里的 click="dosth()"
那么就应该写成

```
private function dosth():void {}
```

如果你想在 MXML 里引用 e:MouseEvent 的话，这样写：

```
click="dosth(event)"
```

注意，这里的 event 一定不能改成别的

然后，dosth 方法可以这样写

```
private function dosth(e:MouseEvent):void {  
    label1.text = "你点了按钮!";  
}
```

也可以这样写

```
private function dosth(event:MouseEvent):void {  
    label1.text = "你点了按钮!";  
}
```

都可以，

(e:MouseEvent)

e 可以随便写

但是 click="doSomething(event)", event 不能改成别的

4、问：在 flex 里怎么调出调试面板。刚才的 trace() 在哪看到？

答：

trace() 里的内容只有在 debug 模式下能看到。用 debug 运行 flex。然后监视你的 Console 面板

(续问：debug 模式跟 run 模式有什么不同？)

续答：debug 下可以查看 trace()

可以设置断点, 可以查看每个变量当前的值

debug 的用法我就不阐述了, 大家自己查帮助

)

5、问：怎么建超连接？

答：

看看这个

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application    creationComplete="init()"    xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
    <mx:Script>
        <![CDATA[
            private function zuoshi():void {
                navigateToURL(new URLRequest("http://8588.cc/pa"), "_blank");
            }
        ]]>
    </mx:Script>
    <mx:Button click="zuoshi()" label="Button"/>
</mx:Application>
```

很简单，点了按钮以后，网页跳到我的相册

navigateToURL 是全局函数。全局函数的意思是，可以被直接调用，而不用导入任何类。

6、问：搜索工具条怎么弄？

答：把东西都拖入 <mx:ApplicationControlBar/> 里

最后 1 堂课, 讲 Flex 高级部分

笔记日期：2006/12/15

可以说是 Flex 客户端技术的核心内容

主要讲 3 个东西：

- 1、Flex 模块技术
- 2、动态加载皮肤
- 3、ApplicationDomain 类的用法

因为前 2 个东西只有在 Flex 2.01 发布以后才能用，所以今天这节课只能当作 Flex 2.01 的预习

先进 Flex 模块技术

简单跟大家讲讲什么是模块、有什么作用、以及优势：

开发一个 Flex 应用程序的时候，如果代码太多，生成的 swf 就越发臃肿，这样的话，别人访问起来，下载时间就很长。因此，我们需要把一个 Flex 程序拆分成多块。然后，主程序的体积就非常小了。不过，Flex 2 最小的程序也有 100KB。

现在我来讲讲怎么做模块——

首先，做模块的时候，我们不再需要用 `mx:Application` 标签为顶级标签了。因为如果你非要以 `mx:Application` 标签为顶级标签的话，那么不论你代码内容有多少，生成的 swf 最小也有 100KB。因为 `Application` 类太大了，swf 的大小与 `Application` 类有关，`Application` 是个非常臃肿的类，我希望在未来的 flex 3.0 里能精简它的体积。

做模块——

首先新建一个 MXML 文件，取名 `abc.mxml`

（目前的 Flex 2.0 可以新建 MXML Application 和 MXML Componet 这都不是做我们模块用的。在 Flex 2.01 中，可能多出一个新建 MXML Module 的选项，我们做模块应该新建它）目前我们使用的 Flex 2.0 新建不了模块

不论 Flex 2.01 中会不会多新建 MXML Module 的选项，我们都可以自己建一个，比如 New—File，也就是新建一个普通文件，取名 `abc.mxml`。

好注意了，每个模块文件的顶级标签都是：

```
<mx:Module>
    .....
</mx:Module>
```

也就是说，`abc.mxml` 至少要包含以下代码：

```
<?xml version="1.0"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml">
</mx:Module>
```

然后，在 `<mx:Module></mx:Module>` 里，可以添加任何你想添加的 flex 组件。就跟 MXML Application 中添加组件一样，也可以添加代码，依然放到

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

里。我现在给大家写个示例模块文件——

`abc.mxml` 模块文件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module      xmlns:mx="http://www.adobe.com/2006/mxml"      creationComplete="init()"
layout="absolute">
    <mx:Button x="49" y="225" label="Button"/>
```

```

<mx:DataGrid x="31" y="27">
    <mx:columns>
        <mx:DataGridColumn headerText="Column 1" dataField="col1"/>
        <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
        <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
    </mx:columns>
</mx:DataGrid>

```

```
</mx:Module>
```

大家可以看到，这个文件除了顶级标签是<mx:Module>以外，其他的跟普通的 MXML 应用程序是一样的。

现在我们把 abc.mxml 编译成 swf，最简单的方法就是直接在命令行里输入

```
mxmmlc abc.mxml
```

这样，abc.mxml 直接编译成了 abc.swf。不过有一点要注意，那就是导入类的问题，如果一个模块中用到太多 flex 组件，在默认情况下，这些 flex 组件的类会被自动导入，也就是说，编译成 swf 以后，swf 体积会很大，模块的意义也没有了。

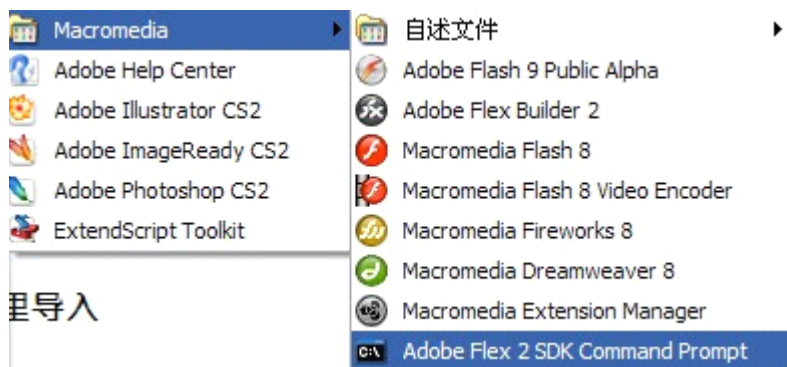
因此，我们需要把一个模块里要用到的类全部在主 flex 程序里导入，假设 main.mxml 是主程序 abc.mxml 是模块，我们要把 abc.mxml 里用到的所有类在 main.mxml 里导入，方法如下：

1、mxmmlc -link-report=report.xml main.mxml（这表示生成一个 report.xml 文件，它起链接作用）

2、接着

```
mxmmlc main.mxml
```

生成 main.swf，主应用程序



在 Adobe Flex 2 SDK Command Prompt 里输入，当然,要找到正确的 mxml 路径

3、最后

```
mxmmlc -load-externs=report.xml abc.mxml
```

这样,abc.mxml 就会从 main.mxml 导入类了

我再说一次，main.mxml 是主程序，abc.mxml 是模块，分 3 步走——

1、mxmmlc -link-report=report.xml main.mxml

生成 report.xml,里面包含 main.mxml 用到的类

2、mxmmlc main.mxml

编译主应用程序到 main.swf 文件

3、mxmmlc -load-externs=report.xml abc.mxml

编译模块到 abc.swf,导入外部 main.swf 中的类

这样，abc.mxml 就会非常非常小了，我估计一般的模块文件只有 10KB 左右。注意，abc.swf

是不可能被单独运行的，如果你单独打开 abc.swf,你看到的就是一片灰色，什么都没有，模块不允许被单独运行，只能加载到主 flex 应用程序里。

好，现在我教大家怎样在主 flex 应用程序里载入模块——

在主应用程序里，我们多了一个新标签<mx:ModuleLoader>，也可以说在 AS 3.0 里多了一个新类 ModuleLoader，最简单的方法是：

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:ModuleLoader url="abc.swf"/>
</mx:Application>
```

这样，主程序一被打开，模块 abc.swf 就立即被加载了，你可以在 AS 3.0 代码里写。比如：

```
<mx:Application creationComplete="init()" xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            private function init():void {
                ml.url="abc.swf";
                ml.loadModule();
            }
        ]]>
    </mx:Script>
    <mx:ModuleLoader id="ml"/>
</mx:Application>
```

以上代码意思是，当主程序创建完毕以后，执行 init()方法，设置 ml 的路径 url="abc.swf"，载入模块 ml.loadModule()。要注意，载入模块就意味着我们可以卸载模块。卸载模块会释放内存，对程序性能提升很有帮助。卸载模块的模仿是 ml.unloadModule()。卸载模块时间没有固定的，按自己需要而定。当程序在某个状态下不需要再用到某个模块的时候，就 unloadModule()，然后如果后来又需要的时候，就 loadModule()，如此反复即可。卸载模块后要重新加载不会重新下载 1 次模块文件，但是卸载模块后会释放内存，这是最重要的。还有就是，不论你刻意的调用 loadModule()方法多少次，模块永远只会加载 1 次。

mx:ModuleLoader 还有很多事件，比如 complete 事件、error 等等，这里我就不详细讲了，用法跟普通组件一样。

现在讲动态加载样式

在 Flex 2.01 中，可以将 CSS 文件编译成 SWF。比如，我们新建一个 CSS 文件，取名 style.css style.css 的内容如下：

```
Application { backgroundAlpha: 0 }
ApplicationControlBar { cornerRadius: 10px; fillColors: #ffffff,#0099DD; fillAlphas: 0.75,0.75;
shadowDistance: 2 }
Panel { headerHeight: 14px; fontSize: 12px; cornerRadius: 10px; roundedBottomCorners: true;
shadowDistance: 0 }
Label { fontSize: 12px; fontWeight: bold }
LinkButton { fontSize: 12px; color: yellow; rollOverColor: white; textRollOverColor: blue; }
```

```
TileList { paddingLeft: 0; paddingRight: 0; paddingBottom: 0; paddingTop: 0; }
HorizontalList { paddingLeft: 0; paddingRight: 0; paddingBottom: 0; paddingTop: 0; }
我们可以把这个 CSS 文件编译成 SWF——
```

方法是 `mxmmlc style.css`

生成 `style.swf`

这个 `css` 当中可以有 `class selector`, `type selector`, 还可以内置皮肤文件, 比如

```
Application {
    backgroundImage : "assets/greenBackground.gif";
    theme-color: #9DBAEB;
}
```

```
Button {
    fontFamily: Tahoma;
    color: #000000;
    fontSize: 11;
    fontWeight: normal;
    text-roll-over-color: #000000;
    upSkin: Embed(source="../../assets/orb_up_skin.gif");
    overSkin: Embed(source="../../assets/orb_over_skin.gif");
    downSkin: Embed(source="../../assets/orb_down_skin.gif");
}
```

仔细看, 这个 `css` 里除了有普通样式定义(比如 `theme-color: #9DBAEB`)以外, 最重要的就是内植皮肤了:

```
upSkin: Embed(source="../../assets/orb_up_skin.gif");
```

这个很关键

我们用 `mxmmlc` 把 `style.css` 编译成 `style.swf`

注意: `style.swf` 当中现在包含所有上面提到的皮肤文件(也就是上面的 `gif` 文件)

`style.swf` 里内置了所有皮肤文件。不过注意, 单独打开 `style.swf` 是什么也没有的, 这个跟模块是一样的。皮肤当然自己做了。

好, 现在我讲怎样在主程序里动态载入皮肤:

在 `mx:Application` 的 `mx:Script` 标签中, 我们首先要导入 `StyleManager` 类, 我们首先要导入 `StyleManager` 类, 它专门负责样式管理。

即

```
<mx:Script>
<![CDATA[
import mx.styles.StyleManager;

public function init():void {
    StyleManager.loadStyleDeclarations("style.swf")
}
]]>
</mx:Script>
```

init()在 mx:Application 发生 complete 事件时调用。大家看到了吗，很简单。

loadStyleDeclarations()是 StyleManager 类的静态方法(类方法,static)，所以 StyleManager 直接可以调用 loadStyleDeclarations()

现在我讲讲 loadStyleDeclarations()方法中的 3 个参数——

第 1 个参数：样式 swf 文件的位置。我们用的是"style.swf"

第 2 个参数：是否更新参数，只接受 2 个值，true 和 false，默认 true。

StyleManager.loadStyleDeclarations("style.swf",true);

这意味着，一旦载入 style.swf 以后，立即更新主应用程序当中所有组件的样式。这里的 true 可以不写，因为默认是 true。

第 3 个参数：是否信任载入内容，只接受 2 个值，true 和 false。

如果 style.swf 和主程序 main.swf 是在一个域名下，那么默认是 true。

如果 style.swf 和主程序 main.swf 分别在 2 个不同域名下，默认 false。

最后，我们讲 loadStyleDeclarations()方法的返回值

loadStyleDeclarations()方法讲返回 IEventDispatcher 类的实例，IEventDispatcher 是事件指派类。这个类可以调用.addEventListener()方法

等等，我先给大家讲讲几个基础，再回来说怎么调用方法。

首先，我们 flex 里 95%的事件类都是 flash.events.Event 类的子类。比如 MouseEvent，FlexEvent、MenuEvent 类等，都是继承 flash.events.Event 类
mx.events.* 和 flash.events.* 所有的类，全部是继承自 flash.events.Event。

现在我们再来讲讲 EventDispatcher 类：

flash.events.EventDispatcher

它是事件指派类，这个类当中含有我们最常用的 addEventListener()方法，以及手动指派事件方法 dispatchEvent()，这是 flash.events.EventDispatcher 类的所有方法。

Method
EventDispatcher (target:*) Aggregates an instance of
addEventListener (type:String, listener:Function, priority:int = 0, useWeakReference:Boolean = false) Registers an event listener that receives notification of an event of the specified type.
dispatchEvent (event:Event) Dispatches an event into the event dispatching mechanism of the object.
hasEventListener (type:String) Checks whether the EventDispatcher has a listener of the specified type of event.
removeEventListener (type:String, listener:Function) Removes a listener from the event dispatcher.
willTrigger (type:String):Boolean Returns a Boolean value indicating whether the EventDispatcher has a listener of the specified type of event.

可以看到，它们都是我们平时最常用的方法。

好，现在我来讲联系——

我们所有的 flex 组件，都是 DisplayObject(显示对象类)的子类。比如 Button、TextInput、Panel 等，都是 DisplayObject 的子类。而 DisplayObject 又是 flash.events.EventDispatcher 的子类。又因为所有子类都可以直接调用父类的方法，所以，所有 flex 组件都可以调用 EventDispatcher 类的所有方法。最常用的就是 addEventListener()了，也就是说，Event 类是用来定义各种事件的。比如，这些都是 event 类的常数，他们分别指各种事件

ACTIVATE : String = "ac
[static] Defines the value
ADDED : String = "added
[static] Defines the value
CANCEL : String = "cancel
[static] Defines the value
CHANGE : String = "change
[static] Defines the value
CLOSE : String = "close"
[static] Defines the value
COMPLETE : String = "complete
[static] Defines the value
CONNECT : String = "connect
[static] Defines the value
DEACTIVATE : String = "deactivate
[static] Defines the value
ENTER_FRAME : String = "enterFrame
[static] Defines the value

比如 COMPLETE 指某个任务完成的事件

Event 类是用来定义各种事件，EventDispatcher 类用来指派，监听各种来自 Event 或者 Event 子类的事件。而所有的 Flex 组件都是 DisplayObject 的子类，DisplayObject 又是 EventDispatcher 的子类，所以，所有组件都能调用 addEventListener()等 EventDispatcher 类的方法。

最后，我说一下 IEventDispatcher，**确切说，它是一个接口类。**

EventDispatcher 类里调用 IEventDispatcher 接口，IEventDispatcher 接口包含所有 EventDispatcher 类的方法,如图

Method
EventDispatcher (target:EventListener):void Aggregates an instance of EventDispatcher.
addEventListener (type:String, listener:EventListener, priority:int = 0, useWeakReference:Boolean = false):void Registers an event listener that receives notification of an event of the specified type.
dispatchEvent (event:Event):Boolean Dispatches an event into the event flow.
hasEventListener (type:String):Boolean Checks whether the EventDispatcher has an event listener of the specified type.
removeEventListener (type:String, listener:EventListener):void Removes a listener from the EventDispatcher.
willTrigger (type:String):Boolean Checks whether the EventDispatcher will trigger an event of the specified type.

好，现在回到前面，讲 `StyleManager.loadStyleDeclarations()` 方法返回的东西。

`loadStyleDeclarations()` 方法返回 `IEventDispatcher` 接口类的实例

我们可以这样赋值：

```
var myEvent:IEventDispatcher = StyleManager.loadStyleDeclarations("styles.swf");
```

这里 `myEvent` 就是 `loadStyleDeclarations()` 返回的值

它是一个 `IEventDispatcher` 接口类的对象，它可以调用 `IEventDispatcher` 接口中所有方法

（问：`loadStyleDeclarations` 返回的值是什么？

答：

```
loadStyleDeclarations():IEventDispatcher
```

```
loadStyleDeclarations()返回 IEventDispatcher 类的对象
```

```
)
```

`myEvent` 可以调用 `IEventDispatcher` 类的方法，如：

```
myEvent.addEventListener(StyleEvent.COMPLETE, dosth);
```

```
public function init():void {
    var myEvent:IEventDispatcher = StyleManager.loadStyleDeclarations("styles.swf");
    myEvent.addEventListener(StyleEvent.COMPLETE, dosth);
}
```

```
private function dosth(event:StyleEvent):void {
    trace("外部样式载入完毕了");
}
```

我直接解释上面的代码，希望大家能听懂——

主程序创建完毕以后执行 `init()` 方法；然后，`StyleManager` 开始载入外部样式文件(CSS 生成的 SWF)，即 `"style.swf"`。

StyleManager.loadStyleDeclarations("styles.swf"); 也就是这句
StyleManager.loadStyleDeclarations("styles.swf") 这一整句返回一个 IEventDispatcher 类的对象。我们把返回的对象赋值给 myEvent 这个 IEventDispatcher 类的对象
即：

```
var myEvent:IEventDispatcher = StyleManager.loadStyleDeclarations("styles.swf");
```

```
myEvent.addEventListener(StyleEvent.COMPLETE, dosth);
```

这句代表监视 StyleEvent 类当中的 COMPLETE 事件, 也就是当 StyleManager 完成载入 styles.swf 以后, 发生 StyleEvent.COMPLETE 事件, 发生 StyleEvent.COMPLETE 事件以后, 执行 dosth 方法

```
private function dosth(event:StyleEvent):void {  
    trace("外部样式载入完毕了");  
}
```

这就很简单了, dosth 方法中输出"外部样式载入完毕了"

```
myEvent.addEventListener("COMPLETE", dosth);
```

这种写法是完全错误的。

请大家注意, 这是经常会犯的一个错误

"COMPLETE" 这里代表什么呢? 什么也不能代表。因为你没有指定 "COMPLETE" 是哪个类的常数

一定要写 某某事件类.COMPLETE。比如 StyleEvent.COMPLETE。这代表 COMPLETE 是 StyleEvent 类 的常数

```
private function dosth(event:StyleEvent):void {  
    trace("外部样式载入完毕了");  
}
```

同样的, dosth 属于监视器方法, 它的参数当中必须对应上面的类, 即 dosth(event:StyleEvent)

myEvent.addEventListener("COMPLETE", dosth); 这样写完全错误, 大家千万不要犯这个错误

myEvent.addEventListener(StyleEvent.COMPLETE, dosth); —— 这才是正确的
dosth(event:StyleEvent)

```
private function dosth(event:StyleEvent):void {  
    trace("外部样式载入完毕了");  
}
```

这是我随便写的, 在 debug 模式下运行, style.swf 载入完毕以后, 控制台就会输出 "外部样式载入完毕了" 的字样

你可以随便改写监视器方法的内容, 比如:

```
private function dosth(event:StyleEvent):void {  
    this.width = 500;  
}
```

这样的话, 载入完 style.swf 以后, 主应用程序的宽度变成 500 了。

好，现在在讲 StyleManager 的另一个方法——

```
StyleManager.unloadStyleDeclarations("style.swf",true);
```

unloadStyleDeclarations()是另一个方法，它用来卸载外部样式：

语法讲解：

```
StyleManager.unloadStyleDeclarations("style.swf",true);
```

第 1 个参数：样式文件地址

第 2 个参数：是否更新，默认 true，true 可以不写。

这样代表刚才加载的样式被卸载了，所有组件恢复原貌。

第 2 个参数的 true 代表马上更新程序样式，false 代表不立即更新

```
StyleManager.loadStyleDeclarations("style.swf",false);
```

代表载入 style.swf 以后，不要立即更新主应用程序的样式

好了，马上讲今天第 3 个课题了

我现在只介绍一下 ApplicationDomain 的用途：

简单讲，如果会用 ApplicationDomain 了，我们可以做以下事情——

1、主应用程序 SWF 可以直接调用模块 SWF 中的所有 public 变量和方法。实现模块和主程序完全相互合作

2、Flex 2 的 SWF 可以和未来 Flash 9 生成的 SWF 用到一起，两者可以互相通讯，跟模块一样。

不论你把 Flex 还是 Flash 当作主应用程序，不论是谁加载谁，他们彼此都可以直接访问彼此的 public 变量和方法。

基本也就这 2 个了，ApplicationDomain 是整个 AS 3.0 客户端最核心的类，如果学会了它，就可以做很多不可思议的事情了

我简单讲

```
public class a {  
    public var str:String;  
    public function do():void {  
        str="1234";  
    }  
}  
  
public class b {  
    public var b1:Button;  
    public function dosth():void {  
        b1.label="按钮";  
    }  
}
```

假设 A 是主应用程序，B 当作模块被 A 加载

如果你会 ApplicationDomain

那么

在 A 中可以直接访问 B 中的 变量 b1 和 方法 dosth()

B 中也可以直接访问 A 中的 变量 str 和 方法 do()

注意哦，我说的是在 a 编译成 a.swf 和 b 编译成 b.swf 以后
a.swf 动态加载 b.swf，他们直接可以互相调用。调用，访问都一样。

作者：Dason