事件

北京理工大学计算机学院
金旭亮

理解“事件”

Windows操作系统是“**事件驱动**”的。

在“事件驱动”的软件系统中，符合某种预设条件的情形出现时，**特定的事件被触发**。

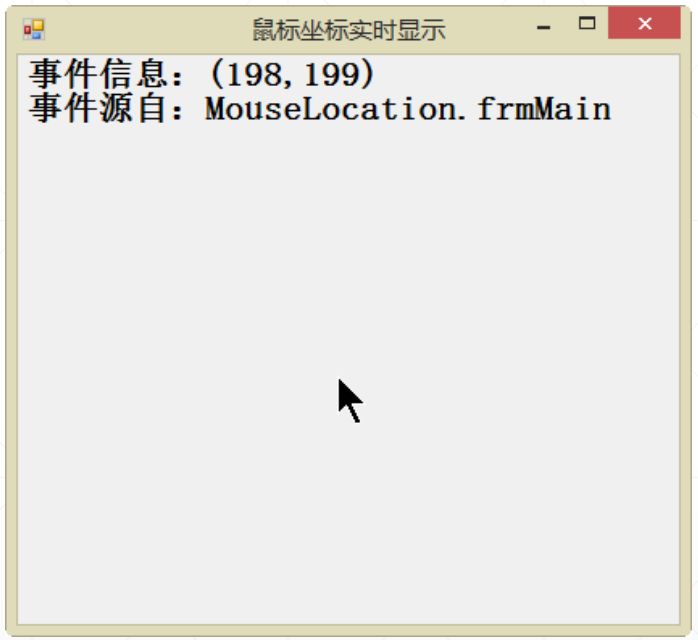
比如，某Windows桌面应用中，用鼠标点击一个按钮，将会触发一个“**Click**”事件；而鼠标在窗体上移动，则触发“**MouseMove**”事件。

“事件”三要素

1. **事件源**，即激发事件的对象；
2. **事件信息**，即事件本身所携带的信息；
3. **事件响应者**，指响应事件的代码，这些代码决定了当事件发生时，需要计算机完成的工作。

下面我们通过一个实例直观地展示事件的三个要素……

通过实例理解事件三要素



MouseLocation示例

事件源对象

事件携带的信息

```
1 reference
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    lblInfo.Text = string.Format(
        "事件信息：({0},{1}) \n事件源自：{2}",
        e.X, e.Y,sender.GetType());
}
```

响应MouseMove事件的响应者

MouseLocation示例是“事件驱动”的

“事件驱动”的软件**运行**方式：

当某个事件触发时，程序员事先写好的响应此事件的代码被调用。

“事件驱动”的软件**开发**方式为：

1. 定义并实现自己的事件，或者是从系统组件库中选择一种现成的事件；
2. 为这一事件编写响应代码。

如何设计并实现一个支持自定义事件的软件系统？

事件的主要特点为***一对多关联***，即一个事件源，可以有多个响应者。

委托与它所引用的方法也具有这种***一对多关联***的特性，因此我们是不是可以……

把事件看成是一个多路委托变量，而事件的响应者就是被此多路委托变量所引用的那些方法。

利用委托实现自定义事件-1

```
//定义事件委托
public delegate void MyEventDelegate(int value );
```

委托的参数，代表事件携带的信息

```
//事件发布者类
2 references
public class  Publisher
{
    //利用多路委托变量保存多个事件响应者方法引用
    public MyEventDelegate MyEvent;
}
```

事件发布者就是“事件源”。

```
//事件响应者类
4 references
public class Subscriber
{
    //事件触发时的回调方法
    2 references
    public void MyMethod(int value )
    {
        Console.WriteLine(value);
    }
}
```

事件触发时，此方法被调用。注意它必须满足事件委托的要求。

利用委托实现自定义事件-2

```
 //一个事件源对象
Publisher p = new Publisher();
//两个事件响应者
Subscriber s1 = new Subscriber();
Subscriber s2 = new Subscriber();

//挂接事件响应方法
p.MyEvent += s1.MyMethod;
p.MyEvent += s2.MyMethod;

//直接调用委托变量，代表触发事件
p.MyEvent(new Random().Next(1,100));
```

这个示例使用多路委托“模拟”实现了“事件的触发与响应”过程

再往前走一步，我们就接近.NET Framework中真正的事件实现机制了……

示例：UseMultiDelegateExample

C#引入关键字event用于自定义事件

```
//事件发布者类
2 references
public class Publisher
{
    //使用C#的event关键字定义一个事件
    public event MyEventDelegate MyEvent;
    //激发事件
    1 reference
    public void FireEvent(int EventArgu)
    {
        if (MyEvent != null)
        {
            MyEvent(EventArgu);
        }
    }
}
```

事件只能由事件发布者“视情况”而触发……

```
//一个事件源对象
Publisher p = new Publisher();
//两个事件响应者
Subscriber s1 = new Subscriber();
Subscriber s2 = new Subscriber();

//挂接事件响应方法
p.MyEvent += s1.MyMethod;
p.MyEvent += s2.MyMethod;

//直接调用委托变量，代表触发事件
p.FireEvent(new Random().Next(1,100));
```

示例：UseEventExample

event 关键字的功用

使用event关键字实现自定义事件和使用多路委托的实现方法非常类似，不同之处在于关键字event增添了一个新的特性：

事件只能由事件源对象自己激发，
外界无法通过访问委托变量直接激发事件。

.NET 事件揭秘

场景：

当在Windows Forms应用程序中给某个控件的事件编写了事件响应方法时，如果又在代码编辑器中删除了这个方法，项目编译将会失败……

“Click事件”探秘之旅

Click事件的定义如下：

```
public event EventHandler Click;
```

Click事件的响应方法必须符合EventHandler委托的要求：

```
public delegate void EventHandler(
        Object sender, EventArgs e
)
```

事件源

事件信息

.NET事件的奥秘，水落石出！

在frmMain.Designer.cs中，我们可以找到这样的事件响应挂接代码.....

```
1 个引用
private void InitializeComponent()
{
    this.btnClickMe = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // btnClickMe
    //
    this.btnClickMe.Location = new System.Drawing.Point(58, 30);
    this.btnClickMe.Name = "btnClickMe";
    this.btnClickMe.Size = new System.Drawing.Size(168, 42);
    this.btnClickMe.TabIndex = 0;
    this.btnClickMe.Text = "Click Me！";
    this.btnClickMe.UseVisualStyleBackColor = true;
    this.btnClickMe.Click += new System.EventHandler(this.btnClickMe_Click);
```

得到结论

.NET事件触发与响应机制确实是建立在委托之上的。

理解了事件的奥秘，就可以灵活应用于实际开发中.....

运行时设定事件的响应函数

```
public partial class frmMain : Form
{
    1 reference
    public frmMain()
    {
        InitializeComponent();
        //运行时才设定事件响应函数
        button1.Click += new EventHandler(button1_Click);
    }

    1 reference
    void button1_Click(object sender, EventArgs e)
    {
        MessageBox.Show("I'm Clicked!");
    }

}
```

再看一个例子……

示例： UseEventHandler

示例： UseEventHandler2

示例关键点：

程序运行时，每次单击“借钱”按钮，都会向“欠钱”按钮的Click事件追加一个事件响应方法。

接着再看下一个例子……

示例： ResponseToEvents

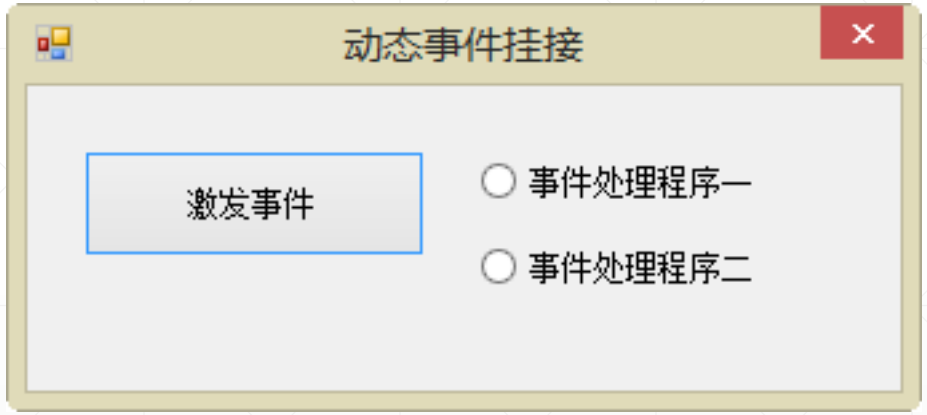
示例关键点：

此示例展示窗体对象的单个方法响应多个文本框对象的KeyDown事件。

向事件追加响应方法时，我们只需要将特定对象所触发的事件与特定的事件响应代码关联上就行了。对于这些方法和事件到底属于哪个对象，无需太在意。

有了前面的基础，我们不难实现以下功能：

在程序运行过程中动态设定事件响应方法



请自己动手实现上述示例程序的功能，遇到困难？
可参考： DynamicEventsInvoke