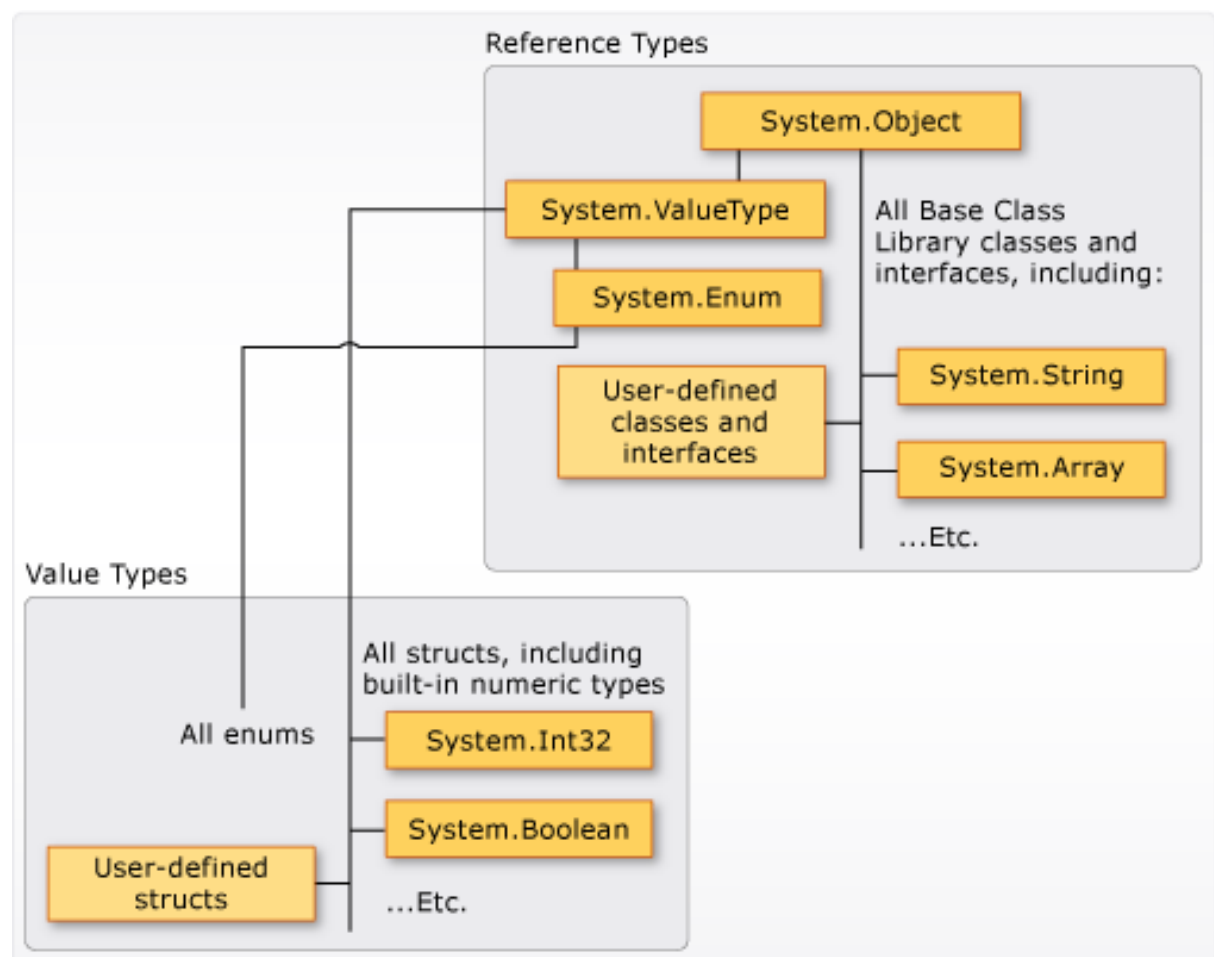


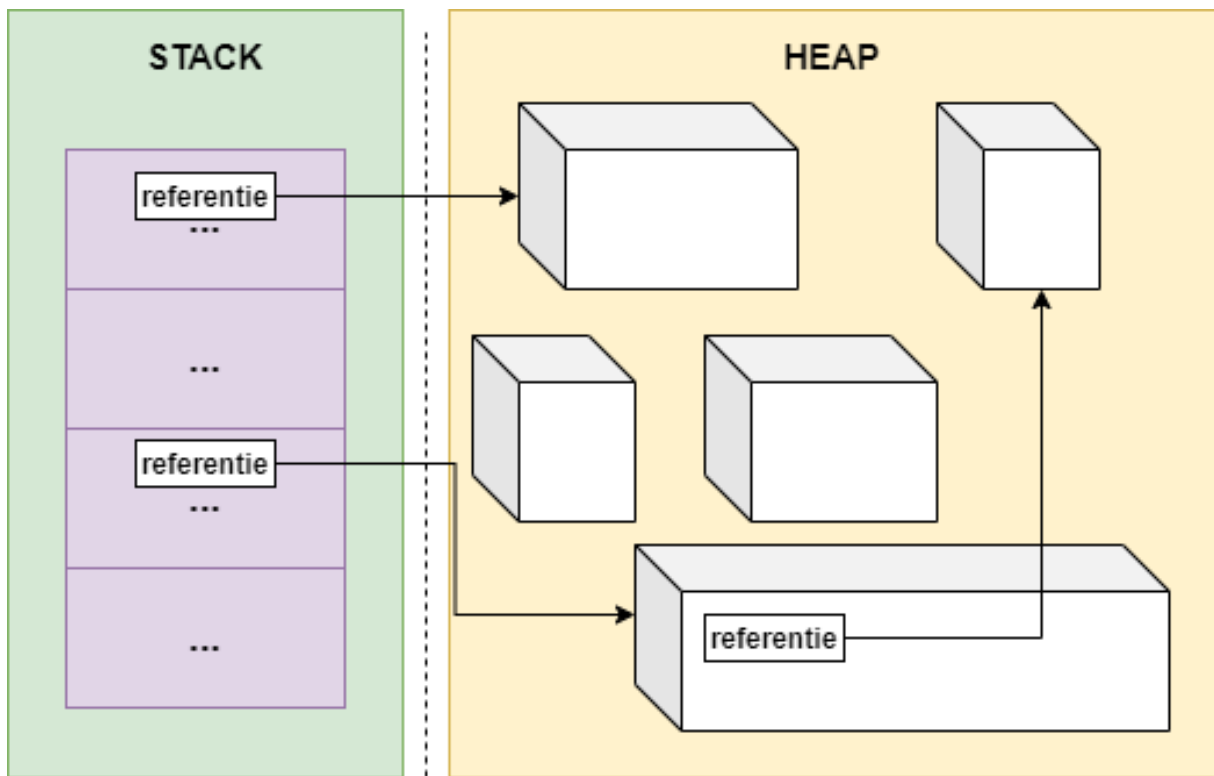
第04期-理解C#中的类型系统

2022年3月14日 14:33

C# 是强类型语言，变量和常量都具有一个类型，方法的输入参数和返回值也是如此，数值类型和表示各种构造的复杂类型，类型可以拥有以下可选项。

- 类型变量所需的存储空间。
- 可以表示的最大值和最小值。
- 包含的成员（方法、字段、事件等）。
- 继承自的基类型。
- 它实现的接口。
- 允许执行的运算种类。





类型系统

值类型和引用类型，值类型的变量包含类型的实例，引用类型的变量包含对类型实例的引用。

在内存分配上，值类型存储在栈上，赋值即拷贝副本，引用类型存储在堆上，也就是说托管堆，赋值指针。

自定义值类型

```
using System;

public struct MutablePoint
{
    public int X;
    public int Y;

    public MutablePoint(int x, int y) => (X, Y) = (x, y);

    public override string ToString() => $"({X}, {Y})";
}

public class Program
{
    public static void Main()
    {
        var p1 = new MutablePoint(1, 2);
        var p2 = p1;
        p2.Y = 200;
        Console.WriteLine($"{nameof(p1)} after {nameof(p2)} is modified: {p1}");
        Console.WriteLine($"{nameof(p2)}: {p2}");
    }
}
```

```

        MutateAndDisplay(p2);
        Console.WriteLine($"{nameof(p2)} after passing to a method: {p2}");
    }

    private static void MutateAndDisplay(MutablePoint p)
    {
        p.X = 100;
        Console.WriteLine($"Point mutated in a method: {p}");
    }
}
// Expected output:
// p1 after p2 is modified: (1, 2)
// p2: (1, 200)
// Point mutated in a method: (100, 200)
// p2 after passing to a method: (1, 200)

```

自定义引用类型

```

using System;
using System.Collections.Generic;

public struct TaggedInteger
{
    public int Number;
    private List<string> tags;

    public TaggedInteger(int n)
    {
        Number = n;
        tags = new List<string>();
    }

    public void AddTag(string tag) => tags.Add(tag);

    public override string ToString() => $"{Number} [{string.Join(", ",
tags)}]";
}

public class Program
{
    public static void Main()
    {
        var n1 = new TaggedInteger(0);
        n1.AddTag("A");
        Console.WriteLine(n1); // output: 0 [A]

        var n2 = n1;
        n2.Number = 7;
        n2.AddTag("B");

        Console.WriteLine(n1); // output: 0 [A, B]
        Console.WriteLine(n2); // output: 7 [A, B]
    }
}

```