**c# Action,Func,Predicate委托**

时间2014-04-15 16:45:00 [博客园-所有随笔区](http://www.tuicool.com/sites/QbYvau) *原文*  <http://www.cnblogs.com/niuge/p/3666621.html>

/// <summary>

/// Action泛型委托：.netframework4下 可以有0到16个参数

/// </summary>

public void TestAction()

{

Action a1 = () => Console.WriteLine("嘿嘿");//Lambda表达式

a1();

Action<string> a2 = s => Console.WriteLine("你好," + s);

a2("阿牛");

Action<int, int> a3 = delegate(int a, int b) {//匿名方法

Console.WriteLine("a+b=" + (a + b));

};

a3(4, 5);

}

/// <summary>

/// Func泛型委托：

/// </summary>

public void TestFunc()

{

Func<string> func1 = () => "你好";//表达式Lambda

Console.WriteLine(func1());

Func<string, string> func2 = s => { return "你好" + s; };//语句Lambda

Console.WriteLine(func2("阿牛"));

Func<int, int, int> func3 = delegate(int a, int b)//匿名方法

{

return a + b;

};

Console.WriteLine(func3(4, 5));

}

/// <summary>

/// Predicate泛型委托

/// </summary>

public void TestPredicate()

{

Predicate<int> pre1 = a => a == 2;

if (pre1(3))

{

Console.WriteLine("3==2");

}

else

{

Console.WriteLine("3<>2");

}

List<string> list = new List<string>();

list.Add("A1");

list.Add("A2");

list.Add("B1");

List<string> list2 = list.FindAll(s => s.StartsWith("B"));

List<string> list3 = list.FindAll(delegate(string s) {

return s.StartsWith("B");

});

}

**一 delegate、Delegate、MulticastDelegate的区别**

1 Delegate：是一个抽象基类，它引用静态方法或引用类实例及该类的实例方法。然而，只有系统和编译器可以显式地从 Delegate 类派生出委托类型。

2 MulticastDelegate：是一个继承于Delegate的类，其拥有一个带有链表格式的委托列表，该列表称为调用列表，在调用多路广播委托 时，将按照调用列表中的委托出现的顺序来同步调用这些委托。平常我们声明一个delegate的类型，都是继承于MulticastDelegate类的 (注意：不能显式地从此类进行派生。[这点与Delegate](http://msdn.microsoft.com/zh-cn/library/system.delegate%28v=VS.80%29.aspx)类是一样的，只有系统和编译器可以显示地进行派生）。

3 delegate 是一个C#关键字，用来定义一个新的委托类型（继承自MulticastDelegate类）。

public delegate void oh(string a);

oh my = new oh( delegate(string a) { Console.WriteLine("a={0}", a); } );

my("good");

一句话， 不能带分号结束

oh my = new oh( a => Console.WriteLine("my a={0}", a) );

my("good");

多句话， 使用 { }

oh my = new oh(a => { a += " friday"; Console.WriteLine("my a={0}", a); });

my("good");

有返回值

public delegate string oh(string a);

oh my = new oh(a => { Console.WriteLine("my a={0}", a); return string.Format("my11 a={0}", a); });

Console.WriteLine(my("good"));

带有 out 参数：

public delegate string oh(string a, out int b);

oh my = new oh( call );

my += (a, out b) => {b= 3000; return "ggg"; }; // 这个是不允许的，出错。

my += delegate(string a, out int b) // 这是允许的

{

a += " queue";

b = 2000;

Console.WriteLine("222 a={0} ; b={1}", a, b);

return string.Format("a={0} ; b={1}", a, b);

};

Console.WriteLine(my("good", out ot));

private static string call(string a, out int b) {

a += " maver";

b = 1000;

Console.WriteLine("111 a={0} ; b={1}", a, b);

return string.Format("a={0} ; b={1}", a, b);

}

匿名委托与 lambda 表达

Action 无返回值：参数可以有多个

Action act = () => Console.WriteLine("This is action test without parameters");

act();

Action<string> act1 = T => Console.WriteLine("T={0}", T);

act1("Good");

Action<string, int> act2 = (s, t) => { s += " 2014";

t += 120;

string str = string.Format("s={0}, t={1}", s, t);

Console.WriteLine(str);

};

act2("Friday", 200);

Func 必须有且只有一个返回值：参数可以有多个， 但 Func 的最后一个参数永远是返回值得类型。

Func<int> f1 = () => 200+300;

Console.WriteLine("Return: {0}", f1() );

Func<int> f1 = delegate() { return 300; };

注意表达式不需要 return , 单个表达式不需要

Func<int, int, string> f2 = (a,b)=>string.Format("{0}+{1}={2}",a,b,(a+b));

Console.WriteLine("Return: {0}", f2(200, 150) );

Func<int, int, string> f2 = (a, b) => { a += 100;

b += 200;

return string.Format("{0}+{1}={2}", a, b, (a + b));

};

Console.WriteLine("Return: {0}", f2(200, 150) );

注意表达式不需要 return , 单个表达式不需要

Func<int, int, int> f2 = (a, b) => a + b;

Console.WriteLine("Return: {0}", f2(200, 150) );

注意表达式必须 return , 而且需要分号结束符；

Func<int, int, int> f2 = delegate(int a, int b) { return a + b; };

Console.WriteLine("Return: {0}", f2(200, 150) );

Predicate 参数只有一个，返回值是 Boolean，且不需要指定返回值的类型。

单个表达式不需要return

Predicate<string> p1 = a => a.StartsWith("B",false,null);

Console.WriteLine("p1: {0}", p1("boby"));

复合表达式需要 return

Predicate<string> p1 = a => { a += "FFFF"; return a.StartsWith("B", false, null); };

Console.WriteLine("p1: {0}", p1("boby"));

匿名代理 delegate

Predicate<string> p1 = delegate(string a) { a += " wood"; return a.EndsWith("D",true,null); };

Console.WriteLine("p1: {0}", p1("boby"));

int[] iarr = new int[] { 120, 30, 46, 59 };

int cnt = iarr.Count(a => { a += 200; return a % 2 == 0; });

Console.WriteLine("Count: {0}", cnt );