本文主要是以C#为例介绍.NET中的三种指针类型：对象引用、非托管指针、托管指针。

变量：给存储单元指定名称，即定义内存单元的名称或者说是标识。

指针：一种特殊的变量，其存储的值是地址而不是值本身。

托管堆（Managed Heap）：这就是NET中的托管堆，用来存放引用类型，它是由GC（垃圾回收器自动进行回收）管理。

调用堆栈（Call Stack）：调用堆栈是一个方法列表，按调用顺序保存所有在运行期被调用的方法。

计算堆栈（Evalution Stack）：每个线程都有自己的线程栈，IL里面的任何计算，都发生在Evalution Stack 上，其实就是个Stack结构。可以push，也可以pop。

1. 对象引用

与值类型变量直接包含值不同，引用类型变量存储的是数据的存储位置（托管堆内存地址）

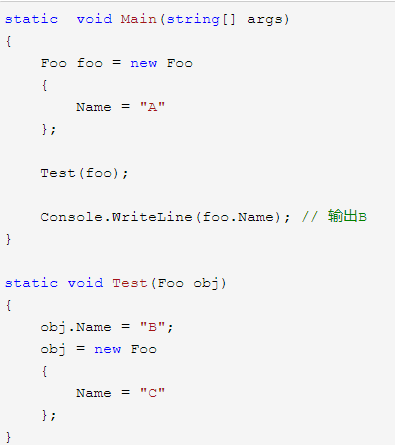
对象引用是在托管堆上分配的对象的开始位置指针。

访问数据时，运行时要先从变量中读取内存位置，再跳转到包含数据的内存位置，这一切都是隐藏在CLR背后发生的事情，我们在使用引用类型的时候，并不需要关心。

1. 值传递和引用传递

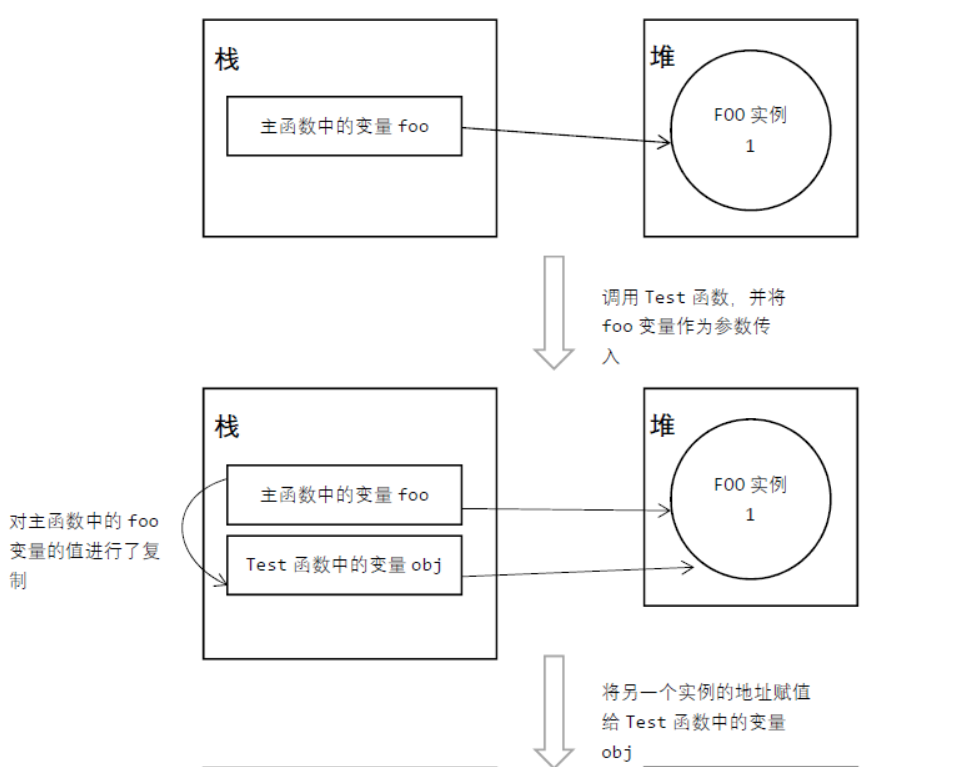
误区：对象在C#中是按引用传递的。

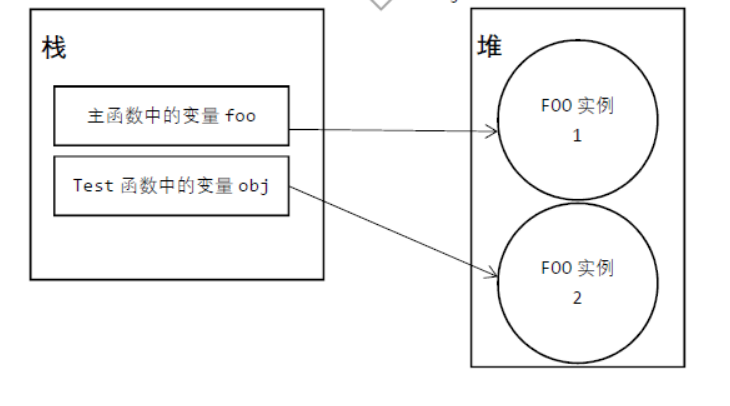
举例：



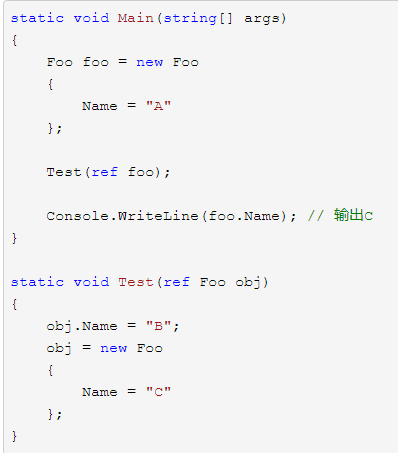
按照引用传递的定义，上述代码输出的应该是C，而实际是B。

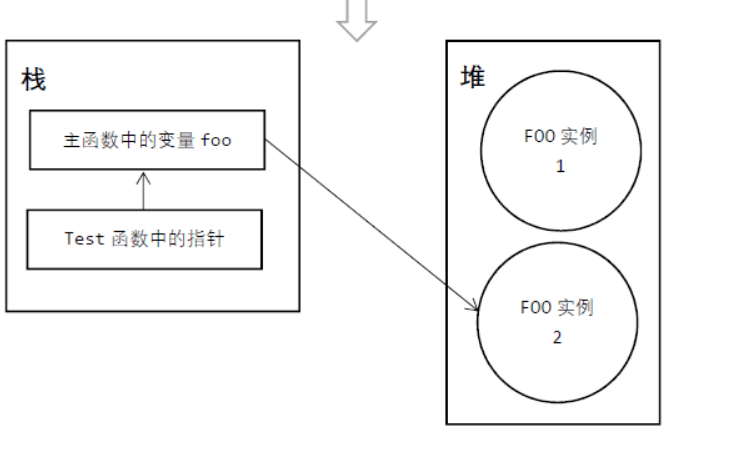
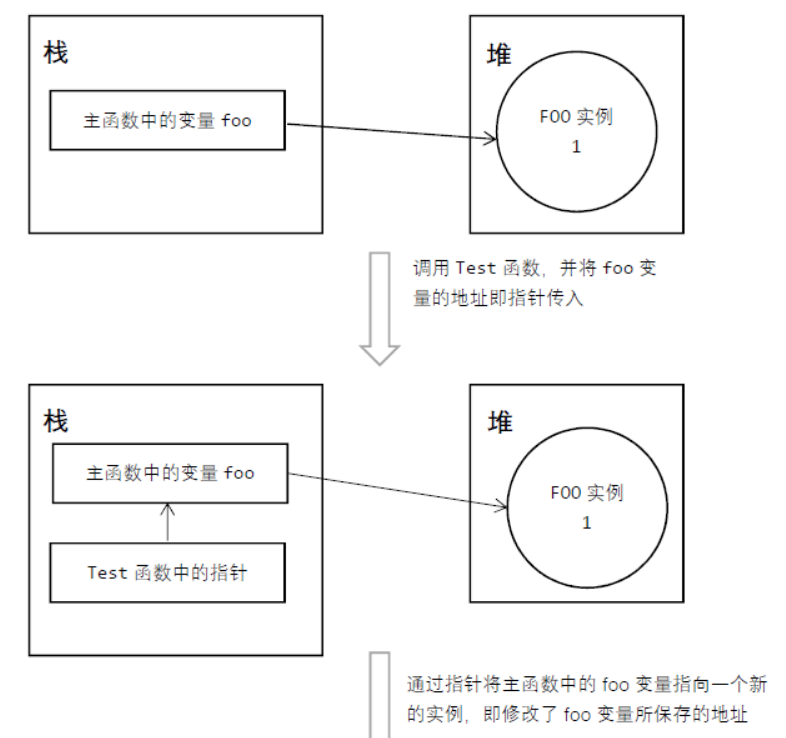
这是因为C#默认是按值传递的，在将Main函数的foo变量传入到Test函数时，会将它所包含的值（引用对象）复制给变量obj（还是值传递）。所以可以通过obj变量修改原来的实例成员，这仅仅是因为引用类型的特性导致的，并不是所谓的引用传递。因为如果将obj变量指向一个新的实例，并不会影响到foo变量，它们两者完全是独立的。





此时使用关键字ref，实现引用传递



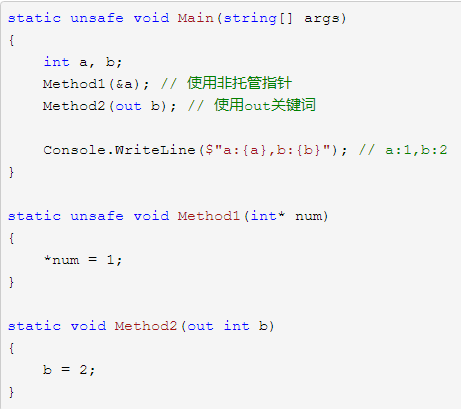


1. 认识托管指针和非托管指针

在C#中，如果我们想要定义一个引用传递的方法，我们需要通过给方法参数加上ref或者out关键字。

同时C#也允许我们通过unsafe关键字编写不安全的代码。

那么这两者有什么区别呢（托管指针和非托管指针）。



静态方法Method1中的参数对应的IL代码int32\* num。

静态方法Method2中的参数对应的IL代码是[out]int32&b，其中[out]即使去除也不影响代码的运行。

在CLR中可以定义两种类型的指针：



也就是说用out、ref定义的指针类型其实对应的就是CLR中的托管指针。

1. 非托管指针

非托管指针的使用主要包括

寻址运算符&

间接寻址运算符\*

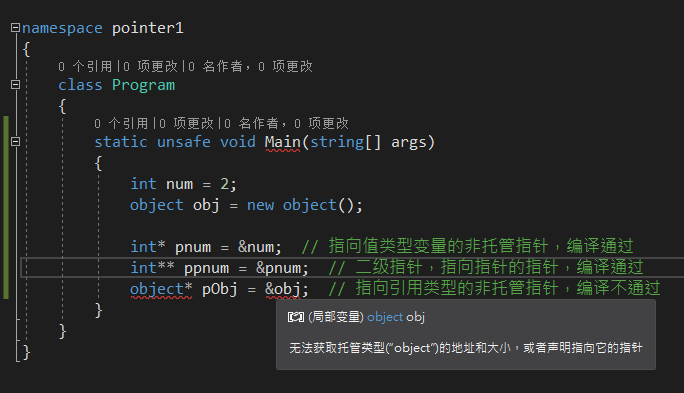
用于结构指针的成员访问运算符->

非托管指针的用法和C\C++基本一致（C\C++中没有托管的概念，本身对数据的处理就是非托管的，所以说基本一致就是其中指针的用法基本一致）。这里列出几个.net中非托管指针的注意点。

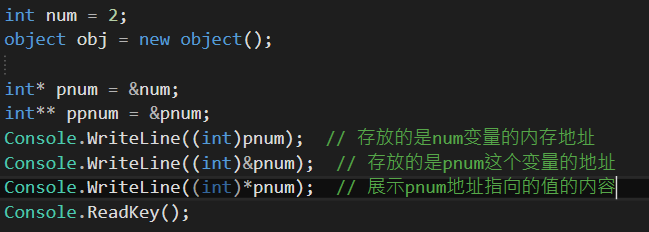
1. 非托管指针不能指向对象引用

我们知道一个一个引用类型，它所存储的是托管堆上实际的内存地址。这个内存地址记录本身也是保存在内存的某个位置上。

我们可以创建指向值类型变量的非托管指针，也可以创建多级非托管指针，但是不能创建指向引用类型变量（存放的是对象的地址即创建的对象的地址，因为对象的地址是一系列的，而引用变量是存放第一个内存地址）的非托管指针。

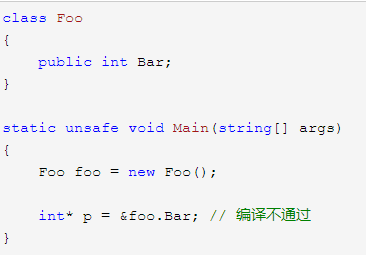


如何运行unsafe代码：需要在configuration property>build中把运行unsafe代码设置为TRUE。



1. 类成员指针

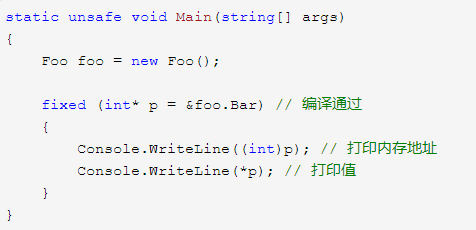
如果我么想要创建一个对象的值类型成员变量的指针，下方的代码是无法通过的。



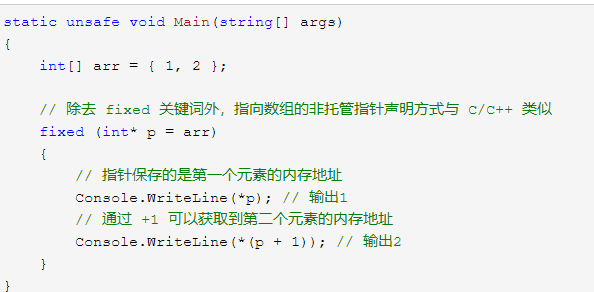
为什么非托管指针不能指向引用类型和类成员？

因为**对于生存在托管堆上的引用类型的实例而言**，在一次GC之后，其内存位置可能会发生变动，包含在实例内的成员变量也就随之发生了位置的移动。对于标识内存位置的指针而言，显然这样的情况是不被允许的。

但是我们可以通过fixed关键字 避免GC时实例内存位置的移动来实现这种类型的指针的创建。



同样，我们也可以利用fixed关键字创建指向值类型数据的指针（数组是引用类型，数组的元素是值类型）



1. 托管指针

当我们在使用引用传递时使用的ref\out关键字其实就是创建了托管指针。

在C#7之前，我们只能在方法参数中见到托管指针的身影，C#7进一步开放了托管指针的功能。例如实现了和非托管指针一样，用于方法的返回值。

个人总结：对于引用类型，如果不通过ref\out关键字创建托管指针，当我们将其作为方法参数时，对其内容的修改是不会对原先创建的引用类型 实例发生改变，而是创建了一个新的拷贝，在对应方法中操作的也是这个拷贝，而不是原先的实例内容；而使用了ref\out关键字后就会生成一个托管指针，相比于非托管指针，托管指针更加安全，不需要进行内存管理等一系列操作，托管指针就可以完完全全指向实例，对实例进行操作；非托管指针跟C\C++中的指针功能几乎一致，拥有较大的权限，适用于高级开发人员。非托管指针的麻烦是因为权限过大而非法修改某些变量；内存泄漏。

内存泄漏：指的是因为疏忽或错误造成程序未能释放已经不再使用的内存的情况。

内存泄漏并不是指内存物理上的消失，而是应用程序分配某段内存后，因为设计错误，失去了对该段内存的控制，因而造成了内存的浪费。