实 验 九　事件机制

9.1　实验目的

事件是面向对象中一种机制，事件机制带给程序运行两个新的特性，其一是允许对象调用方法可由其它对象方法来实现，这进一步提高了面向对象程序的灵活性，其二是允许一个对象的某个行为可以有多个独立的方法执行，事件机制的实现离不开多路广播委托 (MulticastDelegate)。通过本实验掌握事件与编写事件处理器和调用事件的方法。

9.2　事件机制介绍

要理解事件，首先要理解委托 (delegate)，delegate 关键字用于声明一个引用类型，这个类型用于包含某种方法，并且规定了方法的参数列表形式，委托类似于 C++ 中的函数指针，委托在.NET 平台中更加安全和可靠，不会发生越界或地址无效的情况；与 C++ 中较大的区别是，函数指针只能指向静态函数，而委托可以引用静态函数也可以引用非静态成员方法。委托的基类是 System.Delegate 类，但它是个抽象类，因些不能直接将其实例化，System.MulticastDelegate 类是多路广播委托，它可包含有多个方法，System.MulticastDelegate 是抽象类，也不能直接实例化，只有系统和编译器可以显式地从 Delegate 类或 MulticastDelegate 类派生，用户是不允许由委托类进行派生新类的。用户正确使用委托类的方法就是使用关键字 delegate，而委托是异步操作和事件处理的基础。

在使用 delegate 关键字创建委托对象时，应当与要传递的方法具有相同的参数和返回值类型。类或对象可以通过事件向其他类或对象通知发生的相关事情。发送（或引发）事件的类称为“发行者”，接收（或处理）事件的类称为“订户”。对象在引必事件时需要传递参数，参数必须由 System.EventArgs 类派生，C# 中的事件处理实际上是一种具有特殊签名的 delegate，特殊类型的多路广播委托，它的形式如下：

public delegate void myeventhandler(object sender, Eventargs e);

sender 参数代表事件发送者，e 是事件参数类，所有的事件参数类都必须从 System.EventArgs 类派生，结合 delegate，事件的实现有下面的几步：

1. 定义 delegate 对象类型，它的参数列表包含两个参数，第一个参数是事件发送者对象，第二个参数是事件参数类对象。。
2. 定义事件参数自定义类，此类必须由 System.EventArgs 类派生。
3. 定义事件处理方法，也就是普通的类的方法，但要求这个方法应当与 delegate 对象具有相同的参数和返回值类型。
4. 用 event 关键字定义事件对象，它同时也是一个 delegate 对象。

5. 用 += 操作符添加事件到事件队列中，而 -= 操作符能够将事件从队列中删除。
6. 在需要触发事件的地方用调用 delegate 的方式写事件触发方法。
7. 调用事件方法触发事件。

9.3 事件机制程序示例

本程序是事件机制的演示，包含下面几个主要的类，FireEventArgs 表示用户定义的事件参数类，用于事件传递时传递信息；FireAlarm 类是事件源类，在程序中表示发生了火情；FireHandlerClass 类是一个对火情的处理类，表示主人对火情的处理方式；FireWatcherClass 类是另一个对火情的处理类，表示群众对火情的处理方式。当发生火情这个事件时，会有两个对象对事件执行不同的任务。

9.3.1 程序框架

新建一个名为 ConsoleApplication1 控制台应用程序，其位置为 D:
jiao。添加一个类 OutStr，这个类中只有一个静态成员类型为 StreamWriter，对象名为 sw，使用 sw 进行文本信息输出，保存在“dataout.txt”文本文件中，方便查看运行结果，并在 Main 函数中添加适当的代码。改造后的程序如下：

```
    //信息输出类
static class OutStr
{
  public static StreamWriter sw;
}
class Program
{
  static voidMain(string[] args)
  {
    OutStr.sw = new StreamWriter("dataout.txt");
    OutStr.sw.AutoFlush = true;
    OutStr.sw.Close();
  }
}
```

9.3.2 事件参数类定义

```
    //事件状态信息类由 System.EventArgs 类派生
public class FireEventArgs : EventArgs
{
  public FireEventArgs(string room, int ferocity)
  {
    this.room = room;
    this.ferocity = ferocity;
```

```
    }
    public string room; //火情发生地
    public int ferocity; //火情凶猛程度
  }
```

9.3.3 事件源类定义

事件机制有发行者和接收者，发行者发起和产生事件，发行者类本身不提供对事件的处理，在此定义一个类名为 FireAlarm，其定义如下：

```
    public class FireAlarm
  {
    //将火情处理定义为 FireEventHandler 代理 (delegate) 类型，这个代理声明的事件的参数列表
    public delegate void FireEventHandler(object sender, FireEventArgs fe);
    //定义 FireEvent 为 FireEventHandler delegate 事件 (event) 类型.
    public event FireEventHandler FireEvent;
    //激活事件的方法，创建了 FireEventArgs 对象，发起事件，并将事件参数对象传递过去
    public void ActivateFireAlarm(string room, int ferocity)
    {
      FireEventArgs fireArgs = new FireEventArgs(room, ferocity);
      //执行对象事件处理，必须保证处理函数要和声明代理时的参数列表相同
      FireEvent(this, fireArgs);
    }
  }
```

9.3.4 事件处理类定义

事件机制中的接收者将会执行与事件相关的代码，事件处理方法很灵活，而且允许同个事件可由多个处理方式，在此定义了两个类都对相同的事件执行相应的处理逻辑，一个类名为 FireHandlerClass，另一个类名为 FireWatcherClass，参考代码如下：

```
    //用于处理事件的类 FireHandlerClass，这个类定义了实际事件处理代码
  class FireHandlerClass
  {
    //事件处理类的构造函数使用事件源类作为参数
    public FireHandlerClass(FireAlarm fireAlarm)
    {
    //将事件处理的代理 (函数指针) 添加到 FireAlarm 类的 FireEvent 事件中，当事件发生时，就会执行指定的函数；
    fireAlarm.FireEvent += new FireAlarm.FireEventHandler(ExtinguishFire);
    }
    //当起火事件发生时，用于处理火情的事件
```

```
    void ExtinguishFire(object sender, FireEventArgs fe)
    {
    OutStr.sw.WriteLine(" {0} 对象调用，灭火事件 ExtinguishFire 函数.", sender.ToString());
    //根据火情状况，输出不同的信息.
    if (fe.ferocity < 2)
    OutStr.sw.WriteLine(" 火情发生在 {0}，主人浇水后火情被扑灭了",
    fe.room);
    else if (fe.ferocity < 5)
    OutStr.sw.WriteLine(" 主人正在使用灭火器处理 {0} 火势.", fe.room);
    else
    OutStr.sw.WriteLine("{0} 的火情无法控制，主人打 119!", fe.room);
    }
  }
      //用于处理事件的另一个类 FireWatcherClass
  class FireWatcherClass
  {
    //事件处理类的构造函数使用事件源类作为参数
    public FireWatcherClass(FireAlarm fireAlarm)
    {
      //将事件处理的代理 (函数指针) 添加到 FireAlarm 类的 FireEvent 事件中，当事件发生
时，就会执行指定的函数；
      fireAlarm.FireEvent += new FireAlarm.FireEventHandler(WatchFire);
    }
    //当起火事件发生时，用于处理火情的事件
    voidWatchFire(object sender, FireEventArgs fe)
    {
      OutStr.sw.WriteLine(" {0} 对象调用，群众发现火情 WatchFire 函数.", sender.ToString());
      //根据火情状况，输出不同的信息.
      if (fe.ferocity < 2)
      OutStr.sw.WriteLine(" 群众察到火情发生在 {0}，主人浇水后火情被扑灭了", fe.room);
      else if (fe.ferocity < 5)
      OutStr.sw.WriteLine(" 群众察到火情发生在 {0}，群众帮助主人 {0} 火势.", fe.room);
      else
      OutStr.sw.WriteLine(" 群众无法控制 {0} 的火情，消防官兵来到!", fe.room);
    }
  }
```

9.3.5 事件的发起与处理

在主程序中定义事件源类和事件处理类对象，FireAlarm 表示发生了火情，FireHandlerClass 表示主人对火情的处理过程，FireWatcherClass 表示了群从对火情的处理过程，注意事件处理类的构造函数使用了事件发起类对象作为参数。

```
    //定义一个火情发生源类对象；
FireAlarm myFireAlarm = new FireAlarm();
//定义一个火情处理类对象，并将源类对象作为参数传递给这个对象
FireHandlerClass myFireHandler1 = new FireHandlerClass(myFireAlarm);
FireWatcherClass myFireHandle2 = new FireWatcherClass(myFireAlarm);
//发生一种火情，以事件机制执行
myFireAlarm.ActivateFireAlarm("Kitchen", 3);
myFireAlarm.ActivateFireAlarm("Kitchen", 6);
```

9.4 作业

调试运行程序，分析程序输出结果，理解事件调用流程。