## C#中的索引器的简单理解和用法

索引器是一种特殊的类成员，它能够让对象以类似数组的方式来存取，使程序看起来更为直观，更容易编写。 下面我们详细的来了解一下。

作者：宋坤明来源：博客园|2013-06-05 10:11

[收藏](http://developer.51cto.com/art/201306/javascript:favorBox('open');" \o "一键收藏，随时查看，分享好友！" \t "http://developer.51cto.com/art/201306/_self)

[分享](http://developer.51cto.com/art/201306/javascript:;)

索引器是一种特殊的类成员，它能够让对象以类似数组的方式来存取，使程序看起来更为直观，更容易编写。

**1、索引器的定义**

C#中的类成员可以是任意类型，包括数组和集合。当一个类包含了数组和集合成员时，索引器将大大简化对数组或集合成员的存取操作。

定义索引器的方式与定义属性有些类似，其一般形式如下：

1. [修饰符] 数据类型 **this**[索引类型 index]
3. {
4. **get**{//获得属性的代码}
5. **set**{ //设置属性的代码}
6. }

修饰符包括 **public,protected,private,internal,new,virtual,sealed,override, abstract,extern.**

数据类型是表示将要存取的数组或集合元素的类型。

索引器类型表示该索引器使用哪一类型的索引来存取数组或集合元素，可以是整数，可以是字符串；this表示操作本对象的数组或集合成员，可以简单把它理解成索引器的名字，因此索引器不能具有用户定义的名称。 例如：

1. **class** Z
2. {
3. //可容纳100个整数的整数集
4. **private** **long**[] arr = **new** **long**[100];
5. //声明索引器
6. **public** **long** **this**[**int** index]
7. {
8. **get**
9. { //检查索引范围
10. **if** (index < 0 || index >= 100)
11. {
12. **return** 0;
13. }
14. **else**
15. {
16. **return** arr[index];
17. }
18. }
19. **set**
20. {
21. **if** (!(index < 0 || index >= 100))
22. {
23. arr[index] = value;
24. }
25. }
26. }

**2、索引器的使用**

通过索引器可以存取类的实例的数组成员，操作方法和数组相似，一般形式如下：

**对象名[索引]**

其中索引的数据类型必须与索引器的索引类型相同。例如：

1. Z  z=**new**  z();
2. z[0]=100;
3. z[1]=101;
4. Console.WriteLine(z[0]);

表示先创建一个对象z，再通过索引来引用该对象中的数组元素。

C#中并不将索引器的类型限制为整数。例如，可以对索引器使用字符串。通过搜索集合内的字符串并返回相应的值，可以实现此类的索引器。由于访问器可以被重载，字符串和整数版本可以共存。

example：

1. **class** DayCollection
2. {
3. **string**[] days={"Sun","Mon","Tues","Wed","Thurs","Fri","Sat"};
4. **private** **int** GetDay(**string** testDay)
5. {
6. **int** i=0;
7. **foreach**(**string** day **in** days)
8. {
9. **if**(day==testDay)
10. **return** i;
11. i++;
12. }
13. **return** -1;
14. }
15. **public** **int** **this**[**string** day]
16. {
17. **get**{**return** (GetDay(day))}
18. }
19. }

22. **static** **void** Main(**string**[] args)
23. {
24. DayCollection week=**new** DayCollection();
25. Console.WriteLine("Fri:{0}",week["Fri"]);
26. Console.WriteLine("ABC:{0}",week["ABC"]);
27. }

结果：Fri:5

ABC:-1

**3、接口中的索引器**

在接口中也可以声明索引器，接口索引器与类索引器的区别有两个：一是接口索引器不使用修饰符；二是接口索引器只包含访问器get或set，没有实现语句。访问器的用途是指示索引器是可读写、只读还是只写的，如果是可读写的，访问器get或set均不能省略；如果只读的，省略set访问器；如果是只写的，省略get访问器。

例如：

1. **public** **interface** IAddress
2. {
3. **string** **this**[**int** index]{**get**;**set**;}
4. **string** Address{**get**;**set**;}
5. **string** Answer();
6. }

表示所声明的接口IAddress包含3个成员：一个索引器、一个属性和一个方法，其中，索引器是可读写的。

**4、索引器与属性的比较**

索引器与属性都是类的成员，语法上非常相似。索引器一般用在自定义的集合类中，通过使用索引器来操作集合对象就如同使用数组一样简单；而属性可用于任何自定义类，它增强了类的字段成员的灵活性。

属        性                                          索  引  器

|  |  |
| --- | --- |
| 允许调用方法，如同公共数据成员 | 允许调用对象上的方法，如同对象是一个数组 |
| 可通过简单的名称进行访问 | 可通过索引器进行访问 |
| 可以为静态成员或实例成员 | 必须为实例成员 |
| 其get访问器没有参数 | 其get访问器具有与索引器相同的形参表 |
| 其set访问器包含隐式value参数 | 除了value参数外，其set访问器还具有与索引器相同的形参表 |

# **[C# 索引器使用总结](http://www.cnblogs.com/lxblog/p/3940261.html)**

**1、索引器（Indexer）：**

**索引器允许类或者结构的实例按照与数组相同的方式进行索引。索引器类似于属性，不同之处在于他们的访问采用参数。**

最简单的索引器的使用

/// <summary>

/// 最简单的索引器

/// </summary>

public class IDXer

{

private string[] name=new string[10];

//索引器必须以this关键字定义，其实这个this就是类实例化之后的对象

public string this[int index]

{

get

{

return name[index];

}

set

{

name[index] = value;

}

}

}

public class Program

{

static void Main(string[] args)

{

//最简单索引器的使用

IDXer indexer = new IDXer();

//“=”号右边对索引器赋值，其实就是调用其set方法

indexer[0] = "张三";

indexer[1] = "李四";

//输出索引器的值，其实就是调用其get方法

Console.WriteLine(indexer[0]);

Console.WriteLine(indexer[1]);

Console.ReadKey();

}

}

**2、索引器与数组的区别：**

* **索引器的索引值（Index）类型不限定为整数：**

 用来访问数组的索引值（Index）一定为整数，而索引器的索引值类型可以定义为其他类型。

* **索引器允许重载**

        一个类不限定为只能定义一个索引器，只要索引器的函数签名不同，就可以定义多个索引器，可以重载它的功能。

* **索引器不是一个变量**

        索引器没有直接定义数据存储的地方，而数组有。索引器具有Get和Set访问器。

**3、索引器与属性的区别：**

* **索引器以函数签名方式 this 来标识，而属性采用名称来标识，名称可以任意**
* **索引器可以重载，而属性不能重载。**
* **索引器不能用static 来进行声明，而属性可以。索引器永远属于实例成员，因此不能声明为static。**

以字符串作为下标，对索引器进行存取：

//以字符串为下标的索引器

public class IDXer2

{

private Hashtable name = new Hashtable();

//以字符串为下标的索引器

public string this[string index]

{

get

{

return name[index].ToString();

}

set

{

name.Add(index, value);

}

}

}

public class Program

{

static void Main(string[] args)

{

//以字符串为下标的索引器

IDXer2 indexer2 = new IDXer2();

indexer2["A01"] = "张三";

indexer2["A02"] = "李四";

Console.WriteLine(indexer2["A01"]);

Console.WriteLine(indexer2["A02"]);

Console.ReadKey();

}  
 }

  多参数索引器及索引器的重载

/// <summary>

/// 成绩类

/// </summary>

public class Scores

{

/// <summary>

/// 学生姓名

/// </summary>

public string StuName { get; set; }

/// <summary>

/// 课程ID

/// </summary>

public int CourseId { get; set; }

/// <summary>

/// 分数

/// </summary>

public int Score { get; set; }

}

/// <summary>

/// 查找成绩类（索引器）

/// </summary>

public class FindScore

{

private List<Scores> listScores;

public FindScore()

{

listScores = new List<Scores>();

}

//索引器 通过名字&课程编号查找和保存成绩

public int this[string stuName, int courseId]

{

get

{

Scores s = listScores.Find(x => x.StuName == stuName && x.CourseId == courseId);

if (s != null)

{

return s.Score;

}

else

{

return -1;

}

}

set

{

listScores.Add(new Scores() { StuName = stuName, CourseId = courseId, Score = value });

}

}

//索引器重载，根据名字查找所有成绩

public List<Scores> this[string stuName]

{

get

{

List<Scores> tempList = listScores.FindAll(x => x.StuName == stuName);

return tempList;

}

}

}

static void Main(string[] args)

{

//多参数索引器和索引器重载

FindScore fScore = new FindScore();

fScore["张三", 1] = 98;

fScore["张三", 2] = 100;

fScore["张三", 3] = 95;

fScore["李四", 1] = 96;

//查找 张三 课程编号2 的成绩

Console.WriteLine("李四 课程编号2 成绩为:" + fScore["李四", 1]);

//查找所有张三的成绩

List<Scores> listScores = fScore["张三"];

if (listScores.Count > 0)

{

foreach (Scores s in listScores)

{

Console.WriteLine(string.Format("张三 课程编号{0} 成绩为:{1}", s.CourseId, s.Score));

}

}

else

{

Console.WriteLine("无该学生成绩单");

}

Console.ReadKey();

}