泛型

这些类和方法将一个或多个类型的指定推迟到客户端代码声明并实例化该类或方法的时候。

例如，通过使用泛型类型参数 T，您可以编写其他客户端代码能够使用的单个类，而不致引入运行时强制转换或装箱操作的成本或风险，

使用泛型类型可以最大限度地重用代码、保护类型的安全以及提高性能。

泛型最常见的用途是创建集合类。

泛型类和泛型方法同时具备可重用性、类型安全和效率，这是非泛型类和非泛型方法无法具备的。

泛型通常用与集合以及作用于集合的方法一起使用。

间 [System.Collections.Generic](https://msdn.microsoft.com/zh-cn/library/system.collections.generic.aspx)其中包含几个新的基于泛型的集合类

# 泛型的优点

但这种方便是需要付出代价的。添加到 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 中的任何引用或值类型都将隐式地向上强制转换为 [Object](https://msdn.microsoft.com/zh-cn/library/system.object.aspx)。如果项是值类型，则必须在将其添加到列表中时进行装箱操作，在检索时进行取消装箱操作。强制转换以及装箱和取消装箱操作都会降低性能；在必须对大型集合进行循环访问的情况下，装箱和取消装箱的影响非常明显。

另一个限制是缺少编译时类型检查

举例::

// The .NET Framework 2.0 way to create a list

List<int> list1 = new List<int>();

// No boxing, no casting:

list1.Add(3);

比较

System.Collections.ArrayList list = new System.Collections.ArrayList();

// Add an integer to the list.

list.Add(3);

// Add a string to the list. This will compile, but may cause an error later.

list.Add("It is raining in Redmond.");

int t = 0;

// This causes an InvalidCastException to be returned.

foreach (int x in list)

{

t += x;

}

# 类型参数使用

GenericList<float> list1 = new GenericList<float>();

GenericList<ExampleClass> list2 = new GenericList<ExampleClass>();

GenericList<ExampleStruct> list3 = new GenericList<ExampleStruct>();

在每个 GenericList<T> 实例中，类中出现的每个 T 都会在运行时替换为相应的类型参数。

泛型参数命名举例

public interface ISessionChannel<TSession> { /\*...\*/ }

public delegate TOutput Converter<TInput, TOutput>(TInput from);

public class List<T> { /\*...\*/ }

# 类型参数的约束

