昵称： 永远薰薰
园龄： 2年10个月
粉丝： 49
关注： 2
+加关注

< 2020年2月 >

日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
1	2	3	4	5	6	7

搜索

找找看

最新随笔

1.提高PowerShell脚本效率的五个常用方法
2.Visual Studio Debugger中七个鲜为人知的小功能
3.浅谈Ubuntu PowerShell——小白入门教程
4.微软SQL Server认证最新信息（17年5月22日更新），感兴趣的进来看看哟
5.一键帮你复制多个文件到多个机器——PowerShell小脚本（内附PS远程执行命令问题解析）
6.Configure Always On Availability Group for SQL Server on RHEL——Red Hat Enterprise Linux上配置SQL Server Always On Availability Group
7.PowerShell管道入门，看看你都会不（管道例子大全）
8.从零开始——PowerShell应用入门（全例子入门讲解）
9.3分钟带你了解PowerShell发展历程——PowerShell各版本资料整理
10.由Find All References引发的思考。，

我的标签

SQL Server(8)
Linux(7)
PowerShell(6)
RHEL(4)
Ubuntu(3)
Cluster(2)
Visual Studio(2)
Always On Availability Group(2)
C#(2)
C# 7.0(1)
更多

积分与排名

积分 - 38280
排名 - 19622

随笔分类 (16)

C#(2)
PowerShell(6)

详解C# Tuple VS ValueTuple（元组类 VS 值元组）

C# 7.0已经出来一段时间了，大家都知道新特性里面有个对元组的优化：ValueTuple。这里利用详尽的例子详解Tuple VS ValueTuple（元组类VS值元组），10分钟让你更了解ValueTuple的好处和用法。

如果您对Tuple足够了解，可以直接跳过章节"回顾Tuple"，直达章节"ValueTuple详解"，查看值元组的炫丽用法。

回顾Tuple

Tuple是C# 4.0时出的新特性，.Net Framework 4.0以上版本可用。

元组是一种数据结构，具有特定数量和元素序列。比如设计一个三元组数据结构用于存储学生信息，一共包含三个元素，第一个是名字，第二个是年龄，第三个是身高。

元组的具体使用如下：

1. 如何创建元组

默认情况.Net Framework元组仅支持1到7个元组元素，如果有8个元素或者更多，需要使用Tuple的嵌套和Rest属性去实现。另外Tuple类提供创造元组对象的静态方法。

- 利用构造函数创建元组：

```
var testTuple6 = new Tuple<int, int, int, int, int, int>(1, 2, 3, 4, 5, 6);
Console.WriteLine($"Item 1: {testTuple6.Item1}, Item 6: {testTuple6.Item6}");

var testTuple10 = new Tuple<int, int, int, int, int, int, int, Tuple<int, int, int>>(1, 2, 3, 4, 5, 6, 7, new Tuple<int, int, int>(8, 9, 10));
Console.WriteLine($"Item 1: {testTuple10.Item1}, Item 10: {testTuple10.Rest.Item3}");
```

- 利用Tuple静态方法构建元组，最多支持八个元素：

```
var testTuple6 = Tuple.Create<int, int, int, int, int, int>(1, 2, 3, 4, 5, 6);
Console.WriteLine($"Item 1: {testTuple6.Item1}, Item 6: {testTuple6.Item6}");

var testTuple8 = Tuple.Create<int, int, int, int, int, int, int, int>(1, 2, 3, 4, 5, 6, 7, 8);
Console.WriteLine($"Item 1: {testTuple8.Item1}, Item 8: {testTuple8.Rest.Item1}");
```

Note：这里构建出来的Tuple类型其实是Tuple<int, int, int, int, int, int, int, Tuple<int>>，因此testTuple8.Rest取到的数据类型是Tuple<int>，因此要想获取准确值需要取Item1属性。

2. 表示一组数据

如下创建一个元组表示一个学生的三个信息：名字、年龄和身高，而不用单独额外创建一个类。

```
var studentInfo = Tuple.Create<string, int, uint>("Bob", 28, 175);
Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age [{studentInfo.Item2}], Height [{studentInfo.Item3}]");
```

3. 从方法返回多个值

当一个函数需要返回多个值的时候，一般情况下可以使用out参数，这里可以用元组代替out实现返回多个值。

```
static Tuple<string, int, uint> GetStudentInfo(string name)
{
    return new Tuple<string, int, uint>("Bob", 28, 175);
}

static void RunTest()
{
    var studentInfo = GetStudentInfo("Bob");
    Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age [{studentInfo.Item2}], Height [{studentInfo.Item3}]");
}
```

4. 用于单参数方法的多值传递

当函数参数仅是一个Object类型时，可以使用元组实现传递多个参数值。

SQL Server(8)

随笔档案 (18)

2017年7月(1)
2017年6月(8)
2017年5月(9)

最新评论

1. Re:详解C# Tuple VS ValueTuple（元组类 VS 值元组）
好评👍
--大猫。
2. Re:详解C# Tuple VS ValueTuple（元组类 VS 值元组）
指定类型的数组吗？
--金色海洋（jyk）阳光男孩
3. Re:提高PowerShell脚本效率的五个常用方法
第四个 第二个方法没有复制值,效率肯定不一样， 个人看法
--BillYang
4. Re:Visual Studio Debugger中七个鲜为人知的小功能
调试水平又提高了, 不过好像都用不上,不过5 还是会用到的
--BillYang
5. Re:从零开始——PowerShell应用入门（全例子入门讲解）
你好，我想用power shell 对 IIS进行启动停止 刷新 等操作，该使用什么命令呢。
--绿林小溪

阅读排行榜

1. 从零开始——PowerShell应用入门（全例子入门讲解）(115496)
2. 详解C# Tuple VS ValueTuple（元组类 VS 值元组）(34842)
3. SQL Server on Ubuntu——Ubuntu上的SQL Server（全截图）(12242)
4. PowerShell管道入门，看看你都会不（管道例子大全）(9816)
5. 一键帮你复制多个文件到多个机器——PowerShell小脚本（内附PS远程执行命令问题解析）(6000)

评论排行榜

1. 详解C# Tuple VS ValueTuple（元组类 VS 值元组）(11)
2. 从零开始——PowerShell应用入门（全例子入门讲解）(8)
3. DB太大？一键帮你收缩所有DB文件大小（Shrink Files for All Databases in SQL Server）(8)
4. SQL Server on Ubuntu——Ubuntu上的SQL Server（全截图）(6)
5. Visual Studio Debugger中七个鲜为人知的小功能(4)

推荐排行榜

1. 详解C# Tuple VS ValueTuple（元组类 VS 值元组）(32)
2. 从零开始——PowerShell应用入门（全例子入门讲解）(11)
3. SQL Server on Ubuntu——Ubuntu上的SQL Server（全截图）(7)
4. Visual Studio Debugger中七个鲜为人知的小功能(7)
5. 微软SQL Server认证最新信息（17年5月22日更新），感兴趣的进来看看哟(4)

```
static void WriteStudentInfo(Object student)
{
    var studentInfo = student as Tuple<string, int, uint>;
    Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age
[{studentInfo.Item2}], Height [{studentInfo.Item3}]");
}

static void RunTest()
{
    var t = new System.Threading.Thread(new
System.Threading.ParameterizedThreadStart(WriteStudentInfo));
    t.Start(new Tuple<string, int, uint>("Bob", 28, 175));
    while (t.IsAlive)
    {
        System.Threading.Thread.Sleep(50);
    }
}
```

尽管元组有上述方便使用的方法，但是它也有明显的不足：

- 访问元素的时候只能通过ItemX去访问，使用前需要明确元素顺序，属性名字没有实际意义，不方便记忆；
- 最多有八个元素，要想更多只能通过最后一个元素进行嵌套扩展；
- Tuple是一个引用类型，不像其它的简单类型一样是值类型，它在堆上分配空间，在CPU密集操作时可能有太多的创建和分配工作。

因此在C# 7.0中引入了一个新的ValueTuple类型，详见下面章节。

ValueTuple详解

ValueTuple是C# 7.0的新特性之一，.Net Framework 4.7以上版本可用。

值元组也是一种数据结构，用于表示特定数量和元素序列，但是是和元组类不一样的，主要区别如下：

- 值元组是结构，是值类型，不是类，而元组（Tuple）是类，引用类型；
- 值元组元素是可变的，不是只读的，也就是说可以改变值元组中的元素值；
- 值元组的数据成员是字段不是属性。

值元组的具体使用如下：

1. 如何创建值元组

和元组类一样，.Net Framework值元组也只支持1到7个元组元素，如果有8个元素或者更多，需要使用值元组的嵌套和Rest属性去实现。另外ValueTuple类可以提供创造值元组对象的静态方法。

- 利用构造函数创建元组：

```
var testTuple6 = new ValueTuple<int, int, int, int, int, int>(1, 2, 3, 4, 5, 6);
Console.WriteLine($"Item 1: {testTuple6.Item1}, Item 6: {testTuple6.Item6}");

var testTuple10 = new ValueTuple<int, int, int, int, int, int, int, ValueTuple<int, int,
int>>(1, 2, 3, 4, 5, 6, 7, new ValueTuple <int, int, int>(8, 9, 10));
Console.WriteLine($"Item 1: {testTuple10.Item1}, Item 10: {testTuple10.Rest.Item3}");
```

- 利用Tuple静态方法构建元组，最多支持八个元素：

```
var testTuple6 = ValueTuple.Create<int, int, int, int, int, int>(1, 2, 3, 4, 5, 6);
Console.WriteLine($"Item 1: {testTuple6.Item1}, Item 6: {testTuple6.Item6}");

var testTuple8 = ValueTuple.Create<int, int, int, int, int, int, int, int>(1, 2, 3, 4, 5, 6,
7, 8);
Console.WriteLine($"Item 1: {testTuple8.Item1}, Item 8: {testTuple8.Rest.Item1}");
```

注意这里构建出来的Tuple类型其实是Tuple<int, int, int, int, int, int, int, Tuple<int>>，因此testTuple8.Rest取到的数据类型是Tuple<int>，因此要想获取准确值需要取Item1属性。

优化区别：当构造出超过7个元素以上的值元组后，可以使用接下来的ItemX进行访问嵌套元组中的值，对于上面的例子，要访问第十个元素，既可以通过testTuple10.Rest.Item3访问，也可以通过testTuple10.Item10来访问。

```
var testTuple10 = new ValueTuple<int, int, int, int, int, int, int, ValueTuple<int, int,
int>>(1, 2, 3, 4, 5, 6, 7, new ValueTuple<int, int, int>(8, 9, 10));
Console.WriteLine($"Item 10: {testTuple10.Rest.Item3}, Item 10: {testTuple10.Item10}");
```

2. 表示一组数据

如下创建一个值元组表示一个学生的三个信息：名字、年龄和身高，而不用单独额外创建一个类。

```
var studentInfo = ValueTuple.Create<string, int, uint>("Bob", 28, 175);
Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age
[{studentInfo.Item2}], Height [{studentInfo.Item3}]");
```

3. 从方法返回多个值

值元组也可以在函数定义中代替out参数返回多个值。

```
static ValueTuple<string, int, uint> GetStudentInfo(string name)
{
    return new ValueTuple <string, int, uint>("Bob", 28, 175);
}

static void RunTest()
{
    var studentInfo = GetStudentInfo("Bob");
    Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age
[{studentInfo.Item2}], Height [{studentInfo.Item3}]");
}
```

优化区别：返回值可以不明显指定ValueTuple，使用新语法(,,)代替，如(string, int, uint)：

```
static (string, int, uint) GetStudentInfo1(string name)
{
    return ("Bob", 28, 175);
}

static void RunTest1()
{
    var studentInfo = GetStudentInfo1("Bob");
    Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age
[{studentInfo.Item2}], Height [{studentInfo.Item3}]");
}
```

调试查看studentInfo的类型就是ValueType三元组。

优化区别：返回值可以指定元素名字，方便理解记忆赋值和访问：

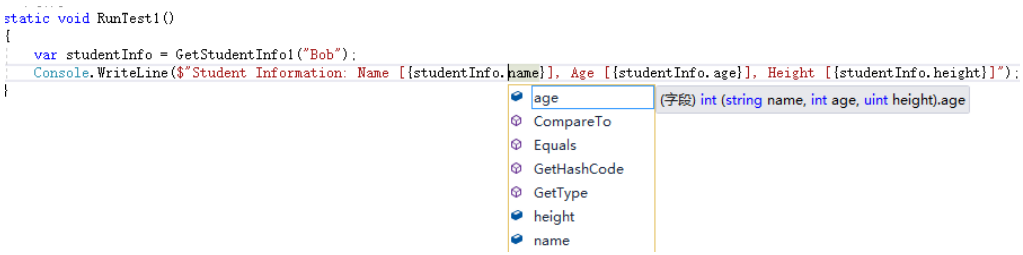
```
static (string name, int age, uint height) GetStudentInfo1(string name)
{
    return ("Bob", 28, 175);
}

static void RunTest1()
{
    var studentInfo = GetStudentInfo1("Bob");
    Console.WriteLine($"Student Information: Name [{studentInfo.name}], Age
[{studentInfo.age}], Height [{studentInfo.height}]");
}
```

方便记忆赋值：

```
static (string name, int age, uint height) GetStudentInfo1(string name)
{
    return ("Bob", 28, 175);
}
1 个引用
```

(string name, **int age**, uint height)

方便访问：



4. 用于单参数方法的多值传递

当函数参数仅是一个Object类型时，可以使用值元组实现传递多个值。

```
static void WriteStudentInfo(Object student)
{
    var studentInfo = (ValueTuple<string, int, uint>)student;
    Console.WriteLine($"Student Information: Name [{studentInfo.Item1}], Age
[{studentInfo.Item2}], Height [{studentInfo.Item3}]");
}

static void RunTest()
{
    var t = new System.Threading.Thread(new
System.Threading.ParameterizedThreadStart(WriteStudentInfo));
    t.Start(new ValueTuple<string, int, uint>("Bob", 28, 175));
    while (t.IsAlive)
    {
        System.Threading.Thread.Sleep(50);
    }
}
```

5. 解构ValueTuple

可以通过var (x, y)或者(var x, var y)来解析值元组元素构造局部变量，同时可以使用符号”_”来忽略不需要的元素。

```
static (string name, int age, uint height) GetStudentInfo1(string name)
{
    return ("Bob", 28, 175);
}

static void RunTest1()
{
    var (name, age, height) = GetStudentInfo1("Bob");
    Console.WriteLine($"Student Information: Name [{name}], Age [{age}], Height
[{height}]");

    (var name1, var age1, var height1) = GetStudentInfo1("Bob");
    Console.WriteLine($"Student Information: Name [{name1}], Age [{age1}], Height
[{height1}]");

    var (_, age2, _) = GetStudentInfo1("Bob");
    Console.WriteLine($"Student Information: Age [{age2}]");
}
```

由上所述，ValueTuple使C#变得更简单易用。较Tuple相比主要好处如下：

- ValueTuple支持函数返回值新语法”(,,)”，使代码更简单；
- 能够给元素命名，方便使用和记忆，这里需要注意虽然命名了，但是实际上value tuple没有定义这样名字的属性或者字段，真正的名字仍然是ItemX，所有的元素名字都只是设计和编译时用的，不是运行时用的（因此注意对该类型的序列化和反序列化操作）；
- 可以使用解构方法更方便地使用部分或全部元组的元素；
- 值元组是值类型，使用起来比引用类型的元组效率高，并且值元组是有比较方法的，可以用于比较是否相等，详见：https://msdn.microsoft.com/en-us/library/system.valuetuple。

[原创文章，转载请注明出处，仅供学习研究之用，如有错误请留言，如觉得不错请推荐，谢谢支持]