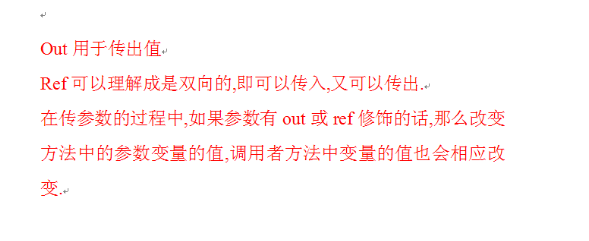
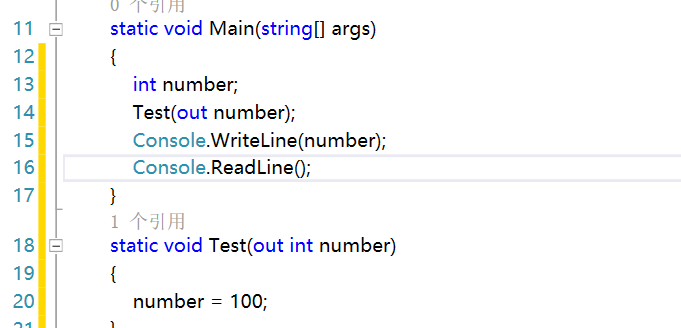
out ref 详解：



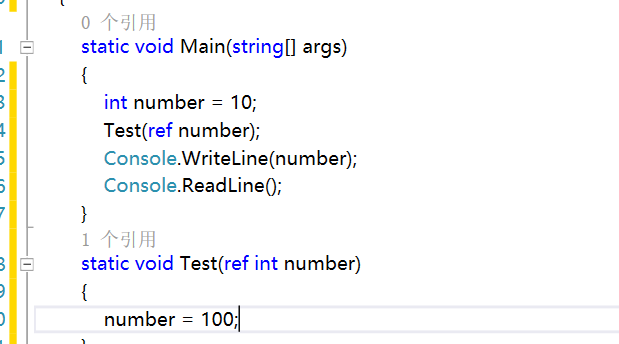
他们的区别在写的时候的区别在哪？

out：



在此图中，out在调用者里是没有赋值的，直接传入方法Test中，这时候在方法Test的时候再赋值

ref:



在此图中，ref再调用者里一开始就已经赋值了，所以number传入方法Test的时候，number=10，这时候在方法中再对number进行赋值

需要注意的是：

out：在方法中必须先赋值才能使用

ref：在方法中可以不用先赋值

原理：

白话：

我们把ref 和out使用的时候当成是借别人的笔记，当我们在别人的笔记进行修改的时候，别人的笔记肯定也会给需改

而不加ref和out使用的时候则当成是复印别人的笔记，变成自己的笔记，当我们对自己的笔记进行修改的时候，别人的笔记也不会给修改

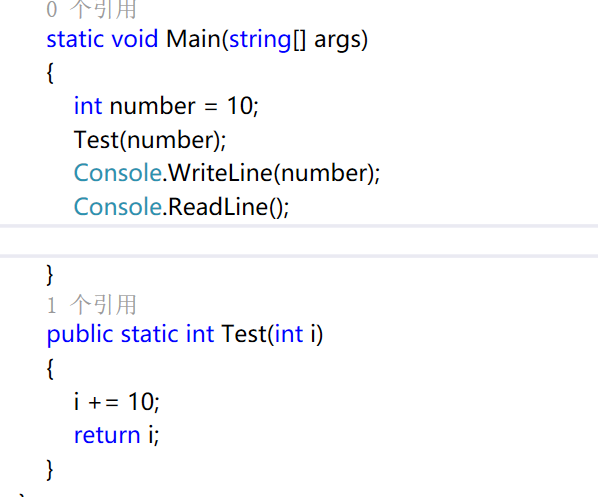
科学：

加了out、ref：使用的时候就是把主函数中变量在内存栈中开辟开辟空间的地址给了自定义函数中的参数的变量，这样他们就共用一个地址，在自定义方法中改变了变量的值，主函数的值也会发生改变

不加out、ref：使用的时候就是主函数变量在内存栈中开辟一块空间并且有他的地址，自定义方法参数中的变量在内存栈中开辟一块空间，并且也有属于他的地址，两者地址不同，主函数调用自定义方法传参的时候只是把主函数变量的值赋值给了自定义变量，并不是内存地址的改变

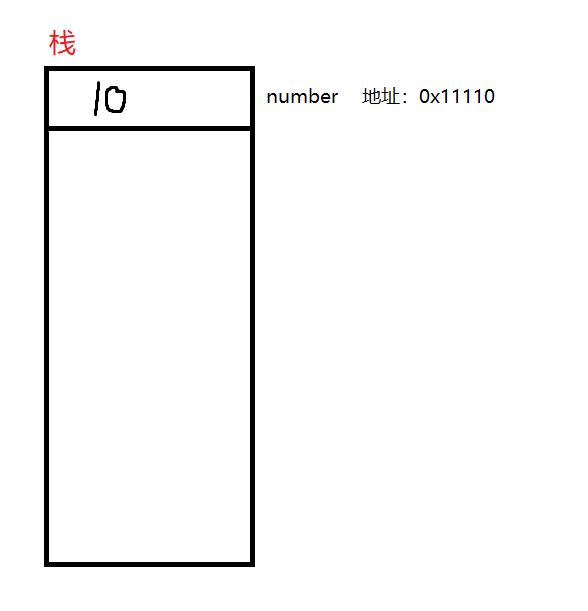
**out 和 ref 其实就是传递一个地址，这个地址是该变量在内存栈中所保存的地址，out和ref传递至的方式是一样的，所以接下来我们来做一下实验**

**先看一下值传递的代码图：**

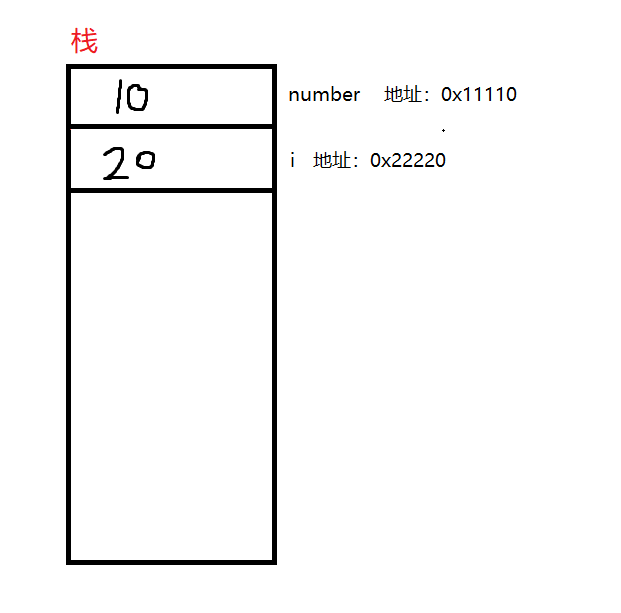
****

看一下内存空间中的情况：

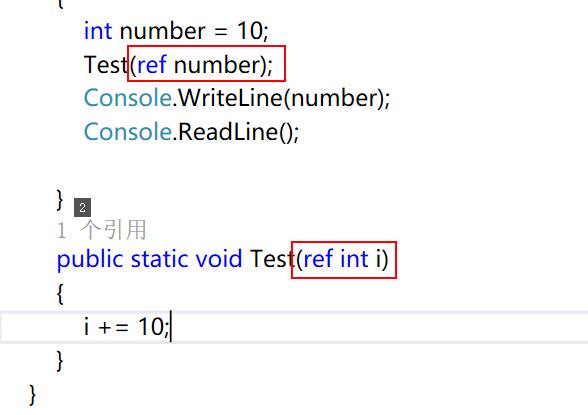
1. int number = 10：在内存的栈中开辟了一块名为空间，这块空间有属于他的地址，并且这块空间里存放的值为10；



1. 通过传参的方式把number的值传给了i，就代表在栈中的内存空间中又开辟了一块新的内存空间，并且有属于他的地址，里面存放的值为10，这时候i+=10；所以值就变成了20

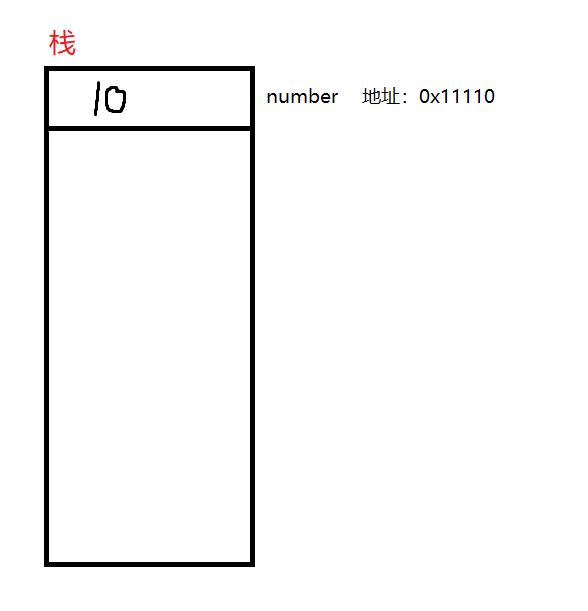


**再看一下加了out和ref变成引用传递后的代码图：**

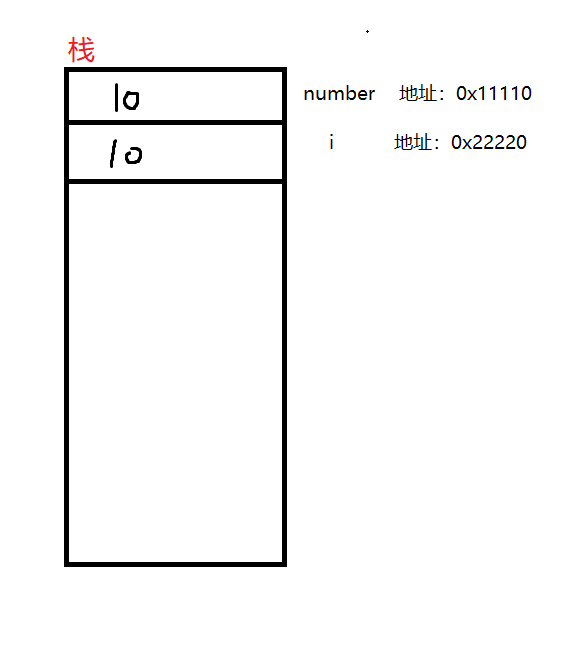


看一下内存空间中的情况：

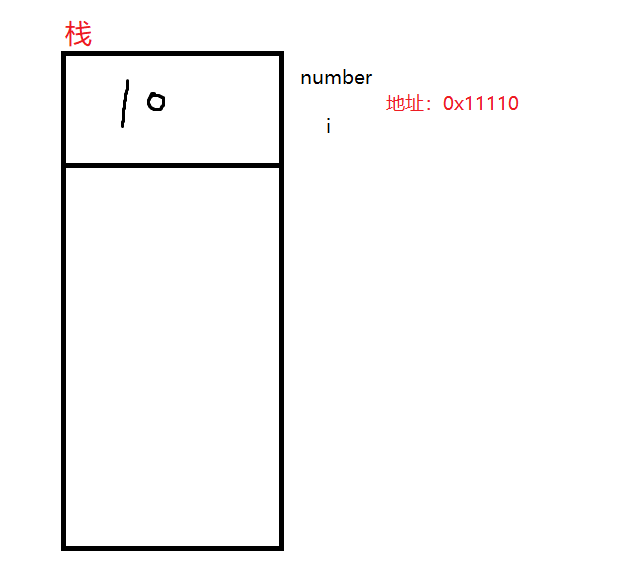
1. int number = 10；在内存的栈中开辟了一块新的内存空间，里面存放的值为10，并且有属于他的地址，变量名为 number



1. 在方法的参数里我们定义了int i ，在不加out和ref的时候是在内存的栈中又开辟了一块新的内存空间，并且有属于他的地址，变量名为i，值为10



1. 一旦我们加了out和ref就变成引用传递了，也就是把number栈中的地址给了i，他们这时候就变成了共用同一块内存空间，所以这时候改变了i的值 number 也会随之改变，因为他们共用一块内存空间中的地址



1. 引用类型也是同理，当我们在带有引用类型的参数中使用ref/out，它也会把引用类型在内存栈中所存放的地址传递过去，我们需要注意的另一点是，引用类型的传递和赋值都是传递的该变量在堆中存放数据的地址，所以当我们在引用类型里使用ref/out的时候，调用函数中的引用类型的参数变量和主函数中的引用类型的变量其实是变成一个相同的变量了（内存栈和内存堆的地址相同）