# 泛型 – 通用你的代码！



讨论三个主题！

## 泛型 是什么？

## 为什么用泛型？

## 泛型 怎么用？

# 定义

Generics是程序设计语言的一种技术，指将程序中数据类型进行参数化，它本质上是对程序的数据类型进行一次抽象，扩展语言的表达能力，同时支持更大粒度的代码复用。对于一些数据类型参数化的类和方法来说，它们往往具有更好的可读性、可复用性和可靠性。在设计集合类和它们的抽象操作时，往往需要将它们定义为与具体数据类型无关，在这种情况下，使用Generics就是非常适合的。举例来说，如果我们需要设计一个Stack类，有时需要元素为int类型的Stack，有时可能需要元素为Boolean或者Object类型的Stack。如果不使用Generics，我们通常需要定义不同的多个类，或者通过继承来实现。通过继承实现往往引起数据类型的转换问题，本文稍后对其进行分析。如果使用Generics技术，将Stack的元素类型进行参数化，那么Stack类的只需要实现一个版本，当需要某元素类型的Stack时，可以将类型作为参数来创建Stack对象。

泛型（Generics） 允许您延迟编写类或方法中的编程元素的数据类型的规范，直到实际在程序中使用它的时候。换句话说，泛型允许您编写一个可以与任何数据类型一起工作的类或方法。

您可以通过数据类型的替代参数编写类或方法的规范。当编译器遇到类的构造函数或方法的函数调用时，它会生成代码来处理指定的数据类型。**Blablabla……**



## 心之火可以燎原-思维抽象过程

本故事纯属虚构，如有雷同必属抄袭！demo1

|  |  |  |
| --- | --- | --- |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31： | (Long time ago)**Attention：** I have a C# v1.0！ |  |
|  | Great！I want to TRY！ | ： |
|  | I need Swap Int！ | ： |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31： | Swap(ref int lhs, ref int rhs)，here you are! |  |
|  | 牛逼！，I need Swap String！ | ： |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31： | Swap(ref string lhs, ref string rhs),done! |  |
|  | I need Swap xxx! | ： |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31: | ……稍等~（I am Thinking！）  1.数学意义上的函数：**y=f(x)，其中x叫变量；**  2.C#语言中的函数/方法(Method)：  public static void Swap(ref int lhs, ref int rhs)  {  var temp = lhs;  lhs = rhs;  rhs = temp;  }；  其中，**lhs，rhs**叫方法Swap的**参数**；  ！！！！为什么不把**Int**也给参数化了？！！！！  public static void Swap<T>(ref T lhs, ref T rhs)  {  var temp = lhs;  lhs = rhs;  rhs = temp;  }  其中，T叫**类型参数。**  @E，here you are!!! |  |



## 什么是 泛型？

-- 把**类型**当作**参数**传递给被调对象的**思想**（包括其代码实现），统统叫**泛型（泛化）**；

Swap<**T**>(ref **T** lhs, ref **T** rhs)中的**T**（Template？Type？）叫做**类型参数**（一下统称Type或T）；

冯·诺依曼之所以牛逼，不仅仅是因为他制定了“计算机工作原理”，而是因为他直截了当的指明了现代计算机架构的极限边界；

就像“宇宙绝对零度”（0k，~ -273.15℃），自然界的温度只能无限逼近，但绝不会比它更低；

以上概念在任何实现泛型的语言上均通用，如 Java、Swift、VB.NET、F#、TypeScript、C++（模板）；

## 一生二，二生三，三生万物！

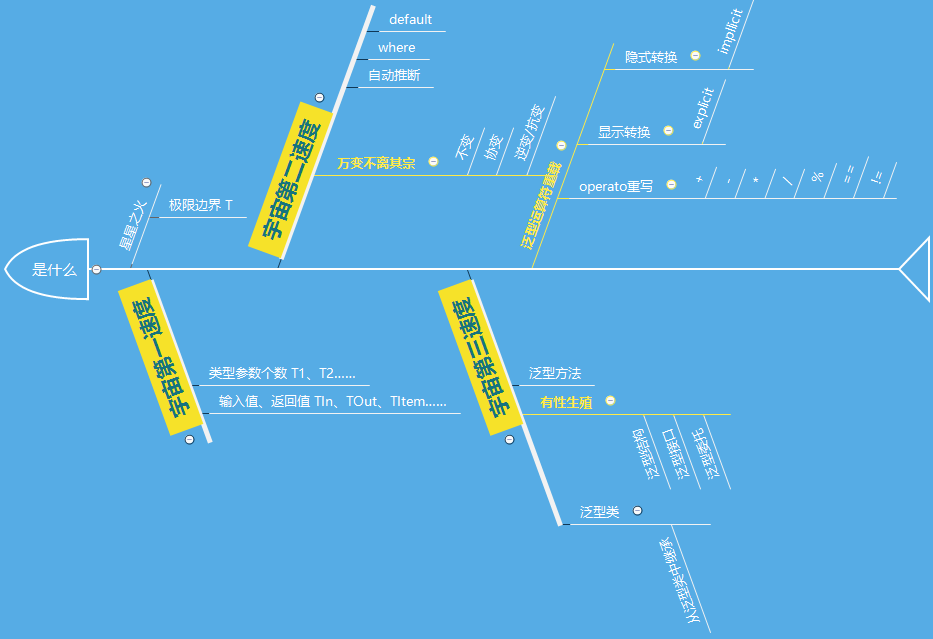
从泛型方法到泛型XX的演化史-从C到C++，再到C#（++++）；

**面向过程 的 编程 = 数据结构 + 算法；**

OOP？

**面向对象 的 编程 = 面向过程 + 抽象；**

**但**，一切的**目标**只有一个：**偷懒**！（**偷懒、偷懒、再偷懒……**）



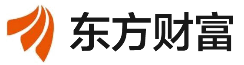
### 一生二

超越不了宇宙第一速度V1（v = 7.9 Km/s），你只能在地球上玩！

|  |  |  |
| --- | --- | --- |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31: | ……修炼中~（I am still Thinking！）  1.C#语言中的函数/方法(Method)，可以有多个参数，如：  public static void Swap(ref int lhs, ref int rhs)中的**lhs，rhs**；  2.！！！！泛型中为什么不把**类型参数**也给多搞几个？！！！！    3.！！！输入部分可以有参数类型T，输出也可以有T（TOut）！！！ |  |
|  | **Attention：** I have a C# v2.0！ |  |

使用泛型是一种增强程序功能的技术。泛型是c#2.0的一个新增加的特性，它为使用c#语言编写面向对象程序增加了极大的效力和灵活性。不会强行对值类型进行装箱和拆箱，或对引用类型进行向下强制类型转换，所以性能得到提高。通过知道使用泛型定义的变量的类型限制，编译器可以在一个高得多的程度上验证类型假设，所以泛型提高了程序的类型安全。它允许程序员将一个实际的数据类型的规约延迟至泛型的实例被创建时才确定。泛型为开发者提供了一种高性能的编程方式，能够提高代码的重用性，并允许开发者编写非常优雅的解决方案。

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。.NET Framework 2.0 版类库提供一个新的命名空间 System.Collections.Generic，其中包含几个新的基于泛型的集合类。建议面向 2.0 版的所有应用程序都使用新的泛型集合类，而不要使用旧的非泛型集合类，如 ArrayList。有关更多信息，请参见 .NET Framework 类库中的泛型（C# 编程指南）。当然，也可以创建自定义泛型类型和方法，以提供自己的通用解决方案，设计类型安全的高效模式。**Blablabla……**

**、****、……统统点赞！！！**

### 二生三

超越了宇宙第二速度V2（v = 11.2Km/s），你就可以去看太阳！

|  |  |  |  |
| --- | --- | --- | --- |
|  | 我弱弱的问个问题：看demo2，**T**怎么**初始化**？    **T**怎么**比大小**？ | ： | |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31: | ……再次修炼中~（I am Rethinking！）  **1.T是引用类型还是值类型。若不知道这个情况，就不能用null赋值。**  ！！！switch中的**default**，默认值（引用类型给null，值类型给默认值，数字类型给0，结构struct根据其各个成员的类型，以相同的方式初始化为0或null）！！！  2.因为T不知道什么类型，所以不能比较大小，为什么不能限定T必须可比较大小？  ！！！Sql 中的**where**加起来！！！  3.外面**传入**的参数已经带有了类型，为什么不自动推断类型？  **去掉<T>**  4.都能自动推断类型了，为什么不能“上下”扩展？  **不变->逆变/抗变 | 协变**，万变不离其宗！in ＆ out 加起来  **Attention：** I have a C# v2.0 Sp1！ | |  |

#### default关键字

对**未知T类型**的变量做初始化！

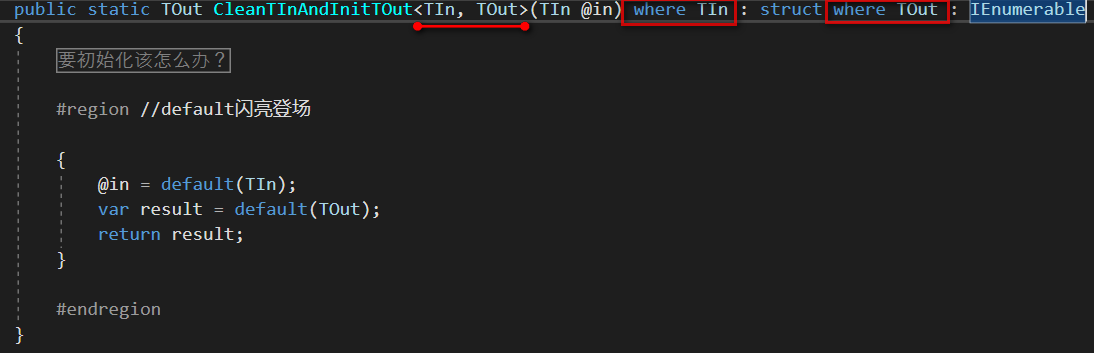
不能把null赋予泛型类型。原因是泛型类型也可以实例化为值类型，而null只能用于引用类型。为了解决这个问题，可以使用default关键字。通过default关键字，将null赋予引用类型，将0赋予值类型。（default关键字根据上下文可以有多种含义。Switch语句使用default定义默认情况。在泛型中，取决于泛型类型是引用类型还是值类型，泛型default将泛型类型初始化为null或0）

#### 约束类型

之前的泛型中，**参数类型**称为**无绑定**（unbounded）类型，因为没有对它们进行任何约束。而通过约束类型，可以把类型限制为用于实例化的泛型**参数类型**。**赋能OR限定**

where**定制未知T类型**的**部分**功能！

还能多个**where**语句一起使用：



允许对个别泛型的类型参数进行约束，包括以下几种形式（假设 C是泛型的类型参数， 是一般类、泛类，或是泛型的类型参数）：T 是一个类。T 是一个值类型。T 具有无参数的公有建构方法。T 实现接口 I 。T 是 C ，或继承自 C 。

|  |  |  |  |
| --- | --- | --- | --- |
| **分类** | **约束** | **定义** | **示例** |
| 主要约束  （0或1个） | struct | 类型必须是值类型 | 需要值类型才能起作用的时候，例如，变量是0,表示某种含义限，要做运算+,-,x,/等 |
| class | 类型必须是引用类型 | 需要引用类型才能起作用的时候，如变量是null，表示某种含义 |
| 次要约束  【裸类型约束】  （0或1个） | <baseclass> | 类型必须是基类**或**继承自基类 可以给这个约束提供任意类名 | 需要继承自基类的某种基本功能，才能起作用 |
| 次要约束  （0或多个） | <interface> | 类型必须是接口**或**实现了接口 | 需要接口提供的某种基本功能，才能起作用 |
| 构造器约束 | new() | 类型必须有一个**公共的无参构造函数** | 需要能实例化T类型的变量，例如在构造函数中实例化 |
| 其他验证性约束  （0或1个） | where T1：T2 | 这个约束也可以指定，类型T1派生自泛型类型T2； | |

如果泛型类需要调用泛型类型中的方法，就必须添加约束。（给泛型类型添加约束时，最好包含泛型参数名称的一些信息。对于编译器而言，参数名不重要，但更具可读性。）

约束：说明

Where T：struct 对于结构约束，类型T必须是值类型

Where T：class 类约束指定类型T必须是引用类型

只能为默认构造函数定义构造函数约束，不能为其他构造函数定义构造函数约束。

使用泛型类型还可以合并多个约束。Where T:IFoo，new（）约束和MyClass<T>声明指定，类型T必须实现IFoo接口，且必须有一个默认构造函数。

（注意：在C#中，where子句的一个重要限制是，不能定义必须由泛型类型实现的运算符。运算符不能在接口中定义。在where子句中，只能定义基类、接口和默认构造函数）。

#### 不变，协变，逆变/抗变，万变不离其宗

在.NET4之前，泛型接口是不变的。.NET4通过协变和抗变为泛型接口和泛型委托添加了一个重要的扩展。协变和抗变指对参数和返回值的类型进行转换。在.NET中，参数类型是**协变**的。方法的返回值是**抗变**的。**隐式转换**

Demo2.3

假定有Shape和Rectangle两个类，Rectangle派生自Shape基类，声明Display()方法接收Shape类型作为参数：

public static void Display(Shape o){}

现在可以传递派生自Shape基类的任意对象。因为Rectangle派生自Shape，所以Rectangle满足Shape的所有要求。

var r = new Rectangle() {Height = 2.5, Width = 5};

Display(r);//in 逆变/抗变

当方法返回一个Shape时，不能把它赋值给Rectangle，因为Shape不一定总是Rectangle，反过来却可以。如果一个方法比如GetRectangle（）返回一个Rectangle：

public static Rectangle GetRectangle(){return new Rectangle();}

就可以把结果赋予某个Shape：

Shape s = GetRectangle();//协变 out

在搞清楚“三变”前，先搞清楚“基类-派生类”之间的一些概念。

##### 向上类型转换and向下类型转换

由低层次类型转换为高层次类型称为**向上类型转换**。向上类型转换是自动进行的，比如把int型变量赋给为long型变量，把long型变量赋给double型变量，转换都是**自动进行**的。由派生类转换为基类也是**向上**提升，也是自动进行的，但转换后，基类的引用符**不能**引用派生类对象**特有的函数（和属性）**。

反之为**向下类型转换。**



##### 不变量（invariant）

意味着泛型类型参数不能更改。

##### 逆变（抗变）量（contravariant）

意味着泛型类型参数可以从一个类更改为他的某个派生类。在C#是用**in关键字**标记逆变量形式的泛型类型参数。逆变量泛型类型参数只能出现在**输入位置**，比如作为方法的参数。

##### 协变量（covariant）

意味着泛型类型参数可以从一个类更改为它的某个基类。C#使用**out关键字**标记协变量形式的泛型类型参数。协变量泛型类型参数只能出现在**输出位置**，比如方法的返回类型。（区别方法中的out）

比如：

public delegate TResult Func<in T, out TResult>(T arg);

其中，T用in标记，这使它成为逆变量，TResult用out标记，这使它成为协变量。

如果声明一个变量：

Func<object, ArgumentException> fn1 = null;

就可以将它转型成为另一个类型参数不同的Func类型：

Func<String, Exception> fn2 = fn1

String从Object派生;Exception是ArgumentException的基类。

### 三生万物

超越了宇宙第三速度V3（v = 16.7Km/s），你才有机会去看外星人！

#### 有性生殖-物种进化的新篇章

|  |  |  |
| --- | --- | --- |
| https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1Mu3b?ver=5c31: | ……再再次修炼中~（I am still Rethinking！）  1.！！！之前只给**方法加持**了“泛型”（泛型方法），我不是还有类、接口、委托、数组……吗？！！！  于是乎：泛型方法、泛型类、泛型接口、泛型委托……**孕育而生**，且以上default、约束等等均适用！  **Attention：** I have a C# v2.0 Sp2！ |  |

接口，结构，委托 也可以同泛型的方式创建；

泛型是具有占位符（类型参数）的类、结构、接口和方法，这些占位符是类、结构、接口和方法所存储或使用的一个或多个类型的占位符。**泛型集合类**可以将类型参数用作它所存储的对象的类型的占位符；类型参数**作为**其字段的类型及其方法的参数类型**出现**。泛型方法可以将其类型参数用作其返回值的类型或者其某个形参的类型。

#### 泛型方法

泛型引入了一种新的编程方式，如何实现相应的算法（尤其是操作和谓词），以用于不同的类，而且它们都是类型安全的。

泛型委托可以去除集合中的算法。我们不必给不同的类型编写功能相同的许多方法或类，只创建一个方法或类即可。另外，使用 GetType 方法可在程序运行时得知泛型及其类型参数的实际类型，更可以运用反射编程。它是独立于类型的方法

公共语言运行时和 C# 语言早期版本中存在一个局限，其中通过将类型与通用基类型 Object 相互强制完成泛化，泛型提供了此局限的解决方案。 通过创建泛型类，可在编译时创建类型安全的集合。另一个减少代码的选项是使用Object类，但使用派生自Object类的类型进行传递不是类型安全的。泛型类使用泛型类型，并可以根据需要用特定的类型替换泛型类型。这就保证了类型安全性；

#### 泛型类

通过泛型类可以创建独立于类型的类；

在创建泛型类时，还需要一些其他C#关键字。例如，不能把null赋予泛型类型。此时，可以使用default关键字。如果泛型类型不需要Object类的功能，但需要调用泛型类上的某些特定方法，就可以定义约束。

##### 从泛型类中继承

1. 如果某个类型在它所继承的基类型中受到了约束，该类型就不能“解除约束”。也就是说，类型T在所继承的基类型中使用时，该类型必须受到至少与基类型相同的约束。



1. 如果继承了一个泛型类型，就必须提供所有必须的类型信息，这可以使用其他泛型类型参数的形式来提供，如上所述，也可以显式提供。这也适用于继承了泛型类型的非泛型类。



**兜底条款**

泛型类型可以实现泛型接口，也可以派生自一个类。泛型类可以派生自泛型基类。

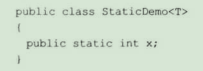
其要求是必须重复接口的泛型类型，或者必须指定基类的类型。

于是，派生类可以是泛型类或非泛型类。例如，可以定义一个抽象的泛型基类，他在派生类中用一个具体的类实现。这允许对特定类型执行特殊的操作。

还可以创建一个部分的特殊操作，如从Query中派生StringQuery类，只定义一个泛型参数，如字符串TResult。要实例化StringQuery，只需要提供TRequest的类型。

###### 静态成员

泛型类的静态成员需要特别关注。泛型类的静态成员只能在类的一个实例中共享。

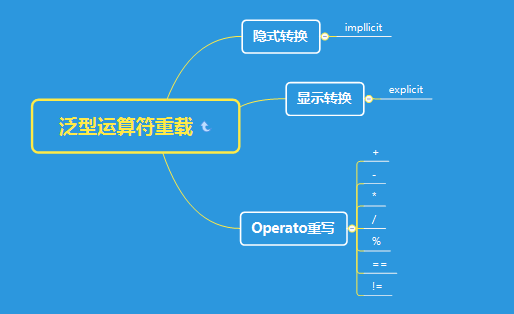


由于同时对一个string类型和一个int类型使用了StaticDemo<T>类，所以存在两组静态字段：

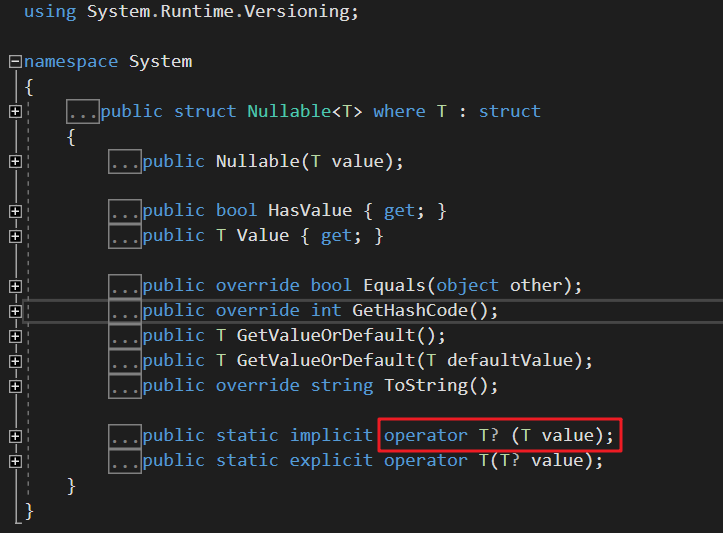


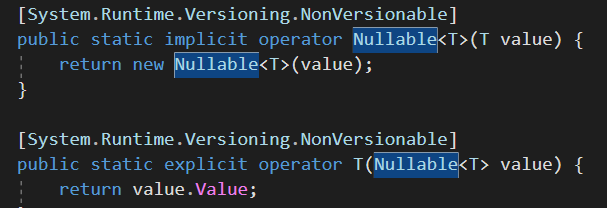
##### 泛型运算符

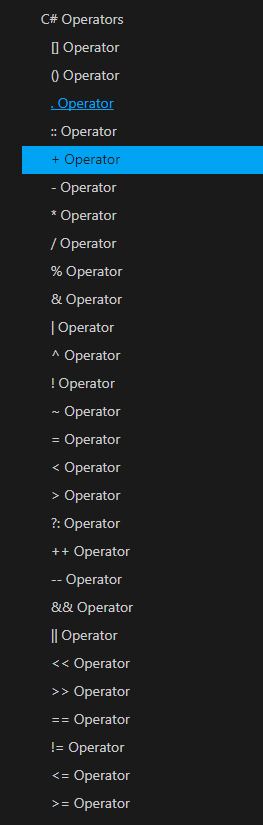
在C#中，可以像其他方法一样进行运算符的重写，这也可以在泛型类中实现。



<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/>







##### 对象的相等

初次看到Object类的成员函数你会感到很惊讶，竟然有三个“相等”函数，如果再加上相等运算符“==”的话，就有4种进行相等比较的方式了。详见EqualsTest

###### ReferenceEquals()函数

ReferenceEquals()函数是一个静态函数，用来测试两个引用符是否指向同一个对象，即两个引用符是否包含相同的内存地址。如果两个引用符指向同一个对象，函数返回true，否则返回false。如果两个引用符的值都是null，也返回true,这种比较称为引用比较。

###### 实例Equals()函数

在Object类中，实例版Equals()函数是一个虚函数，默认状态下只进行引用比较。但我们可以在派生类中重写该函数，实现值比较。如果两个对象是值类型，则判断两个值是否相等；如果两个对象是引用型，则判断两个对象的数据成员是否相等。

###### 静态Equals()函数

静态版Equals()函数的功能与实例版基本一样，实际上静态版Equals()函数通过调用实例版函数进行比较，只是在调用"objA.Equals(objB)”前，会先检查两个参数是否均为空引用，如果均为空引用，返回true，如果一个为空、一个不为空，返回false.

###### 相等运算符==

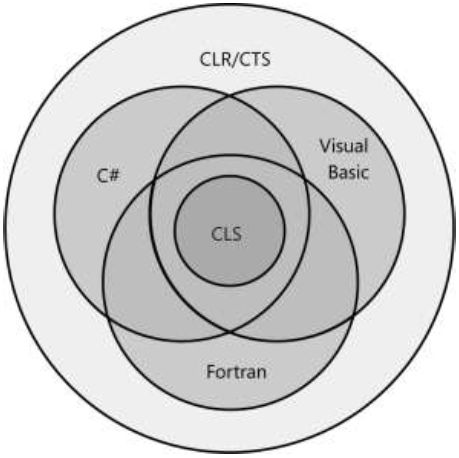
这是我们很熟悉的运算符。默认状态下，若两个对象为值类型，相等运算符“==”比较两个对象的值；若两个对象为引用类型，相等运算符“==”比较两个引用符。但我们可以通过重载运算符的方法加以改变。

通用类型系统CTS（Common Type System）中的基本数据类型：

|  |  |  |  |
| --- | --- | --- | --- |
| **CTS类型** | **C#名称** | **C++名称** | **说明** |
| System.Sbyte | sbyte | char | 8位有符号整数 |
| System.Byte | byte |  | 8位无符号整数 |
| System.Short | short | (signed) short | 16位有符弓整数 |
| System.Ushort | ushort | unsigned short | 16位无符号整数 |
| System.lnt32 | int | (signed) int | 32位有符号整数 |
| System.Uint32 | uint | unsigned int | 32位无符号整数 |
| System.Int64 | long | (signed) long | 64位有符号整数 |
| System.Uint64 | ulong | unsigned long | 64位尢符号幣数 |
| System.Single | float | float | 32位单精度实数 |
| System.Double | double | double | 64位双精度实数 |
| System.Decimal | decimal |  | 高精度实数 |
| System.Boolean | bool | bool | 布尔型 |
| System.Char | char | wchar\_t | 字符型 |
| System.String | string |  | 字符串 |
| System.Object | object |  | 所有CTS类型的基类 |



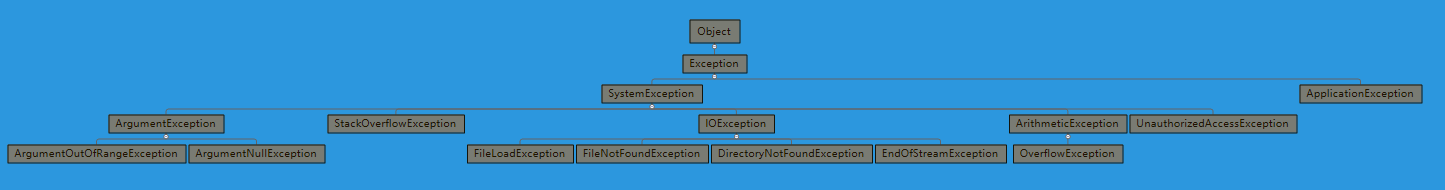
公共语言规范（Common Language pecification，CLS），这是所有.NET语言都支持的语言规范。

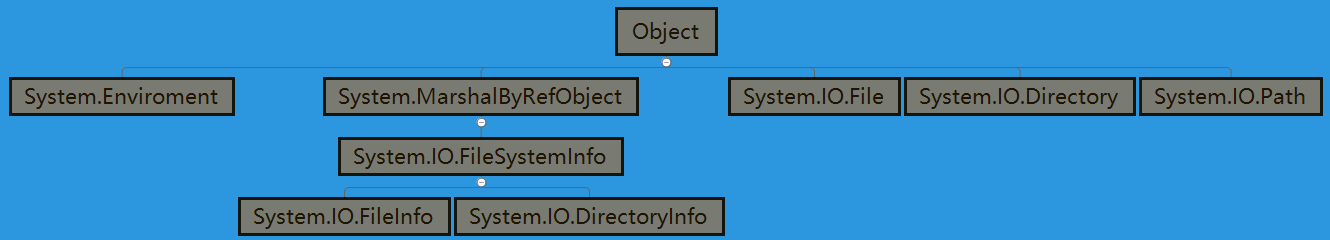


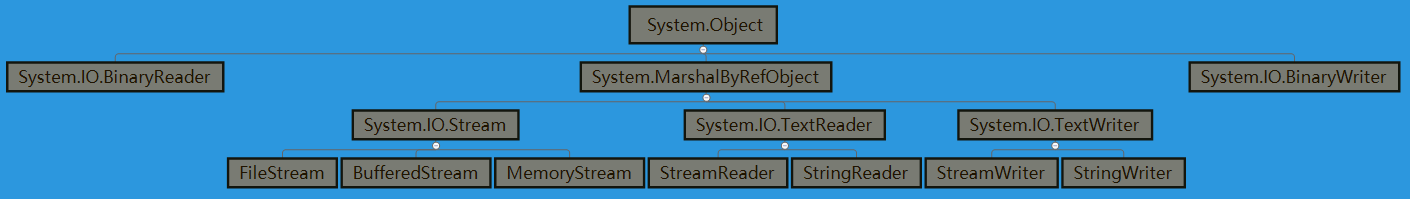
通俗地说就是公共语言规范中的语法规则在所有.NET语言中都成立，它是所有.NET语言都支持的语法规则的交集。如此一来，符合CLS规范的代码转换为中间语言后就可以被.NET上任何语言访问，从而确保不同语言的互操作性。

编写不符合CLS规范的代码是完全可以的，但这时就不能保证**不同语言间的互操作性**。所以在编写用于共享的类时，我们一般只用完全兼容CLS的代码，以增强类的交互性；在私有类中，可以编写非CLS代码，因为私有类不用来交互。C#中不兼容CLS的特性非常少。

|  |  |
| --- | --- |
| **命名空间** | **说明** |
| System | 包含大量常用类 |
| System.Windows.Forms | 包含大量Windows控件类 |
| System.lO | 包含用于输入/输出、读写文件的类 |
| System.Data | 包含用于访问与操纵数据库的类 |
| System.Web | 包含用于创建与维护Web程序的类 |
| System.Xrnl | 包含用创建与维护XML数据的类 |
| System.Collections | 包含用于定义集合的类和接口 |
| System.Collections.Generic | 包含用于定义范型的类和接口 |
| System.Net | 包含为网络协议提供编程接口的类 |
| System.Text | 包含处理字符编码的类和接口 |
| System.Threading | 包含管理进程的类和接口 |
| System.Drawing | 包含具有基本图形功能的类 |







#### 泛型结构

与类相同，只是有些微小区别：结构是值类型，不是引用类型。

#### 泛型数组

见Demo2.1

#### 泛型接口

泛型并不限于类，还可以创建泛型接口，泛型方法(可以在非泛型类上定义)，甚至泛型委托。

其继承规则与类相同。如果继承了一个基泛型接口，就必须遵循“保持基接口泛型类型参数的约束”等规则。

使用泛型可以定义接口，在接口中定义的方法可以带泛型参数。.NET为不同的情况提供了许多泛型接口，同一个接口常常存在比较老的非泛型版本。

#### 泛型委托

delegate T NumberChanger<T>(T n); 见Demo2.2

public delegate TOut MyDelegate<out TOut, in TIn>(TIn op1, TIn op2) where TIn : TOut;

#### 命名约定

如果在程序中使用泛型，在区分泛型类和非泛型类时就会有一定的帮助。下面是泛型类型的命名规则：

泛型类型的名称用字母T做前缀

如果没有特殊的要求，泛型类型允许用任意类型替代，且只使用了一个泛型类型，就可以用字符T作为泛型类型的名称。

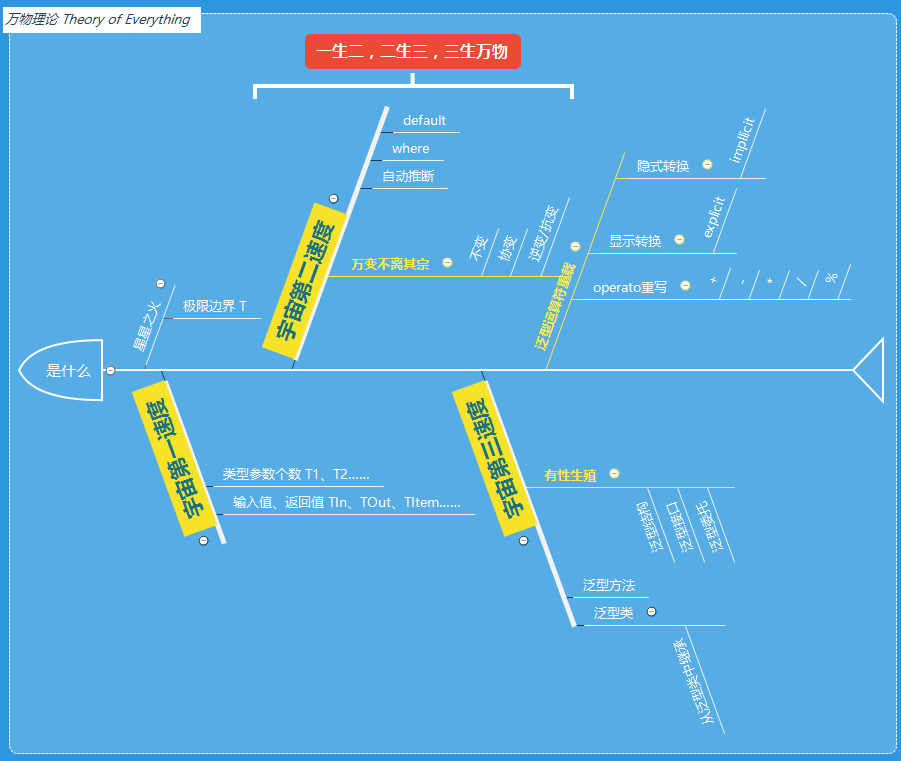
Public Class List<T>{ }

Public Class LinkedList<T>{ }

如果泛型类型有特定的要求（例如，它必须实现一个接口或派生自基类），或者使用了两个或多个泛型类型，就应给泛型类型使用描述性的名称：

Public delegate void EventHandler<TEventArgs>(object sender,TEventArgs e)

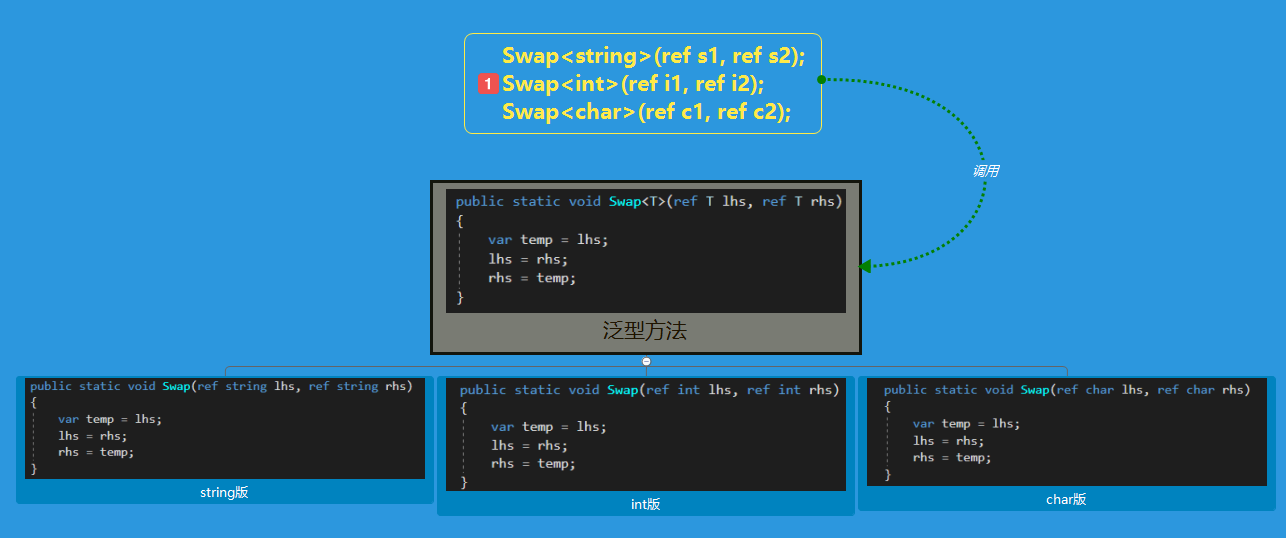
Public Class SortedList<TKey,Tvalue>{}



# 优势

## 与C++模板的对比

C++模板和.NET泛型之间有一个很大的区别。对于C++模板，在用特定的类型实例化模板时，需要模板的源代码。



通常，C++在创建类时，它会编译为一个具体的类型，然后在代码中使用。C++在创建泛型类时，会编译为许多类型，然后进行实例化。在.NET中，类有无限多个。在后台，.NET运行库允许在需要时**动态生成泛型类**。在通过实例化来请求生成之前，B的某个泛型类A甚至不存在。

注意：这是C++模板和C#泛型类的一个区别，在C++中，编译器可以检测出在哪里使用了模板的某个特定类型，例如，模板B的A类型，然后编译需要的代码，来创建这个类型。而在C#中，所有的操作都在运行期间进行。

## 源代码保护

泛型不仅是C#编程语言的一部分，而且与程序集中的IL（Intermediate Language，中间语言）代码紧密地集成。泛型不仅是C#的一种结构，而且是CLR（Common Language Runtime）定义的。所以可以跨语言，如在Visual Basic中用一个特定的类型实例化该泛型；

## 类型安全

如果使用对象，就可以在这个集合中添加任意类型。并不是集合中的所有元素都可以强制转换为int，所以会出现一个运行时异常；

它有助于您最大限度地重用代码、保护类型的安全以及提高性能。

您可以创建泛型集合类。.NET 框架类库在 System.Collections.Generic 命名空间中包含了一些新的泛型集合类。您可以使用这些泛型集合类来替代 System.Collections 中的集合类。

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。

您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。

## 更清晰的代码

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。

在泛型数据类型中所用类型的信息可在运行时通过使用反射来获取。

### 二进制代码的重用

泛型类可以定义一次，并且可以用许多不同的类型实例化。不需要像C++模板那样访问源代码。

泛型类型可以在一种语言中定义，在任何其他.NET语言中使用。

### 代码的扩展

在用不同的特定类型实例化泛型时，会创建多少代码？因为泛型类的定义会放在程序集中，所以用特定类型实例化泛型类不会在IL代码中复制这些类。但是在JIT编译器把泛型类型编译为本地代码时，会给每个值类型创建一个新类。引用类型共享同一个本地类的所有相同的实现代码。这是因为引用类型在实例化的泛型类中只需要4个字节的内存地址（32位操作系统），就可以引用一个引用类型。

值类型包含在实例化的泛型类的内存中，同时因为每个值类型对内存的要求都不同，所以要为每个值类型实例化一个新类。

## 更佳的性能

性能-由于.NET Framework 泛型的类型参数之实际类型在运行时均不会被消除，运行速度会因为类型转换的次数减少而加快。

C# 语言和公共语言运行时 (CLR) 的 2.0 版本中添加了泛型。 泛型将类型参数的**概念**引入 .NET Framework，这样就可以设计具有以下特征的类和方法：在客户端代码声明并初始化这些类和方法之前，这些类和方法会延迟指定一个或多个类型。 例如，通过使用泛型类型参数 T，可以编写其他客户端代码能够使用的单个类，而不会产生运行时转换或装箱操作的成本或风险。

System.Collections和System.Collections.Generic 泛型和非泛型集合类。对值类型使用非泛型集合类，在把值类型转换为引用类型，和把引用类型转换为值类型时，需要进行装箱和拆箱操作。装箱和拆箱操作很容易使用，但性能损失比较大，遍历许多项时尤其如此。

## 微软VS巨硬

# 实践

## 微软泛型命名空间

每个线程都有自己的托管栈（stack）-线程托管栈。

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

System名称空间包含．NET应用程序使用的大多数基本类型。

System.Text名称空间包含与字符串处理和编码相关的类型，

System.Linq名称空间为Linq提供支持。

但**System.Collections.Genenc**名称空间是个**什么鬼**?为什么要在**默认情况**下把它**包含**在控制台应用程序中?

这个名称空间包含用于**处理集合的泛型类型**，使用得非常频繁，用using语句配置它，使用起来就不必添加限定符了。它们可以使工作更容易完成，可以亳不费力地**创建强类型化的集合类**。

|  |  |
| --- | --- |
| List<T> | T类型对象集合 |
| Dictionary<TKey, TValue> | 与K类型的键值相关的V类型的项的集合 |

使用泛型类型可以最大限度地重用代码、保护类型安全性以及提高性能。

泛型最常见的用途是创建集合类。.NET Framework 类库在 System.Collections.Generic 命名空间中包含几个新的泛型集合类。应尽可能使用这些类来代替某些类，如 System.Collections 命名空间中的 ArrayList，参考FCL（Framework Class Library）.NET框架基础类库：

<https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic?view=netframework-4.7>

## 色即是空，空即是色-四大皆空的可空类型

值类型（大多数基本类型，例如int，double和所有结构）区别于引用类型（string和所有的类）的一种方式是值类型**必须**包含一个值，它们可以在声明之后、赋值之前，在未赋值的状态下存在，但不能以任何方式使用。而引用类型可以是**null**。

有时**让值类型为空**是很有用的(尤其是处理数据库时)，泛型使用System.Nullable<T>类型提供了使值类型为空的一种方式。demo3

System.Nullabel<int> nullableInt；

### ?、??、运算符和可空类型的由来

#### 特殊的bool？

|  |  |  |  |
| --- | --- | --- | --- |
| op1 | op2 | op1 & op2 | op1 | op2 |
| true | true | true | true |
| true | false | false | true |
| true | null | null | true |
| false | true | false | true |
| false | false | false | false |
| false | null | false | null |
| null | true | null | true |
| null | false | false | null |
| null | null | null | null |

#### ??（空接合运算符）

为了进一步减少处理可空类型所需的代码量，使可空变量的处理变得更简单，可以使用??运算符。这个运算符称为空接合运算符（null coalescing operator)，是一个二元运算符，允许给可能等于null的表达式提供另一个值。如果第一个操作数不是null，该运算符就等于第一个操作数，否则，该运算符就等于第二个操作数。在功能上，下面的两个表达式是相同的：

double? op1 = 1, op2 = 2;

//double? op1 = null, op2 = 2;

var result = op1 ?? op2; //与下一句等价

//var result = op1 == null ? op1 : op2;

在上述代码中，op1可以是任意可空表达式，包括引用类型和可空类型。因此如果可空类型是null，就可以使用??运算符提供要使用的默认值。

int? op1 = null;

int result = op1 \* 2 ?? 5;

//int? result = op1 \* 2 ?? 5;

该示例中，op1是null，所以op1\*2也是null。但是??运算符检测到这个情况，并把值5赋予result.这里特别要注意，在结果中放入int类型的变量result不需要显式转换。??运算符会自动处理这个转换。可以把??等式的结果放在int?中。

在处理可空变量时，??运算符有许多用途，它也是提供默认值的一种便捷方式，不需要使用if结构中的代码块或容易混淆的三元运算符。