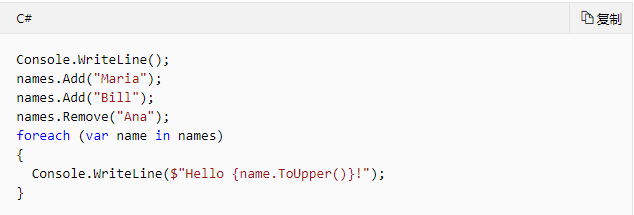
创建的集合使用List<T>（列表）类型。此类型存储一系列元素。元素类型是在尖括号内指定。



List<T>类型的一个重要方面是，既可以扩大，也可以收缩，方便用户添加或删除元素。

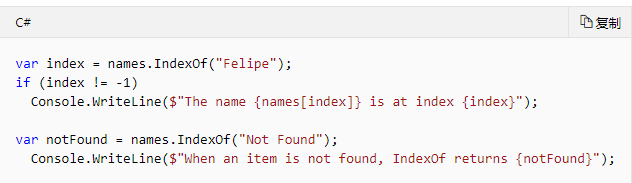


还可以按索引下标引用各项。

Names[0],names[1]等

不得访问超出列表末尾的项。可以使用count属性确定列表长度。

一般使用列表中存储的数据可能非常多，有时可能会包含数以千计的元素。IndexOf方法可搜索项，并返回此项的索引。



还可以对列表中的项进行排序。Sort方法按正常顺序（如果是字符串的话，按字母顺序排序）对列表中的所有项进行排序。

**集合和数据结构**

类似的数据在作为集合而存储和操作时通常可以得到更高效的处理。

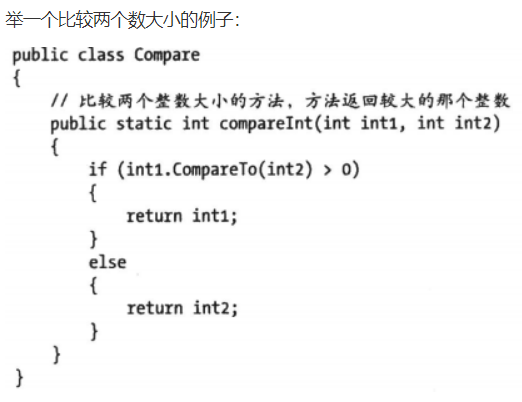
泛型和非泛型集合：

什么是泛型：（内容网址）https://www.cnblogs.com/lyg520/p/9798743.html

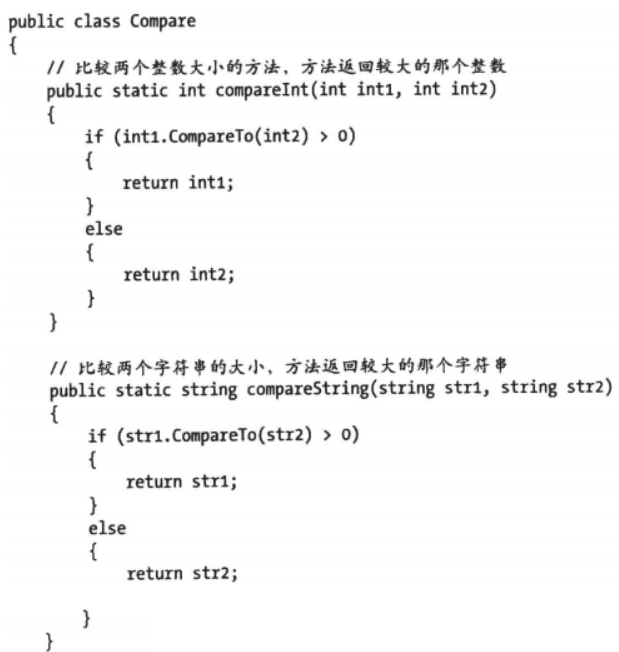
泛型的就是“通用类型”，它可以代替任何的数据类型，使类型参数化，从而达到只实现一个方法就可以操作多个数据类型的目的。

为什么要使用泛型

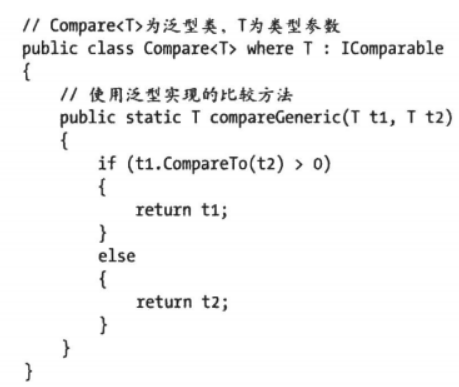
例子如下所示：



只对两个int类型数据比较大小是完全没有问题的，但是如果客户现在增加需要“又可以实现两个字符串大小的比较”，此时就不得不在类中添加一个新的比较大小的方法了。



我们可以看到两个方法中有大部分代码是类似的，所以微软提出了一个激动人心的特性—泛型，他使得类型可以被参数化（即输入的是什么就是什么类型）



Where语句是类型参数的约束它用来使参数可以适用于CompareTo方法。

向泛型中加入元素的效率远比非泛型数组高，原因是非泛型ArrayList的Add（Object value）方法中，参数为object类型，当把int参数i传入方法时，会发生装箱操作，从而导致性能的损失，使运行时间变长。

泛型可以保证类型安全，当你项int类型数组添加string类型的值时，会造成“无法从string类型转换为int类型”的错误，因为你用int类型初始化了泛型类型。

泛型参数解析：

1. 类型参数

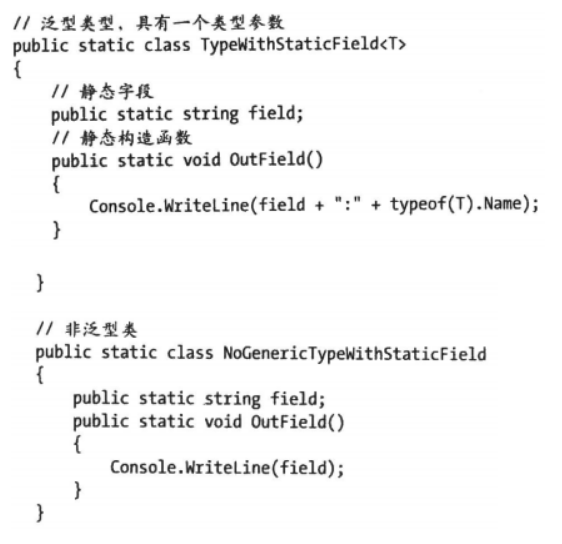
根据泛型类型参数是否已经提供实际类型，可分为未绑定的泛型和已构造的泛型，

开放类型指包含参数类型的泛型，所有未绑定的类型都属于开放类型；而封闭类型指已经为每个参数都指定了实际数据类型的泛型



1. 泛型中的静态字段和静态函数问题

对于非泛型类，定义了一个静态字段，不管是创建了多少个该类的实例，也不管从该类派生出多少个实例，都只存在一个字段，但每个封闭的泛型类型中都有仅属于他自己的静态字段。

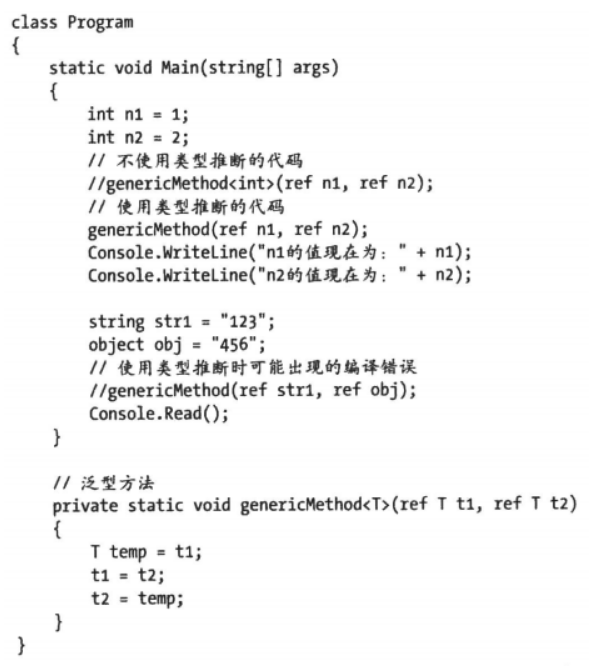




这是因为，在使用实际类型参数代替泛型参数时，编译器会根据不同的类型参数，重新生成类型。对于静态构造函数，也是如此，每个封闭的泛型类型都有一个静态构造函数。

1. 类型参数的推断

写泛型的时候可以省略掉“<>”，具体实际类型由编译器自行判断



Ref关键字（来源自MSDN）：ref关键字使参数按引用传递。

举例如下所示：  
class Program

{

static void Main(string[] args)

{

int a = 1;

int b = 2;

Console.WriteLine("交换前\ta={0}\tb={1}\t",a,b);

Swap(a,b);

Console.WriteLine("交换后\ta={0}\tb={1}\t",a,b);

Console.Read();

}

//交换a,b两个变量的值

private static void Swap(int a,int b)

{

int temp = a;

a = b;

b = temp;

Console.WriteLine("方法内\ta={0}\tb={1}\t",a,b);

}

}

运行结果：

交换前 a=1 b=2

方法内 a=2 b=1

交换后 a=1 b=2

**原因分析**：int类型为值类型，它存在于线程的堆栈中。当调用swap（a,b）方法时，相当于把a，b的值（即1，2）**拷贝一份**，然后在方法内交换这两个值。交换完之后，a还是原来的a，b还是原来的b。这就是C#中按值传递的原理，传递的变量所对应数据的一个拷贝，而非引用。

Ref的效果是，当控制权传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。

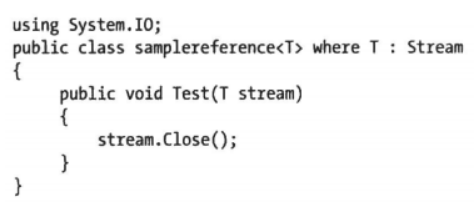
若要使用ref参数，则方法定义和调用方法都必须显示使用ref关键字。

总的来说就是：

通常在值传递时，只是传递的对象的副本，对副本的操作，不能影响到对象本身，而加上ref关键字后，所做的任何修改，都将影响到对象本身。

1. 类型参数的约束
2. 引用类型约束

表示形式：T:class



以上代码中，where T:Stream告诉编译器：传入的类型实参必须是System.IO.Stream，或者是从Stream派生出的一个类型。

如果一个类型参数没有指定约束，那么默认T为System.Object类型。但若在代码中显示指定了System.Object约束，则编译器就会报错。

约束不能是特殊类Object。

1. 值类型约束

表现形式：T:struct

它确保传递的类型实参是只类型（包含枚举），但这里的值类型不包含可空类型。

泛型集合

字符串可以说是一个字符的集合，和字符串一样，数据对象也可以是集合的方法存在，所以泛型类对象也可以是集合的方式存在（泛型集合）。

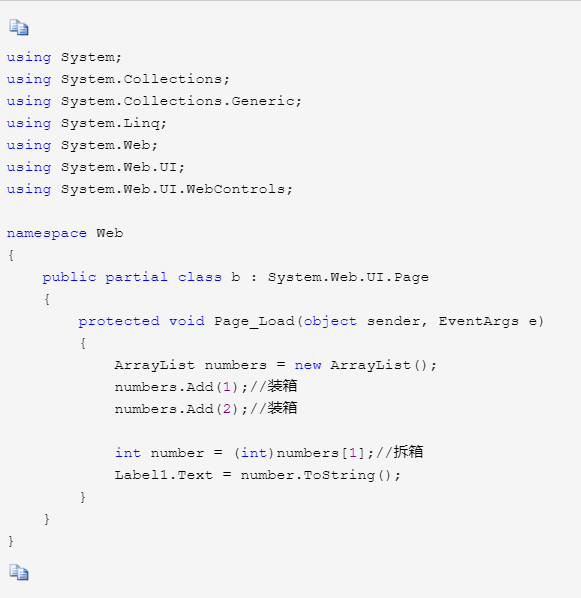
同传统的集合相比，泛型集合是一种强类型的集合。它解决了类型安全问题，同时避免了集合中每次的装箱和拆箱的操作，提升了性能。

泛型集合类型：

1. List，这是我们应用最多的泛型种类，它对应ArrayList（非泛型集合）集合。
2. Dictionary，对应Hashtable集合。
3. Collection对应于CollectionBase
4. ReadOnlyCollection对应于ReadOnlyCollectionBase。
5. Queue，Stack和SortedList，它们对应于与它们同名的非泛型类。

比较泛型集合和非泛型集合：

以ArrayList和list为例

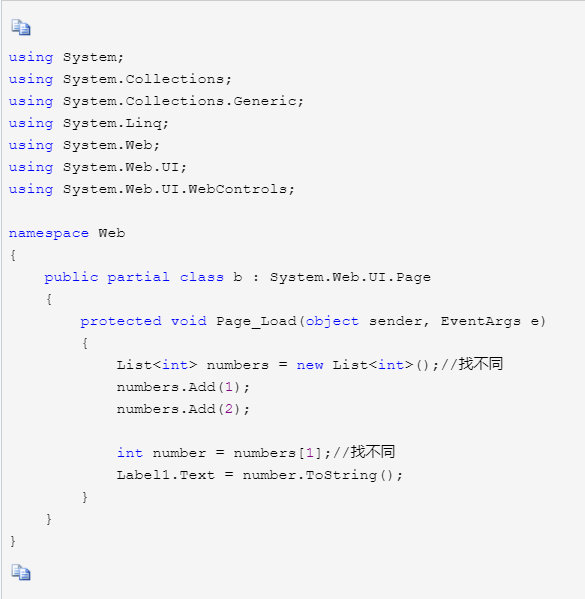


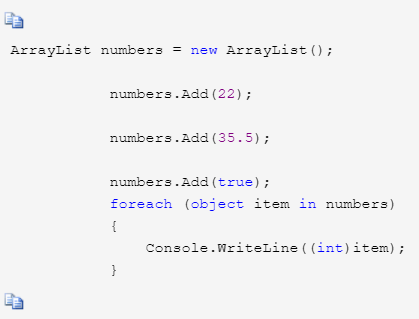
首先，ArrayList中将所有元素都看成Object类型，是引用类型。调用Add方法增加两个整数，在这个过程中，整数1,2被CLR装箱成object类型，而后两个元素又被拆箱。

整个过程如下：

1. 在托管堆中分配一个新的object
2. 基于栈的数据必须移动到刚分配的内存区中
3. 当拆箱时，位于堆中的数据又得移动到栈中
4. 堆中无用的数据进行垃圾回收

当涉及到大量装箱与拆箱操作时，必然会影响应用程序的性能。而是用泛型的集合类时就会减少装箱和拆箱的工作。





因为int，float等数值类型可以被装箱成object，因此即使ArrayList增加的类型不一致，编译器也不会报错。

但是如果是泛型类型比如List<T>,那么在编译时就会进行类型检查。防止运行时错误。

List<T>和ArrayList 的区别：

List<T>和ArrayList的相同点：添加元素、删除元素、通过索引访问元素方法相同。

不同点：

ArrayList可以添加任意类型元素，List<T>对添加的元素具有类型约束

ArrayList添加装箱时，读取时拆箱；List<T>不需要装箱，拆箱操作