**数组、List和ArrayList的区别**

数组在内存中是连续存储的，所以它的索引速度是非常的快，而且赋值与修改元素也很简单，

string[] s=new string[3];

//赋值

s[0]="a"; s[1]="b"; s[2]="c";

//修改

s[1]="b1";

但是数组也存在一些不足的地方。比如在数组的两个数据间插入数据也是很麻烦的，还有我们在声明数组的时候，必须同时指明数组的长度，数组的长度过长，会造成内存浪费，数组和长度过短，会造成数据溢出的错误。这样如果在声明数组时我们并不清楚数组的长度，就变的很麻烦了。C#中最先提供了ArrayList对象来克服这些缺点。

ArrayList是.Net Framework提供的用于数据存储和检索的专用类，它是命名空间System.Collections下的一部分。它的大小是按照其中存储的数据来动态扩充与收缩的。所以，我们在声明ArrayList对象时并不需要指定它的长度。ArrayList继承了IList接口，所以它可以很方便的进行数据的添加，插入和移除.

ArrayList list = new ArrayList();

//新增数据

list.Add("abc"); list.Add(123);

//修改数据

list[2] = 345;

//移除数据

list.RemoveAt(0);

//插入数据

list.In从上面示例看，ArrayList好像是解决了数组中所有的缺点，那么它应该就是完美的了，为什么在C#2.0后又会出现List呢？

在list中，我们不仅插入了字符串"abc"，而且又插入了数字123。这样在ArrayList中插入不同类型的数据是允许的。因为ArrayList会把所有插入其中的数据都当作为object类型来处理。这样，在我们使用ArrayList中的数据来处理问题的时候，很可能会报类型不匹配的错误，也就是说ArrayList不是类型安全的。既使我们保证在插入数据的时候都很小心，都有插入了同一类型的数据，但在使用的时候，我们也需要将它们转化为对应的原类型来处理。这就存在了装箱与拆箱的操作，会带来很大的性能损耗。

装箱与拆箱的概念： 简单的来讲:

装箱：就是将值类型的数据打包到引用类型的实例中 比如将int类型的值123赋给object对象o

　　　　int i=123; object o=(object)i;

拆箱：就是从引用数据中提取值类型 比如将object对象o的值赋给int类型的变量i

　　　　object o=123; int i=(int)o;

装箱与拆箱的过程是很损耗性能的。

正是因为ArrayList存在不安全类型与装箱拆箱的缺点，所以在C#2.0后出现了泛型的概念。而List类是ArrayList类的泛型等效类。它的大部分用法都与ArrayList相似，因为List类也继承了IList接口。最关键的区别在于，在声明List集合时，我们同时需要为其声明List集合内数据的对象类型。 比如：

List<int> list = new List<int>();

//新增数据

list.Add(123);

//修改数据

list[0] = 345;

//移除数据

list.RemoveAt(0);

上例中，如果我们往List集合中插入string字符"hello world"，IDE就会报错，且不能通过编译。这样就避免了前面讲的类型安全问题与装箱拆箱的性能问题了。

同时 List不能被构造，但可以向上面那样为List创建一个引用，而ListArray就可以被构造。

List list; //正确 list=null;

List list=new List(); // 是错误的用法

List list = new ArrayList();这句创建了一个ArrayList的对象后把上溯到了List。此时它是一个List对象了，有些ArrayList有但是List没有的属性和方法，它就不能再用了。 而ArrayList list=new ArrayList();创建一对象则保留了ArrayList的所有属性。

List泛型的好处：

通过允许指定泛型类或方法操作的特定类型，泛型功能将类型安全的任务从您转移给了编译器。不需要编写代码来检测数据类型是否正确，因为会在编译时强制使用正确的数据类型。减少了类型强制转换的需要和运行时错误的可能性。泛型提供了类型安全但没有增加多个实现的开销。sert(0, "hello world");