作业：

```
            Dictionary<int, string> dic = new Dictionary<int, string>
();
            string x = "1=壹|2=贰|3=叁|4=肆|5=伍|6=陆|7=柒|8=捌|9=玖";
            string[] datas = x.Split('|');

            for (int i = 0; i < datas.Length; i++)
            {
                string[] str = datas[i].Split('=');
                dic.Add(int.Parse(str[0]), str[1]);
            }

            int w = int.Parse(Console.ReadLine());

            if (dic.ContainsKey(w))
            {
                Console.WriteLine(dic[w]);
            }
```

C#的值类型和引用类型

值类型：

1 基本数据类型 int float long double bool

2 枚举类型 enum

3 结构类型 struct

引用类型：

1 类 基类：System.Object string class（自定义）

2 接口 interface

3 数组 int[] char[]等 Array

不管是值类型还是引用类型 都继承于System.Object根父类

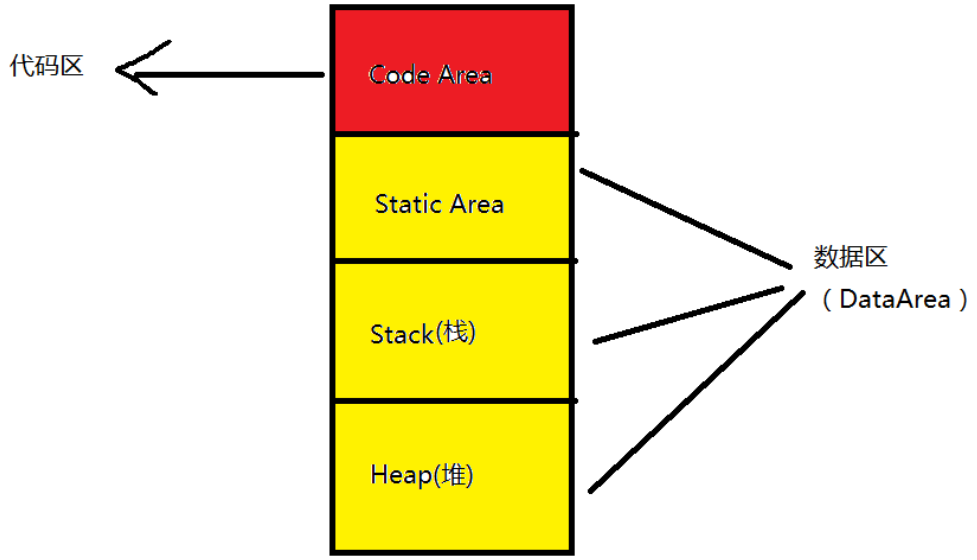
值类型的赋值是变量值的复制

引用类型的赋值是赋值对象的引用

内存知识概括

内存主要分为两个大区域： 数据区 代码区

数据区细分：静态区域 堆区 栈区



堆和栈 (Stack Heap)

在C/C++中

Stack就叫栈区 由编译器自动分配内存和释放 存放函数的参数值 局部变量的值等

Heap叫做堆区 由程序员分配释放 若程序员不释放 程序结束时OS回收

在C#中

Stack叫堆栈（简称栈） Heap叫托管堆（简称堆）

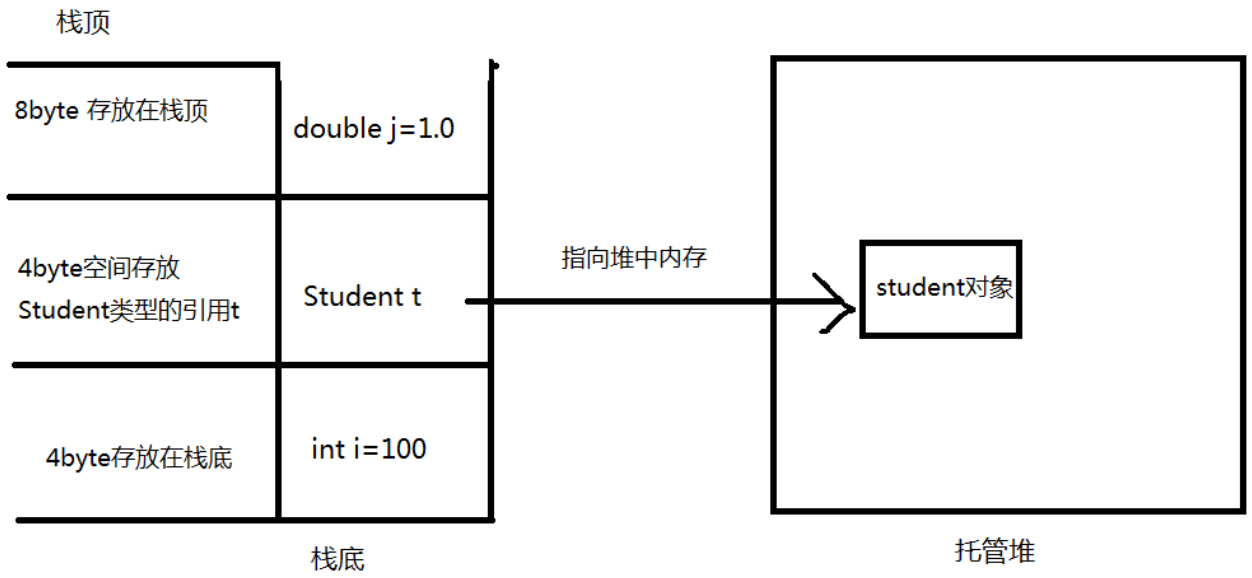
作为.NET开发无需手动释放内存或对内存进行管理

CLR(公共语言运行库)会对托管堆进行自动的管理和垃圾清理（GC）

栈区：存放 方法的参数 局部变量 返回值等数据 由编译器自动释放

堆区：存放引用类型的对象 由CLR内存管理机制释放

栈的释放流程：先进后出



栈的空间很小 一般的编译器2M-1M左右 栈需要频繁的释放和开辟

栈的释放顺序：先进后出 后进先出

第一个存入栈中的数据 存放在栈底 最后存储的数据存放在栈顶（入栈）

在释放数据时 由栈顶开始释放 所以最后存入栈的数据第一个被释放 （出栈）

使用栈区需要注意的问题：

一般来说 一个变量出了作用域后就会被从栈中释放空间删除数据

比如递归算法 如果出现死递归的情况 就会很容易出现（StackOverflow）栈溢出的情况

原因：

递归不断调用自身 所有的临时变量都没有从栈中释放过

当栈的空间不足时 就会出现栈溢出

```
static int Fn(int x)
{
    Fn(x);
    return 100;
}
```

运行结果：StackOverflow

但是因为不用存储对应的引用关系 所以栈的读写速度很快 效率更高

而堆因为在使用时需要进行查找等其他操作 所以运行效率比栈要低

但是堆因为有足够的内存空间 所以占用内存比较大的对象更适合存放在堆中

效率问题：

因为栈是有序排列 且 数据量少 所以读写很快

而堆的数据时杂乱无章的 读写堆中数据时需要查找引用关系 所以读写速度较慢

正因为.NET由自动内存管理机制和垃圾回收机制

不能像C++一样自由操作内存

所以C#的开发简单 但是效率不高

内存优化1：泛型减少拆装箱带来的内存操作

在大多数情况下 不要使用object类型来进行参数的传递 转而使用泛型

字符串的优化：

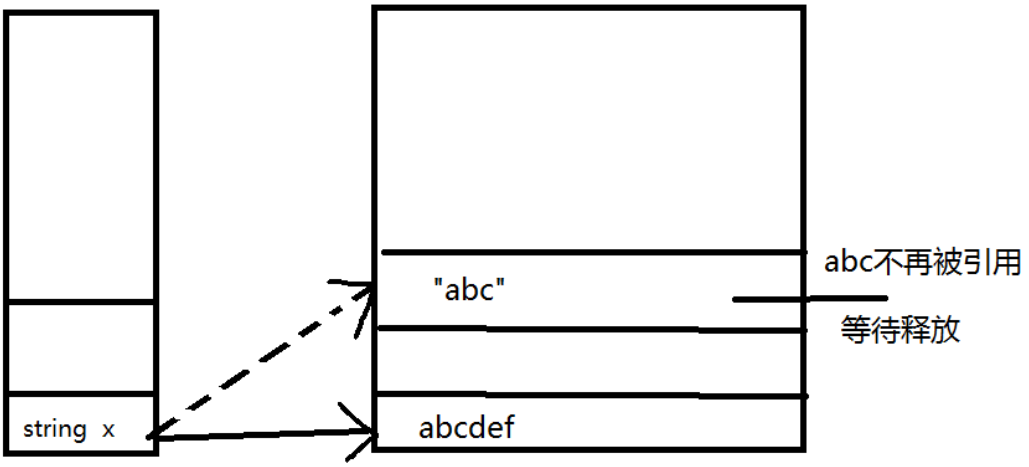
字符串的特性

字符串是个引用类型 所以存储在堆中

字符串是不可变的

当创建了一个字符串对象 他的长度和内容都是固定不变的

```
string x = "abc";
x += "def";
Console.WriteLine(x);
```



1 原先的x指向堆内存中的abc字符串

2 当使用+运算后 堆中内存会创建一个新对象“abcdef” x指向新字符串对象

3 原先的abc字符串 没有变量引用了 会等待垃圾回收

StringBuilder

和string一样 StringBuilder的内部也是一个char[]

功能比较简单 主要就是增删改

在创建SB时 可以指定字符串的容量

对SB进行操作时 如果超过了指定的容量 会创建新数组

数组大小翻倍 旧数据等待回收

```
StringBuilder sb = new StringBuilder(10);

            StringBuilder sb1 = new StringBuilder("HelloWorld");
            sb1.Capacity = 20;//指定容量

            StringBuilder sb2 = new StringBuilder("HelloWorld", 20);


            Console.WriteLine(sb.Capacity);//10
            sb.Append(888);
            sb.Append('w');
            sb.Append('w');
            sb.AppendFormat("{0}{1}", 5, 7);

```

```
            Console.WriteLine(sb.Length);

            Console.WriteLine(sb);

            sb.Append(888);
            sb.Append('x');
            sb.Append('y');
            Console.WriteLine(sb.Capacity);

            for (int i = 0; i < sb.Length; i++)
            {
                Console.WriteLine(sb[i]);
            }
```

练习：

定义一个玩家数据类 PlayerData

类中包含玩家信息：

ID Username Password level

在默认的构造方法中给成员赋值

创建玩家数据类的对象 把所有的数据存储在一个字符串中

"ID=100|username=abc123|password=777888|level=20"

```
    public class PlayerData
    {
        public int id;
        public string username;
        public string password;
        public int level;

        public PlayerData()
        {
            id = 100;
            username = "game";
            password = "123999";
            level = 20;
        }
    }
```

```
    static void Main(string[] args)
    {
            PlayerData pd = new PlayerData();
            StringBuilder sb = new StringBuilder(25);

            string format = "{0}={1}";
            sb.AppendFormat(format, "id", pd.id);
            sb.Append("|");
            sb.AppendFormat(format, "password", pd.password);
            sb.Append("|");
            sb.AppendFormat(format, "username", pd.username);
            sb.Append("|");
            sb.AppendFormat(format, "level", pd.level);
            Console.WriteLine(sb);
     }
```