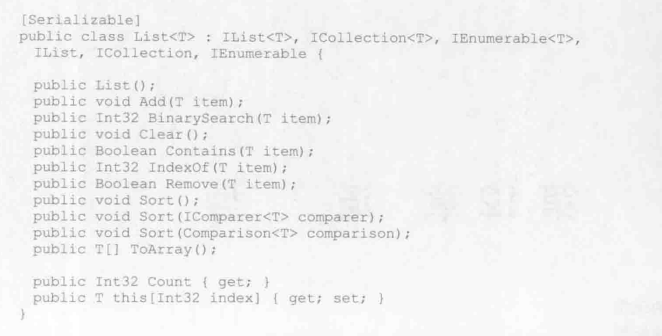
泛型的好处在于“代码重用”

CLR允许在引用类型、值类型、接口类型中定义泛型方法。CLR允许创建泛型引用类型和泛型值类型，还允许创建泛型委托和泛型接口，但不允许创建泛型枚举类型。

T作为一个未指定的数据类型，可以加在类后，加在尖括号中比如List<T>。定义泛型类型或方法时，为类型指定的任何变量（在这里是T）都称为“类型参数”。

下面这个例子：



T作为接口和类的类型参数、方法的形参类型、方法的返回值的类型。

使用泛型类型或者方法时指定的具体数据类型称为“类型实参”。

使用泛型的优点：

1.源代码保护。使用泛型算法的开发人员不需要访问算法的源代码。

Ps：因为C#的泛型是运行时实例化，所以可以在不提供源码的同时提供给用户；而C++的模板类是编译时实例化，所以必须在编译前修改模板参数来生成特定类型，因此必须把源码提供给用户。

2.保证类型安全。泛型可以在编译时检查类型，从而避免了在运行时出现类型错误的情况。这可以提高代码的可靠性和可维护性。

3.减少强转次数。因为泛型保证类型安全，所以编译器会保证类型安全性，所以能避免强制转换，因此代码的可读性更好。

4.避免装箱。减少垃圾回收的次数。

开放类型和封闭类型

具有泛型类型参数的类称为“开放类型”；为所有类型参数都传递了实际的数据类型，该类型即“封闭类型”。CLR禁止开放类型构造实例，CLR允许构造封闭类型实例。

Ps：List<T>中定义的任何静态字段不会在它的封闭类型间共享（例如不会在List<String>和List<DateTime>间共享），每个封闭类型都有自己的静态字段。如果泛型类型（开放类型）定义了静态构造器，那么针对它的每个封闭类型都会执行一次静态构造器（例如P240的定义处理枚举类型的泛型类型，并且泛型类传入枚举类型只能通过该方式确保类型安全）.

如何通过泛型类型和继承来确保类型安全并防止值类型装箱？

指定类型实参不会影响继承层级结构。为泛型类指定类型实参时，它不会改变该类的继承关系。例如有一个名为MyBaseClass的基类和一个名为MyDerivedClass<T>的派生类，无论为T指定什么类型实参，MyDerivedClass<T>都将继续从MyBaseClass派生。

P241页例子：

文本

描述已自动生成

定义一个封闭类型基类再定义它的开放类型子类

不应该定义一个类型作为泛型类型的子类，这会破坏类型的安全性

例子（P242）：



sameType的结果为false。如果DateTime为List<DateTime>的子类，那么可以用List<DateTime>类型接收一个DateTime的变量（因为子类可以装箱成父类），这样会导致混淆，另外如果在子类中重写了一些方法会破坏安全性。

上面例子中如果是单纯为了使得代码易读而构造一个子类继承List<DateTime>类型，那么可以采用using指令定义一个名称给它赋值对应的类型，例子如下：



如果这么做，

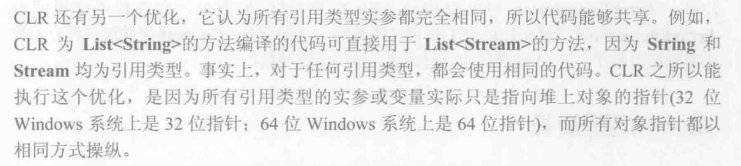


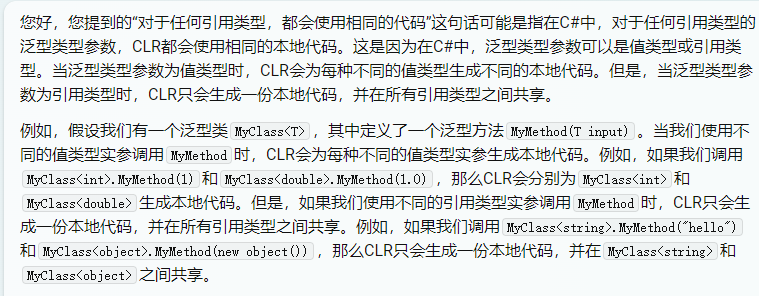
这行代码的sameType为True

可以用隐式局部变量var推测T的实际类型

代码爆炸现象：代码爆炸是指当使用泛型类型参数的方法进行JIT编译时，CLR获取方法的IL，用指定的类型实参进行替换，然后创建恰当的本地代码。然而，这样做有一个缺点：CLR要为每种不同的方法/类型组合生成本地代码。这个现象被称为“代码爆炸”，它可能造成引用程序集的显著增大，从而影响性能。

CLR是如何解决代码爆炸的问题的：CLR认为所有引用类型实参完全相同（所以生成的本机代码是相同的）。s





（如何查看生成的本机代码？）

如果泛型类型的类型实参为值类型，CLR就必须专门为对应的值类型生成本机代码（因为值类型大小不定且可能用不同的本机CPU指令操纵这些值）

泛型接口

如何定义一个泛型接口



封闭类型实现泛型接口

图片包含 文本

描述已自动生成

开放类型实现泛型接口

图片包含 文本

描述已自动生成

Ps：ArrayEnumerator可以枚举一组T对象

泛型委托

如何定义泛型委托

文本

描述已自动生成

逆变和协变

协变：协变允许您将一个泛型类型的实例赋值给另一个泛型类型，只要它们的类型实参具有继承关系。（实例是子类）

逆变：允许将一个泛型类型的实例赋值给另一个泛型类型，只要它们的类型实参具有相反的继承关系。（实例是父类）

逆变用于传入的参数值，协变用于返回值（原理是里氏替换原则）

逆变用于输入参数，因为它允许您使用更具体的类型来替换方法签名中输入参数的更一般类型。例如，假设您有一个接受 Animal 类型参数的方法，您可以使用 Cat 类型（Animal 的子类）来替换该参数，因为 Cat 是 Animal 的一种更具体的类型。

协变用于返回值，因为它允许您使用更一般的类型来替换方法签名中返回值的更具体类型。例如，假设您有一个返回 Cat 类型值的方法，您可以使用 Animal 类型（Cat 的基类）来替换该返回值，因为 Animal 是 Cat 的一种更一般的类型。

协变的例子：

协变的关键字为out，加在尖括号内的T之前



逆变的例子：

协变的关键字为in，加在尖括号内的T之前

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Ps：值类型不具有逆变和协变，因为值类型需要装箱

文本

描述已自动生成

泛型方法

如何定义泛型方法

图片包含 文本

描述已自动生成

泛型参数可以定义返回值，参数值，方法名之后的括号内是方法的类型参数

Ps：在同时存在非泛型方法和泛型方法时，编译器优先考虑非泛型方法

静态泛型方法必须适用于当前存在的所有类型，即T可以是当前的所有类型

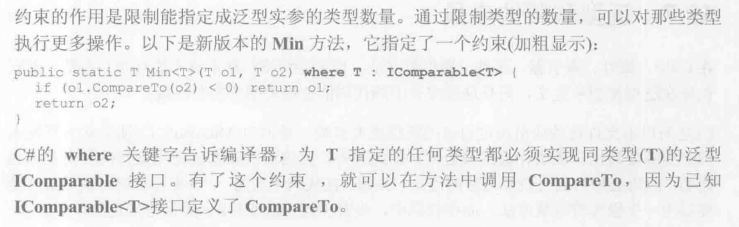
报纸上的文字

描述已自动生成

通过上面的第二个例子，这样使用泛型方法十分不方便，因为不是所有类型都拥有某个方法，比如上方例子中不是所有类都拥有CompareTo方法，因而报错。

泛型约束

泛型约束可以解决上面第二个例子中的问题，可以通过where关键字使得当前泛型必须实现IComparable<T>接口，使得上面例子中的代码不再报错

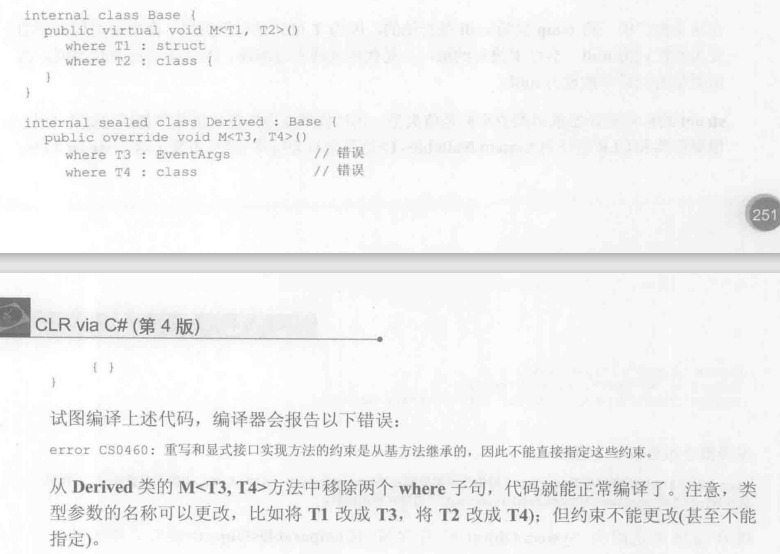


Where关键字告诉编译器给类型参数T指定的类型必须实现同类型T的泛型IComparable<T>接口，这个接口定义了CompareTo方法

Ps：泛型约束可以约束使用到类型参数的任何地方（如类、委托、接口、方法）

在重写虚方法时，重写的方法必须指定相同数量的类型参数，并且这些类型参数会继承基类的约束，并且不能为这个重写的虚方法指定更多的约束（但约束的名称是可以改变的，见P251）。

例子如下：



约束可以分为主要约束、次要约束、构造器约束

主要约束：可以指定0个或者1个主要约束。主要约束可以是（非特殊的）引用类型，代表类型参数T必须是约束中的引用类型或其子类；还可以是“class”代表T必须是引用类型；还可以是“struct”代表T必须是值类型。

次要约束：可以指定0个或者若干个次要约束。代表类型实参T必须继承这些接口。

构造器约束：可以指定0个或1个构造器约束，它向编译器承诺类型实参实现了公共无参构造器和非抽象类型。因为每个类型都含有父类的无参构造器，所以约束的关键字是new（）。

Ps：泛型约束的种类

图片包含 文本

描述已自动生成

两个泛型类型参数不能相互比较

报纸上的文字

描述已自动生成