**泛型接口（C# 编程指南）**

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>](https://msdn.microsoft.com/zh-cn/library/4d7sx9hd.aspx) 而不使用 [IComparable](https://msdn.microsoft.com/zh-cn/library/system.icomparable.aspx)，这样可以避免值类型的装箱和取消装箱操作。.NET Framework 类库定义了若干泛型接口，以用于 [System.Collections.Generic](https://msdn.microsoft.com/zh-cn/library/system.collections.generic.aspx) 命名空间中的集合类。

将接口指定为类型参数的约束时，只能使用实现此接口的类型。下面的代码示例显示从 SortedList<T> 类派生的 GenericList<T> 类。有关更多信息，请参见 [泛型介绍（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/0x6a29h6.aspx)。 SortedList<T> 添加约束 where T : IComparable<T>。这将使 SortedList<T> 中的 **BubbleSort** 方法能够对列表元素使用泛型 [CompareTo](https://msdn.microsoft.com/zh-cn/library/43hc6wht.aspx) 方法。在此示例中，列表元素为简单类，即实现 Person 的 IComparable<Person>。

//Type parameter T in angle brackets.

public class GenericList<T> : System.Collections.Generic.IEnumerable<T>

{

protected Node head;

protected Node current = null;

// Nested class is also generic on T

protected class Node

{

public Node next;

private T data; //T as private member datatype

public Node(T t) //T used in non-generic constructor

{

next = null;

data = t;

}

public Node Next

{

get { return next; }

set { next = value; }

}

public T Data //T as return type of property

{

get { return data; }

set { data = value; }

}

}

public GenericList() //constructor

{

head = null;

}

public void AddHead(T t) //T as method parameter type

{

Node n = new Node(t);

n.Next = head;

head = n;

}

// Implementation of the iterator

public System.Collections.Generic.IEnumerator<T> GetEnumerator()

{

Node current = head;

while (current != null)

{

yield return current.Data;

current = current.Next;

}

}

// IEnumerable<T> inherits from IEnumerable, therefore this class

// must implement both the generic and non-generic versions of

// GetEnumerator. In most cases, the non-generic method can

// simply call the generic method.

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

}

public class SortedList<T> : GenericList<T> where T : System.IComparable<T>

{

// A simple, unoptimized sort algorithm that

// orders list elements from lowest to highest:

public void BubbleSort()

{

if (null == head || null == head.Next)

{

return;

}

bool swapped;

do

{

Node previous = null;

Node current = head;

swapped = false;

while (current.next != null)

{

// Because we need to call this method, the SortedList

// class is constrained on IEnumerable<T>

if (current.Data.CompareTo(current.next.Data) > 0)

{

Node tmp = current.next;

current.next = current.next.next;

tmp.next = current;

if (previous == null)

{

head = tmp;

}

else

{

previous.next = tmp;

}

previous = tmp;

swapped = true;

}

else

{

previous = current;

current = current.next;

}

}

} while (swapped);

}

}

// A simple class that implements IComparable<T> using itself as the

// type argument. This is a common design pattern in objects that

// are stored in generic lists.

public class Person : System.IComparable<Person>

{

string name;

int age;

public Person(string s, int i)

{

name = s;

age = i;

}

// This will cause list elements to be sorted on age values.

public int CompareTo(Person p)

{

return age - p.age;

}

public override string ToString()

{

return name + ":" + age;

}

// Must implement Equals.

public bool Equals(Person p)

{

return (this.age == p.age);

}

}

class Program

{

static void Main()

{

//Declare and instantiate a new generic SortedList class.

//Person is the type argument.

SortedList<Person> list = new SortedList<Person>();

//Create name and age values to initialize Person objects.

string[] names = new string[]

{

"Franscoise",

"Bill",

"Li",

"Sandra",

"Gunnar",

"Alok",

"Hiroyuki",

"Maria",

"Alessandro",

"Raul"

};

int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 108, 72, 30, 35 };

//Populate the list.

for (int x = 0; x < 10; x++)

{

list.AddHead(new Person(names[x], ages[x]));

}

//Print out unsorted list.

foreach (Person p in list)

{

System.Console.WriteLine(p.ToString());

}

System.Console.WriteLine("Done with unsorted list");

//Sort the list.

list.BubbleSort();

//Print out sorted list.

foreach (Person p in list)

{

System.Console.WriteLine(p.ToString());

}

System.Console.WriteLine("Done with sorted list");

}

}

可将多重接口指定为单个类型上的约束，如下所示：

class Stack<T> where T : System.IComparable<T>, IEnumerable<T>

{

}

一个接口可定义多个类型参数，如下所示：

interface IDictionary<K, V>

{

}

适用于类的继承规则同样适用于接口：

interface IMonth<T> { }

interface IJanuary : IMonth<int> { } //No error

interface IFebruary<T> : IMonth<int> { } //No error

interface IMarch<T> : IMonth<T> { } //No error

//interface IApril<T> : IMonth<T, U> {} //Error

如果泛型接口为逆变的，即仅使用其类型参数作为返回值，则此泛型接口可以从非泛型接口继承。在 .NET Framework 类库中，[IEnumerable<T>](https://msdn.microsoft.com/zh-cn/library/9eekhta0.aspx) 从 [IEnumerable](https://msdn.microsoft.com/zh-cn/library/system.collections.ienumerable.aspx) 继承，因为 [IEnumerable<T>](https://msdn.microsoft.com/zh-cn/library/9eekhta0.aspx) 只在 [GetEnumerator](https://msdn.microsoft.com/zh-cn/library/s793z9y2.aspx) 的返回值和 [Current](https://msdn.microsoft.com/zh-cn/library/58e146b7.aspx) 属性 getter 中使用 T。

具体类可以实现已关闭的构造接口，如下所示：

interface IBaseInterface<T> { }

class SampleClass : IBaseInterface<string> { }

只要类参数列表提供了接口必需的所有参数，泛型类便可以实现泛型接口或已关闭的构造接口，如下所示：

interface IBaseInterface1<T> { }

interface IBaseInterface2<T, U> { }

class SampleClass1<T> : IBaseInterface1<T> { } //No error

class SampleClass2<T> : IBaseInterface2<T, string> { } //No error

对于泛型类、泛型结构或泛型接口中的方法，控制方法重载的规则相同。有关更多信息，请参见 [泛型方法（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/twcad0zb.aspx)。

## 泛型接口完整实例

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace csharp\_basic\_generic\_interface  {  public interface IOutputable<T> //泛型接口  {  void output(T t);  }  class MyOutput<T> : IOutputable<T>//这个泛型类实现泛型接口  {  public void output(T t)  {  Console.WriteLine(t);  }  }  public class Person : IComparable  {  string name;  int age;  public string Name  {  get { return name; }  set { name = value; }  }  public int Age  {  get { return age; }  set { age = value; }  }  public Person(string s, int i)  {  name = s;  age = i;  }  // This will cause list elements to be sorted on age values.    public override string ToString()  {  return name + ":" + age;  }  // Must implement Equals.  public bool Equals(Person p)  {  return (this.age == p.age && this.name == p.name);  }  public int CompareTo(object obj)  {  Person p = (Person)obj;  return this.age - p.age;  }  }  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;  }  }  }  }  }    internal class Program  {    static void TestInfterface()  {  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();  }  static void TestInfterface2()  {  Book[] bookArr = new Book[2];  Book book1 = new Book(2, "语文");  Book book2 = new Book(1, "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();  }  static void TestInfterface3()  {  Person[] persons = { new Person("Jack", 20), new Person("Mary", 19), new Person("Linda", 21) };  SortHelper<Person> helper = new SortHelper<Person>();  helper.BubbleSort(persons);  foreach (Person p in persons)  {  Console.WriteLine("Name:{0},Age:{1}",p.Name,p.Age);  }  Console.Read();  }  static void TestMyOutput()  {  Person[] persons = { new Person("Jack", 20), new Person("Mary", 19), new Person("Linda", 21) };  MyOutput<Person> output = new MyOutput<Person>();  foreach (var item in persons)  {  output.output(item);  }  }  static void Main(string[] args)  {  //TestInfterface();  //TestInfterface2();  //TestInfterface3();  TestMyOutput();  }  }  } |

**请参阅**

[C# 编程指南](https://msdn.microsoft.com/zh-cn/library/67ef8sbd.aspx)

[泛型介绍（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/0x6a29h6.aspx)

[接口（C# 参考）](https://msdn.microsoft.com/zh-cn/library/87d83y5b.aspx)

[.NET Framework 中的泛型](https://msdn.microsoft.com/zh-cn/library/ms172192.aspx)