可选参数：方法、构造函数、索引器或委托的定义可以指定其形参为必需还是可选。任何调用都必须为所有必需的形参提供实参，但可以为可选的形参省略实参。每个可选形参都有一个默认值作为其定义的一部分。如果没有为该形参发送实参，则使用默认值。

例子：

图片包含 文本

描述已自动生成

其中optionalInt和optionalBool为可选参数，等于号后面的即为默认值，可选参数的位置必须在非可选参数之后，如上例的两个可选参数在签名中的位置放在了required之后。

命名参数：通过命名实参，不再需要将实参的顺序与所调用方法的形参列表中的形参顺序相匹配。每个形参的实参都可按形参名称进行指定。

例子：

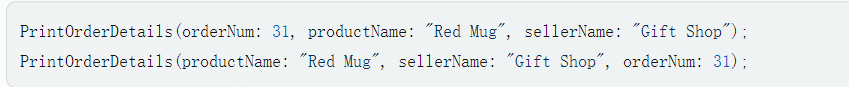
文本

中度可信度描述已自动生成

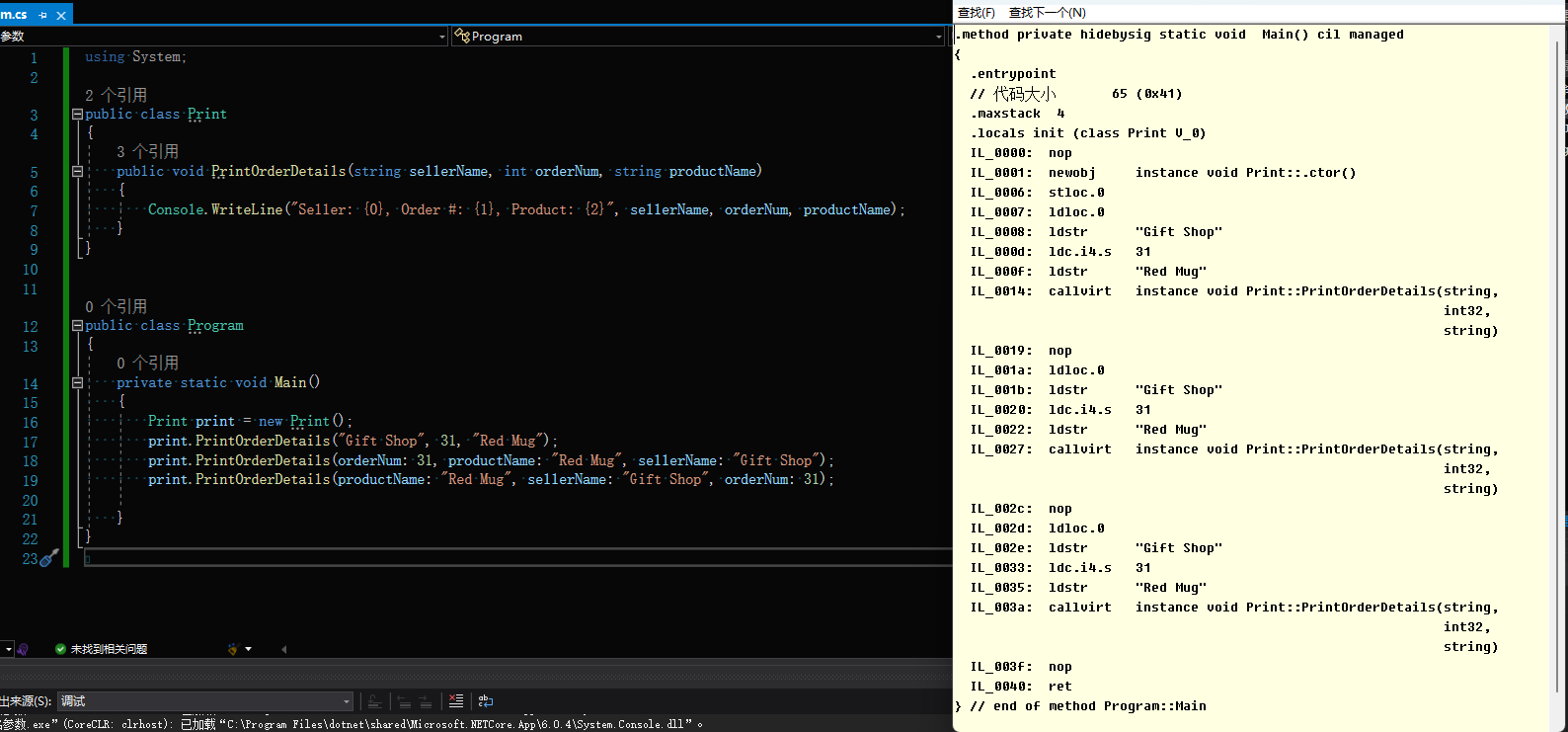
这是正常去调用这个函数：



如果不记得顺序只记得参数的名字，那以“形参名+冒号+实参”调用



Ps：使用命名参数打乱调用时传入参数的顺序，并不会改变实际传入参数的顺序，下面Main函数的IL代码表明，三次调用都只是改变了参数调用时输入的顺序，但是生成的IL代码完全一样，



使用命名参数和可选参数的规则和原则：

1.可为方法、构造器方法和有参数行（C#索引器）的参数设置默认值，还可以为委托定义一部分的参数指定默认值。

2.有默认值的参数必须放在没有默认值的参数之后。

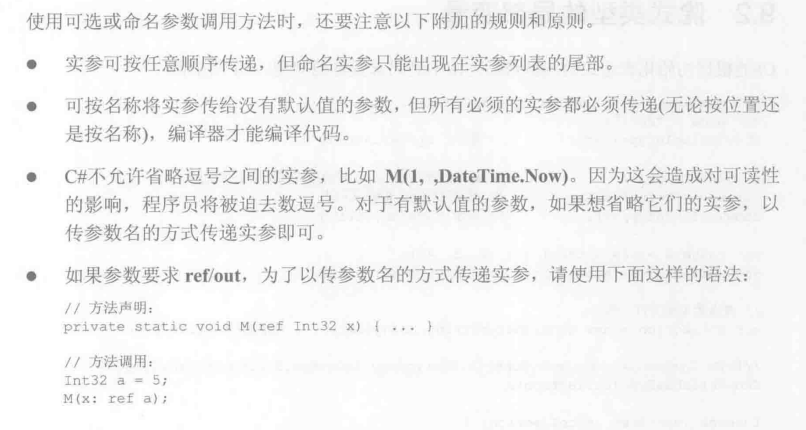
3.默认值必须是编译时可以确定（以嵌入的方式初始化）的常量值。

4.不能重命名形参。

5.如果更改了参数的默认值，如果没有重新编译包含call site的代码，它在调用方法时会传递旧的默认值。

6.如果参数使用了ref和out关键字进行标识，就不能设为默认值。

Ps：call site是发出调用的地方，可理解成调用了一个目标方法的表达式或代码行



类似ExtensionAttribute，DefaultParameterValueAttribute和OptionalAttribute也是起类似作用。

隐式类型局部变量（var）：

Var存在的原因:C#能根据初始化表达式的类型判断方法中的局部变量的类型。

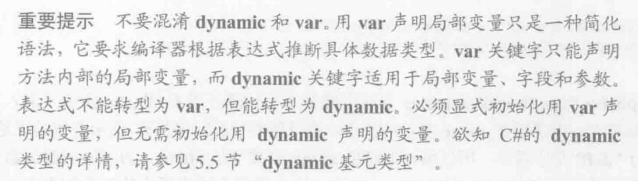
Var存在的意义：隐式局部变量 var 的存在意义在于它可以让编译器从初始化表达式推断出变量的类型。这样，开发人员就不必考虑变量的具体类型，可以提高开发效率。例如有一个表达式 var x = new List<int>()，则编译器会自动推断出 x 的类型为 List<int>。

使用var需要注意的几点（其他的见第五章笔记中dynamic和var的对比）：

1.var只能用于方法内部的局部变量。不能用于声明方法的参数类型以及返回值类型等。

2.var不能用于声明静态字段。

3.不能将null赋值给var局部变量。



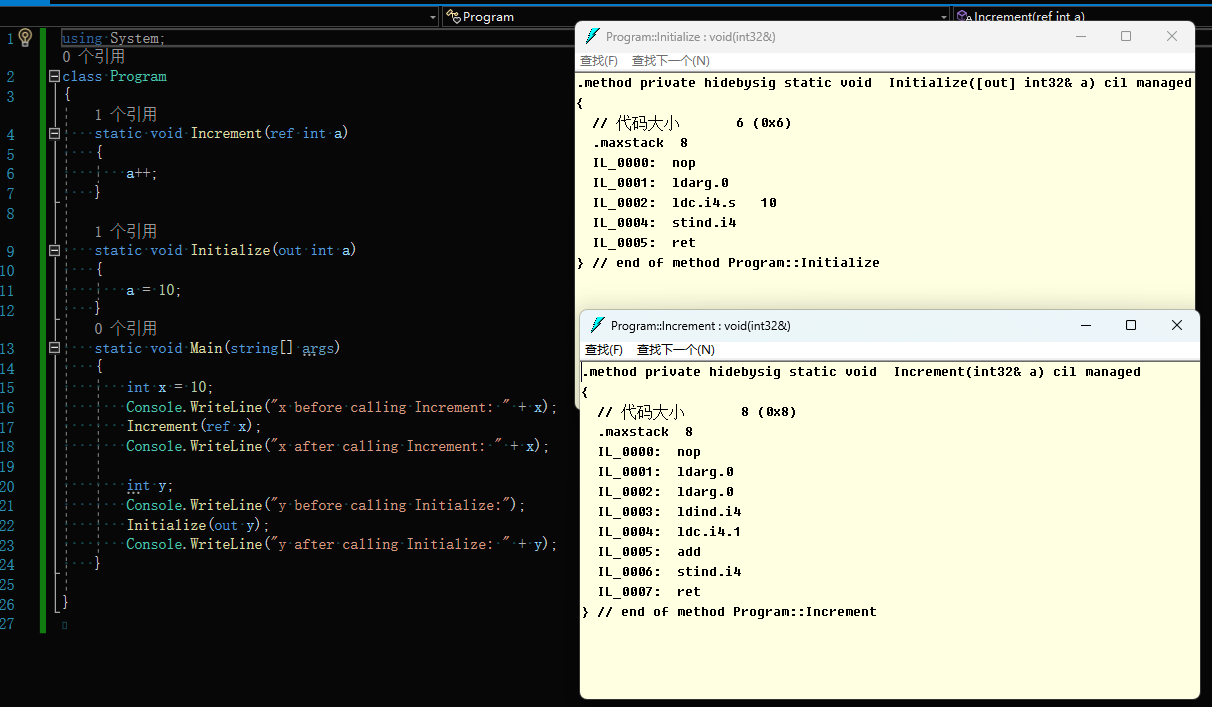
以传引用的方式向方法传递参数

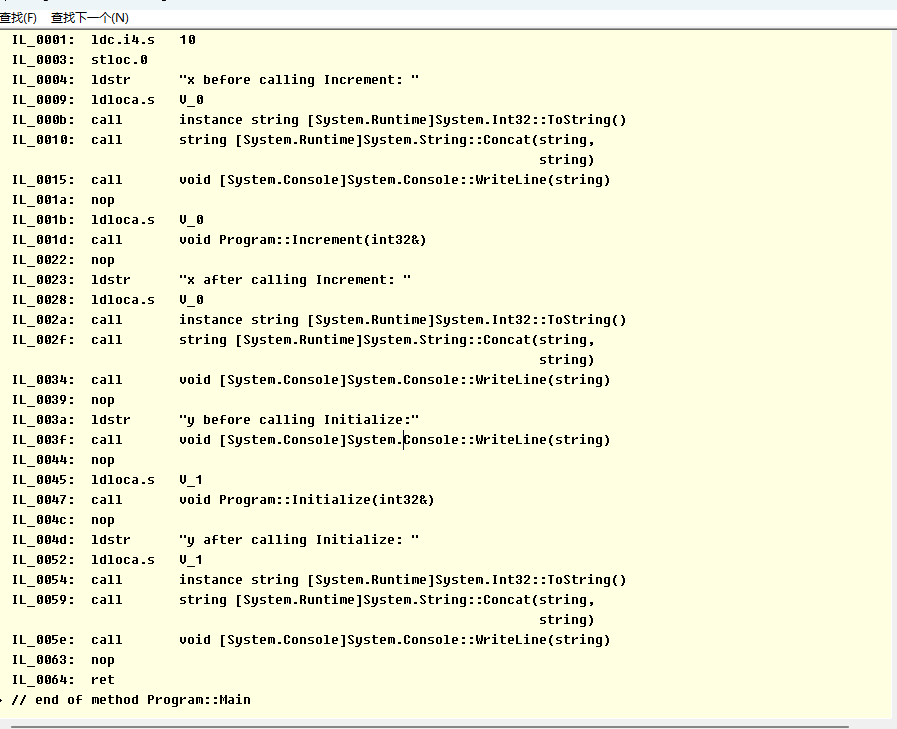
CLR允许以传引用的的方式传递参数，这在C#中体现在ref和out这两个关键字。并且CLR不区分ref和out关键字，无论使用哪个关键字都生成相同的IL代码。

Ref关键字，ref必须在该函数外部进行初始化（在调用该函数前必须初始化了这个变量）。

Out关键字，out必须在该函数内部进行初始化（在函数内必须有该变量的初始化语句）。

确实使用ref和out生成的IL代码都是一样的，即元数据签名一样（正因如此重载中ref和out是“完全一样的”）





Ps：为大的值类型使用out或者ref可以提升代码的执行效率，因为避免了在进行方法调用时复制值类型实例，ref和out效率相当。

仅当方法“返回”对“方法知道的一个对象”的引用时，为引用类型使用out和ref才有意义。（这句话怎么理解P194）

当方法返回时，调用方法时传递给该方法的变量将引用一个新的对象，即改变对象的引用，这样使用ref和out关键字传递引用类型参数才有意义。



向方法传递可变数量的参数

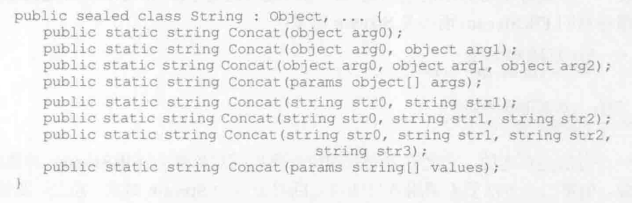
变长参数的例子：

Params关键字后接一个数组作为变长参数，且params关键字仅能用于方法签名中的最后一个参数，且params只能标识一维数组，即变长参数只能是一维数组，而且可以为变长参数数组传递null。



使用变长参数的意义：变长参数的使用意义在于它允许在调用方法时传入不定长度的参数。这样开发人员就不必考虑参数的具体个数，可以提高开发效率。但变长参数会对性能造成一定影响。而为了减小这个影响，在参数少的时候提供重载，使用重载的方法，在参数多过一定程度时再使用含有变长参数的方法。

本书P197提供了一个例子：



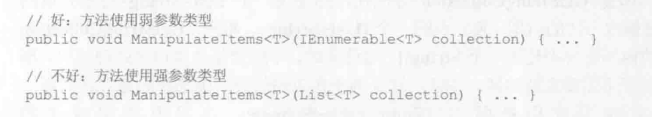
参数和返回类型的设计规范：

参数类型最好为弱类型，返回类型最好为强类型。（P197中“声明方法的参数类型时，应尽量指定最弱的类型，宁愿要接口也不要基类”）

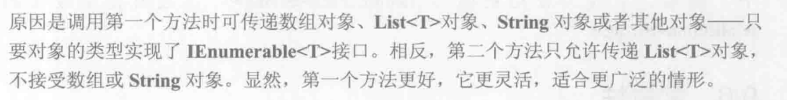
Ps：什么是强参数类型，什么是弱参数类型。

强类型是使用前必须指定类型的变量，弱类型是运行时判定类型的变量。

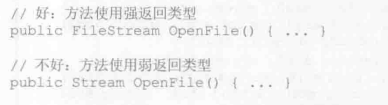
声明相关例子：



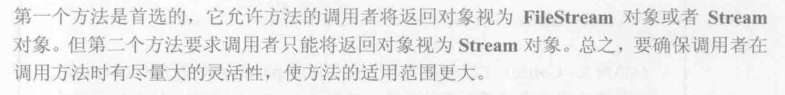
解释：



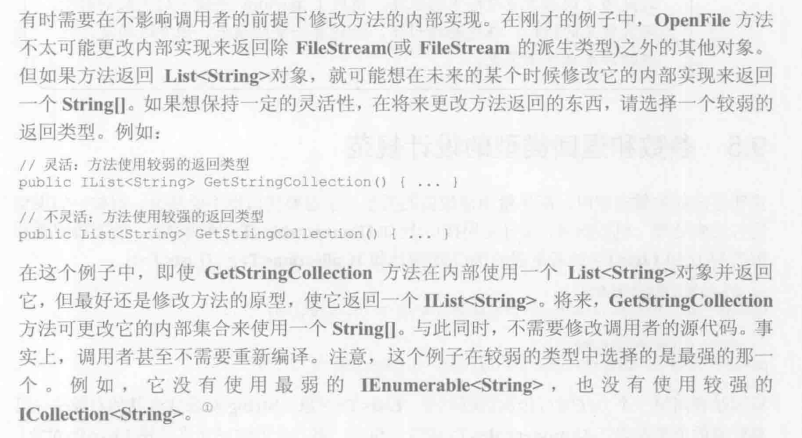
返回相关例子：



解释（因为FileStream可以转换为它的基类，而它是强返回类型，可以有更多的基类，自然灵活性更强。在将来不改变方法实现的情况下，返回使用强类型会更好）：



而如果将来会修改方法的内部实现，则恰恰相反应该使用较弱的返回类型（这是特殊情况），与上面的例子恰恰相反，因为要在不影响调用的情况下更改实现，自然弱类型会有更多的操作空间。例子：



常量性

什么是常量性？

常量性是指对象的某些属性或方法在对象的整个生命周期内保持不变。在 C++ 中，可以使用 const 关键字来声明常量数据成员和常量成员函数。常量数据成员在对象创建后不能被修改，而常量成员函数不能修改对象的任何非静态数据成员。这样可以确保对象的某些属性在整个生命周期内保持不变，从而提高代码的可靠性和可维护性。

CLR不支持常量性，原因是，如果支持常量性CLR就要对每个写入操作进行验证以确定更改的不是常量，这对性能影响很大。且在违反常量性的地方会造成CLR抛出异常，且会给开发人员带来大量复杂性。