
07.04.23.

北島



CA 固解 第十章

語句

10.9 switch

```
static void Main()
{
    var shapes = new List<Shape>();
    shapes.Add(new Circle() { Radius = 7 });
    shapes.Add(new Square() { Side = 5 });
    shapes.Add(new Triangle() { Height = 4 });
    var nullSquare = (Square)null;
    shapes.Add(nullSquare);

    foreach (var shape in shapes)
    {
        switch(shape) //判断类型或者 shape 变量的值
        {
            case Circle circle: //等价于 if(shape is Circle)
                Console.WriteLine($"This shape is a circle of radius { circle.Radius }");
                break;
            case Square square when square.Side > 10: //依赖局部一部分 Square
                Console.WriteLine($"This shape is a large square of side { square.Side }");
                break;
            case Square square:
                Console.WriteLine($"This shape is a square of side { square.Side }");
                break;
            case Triangle triangle: //等价于 if(shape is Triangle)
                Console.WriteLine($"This shape is a triangle of side { triangle.Height }");
                break;
            //case Triangle triangle when triangle.Height < 5: //编译错误
            //break;
            case null:
                Console.WriteLine($"This shape could be a Square, Circle or a Triangle");
                break;
            default:
                throw new ArgumentException(
                    message: "shape is not a recognized shape",
                    paramName: nameof(shape));
        }
    }
}
```

范围大小
问题

case 语句操作多重类型模式

10.13 标签

Identifier Statement 只能在块内 jmp

例如，下面的代码展示了标签的有效使用，该标签和一个局部变量

```
{
    int xyz = 0; // 变量 xyz
    ...
    xyz: Console.WriteLine("No problem."); // 标签 xyz
}
```

gots Identifier 多跟着，like C#
switch gots const

10.15 using语句

用于即时资源释放

Resource 三种资源声明

Way1.

❑ using语句隐式产生处置该资源的代码。

`using (ResourceType Identifier = Expression) Statement`

↑
分配资源

↑
使用资源

❑ 创建资源的 Dispose 方法的调用，并把它放进 finally 块。

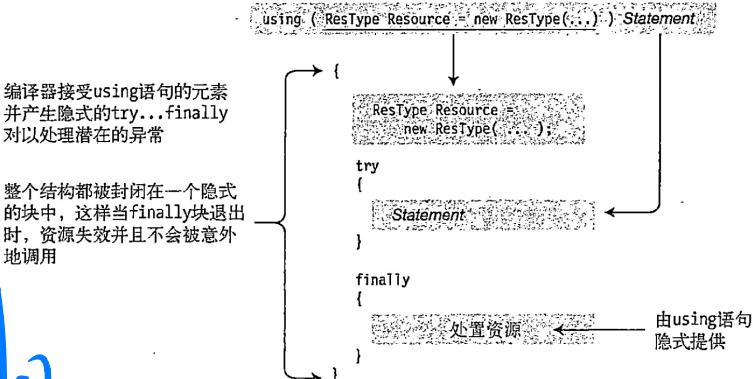


图 10-11 using 语句的效果

Using 可能会 被重用

Way 2.

using 语句的另一种形式如下：

关键字 资源 使用资源
↓ ↓ ↓
using (Expression) EmbeddedStatement

在这种形式中，资源在 using 语句之前声明。

```
TextWriter tw = File.CreateText("Lincoln.txt");                          // 声明资源  
using ( tw )                                                                    // using 语句  
    tw.WriteLine("Four score and seven years ago, ...");
```

不能防 using
释放了它的非托管资源
之后再使用如上文
的 tw?

CH 圖解 第十一章

結構

diff between 类与结构

类：引用

结构：值，不可派生，声明时不实现属性与字段 initialize
(赋值)

11.4 构造 func and 析构 func
↓ 可有默认构造 不可有

↓ 析构

自声明
必须有
arg v

```
{  
    static void Main()  
    {  
        // 调用隐式构造函数  
        Simple s1 = new Simple();  
        Simple s2 = new Simple(s1, 10);  
  
        // 调用构造函数  
        Console.WriteLine($"{ s1.X },{ s1.Y }");  
        Console.WriteLine($"{ s2.X },{ s2.Y }");  
    }  
}
```

也可以不使用 new 运算符创建结构的实例。然而，如果这样做，有一些限制，如下：
□ 在正式设置数据成员之后，才能使用它们的值。
□ 在对所有数据成员赋值之后，才能调用结构的函数成员。

← new struct 的重要
这里似乎与 C++ 的不同

11.5 密封性(一些修饰符不可用)

结构总是显式密封的，因此，不能从它们派生其他结构。

由于结构不支持继承，个别类成员修饰符用在结构成员上将没有意义，因此不能在结构成员声明中使用。不能用于结构的修饰符如下：

- protected ✓
- protected internal ✓
- abstract ✓
- sealed ?
- virtual ✓

结构本身派生自 System.ValueType，而 System.ValueType 派生自 object。

两个可以用于结构成员并与继承相关的关键字是 new 和 override 修饰符，当创建一个和基类 System.ValueType 的成员同名的成员时可使用它们。所有结构都派生自 System.ValueType。

11.9 tips.

简单类其实在.NET中为structure

C# 因解 第二章

板書

define:

```
关键字 枚举名称
↓ ↓
enum Trafficlight
{
    Green,   ←逗号分隔，没有分号
    Yellow,  ←逗号分隔，没有分号
    Red
}
```

→ 这仨都是整数常量 0, 1, 2(默认)

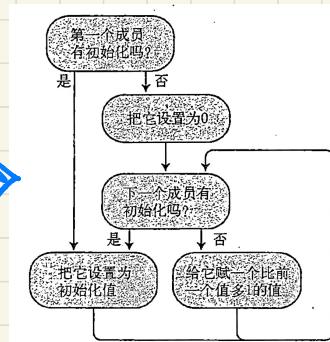


图 12-3 成员赋值的法则

将枚举值赋值给 enum 变量，打印出来为成员名

12.1.1 设置底层类型与赋值

example: enum ex: ulong

```
{ e1=100, } { 独立
e2=100, } { 独立
e3=102,
```

12.2 位标志

在本例中，特性出现在枚举声明之前。特性

```
[Flags]  
enum CardDeckSettings : uint  
{  
    SingleDeck = 0x01, //位 0  
    LargePictures = 0x02, //位 1  
    FancyNumbers = 0x04, //位 2  
    Animation = 0x08 //位 3  
}
```

图 12-4 阐明了这个枚举。



→ CardDeckSettings cds = ... | ... &
之前沙盒黑白名单
cds.HasFlag(位标志) return bool
enum方法

12.2.1 Flags特性

Flags 特性不会改变计算结果，却提供了一些方便的特性。首先，它通知编译器、对象浏览器以及其他查看这段代码的工具，该枚举的成员不仅可以作用单独的值，还可以组合成位标志。这样浏览器就可以更恰当地解释该枚举类型的变量。

其次，它允许枚举的 ToString 方法为位标志的值提供更多的格式化信息。ToString 方法以一个枚举值为参数，将其与枚举的常量成员相比较。如果与某个成员相匹配，ToString 返回该成员的字符串名称。

```
class Program {  
    static void Main() {  
        CardDeckSettings ops;  
        ops = CardDeckSettings.FancyNumbers; //设置一个标志  
        Console.WriteLine(ops.ToString());  
        ops = CardDeckSettings.FancyNumbers | CardDeckSettings.Animation; //设置两个标志  
        Console.WriteLine(ops.ToString()); //输出什么呢?  
    }  
}
```

无Flags为纯数字
有Flags为位标志或负数

12.3 tips:

成员无修饰符，与枚举访问性一致。

静态，未声明也可访问（const问题？）

不可比较不同enum类型的成员

GetName (enum, int),

GetNames (enum)

ToString () [Flags] 需求

C++ 图解 第十三章

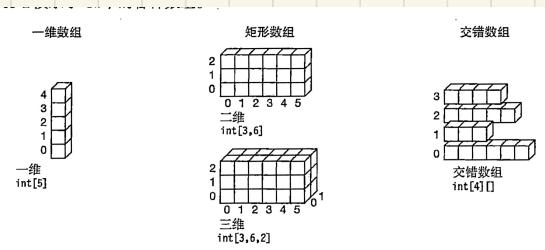
数组

B.1.2 重要细节：

(并不支持 dynamic array, [] 放基类型)

B.2 数组类型：

{ 一维
多维 } 矩型数组 → 子数组相同长
交错数组 → 子数组都是独立数组多维 Arr
→ 可不同长子 arr
→ 二维用一个[]



13.3 数组都是对象
So 也是引用类型，但值引/引用

13.4 /上 声明与实例化

声明时不可声明 Arr 长度吗 int[] A int[,] C
int[,] B

实例化：int[,] C = new int[3, 4, 5]

13.7 初始化

```
int[] intArr = new int[4];
```

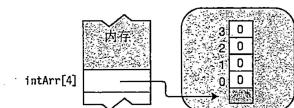


图 13-6 一维数组的自动初始化

↑ 为什十的仰角。

初始化列表
↓
int[] intArr = new int[] { 10, 20, 30, 40 };
↓
没有连接运算符



图 13-7 一维数组的显式初始化

初始化列表由逗号分隔

```
int[,] intArray2 = new int[,] { {10, 1}, {2, 10}, {11, 9} };
```



图 13-8 初始化矩形数组

intArray2 有 3 个两个元素为一组的分组

快捷：可只提供初始化部分，如 `int[,] intArray = {{1,2,3}}`

Q 二三维数组

三维数组

new

隐式：var 开头，要保留创建表达式，且未说明将保留
仅省略了明确的的数据类型

```

    显式          显式
    ↓             ↓
int [] intArr1 = new int[] { 10, 20, 30, 40 };
var   intArr2 = new   [] { 10, 20, 30, 40 };
    ↑           ↑

关键字      推断
    ↓
int[,] intArr3 = new int[,] { { 10, 1 }, { 2, 10 }, { 11, 9 } };
var   intArr4 = new   [,] { { 10, 1 }, { 2, 10 }, { 11, 9 } };

    ↓
    秩说明符
string[] sArr1 = new string[] { "life", "liberty", "pursuit of happiness" };
var   sArr2 = new   [] { "life", "liberty", "pursuit of happiness" };

```

1276 総合内容

13.8 交错数组

不可在声明中初始化顶层
层之外的数组

The image shows two handwritten Chinese characters in blue ink on lined paper. The character '北' (Bei) is at the top left, and '天津' (Tianjin) is at the top right. Below each character is a stroke order diagram. The '北' diagram shows a vertical stroke on the left, followed by four horizontal strokes forming a rectangle. The '天津' diagram shows a vertical stroke on the left, followed by a horizontal stroke, and then a vertical stroke on the right. Arrows indicate the direction of each stroke.

- 第一个维度的长度是 3。
 - 声明可以读作“`jagArr` 是 3 个 `int` 数组的数组”。
 - 注意，图中有 4 个数组对象，其中一个针对顶层数组，另外 3 个针对子数组。

```
int[][] jagArr = new int[3][]; //声明并创建顶层数组  
...  
//声明并创建子数组
```

↑
顶层内存管理

↓
子数组, 长度可不同

13.8.3 实例化：

- 先实例化顶层数组
- 实例化每个子数组

```
int[,] Arr;           // 带有二维数组的交错数组
Arr = new int[3][];
Arr[0] = new int[,] { { 10, 20 },
                     { 100, 200 } };
Arr[1] = new int[,] { { 30, 40, 50 },
                     { 300, 400, 500 } };
Arr[2] = new int[,] { { 60, 70, 80, 90 },
                     { 600, 700, 800, 900 } };

for (int i = 0; i < Arr.GetLength(0); i++)
{
    ↓ 获取 Arr 维度 0 的长度
    for (int j = 0; j < Arr[i].GetLength(0); j++)
        ↓ 获取 Arr[i] 维度 0 的长度
        for (int k = 0; k < Arr[i,j].GetLength(1); k++)
            ↓ 获取 Arr[i,j] 维度 1 的长度
            Console.WriteLine(
                $"[{i}][{j}][{k}] = {Arr[i,j,k]}");
    Console.WriteLine("");
}
Console.WriteLine("");
```

矩阵+交错

13.10 foreach

■ statement 是遍历数组中的每一个元素：

显式类型迭代变量声明

```
foreach( Type Identifier in Array Name )
    Statement
```

隐式类型迭代变量声明

```
foreach( var Identifier in Array Name )
    Statement
```

迭代变量声明

操作值类型会编译报错？

引用类型可改变引用的内容

多维中是逐个处理

交错中是要给每一维用独立的foreach, 2维就要2套
类推...

13.11 数组协变

在某些情况下，即使某个对象不是数组的基类型，也可以把它赋值给数组元素。这种属性叫作数组协变（array covariance）。在下面的情况下可以使用数组协变。

- 数组是引用类型数组。
- 在赋值的对象类型和数组基类型之间有隐式转换或显式转换。

由于在派生类和基类之间总是有隐式转换，因此总是可以将一个派生类的对象赋值给为基类声明的数组。

派生 ~~基~~ 基
~~派生~~ 引用

13. 数组方法

clone } 强：产生俩独立 Array

{ 弱：产生俩 Array，但指向堆内同数据