C#基础系列：再也不用担心面试官问我“事件”了 - 文章 - 伯乐在线

前言：作为.Net攻城狮，你面试过程中是否遇到过这样的问题呢：什么是事件？事件和委托的区别？既然事件作为一种特殊的委托，那么它的优势如何体现？诸如此类…你是否也曾经被问到过？你又是否都答出来了呢？

关于面试中涉及到的事件的问题，我们只需要抓住几个关键点就好了：

（1）事件是委托的封装，可以理解为一种特殊的委托。

（2）事件里面其实就两个方法(即add_event()和remove_event())和一个私有的委托变量，这两个方法里面分别是对这个私有的委托变量进行的合并和移除，当调用事件的+=时其实是调用的事件里面的add_event()方法，同样-=调用的是remove_event()方法。

（3）事件只能够从对象外部增加新的响应方法和删除已知的响应方法，而不能主动去触发事件和获取其他注册的响应方法等信息。如果使用公有的delegate则不能做这些限制，也就是说事件对委托做了限制，使委托使用起来更加方便。也有人说事件是对委托的阉割，大概也是这个意思。

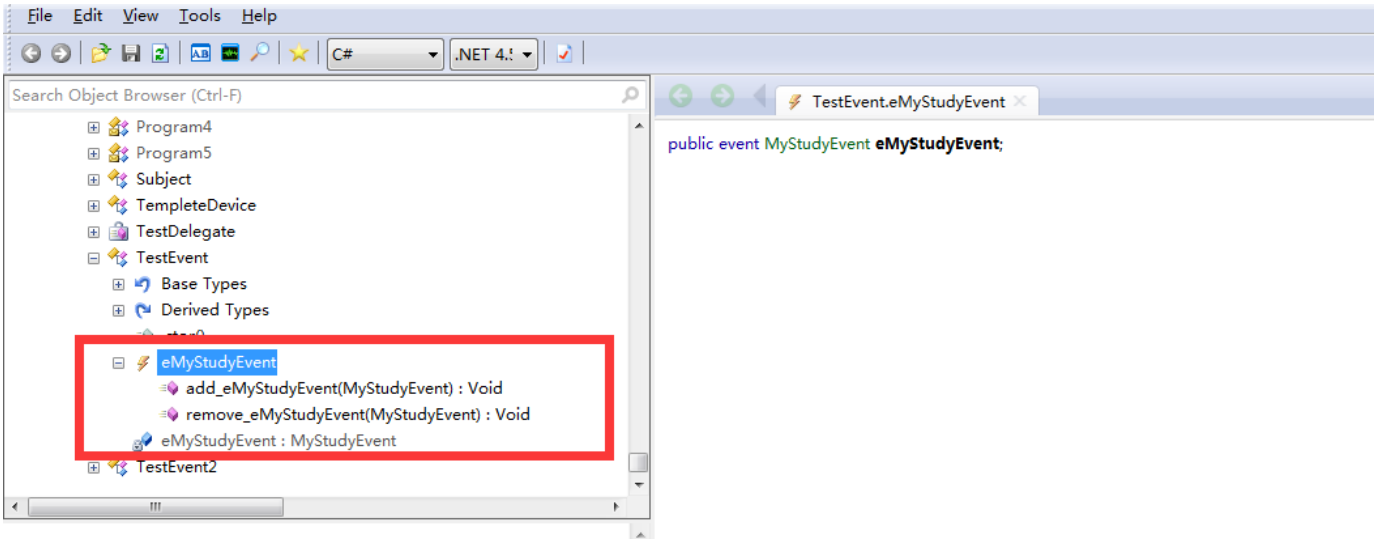
如果回答的时候抓住了以上的3点，那么我想你的面试应该不会太差。毕竟面试那么短的时间，有一两个亮点就很不错了，你说呢。哪怕你对事件机制完全不懂，为了面试记住其中两点也是很好的，工作经验咱们没有，换工作的经验可不能没有哦~~扯远了，关于面试就到此为止。如果你还想继续将事件了解透彻，别着急，慢慢往下看。

1、事件的定义及由来：

定义事件：

```
public delegate void MyStudyEvent(object sender, EventArgs e);
public class TestEvent
{
    public event MyStudyEvent eMyStudyEvent;
}
```

将这段代码生成dll后，通过反编译工具reflector我们可以看到：



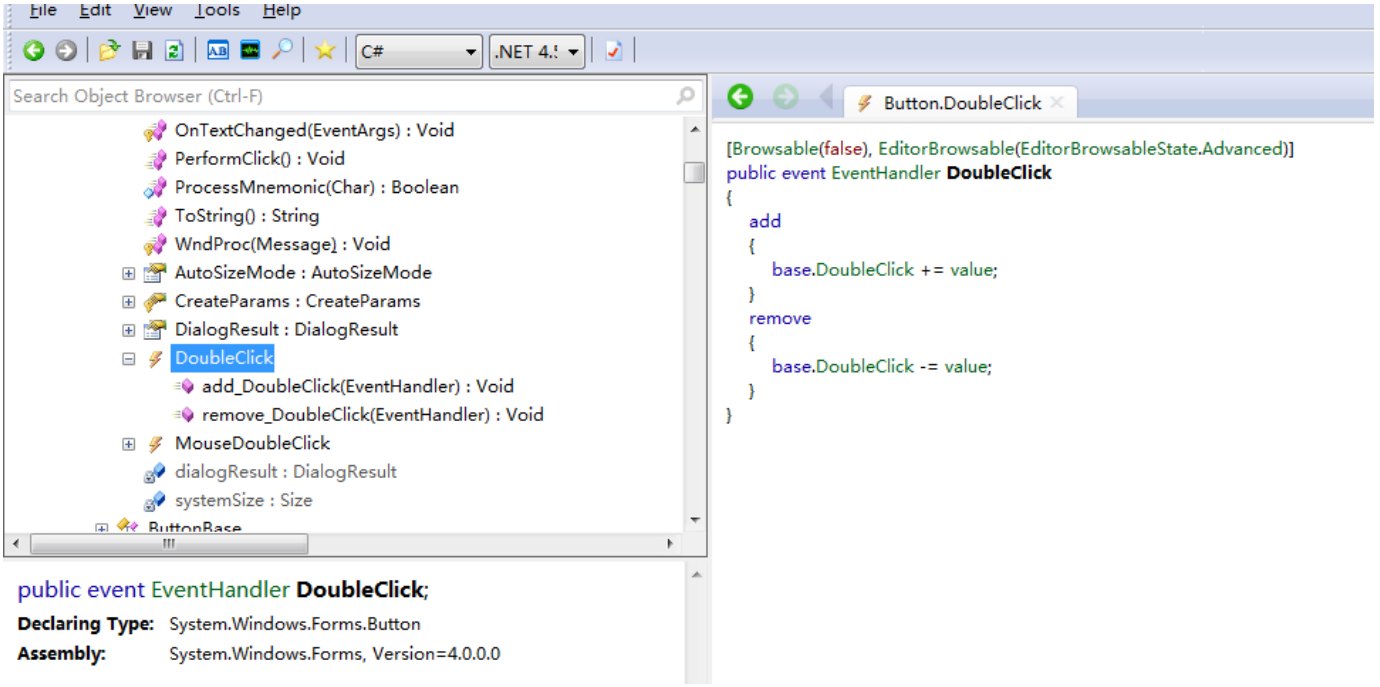
正如上文所说，可以看到当定义一个事件public event MyStudyEvent eMyStudyEvent的时候，编译器会自动给他生成两个方法add和remove，以及一个private的委托变量eMyStudyEvent。我们将反编译代码copy出来看看。

可以看到这两个方法的主要作用就是在向private变量eMyStudyEvent里面添加委托和移除委托。当调用事件的+=和-=时，eMyStudyEvent里面就合并和移除传过来的委托，当事件触发的时候，eMyStudyEvent变量就执行。这样设计也正好符合封装的原则，保证了内部变量的安全性。

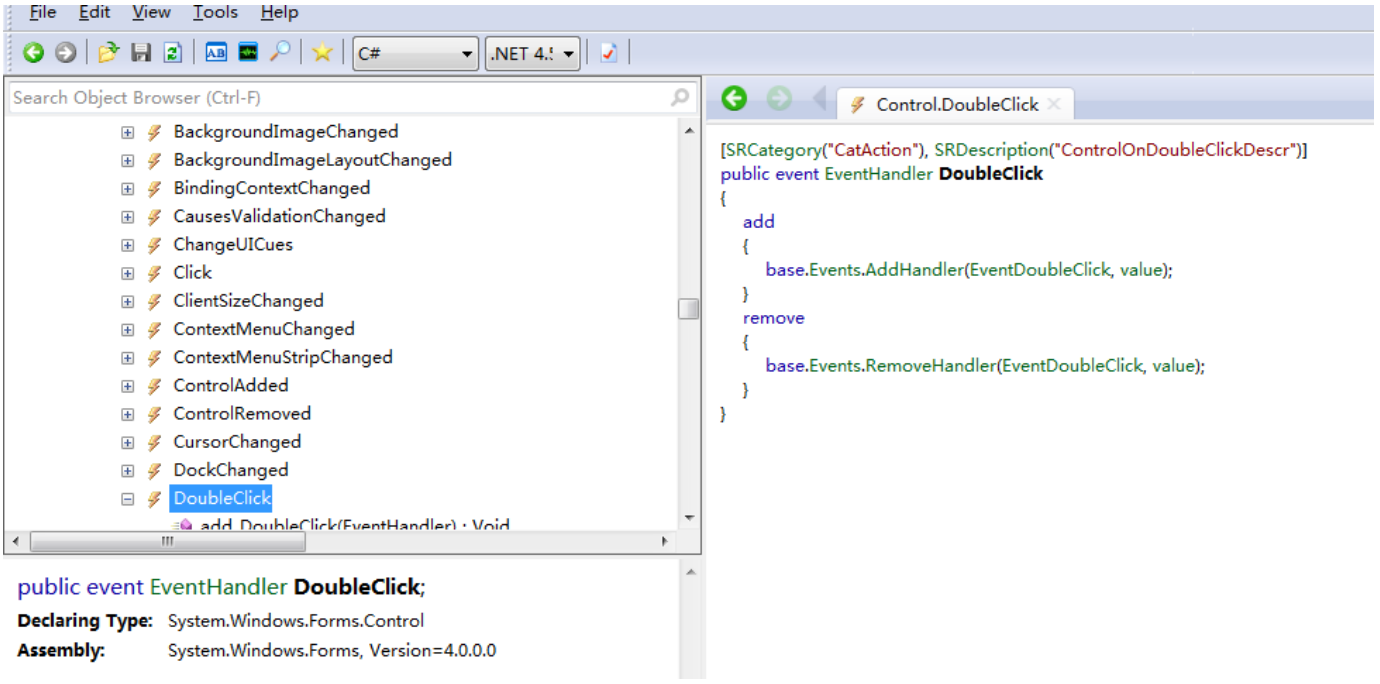
```
C#//私有委托变量
private MyStudyEvent eMyStudyEvent;
//add方法合并委托到eMyStudyEvent里面
public void add_eMyStudyEvent(MyStudyEvent value)
{
        MyStudyEvent event3;
        MyStudyEvent eMyStudyEvent = this.eMyStudyEvent;
        do
        {
            event3 = eMyStudyEvent;
            MyStudyEvent event4 = (MyStudyEvent)System.Delegate.Combine(event3, value);
            eMyStudyEvent = Interlocked.CompareExchange<MyStudyEvent>(ref
this.eMyStudyEvent, event4, event3)
        }
        while (eMyStudyEvent != event3);
}//remove方法移除eMyStudyEvent里面已存在的委托
public void remove_eMyStudyEvent(MyStudyEvent value)
{
        MyStudyEvent event3;
        MyStudyEvent eMyStudyEvent = this.eMyStudyEvent;
        do
        {
            event3 = eMyStudyEvent;
            MyStudyEvent event4 = (MyStudyEvent)System.Delegate.Remove(event3, value);
```

```
                eMyStudyEvent = Interlocked.CompareExchange<MyStudyEvent>(ref
this.eMyStudyEvent, event4, event3);
        }
        while (eMyStudyEvent != event3);
}
```

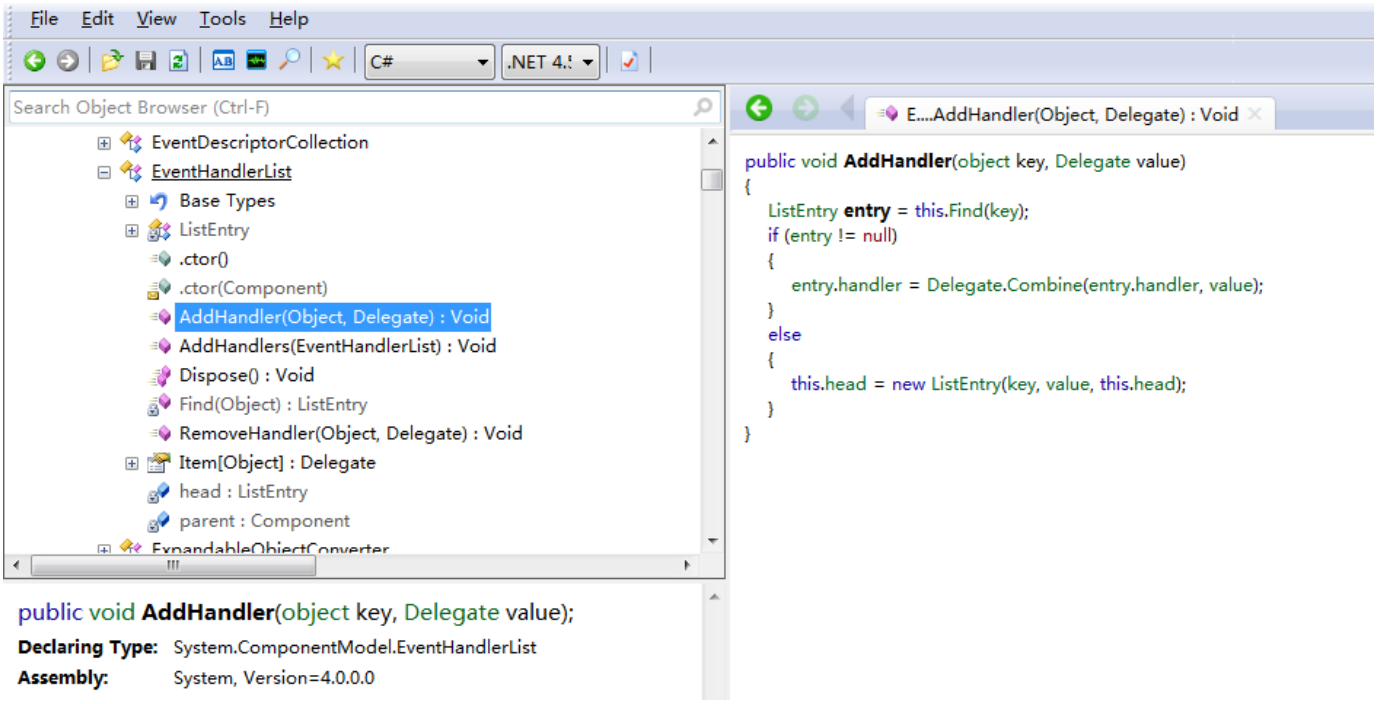
2、Framework里面的事件：既然自定义的事件是这样的，那么有人就要问了，.Net里面的事件是否也是如此。我们直接通过反编译工具来看。我们找到System.Windows.Forms下面的Button类，这个也是我们Winform里面使用最多的按钮，我们来看看我们经常使用的事件。



base.DoubleClick转到定义：



Events.AddHandler()转到定义：



、

是不是很眼熟，也是通过Delegate.Combine()来合并委托。

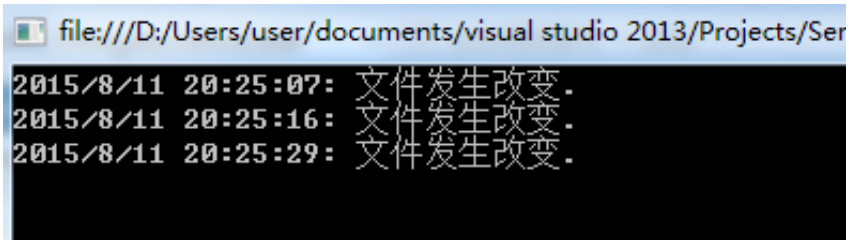
3、自定义事件的使用。事件使用的例子园子里面文章也很多，此片博主就以监听文件夹里面的1.txt文件是否存在为例说明事件的使用以及使用事件和委托的区别。

首先定义事件以及触发事件的方法：

C#

```
        static FileWatch FileWatchEventSource;
        static void Main(string[] args)
        {
                FileWatchEventSource = new FileWatch();
                //1. 启动后台线程添加监视事件
                var thrd = new Thread(new
ThreadStart(FileWatchEventSource.MonitorFile));
                thrd.IsBackground = true;
                thrd.Start();
                //2.注册本地事件处理方法
                FileWatchEventSource.FileWatchEvent += new
FileWatchEventHandler(OnFileChange);
                Console.ReadLine();
        }
        private static void OnFileChange(object Sender, EventArgs e)
        {
                Console.WriteLine(DateTime.Now.ToString() + ": 文件发生改变.");
        }
```

启动程序后，每当在文件夹里面删除和创建1.txt的时候就会打印提示文件改变。

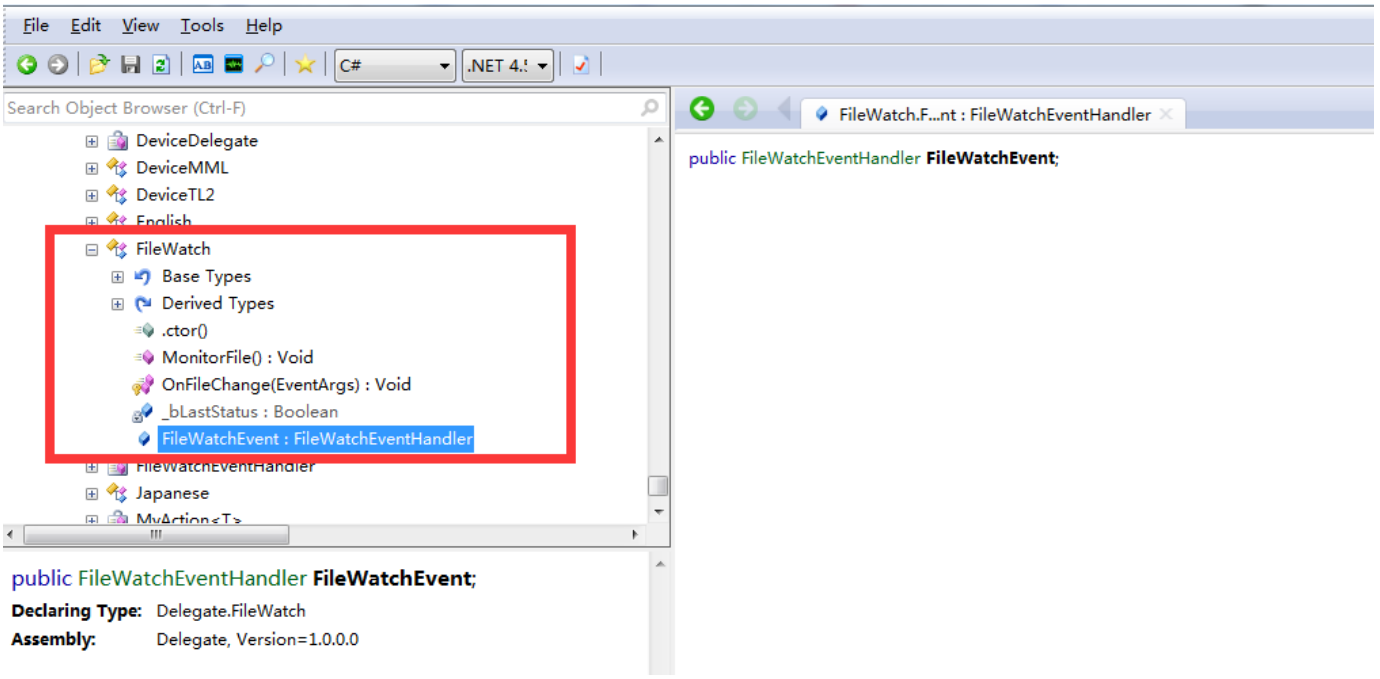


博主好奇心重，貌似这种监听直接用委托也可以实现了。于是乎将event事件变量直接改成委托变量。

C#

然后运行。发现可以得到一样的结果。

只是在反编译的时候没有add和remove方法而已：



这里正是需要说明的上面的面试回答第三条：事件只能通过+=和-+去阉割委托，而不能主动触发事件。如当使用事件的机制public event FileWatchEventHandler FileWatchEvent时。在Main函数里面

C#

```
                static void Main(string[] args)
                {
                        FileWatchEventSource = new FileWatch();
                        //1. 启动后台线程添加监视事件
                        var thrd = new Thread(new
ThreadStart(FileWatchEventSource.MonitorFile));
                        thrd.IsBackground = true;
                        thrd.Start();
                        //2.注册本地事件处理方法
                        FileWatchEventSource.FileWatchEvent += new
FileWatchEventHandler(OnFileChange);
                //这样写是报错的。
                        FileWatchEventSource.FileWatchEvent(null, null);
```

```
            Console.ReadLine();
        }
```

这样写是会报错的FileWatchEventSource.FileWatchEvent(null, null);因为事件不能主动触发，而改成委托后这样写就是正确的。并且如果是event变量，除了+=和-=操作，其他所有操作都会报错：

从这里可以看出，事件对委托进行了封装和约束。

总而言之，言而总之，事件和委托，打一个不太恰当的比喻，就类似面包和面粉，面包是由面粉加工而来的，当我们肚子饿了的时候，面包直接就能吃，面粉还需要加工。而当我们需要面条的时候，面粉就派上用场了，面包对于我们来说是用不了的。不知道这样解释好理解否。说了这么多，确实有点绕，需要好好体会下。