C#基础温习：理解委托和事件 - 文章 - 伯乐在线

原文出处： gao-yang

1. 委托

> 委托类似于C++中的函数指针（一个指向内存位置的指针）。委托是C#中类型安全的，可以订阅一个或多个具有相同签名方法的函数指针。简单理解，委托是一种可以把函数当做参数传递的类型。很多情况下，某个函数需要动态地去调用某一类函数，这时候我们就在参数列表放一个委托当做函数的占位符。在某些场景下，使用委托来调用方法能达到减少代码量，实现某种功能的用途。

1.1 自定义委托

声明和执行一个自定义委托，大致可以通过如下步骤完成：

1. 利用关键字delegate声明一个委托类型，它必须具有和你想要传递的方法具有相同的参数和返回值类型；
2. 创建委托对象，并且将你想要传递的方法作为参数传递给委托对象；
3. 通过上面创建的委托对象来实现该委托绑定方法的调用。

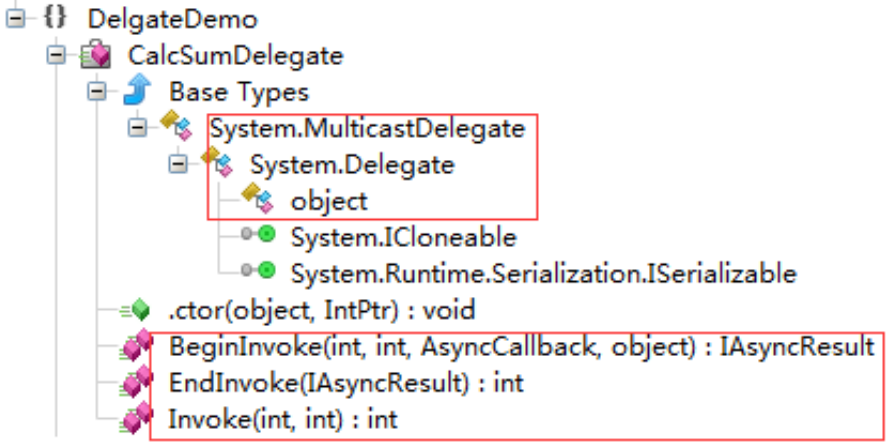
下面一段代码，完成了一次应用委托的演示：

C#

```
        //step01：使用delegate关键字声明委托
        public delegate int CalcSumDelegate(int a, int b);class Program
        {
                static void Main(string[] args)
                {
                        //step03：实例化这个委托，并引用方法
                        CalcSumDelegate del = new CalcSumDelegate(CalcSum);
                        //step04：调用委托
                        int result = del(5, 5);
                        Console.WriteLine("5+5=" + result);
                }//step02：声明一个方法和委托类型对应
                public static int CalcSum(int a, int b)
                {
                        return a + b;
                }
```

```
    }
```

通过上面4个步骤，完成了委托从声明到调用的过程。接着，咱也学着大神用ILSpy反编译上面的代码生成的程序集。截图如下：



（1）自定义委托继承关系是：System.MulticastDelegate —> System.Delegate —>System.Object。
（2）委托类型自动生成3个方法：BeginInvoke、EndInvoke、Invoke。查资料得知，委托正是通过这3个方法在内部实现调用的。

Invoke 方法，允许委托同步调用。上面调用委托的代码del(5, 5)执行时，编译器会自动调用del.Invoke(5,5)；
BeginInvoke 方法，以允许委托的异步调用。假如上述委托以异步的方式执行，则要显示调用dal.BeginInvoke(5,5)。

> 注意：BeginInvoke 和 EndInvoke 是.Net中使用异步方式调用同步方法的两个重要方法，具体用法详见微软官方示例。

1.2 多播委托

一个委托可以引用多个方法，包含多个方法的委托就叫多播委托。下面通过一个示例来了解什么是多播委托：

C#

```
        //step01：声明委托类型
        public delegate void PrintDelegate();
        public class Program
        {
                public static void Main(string[] args)
                {
                        //step03：实例化委托，并绑定第1个方法
                        PrintDelegate del = Func1;
                        //绑定第2个方法
                        del += Func2;
                        //绑定第3个方法
```

```
                del += Func3;
                //step04：调用委托
                del();//控制台输出结果：
                //调用第1个方法!
                //调用第2个方法!
                //调用第3个方法!
        }
        //step02：声明和委托对应签名的3个方法
        public static void Func1()
        {
                Console.WriteLine("调用第1个方法!");
        }
        public static void Func2()
        {
                Console.WriteLine("调用第2个方法!");
        }
        public static void Func3()
        {
                Console.WriteLine("调用第3个方法!");
        }
}
```

可以看出，多播委托的声明过程是和自定义委托一样的，可以理解为，多播委托就是自定义委托在实例化时通过 “+=” 符号多绑定了两个方法。

（1）为什么能给委托绑定多个方法呢？

自定义委托的基类就是多播委托MulticastDelegate ，这就要看看微软是如何对System.MulticastDelegate定义的：

MulticastDelegate 拥有一个带有链接的委托列表，该列表称为调用列表，它包含一个或多个元素。在调用多路广播委托时，将按照调用列表中的委托出现的顺序来同步调用这些委托。如果在该列表的执行过程中发生错误，则会引发异常。（摘自MSDN）（2）为什么使用“+=”号就能实现绑定呢？

先来看上述程序集反编译后的调用委托的代码：

```
// DelegateDemo.Program
public static void Main(string[] args)
{
    PrintDelegate del = new PrintDelegate(Program.Func1);
    del = (PrintDelegate)Delegate.Combine(del, new PrintDelegate(Program.Func2));
    del = (PrintDelegate)Delegate.Combine(del, new PrintDelegate(Program.Func3));
    del();
    Console.ReadKey();
}
```

“+=”的本质是调用了Delegate.Combine方法，该方法将两个委托连接在一起，并返回合并后的委托对象。

（3）多播委托能引用多个具有返回值的方法嘛？

答案是，当然能。委托的方法可以是无返回值的，也可以是有返回值的。不过，对于有返回值的方法需

要我们从委托列表上手动调用。否则，就只能得到委托调用的最后一个方法的结果。下面通过两段代码验证下：

C#

（1）总结上面事件使用的几个步骤：

step01：用event关键字定义事件，事件必须要依赖一个委托类型；

step02：在类内部定义触发事件的方法；

step03：在类外部注册事件并引发事件。

（2）public event EventHandler PlayOverEvent

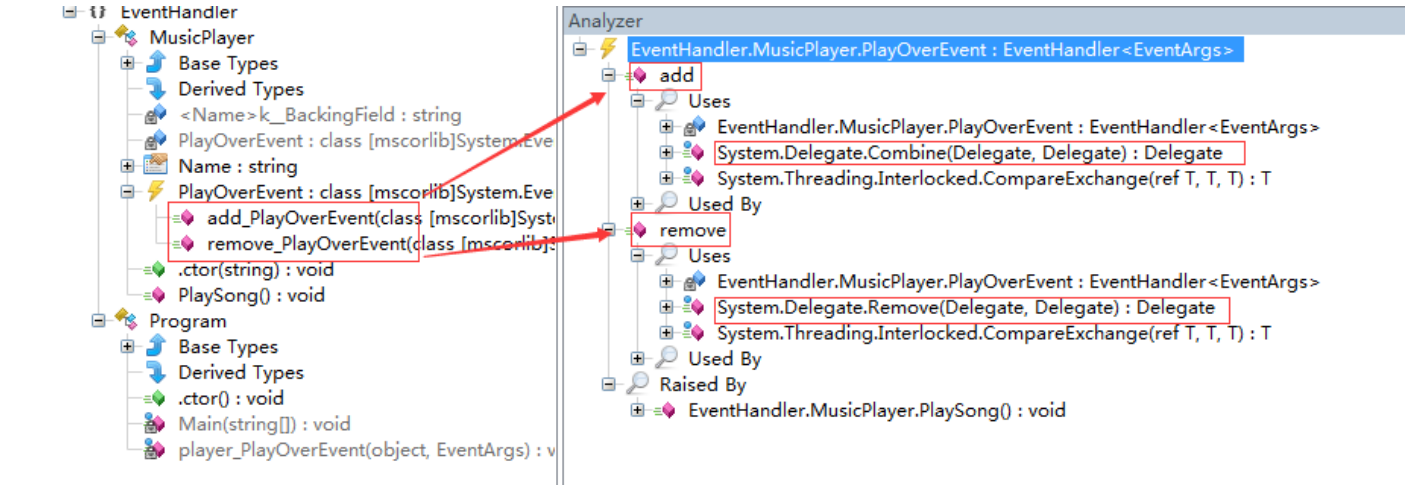
这句代码在MusicPlayer类定义了一个事件成员PlayOverEvent，我们说事件依赖于委托、是委托的特殊实例，所以EventHandler肯定是一个委托类型。下面我们来验证一下：

```
// 摘要:
//     表示将处理事件的方法。
//
// 参数:
//   sender:
//     事件源。
//
//   e:
//     一个 System.EventArgs，其中包含事件数据。
//
// 类型参数:
//   TEventArgs:
//     由该事件生成的事件数据的类型。
[Serializable]
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e);
```

EventHandler是微软封装好的事件委托，该委托没有返回值类型，两个参数：sender事件源一般指的是事件所在类的实例；TEventArgs事件参数，如果有需要创建，要显示继承System.EventArgs。

```
MusicPlayer player = new MusicPlayer("自由飞翔");
//注册事件
player.PlayOverEvent += player_PlayOverEvent;
player.PlaySong();
```

从上面代码我们观察到，事件要通过”+=”符号来注册。我们猜想，事件是不是像多播委托一样通过Delegate.Combine方法可以绑定多个方法？还是通过反编译工具查看下：



我们看到PlayOverEvent事件内部生成了两个方法：add_ PlayOverEvent和remove_ PlayOverEvent。add方法内部调用Delegate.Combine把事件处理方法绑定到委托列表；remove方法内部调用Delegate.Remove从委托列表上移除指定方法。其实，事件本质上就是一个多播委托。

3. 参考文章

[1] Edison Chou，http://www.cnblogs.com/edisonchou/p/4827578.html

[2] jackson0714，http://www.cnblogs.com/jackson0714/p/5111347.html

[3] Sam Xiao， http://www.cnblogs.com/xcj26/p/3536082.html

- 前端 - JavaScript, HTML5, CSS
- Java - 专注Java技术分享
- Python - 专注Python技术分享