# **初步理解委托、事件、匿名方法和Lambda**

最经在学习LinqtoSql，然后扯到Lambda表达式，然后扯到匿名方法，然后扯到委托，最后扯到事件处理。。。后来发现对委托这个概念和事件处理这个过程理解得不是很清晰，遂得一下学习笔记。那里说得不对，请大家多多指教！

## 第一部分：理解委托

委托委托，顾名思义，就是类似于中间人的意思，有些事情你不做，委托别人去做，比如你想相亲，但你不想去主动约女孩子，那你可以交给媒婆去帮你约。

如果你学过C++，请把委托理解成函数指针，都是为了调用函数。函数指针可以调用符合该函数指针要求的函数。什么叫符合该函数指针要求？就是被调用的函数拥有和该函数指针一样的返回类型、参数的个数相同、对应参数的类型一致。

函数指针举例说明。**int (\*p)(int,int);**定义了一个函数指针p,p只能调用 函数原型（或者说函数签名）是 返回类型为int、只有两个参数并且参数类型都是int的函数。

**委托举例说明：delegate void MyDelegate(int a,int b);**声明了一种委托类型 MyDelegate（委托有不同类型，就好像中间人有不同类型，你想相亲就找媒婆这种中间人，你想看风水，就找风水大师这种中间人），这种委托类型只能调用 返回类型是void、只有两个参数并且参数类型都是int的函数。跟函数指针是不是很像？

委托其实就是一个类，委托的基类是Delegate 类，只不过我们不能显示地从 Delegate 类派生出委托类（Delegate 类不是委托类型，它的派生类才是），这是给系统和编译器用的。只不过类实例化之后就叫对象，而委托实例化之后还是叫委托而已，另外实例化委托必须提供一个相对应的函数作为参数。

举例说明定义委托：（参考自：<http://www.cnblogs.com/warensoft/archive/2010/03/19/1689806.html>）

1、定义一个函数Max，函数签名位：返回类型为int，两个参数，参数类型都是int

****int Max(int x,int y)  {return x>y?x:y;}****

2、创建一个委托类型（名字是MyDelegate），并声明该委托类型可以调用的函数的函数原型（函数签名）为：返回类型为int，两个参数，参数类型都是int。注意，委托是一个类，创建委托类型要放在函数外面

****delegate int MyDelegate(int a,int b);****

3.建立一个委托类的实例（或者说MyDelegate的实例），并指向要调用的方法，有两种方式：

//****利用委托类的构造方法指定，这是最为常见的一种方式****

****MyDelegate md = new MyDelegate(Max);****

****//利用自动推断方式来指明要调用的方法，该形式更类型于函数指针****

****MyDelegate md = Max;****

用第一种方式创建委托实例时必须提供一个函数签名符合该委托类型的函数作为参数。

4、利用委托类实例调用所指向的方法

**int c = md(4,5);**

**总结：可以把委托理解成函数指针，可以调用函数签名和委托类型一致的函数**

## 第二部分：事件处理（本文主要想表达对委托的理解，所以这部分以后再详说）

事件处理涉及到两个对象：一个是事件源（就是触发该事件的对象），另一个是事件接收者（提供了处理方法的类）。两者本无联系。

事件处理的过程：首先事件源触发了某事件，委托捕获到这件事（委托通过订阅该事件实现），委托调用事件接收类中的相应函数进行处理。

举例说明：this.button1.Click += new System.EventHandler(this.button1\_Click);

事件源是button1，触发的事件是 按钮单击事件 Click，委托是 System.EventHandler ，事件处理函数是 button1\_Click 。上面代码的意思是，System.EventHandler （委托）对 this.button1（事件源）的  Click （事件） 进行订阅，当this.button1 的Click事件发生时，委托接收事件源传过来的一些参数，接着调用 this.button1\_Click (事件处理函数) 并把从事件源那里接收的参数传给该函数作为参数。

## 第三部分：匿名方法

匿名方法就是没有名字的方法（函数），既然没有名字，就是说只有在定义的时候能调用，在其他地方就不能调用了（没有名字啊，那就找不到嘛）。为什么要用到匿名方法呢？调用函数是需要花销的，但有时候调用的一些方法很短小（例如一句话方法）、只完成很少的功能，这个时候就有点得不偿失了，此时就可以定义一个匿名方法来完成这个功能了，而匿名方法作为内联代码，花销相对小很多。匿名方法都是和委托连在一起用的（以后还有lambda表达式），以前创建委托实例时需要传递一个函数名给它作为参数，现在可以通过匿名方法直接把一段代码传进去了

****定义匿名方法****：用到delegate关键字，如：****delegate(int a, int b){return a > b ? a : b ;}****   代码定义了一个匿名方法，该方法的参数是 int a 和 int b ，方法体是 {return a > b ? a : b}。如果只是定义一个匿名方法没有意义，因为定义完过后你就再也不能用这个匿名方法，所以匿名方法要在定义的时候就马上使用。一般来说就是初始化委托了。

使用匿名方法：

namespace DelegateTest

{

class Program

{

//创建委托类型

delegate int Mydelegate(int x, int y);

static void Main(string[] args)

{

//创建委托实例，指向匿名函数

Mydelegate mydelegate = delegate(int x,int y) { return x + y; };

//通过委托调用匿名方法

Console.WriteLine(mydelegate(3,2));

Console.ReadKey();

}

}

}

****总结：匿名方法就是没有名字的方法，只能通过委托来被调用（这句话可能有误，匿名方法也可以直接订阅事件作为事件处理函数，当事件发生时就接收相应参数并执行函数体）。****

## 第四部分：Lambda

上面说到可以通过匿名方法以内联代码的方式来简介地实现委托，但还有更简洁的方法，那就是用lambda表达式来代替匿名方法，在这里，lambda表达式就是匿名方法的简洁版。

Lambda表达式的语法如下：

(param1, param2 ...,paramN) =>

{

表达式1;

表达式2;

return 返回值;

}

param1, param2 ...,paramN 就是参数，不用确定类型，编译器会做这个工作，花括号了就是lambda表达式要执行的语句，如果对应的委托类型有返回值，那么就要有return 语句。

把第三部分那个匿名方法的例子改成用lambda表达式实现：

 namespace DelegateTest

{

class Program

{

//创建委托类型

delegate int Mydelegate(int x, int y);

static void Main(string[] args)

{

//创建委托实例，指向匿名函数

Mydelegate mydelegate = delegate(int x,int y) { return x + y; };

//通过委托调用匿名方法

Console.WriteLine(mydelegate(3,2));

Mydelegate youdelegate = (x, y) => { return (x - y); };

Console.WriteLine(youdelegate(3, 2));

Console.ReadKey();

}

}

总结：匿名方法就是没有名字的函数，Lambda表达式就是匿名方法的简洁版（指的是用在委托上这方面，Lambda还有其他方面的用处），两者都只是让代码更简洁，但在更底层层次本质是一样的