C#基础系列：委托实现简单设计模式（1） - 文章 - 伯乐在线

前言：这篇简单介绍下委托的使用。当然啦，园子里面很多介绍委托的文章都会说道：委托和事件的概念就像一道坎，过了这个槛的人，觉得真是太容易了，而没有过去的人每次见到委托和事件就觉得心里发慌。确实这东西就像最开始学C语言的指针一样，令人有一种很纠结的感觉，总觉得要调用一个方法直接调用就行了，为啥非要定义一个委托时执行这个方法呢。其实在C#里面很多的技术都是为了重用和简化代码而生，委托也不例外，很多使用C#多态去实现的设计模式其实都可以使用委托的方式去改写，可以理解为一种轻量级的设计模式吧。博主打算抽一篇专门分享下多态和委托实现设计模式的异同。这篇就先介绍简单委托的使用。

一、什么是委托：C# 中的委托（Delegate）类似于 C 或 C++ 中函数的指针。用博主的话说，委托就是一种允许将方法名称作为参数传递的引用类型。它定义的是方法的类型，可以说就是方法的抽象，那么反过来说，方法可以理解为委托的实例，如public delegate void TestDelegate(string str);这种委托定义的就是所有参数类型为string，没有返回值的方法的一种抽象。

二、为什么要使用委托：记得博主刚开始做项目的时候看到委托的写法就头大，总觉得这是没事找事，唯一的好处貌似就是代码看上去很酷~~随着工作的累积，发现项目中某些小的需求使用这种轻量级的委托来实现的时候确实能减少很多代码量。

三、委托的使用：

1、.Net Framework 里面的委托类型：使用过委托的朋友可能注意到了C#里面定义了两种类型的委托变量，基本能满足我们的一般需求。

（1）Action类型的委托：C#里面定义Action委托用于抽象化那种没有返回值的方法。将Action变量转到定义可知它的最简单形式：

C#

```
static void Main(string[] args)
{
            //1.无参无返回值方法
            var oAction1 = new Action(Test1);
            oAction1.Invoke();//调用方式一
            oAction1();//调用方式二
            //2.有参无返回值
            var oAction2 = new Action<int, int, int>(Test5);
            oAction2.Invoke(1, 2, 3);
```

```
            oAction2(1, 2, 3);
            //匿名方法的调用
            var oAction3 = new Action<int, int, int>((a,b,c) => {
            //......
            });
            oAction3.Invoke(1, 2, 3);
    }
```

（2）Func类型的委托：还记得Linq里面的扩展方法Where()、Select()等方法的参数吗。public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate)。这里的参数就是一个Func类型的委托。C#里面Func类型的委托用于处理有参数有返回值的方法。不多说，上代码:

C#

```
public delegate void MyAction<in T>();
public delegate TResult MyFunc<in T, out TResult>(T arg);
```

其实使用起来和系统的Action和Func基本没有区别。

3、委托的合并和拆解就放在事件里面分享了。这篇且过之。。。

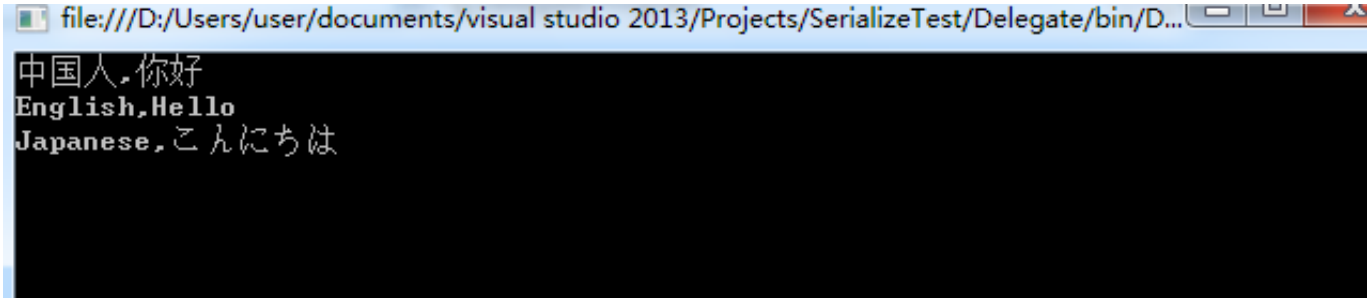
4、如果按照上面的方法去使用委托，那真的是要别扭死人了，因为调用方法直接用方法名调用就好了，何必还要定义一个委托变量去调用，这不是将简单问题复杂化么。确实，上面只是为了介绍委托而写的代码，实际项目中肯定不会这么用。其实委托在项目中一般用在将委托变量作为参数传递或者函数回调。来看下面代码：

C#

```
class Program
        {
                static void Main(string[] args)
                {
                        Person strHelper = new Person();
                        string r1 = strHelper.ProcessFunc("中国人", "你好", new
MyDelegate(strHelper.ChineseSayHello));
                        string r2 = strHelper.ProcessFunc("English", "Hello", new
MyDelegate(strHelper.EnglishSayHello));
                        string r3 = strHelper.ProcessFunc("Japanese", "こんにちは", new
MyDelegate(strHelper.JapaneseSayHello));
                        Console.WriteLine(r1);
                        Console.WriteLine(r2);
                        Console.WriteLine(r3);
                        Console.ReadKey();
                }
        }
        public delegate string MyDelegate(string s1, string s2);
```

```
        public class Person
        {
                public string ProcessFunc(string s1, string s2, MyDelegate process)
                {
                        return process(s1, s2);
                }
                public string ChineseSayHello(string s1, string s2)
                {
                        return s1 +","+ s2;
                }
                public string EnglishSayHello(string s1, string s2)
                {
                        return s1 + "," + s2;
                }
                public string JapaneseSayHello(string s1, string s2)
                {
                        return s1 +","+ s2;
                }
        }
```

得到结果：



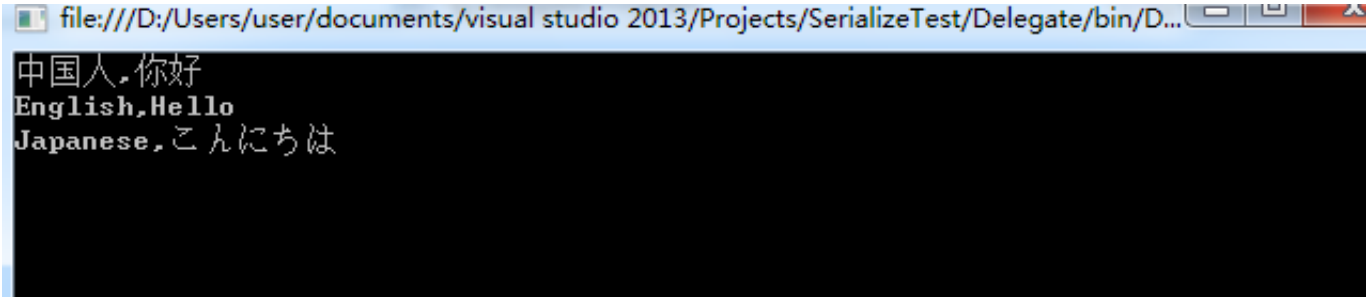
public string ProcessFunc(string s1, string s2, MyDelegate process)里面定义了一个回调函数，可以将任意一个符合这个委托的方法传递进去，得到想对应的结果。细看这种设计是不是和工厂设计模式十分相似，我简单构造了个工厂：

C#

```
public class Person
        {
                public virtual string SayHello(string s2)
                {
                        return s2;
                }
        }
public class Chinese : Person
        {
                public override string SayHello(string s2)
                {
```

```
                return "中国人," + s2;
            }
        }
        public class English : Person
        {
            public override string SayHello(string s2)
            {
                return "English," + s2;
            }
        }
        public class Japanese : Person
        {
            public override string SayHello(string s2)
            {
                return "Japanese," + s2;
            }
        }
//Main函数里面调用
class Program
        {
            static void Main(string[] args)
            {
                var r1 = GetPerson("你好").SayHello("你好");
                var r2 = GetPerson("Hello").SayHello("Hello");
                var r3 = GetPerson("こんにちは").SayHello("こんにちは");
                Console.WriteLine(r1);
                Console.WriteLine(r2);
                Console.WriteLine(r3);
                Console.ReadKey();
            }
            public static Person GetPerson(string strType)
            {
                if (strType == "你好")
                    return new Chinese();
                else if (strType == "Hello")
                    return new English();
                else
                    return new Japanese();
            }
        }
```

得到结果和上面相同：



这样一比较是不是对委托的用法有点感觉了呢~~如果你不怕麻烦，可以在项目是用起来试试，相信你会有收获~~

合作联系

Email：bd@Jobbole.com

QQ： 2302462408 （加好友请注明来意）

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子
头条 - 分享和发现有价值的内容与观点
相亲 - 为IT单身男女服务的征婚传播平台
资源 - 优秀的工具资源导航
翻译 - 翻译传播优秀的外文文章
文章 - 国内外的精选文章
设计 - UI,网页，交互和用户体验
iOS - 专注iOS技术分享
安卓 - 专注Android技术分享
前端 - JavaScript, HTML5, CSS
Java - 专注Java技术分享
Python - 专注Python技术分享