List泛型集合

与ArrayList类似，但有两处不同

1.List泛型在使用前需要声明类型，之后也只能装入该类型的数据，但取数据时不需要判别类型

2.List泛型存取数据时没有发生装箱和拆箱操作，性能更好

一、List的创建

注：List泛型位于System.Collections库中

```
//创建泛型集合对象
List<int> list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);

list.AddRange(new int[] {1,2,3,4,5,6,8});
list.AddRange(list);    //也可以装自己
```

二、List泛型和数组的相互转换

```
//注：List泛型可以和相同类型的数组互相转换
//int型
List<int> list = new List<int>();
int[] nums = list.ToArray();
List<int> list2 =  nums.ToList();

//byte型
List<byte> listByte = new List<byte>();
byte[] bytes = listByte.ToArray();
List<byte> listByte2 = bytes.ToList();
```

三、练习

```
//练习：
//将一个数组中的奇数放到一个集合中，再将偶数放到另一个集合中
//最终将两个集合合并为一个集合，并且奇数显示在左边，偶数显示在右边
int[] nums = {1,3,5,7,8,10,12,11,15,17,20,24,26,48,36,58};
List<int> evenNumList = new List<int>();
List<int> unevenNumList = new List<int>();
for (int i = 0; i < nums.Length; i++)
{
    if(nums[i] % 2 == 0){
        evenNumList.Add(nums[i]);
    }else{
```

```
                unevenNumList.Add(nums[i]);
            }
        }
        unevenNumList.AddRange(evenNumList);
        foreach (int item in unevenNumList)
        {
            System.Console.Write(item + "  ");
        }

        //奇偶排两列:
        int max = evenNumList.Count;    //用于确定奇数和偶数谁更多
        if(max < unevenNumList.Count){
            max = unevenNumList.Count;
        }
        if(max == unevenNumList.Count){     //奇数多于偶数
            for(int i = 0;i < unevenNumList.Count;i++){
                if(i <= evenNumList.Count -1){
                    System.Console.WriteLine(unevenNumList[i] + "\t" +
evenNumList[i]);     //按偶数的数量打印
                }else{
                    System.Console.WriteLine(unevenNumList[i] + "\t" + "");     //奇数
全打印，偶数不足补空
                }
            }
        }else{      //偶数多于奇数
            for(int i = 0;i < evenNumList.Count;i++){
                if(i <= unevenNumList.Count -1){
                    System.Console.WriteLine(unevenNumList[i] + "\t" +
evenNumList[i]);     ///按奇数的数量打印
                }else{
                    System.Console.WriteLine("" + "\t" + evenNumList[i]);       //偶数
全打印，奇数不足补空
                }
            }
        }
```

装箱与拆箱

1.装箱：将值类型转换为引用类型 *(有疑问：string和object都是引用类型且string继承object，若将string类型转换成object类型会不会发生装箱？)*

2.拆箱：将引用类型转换为值类型

3.注意：两种类型必须存在继承关系才可能发生装拆箱。(接口也可以) 若不存在继承关系，即使发生值与引用的转换也不可能存在装拆箱 例：

```
        int n = 10;
        object obj = n;
        int  n2 = (int)obj;     //发生一次装箱
```

ArrayList使用的方法参数多为Object型，所以我们赋非object型数据进去时就会进行一次装箱
如：arrList.Add(object value)
    arrList.Add(2) //将int型的2转换成object型，装箱一次

List<>因为使用前需要声明类型，因此读、写都是相同类型，不会发现装箱与拆箱操作

性能测试

```
    //装箱与不装箱性能测试
    //1.装箱
    ArrayList arrList = new ArrayList();
    Stopwatch sw = new Stopwatch();
    sw.Start();
    for (int i = 0; i < 100000000; i++) //ArrayList赋值要进行装箱操作，循环1亿次测试性能
    {
        arrList.Add(i);
    }
    sw.Stop();
    System.Console.WriteLine("1亿次装箱共用时：" + sw.Elapsed);      //耗时5.9249秒

    //2.不装箱
    List<int> list = new List<int>();
    Stopwatch sw2 = new Stopwatch();
    sw2.Start();
    for (int i = 0; i < 10000000; i++)  //List不发生装箱拆箱操作，循环1亿次测试性能
    {
        list.Add(i);
    }
    sw2.Stop();
    System.Console.WriteLine("List进行1亿次共用时：" + sw2.Elapsed);     //耗时0.0456秒
    //可见装箱与否有超过100倍的差距
```

注意事项：

无继承(包括接口)关系不可能发生装拆箱

```
    //如int和string，不存在继承关系
    string str = "123";
    int num = Convert.ToInt32(str);      //不发生装拆箱
```