实例构造器和类（class）（引用类型）

创建引用类型实例时，首先为实例的数据字段分配内存，然后初始化对象的附加字段(即类型对象指针和同步块索引)，最后调用类型的实例构造器设置对象的的初始状态。

构造引用类型对象时，在调用类型的实例构造器之前，给对象分配的内存会先被归零，此时没有被构造器显示重写的字段会被置为0或者null。

这是一个例子：

图形用户界面, 文本, 应用程序

描述已自动生成

这个例子中,当创建一个引用类型的对象时，CLR会在堆上为该对象分配内存。在调用构造函数之前，这些内存会被清零，这意味着所有字段都会被初始化为其默认值（0或null）。紧接着，CLR会执行“字段初始化器“，为字段赋予指定的初始值（本例中在这个阶段，myInt从原本的0被赋值为1）。最后，CLR会调用构造函数（此时myDouble从原本的0被赋值为3.14）。

所谓“字段初始化器”是C#编译器提供的一个语法，以内联（即嵌入）方式初始化字段，看IL代码可以得知像myInt=1这样的语法被转化为实例构造器中的代码来执行对字段的初始化（但这个赋值操作，会在构造器中的语句执行之前执行）

以下是IL代码（IL\_0002那行）

文本

描述已自动生成

关于所谓“字段初始化器”的理解，public int myInt=1相当于在执行构造器里其他的语句前，执行了一次myInt=1的赋值操作

文本

中度可信度描述已自动生成

如果类没有显式定义任何构造器，C#编译器将定义一个默认的无参构造器，这个无参构造器的实现也仅仅是调用了基类的构造器。

手机屏幕截图

描述已自动生成

Ps：如果基类中仅定义了一个有参构造器，C#编译器将不会实现那个默认的无参构造器，这时需要显式调用基类的构造器，就像下面这个例子这样使用base然后再括号中传入参数

图形用户界面, 文本, 应用程序

描述已自动生成

另外，在实例化一个类时，会首先调用其基类的构造器，然后再调用自己的构造器（最终object类的默认构造器会得到调用）。

如果类的修饰符为abstract（即抽象类），那么编译器生成的默认构造器的可访问性就为protected；否则构造器会被赋予public可访问性。如果类的修饰符为static（即静态类、抽象密封类，理解为同时加上了sealed和abstract），那么编译器不会在类的定义中生成那个默认的实例构造器（也就是静态类无法实例化）。

极少数时候可以在不调用实例构造器的前提下创建类型的实例。比如Memberwiseclone方法和反序列代码使用的GetUninitializedObject方法。下面是Memberwiseclone方法的例子，并未调用构造函数也可以实例化myClass2

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

Ps：C#中没有“正常手段完成的深拷贝”，C#中只有浅拷贝。Memberwiseclone是浅拷贝

不要在构造器里以虚方式调用虚方法

一些文字和图片的手机截图

描述已自动生成

但可以以非虚方式调用虚方法（调用指定的类型对象的方法）

图形用户界面, 文本, 应用程序

描述已自动生成

结果（假设MyBaseClass拥有基类且它的基类里也有该虚方法时，是这样的结果）

图形用户界面, 应用程序

描述已自动生成

把base.MyVirtualMethod();改为this.MyVirtualMethod();，结果：

文本

描述已自动生成

证明以非虚方式调用虚方法是可以的

在同一个类型中使用许多不同签名的构造器会导致一个问题，编译器可能对同一个字段进行了多次初始化，这在IL代码中也会体现。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

这个例子中两个构造器的IL\_0001和0002都执行了一样的操作，就是把myInt赋值为1.

解决的方法是创建单个构造器来执行共有的字段的初始化，其他构造器显式调用这个初始化共有字段的构造器。

下面是书上的例子

一些文字和图片的手机截图

描述已自动生成

对应的IL代码，确实避免了重复初始化

图形用户界面, 文本, 应用程序

描述已自动生成

实例构造器和结构（struct结构体）（值类型）

书上的一个例子：

文本

低可信度描述已自动生成

CLR不会为包含在引用类型中的每个值类型的字段调用构造器，只会初始化为0或者null。本例中的m\_topLeft和m\_bottomRight被初始化为null。

CLR允许为值类型定义构造器。但必须显式调用，值类型的构造器才会执行（也就是说C#编译器不会为值类型生成“默认构造器”，不显式调用就不会调用）。

书上P165的例子很好地反映了“值类型的构造器必须显式调用，否则不会被调用”。

Ps：值类型不允许定义无参构造器(C#编译器不允许但是CLR允许)，比如在一个struct结构体内定义了一个无参构造器，那么会报错“error CS0568：结构不能包含显示的无参数构造器”。

因此也不可以使用内联（嵌入）的方式为结构体内实例字段初始化。例子如下，

这会报错 error CS0573 结构中不能有实例字段初始值设定项

文本

描述已自动生成

同时必须给结构体的所有字段都进行初始化，下面例子，会报错 error CS0171：在控制返回到调用方之前，字段xxx必须完全赋值

手机屏幕截图

描述已自动生成

值类型使用“new”操作符会直接把所有字段都置为0，下面这个书上的例子很奇怪（我不太理解），这个例子使用new操作符避免了上个例子中的不完全赋值的报错

文本

中度可信度描述已自动生成

在栈上的值类型必须要进行赋值。

手机截图图有时间和文字

描述已自动生成

类型构造器（静态构造器）

类型构造器的作用是设置类型的初始状态（初始化静态字段），类型默认没有定义类型构造器（如果定义也仅能定义一个），所有类都可以定义类型构造器（无论是否是静态类），且类型构造器不能有参数。下面是书上举出的一个例子，分别定义了引用类型和值类型的类型构造器。

文本, 信件

描述已自动生成

可以看到类型构造器有“static”修饰符且不能有参数，且类型构造器默认private，加任何修饰符都会导致报错。

类型构造器不能直接调用，并且仅应由CLR（公共语言运行时） 调用（可以自动调用它们）。用户无法控制在程序中执行静态构造函数的时间。自动调用静态构造函数。它在创建第一个实例或引用该类（不是其基类）中声明的任何静态成员之前初始化类。且在每个AppDomain中，每个类型的类型构造器只会执行一次，而同时可能会有多个线程同时试图执行类型构造器，所有保证执行一次的方法是加互斥锁（是线程安全的）（具体见P167下方）。即使是双向依赖的两个类的静态类型构造器也只会分别执行一次（见P168）。

下面是书上P167原文

报纸上的文字

描述已自动生成

什么情况下CLR一定会调用值类型的静态类型构造器呢？

文本, 信件

描述已自动生成

1.当访问静态字段/调用静态方法

2.显式调用实例构造器的时候

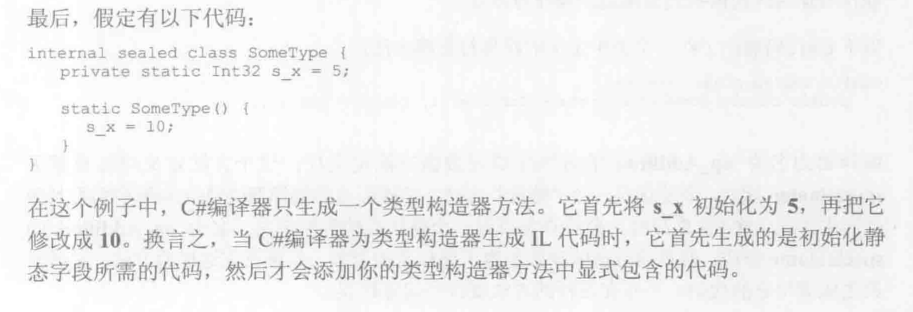
3.重写了继承的方法的时候

[关于值类型的类型构造器的一些思考和笔记\_给我希望的博客-CSDN博客](https://blog.csdn.net/weixin_43374476/article/details/109996250)

Ps：

1.调用类型构造器不同于调用实例构造器，调用类型构造器时不会调用其基类型的类型构造器，因为静态字段无法从基类继承。

2.如同类的实例构造器使用嵌入的方式初始化实例的字段，调用静态类型构造器前也会以内联的方式把静态字段初始化。例子如下

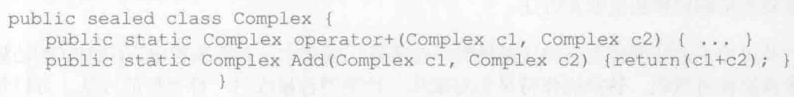


操作符重载

操作符重载必须是public和static修饰符的方法，下面是一个C#操作符重载的例子：



必须是公共的静态方法，static后是类名，关键字是operator并在之后加上想要重载的操作符，如上例是“+”，加号。以下是加上了所有语言代码都可以调用的“友好的运算符重载”，但会有性能损失（添加一个Add方法使用重载之后的操作符）

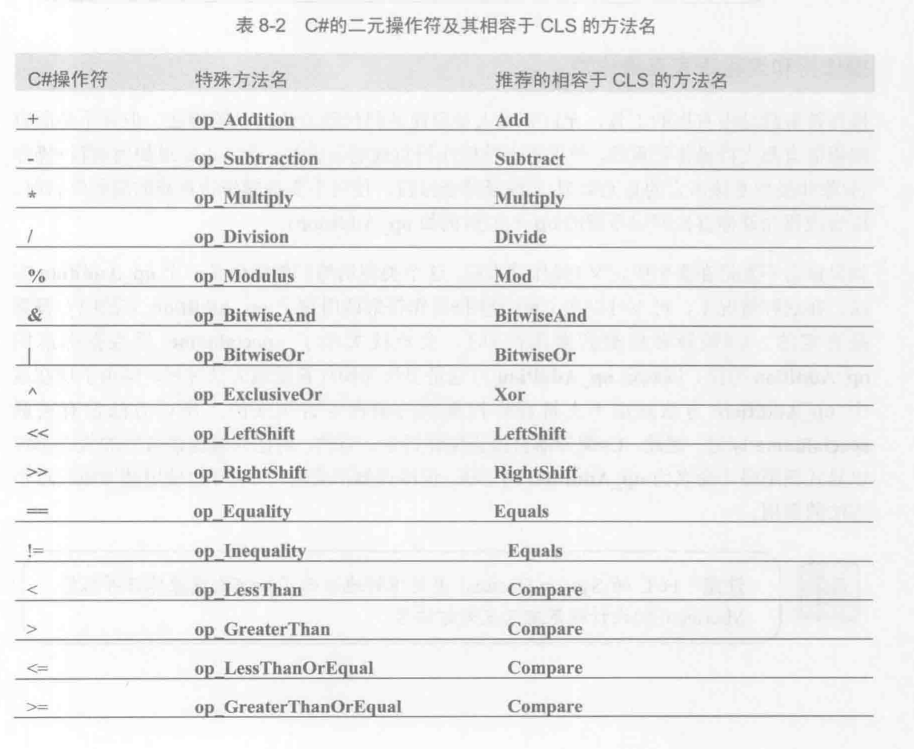


Ps：操作符重载只和编译器、编程语言有关，和CLR没关系。

以下是C#允许重载的一元操作符和二元操作符

手机屏幕截图

描述已自动生成



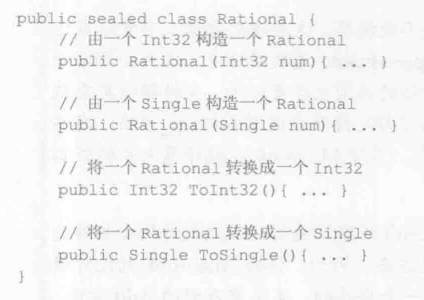
几元操作符代表有几个操作数，比如++为一元操作符，只需要一个字段。

转换操作符方法

当源类型和目标类型都是编译器识别的基元类型时，编译器知道如何生成转换对象所需的代码；但源目标或者目标类型不是基元类型，编译器会生成代码要求CLR执行转换（强制转型），但这种情况下CLR只是检查源对象的类型和目标类型（以及其派生类型）是不是相同的。

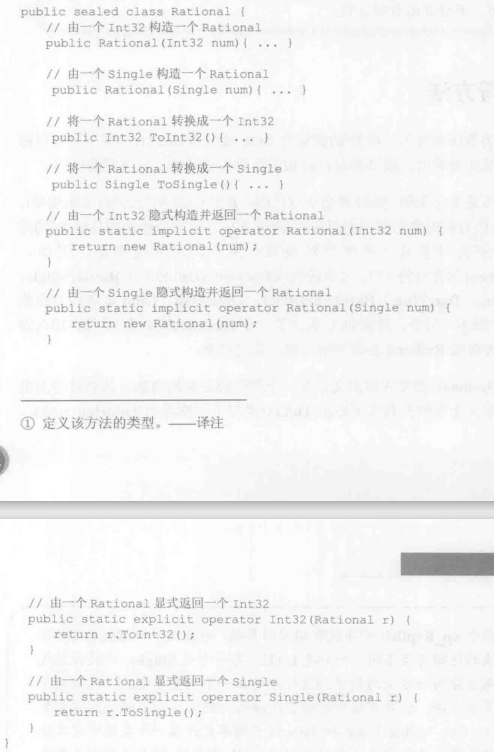
如何把对象从一种类型转换为另一种全然不同的类型？书上P173-P175给出有关Rational类型（有理数类型）的转换。

如何为Rational类型正确定义转换构造器和方法(创建一个Rational对象时传入不同类型参数然后得到一个Rational类型对象)



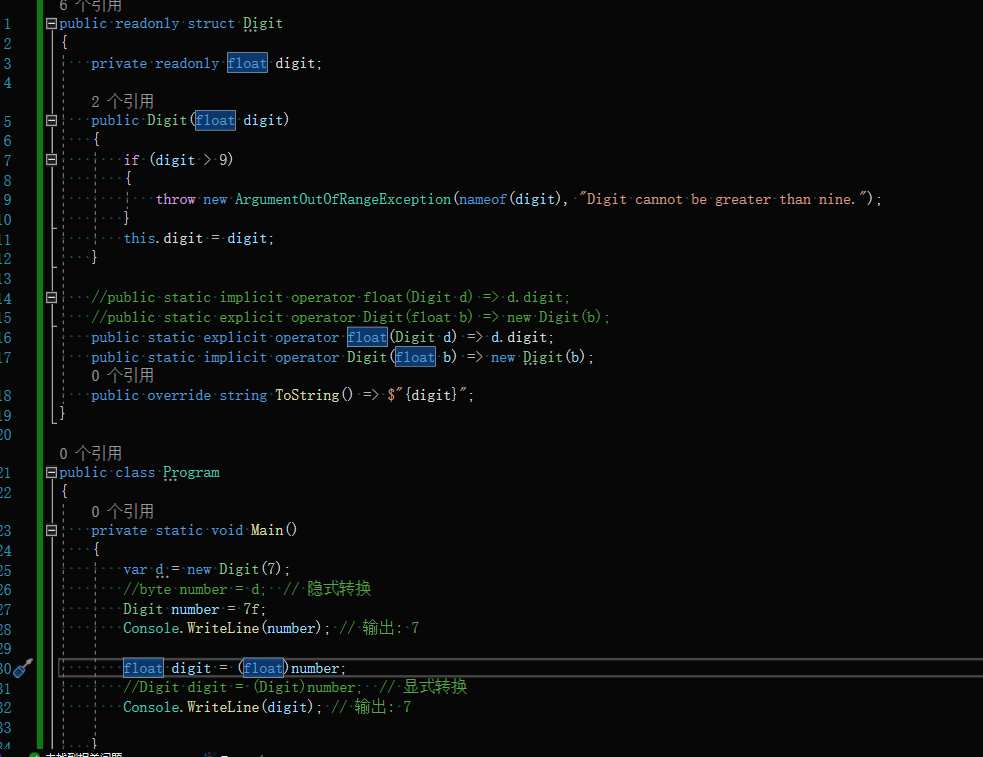
除了操作符重载，C#还提供转换符重载，通过implicit和explicit指定是隐式还是显式转换，再在这个关键字后加上operator关键字向编译器表明是操作符重载，然后指定对象要转换为什么类型，括号内是源类型

以下是书上P174例子

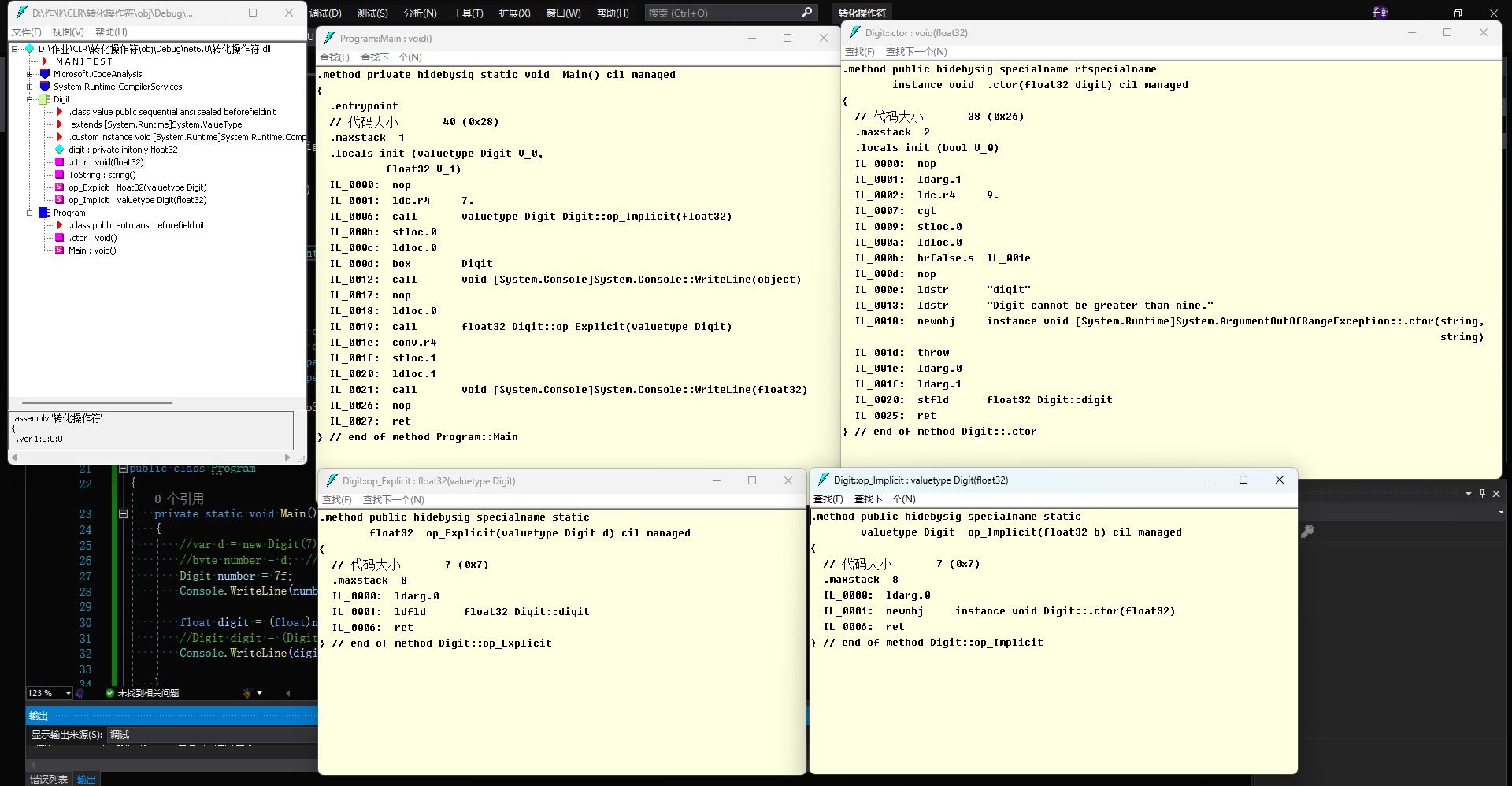


Ps：只有在不造成精度损失时才可以定义隐式转换操作符，否则应该定义显式转换操作符，且显式转换失败时应该让显式转换操作符方法抛出OverflowException或者InvalidOperationException异常

下面是另外一个例子，包含了抛出异常：



IL代码：



通过IL代码得知，隐式转换和显式转换本身并不需要使用new操作符来完成实例化。但是，在执行用户定义的转换操作符方法时，这些方法可能会使用new操作符来创建新的实例（这在Main的含有new操作符的那一行体现了）。

Ps：使用is和as方法时并不会调用强制转换操作符，所以as的速度比强转快。

扩展方法

使用扩展方法的意义：它允许向现有类型添加新方法，而无需创建新的派生类型、重新编译或以其他方式修改原始类型。扩展方法是一种静态方法，但可以像扩展类型上的实例方法一样进行调用。这意味着您可以使用扩展方法来增强现有类型的功能，而无需修改现有代码。

书上扩展方法的例子：

文本

描述已自动生成

像扩展类型上的实例方法一样进行调用，例子：

图片包含 文本

描述已自动生成

这是另一个例子：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

定义扩展方法的规则和原则：

1.C#只支持扩展方法。不支持扩展属性、扩展事件、扩展操作符等等。

2.扩展方法（即第一个参数前有“this”的方法）必须在非泛型静态类中声明，但必须至少有一个参数，this关键字只能在第一个参数前标记。

3.扩展方法只能在静态类中定义，但不可以在嵌套类内定义。

4.C#编译器可能找到其他的同名扩展方法，使用扩展方法时记得加上该扩展方法的静态类所在的命名空间。

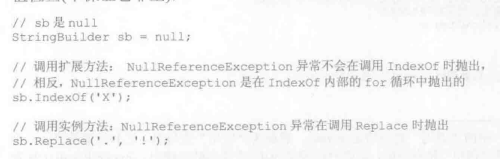
5.多个静态类可以定义相同的扩展方法。

6.用一个扩展方法扩展一个类型时，同时也扩充了它的派生类型。例如为Object类扩展了方法，那么所有Object的子类（即所有类）都会拥有这个扩展方法。因此需要谨慎使用扩展方法。

7.扩展方法可能存在版本问题。具体见P179。

由于扩展方法实际时对一个静态方法的调用，所以CLR不会生成代码对调用方法的表达式的值进行null检查，即不会保证这个值非空。

书上给出的例子



可以为接口定义扩展类型，那么如果类型实现了该接口，就会拥有这个扩展方法

文本

描述已自动生成

可以为委托定义扩展方法

文本

描述已自动生成

可以为枚举定义扩展方法

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

可以为委托定义扩展方法

文本

描述已自动生成

然而不是所有语言都像C#一样支持扩展方法，所以存在一个ExtensionAttribute类用于指示方法是扩展方法，或者指示类或程序集包含扩展方法。而在C#中使用this指定扩展方法时编译器自动为该方法应用了一个定制特性用于标记扩展方法，扩展方法所属的静态类的元数据中也会应用这个特性。

分部方法：

分部方法的使用规则和原则：

1.只能在分部类或者结构体中声明。

2.分部方法的返回类型始终是void，且不能用out修饰符标记的参数。存在这样的限制的原因是方法在运行时可能不存在，所以不能将变量初始化为“方法也许会返回的东西”。分部方法可以有ref，可以是泛型方法，可以是实例方法和静态方法，也可以标记为unsafe。

3.分部方法的声明和实现必须有完全一致的签名。

4.如果没有对应的实现部分，就不能在代码中创建一个委托来引用这个分部方法。

5.分部方法会被视为private方法，因此它和静态构造器前都不需要且不允许加上private修饰符。

分部方法的意义和使用场景：

分部方法允许将一个方法的定义拆分到多个文件中。一个分部类或结构可以包含分部方法。类的一个部分包含方法的签名，可以在同一部分或另一部分中定义实现。如果未提供该实现，则会在编译时删除方法以及对方法的所有调用。根据方法签名，可能需要实现。

分部方法的存在意义在于它为开发人员提供了一种灵活且强大的方式来组织和管理代码。它可以帮助您更好地处理大型项目，使多位程序员能够同时对同一个类进行处理。此外，当使用自动生成的源文件时，可以添加代码而不需要重新创建源文件。这样可以提高代码的可读性和可维护性，并且可以帮助更快地开发出高质量的应用程序。