

第08期-深入理解元组类型

2022年4月13日 9:52

元组的概念

提供了简洁的语法，用于将多个数据元素分组成一个轻型数据结构，元组是值类型。

声明元组

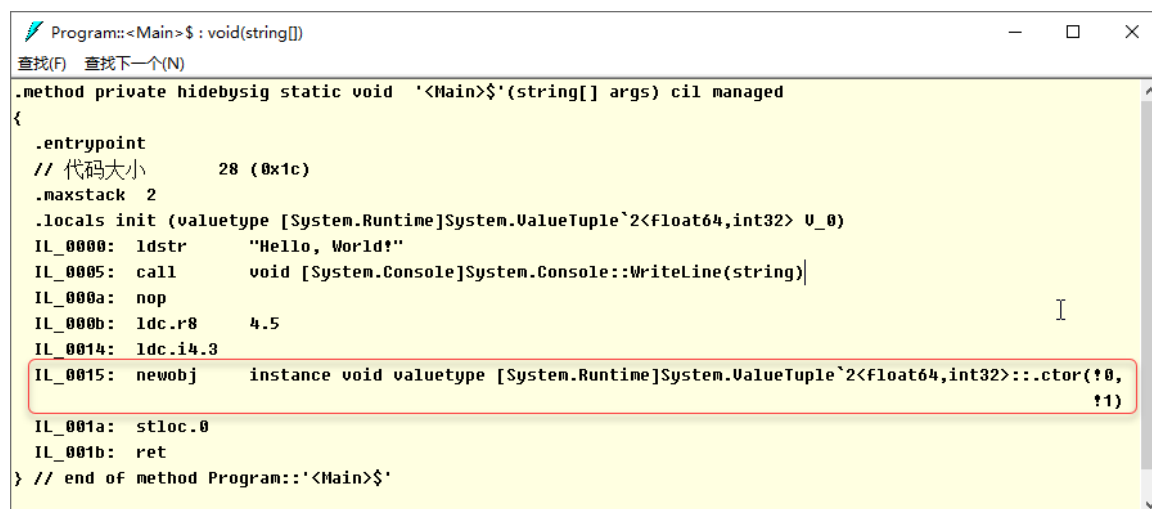
```
(double Sum, int Count) t1 = (4.5, 3);
```

```
(double, int) t2 = (4.5, 3);
```

```
var t3 = (Sum: 4.5, Count: 3);
```

```
var sum = 4.5;  
var count = 3;  
var t4 = (sum, count);
```

编译结果

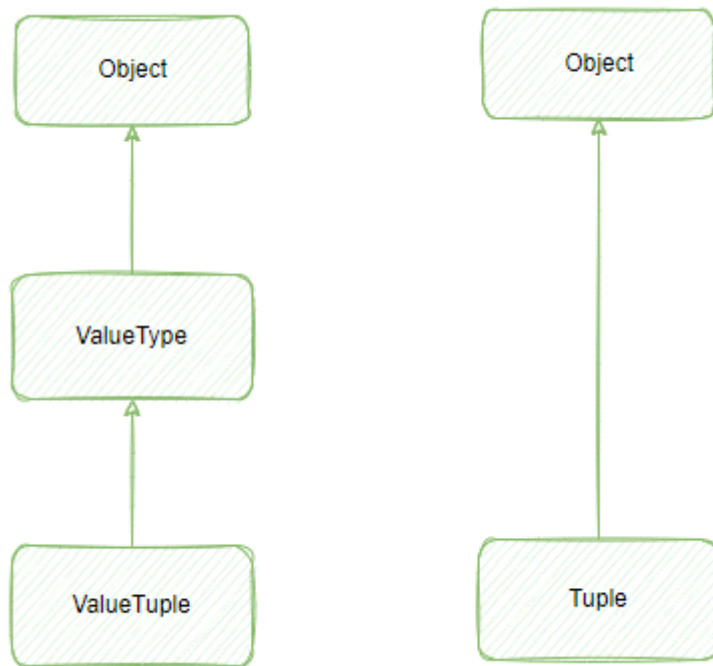


```
Program::<Main>$ : void(string[])  
查找(F) 查找下一个(N)  
.method private hidebysig static void '<Main>$'(string[] args) cil managed  
{  
    .entrypoint  
    // 代码大小      28 (0x1c)  
    .maxstack 2  
    .locals init (valuetype [System.Runtime]System.ValueTuple`2<float64,int32> U_0)  
    IL_0000: ldstr      "Hello, World!"  
    IL_0005: call       void [System.Console]System.Console::WriteLine(string)|  
    IL_000a: nop  
    IL_000b: ldc.r8      4.5  
    IL_0014: ldc.i4.3  
    IL_0015: newobj     instance void valuetype [System.Runtime]System.ValueTuple`2<float64,int32>::.ctor(!0,  
    !1)  
    IL_001a: stloc.0  
    IL_001b: ret  
} // end of method Program::'<Main>$'
```

元组的支持

元组功能需要 [System.ValueTuple](#) 类型和相关泛型类型的支持，不同于 [System.Tuple](#) 类型表示的元组：

- System.ValueTuple 类型是值类型。 System.Tuple 类型是引用类型。
- System.ValueTuple 类型是可变的。 System.Tuple 类型是不可变的。
- System.ValueTuple 类型的数据成员是字段。 System.Tuple 类型的数据成员是属性。



TupleExtensions 扩展类提供转换：Deconstruct、ToTuple 和 ToValueTuple

元组作为方法返回值

```

(int min, int max) FindMinMax(int[] input)
{
    if (input is null || input.Length == 0)
    {
        throw new ArgumentException("null or empty array.");
    }

    var min = int.MaxValue;
    var max = int.MinValue;

    foreach (var i in input)
    {
        if (i < min)
        {
            min = i;
        }
        if (i > max)
        {
            max = i;
        }
    }

    return (min, max);
}
  
```

元组赋值和析构

元组赋值和元组相等比较不会考虑字段名称，元组之间的赋值需要满足的条件：**数量相同**和**元素类型匹配**。元组的赋值是按照元素顺序**依次**赋值的。

```
(int, double) t1 = (17, 3.14);
(double First, double Second) t2 = (0.0, 1.0);
t2 = t1;
Console.WriteLine($"{nameof(t2)}: {t2.First} and {t2.Second}");
// Output:
// t2: 17 and 3.14

(double A, double B) t3 = (2.0, 3.0);
t3 = t2;
Console.WriteLine($"{nameof(t3)}: {t3.A} and {t3.B}");
// Output:
// t3: 17 and 3.14
```

析构元组和其它类型

```
int[] arr = new int[] { 1, 2, 3 };

var (m1, m2) = test.FindMinMax(arr);

(int mi, var ma) = test.FindMinMax(arr);

int minValue=0;
(minValue,int maxValue)= test.FindMinMax(arr);

(minValue, _) = test.FindMinMax(arr);
```

自己的类型支持元组析构

```
public void Deconstruct(out string fname, out string mname, out string lname)

var (fName, mName, lName) = p;
```

Deconstruct 可重载，也可以定义为扩展方法。

某些系统类型支持

`KeyValuePair<TKey,TValue>.Deconstruct(TKey, TValue)`

记录类型默认提供析构方法

```
public abstract record Person(string FirstName, string LastName);

public record Teacher(string FirstName, string LastName, int Grade)
    : Person(FirstName, LastName);

public record Student(string FirstName, string LastName, int Grade)
    : Person(FirstName, LastName);

public static void Main()
{
    Person teacher = new Teacher("Nancy", "Davolio", 3);
    var (firstName, lastName) = teacher; // Doesn't deconstruct Grade
    Console.WriteLine($"{firstName}, {lastName}");// output: Nancy, Davolio

    var (fName, lName, grade) = (Teacher)teacher;
    Console.WriteLine($"{fName}, {lName}, {grade}");// output: Nancy, Davolio, 3
}
```