2.0 版 C# 语言和公共语言运行时 (CLR) 中增加了泛型。泛型将类型参数的概念引入 .NET Framework，类型参数使得设计如下类和方法成为可能：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，如下所示：

// Declare the generic class.

public class GenericList<T>

{

void Add(T input) { }

}

class TestGenericList

{

private class ExampleClass { }

static void Main()

{

// Declare a list of type int.

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

// Declare a list of type string.

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

// Declare a list of type ExampleClass.

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

}

}

# 泛型概述

* 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。
* .NET Framework 类库在 [System.Collections.Generic](https://msdn.microsoft.com/zh-cn/library/system.collections.generic.aspx) 命名空间中包含几个新的泛型集合类。应尽可能地使用这些类来代替普通的类，如 [System.Collections](https://msdn.microsoft.com/zh-cn/library/system.collections.aspx) 命名空间中的 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx)。
* 您可以创建自己的泛型接口、泛型类、泛型方法、泛型事件和泛型委托。
* 可以对泛型类进行约束以访问特定数据类型的方法。
* 关于泛型数据类型中使用的类型的信息可在运行时通过使用反射获取。

# 泛型的优点（C# 编程指南）

在公共语言运行时和 C# 语言的早期版本中，通用化是通过在类型与通用基类型 [Object](https://msdn.microsoft.com/zh-cn/library/system.object.aspx) 之间进行强制转换来实现的，泛型提供了针对这种限制的解决方案。通过创建泛型类，您可以创建一个在编译时类型安全的集合。

使用非泛型集合类的限制可以通过编写一小段程序来演示，该程序使用 .NET Framework 类库中的 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 集合类。 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 是一个使用起来非常方便的集合类，无需进行修改即可用来存储任何引用或值类型。

// The .NET Framework 1.1 way to create a list:

System.Collections.ArrayList list1 = new System.Collections.ArrayList();

list1.Add(3);

list1.Add(105);

System.Collections.ArrayList list2 = new System.Collections.ArrayList();

list2.Add("It is raining in Redmond.");

list2.Add("It is snowing in the mountains.");

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；因为 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 会将所有项都强制转换为 [Object](https://msdn.microsoft.com/zh-cn/library/system.object.aspx)，所以在编译时无法防止客户端代码执行类似如下的操作：

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;

}

尽管将字符串和 **ints** 组合在一个 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 中的做法在创建异类集合时是完全可接受的，并且有时需要有意为之，但这种做法很可能产生编程错误，并且直到运行时才能检测到此错误。

在 C# 语言的 1.0 和 1.1 版本中，只能通过编写自己的特定于类型的集合来避免 .NET Framework 基类库集合类中的通用代码的危险。当然，由于此类不可对多个数据类型重用，因此将丧失通用化的优点，并且您必须对要存储的每个类型重新编写该类。

[ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 和其他相似类真正需要的是：客户端代码基于每个实例指定这些类要使用的具体数据类型的方式。这样将不再需要向上强制转换为 **T:System.Object**，同时，也使得编译器可以进行类型检查。换句话说，[ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 需要一个类型参数。这正是泛型所能提供的。在 **N:System.Collections.Generic** 命名空间的泛型 [List<T>](https://msdn.microsoft.com/zh-cn/library/6sh2ey19.aspx) 集合中，向集合添加项的操作类似于以下形式：

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

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

// No boxing, no casting:

list1.Add(3);

// Compile-time error:

// list1.Add("It is raining in Redmond.");

对于客户端代码，与 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 相比，使用 [List<T>](https://msdn.microsoft.com/zh-cn/library/6sh2ey19.aspx) 时添加的唯一语法是声明和实例化中的类型参数。虽然这种方式稍微增加了编码的复杂性，但好处是您可以创建一个比 [ArrayList](https://msdn.microsoft.com/zh-cn/library/system.collections.arraylist.aspx) 更安全并且速度更快的列表，对于列表项是值类型的情况尤为如此。

# 泛型类型参数（C# 编程指南）

在泛型类型或方法定义中，类型参数是客户端在实例化泛型类型的变量时指定的特定类型的占位符。泛型类（如 [泛型介绍（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/0x6a29h6.aspx) 中列出的 GenericList<T>）不可以像这样使用，因为它实际上并不是一个类型，而更像是一个类型的蓝图。若要使用 GenericList<T>，客户端代码必须通过指定尖括号中的类型参数来声明和实例化构造类型。此特定类的类型参数可以是编译器识别的任何类型。可以创建任意数目的构造类型实例，每个实例使用不同的类型参数，如下所示：

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。通过这种替换方式，我们使用一个类定义创建了三个独立的类型安全的有效对象。有关 CLR 如何执行此替换的更多信息，请参见[运行时中的泛型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/f4a6ta2h.aspx)。

## 类型参数命名准则

* 务必使用描述性名称命名泛型类型参数，除非单个字母名称完全可以让人了解它表示的含义，而描述性名称不会有更多的意义。
* public interface ISessionChannel<TSession>; { /\*...\*/ }
* public delegate TOutput Converter<TInput, TOutput>(TInput from);
* public class List<T> { /\*...\*/ }
* 考虑使用 T 作为具有单个字母类型参数的类型的类型参数名。
* public int IComparer<T>() { return 0; }
* public delegate bool Predicate<T>(T item);
* public struct Nullable<T> where T : struct { /\*...\*/ }
* 务必将“T”作为描述性类型参数名的前缀。
* public interface ISessionChannel<TSession>
* {
* TSession Session { get; }
* }
* 考虑在参数名中指示对此类型参数的约束。例如，可以将带有 ISession 约束的参数命名为 TSession。

**相关章节**

有关更多信息：

* [泛型介绍（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/0x6a29h6.aspx)
* [泛型的优点（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/b5bx6xee.aspx)
* [泛型类型参数（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/0zk36dx2.aspx)
* [类型参数的约束（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/d5x73970.aspx)
* [泛型类（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/sz6zd40f.aspx)
* [泛型接口（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/kwtft8ak.aspx)
* [泛型方法（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/twcad0zb.aspx)
* [泛型委托（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/sx2bwtw7.aspx)
* [泛型代码中的默认关键字（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/xwth0h0d.aspx)
* [C++ 模板和 C# 泛型之间的区别（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/c6cyy67b.aspx)
* [泛型和反射（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173128.aspx)
* [运行时中的泛型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/f4a6ta2h.aspx)
* [.NET Framework 类库中的泛型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ac30fak4.aspx)

**请参阅**

[System.Collections.Generic](https://msdn.microsoft.com/zh-cn/library/system.collections.generic.aspx)

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

[类型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173104.aspx)

[<typeparam>（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173191.aspx)

[<typeparamref>（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173192.aspx)