五分钟重温委托，匿名方法，Lambda，泛型委托，表达式树 - 文章

这些对老一代的程序员都是老生常谈的东西，没什么新意，对新生代的程序员却充满着魅力。曾经新生代，好多都经过漫长的学习，理解，实践才能掌握委托，表达式树这些应用。今天我尝试用简单的方法叙述一下，让大家在五分钟内看完这篇博客。

第一分钟：委托
有些教材，博客说到委托都会提到事件，虽然事件是委托的一个实例，但是为了理解起来更简单，今天只谈委托不谈事件。先上一段代码：

下边的代码，完成了一个委托应用的演示。一个委托分三个步骤：

```
C#
public partial class WebForm3 : System.Web.UI.Page
{
        //step01：首先用delegate定义一个委托 。
        public delegate int CalculatorAdd(int x, int y);
        protected void Page_Load(object sender, EventArgs e)
        {
                //step03：用这个方法来实例化这个委托。
                CalculatorAdd cAdd = new CalculatorAdd(Add);
                //int result = cAdd(5, 6);
                int result = cAdd.Invoke(5,6);
        }
        // step02：声明一个方法来对应委托。
        public int Add(int x, int y)
        {
                return x + y;
        }
}
```

step01：首先用delegate定义一个委托 。

step02：声明一个方法来对应委托。

step03：用这个方法来实例化这个委托。

至此，一个委托的应该就完成了，就可以调用委托了。

第二分钟：匿名方法

在上一分钟已经知道了，完成一个委托应用分三步走，缺一步都不行，如果要跨大步，当心步子大了扯着蛋。但是微软不怕扯着蛋，非要把三步做成两步来走啊！所以微软就用匿名方法来简化上边的三个步骤。匿名方法这个玩意儿怎么说呢，在C#中完全是可有可无的东西，只是为C#锦上添花，有人别出心裁给它取个名字叫语法糖。

C#

```
public partial class WebForm3 : System.Web.UI.Page
{
        //step01：首先用delegate定义一个委托
        public delegate int CalculatorAdd(int x, int y);
        protected void Page_Load(object sender, EventArgs e)
        {
                //step02：用这样的写法 delegate(int x, int y) { return x + y; }，把一个方法赋值给委托
                CalculatorAdd cAdd = delegate(int x, int y) { return x + y; };
                int result = cAdd.Invoke(5, 6);
        }
}
```

step01：首先用delegate定义一个委托 。

step02：用这样的写法 delegate(int x, int y) { return x + y; }，把一个方法赋值给委托，其实这种写法就是匿名方法。

这时会惊奇的发现，这不是三步当着两步走了哇？

第三分钟：Lambda表达式

原本很简单的程序，加上几个delegate关键字，这代码一下就变得深奥了，深奥的东西懂的人就变少了，所以这个还可以作为加薪的筹码。但是微软对C#的设计理念是简单易用。微软就想方设法的来简化delegate(int x, int y) { return x + y; }这个匿名方法，Lambda就出现了。下边我来看几种lambda表达式的写法：

C#

```
public partial class WebForm3 : System.Web.UI.Page
{
        public delegate int CalculatorAdd(int x, int y);
        protected void Page_Load(object sender, EventArgs e)
        {
                //方法一：
                CalculatorAdd cAdd1 = (int x, int y) => { return x + y; };
                int result1 = cAdd1(5, 6);
                //方法二：
                CalculatorAdd cAdd2 = (x, y) => { return x + y; };
                int result2 = cAdd2(5, 6);
```

```
                //方法三：
                CalculatorAdd cAdd3 = (x, y) => x + y;
                int result3 = cAdd2(5, 6);
        }
}
```

方法一：简单的把delegate去掉，在()与{}之间加上 “=>”。

方法二：在方法一的基础上把参数类型都干掉了。

方法三：要干就干彻底些，把{},以及return关键字都去掉了。

这几种方法随便怎么写都行，不过就是害苦了初学者，一会儿看到这种写法，一会儿看到那种写法，把人搞的神魂颠倒人，如果没人指点，确实会迷糊，难就难在这儿。

第四分钟：泛型委托

随着.net版本的不升级，新版本总要区别于旧版本吧，不然微软的工程师怎么向他们的老大交差呀？所以微软又来玩新花样了。

C#

```
public partial class WebForm3 : System.Web.UI.Page
{
        protected void Page_Load(object sender, EventArgs e)
        {
                //方法一：
                Funcint,int,int> cAdd1 = (int x, int y) => { return x + y; };
                int result1 = cAdd1(5, 6);
                //方法二：
                Funcint, int, int> cAdd2 = (x, y) => { return x + y; };
                int result2 = cAdd2(5, 6);
                //方法三：
                Funcint, int, int> cAdd3 = (x, y) => x + y;
                int result3 = cAdd2(5, 6);
        }
}
```

不管是匿名方法还是Lambda表达式，完成一个委托的应用，都逃不过两个步骤，一步是定义一个委托，另一步是用一个方法来实例化一个委托。 微软干脆把这两步都合成一步来走了。用Func来简化一个委托的定义。

至此一个委托的应用就可用 Func cAdd3 = (x, y) => x + y; 这样一句话来完成了，其中的Func就是所谓的泛型委托。

第五分钟：表达式树

表达式树其实与委托已经没什么关系了，非要扯上关系，那就这么说吧，表达式树是存放委托的容器。如果非要说的更专业一些，表达式树是存取Lambda表达式的一种数据结构。要用Lambda表达式的时候，

直接从表达式中获取出来，Compile()就可以直接用了。如下代码：

C#