个人资料

王超_Eric

访问： 4965次
积分： 185
等级： BLOG 2
排名： 千里之外

原创： 14篇 转载： 2篇
译文： 0篇 评论： 1条

C# => Lambda表达式理解

标签： lambda c#

2016-08-18 17:36 465人阅读 评论(0) 收藏 举报

分类： C#（5）

本文参考网上的博客和找到的资料加上个人理解编写的，主要的代码借鉴：
http://www.cnblogs.com/knowledgesea/p/3163725.html、百度百科

希望能够帮助理解lambda表达式。

定义："Lambda表达式"是一个匿名函数，是一种高效的类似于函数式编程的表达式

好处：Lambda简化了匿名委托的使用，减少开发中需要编写的代码量。
具体内容：它可以包含表达式和语句，并且可用于创建委托或表达式目录树类型，支持带有可绑定到委托或表达式树的输入参数的内联表达式。
写法：所有Lambda表达式都使用Lambda运算符=>，该运算符读作"goes to"。Lambda运算符的左边是输入参数(如果有)，右边是表达式或语句块。Lambda表达式x => x * x读作"x goes to x times x"。

接下来从例子中慢慢学习：

```csharp
namespace LambdaLearn
{
    public class Person
    {
        public string Name { get; set; }
        public int Age  {  get;set; }
    }
    class Program
    {

        public static List<Person> PersonsList()//方法返回Person类的List集合
        {
            List<Person> persons = new List<Person>();
            for (int i = 0; i < 7; i++)
            {
                Person p = new Person() { Name = i + "人物年龄", Age = 8 - i, };
                persons.Add(p);
            }
            return persons;
        }

        static void Main(string[] args)
        {
            List<Person> persons0 = PersonsList();
            List<Person> persons1 = persons.Where(p => p.Age > 6).ToList();   //所有Age>6的Person
```

ShaderLab - 第一天 - Sh (114)

评论排行

unity - 二进制文件操作-存 (1)
unity - 对象池技术的实现 (0)
unity - 屏幕滑动操作方案 (0)
unity - 优化你的游戏 (0)
Unity - 延时操作方案 (0)
Unity - 判断当前所属平台 (0)
C# => Lambda表达式理 (0)
Unity - Fixed function sh (0)
C#socket建立服务器并进 (0)
C#委托的学习 (0)

推荐文章

* 而立之年——三线城市程序员的年终告白
* Java集合框架中隐藏的设计套路
* Python脚本下载今日头条视频(附加Android版本辅助下载器)
* 人工智能的冷思考
* React Native 实战系列教程之热更新原理分析与实现

最新评论

unity - 二进制文件操作-存储与读
qq_28635239: BinaryWriter bw = new BinaryWriter(new FileStream ...


```
的集合
            Person per = persons.SingleOrDefault(p => p.Age == 1);  //Age=1的单个people类
            List<Person> persons2 = persons.Where(p => p.Name.Contains("年龄")).ToList();   //所有Name包含年龄的Person的集合
        }
    }
}
```

从例一可以看出一点lambda表达式的简单用法，接着往下看。

Lambda简化了匿名委托的使用，我们可以看一看下面例子怎样简化的。如果委托与事件不是很懂请看：http://blog.csdn.net/u013236878/article/details/52243017

例二：用lambda表达式简化委托

利用委托处理方法：

```
[csharp]
//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;
        }
```

利用lambda表达式处理方法：

```
[csharp]
//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，我们应该能明白了：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。

再看一个稍微复杂一点的例子：

```
[csharp]
//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();
        }
[code]csharpcode:
/// <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();
        }
```

接下来看一下lambda的具体写法形式:隐式表达即没有指定参数类型(因为编译器能够根据上下文直接推断参数的类型)

```
[csharp]
(x, y) => x * y              //多参数，隐式类型=>表达式
x => x * 5                   //单参数，隐式类型=>表达式
x => { return x * 5; }       //单参数，隐式类型=>语句块
(int x) => x * 5             //单参数，显式类型=>表达式
(int x) => { return x * 5; } //单参数，显式类型=>语句块
() => Console.WriteLine()    //无参数
```

看完以上内容，理解lambda表达式已经不会有太大问题了，接下来的内容稍微会深一点（至少我理解了很久Orz...）

简单了解一下lambda背景：

Lambda 用在基于方法的 LINQ 查询中，作为诸如 Where 和 Where 等标准查询运算符方法的参数。

使用基于方法的语法在 Enumerable 类中调用 Where 方法时（像在 LINQ to Objects 和 LINQ to XML 中那样），参数是委托类型 System.:::.Func<(Of <(T, TResult>)>)。使用 Lambda 表达式创建委托最为方便。例如，当您在 System.Linq.:::.Queryable 类中调用相同的方法时（像在 LINQ to SQL 中那样），则参数类型是 System.Linq.Expressions.:::.Expression<Func>，其中 Func 是包含至多五个输入参数的任何 Func 委托。同样，Lambda 表达式只是一种用于构造表达式目录树的非常简练的方式。尽管事实上通过 Lambda 创建的对象的类型是不同的，但 Lambda 使得 Where 调用看起来类似。

背景这种想要深入研究的可以都了解一下，本文只是帮助了解lambda，这里就不多说了。

下列规则适用于 Lambda 表达式中的变量范围：

捕获的变量将不会被作为垃圾回收，直至引用变量的委托超出范围为止。

在外部方法中看不到 Lambda 表达式内引入的变量。

Lambda 表达式无法从封闭方法中直接捕获 ref 或 out 参数。

Lambda 表达式中的返回语句不会导致封闭方法返回。

Lambda 表达式不能包含其目标位于所包含匿名函数主体外部或内部的 goto 语句、break 语句或 continue 语句。

Lambda表达式的本质是“匿名方法”，即当编译我们的程序代码时，“编译器”会自动将“Lambda表达式”转

换为“匿名方法”，如下例：

```csharp
string[] names={"agen","balen","coure","apple"};
string[] findNameA=Array.FindAll<string>(names,delegate(string v){return v.StartsWith("a");});
string[] findNameB=Array.FindAll<string>(names,v=>v.StartsWith("a"));
```

上面中两个FindAll方法的反编译代码如下：

```csharp
string[]findNameA=Array.FindAll<string>(names,delegate(stringv){returnv.StartsWith("a");});
string[]findNameB=Array.FindAll<string>(names,delegate(stringv){returnv.StartsWith("a");});
```

Lambda表达式的语

参数列表 **=>** 语句或

其中“参数列”中可包 与委托对应），如果参数列中有0个或1个以上参数，则必须使用括号括住参数列，如下：

() => Console.Write("0个参数")

I => Console.Write("1个参数时参数列中可省略括号，值为：{0}",i)

(x,y) => Console.Write("包含2个参数，值为：{0}*{1}",x,y)

而“语句或语句块”中如果只有一条语句，则可以不用大括号括住否则必须使用，如下：

I => Console.Write("只有一条语句")

I => { Console.Write("使用大括号的表达式"); }

关闭

I => { i++;Console.Write("两条语句的情况"); }

如果“语句或语句块”有返回值时，如果只有一条语句则可以不输写“return”语句，编译器会自动处理，否则必须加上，如下示例：

“Lambda表达式”是委托的实现方法，所以必须遵循以下规则：

访问页面被拦截

1）“Lambda表达式”的参数数量必须和“委托”的参数数量相同；

2）如果“委托”的参数中包括有ref或out修饰符，则“Lambda表达式”的参数列中也必须包括有修饰符；

因为您没有权限访问该页面，
如有疑问请联系您的网络管理

顶 0 踩 0

上一篇 Unity - Fixed function shader学习

下一篇 Unity - 判断当前所属平台

我的同类文章

C#（5）

- unity - 二进制文件操作-存储... 2016-09-05 阅读 975
- Unity - 延时操作方案 2016-08-18 阅读 588
- C#socket建立服务器并进行... 2016-08-18 阅读 293
- C#委托的学习 2016-08-18 阅读 54
- C#正则表达式的学习 2016-08-18 阅读 26

参考知识库

.NET知识库

2860 关注 | 815 收录

猜你在找

JavaAndroid客户端和C#服务端Web Api接口开发
使用C#开发信息管理系统
C#开发微信订阅号、服务号视频教程
XML编程
.NET平台和C#编程从入门到精通
知识库分享系列 二NETASPNET
编程的智慧
转载JAVA知识点上
Java面试题集收藏
Java面试题全集上

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap