

C# 中的枚举器

Iterators In C#

Jesse Liberty

译者：张子阳

www.tracefact.net

jimmy_dev@163.com

出处: <http://www.ondotnet.com/pub/a/dotnet/2004/06/07/liberty.html>

术语表

Iterator: 枚举器

如果你正在创建一个表现和行为都类似于集合的类, 允许类的用户使用 foreach 语句对集合中的成员进行枚举将会是很方便的。这在 C# 2.0 中比 C# 1.1 更容易实现一些。作为演示, 我们先在 C# 1.1 中为一个简单的集合添加枚举, 然后我们修改这个范例, 使用新的 C#2.0 枚举构建方法。

我们将以创建一个简单化的 List Box 作为开始, 它将包含一个 8 字符串的数组和一个整型, 这个整型用于记录数组中已经添加了多少字符串。构造函数将对数组进行初始化并使用传递进来的参数填充它。

```
public ListBox(params string[] initialStrings)
{
    strings = new String[8];

    foreach (string s in initialStrings)
    {
        strings[ctr++] = s;
    }
}
```

除此以外, ListBox 类还需要一个 Add 方法(进行添加 string 的操作) 和 一个返回数组中字符串个数的方法。

```
public void Add(string theString)
{
    strings[ctr] = theString;
    ctr++;
}

public int GetNumEntries()
{
    return ctr;
}
```

NOTE: 实际开发中, 通常使用 ArrayList, 而不是固定大小的数组。在这里为了程序简单就没有做数组下标越界的检测。

从感觉上看, ListBox 像是一个集合, 如果可以使用集合中通常使用的 foreach 循环来获取 listBox 中的所有字符串将会是非常便利的。如此的话, 可以这样书写代码:

```
ListBox lb = new ListBox("a", "b", "c", "d", "e", "f", "g", "h");
foreach (string s in lb) {
    Console.WriteLine(s);
}
```

但是，会得到这样一个错误：

“Iterator.ListBox” 不包含 “GetEnumerator” 的公共定义，因此 foreach 语句不能作用于 “Iterator.ListBox” 类型的变量

想要使用 foreach 语句，还必须实现 IEnumerable 接口。

这个接口只要求实现一个方法： GetEnumerator。这个方法必须返回一个实现了 IEnumerable 接口的对象。除此以外，我们需要返回的这个对象不仅实现了 IEnumerable，而且知道如何枚举 ListBox 对象。你将需要创建一个 ListBoxEmunerator (在下面描述)：

NOTE: IEnumerable 和 IEnumerator 是不同的接口，请不要搞混了。

```
public IEnumerator GetEnumerator()
{
    return new ListBoxEnumerator();
}
```

现在，ListBox 可以使用 foreach 循环了：

```
ListBox lbt = new ListBox("Hello", "World");

lbt.Add("Who");
lbt.Add("Is");
lbt.Add("John");
lbt.Add("Galt");

foreach (string s in lbt)
{
    Console.WriteLine("Value: {0}", s);
}
```

先是实例化这个 ListBox，并初始了两个字符串，随后又添加了四个。foreach 循环接受 ListBox 实例，并且迭代它，依次返回字符串。输出是：

```
Hello
World
Who
Is
John
```

实现 IEnumerator 接口

注意到 `ListBoxEnumerator` 不仅需要实现 `IEnumerator` 接口, 对于 `ListBox` 类它也需要一些特别了解; 特别是, 它必须可以获得 `ListBox` 的字符串数组并且遍历其所包含的字符串。`IEnumerable` 类和与其相关的 `IEnumerator` 类之间的关系有一点微妙。实现 `IEnumerator` 接口的最好办法是在 `IEnumerable` 类里创建一个嵌套的 `IEnumerator` 类。

```
public class ListBox : IEnumerable
{
    // 嵌套的私有 ListBoxEnumerator 类实现
    private class ListBoxEnumerator : IEnumerator
    {
        // 代码实现...
    }
    // ListBox 类的代码...
}
```

注意 `ListBoxEnumerator` 需要对它所嵌入的 `ListBox` 类的一个引用。你可以通过 `ListBoxEnumerator` 的构造函数来传递。

为了实现 `IEnumerator` 接口, `ListBoxEnumerator` 需要两个方法: `MoveNext` 和 `Reset`, 还有一个属性: `Current`。这些方法和属性的任务是创建一个状态机制, 确保你可以在任何时候得知 `ListBox` 中的哪个元素是当前元素, 并获得那个元素。

在这个例子中, 这种状态机制是通过维护一个标明当前 `string` 的索引值来完成的, 并且, 你可以通过对外部类的 `string` 集合进行索引来返回这个当前的 `string`。为了达到这个目标, 你需要一个成员变量保存对于外部 `ListBox` 对象的引用, 以及一个整型用于保存当前索引。

```
private ListBox lbt;
private int index;
```

每次 `Reset` 方法被调用的时候, `index` 被置为 `-1`。

```
public void Reset()
{
    index = -1;
}
```

每次 `MoveNext` 被调用的时候, 外部类的数组检查时候已经到了末尾, 如果是这样, 方法返回 `false`。如果集合中还有对象, `index` 将增加, 并且方法返回 `true`。

```
public bool MoveNext()
```

```

{
    index++;
    if (index >= lbt.strings.Length)
    {
        return false;
    }else
    {
        return true;
    }
}

```

最后,如果 MoveNext 方法返回 True,foreach 循环将调用 Current 属性。ListBoxEnumerator 的 Current 属性的实现是索引外部类(ListBox)中的集合,并且返回找到的对象(这个例子中,是一个字符串)。注意,返回一个 Object 是因为 IEnumerator 接口中 Current 属性的签名如此。

```

public object Current
{
    get {
        return(lbt[index]);
    }
}

```

在 1.1 中,所有想要通过 foreach 循环来迭代的类都需要实现 IEnumerable 接口,于是,必须创建一个实现了 IEnumerator 的类。最糟的是,enumerator 返回的值并不是类型安全的。记得 Current 属性返回一个 Object 对象;它仅仅简单的假设你所返回的值与 foreach 循环所期望的相符合。

C# 2.0 的解救办法

使用 C# 2.0 这些问题如同五月末的雪般融化了。在这个例子的 2.0 版本中,我重写上面的列表,使用 C# 2.0 的两个新特性:泛型和枚举器。

我以重新定义实现 IEnumerable<string>的 ListBox 作为开始:

```

public class ListBox : IEnumerable<string>

```

这样做确定这个类可以在 foreach 循环中使用,同时确保迭代的值是 string 类型。

现在,从上个例子中挪去整个嵌套类,并且用下面的代码替换 GetEnumerator 方法。

```

public IEnumerator<string> GetEnumerator()
{
    foreach (string s in strings)
    {

```

```
yield return s;
}
}
```

GetEnumerator 方法使用了新的 yield 语句。yield 语句返回一个表达式。yield 语句仅在迭代块中出现，并且返回 foreach 语句所期望的值。那也就是，对 GetEnumerator 的每次调用都将产生集合中的下一个字符串；所有的状态管理已经都为你做好了！

就这样了，你已经完成了。不需要为每个类型实现你自己的 enumerator，不需要创建嵌套类。你已经移除了至少 30 行代码，并且极大地简化了你的代码。程序继续像期望的那样运行，但是状态管理不再是你的任务，所有的都为你做好了。更进一步，由枚举器所返回的值一定是 string 类型，如果你想要返回其他类型，你可以修改 IEnumerable 泛型语句，IEnumerable 泛型语句将反射新类型。

关于 Yield 的更多内容

作为对上一节的一些说明，应该告诉你：实际上，你可以在 yield 语句块中 yield 一个以上的值。这样，下面的语句是完全正确的 C# 语句：

```
public IEnumerator GetEnumerator()
{
    yield return "Who";
    yield return " is";
    yield return "John Galt?";
}
```

假设上面的代码位于一个名为 foo 的类中，你可以这样写：

```
foreach ( string s in new foo())
{
    Console.Write(s);
}
```

输出结果将会是：

```
Who is John Galt?
```

如果你现在停下来思考一下，这些也是之前的代码所做的工作。它遍历了自己的 foreach 循环，并且产生出它所找到的每个 string 字符串。

本文的源代码可以在 <http://www.tracefact.net/SourceCode/Iterators-In-CSharp.rar> 下载。