C#中的值类型和引用类型：

值类型：基本数据类型，结构类型，枚举类型。

引用类型：类，数组，接口，委托，指针。

值类型隐式派生于System.ValueType，不能被继承。

CLR：Common Language Runtime，公共语言运行库，类似Java中的虚拟机。

值类型表达式的值是数据本身。

引用类型表达式的值是引用。

引用的值是一个数据块，指向引用类型的实例。

引用是一个小的数据块，它包含了一些信息，CLR可以根据这些信息来找到引用所指向的对象。引用本身一般存储在栈上。

引用类型的实例总是分配在托管堆上，小于85000byte，分配在GC堆上，大于85000byte，分配在LOH（Large Object Heap）堆上。

值类型可以存储在栈上，但不一定就是存储在栈上。

例：程序reference\_value\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 值类型存储在堆上

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace reference\_value\_test1

{

delegate int CountIt(int end);

class ReferenceValueTest

{

// 委托作为返回值

static CountIt Counter()

{

int sum = 0;

// 局部变量sum被匿名函数捕获

CountIt obj = delegate(int end)

{

for (int i = 0; i <= end; ++i)

{

Console.WriteLine(i);

sum += i;

}

return sum;

};

return obj;

}

static void Main(string[] args)

{

CountIt count = Counter();

int result = 0;

result = count(3);

Console.WriteLine("Summation of 3 is " + result);

Console.WriteLine();

result = count(5);

Console.WriteLine("Summation of 5 is " + result);

}

}

}

输出结果：

0

1

2

3

Summation of 3 is 6

0

1

2

3

4

5

Summation of 5 is 21

从结果中可以看出，执行两次count()后，第二次的结果为第一次的结果加上自身的运算结果，原因是：在Counter()函数中定义的局部变量sum并未定义在栈上（如果定义在栈上，第一次执行完后，应该释放掉），而是定义在了托管堆上，可见局部变量不一定都是定义在栈上。在Lambda表达式中也会出现同样的状况。

值类型作为引用类型的字段时，同引用类型的实例一起存储在托管堆上。

例：程序reference\_value\_test2

public class NestedValueInRef

{

// a作为引用类型实例的一部分将分配在托管堆上

private int a;

}

值类型嵌套引用类型：值类型如果（注意分配在堆上的情况）不分配在堆上，就会分配在栈上，而其引用类型的字段分配在托管堆上，并被栈上的引用所持有。

public struct ValueTypeStruct

{

private object \_referenceType;

}

// ValueTypeStruct的局部变量obj分配在栈上

// 它的\_referenceType字段对应的实例分配在托管堆上，其引用

// 位于栈上被obj持有

ValueTypeStruct obj = new ValueTypeStruct();

在C#中，默认情况下参数的传递都是按值传递的（这和C++相似），即传递的是参数的副本。注意：无论是值类型的参数，还是引用类型的参数，默认情况下都是按值传递的，对于值类型的参数，复制的是该参数所存储的数据，对于引用类型的参数，复制的是引用本身，引用的副本和原始引用都指向同一个引用类型的实例对象，这一点极其类似C++中的指针。

例：程序reference\_value\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 引用类型和值类型

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace reference\_value\_test3

{

class ReferenceValueTest

{

static void N(int j)

{

j = 10;

}

static void M(StringBuilder sb)

{

sb = null;

}

static void P(StringBuilder sb)

{

sb.Append(", World");

}

static void Main(string[] args)

{

int i = 5;

N(i);

// 按值传递，函数N内部的修改不会影响i的值

Console.WriteLine(i); // i = 5

StringBuilder sb = new StringBuilder("Hello");

M(sb);

// 按值传递，函数M内部修改sb本身不会影响到sb

Console.WriteLine(sb); // Hello

P(sb);

// 在函数P中，修改了sb所指向的实例对象

Console.WriteLine(sb); // Hello, World

}

}

}

按引用传递，可以参考：C#学习13-值类型的传递。