**泛型方法（C# 编程指南）**

泛型方法是使用类型参数声明的方法，如下所示：

static void Swap<T>(ref T lhs, ref T rhs)

{

T temp;

temp = lhs;

lhs = rhs;

rhs = temp;

}

下面的代码示例演示一种使用 **int** 作为类型参数的方法调用方式：

public static void TestSwap()

{

int a = 1;

int b = 2;

Swap<int>(ref a, ref b);

System.Console.WriteLine(a + " " + b);

}

也可以省略类型参数，编译器将推断出该参数。下面对 Swap 的调用等效于前面的调用：

Swap(ref a, ref b);

相同的类型推理规则也适用于静态方法和实例方法。编译器能够根据传入的方法实参推断类型形参；它无法仅从约束或返回值推断类型形参。因此，类型推理不适用于没有参数的方法。类型推理在编译时、编译器尝试解析重载方法签名之前进行。编译器向共享相同名称的所有泛型方法应用类型推理逻辑。在重载解析步骤中，编译器仅包括类型推理取得成功的那些泛型方法。

在泛型类中，非泛型方法可以访问类级别类型参数，如下所示：

class SampleClass<T>

{

void Swap(ref T lhs, ref T rhs) { }

}

如果定义采用相同类型参数作为包含类的泛型方法，编译器将生成警告 CS0693，因为在方法范围内为内部 T 提供的参数隐藏了为外部 T 提供的参数。如果需要使用其他类型参数（而不是实例化类时提供的类型参数）来灵活地调用泛型类方法，请考虑为方法的类型参数提供另一个标识符，如下面示例的 GenericList2<T> 中所示。

class GenericList<T>

{

// CS0693

void SampleMethod<T>() { }

}

class GenericList2<T>

{

//No warning

void SampleMethod<U>() { }

}

使用约束对方法中的类型参数启用更专门的操作。此版本的 Swap<T> 现在名为 SwapIfGreater<T>，它只能与实现 [IComparable<T>](https://msdn.microsoft.com/zh-cn/library/4d7sx9hd.aspx) 的类型参数一起使用。

void SwapIfGreater<T>(ref T lhs, ref T rhs) where T : System.IComparable<T>

{

T temp;

if (lhs.CompareTo(rhs) > 0)

{

temp = lhs;

lhs = rhs;

rhs = temp;

}

}

泛型方法可以使用许多类型参数进行重载。例如，下列方法可以全部位于同一个类中：

void DoWork() { }

void DoWork<T>() { }

void DoWork<T, U>() { }

## 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)

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

[方法（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173114.aspx)