C#回调函数应用示例，形象比喻方法助理解，整理了一个简单的例子来说明回调函数的用法:   
namespace CallBackFunction   
{   
    class Program   
    {   
        static void Main(string[] args)   
        {   
            Program prog = new Program();//在静态函数Main中调用非静态方法时,必须先实例化该类对象,方可调用GetSum方法   
            SumClass sc = new SumClass();//实例化SumClass类

 //问题1：回调函数是指GetSum函数还是SumAll函数

GetSum是回调函数，其实也是一个普通的函数，只是相对于SumAll来说会回调GetSum这个函数； 就是说在SumAll函数执行过程中，当满足一定条件的时候会执行GetSum函数逻辑

//问题2：GetSum函数作为参数传入SumAll中时自己没有带参数

作为参数传给SumAll函数只要告诉调用什么特征的函数（函数特征：参数列表，参数类型， 返回值，返回类型），这个函数特征就是通过委托的形式来接收所有符合委托函数特征的所有函数， 而GetSum就是符合这个函数特征的，所以可以传入， 然后在SumAll函数中适时通过函数GetSum加()的方式传入指定参数来调用GetSum函数

//问题3：SumAll函数在GetSum函数中得到了什么，也就是GetSum函数传给了SumAll函数什么东西

SumAll函数通过委托接收符合条件的GetSum函数， 然后在SumAll中就获取到了GetSum的函数名，及需要传入什么参数类型的列表， 在SumAll中调用GetSum函数入指定类型参数

//问题4：怎么理解这个回调机制的内部实现过程

这个内部实现其实就是，函数通过委托定义，函数只能接收符合委托定义的函数特征的函数， 通过传入符合委托函数特征的函数， 然后在函数中调用委托协议方法时，执行器获取传入的具体函数名称，及需要传入指定类型的参数列表，对传入函数进行调用。

            int result=sc.SumAll(prog.GetSum);   
            Console.WriteLine(result.ToString());   
        }

        private int GetSum(int a, int b)   
        {   
            return (a + b);   
        }   
    }

    class SumClass   
    {

//此处声明的委托的参数列表与返回值与SumAll回调的函数相同  
        public delegate int Sum(int num1, int num2);   
        public int SumAll(Sum sum)   
        {   
            //可以进行些别的操作   
            return sum(1, 2);//调用传入函数的一个引用   
        }   
        //可以封装更多的业务逻辑方法   
    }   
}   
这个例子非常的简单,假设SumClass类是一个被封装好的,实现某种业务逻辑的类.   
其中包含一个委托(delegate)Sum返回值为int型,有两个int型的参数.   
还包括一个返回值为int方法SumAll,注意这个方法的参数,是Sum类型的参数,也可以理解为是一个函数的引用(这个函数就是回调函数).作为参数的sum,可以说是某个函数的引用,这个函数返回值为int,并提供两个整型参数.现在大概可以理解了吧?所谓委托可以被看作是一组方法签名相同的函数的引用的"类",其每一个实例都是符合该方法签名的函数的一个引用.在该方法中可以进行一些操作,并调用传进来的函数引用(sum),而此时sum这个方法内的具体业务逻辑是怎样的并不清楚,SumClass就充当了一个接口的作用,这时接口就需要调用SumClass类的客户端来通过定义一个"回调函数"来实现其具体功能了.   
   
再看Program类,这个类其实就是调用SumClass这个类的一个"客户端".   
先来看GetSum这个方法,返回值为int,接受两个int型的参数...如果前面说的你都明白了,那么你一定很清楚GetSum就是作为客户端的"回调函数"用来作为参数的.这个函数实现了两个参数是相加并返回其和的业务逻辑.   
在Main函数中,调用SubClass类的SubAll方法,将Programe类的GetSum方法的引用作为参数传递到SubAll方法中,也就是在SubAll中调用GetSum这个相加两个参数的方法,将1和2相加返回3,并将result(3)输出到屏幕上.   
通常情况下SubClass类会被封装到DLL中,而参数类型即为一个委托,传入此委托的一个实例(回调函数)来帮助SubClass实现其业务逻辑.   
最后再举一个生活中的例子帮助大家来理解:   
一天你老婆让你去市场买菜,而你因为偷懒就让你的儿子去买,把菜买回来了你直接将菜交给老婆.而老婆只是想要菜,并不关心菜是谁买的.第二天你老婆让你去买米,结果你又交由儿子处理.第三天老婆让去买肉,可怜的儿子再次被你叫去了...   
从这些事情我们可以抽象出一个类来.   
这个类就是实现不管老婆让你去做什么,你都叫儿子去代劳.那么我们就可以定义一个"让儿子去做"这样的委托.然后传入"买菜","买肉"等函数进来,这些函数就是回调函数.这样不论老婆需要"买菜","买肉"或者其他任何的事情,都可以通过你这个类来完成.

.Net编程中最经常用的元素，事件必然是其中之一。无论在ASP.NET还是WINFrom开发中，窗体加载(Load),绘制(Paint),初始化(Init)等等。“protected void Page\_Load(object sender, EventArgs e)”这段代码相信没有人不熟悉的。细心一点一定会发现，非常多的事件方法都是带了“object sender, EventArgs e”这两个参数。这是不是和委托非常相似呢？一、委托（有些书中也称为委派）委托是什么呢？这个名字的意思已经赋予了我们想象的空间，你是编程的，你现在正在写一个ASP.NET网页，而JS是你不熟悉的，于是你委托你的一位同事来帮助你完成JS部分。这就是委托，把你所不能做的事情交给其他人去做。而怎么知道是哪个人去做呢？当然是要知道名字！而为了区别名字一样的不同人，因此，需要描述一个特征。在C#中，委托的作用是这样描述的：委托就像一个函数的指针，在程序运行时可以使用它们来调用不同的函数。这个其实和你委托同事完成 JS代码一样。如果有两位同事可以做这件事情，他们只要做的结果能够满足你的需求（就像一个接口），尽管他们做的过程不一样，并且作出的效果也不一样，但是，能够达到你的要求就可以了。1、简单的委托那委托需要承载哪些信息呢？首先，它存储了方法名，还有参数列表(方法签名),以及返回的类型。比如：delegate string/\*返回类型\*/ ProcessDelegate(int i);这就是一个委托的定义。蓝色部分是声明委托的关键字，红色部分是返回的类型，而黑色部分是委托的类型名，和一个类名差不多，而()里的就是参数部分。它的意思是，你要使用这个委托来做事情的话，那么，做事情的方法必须满足以下条件：1、返回类型和委托的返回类型一致，这里是string类型；2、能且只能有一个参数，并且是int类型。OK,满足以上两个条件，一切就可以工作了:)例如： 1 using System; 2 using System.Collections.Generic; 3 using System.Text; 4 5 namespace TestApp 6 { 7 /// 8 /// 委托 9 /// 10 /// 11 /// 12 /// 13 public delegate string ProcessDelegate(string s1, string s2);14 15 class Program16 {17 static void Main(string[] args)18 {19 /\* 调用方法 \*/20 ProcessDelegate pd = new ProcessDelegate(new Test().Process);21 Console.WriteLine(pd("Text1", "Text2"));22 }23 }24 25 public class Test26 {27 /// 28 /// 方法29 /// 30 /// 31 /// 32 /// 33 public string Process(string s1,string s2)34 {35 return s1 + s2;36 }37 }38 }输出的结果是:Text1Tex22、泛型委托泛型的委托，就是然参数的类型不确定,例如代码改写为：using System;using System.Collections.Generic;using System.Text;namespace TestApp{ /// /// 委托 /// /// /// /// public delegate string ProcessDelegate(T s1, S s2); class Program { static void Main(string[] args) { /\* 调用方法 \*/ ProcessDelegate pd = new ProcessDelegate(new Test().Process); Console.WriteLine(pd("Text1", 100)); } } public class Test { /// /// 方法 /// /// /// /// public string Process(string s1,int s2) { return s1 + s2; } }}输出的结果就是：Text1100泛型的详细内容不属于本文的介绍范围，这里不加多说了。二、事件在某件事情发生时，一个对象可以通过事件通知另一个对象。比如，前台完成了前台界面，他通知你，可以把前台和你开发的程序整合了。这就是一个事件。可以看出事件是在一个时间节点去触发另外一件事情，而另外一件事情怎么去做，他不会关心。就事件来说，关键点就是什么时候，让谁去做。在C#中，时间定义关键字是event。例如：event ProcessDelegate ProcessEvent;整个事件定义方法以及执行过程：using System;using System.Collections.Generic;using System.Text;namespace TestApp{ /// /// 委托 /// /// /// /// public delegate void ProcessDelegate(object sender, EventArgs e); class Program { static void Main(string[] args) { /\* 第一步执行 \*/ Test t = new Test(); /\* 关联事件方法，相当于寻找到了委托人 \*/ t.ProcessEvent += new ProcessDelegate(t\_ProcessEvent); /\* 进入Process方法 \*/ Console.WriteLine(t.Process()); Console.Read(); } static void t\_ProcessEvent(object sender, EventArgs e) { Test t = (Test)sender; t.Text1 = "Hello"; t.Text2 = "World"; } } public class Test { private string s1; public string Text1 { get { return s1; } set { s1 = value; } } private string s2; public string Text2 { get { return s2; } set { s2 = value; } } public event ProcessDelegate ProcessEvent; void ProcessAction(object sender, EventArgs e) { if (ProcessEvent == null) ProcessEvent += new ProcessDelegate(t\_ProcessEvent); ProcessEvent(sender, e); } //如果没有自己指定关联方法，将会调用该方法抛出错误 void t\_ProcessEvent(object sender, EventArgs e) { throw new Exception("The method or operation is not implemented."); } void OnProcess() { ProcessAction(this, EventArgs.Empty); } public string Process() { OnProcess(); return s1 + s2; } }} 感觉到了什么？是不是和代码注入了差不多，相当于是可以用任意符合委托接口(委托确实很像接口)的代码，注入到Process过程。在他返回之前给他赋值。三、回调函数打了这么多字，好累啊!回调函数就是把一个方法的传给另外一个方法去执行。在C#有很多回调函数，比如异步操作的时候。这里先举个例子：using System;using System.Collections.Generic;using System.Text;namespace TestApp{ /// /// 委托 /// /// /// /// public delegate string ProcessDelegate(string s1, string s2); class Program { static void Main(string[] args) { /\* 调用方法 \*/ Test t = new Test(); string r1 = t.Process("Text1", "Text2", new ProcessDelegate(t.Process1)); string r2 = t.Process("Text1", "Text2", new ProcessDelegate(t.Process2)); string r3 = t.Process("Text1", "Text2", new ProcessDelegate(t.Process3)); Console.WriteLine(r1); Console.WriteLine(r2); Console.WriteLine(r3); } } public class Test { public string Process(string s1,string s2,ProcessDelegate process) { return process(s1, s2); } public string Process1(string s1, string s2) { return s1 + s2; } public string Process2(string s1, string s2) { return s1 + Environment.NewLine + s2; } public string Process3(string s1, string s2) { return s2 + s1; } }}输出结果：Text1Text2Text1Text2Text2Text1Process方法调用了一个回调函数，当然这里只执行了回调函数。可以看出，可以把任意一个符合这个委托的方法传递进去，意思就是说这部分代码是可变的。而设计上有一个抽离出可变部分代码的原则，这种用法无疑可以用到那种场合了。