不使用系统的方法 比较两个字符串的大小

```
        static int Fun(string s1,string s2)
        {
            if (!string.IsNullOrEmpty(s1) && string.IsNullOrEmpty(s2))
                return 1;

            if (string.IsNullOrEmpty(s1) && !string.IsNullOrEmpty(s2))
                return -1;

            if (string.IsNullOrEmpty(s1) && string.IsNullOrEmpty(s2))
                return 0;

            int length = s1.Length > s2.Length ? s1.Length :
s2.Length;

            for (int i = 0; i < length; i++)
            {
                if (i > s1.Length - 1)
                    return -1;

                if (i > s2.Length - 1)
                    return 1;

                if (s1[i] > s2[i])
                    return 1;

                if (s2[i] > s1[i])
                    return -1;
            }

            return 0;
        }
```

"1，屠龙刀，这是一把至尊宝刀|2,倚天剑,这是一把剑|3,降龙十八掌,这是一本武功秘籍"

```
public class Item
 {
     private int id;
     public int ID
     {
         get { return id; }
         set { id = value; }
     }

     private string name;
     public string Name
     {
         get { return name; }
         set { name = value; }
     }

     private string explain;
     public string Explain
     {
         get { return explain; }
         set { explain = value; }
     }


     public Item(string data)
     {
         string[] datas = data.Split(',');
         this.id = int.Parse(datas[0]);
         this.name = datas[1];
         this.explain = datas[2];
     }

 }

 class Program
 {
     static void Main(string[] args)
     {
         string str = "1,屠龙刀,这是一把至尊宝刀|2,倚天剑,这是一把剑|3,降龙十八掌,这是一本武功秘籍";
         Item[] items;
```

```
            string[] datas = str.Split('|');
            items = new Item[datas.Length];

            for (int i = 0; i < datas.Length; i++)
            {
                items[i] = new Item(datas[i]);
            }

            for (int i = 0; i < items.Length; i++)
            {
                Console.WriteLine(items[i].ID.ToString() +
items[i].Name + items[i].Explain);
            }
        }
    }
```

字符串的合并

String.Concat

String.Join

```
string str1 = "你好，  ";
string str2 = "我是C#";
string[] array = { "你好", "我是", "C#" };

Console.WriteLine(string.Concat(str1, str2));
Console.WriteLine(string.Join("||",array));
```

字符串的替换和插入

Replace 替换

Insert 插入

Trim 去空格

```

```

```
            string str1 = "你好,我是C# ";
            Console.WriteLine(str1.Replace(',', '|'));
            Console.WriteLine(str1.Replace("C#", "C++"));


            string str = "HelloWorld";
            Console.WriteLine(str.Insert(5, "haha"));

            string str2 = "   Hello World  ";

            Console.WriteLine(str2.Trim());
```

字符串的格式化

```
string format = "{1}活动将于{0}点后{2}";
string x = string.Format(format, "答题", 3, "开启");
Console.WriteLine(x);
Console.WriteLine("{0}说:{1}","张三","你好");
```

索引值必须从0开始 而且必须是连续的

格式化：

格式化数字

```
 Console.WriteLine(string.Format("{0:N1}", 12345));
 Console.WriteLine(string.Format("{0:N2}", 12345));
 Console.WriteLine(string.Format("{0:N3}", 12345.2));
 Console.WriteLine(string.Format("{0:F1}", 12345));
 Console.WriteLine(string.Format("{0:F2}", 12345));

 Console.WriteLine((12345 / 100.0).ToString("#.###"));
 Console.WriteLine((12345 / 100).ToString("#.##"));
```

其他的格式化写法：

```
//格式化货币 转换时默认保留两位小数
Console.WriteLine(string.Format("{0:C}", 1.4));

 //格式化货币 保留一位小数 四舍五入
Console.WriteLine(string.Format("{0:C1}", 111.26));

//格式化十进制
Console.WriteLine(string.Format("{0:D3}", 22));//022
Console.WriteLine(string.Format("{0:D6}", 0x1F));//000031

//格式化十六进制
Console.WriteLine(string.Format("{0:X3}", 15));//00F

//格式化百分比
Console.WriteLine(string.Format("{0:P2}", 0.2458));//24.58%
```

字符串拘留池机制

字符串的长度和大小不可控 有可能需要很大的内存空间

如果每次字符串的赋值或复制都去开辟新空间 不仅仅会浪费空间也会影响系统性能

所以.NET提供了字符串的拘留池机制(缓存池)

字符串拘留池原理：

当使用拘留池时 在程序内部创建了一个容器 里面存储了字符串的内容以及字符串在托管堆上的引用

当需要分配新字符串对象时 会首先在容器中进行检查 查看是否包含了字符串对象

如果存在 则返回已经存在的字符串对象的引用

如果不存在 则会新分配对象 开辟新内存 然后把新对象添加到容器中

```
//调试 断点  窗口->内存->内存1 输入str str2查看内存地址
string str = "HelloWorld";
```

```
string str2 = "HelloWorld";
//直接让str2指向字符串拘留池里存储的字符串 所以 str2和str引用地址相同

//str3使用new进行初始化 拘留池机制不生效 所以和str2，str引用地址不同
string str3 = new string(new char[]
{ 'H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd' });

//再次赋值  是重新创建字符串 会到拘留池里查找 所以str3和str str2地址相同
str3 = "HelloWorld";
Console.WriteLine();
```

练习：

1 写一个方法 判断一个字符串是不是回文

上海自来水来自海上 aba abba

```
         static void Main(string[] args)
        {
            string data = "上海自来水来自海上";
            Console.WriteLine(Fun(data));
        }
        static bool Fun(string data)
        {
            if (string.IsNullOrEmpty(data))
                return false;

            for (int i = 0; i < data.Length / 2; i++)
            {
                if (data[i] != data[data.Length-1 - i])
                {
                    return false;
                }
            }

            return true;
        }
```

2 把一个字符串大小写互转

HelloWorld _ hELLOwORLD

```
        static string Fun(string data)
        {
            if (string.IsNullOrEmpty(data))
                return null;

            char[] cs = data.ToCharArray();

            for (int i = 0; i < cs.Length; i++)
            {
                if (cs[i] >= 'A' && cs[i] <= 'Z')
                {
                    cs[i] = (char)(cs[i] + 32);
                }
                if (cs[i] >= 'a' && cs[i] <= 'z')
                {
                    cs[i] = (char)(cs[i] - 32);
                }
            }
            data = new string(cs);
            return data;
        }
```

Enum 枚举

枚举 一组命名整型常量

```
     enum Animal
    {
        Monkey=100,//Monkey=value
        Dog,
        Cat=300,
```

```
        Rabbit,
        Mouse
    }

    class Program
    {
        static void Main(string[] args)
        {
            Animal ani = Animal.Monkey;
            Console.WriteLine(ani);
            Console.WriteLine((int)ani);//枚举值默认第一个值是0
            ani = Animal.Rabbit;
            Console.WriteLine((int)ani);

            switch (ani)
            {
                case Animal.Monkey:
                    break;
                case Animal.Dog:
                    break;
            }
        }
    }
```

默认情况下 枚举的第一个元素值从0开始 后面的值都是连续的

可以自定义枚举值 值和值之间也可以不连续

Struct 结构体

Struct和Class类似 都可以实现对象

不同点：

类是引用类型 结构是值类型

类支持继承 结构体不支持继承

结构不能声明默认的构造方法

C# 结构体的特征：

1 可以包括方法 字段 索引 运算符重载方法和事件

2 结构体可以自定义构造方法（不能定义默认构造方法） 析构不能定义

3 结构体不能被继承 也不能继承于其他结构和类

4 abstract virtual protected 关键不能用于结构体

5 结构体可以使用new关键字进行实例化 也可以不使用new进行实例化

6 不使用new进行实例化时 必须对成员变量在外部赋值 才可以使用

7 结构体可以实现1个或者多个接口(interface)

```
    struct Student
    {
        public int id;
        public string name;
        public int age;

        //不能显式定义结构体的默认构造
        //public Student()
        //{
        //    id = 100;
        //    name = "game";
        //    age = 22;
        //}

        //析构函数不能定义
        //~Student() { }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //第一种 使用new关键字进行实例化
            Student s1 = new Student();
            Console.WriteLine(s1.id);
            Console.WriteLine(s1.name);
            Console.WriteLine(s1.age);
```

```
28
29              //第二种 不使用new关键字进行实例化 但是必须在外部给变量赋值
30              Student s2;
31              s2.name = "噶，e ";
32              Console.WriteLine(s2.name);
33          }
34      }
```

结构体存在的意义：

因为结构体是值类型 所以对内存的操作都是在栈上进行的

栈的空间虽然比堆小很多 但是因为内存分配的机制 栈的读写速度和效率都要高于堆

所以很多成员数据不是特别大 且 无需使用到继承关系的类型 可以使用struct来定义结构

C#的基本数据类型 也都是由结构体实现的 int char double等

接口(interface)和多继承

C#和C++不同 C#中一个类不能同时继承2个或2个以上的父类(不支持多继承)

所以要实现多继承的功能 就必须通过interface来实现

interface接口的语法很类似C#的抽象类（abstract）

接口为所有子类定义了标准的结构(属性 方法 事件)

让子类在结构上和接口定义的保持一致

```
1   interface IMyInterface
2       {
3           //接口成员 不能有字段
4           //int x;
5           //不能使用 public private等访问修饰符 默认都是public
6           int X { get; set; }
7
8           void Fun1();//接口成员方法 不能实现
9
10      }
11
12      //类继承与接口
```

```
    class MyClass : IMyInterface
    {
        private int x;
        public int X
        {
            get { return x; }
            set { x = value; }
        }

        //子类实现 父类接口提供的方法 添加public修饰符
        public void Fun1()
        {
        }
    }
```

结合以上代码：接口和抽象类很相似 都是负责定义结构格式 由子类去定义对方法的实现

多继承的实现：

```

    class ClassA
    {
    }

    interface IMyInterface
    {
        void Fun1();//接口成员方法 不能实现

    }

    interface IMyInterface2
    {
        void Fun2();
    }

    //类继承于1个类和多个接口
    class MyClass : ClassA,IMyInterface,IMyInterface2
    {
        //子类实现 父类接口提供的方法 添加public修饰符
        public void Fun1() {}
```

```
        public void Fun2() {}
    }

    struct MyStruct:IMyInterface2,IMyInterface
    {
        public void Fun1() { }
        public void Fun2() { }
    }

```

接口和抽象类的相同点：

1 都可以被继承

2 都不能被实例化

3 都可以包含方法声明

4 派生类都必须实现未实现的方法

接口和抽象类的区别：

1 抽象类是一个不完整的类 需要进一步对其细化

而接口只是一种行为规范

2 抽象类可以定义字段 但是接口不能

3 接口可以被多重实现 抽象类只能被单一继承

4 抽象类是从一系列相关对象中抽象出来的概念 因此反应事物的内部共性

接口是为了满足外部调用而定义的一个功能约定 因此反应的是事物的外部共性

5 接口基本上不具备继承的任何具体特点 它仅仅承诺能够调用的方法

6 抽象类实现的具体方法默认是virtual的 但是接口中的方法默认都是非virtual的

拆箱和装箱

拆箱和装箱就是引用类型和值类型之间的转换

C#所有类型的根父类是object类 是所有层次结构的根（最终基类）

```
public class Object
{
    public Object();

    public virtual bool Equals(object obj);
    public static bool Equals(object objA, object objB);
    public virtual int GetHashCode();
    public Type GetType();
    protected object MemberwiseClone();
    public static bool ReferenceEquals(object objA, object objB);
    public virtual string ToString();
}
```

当需要把值类型转换为引用类型时 只需要隐式转换为Object 这就是装箱

当需要把一个Object类型转换为值类型时 只需要显式的转换为值类型 这就是拆箱

拆箱： 引用类型->值类型

装箱： 值类型->引用类型

```
            int num = 100;
            Type type = typeof(int);

            Console.WriteLine(typeof(string));
            Console.WriteLine(typeof(int));
            Console.WriteLine(num.GetType());

            //装箱 值类型转换为引用类型
            object numObj = num;

            if (numObj is int)
            {
                //Int不能使用as进行强转 因为as可能会返回null 但是int等其他值类型不能为null
                int num2 = (int)numObj;//拆箱
            }

            if (numObj.GetType() == typeof(int))
            {
                int num2 = (int)numObj;//拆箱
            }
```

这里int32是一个值类型 因为是一个struct

而根父类 object是class 所以是引用类型

这里装箱和拆箱 就是实现引用类型和值类型的互转

但是因为拆箱多了一个执行过程

所以对于数据量比较大的操作 过于频繁的拆装箱会影响效率

C#实现不定长 不定类型的参数列表

```
        static void Main(string[] args)
        {
            Fun("HelloWorld", 100, 2.3f, 100.2, 'x');
        }

        static void Fun(params object[] objs)
        {
            if (objs.Length <= 0)
                return;

            if (objs[0] != null && objs[0] is string)
            {
                Console.WriteLine(objs[0] as string);
            }
            if (objs[1] != null && objs[1].GetType()==typeof(int))
            {
                Console.WriteLine((int)objs[1]);
            }
        }
```

写一个动物基类

包含 吃 跑 睡觉 三个方法

写多个接口

接口1 游泳

接口2 飞行

接口3 跳

接口4 爬

分别实现 鲨鱼 袋鼠 天鹅 青蛙和蛇类 使用多继承实现