C# 委托（Delegates）使用详解

委托是C#编程一个非常重要的概念，也是一个难点。本文将系统详细讲解委托。

1. 委托是什么？

其实，我一直思考如何讲解委托，才能把委托说得更透彻。说实话，每个人都委托都有不同的见解，因为看问题的角度不同。个人认为，可以从以下2点来理解：

（1） 从数据结构来讲，委托是和类一样是一种用户自定义 类型 。

（2） 从 设计模式 来讲，委托（类）提供了 方法 （对象）的抽象。

既然委托是一种类型，那么它存储的是什么数据？

我们知道，委托是方法的抽象，它存储的就是一系列具有相同签名和返回回类型的方法的地址。调用委托的时候，委托包含的所有方法将被执行。

2. 委托类型的定义

委托是类型，就好像类是类型一样。与类一样，委托类型必须在被用来创建变量以及类型对象之前声明。

```
delegate void MyDel(int x);
```

委托类型声明：

（1） 以deleagate关键字开头。

（2）返回类型+委托类型名+参数列表。

3. 声明委托变量

```
MyDel del1,del2;
```

4. 初始化委托变量

(1) 使用new运算符

new运算符的操作数的组成如下：

- 委托类型名
- 一组圆括号，其中包含作为调用列表中的第一个成员的方法的名字。方法可以是实例方法或静态方法。

```
del1 = new MyDel( myInstObj.MyM1 );
del2 = new MyDel( SClass.OtherM2 );
```

(2)使用快捷语法

快键语法，它仅由方法说明符构成。之所以能这样，是因为在方法名称和其相应的委托类型之间有隐式转换。

```
del1 = myInstObj.MyM1;
del2 = SClass.OtherM2;
```

5. 赋值委托

由于委托是引用类型，我们可以通过给它赋值来改变包含在委托变量中的方法地址引用。旧的引用会被垃圾回收器回收。

```
MyDel del;
del = myInstaObj.MyM1; //委托初始化
del = SClass.OtherM2;//委托重新赋值，旧的引用将被回收
```

6. 组合委托

委托可以使用额外的运算符来组合。这个运算最终会创建一个新的委托，其调用列表是两个操作数的委托调用列表的副本的连接。

委托是恒定的，操作数委托创建后不会被改变。 委托组合拷贝的是操作数的副本 。

```
MyDel del1 = myObj.MyMethod;
MyDel del2 = SClass.OtherM2;
MyDel del3 = del1 + del2;   //组合调用列表
```

7. 委托加减运算

可以使用+=运算符，为委托新增方法。

同样可以使用-=运算符，为委托移除方法。

```
MyDel del = myObj.MyMethod;
del += SClass.OtherM2; // 增加方法
del -= myObj.MyMethod; // 移除方法
```

8. 委托调用

委托调用跟方法调用类似。委托调用后，调用列表的每个方法将会被执行。

在调用委托前，应判断委托是否为空 。调用空委托会抛出异常。

```
if(null != del)
{
     del();//委托调用
}
```

9. 匿名方法

匿名方法是在初始化委托时内联声明的方法。

基本结构：

```
deleage( 参数 ) { 语句块 }
```

例如：

```
delegate int MyDel (int x); //定义一个委托

MyDel del = delegate( int x){ return x; };
```

从上面我们可以看到， 匿名方法是不会显示声明返回值的 。

10. Lambda表达式

Lambda表达式主要用来简化匿名方法的语法。在匿名方法中，delegate关键字有点多余，因为编译器已经知道我们将方法赋值给委托。通过几个简单步骤，我们就可以将匿名方法转换为Lambda表达式：

- 删除delegate关键字
- 在参数列表和匿名方法主体之间防Lambda运算符=>。Lambda运算符读作”goes to”。

```
MyDel del = delegate( int x) { return x; };//匿名方法
MyDel del2 = (int x) => {return x;};//Lambda表达式
MyDel del3 = x => {return x};//简写的Lambda表达式
```

来自： http://www.codeceo.com/article/csharp-delegates-usage.html