**[Lambda表达式详解](http://www.cnblogs.com/knowledgesea/p/3163725.html)**

**前言**

        1、天真热，程序员活着不易，星期天，也要顶着火辣辣的太阳，总结这些东西。

        2、夸夸lambda吧：简化了匿名委托的使用，让你让代码更加简洁，优雅。据说它是微软自c#1.0后新增的最重要的功能之一。

**lambda简介**

     lambda运算符：所有的lambda表达式都是用新的lambda运算符 " => ",可以叫他，“转到”或者 “成为”。运算符将表达式分为两部分，左边指定输入参数，右边是lambda的主体。

        lambda表达式：

               1.一个参数：param=>expr

               2.多个参数：（param-list）=>expr

        上面这些东西，记着，下面我们开始应用并阐述lambda，让你乐在其中。

**lambda应用阐述**

         阐述这技术，我先上一个例子，然后再慢慢深入分析。例子如下：

namespace 阐述lambda

{

public class Person

{

public string Name { get; set; }

public int Age { get;set; }

}

class Program

{

public static List<Person> PersonsList()

{

List<Person> persons = new List<Person>();

for (int i = 0; i < 7; i++)

{

Person p = new Person() { Name = i + "儿子", Age = 8 - i, };

persons.Add(p);

}

return persons;

}

static void Main(string[] args)

{

List<Person> persons = PersonsList();

persons = persons.Where(p => p.Age > 6).ToList(); //所有Age>6的Person的集合

Person per = persons.SingleOrDefault(p => p.Age == 1); //Age=1的单个people类

persons = persons.Where(p => p.Name.Contains("儿子")).ToList(); //所有Name包含儿子的Person的集合

}

}

}

     看啦上面的例子，相信你能看出它确实是个甜枣，呵呵，下面我们来看下（p=>p.Age>6）这样的表达式，到底是怎么回事。。

     首先我们看下委托

//委托 逛超市

delegate int GuangChaoshi(int a);

static void Main(string[] args)

{

GuangChaoshi gwl = JieZhang;

Console.WriteLine(gwl(10) + ""); //打印20，委托的应用

Console.ReadKey();

}

//结账

public static int JieZhang(int a)

{

return a + 10;

}

    再看表达式

//委托 逛超市

delegate int GuangChaoshi(int a);

static void Main(string[] args)

{

// GuangChaoshi gwl = JieZhang;

GuangChaoshi gwl = p => p + 10;

Console.WriteLine(gwl(10) + ""); //打印20，表达式的应用

Console.ReadKey();

}

     委托跟表达式的两段代码，我们可以看出一些东东吧：其实表达式（p => p + 10;）中的 p 就代表委托方法中的参数，而表达式符号右边的 p+10，就是委托方法中的返回结果。 大侠绕道，小虾理解下。

    下面再上两个稍微复杂点的理解理解。

    1.多参数的

//委托 逛超市

delegate int GuangChaoshi(int a,int b);

static void Main(string[] args)

{

GuangChaoshi gwl = (p,z) => z-(p + 10);

Console.WriteLine(gwl(10,100) + ""); //打印80，z对应参数b，p对应参数a

Console.ReadKey();

}

        2. lambda主体运算复杂

/// <summary>

/// 委托 逛超市

/// </summary>

/// <param name="a">花费</param>

/// <param name="b">付钱</param>

/// <returns>找零</returns>

delegate int GuangChaoshi(int a,int b);

static void Main(string[] args)

{

GuangChaoshi gwl = (p, z) =>

{

int zuidixiaofei = 10;

if (p < zuidixiaofei)

{

return 100;

}

else

{

return z - p - 10;

}

};

Console.WriteLine(gwl(10,100) + ""); //打印80，z对应参数b，p对应参数a

Console.ReadKey();

}

上面这些例子，好好理解下，下面我要介绍一个系统指定的 Fun<T>委托。

**Func<T>委托**

 T 是参数类型，这是一个泛型类型的委托，用起来很方便的。

 先上例子

static void Main(string[] args)

{

Func<int, string> gwl = p => p + 10 + "--返回类型为string";

Console.WriteLine(gwl(10) + ""); //打印‘20--返回类型为string’，z对应参数b，p对应参数a

Console.ReadKey();

}

说明：我们可以看到，这里的p为int 类型参数， 然而lambda主体返回的是string类型的。

再上一个例子

static void Main(string[] args)

{

Func<int, int, bool> gwl = (p, j) =>

{

if (p + j == 10)

{

return true;

}

return false;

};

Console.WriteLine(gwl(5,5) + ""); //打印‘True’，z对应参数b，p对应参数a

Console.ReadKey();

}

说明：从这个例子，我们能看到，p为int类型，j为int类型，返回值为bool类型。

看完上面两个例子，相信大家应该明白啦Func<T>的用法：多个参数，前面的为委托方法的参数，最后一个参数，为委托方法的返回类型。

**lambda表达式树动态创建方法**

static void Main(string[] args)

{

//i\*j+w\*x

ParameterExpression a = Expression.Parameter(typeof(int),"i"); //创建一个表达式树中的参数，作为一个节点，这里是最下层的节点

ParameterExpression b = Expression.Parameter(typeof(int),"j");

BinaryExpression be = Expression.Multiply(a,b); //这里i\*j,生成表达式树中的一个节点，比上面节点高一级

ParameterExpression c = Expression.Parameter(typeof(int), "w");

ParameterExpression d = Expression.Parameter(typeof(int), "x");

BinaryExpression be1 = Expression.Multiply(c, d);

BinaryExpression su = Expression.Add(be,be1); //运算两个中级节点，产生终结点

Expression<Func<int, int, int, int, int>> lambda = Expression.Lambda<Func<int, int, int, int, int>>(su,a,b,c,d);

Console.WriteLine(lambda + ""); //打印‘(i,j,w,x)=>((i\*j)+(w\*x))’，z对应参数b，p对应参数a

Func<int, int, int, int, int> f= lambda.Compile(); //将表达式树描述的lambda表达式，编译为可执行代码，并生成该lambda表达式的委托；

Console.WriteLine(f(1, 1, 1, 1) + ""); //打印2

Console.ReadKey();

}

这段代码，放上来，仔细理解下，理解透彻啦，lambda表达式基本上也没什么啦。呵呵。。