关于 flex 事件的讲解

一．引

很多新人对 Flex 的事件机制都不太熟悉，在使用过程中难免会出现各种问题，这是一个非常普遍的问题，为了更快更好的帮助大家，将介绍一下 Flex 中事件的各种机制和用法。

Flex 的精髓之一就是事件和绑定机制，了解之后，能帮助大家更灵活的设计程序，也对新手上路有一定的帮助。

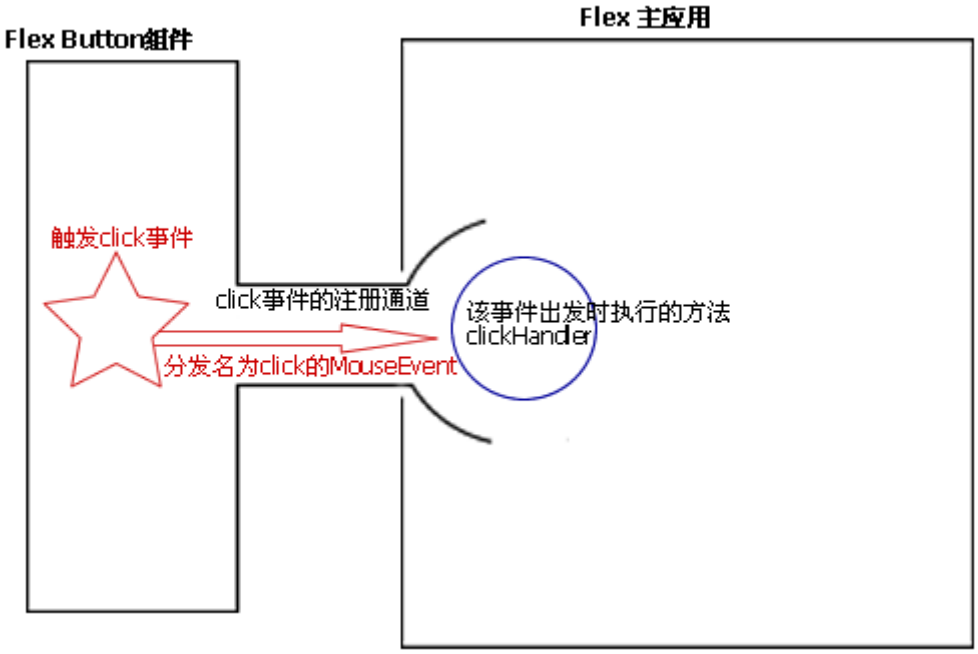
讲解可能不太系统，也不全面，有很多没有深入。如果高手看到后有疑问，欢迎指正。当然各位也可以提出自己的看法，或者经验分享，谢谢。

二．事件机制介绍

1. 什么是事件机制

事件可以看作是一种触发机制，当满足了一定的条件后，会触发这个事件。比如 MouseEvent 就是指的当鼠标进行操作之后触发的一系列的事件。很多控件中都有 click 事件，这个事件就是一个 MouseEvent 的实例，当点击鼠标后，系统会自动抛出一个名称为 click 的 MouseEvent 事件（这种方法我们将在后面介绍到）。如果此时在 click 上注册一个方法，那么触发该事件时就会执行这个方法。

大致示意图

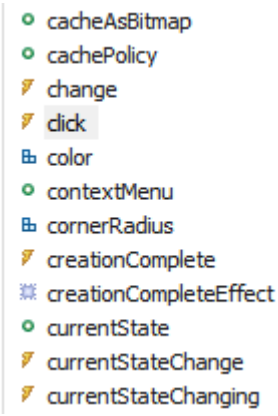


该示意图对应的 Flex 主应用的 mxml 代码

```
<mx:Script>
 <![CDATA[
      import mx.controls.Alert;
      private function clickHandler(e:MouseEvent){
          Alert.show(e.currentTarget.toString());
```

```
    }
]]>
</mx:Script>
<mx:Button id="testBtn" click="clickHandler(event)" label="测试">
</mx:Button>
```

在我们写代码时，编辑器的代码补全提示列表中，有很多不同的图标，如图



那些带有闪电的就是事件，三个小块的就是样式，空心圆圈的是属性，实心圆点的是公有方法，还有一个是效果。

我们能在这个列表中看到的事件，我把它称之为事件注册通道。（官方仍然称它为事件，但是它又和普通的事件含义不同。关于事件注册通道会再下面讲述到）

2. 事件注册通道

上面说到了，这些通道是只能在 mxml 的代码提示中可以看到的，他的作用就是给 mxml 组件提供 事件触发时所执行的方法的注册通道，而且能在代码提示中可见，这样给组件提供了很大的抽象的好处，我们可以很清楚的告诉组件的使用者，组件里包含哪些事件给你调用。

为什么把他区别对待？除了代码提示外，他还有一些实现上的不同。
Button 的 click 事件是继承自核心类 InteractiveObject，遗憾我们看不到他的源码，但是说明了“事件注册通道”是可以继承的。

我们会在自定义事件中讲述到如何声明“事件注册通道”。

3. 事件触发方法

注册通道中如果填入了函数，那么就代表触发该事件时，会执行这个方法。
click="clickHandler(event)"
我们看到这个方法有一个 event 对象作为参数传入，新人可能会问到，这个 event 对象哪里来的？我也没声明这个变量啊。他实际上是注册通道传给他的，默认变量名就是 event。我们如果想在事件触发时传其他的参数，可以通过自定的事件对象来实现。

这个对象就是这个组件分发的事件对象，即 name 为“ click ”的 MouseEvent 的一个实例。

这个 event 对象包含了触发该事件时的各种信息，比如触发对象是哪个，触发时鼠标点在哪里等等，不同的 event 类会包含不同的属性，比如 KeyboardEvent 包含了键盘点击了哪个键。

我们也可以通过自定义一个事件类，来传递我们自己想要的各种信息。（这在后面将介绍到）

4. 事件分发（重点了）

最终继承自 EventDispatcher 的对象都会含有 dispatchEvent 这个方法，他有一个参数，事件对象。

之前说到的事件注册通道，他只是一个通道，实际上事件是由这个方法来分发出去的，通道只是一个管道而已。

他的作用就是分发一个事件对象，他的分发是没有目的的，一种广播形式的，Flex 的事件监听线程会接收到各种各样的事件（我们称之为捕获事件，这在后面会介绍到），那么哪种才是你要的事件，标识就通过 name 来区分。

1）事件对象

在分发事件时，将会分发一个事件对象出去。不管是那个事件类，都是继承自 flash.events.Event 对象的，他包含一些比较重要的属性，name 和 bubbles。

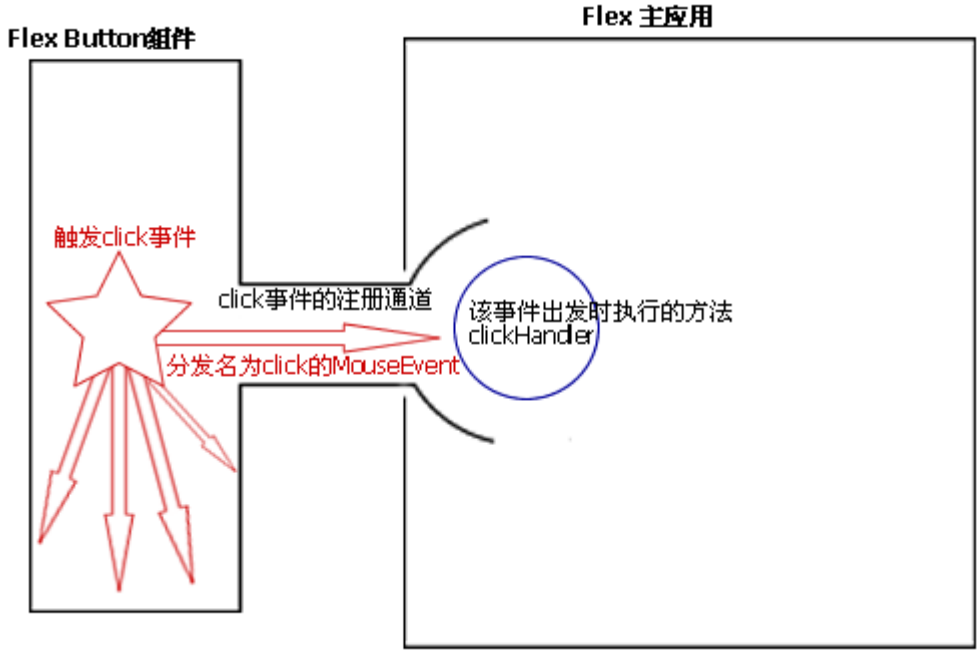
Name 是这个对象在被捕捉到时的凭证。

Bubbles 是个布尔值，决定了该对象是否会向上传递。什么意思呢？画个图就明白了。

比如说，当 button 组件分发 click 事件对象时，设置的 bubbles 为 false，那么他的分发是这样的

示意代码

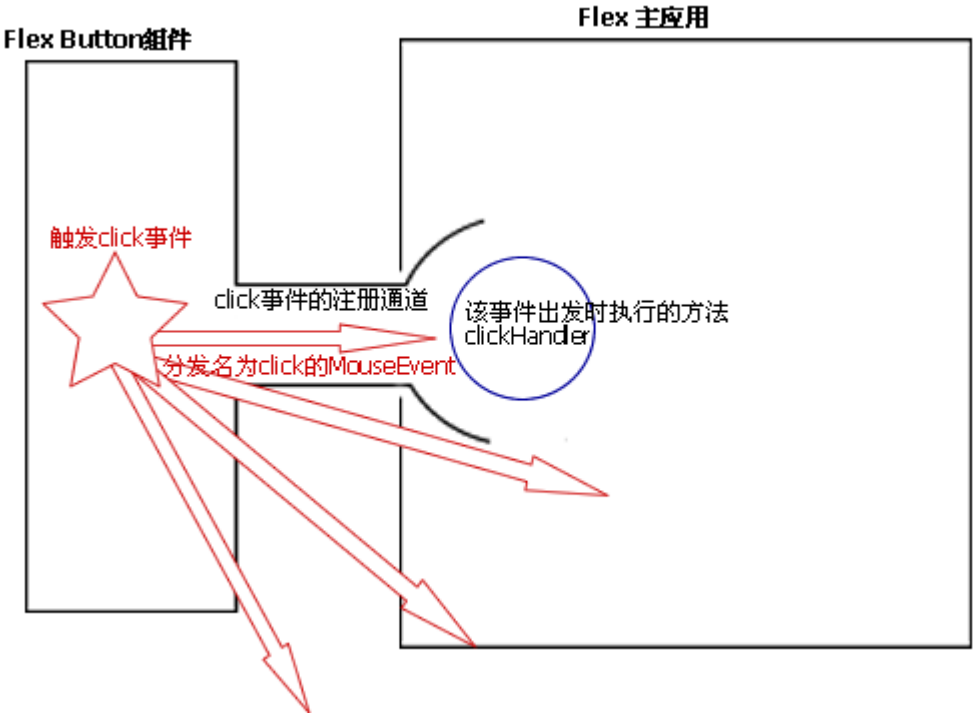
```
dispatchEvent(new MouseEvent( “click” , false ));
```



事件对象无法跨越组件本身，当然，除了之前讲到的注册通道（这样就很形象了吧）因此，如果没有注册通道，在 Flex 主应用中，就无法捕获到这个 button 组件分发出的事件。

如果我们将 Bubbles 设为 true，他看起来就是这样

```
dispatchEvent(new MouseEvent( "click" , true ));
```



可以看到，这个事件可以跨过组件本身，到达 Flex 主应用里。不止这样，在帮助手册中明确说到，如果在传递过程中间一直没有被捕获的话，这个事件会逐层上传，直到最终的 stage，那时如果还没被捕获，这个事件就会被销毁掉。

这样一来，即使我们没有 click 的事件通道，只要我们在 Flex 主应用中添加事件监听器（addEventListener）那么我们就可以获得到这个分发出的 click 事件了。

那么，注册通道不是没用了吗？不是，之前说到过，注册通道是现式的，可见的，因此如果你的组件要给其他人使用，那么就非常一目了然，而不必知道你源码中究竟分发了什么事件。但是，不要监听和注册同一个事件，这样会重复执行的。（后面将讲到）

5. 事件监听

在分发中，我们讲到，如果不是通过注册通道来调用触发事件，那么我们是需要一个监听来捕捉的。如何捕捉到分发出的事件，就是通过事件的 name 值。

比如:

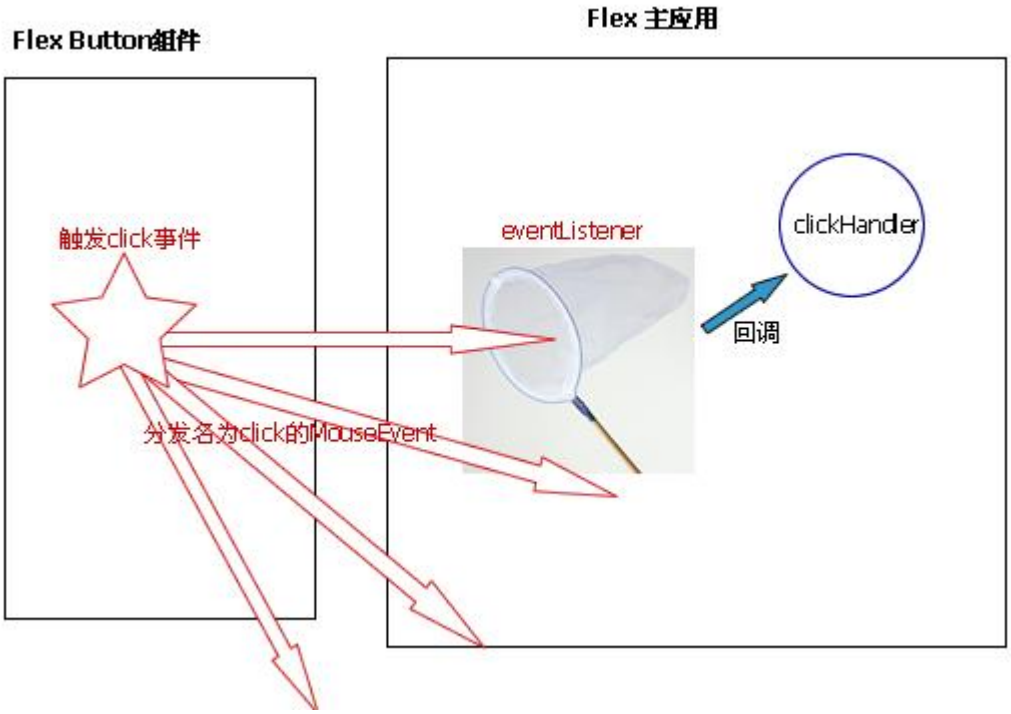
```
<mx:Application xmlns:mx=http://www.adobe.com/2006/mxml
        layout="absolute" xmlns:components="components.*"
        creationComplete="init()"
        >
        <mx:Script>
        <![CDATA[
          private function init(){
            testBtn.addEventListener("click", clickHandler);
          }
```

Flex 的事件中都提供了一些静态常量，让我们调用，避免我们打错了。因此这句话可以这么写

```
testBtn.addEventListener(MouseEvent.CLICK,clickHandler);
```

我们看到，监听的回调方法中没有传递参数，是的，这和通道的写法有些不同，这里的回调方法（即 clickHandler）只是个引用，并不是代表方法的执行，他的含义是，告诉 eventLinstener，如果捕捉到 click 事件，那么就去找 clickHandler，并执行它，event 对象参数在执行时动态的传递。（如果熟悉 ajax 的朋友这里应该很容易懂了）

他作用起来就是这样



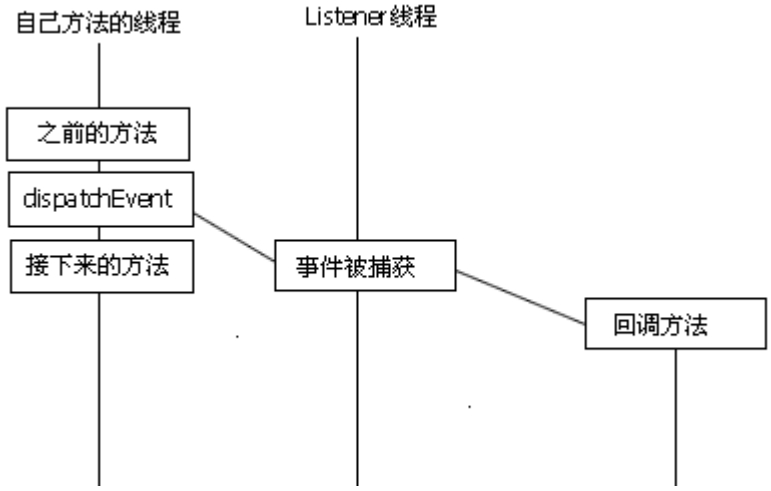
如果你又注册了 click 的事件通道，那么这两个都会生效，显然这是多余的。

6. 关于异步和执行顺序

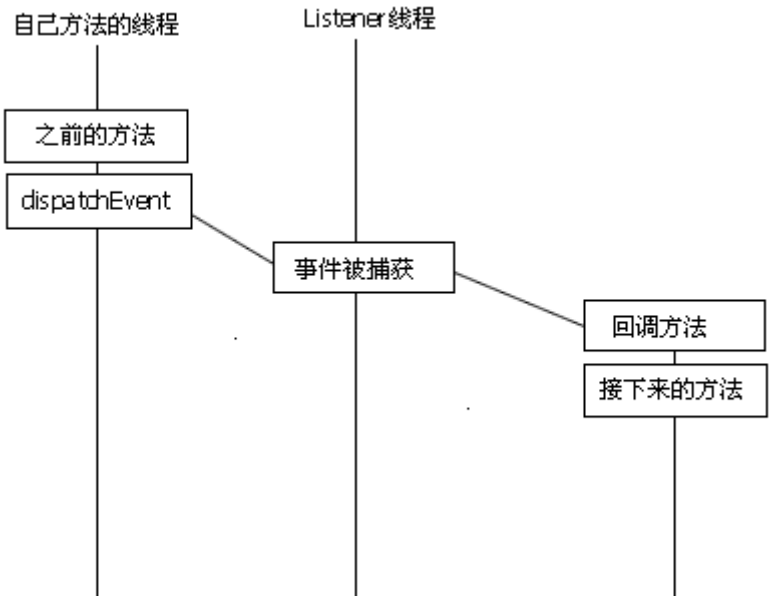
可以看到，事件的分发，和事件捕获的回调（clickHandler）并不是同步的，因为在 dispatchEvent 之后，该线程不会停下来，等待 listener 捕捉，而是继续做自己的事。

当你分发了 click 事件后，仍然会继续执行 click 之后的操作，Lintenser 是一个独立的线程，他至始至终都在工作，当他捕捉到你 add 的事件后，则会去调用回调的函数。分发任何事件都是这样。

异步示意图



上图可以看出，回调方法执行的顺序甚至还不如 dispatchEvent 之后的方法。如果接下来的方法依赖于事件回调，那么把接下来的方法写到回调方法中去



三．绑定机制

在我们了解了事件机制后，那么理解绑定就不难了。绑定其实也是事件机制的运用

1. 什么是绑定

绑定的作用在于，将 Flex 中的变量、类、方法等与组件的值进行绑定。例如，一个变量如果被绑定后，那么引用该变量的组件的相关属性也会发生改变。我们用一个实例来表示

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx=http://www.adobe.com/2006/mxml
layout="absolute" xmlns:components="components.*"
>
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;

        [Bindable]
        private var isSelected:Boolean;

        private function clickHandler(e:MouseEvent){
            //Alert.show(e.currentTarget.toString());
            isSelected=isSelected?false:true; //这句话的意思是如果
            isSelected为true，改变它为false，如果它为false，改变它为true；
            Alert.show(isSelected.toString());
        }
    ]]>
</mx:Script>
    <mx:Button id="testBtn"   click="clickHandler(event)" label="测试" />
    <mx:CheckBox x="60" selected="{isSelected}" />
</mx:Application>
```

上述程序的效果就是，当点击 button 时，button 不是直接改变 checkbox 的选中状态，而是改变 isSelected 这个变量，由于 isSelected 是被绑定了的，那么会关联的改变 CheckBox 的选中状态。

这样看起来有些多此一举，完全可以直接改变 checkbox 的 selected 属性，我只是为了演示一下效果。如果说你的 checkbox 是动态构造的上百个，你不会去一个个的改变他吧。

因此，我们多数会将一个数据源进行绑定声明，这样引用了这个数据源的控件，比如 datagrid，在数据源发生了改变时，即使你不重新设置 dataProvider，列表的数据也会刷新。当然，还有很多应用等待你去尝试。

如果这个代码中取消了[**Bindable**]的声明，会怎么样？isSelected不会改变了吗？
isSelected会改变，我们alert出来的结果也会显示结果改变了，但是checkbox的选择状态不会改变，因为当一个组件由创建到最终显示出来时是经过很多方法的 ,比如addChild ,commitProperties ,updateDisplayList等 ,updataDisplayList则是类似刷新显示效果一样的方法。
仅仅改变属性，而不去更新显示效果那么组件不会因为属性的改变而发生任何变化。

绑定的原理也是利用的事件分发。更复杂的绑定有待你去自己发现了

四. 自定义事件的分发

这部分就不长篇大论了，因为各位应该已经掌握了事件的原理，因此贴出演示源码，并进行些简单的解释。

1. 自定义事件 components/MyEventTest.as

```
package components
{
    import mx.events.FlexEvent;
    public class MyEventTest extends FlexEvent
    {
        public static const ONCHANGE:String = "onChange";
        public var eventInfo:String; //自定义的事件信息
        public function    MyEventTest(s:String){
            super(s); //如果在构造时不设bubbles，默认是false，也就是不能传递的。
            eventInfo="这个事件是:"+s;
        }
    }
}
```

2. 自定义组件 components/ComponentForEvent.as

```
package components
```

```
{
    import flash.events.EventDispatcher;

    //这个就是声明事件注册通道的方法了。Type是该事件的类
    [Event(name="onChange", type="components.MyEventTest")]

    public class ComponentForEvent extends EventDispatcher
    {
        private var name:String;
        public function changeName(newName:String){
            this.name=newName;
            dispatchEvent(new MyEventTest(MyEventTest.ONCHANGE) );
        }
    }
}
```

3. App.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" xmlns:components="components.*"
    >
    <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      private function   changeName(){
        cfe.changeName("新名称");
      }
    ]]>
    </mx:Script>
    <mx:Button id="testBtn"   click=" changeName ()" label="测试" />
    <components:ComponentForEvent
        id="cfe" onChange="{Alert.show(event.eventInfo)}"   />

</mx:Application>
```