**c#: 泛型接口、泛型方法**

泛型可以自定义泛型接口、[泛型类](https://so.csdn.net/so/search?q=%E6%B3%9B%E5%9E%8B%E7%B1%BB&spm=1001.2101.3001.7020)、泛型方法、泛型事件、泛型委托。

1>自定义泛型接口

和普通接口一样，一个泛型接口通常也是与某些对象相关的约定规程。泛型接口的声明如下：

interface  [接口名]<T>

{

[接口体]

}

在c#中，通过尖括号“<>”将类型参数括起来，表示泛型。声明泛型接口时，与声明一般接口的唯一区别是增加了一个<T>。一般来说，声明泛型接口与声明非泛型接口遵循相同的规则。

泛型接口定义完成之后，就要定义此接口的子类。定义泛型接口的子类有以下两种方法。

(1)直接在子类后声明泛型。

(2)在子类实现的接口中明确的给出泛型类型。

**interface Inter<T>**

**{**

**void show(T t);**

**}**

**//定义接口Inter的子类InterImpA,明确泛型类型为String （2）**

**public class InterImpA : Inter<String>**

**{**

**//子类InterImpA重写方法show,指明参数类型为string**

**public void show(String t)**

**{**

**Console.WriteLine(t);**

**}**

**}**

**//定义接口Inter的子类InterImpB，直接声明 （1）**

**public class InterImpB<T> : Inter<T>**

**{**

**public void show(T t)**

**{**

**Console.WriteLine(t);**

**}**

**}**

**static void Main(string[] args)**

**{**

**InterImpA i = new InterImpA();**

**i.show("aaa");**

**InterImpB<Int32> j = new InterImpB<Int32>();**

**j.show(555);**

**Console.Read();**

**}**

运行结果：

https://img-blog.csdn.net/20180103164941575

2>泛型函数

i)非泛型类中的泛型方法

**public class SortHelper**

**{**

**public void BubbleSort<T>(T[] array) where T : IComparable**

**{**

**int length = array.Length;**

**for (int i = 0; i <= length - 2; i++)**

**{**

**for (int j = length - 1; j >= 1; j--)**

**{**

**//如果前面的元素较大，交换相邻两个元素**

**if (array[j].CompareTo(array[j - 1]) < 0)**

**{**

**T temp = array[j];**

**array[j] = array[j - 1];**

**array[j - 1] = temp;**

**}**

**}**

**}**

**}**

**}**

**static void Main(string[] args)**

**{**

**SortHelper sortHper = new SortHelper();**

**string[] arr = new string[3]{"A","c","b"};**

**sortHper.BubbleSort<string>(arr);**

**foreach(string s in arr)**

**{**

**Console.WriteLine(s);**

**}**

**Console.Read();**

**}**

运行结果：

https://img-blog.csdn.net/20180103181442846  
注：因为“<”和“>”运算符只对数值类型参数有效，要比较两个泛型类型参数值的大小，不能直接用“<”和">"运算符。解决方法是约束泛型T为“where T:IComparable”，表明泛型T是继承自IComparable接口，可以使用IComparable接口的CompareTo方法比较大小。

泛型方法的泛型参数可以用在该方法的形参，方法体，返回值三处。上述例子的泛型方法的泛型参数用在了形参处。

ii)泛型类中的泛型方法

非泛型类的泛型方法，泛型参数直接在方法名称后面声明；泛型类的泛型方法是在类名称后面加一个尖括号，使用这个尖括号来传递占位符(也就是类型参数)，再在类中定义泛型方法。

前面BubbleSort泛型方法也可以这样定义：

**public class SortHelper<T> where T:IComparable**

**{**

**public void BubbleSort(T[] array)**

**{**

**int length = array.Length;**

**for (int i = 0; i <= length - 2; i++)**

**{**

**for (int j = length-1; j >= 1; j--)**

**{**

**//如果前面的元素较大，交换相邻两个元素**

**if (array[j].CompareTo(array[j - 1]) < 0)**

**{**

**T temp = array[j];**

**array[j] = array[j-1];**

**array[j-1] = temp;**

**}**

**}**

**}**

**}**

**}**

**static void Main(string[] args)**

**{**

**SortHelper<string> sortHper = new SortHelper<string>();**

**string[] arr = new string[3]{"A","c","b"};**

**sortHper.BubbleSort(arr);**

**foreach(string s in arr)**

**{**

**Console.WriteLine(s);**

**}**

**Console.Read();**

**}**

上述代码中，因为泛型类已经声明了泛型参数T，可以直接使用下面的语句：public void BubbleSort(T[] array)定义泛型类的泛型方法。要调用泛型类的泛型方法时，首先声明泛型类的对象，接着使用泛型类的对象调用泛型方法时与调用普通方法完全一样。

3、泛型约束

泛型约束是使用where关键字实现的，用来约束T到一个指定范围。

常用约束类型

where T：结构                   类型参数必须是值类型。可以指定除Nullable以外的任何值类型

where T：类                       类型参数必须是引用类型，包括任何类、接口、委托或数组类型

where T ：new()                类型参数必须具有无参数的公共构造函数。当与其他约束一起使用时，new()约束必须最后指定

where T：<基类名>            类型参数必须是指定的基类或派生自指定的基类

where T：<接口名称>        类型参数必须是指定的接口或实现指定的接口。可以指定多个接口约束。约束接口也可以是泛型的  
public class Book : IComparable

    {

        private int id;

        private string title;

        public Book()

        {

        }

        public Book(int id, string title)

        {

            this.id = id;

            this.title = title;

        }

        public int ID

        {

            get { return id; }

            set { id = value; }

        }

        public string Title

        {

            get { return title; }

            set { title = value; }

        }

        public int CompareTo(object obj)

        {

            Book book = (Book)obj;

            return this.ID.CompareTo(book.ID);

        }

    }

    public class SortHelper<T> where T : IComparable

    {

        public void BubbleSort(T[] array)

        {

            int length = array.Length;

            for (int i = 0; i <= length - 2; i++)

            {

                for (int j = length - 1; j >= 1; j--)

                {

                    //如果前面的元素较大，交换相邻两个元素

                    if (array[j].CompareTo(array[j - 1]) < 0)

                    {

                        T temp = array[j];

                        array[j] = array[j - 1];

                        array[j - 1] = temp;

                    }

                }

            }

        }

    }

    static void Main(string[] args)

    {

        Book[] bookArr = new Book[2];

        Book book1 = new Book(1, "语文");

        Book book2 = new Book(2,"数学");

        bookArr[0] = book1;

        bookArr[1] = book2;

        SortHelper<Book> Sort = new SortHelper<Book>();

        Sort.BubbleSort(bookArr);

        foreach (Book b in bookArr)

        {

            Console.WriteLine("ID={0},Title={1}",b.ID,b.Title);

        }

        Console.Read();

    }

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

在比较复杂类型对象Book,book1和book2到底谁大，这涉及一个判断依据的问题。如何来实现这种复杂类型对象的比较呢？答案是：让需要进行比较的对象类实现IComparable接口。也就是说，只有实现了IComparable接口的类型才能作为类型参数被传入，即需要对传入参数的类型进行一些约定，这就是要讲的泛型约束。

c#泛型要求对“任何泛型类型或泛型方法的类型参数”的任何假定，都要基于“显式约束”，以维护c#所要求的类型安全。“显式约束”有where字句表达，可以指定”基类约束”，“接口约束”，“构造器约束”，“值类型/引用类型约束”共4种约束。“显式约束”并非必需，如果没有指定“显式约束”，泛型类型参数只能访问System.Object类型中的公有方法。