为泛型集合类或表示集合中项的泛型类定义接口通常很有用。对于泛型类，使用泛型接口十分可取，例如**使用 IComparable<T> 而不使用 IComparable**，这样可以**避免值类型的装箱和取消装箱**操作。.NET Framework 2.0 类库定义了若干新的泛型接口，以用于 **System.Collections.Generic** 命名空间中新的集合类。

//Type parameter T in **angle brackets.**
public class GenericList**<T>** : System.Collections.Generic.IEnumerable<T>

==========

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace 泛型
{
    class 泛型接口
    {
        public static void Main()
        {
            PersonManager man = new PersonManager();
            Person per = new Person();
            man.PrintYourName(per);
            Person p1 = new Person();
            p1.Name = "p1";
            Person p2 = new Person();
            p2.Name = "p2";
            man.SwapPerson<Person>(ref p1, ref p2);
            Console.WriteLine( "P1 is {0} , P2 is {1}" , p1.Name ,p2.Name);
            Console.ReadLine();
        }
    }
    //泛型接口
    interface IPerson<T>
    {
        void PrintYourName( T t);
    }
    class Person
    {
        public string Name = "aladdin";
    }
    class PersonManager : IPerson<Person>
    {
        #region IPerson<Person> 成员
        public void PrintYourName( Person t )
        {
            Console.WriteLine( "My Name Is {0}!" , t.Name );
        }
        #endregion
        //交换两个人，哈哈。。这世道。。
        //泛型方法T类型作用于参数和方法体内
        public void SwapPerson<T>( ref  T p1 , ref T p2)
        {
```

```
            T temp = default(T) ;
            temp = p1;
            p1 = p2;
            p2 = temp;
        }
    }
}
```

==========

1.什么是泛型?

在介绍泛型之前，先看一个小的事例。

```
/// <summary>
/// 计算两个整形相加
/// </summary>
/// <param name="a"></param>
/// <param name="b"></param>
/// <returns></returns>
public int Add(int a, int b)
{
    return a + b;
}
/// <summary>
/// 计算两个浮点型相加
/// </summary>
/// <param name="a"></param>
/// <param name="b"></param>
/// <returns></returns>
public decimal Add(decimal a, decimal b)
{
    return a + b;
}
```

上述是两个非常简单的方法，实现了两个数的相加操作。但是如果细心的话，也会发现两个方法有相同的地方，我们可以写成一下的格式 **public T Add(T a,T b)**,至于T是什么东东，不管他，反正就是一个占位符，就好比上述的Int 或者Decimal，所以如果有这样一个占位符，能够代替所有的类型不就好了吗？因此.net专门提供了专门的语法来替代占位符，其中一种就是在定义类型的时候传递，此时，类型也就变成了泛型类。

小结：泛型允许我们声明类型参数化的代码，我们可以用不同的类型进行实例化。也就是说，我们可以用"类型占位符"来写代码，然后在创建类的实例时提供真实的类型。

2.泛型类

声明一个泛型类也特别的简单，跟平常的类书写大致相似，只不过把需要改变的地方用类型占位符替换即可。

```
/// <summary>
/// 泛型类
/// </summary>
/// <typeparam name="T1">泛型参数</typeparam>
/// <typeparam name="T2">泛型参数</typeparam>
class SomeClass<T1,T2> where T1 :new() where T2:new()
{
    //根据泛型参数，创建一个类
```

```
    public T1 SomeVar = new T1();
    public T2 OtherVar = new T2();

}
```

看了上述代码是否感觉特别的清爽，就是这么简单，与普通类的区别就是在类名之后放置了一组尖括号，并且在尖括号中用逗号分隔占位符来表示希望提供的类型。而在泛型类声明的主体中使用类型参数来表示应该被替代的类型。

3.类型参数的约束

如果你在泛型类中，直接运用占位符来执行+ -操作的话，编译器会直接报错，因为此时编译器并不知道T的具体类型，尽管所有的类型都继承自System.Object，因为泛型类并不知道他们保存的是什么类型，不会知道这些类型实现的成员。

看上述实例，就会发现为何在泛型类中多出了一个where字句，这就是用来约束占位符。

1.每个占位符都有自己的where子句

2.如果形参有多个约束，他们在where子句中使用逗号分隔

3.一般用到的约束如下

Class：任何引用类型，包括类、数组、委托和接口都可以用作实参

Struct：任何值类型都可以被用作类型实参

Interface：只有这个接口或者实现接口的类型才能用作实参

New():任何带有无参构造函数的类型都可以用作实参，这叫做构造函数约束

```
/// <summary>
/// 泛型类
/// </summary>
/// <typeparam name="T1">泛型参数</typeparam>
/// <typeparam name="T2">泛型参数</typeparam>
class SomeClass<T1,T2> where T1 :class,new() where T2:class,new()
{
    //根据泛型参数，创建一个类
    public T1 SomeVar = new T1();
    public T2 OtherVar = new T2();
}
```

```
/// <summary>
/// 泛型类
/// </summary>
/// <typeparam name="T1">泛型参数</typeparam>
/// <typeparam name="T2">泛型参数</typeparam>
class SomeClass<T1,T2> where T1 :class,new() where T2:class,new()
{
    //根据泛型参数，创建一个类
    public T1 SomeVar = new T1();
    public T2 OtherVar = new T2();
}
```

看上述代码，主要是注意一下泛型约束的格式规则

注意：泛型约束也会涉及到一个次序的问题，存在构造函数的约束，new()必须放在最后

4.泛型方法

既然知道了泛型类，泛型方法也就明白多了，写法基本上是相同的。

```
 /// <summary>
      /// 泛型方法
      /// </summary>
      /// <typeparam name="S">泛型参数</typeparam>
      /// <typeparam name="T">泛型参数</typeparam>
      /// <param name="p">参数</param>
      public void PrintData<S, T>(S p) where S : class
      {
         ///代码体
      }
```

上面就是一个泛型方法的实例，基本的操作与普通的方法相似。