第 6 章

路由事件——绝情谷底玉蜂飞

> 黄蓉凝目看去，只见那两只玉蜂双翅上也都有字。那六个字也是一模一样，右翅是“情谷底”；左翅是“我在绝”。黄蓉大奇，暗想：“造物虽奇，也绝造不出这样一批蜜蜂来之理。其中必有缘故。”……
>
> 黄蓉不答，只是轻轻念着：“情谷底，我在绝。情谷底，我在绝。”她念了几遍，随即省悟：“啊！那是‘我在绝情谷底’。是谁在绝情谷底啊？难道是襄儿？”心中怦怦乱跳……
>
> ——《神雕侠侣》,“第三十八回 生死茫茫”[1]

这一段讲的是小龙女深陷绝情谷底，用花树上的细刺在玉蜂翅上刺下“我在绝情谷底”6 个字，盼望玉蜂飞上之后能为人发现。结果蜂翅上的细字被周伯通发现，而被黄蓉隐约猜到了其中含义。

本章内容如下。

（1）从玉蜂说起，回顾.NET 事件模型。
（2）什么是路由事件？
（3）CLR 事件足够完美，为什么还需要路由事件？
（4）言归正传，话路由事件。
（5）路由事件的实例。
（6）接下来做什么。

6.1 从玉蜂说起，回顾.NET 事件模型

木木熟悉神雕侠侣的故事，他根据“玉蜂传信”信手画了一幅有趣的图，如图 6-1 所示。

其实这幅“玉蜂传信图”暗合.NET 的事件模型，小龙女是事件的发布者，她发布了事件“我在绝情谷底”。老顽童和黄蓉是事件的订阅者，不过他并没有处理该事件，而黄蓉处理了事件，隐约能猜出其中含义。至于可怜的小杨过，则根本没有订阅事件，只是苦苦念叨“龙儿，龙儿，你在哪儿……”而玉蜂正是传递信息的事件，事件、事件的发布者和事件的订阅者构成了.NET 事件模型的 3 个角色。在.NET 中，一个事件用关键字 event 来表示。如代码 6-1 所示。

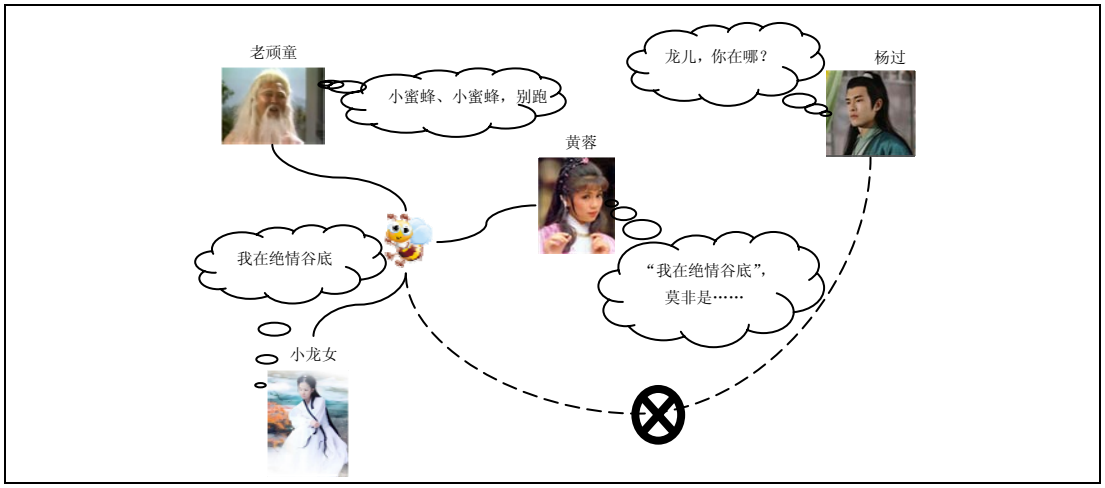


图 6-1　玉蜂传信图

```
        public delegate void WhiteBee(string param); //声明了玉蜂的委托
        // 小龙女类
        class XiaoLongnv
        {
            public event WhiteBee WhiteBeeEvent;    //玉蜂事件
            public void OnFlyBee()
            {
                Console.WriteLine("小龙女在谷底日复一日地放着玉蜂，希望杨过有一天能看
到……");
                WhiteBeeEvent(msg);
            }
            private string msg = "我在绝情谷底";
        }

        // 老顽童类
        class LaoWantong
        {
            public void ProcessBeeLetter(string msg)
            {
                Console.WriteLine("老顽童：小蜜蜂、小蜜蜂，别跑");
            }
        }

        // 黄蓉类
        class Huangrong
        {
            public void ProcessBeeLetter(string msg)
            {
                Console.WriteLine("黄蓉：\"{0}\",莫非......",msg);
            }
        }

        // 杨过类
        class YangGuo
        {
            public void ProcessBeeLetter(string msg)
            {
                Console.WriteLine("杨过：\"{0}\",我一定会找她！", msg);
            }
```

```
    public void Sign()
    {
        Console.WriteLine("杨过叹息：龙儿，你在哪儿……");
    }
}
static void Main(string[] args)
{
   // 第 1 步 人物介绍
   XiaoLongnv longnv = new XiaoLongnv();   //小龙女
   LaoWantong wantong = new LaoWantong();  //老顽童
   Huangrong rong = new Huangrong();       //黄蓉
   YangGuo guo = new YangGuo();           //杨过

   // 第 2 步 订阅事件,唯独没有订阅杨过的 ProcessBeeLetter;
   longnv.WhiteBeeEvent += wantong.ProcessBeeLetter;
   longnv.WhiteBeeEvent += rong.ProcessBeeLetter;
   // longnv.WhiteBeeEvent += guo.ProcessBeeLetter; //杨过没有订阅小龙女的玉蜂事件

   // 第 3 步 小龙女玉蜂传信
   longnv.OnFlyBee();

   // 第 4 步 杨过叹息
   guo.Sign();
}
```

代码 6-1　详见 mumu_whitebee 工程

程序运行结果如图 6-2 所示。

图 6-2　运行结果

不过这种事件看起来不像.NET 的事件，于是木木改写为如代码 6-2 所示的代码。

```
// 新增一个 WhiteBeeEventArgs 类
public class WhiteBeeEventArgs : EventArgs
{
    public readonly string _msg;
    public WhiteBeeEventArgs(string msg)
    {
        this._msg = msg;
    }
```

```
        }

        public delegate void WhiteBeeEventHandler(object sender,WhiteBeeEventArgs
e); //声明了玉蜂的委托
        // 小龙女类
        class XiaoLongnv
        {
            public event WhiteBeeEventHandler WhiteBeeEvent;    //玉蜂事件
            public void OnFlyBee()
            {
                Console.WriteLine("小龙女在谷底日复一日地放着玉蜂，希望杨过有一天能看
到……");
                WhiteBeeEventArgs args = new WhiteBeeEventArgs(msg);
                WhiteBeeEvent(this,args);
            }
            private string msg = "我在绝情谷底";
    }

        // 老顽童类
        class LaoWantong
        {
            public void ProcessBeeLetter(object sender, WhiteBeeEventArgs e)
            {
                Console.WriteLine("老顽童：小蜜蜂、小蜜蜂，别跑");
            }
     }
        // 黄蓉类
        class Huangrong
        {
            public void ProcessBeeLetter(object sender, WhiteBeeEventArgs e)
            {
                Console.WriteLine("黄蓉：\"{0}\",莫非......",e._msg);
            }
     }

        // 杨过类
         class YangGuo
        {
            public void ProcessBeeLetter(object sender, WhiteBeeEventArgs e)
            {
                // 真的是龙儿吗？
    XiaoLongnv longnv = sender as XiaoLongnv;
                if(longnv != null)
                    Console.WriteLine("杨过：\"{0}\",我一定会找她！", e._msg);
            }
            public void Sign()
            {
                Console.WriteLine("杨过叹息：龙儿，你在哪儿……");
            }
    }

    static void Main(string[] args)
        {
          // 第1步 人物介绍
          XiaoLongnv longnv = new XiaoLongnv();   //小龙女
          LaoWantong wantong = new LaoWantong();  //老顽童
          Huangrong rong = new Huangrong();      //黄蓉
          YangGuo guo = new YangGuo();          //杨过
```

```
        // 第 2 步 订阅事件,唯独没有订阅杨过的 ProcessBeeLetter;
        longnv.WhiteBeeEvent += wantong.ProcessBeeLetter;
        longnv.WhiteBeeEvent += rong.ProcessBeeLetter;
        // longnv.WhiteBeeEvent += guo.ProcessBeeLetter; //杨过没有订阅小龙女的玉蜂事件

        // 第 3 步 小龙女玉蜂传信
        longnv.OnFlyBee();

        // 第 4 步 杨过叹息
        guo.Sign();
    }
```

代码 6-2　详见 mumu_netwhitebee 工程

上述代码中加粗部分是经过修改的，可以归纳为如下几点。

（1）委托类型的名称修改为以 EventHandler 结束，原型有一个 void 返回值并接受两个输入参数，即一个 Object 类型和一个 EventArgs 类型（或继承自 EventArgs）。

修改前	`public delegate void WhiteBee(string param);`
修改后	`public delegate void WhiteBeeEventHandler(object sender,WhiteBeeEventArgs e);`

（2）事件的命名为委托去掉 EventHandler 之后剩余的部分。

修改前	`public event WhiteBee WhiteBeeEvent;`
修改后	`public event WhiteBeeEventHandler WhiteBee;`

（3）继承自 EventArgs 的类型应该以 EventArgs 结尾。

修改前	无事件参数

修改后	`public class WhiteBeeEventArgs : EventArgs`
	`{`
	`  public readonly string _msg;`
	`  public WhiteBeeEventArgs(string msg)`
	`  {`
	`  this._msg = msg;`
	`  }`
	`}`

这样修改的目的不仅仅符合.NET 规范，而且带来了更大的灵活性。例如，如果杨过收到了小龙女的玉峰传信，则可以通过 sender 参数来判断是否真是小龙女，甚至还可以了解更多有关的细节：

```
public void ProcessBeeLetter(object sender, WhiteBeeEventArgs e)
{
   // 真的是龙儿吗?
XiaoLongnv longnv = sender as XiaoLongnv;
   if(longnv != null)
        Console.WriteLine("杨过：\"{0}\",我一定会找她！", e._msg);
}
```

6.2 路由事件的定义

什么是路由事件呢？MSDN 从功能和实现两种视角给出了路由事件的定义：

“Functional definition: A routed event is a type of event that can invoke handlers on multiple listeners in an element tree, rather than just on the object that raised the event.

“Implementation definition: A routed event is a CLR event that is backed by an instance of the RoutedEvent class and is processed by the Windows Presentation Foundation (WPF) event system.”

以 Button 的 Click 事件为例，该事件是个路由事件。可以通过 Reflector 查看 ButtonBase 的源码，如代码 6-3 所示。

```
    public abstract class ButtonBase : ContentControl, ICommandSource
        {
            // 路由事件的定义
    public static readonly RoutedEvent ClickEvent;

            // 传统的事件包装器
            public event RoutedEventHandler Click
            {
                add
                {
                    base.AddHandler(ClickEvent, value);
                }
                remove
                {
                    base.RemoveHandler(ClickEvent, value);
                }
            }
            // 事件的注册
            static ButtonBase()
            {
                ClickEvent       =       EventManager.RegisterRoutedEvent("Click",
RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(ButtonBase));
                    ……
            }
            ……
        }
```

代码 6-3　ButtonBase 的 Click 事件

同依赖属性一样，路由事件也需要注册，不同的是使用 EventManager.RegisterRoutedEvent 方法。

同依赖属性，用户不会直接使用路由事件，而是使用传统的 CLR 事件。有两种方式关联事件及其处理函数，在代码中，仍然按照原来的方法关联和解除关联事件处理函数（+=和-=），如代码 6-4 所示。

```
    Button b2 = new Button();
    // 关联事件及其处理函数
    b2.Click += new RoutedEventHandler(Onb2Click);

    //事件处理函数
    void Onb2Click(object sender, RoutedEventArgs e)
```

```
{
    //logic to handle the Click event
}
```

代码 6-4　关联事件和事件处理函数

WPF 实现的机制已经发生变化，传统的 CLR 事件关联和解除关联处理函数均通过委托来实现，而路由事件则通过 AddHandler 和 RemoveHandler 方法实现。二者在 UIElment 中定义，多数 WPF 的类需要继承自该类，如表 6-1 所示。

表 6-1　CLR 事件和路由事件关联和解除关联事件处理函数的差别

传统的 CLR 事件+=的背后	Delegate.Combine(……)
路由事件+=的背后	AddHandler(……)
传统的 CLR 事件-=的背后	Delegate.Remove(……)
路由事件-=的背后	RemoveHandler(……)

上面的代码也可以改写为代码 6-5：

```
Button b2 = new Button();
// 关联事件及其处理函数
b2.AddHandler(Button.ClickEvent, new RoutedEventHandler(Onb2Click));

// 事件处理函数
void Onb2Click(object sender, RoutedEventArgs e)
 {
    //logic to handle the Click event
 }
```

代码 6-5　关联事件和事件处理函数

在 XAML 中，事件和处理函数这样关联，如代码 6-6 所示。

```
<Button Click=" Onb2Click ">button</Button>
```

代码 6-6　在 XAML 中关联事件和事件处理函数

传统的事件触发往往直接调用其委托（因为事件的本质是委托），而路由事件则通过一个 RaiseEvent 方法触发，调用该方法后所有关联该事件的对象都会得到通知。在 ButtonBase 中即有代码 6-7：

```
RoutedEventArgs e = new RoutedEventArgs(ClickEvent, this);
base.RaiseEvent(e);
```

代码 6-7　ButtonBase 里触发事件

路由事件通过 EventManager.RegisterRoutedEvent 方法注册；通过 AddHandler 和 RemoveHandler 来关联和解除关联的事件处理函数；通过 RaiseEvent 方法来触发事件；通过传统的 CLR 事件封装后供用户调用，使得用户如同使用传统的 CLR 事件一样使用路由事件。

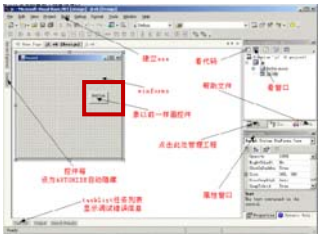
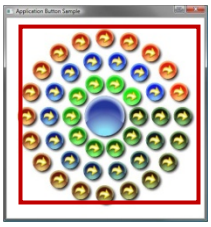
6.3 路由事件的作用

虽然木木已经明白了什么是路由事件，但是头脑中仍有一个很大的疑问：WPF 为什么引入路由事件，难道 CLR 事件就不能满足应用吗？这个还是得从 WPF 的界面元素的特点上说起。

以按钮为例，在 WinForm 时代一个按钮就是一个按钮；在 WPF 中一个按钮可以是其他部件。

表 6-2 所示为利用 Google 图片搜索功能分别用 WinForm Button 和 WPF Button 关键词搜索任意尺寸和任意类型的前几位图片（2009 年 7 月 4 日搜索结果，用框标识 Button）。

表 6-2　互联网上的 WinForm 和 WPF 按钮

WinForm			
WPF			

最直观的反应是 WPF 的按钮要比 WinForm 来得炫目，但绝不仅于此。WPF 中的几乎任何一个界面元素都可以任意嵌套，并且装配极其容易。如代码 6-8 所示。

```
        <Button Margin="50" Background="AliceBlue" BorderBrush="Black"
BorderThickness="2">
          <StackPanel >
            <TextBlock Margin="3" >
              Image and picture Button</TextBlock>
            <Image Source="happyface.jpg" Stretch="None" />
            <TextBlock Margin="3" >
              Courtesy of the StackPanel</TextBlock>
          </StackPanel>
        </Button>
```

代码 6-8　一个嵌套按钮的例子

程序运行结果和按钮的结构如图 6-3 所示。

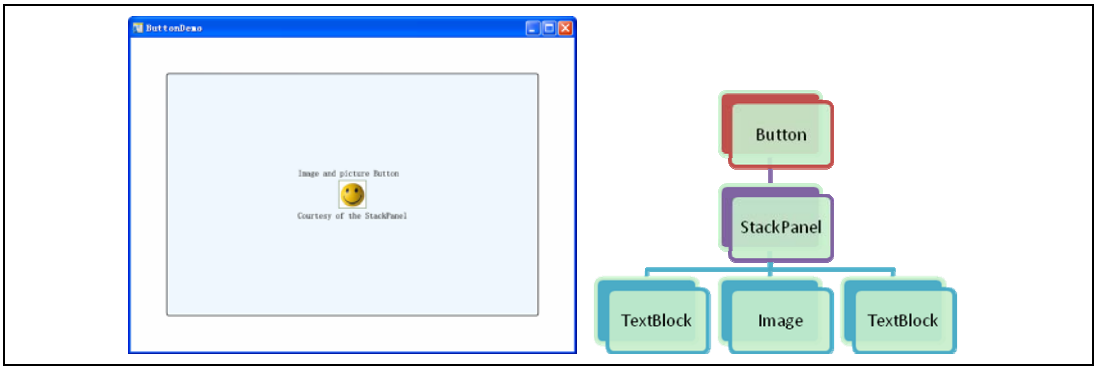


图 6-3　程序运行结果和按钮的结构

如果在图片（Image）上单击一下，应该哪个元素去响应这个点击事件呢，是图片？还是按钮呢？理想的情况当然是无论鼠标点击到图片，还是文本块（TextBlock），都应该由按钮来处理。使用传统的 CLR 事件，程序员势必要对所有元素的事件响应处理函数重写，而且每个事件处理函数最后都要调用按钮的事件处理函数，如图 6-4 所示。

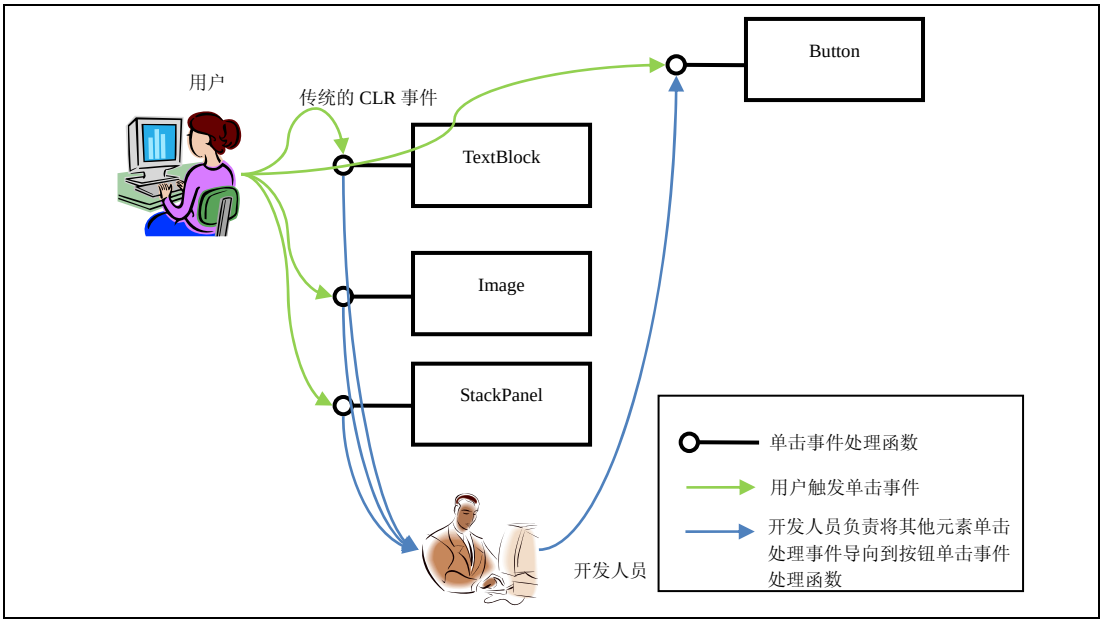


图 6-4　传统的 CLR 事件处理流程

路由事件则完全不同，它可以向上游走（Bubble）或向下沉底（Tunnel）。在这个例子中，用户单击到文本块（TextBlock）或者图片（Image）时事件可以经由 TextBlock/Image——StackPanel——Button 一路上浮，游走到按钮的事件处理函数中，而开发人员不必做任何工作，如图 6-5 所示。

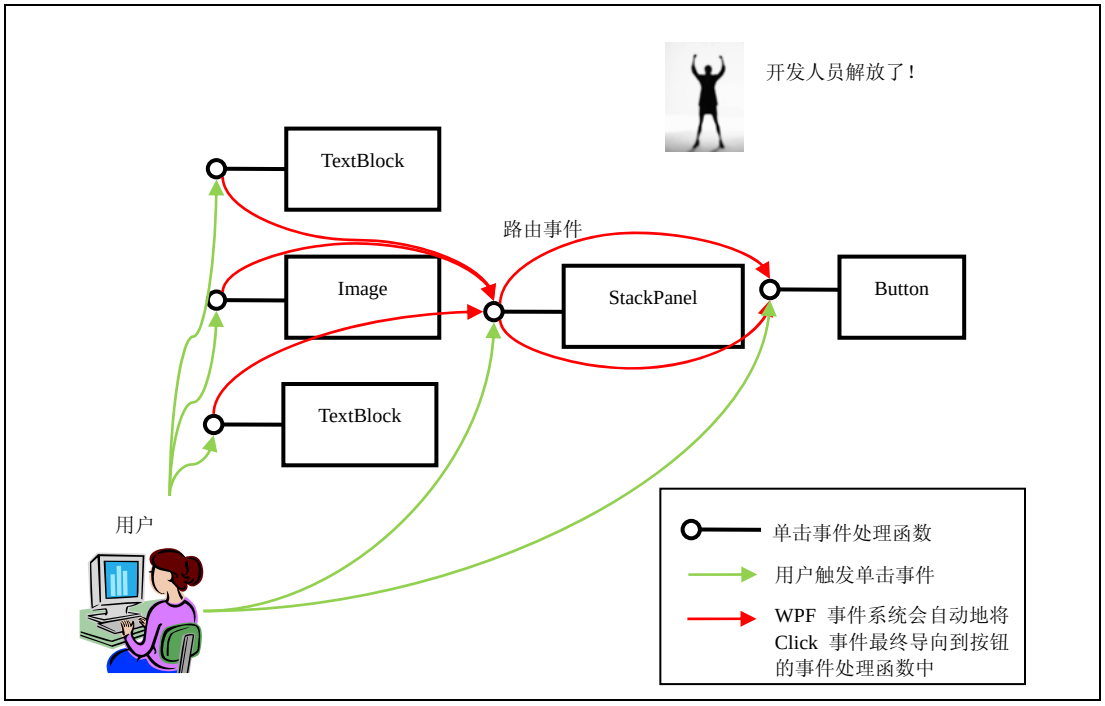


图 6-5　路由事件处理流程

路由事件的引入既能满足控件的任意组合，又能保证控件 Hit-test 的行为的完整性，何乐而不为呢？

6.4　路由事件

6.4.1　识别路由事件

和依赖属性一样，如果一个事件是路由事件，则在 MSDN 文档中会有路由事件（Routed Event Information）一节描述其路由信息。而普通的 CLR 事件（如 UIElement.IsVisibleChanged）则没有这一节信息，如图 6-6 所示。

图 6-6 描述的路由信息主要包括唯一标识 ClickEvent、事件委托形式 RoutedEventHandler，以及路由策略，Click 事件的路由策略是 Bubbling。

图 6-6　ButtonBase.Click 事件页面中包含 Routed Event Information

6.4.2 路由事件的旅行

1．路由事件的旅行策略

一个人在外旅行，沿途会休息，会遇到各种各样好玩的事情。路由事件的旅行相对要简单很多。归纳起来在路由事件的旅行当中，一般只出现两种角色：一是事件源，由其触发事件，是路由事件的起点；二是事件监听者，通常针对监听的事件有一个相应的事件处理函数。当路由事件经过事件监听者，就好比经过一个客栈，要做短暂的停留，由事件处理函数来处理该事件。

路由事件的策略有如下 3 种。

（1）Bubbling：事件从事件源出发一路上溯直到根节点，很多路由事件使用该策略。

（2）Direct：事件从事件源出发，围绕事件源转一圈结束。

（3）Tunneling：事件源触发事件后，事件从根节点出发下沉直到事件源。

上述三种策略都只能看做是路由事件的一个旅行计划，实际上当路由事件开始旅行的时候，由于事件监听者的干预，它的旅行计划会有所改变。这一点，实际上也很好理解，我们每次出游的时候都会有个预定的路线，但是事实上什么飞机晚点、黑店勒索、山洪暴发等诸如此类的事情会影响我们的预定路线，最终导致实际路线和预定路线有所偏差。

2．改变旅行策略因素之一——事件处理函数

一个最基本的路由事件处理函数的原型如代码 6-9 所示。

```
public delegate void RoutedEventHandler( Object sender, RoutedEventArgs e)
```

代码 6-9　路由事件处理函数原型

事件处理函数之间有微小差异，如鼠标事件的处理函数原型如代码 6-10 所示。

```
public delegate void MouseEventHandler(Object sender,MouseEventArgs e)
```

代码 6-10　鼠标事件的处理函数原型

这种事件处理函数有如下两个特点。

（1）返回原型为 void。

（2）有两个参数，第 1 个是一个 Object 类型的对象，表示拥有该事件处理函数的对象；第 2 个是 RoutedEventArgs 或者是 RoutedEventArgs 的派生类，带有其路由事件的信息。

RoutedEventArgs 结构包括 4 个成员变量，如表 6-3 所示。

表 6-3　RoutedEventArgs 结构

名称	描述
Source	表明触发事件的源，如当键盘事件发生时，触发事件的源是当前获得焦点的对象；当鼠标事件发生时，触发事件的源是鼠标所在的最上层对象
OriginalSource	表明触发事件的源，一般来说 OriginalSource 和 Source 相同，区别在于 Source 表示逻辑树上的元素；OriginalSource 是可视树中的元素。如单击窗口的边框，Source 为 Window；OriginalSource 为 Border
RoutedEvent	路由事件对象
Handled	布尔值，为 true，表示该事件已处理，这样可以停止路由事件

Handled 属性是改变路由事件旅行的“元凶”。一旦在某个事件处理函数中将 Handled 的值设置为 true，路由事件就停止传递。如：

```
private void ***_Click(object sender, RoutedEventArgs e)
{
   ……
   e.Handled = true;
}
```

一个事件被标记为处理，事件处理函数则不可处理该事件。但是也有例外，WPF 还提供了一种机制，即使事件被标记为处理，事件处理函数仍然可以处理，但是关联事件及其处理函数需要稍做处理。AddHandler 重载了两个方法，其中之一如下所示，需要将第 3 个参数设置为 true：

```
public void AddHandler(RoutedEvent routedEvent, Delegate handler, bool handledEventsToo)
```

换句话说，因事件标识为处理而终止只是一种假象，路由事件的旅行仍然在继续，只不过普通的事件处理函数无法处理它。

3．改变旅行策略因素之二——类和实例事件处理函数

事件处理函数有两种类型：一是前面所说的普通事件处理函数，称为“实例事件处理函数”（Instance Handlers）；二是通过 EventManager.RegisterClassHandler 方法将一个事件处理函数和一个类关联起来，这种事件处理函数，称为“类事件处理函数”（Class Handlers），其优先权高于前者。也就是说事件在旅行时，会先光临类事件处理函数，然后再光临实例事件处理函数。

4．路由事件的旅行图

木木做了一个奇怪的梦，醒后记录了梦里的内容。

（1）主角是个路由事件，渴望旅行。

（2）路由事件有 3 种旅行计划：一是北上观北国风光千里冰封（Bubbling）；二是南下赏江南秀色，小桥流水（Tunneling）；三是同城一日游（Direct）。

（3）路由事件旅行线路中有 4 种不同的客栈，即普通客栈（响应事件，但是不会把事件标记为“已

处理”的实例事件处理函数）、黑心客栈（响应事件，而且会把事件身上所带的钱和衣物全部抢走。标记事件为“处理”，以至几乎所有的客栈都不会再让事件住宿）、政府指定客栈（类事件处理函数，事件必须优先在该函数响应）和慈善客栈（即使事件已被标记为处理，一无所有，该客栈仍然会欢迎事件住宿）。

（4）事件在旅行时会有两种状态：未处理和已处理状态。事件默认时都是未处理状态，这个时候的事件就好比一个大款，每个客栈看到事件，都会热情地请事件留宿，而“大款”事件也从来不拒绝客栈。但是事件如果一不小心留宿黑心客栈，黑心客栈会抢走它所有的钱和衣物（标记为“处理”状态），这个时候再一路旅行，由于衣衫不整，事件只能走隐蔽的路线，所有的客栈也都会很冷漠地对待它，只有慈善客栈仍然欢迎事件留宿。

图 6-7 所示为路由事件的旅行北上图（Bubbling 策略）。

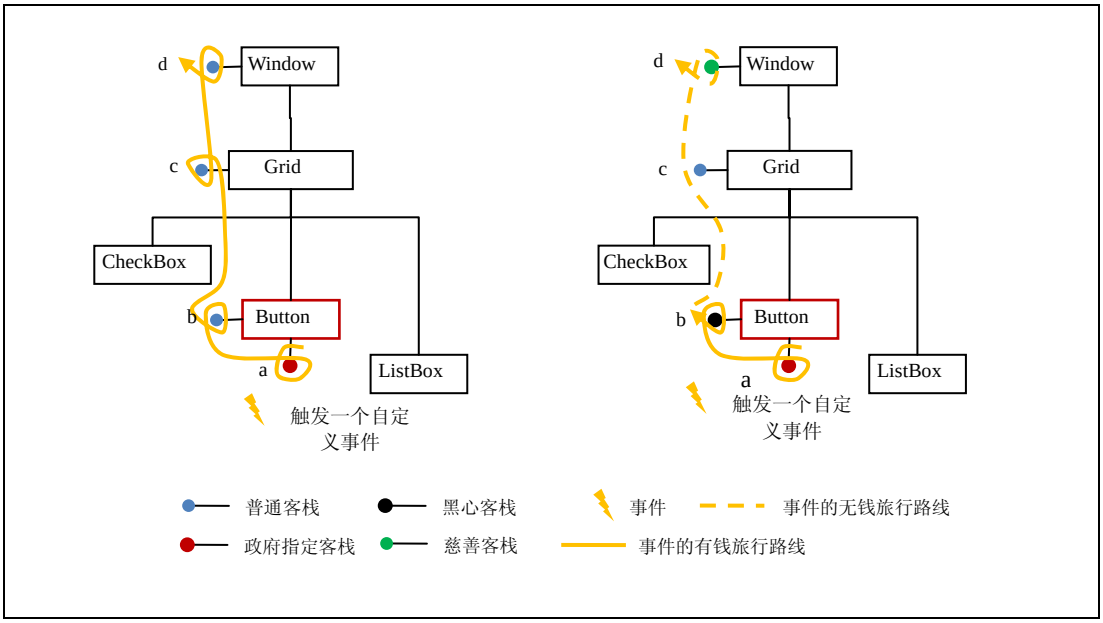


图 6-7　路由事件的旅行北上图

路由事件北上旅行之一从 Button 出发，首先在政府指定客栈停留（a），然后在 Button 的普通客栈（b）停留。沿途从 c 到 d 都是普通客栈，平安到达 Window。

路由事件北上旅行之二从 Button 出发，仍然在政府指定客栈停留（a），然后在 b 客栈停留。结果 b 是一个黑店，于是路由事件只好落荒而逃。客栈 c 不接待它，最后只有 d 这个慈善客栈接待了它。

图 6-8 所示为路由事件的旅行南下图。

路由事件旅行南下之一非常顺利，从 Window 的普通客栈（d）一路南下到 Button。

路由事件旅行南下之二出门不顺利，遇上黑店（d）。于是经过普通客栈（c）不能留宿，终于经过慈善客栈（b）留宿结束了旅行。

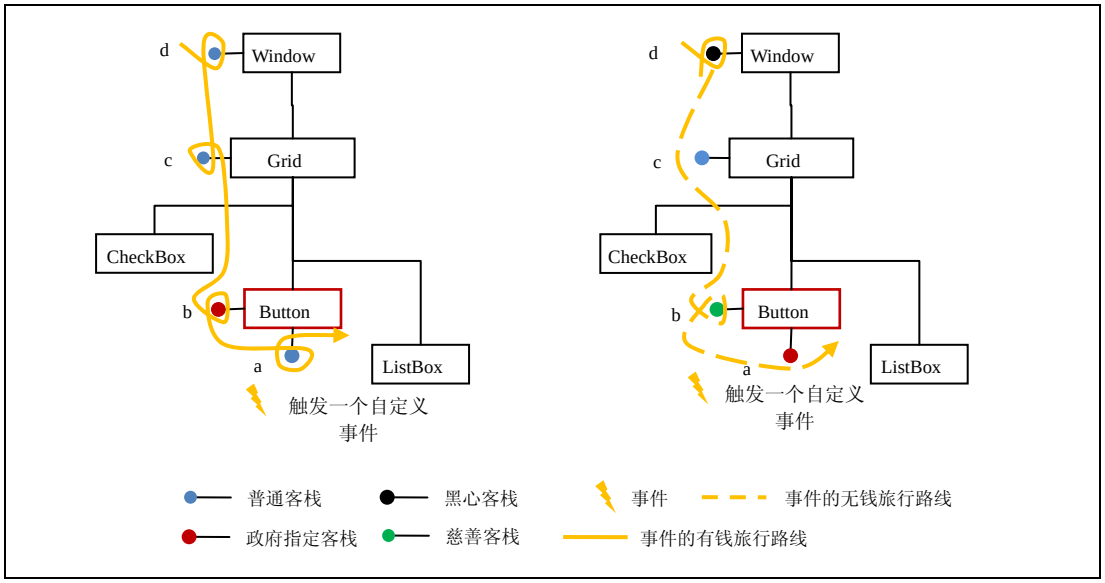


图 6-8　路由事件的旅行南下图

图 6-9 所示为路由事件的同城一日游。

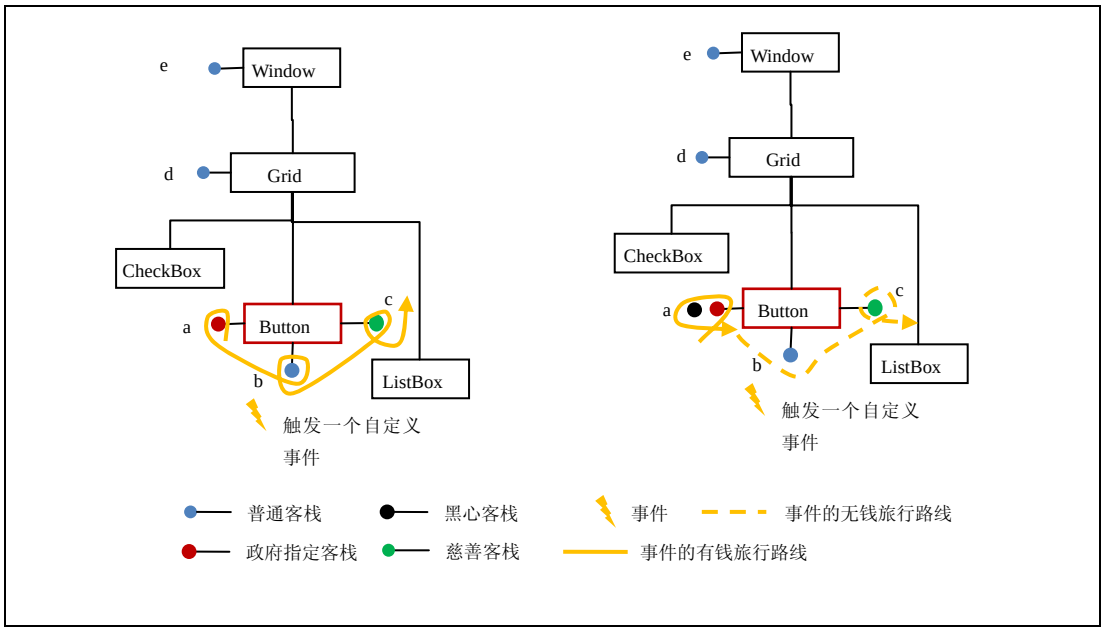


图 6-9　路由事件的同城一日游

路由事件的同城一日游之一出门顺利，在 Button 中一路经过政府指定客栈（a）、普通客栈（b）和慈善客栈（c）。

路由事件的同城一日游之二出门极其不顺利，路遇一个政府指定客栈。但是这个客栈明显辜负了政府的期望，是个黑店。于是普通客栈不欢迎路由事件，只有慈善客栈欢迎。

6.5 路由事件示例

自定义一个路由事件，名为“CustomClickEvent”。单击按钮时，这个事件就会触发为 Window，Grid 和 Button 装配不同的事件处理函数。然后单击按钮，观察路由事件的路由，如图 6-10 所示。

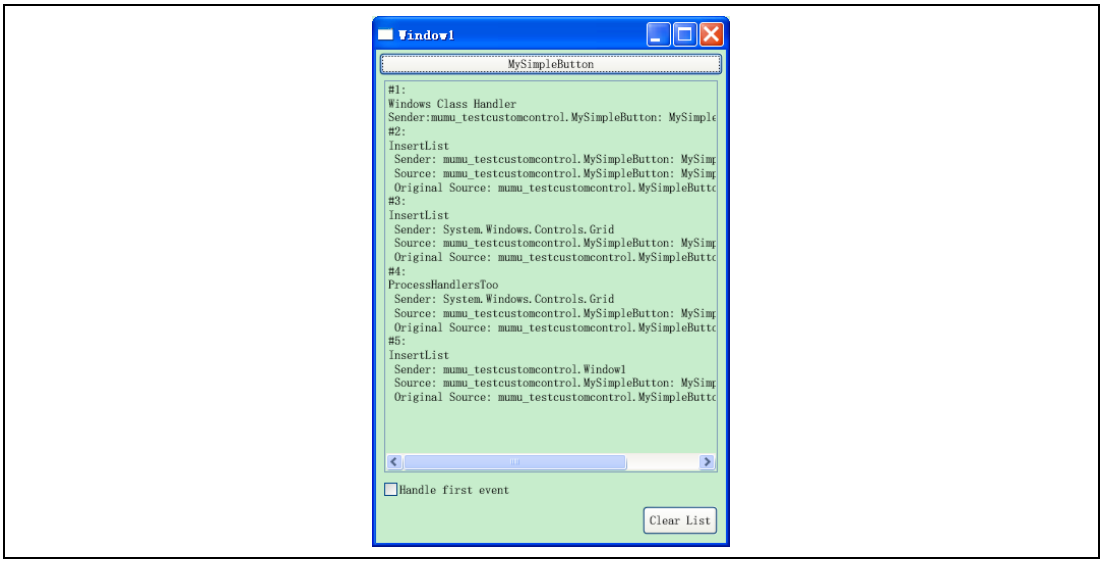


图 6-10　自定义路由事件

（1）需要继承一个按钮类，然后自定义 CustomClickEvent 的路由事件。其路由策略为 Bubble，如代码 6-11 所示。

MySimpleButton.cs

```
    public class MySimpleButton : Button
    {
        static MySimpleButton()
        {
        }
        // 创建和注册该事件，该事件路由策略为 Bubble
        public    static    readonly    RoutedEvent    CustomClickEvent    =
EventManager.RegisterRoutedEvent(
            "CustomClick", RoutingStrategy.Bubble, typeof(RoutedEventHandler),
typeof(MySimpleButton));
        // CLR 事件的包装器
        public event RoutedEventHandler CustomClick
        {
            add { AddHandler(CustomClickEvent, value); }
            remove { RemoveHandler(CustomClickEvent, value); }
        }
        // 触发 CustomClickEvent
        void RaiseCustomClickEvent()
        {
            RoutedEventArgs newEventArgs = new RoutedEventArgs(MySimpleButton.
CustomClickEvent);
            RaiseEvent(newEventArgs);
        }
        // OnClick 触发 CustomClickEvent
```

```
            protected override void OnClick()
            {
                RaiseCustomClickEvent();
            }
        }
```

代码 6-11　自定义一个路由事件

（2）设计个应用程序的界面，为 Window、Grid 和 MySimpleButton 关联相应的事件处理函数，如代码 6-12 所示。

```
    Window1.xaml
    <Window x:Class="mumu_testcustomcontrol.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            xmlns:custom ="clr-namespace:mumu_testcustomcontrol"
        Title="Window1"       Name="window1"       Height="300"       Width="300"
custom:MySimpleButton.CustomClick="InsertList">
        <Grid Margin="3" custom:MySimpleButton.CustomClick="InsertList" Name="grid1" >
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="*"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
                <RowDefinition Height="Auto"></RowDefinition>
            </Grid.RowDefinitions>
            <custom:MySimpleButton x:Name ="simpleBtn" CustomClick="InsertList">
                MySimpleButton
            </custom:MySimpleButton>
            <ListBox Margin="5" Name="lstMessages" Grid.Row="1"></ListBox>
            <CheckBox   Margin="5"   Grid.Row="2"   Name="chkHandle">Handle   first
event</CheckBox>
            <Button      Grid.Row="3"    HorizontalAlignment="Right"    Margin="5"
Padding="3" Click="cmdClear_Click">Clear List</Button>
        </Grid>
    </Window>

    Window1.xaml.cs
    public partial class Window1 : Window
        {
            public Window1()
            {
                InitializeComponent();
            }

            protected int eventCounter = 0;

            private void InsertList(object sender, RoutedEventArgs e)
            {
                eventCounter++;
                string message = "#" + eventCounter.ToString() + ":\r\n"+
                    "InsertList\r\n"+
                    " Sender: " + sender.ToString() + "\r\n" +
                    " Source: " + e.Source + "\r\n" +
                    " Original Source: " + e.OriginalSource;
                lstMessages.Items.Add(message);
                e.Handled = (bool)chkHandle.IsChecked;
            }

            private void cmdClear_Click(object sender, RoutedEventArgs e)
            {
                eventCounter = 0;
```

```
            lstMessages.Items.Clear();
        }
    }
```

代码 6-12　Window1.xaml 和 Window1.xaml.cs 文件

（3）添加特殊的事件处理函数，为 MySimpleButton 添加一个指定客栈——类事件处理函数 CustomClickClassHandler（代码①）。为了通知外部窗口，把路由事件旅游的信息输出在 ListBox 列表中，因此需要添加一个普通的 CLR 事件 ClassHandlerProcessed（代码②）。在 Windows 的 Load 事件中为 Grid 添加一个慈善客栈 ProcessHandlersToo（代码③），如代码 6-13 所示。

MySimpleButton.cs

```
            public class MySimpleButton : Button
            {
                static MySimpleButton()
                {
①                  // 将 CustomClickEvent 和一个 Class Handler 关联起来
                   EventManager.RegisterClassHandler(typeof(MySimpleButton),
    CustomClickEvent, new RoutedEventHandler(CustomClickClassHandler), false);
                }
                ……
                // 普通 CLR 事件
②               public event EventHandler ClassHandlerProcessed;
                public  static  void  CustomClickClassHandler(object  sender,
    RoutedEventArgs e)
                {
                    MySimpleButton simpleBtn = sender as MySimpleButton;
                    EventArgs args = new EventArgs();
                    simpleBtn.ClassHandlerProcessed(simpleBtn, args);
                }
            }
```

Window1.xaml.cs

```
            public partial class Window1 : Window
            {
                public Window1()
                {
                    InitializeComponent();
                    // MySimpleButton 的类事件处理函数处理过 Window 能够得到通知
                    this.simpleBtn.ClassHandlerProcessed += new
     EventHandler(simpleBtn_RaisedClass);
                }
                ……

                private  void simpleBtn_RaisedClass(object sender,EventArgs e)
                {
                    eventCounter++;
                    string message = "#" + eventCounter.ToString() + ":\r\n" +
                    "Windows Class Handler\r\n" + "Sender:" + sender.ToString();
                    lstMessages.Items.Add(message);
                }

                private void ProcessHandlersToo(Object sender, RoutedEventArgs e)
                {
                    eventCounter++;
                    string message = "#" + eventCounter.ToString() + ":\r\n" +
                        "ProcessHandlersToo\r\n"+
                        " Sender: " + sender.ToString() + "\r\n" +
                        " Source: " + e.Source + "\r\n" +
```

```
                    " Original Source: " + e.OriginalSource;
                lstMessages.Items.Add(message);
            }
③           private void Window_Loaded(object sender, RoutedEventArgs e)
            {
                grid1.AddHandler(MySimpleButton.CustomClickEvent,     new
RoutedEventHandler(ProcessHandlersToo), true);
            }
        }
```

代码 6-13　添加特殊的事件处理函数

通过查看列表中的信息观察，就可以印证路由事件的旅行北上图。

6.6　接下来做什么

事件一直是个挥之不去的话题。如果从纵向的角度去了解路由事件的话，那么它的历史发展轨迹如下：

（1）第 1 阶段：“上古”Win32 时期，回调函数和函数指针。

（2）第 2 阶段：“上古”MFC 时期，MFC 的消息映射和类的函数指针。

（3）第 3 阶段：.NET 时期或者说 WinForm 时期，事件和委托。

（4）第 4 阶段：当代，WPF 时期及路由事件。

如果希望了解路由事件的演变，请参考笔者的博文《路由事件的演变史》；如果希望了解委托和事件，请参考如下两篇关于委托和事件的博文。

（1）“2007 C#中的委托和事件”http://www.cnblogs.com/jimmyzhang/archive/2007/09/ 23/903360.html。

（2）“2008 C#中的委托和事件”http://www.cnblogs.com/JimmyZhang/archive/2008/ 08/22/1274342.html。

如果希望深入了解委托，请参见 Jeffery Richer 所著的《.Net 框架程序设计 CLR Via C#》（第 2 版）第 15 章。接下来，我们将要讲述的是 WPF 的 Command 模型，此为心法一卷最后一章。

参考文献

[1] “北风之神”风清远整理，“云中孤雁”制作 《金庸全集典藏版 神雕侠侣》，“第三十八回 生死茫茫”。

[2] MSDN Library for Visual Studio 2008 SP1 Routed Events Overview。