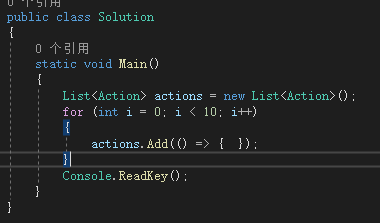
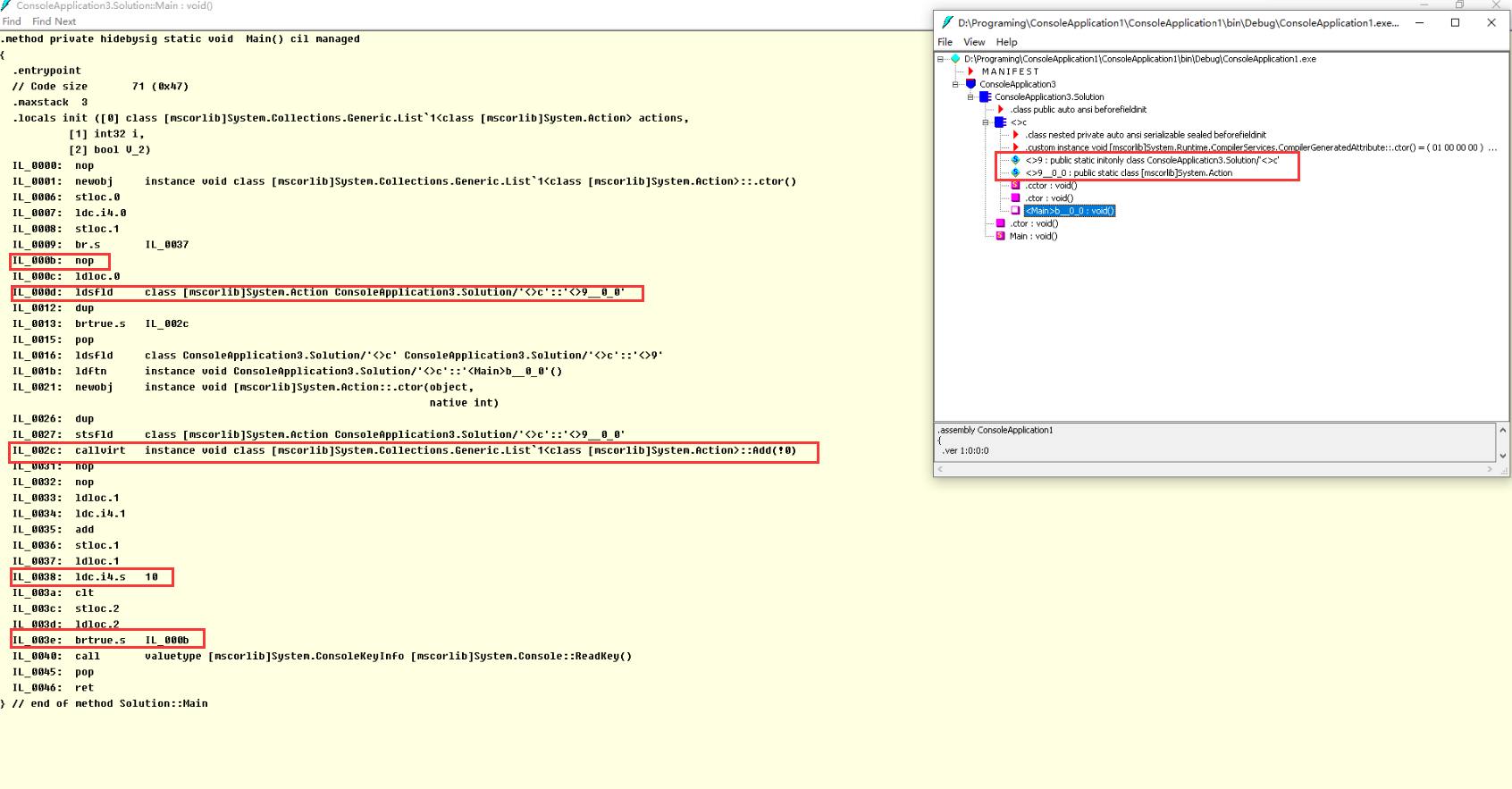
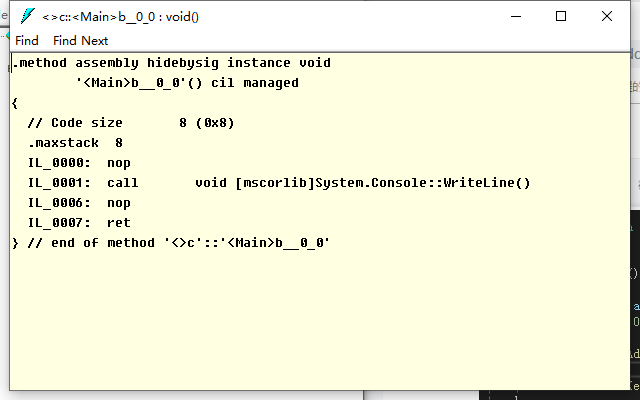
先看一个最简单的闭包

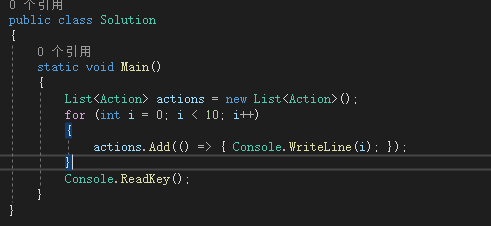


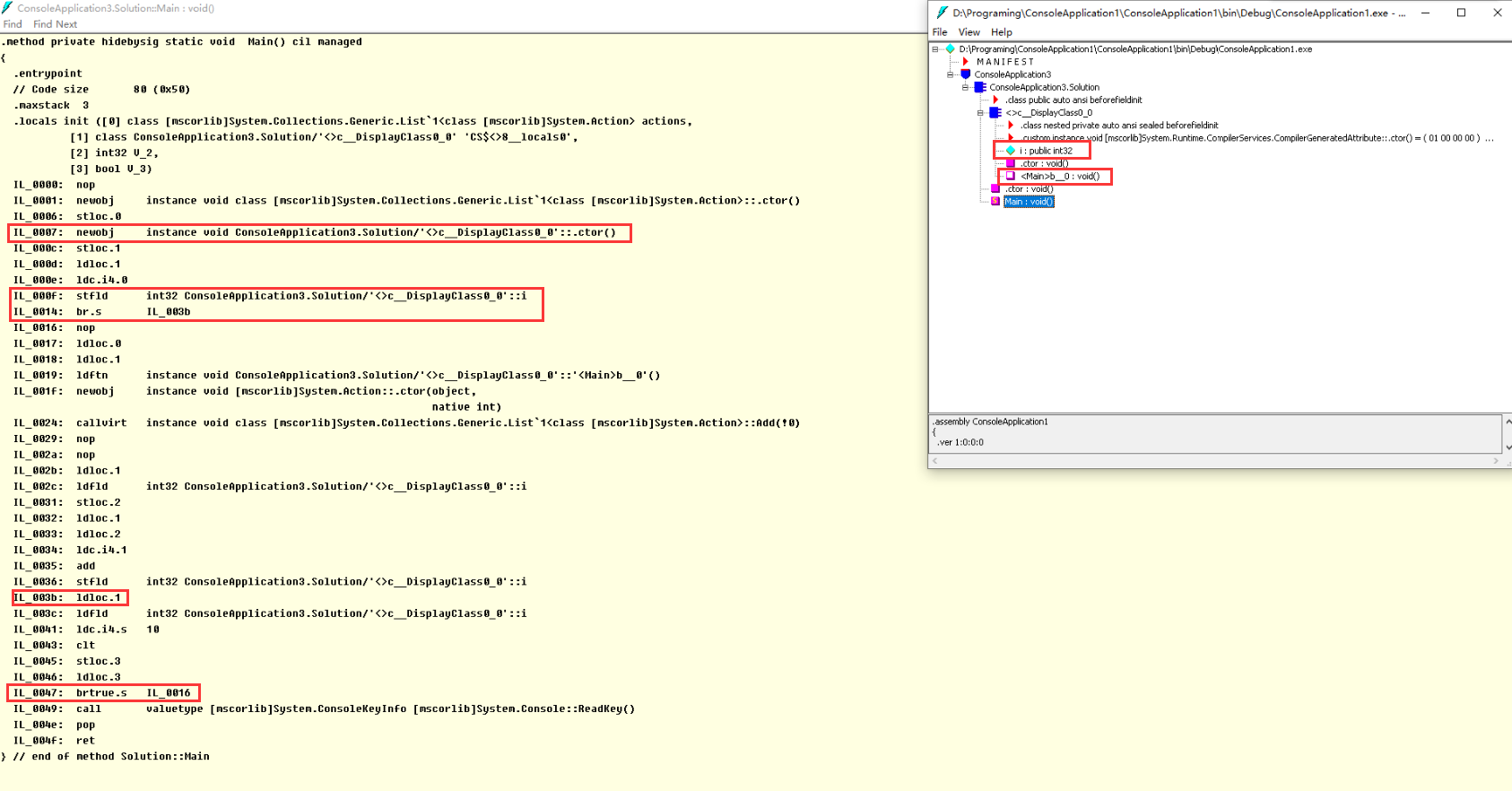


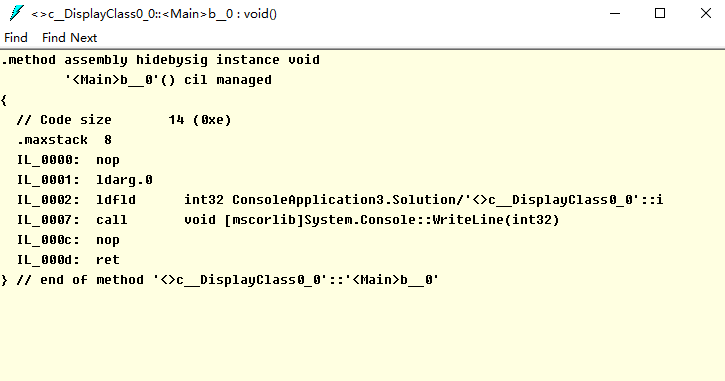


实际上这个语法糖C#会生成内部静态类，然后在循环判断结束，会先去判断这个静态变量是否为空，为空则会用内部静态类的'<Main>b\_\_0\_0'生成一个action实例，将实例赋值给静态变量Action;

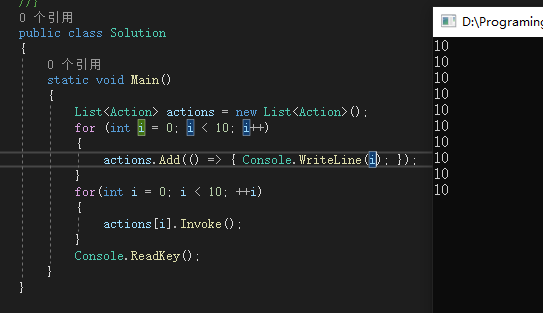
可以猜猜当顺序执行下列action时的输出是什么



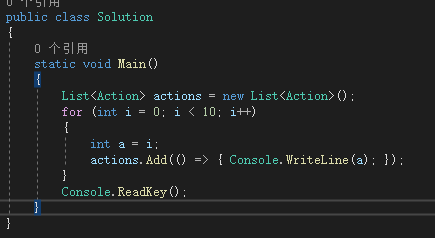




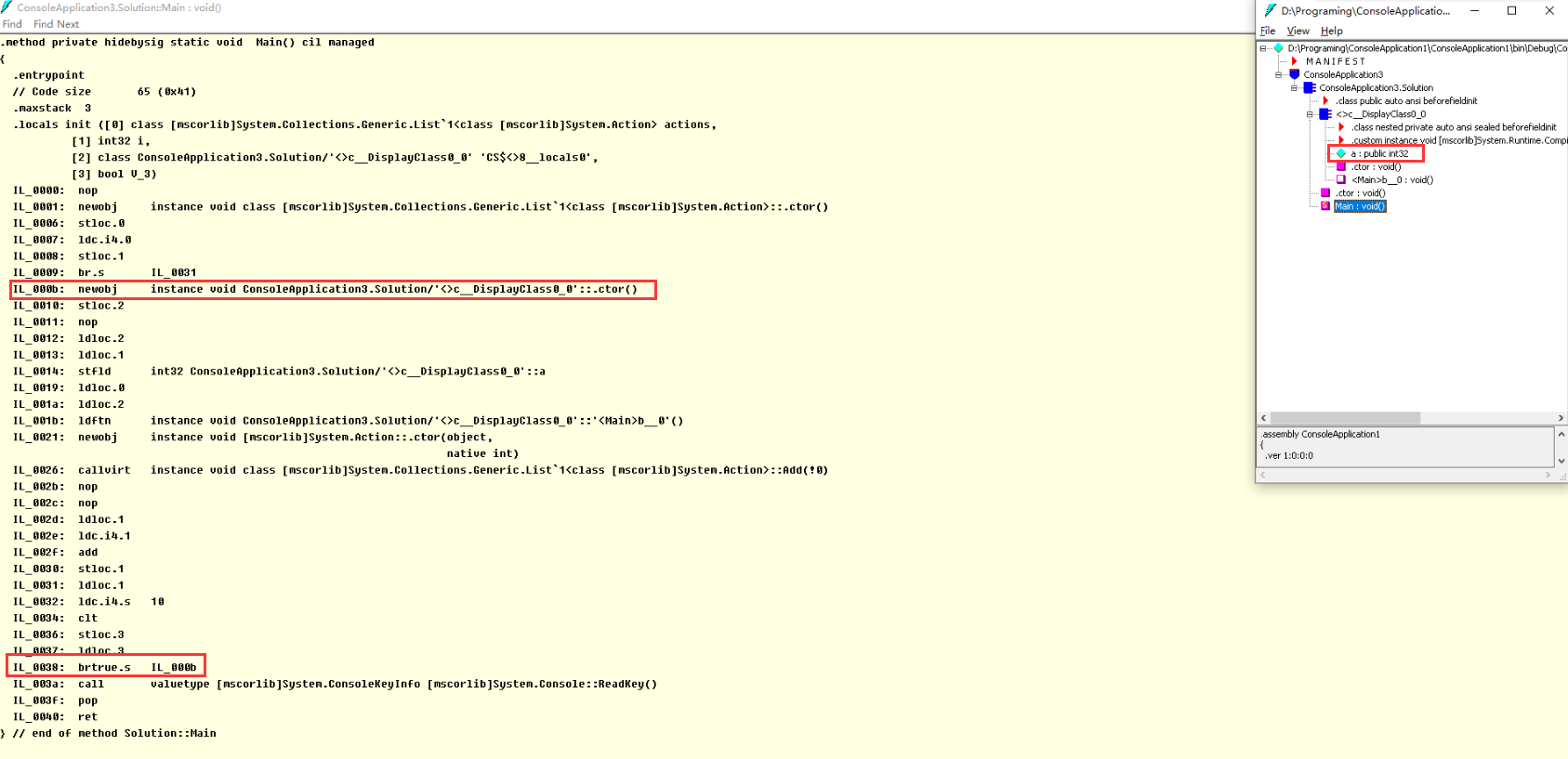
可以看到这个闭包类是在循环外部生成的，所以所有action指向同一个实例的同一个方法，故他们的输出是相同的

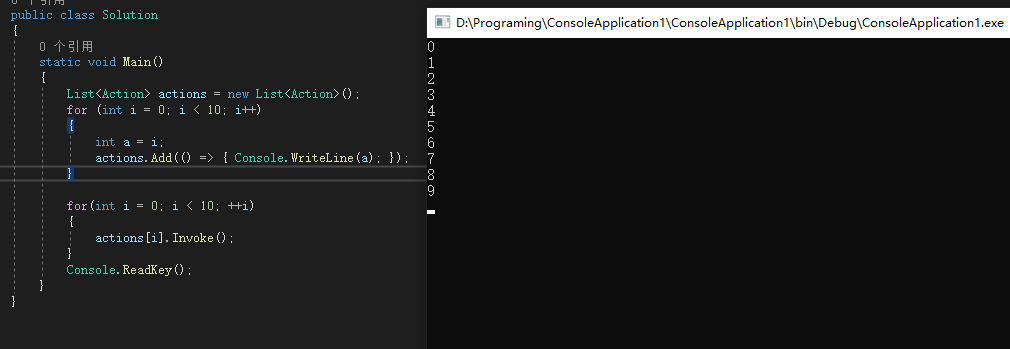


下面是能够正常输出0,1,2,3,4,5,6,7,8,9的经典写法



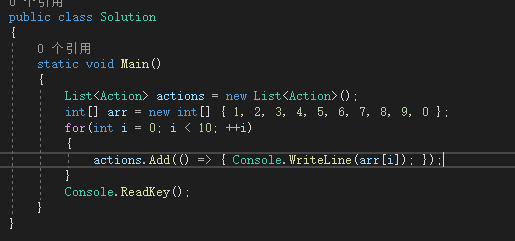
可以看到只要生成10个实例，就能分别输出1-9了；

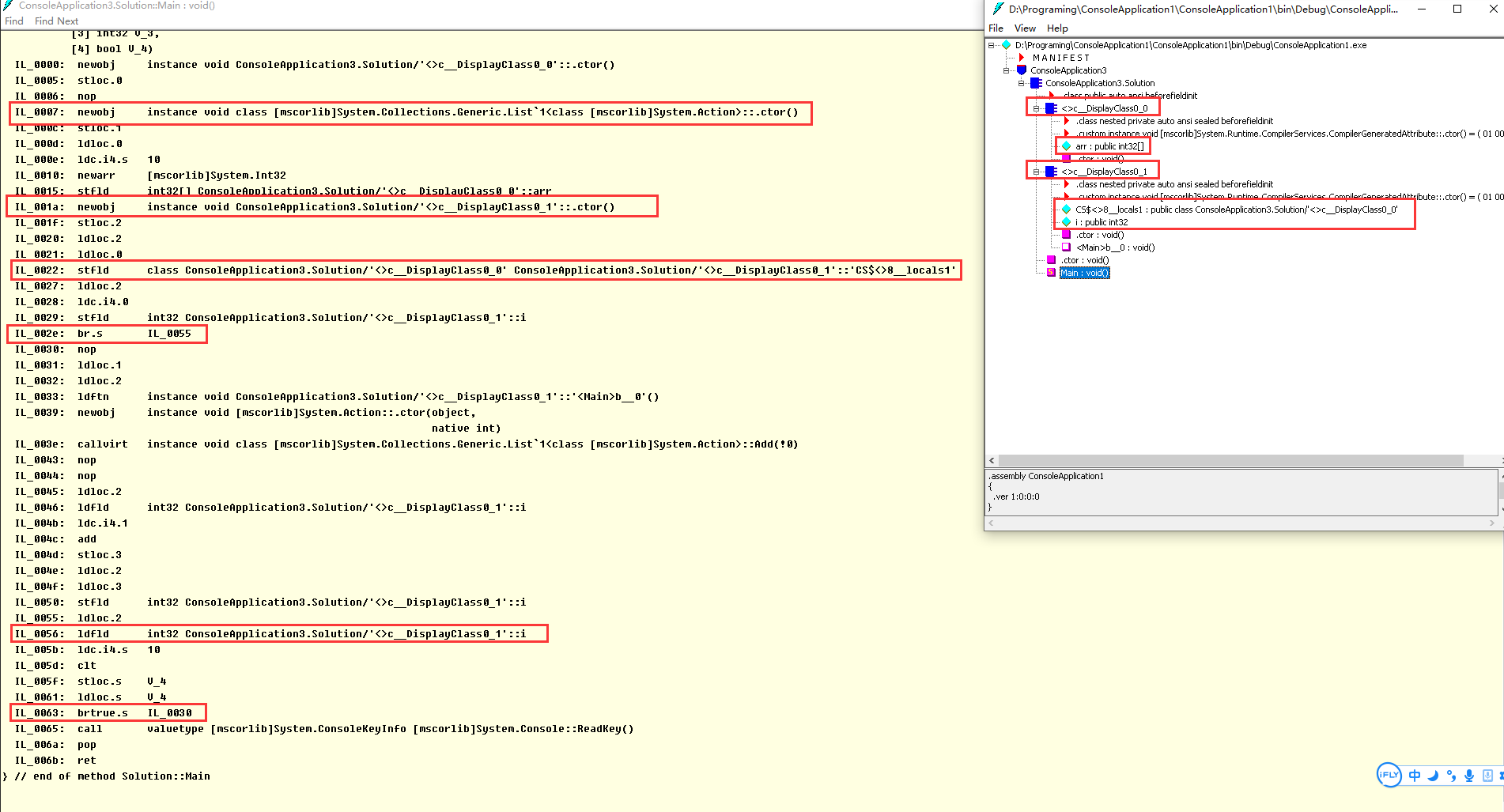




其他的发现：

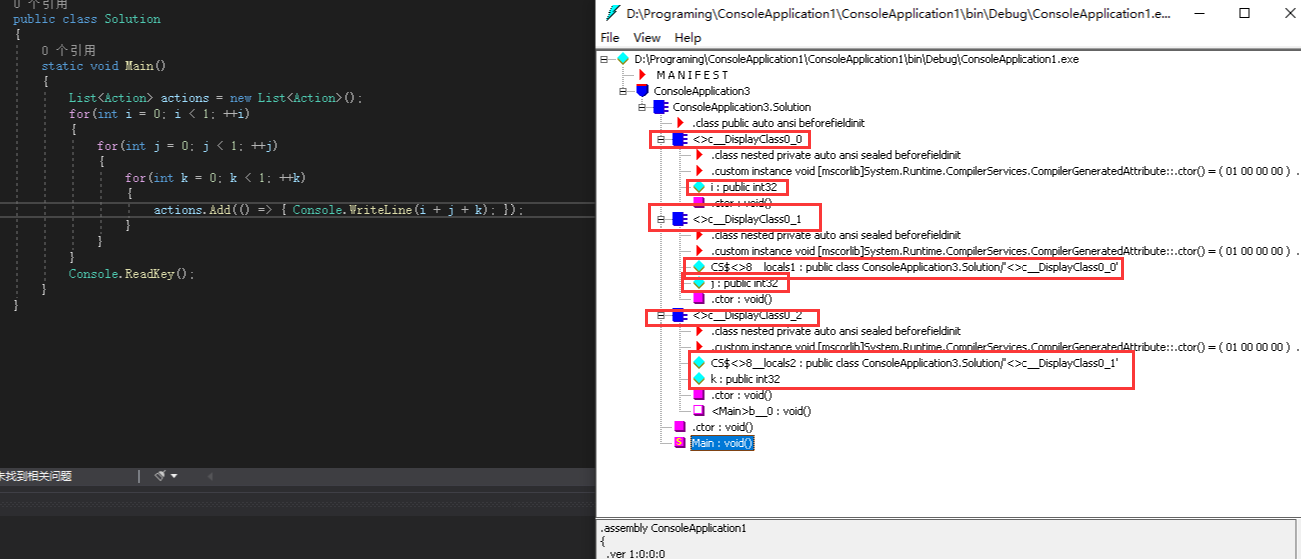
如果同时使用不同作用域变量的发现，会发现生成了两个闭包类；





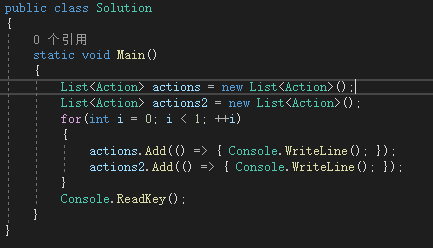


进一步验证想法：

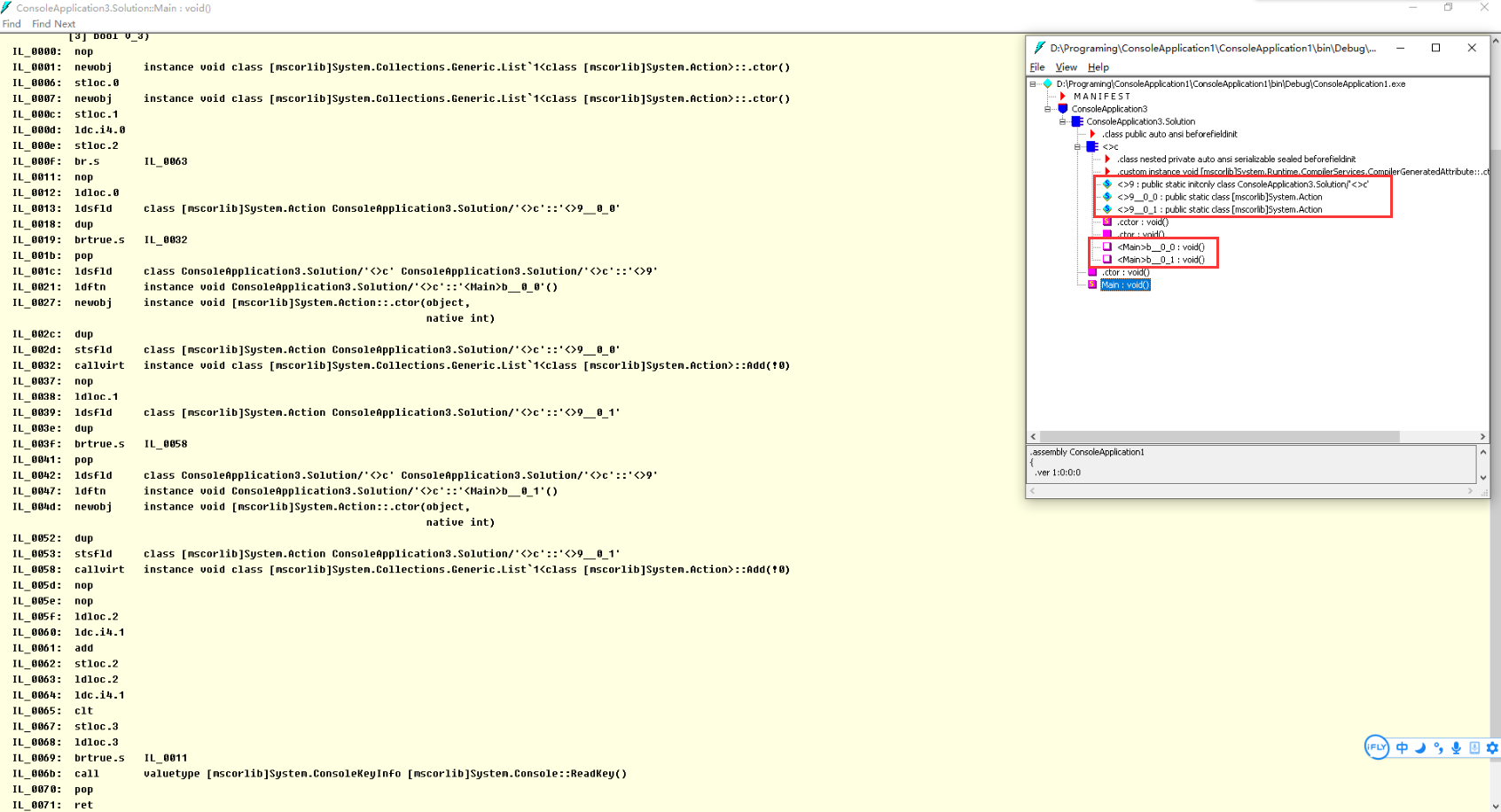


IL代码没太大意义，主要是看内部实现，3个不同生命周期的变量被闭包，生成了3个闭包类；

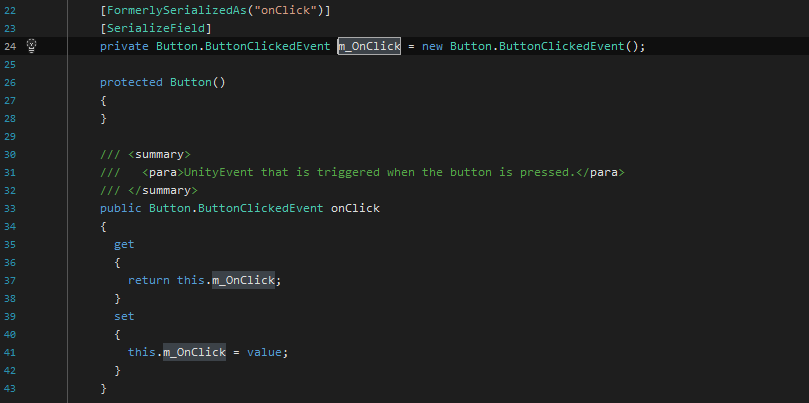
一个更有意思的发现，不妨猜猜下面代码会生成几个闭包类，



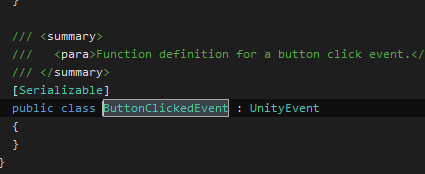
其实只会生成1个，但是另一个发现，虽然这两个闭包的内容完全一直，但是还是会生成两个Method



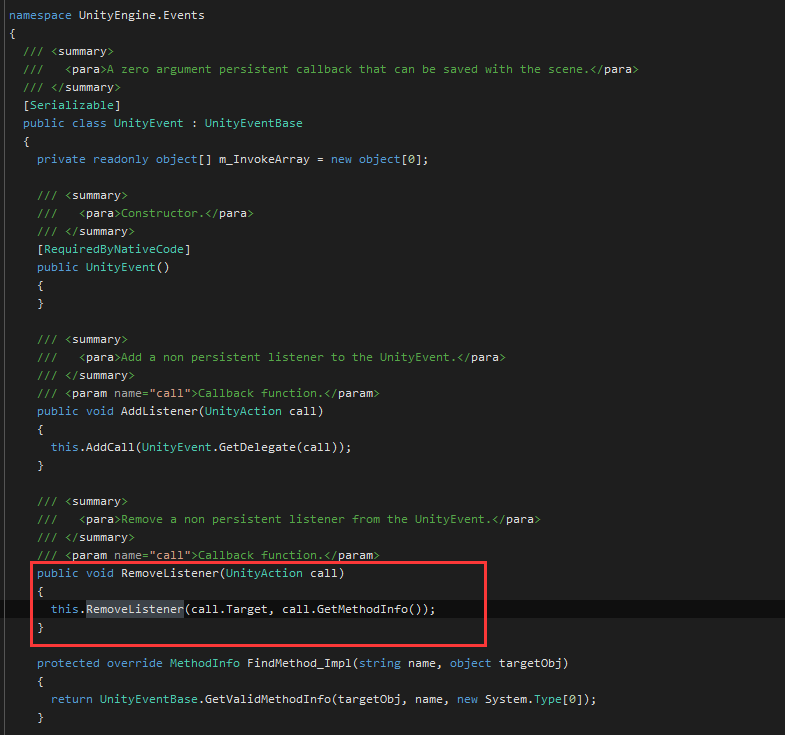
那么转到Unity，先看看Unity里Button的OnClick事件的定义；



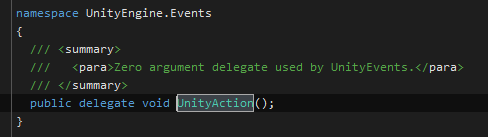
继承自UnityEvent；



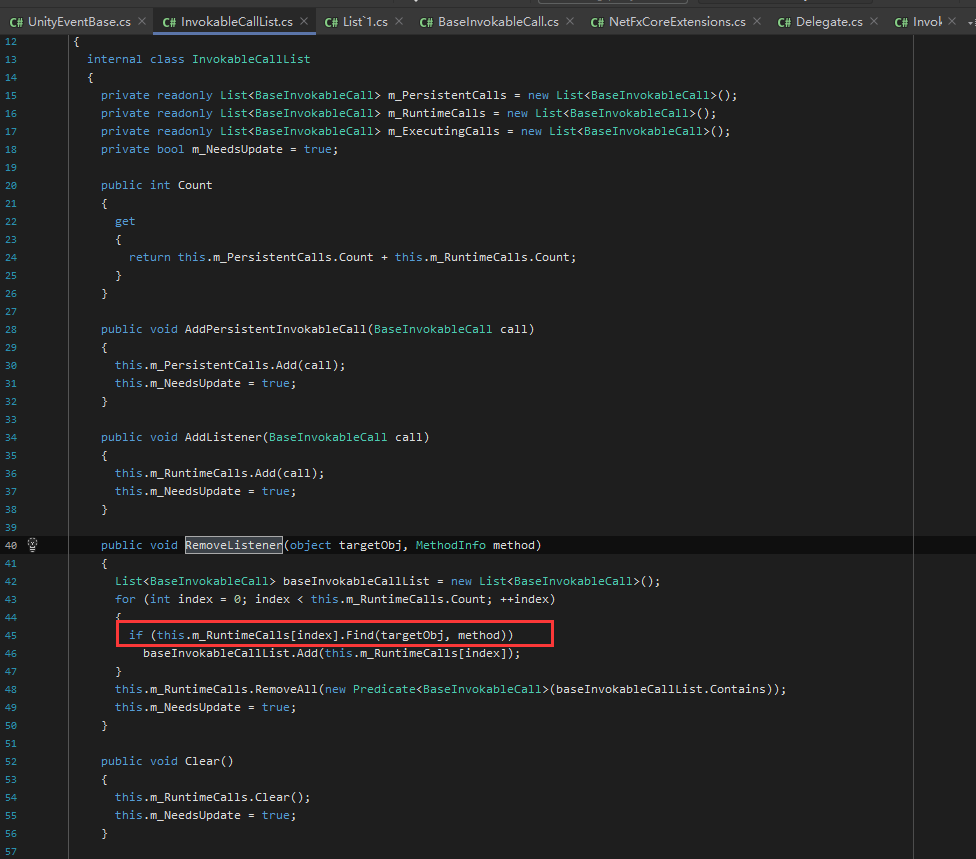
主要看RemoveLister的逻辑;



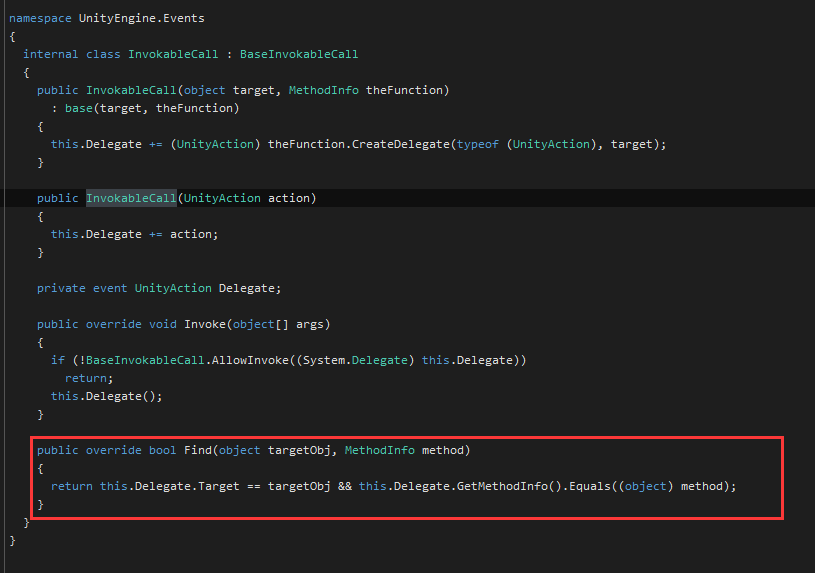
先看看UnityAction，是一个无参无返回值的委托；

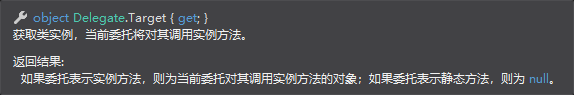


再看RemoveListener的实现，



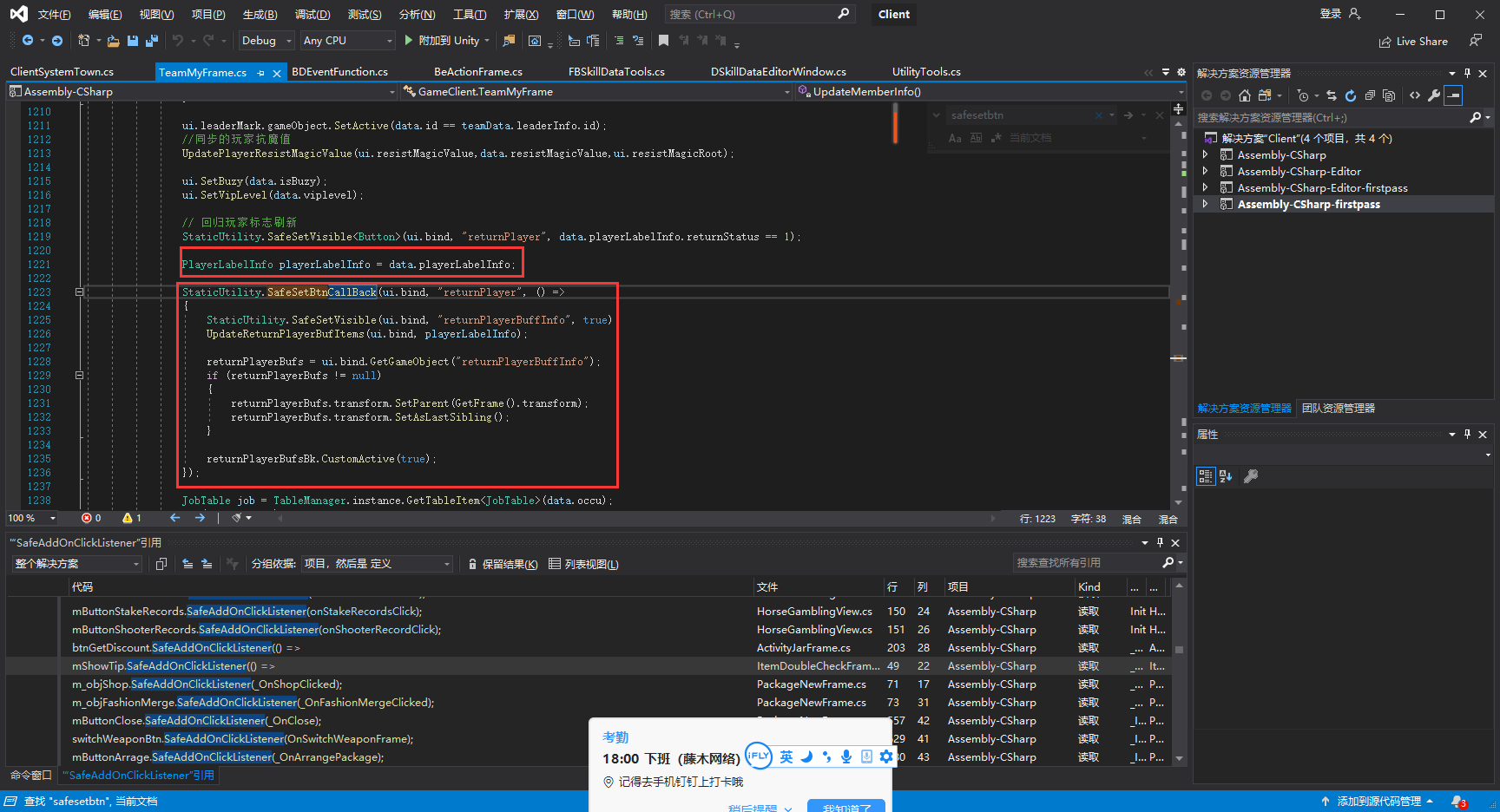
是靠,比较委托的实例是否相同&&比较委托方法是否相同,来判断是否移除事件；



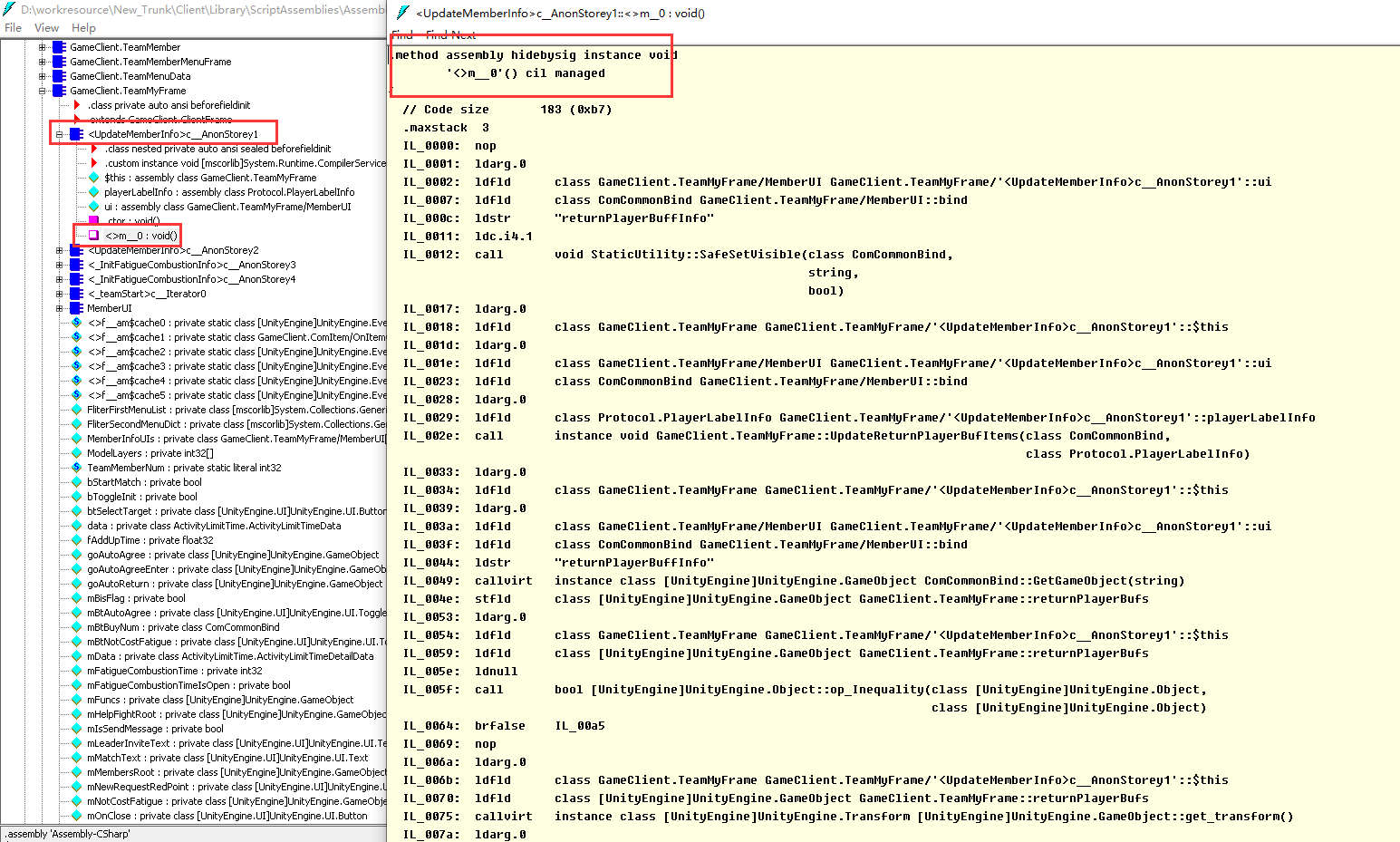


如果是闭包的写法，实际上是表示两个闭包类的实例是否一致，以及是否是同一个闭包类的同一个函数；

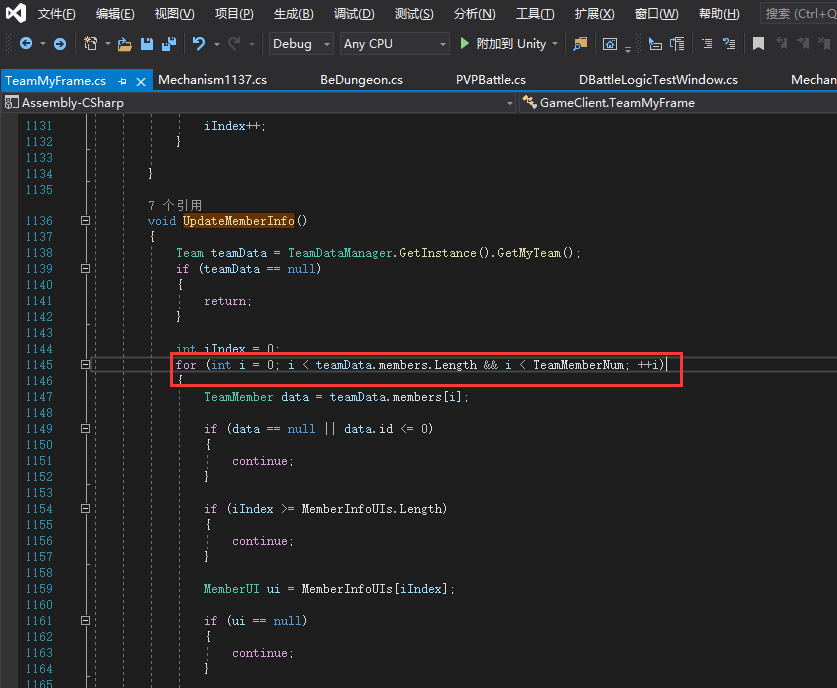
出事的代码：

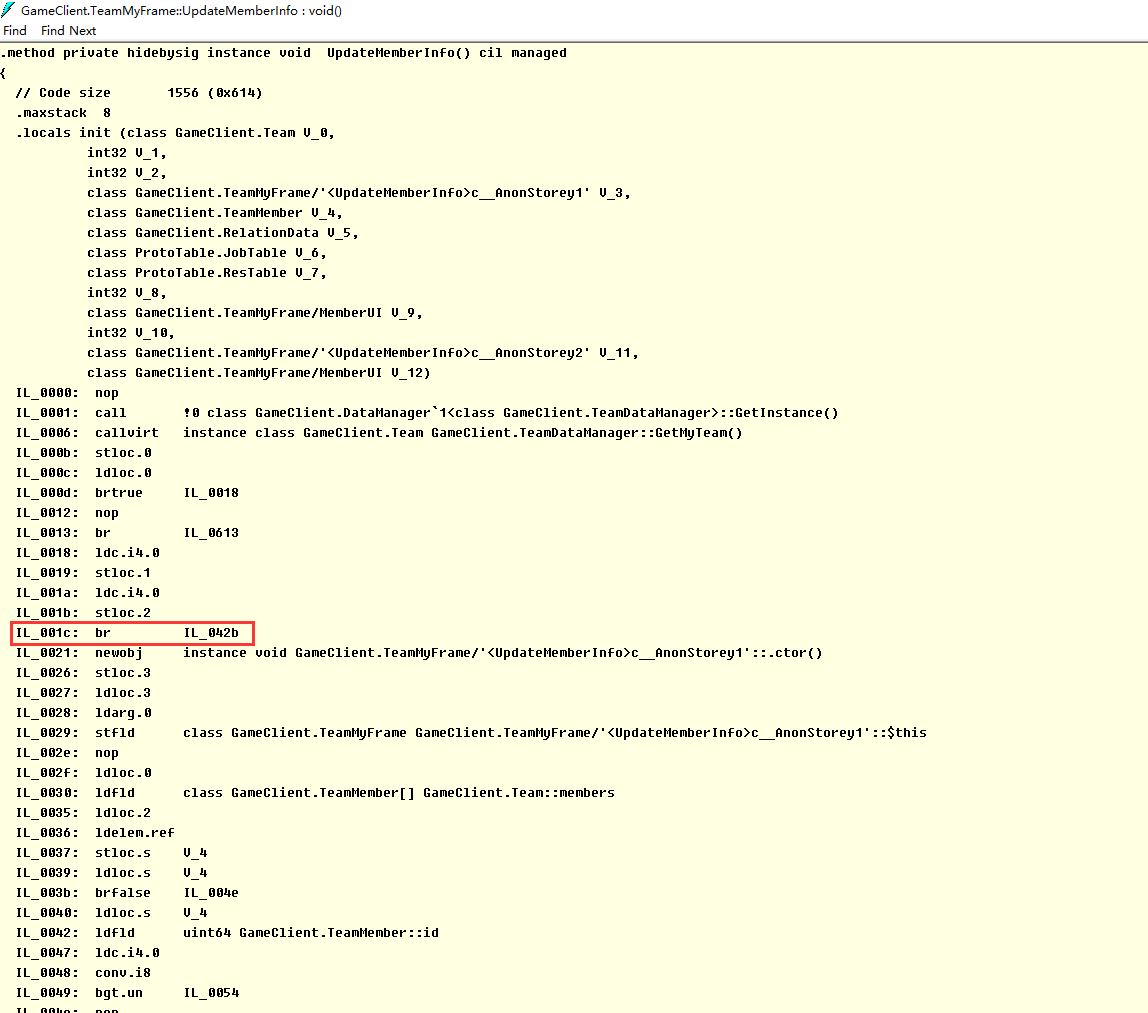


首先看一下这个地方确实会产生闭包类；

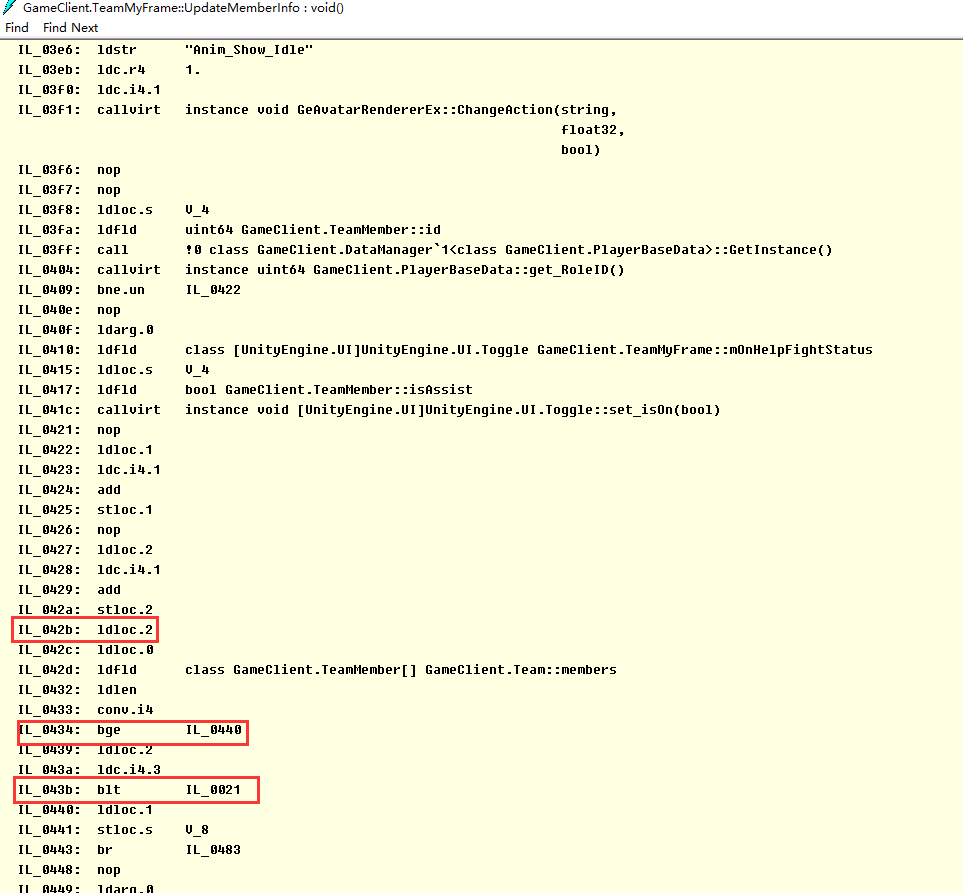


这个闭包类是在for循环内部生成实例的，先看到For循环部分

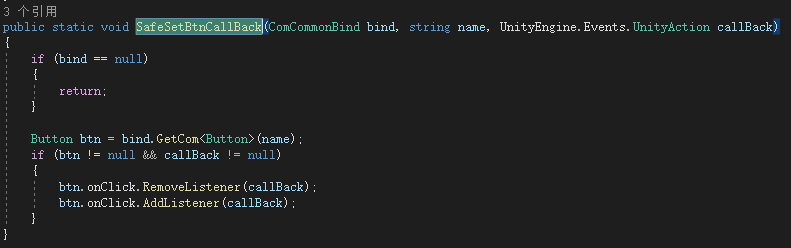




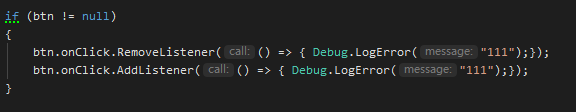
这块IL代码正好对应For循环2个判断条件，如果都成立则回到IL\_0021，直接newobj一个闭包实例，也就是说每一次都生成一个新的实例；

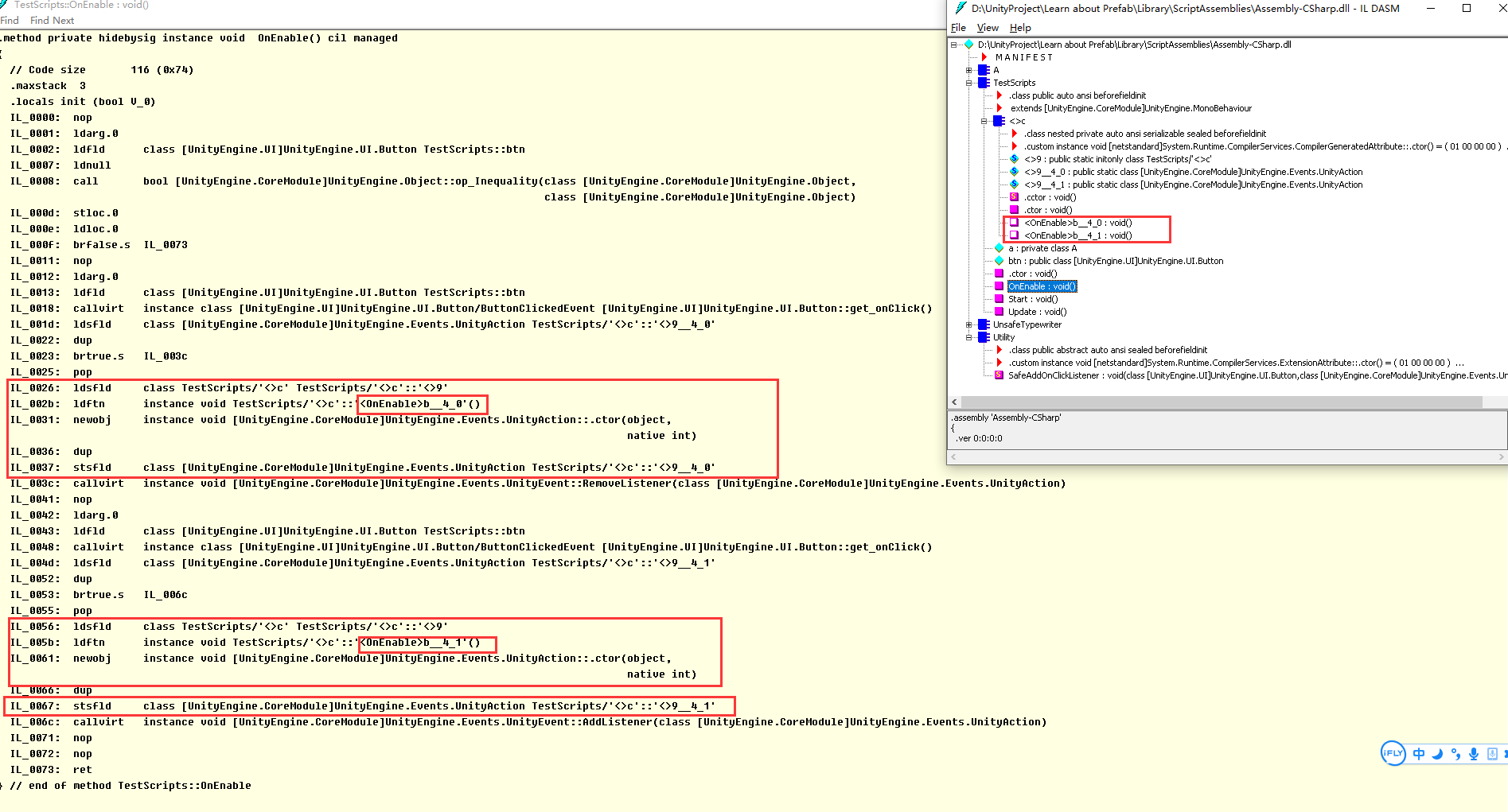


再看移除监听事件的代码，为什么说地方移除事件移除不掉，回到之前的两个判断条件，1.是否是同一个闭包实例，2.是否是同一个方法。第一个条件满足不了，所以这个移除事件是移不掉的；



最后再补充一种错误的写法；





可以看到RemoveListener对应'<>c'::'<OnEnable>b\_\_4\_0'()方法而AddListerner对应'<>c'::'<OnEnable>b\_\_4\_1'()方法。对应之前的条件2.是否是同一个方法，不是，所以也没办法移除事件。

总结就是，移除事件的条件有2个，是否是同一个对象实例，以及是否是同一个方法；而产生问题的原因主要是，闭包时，对局部变量的引用，会导致多次生成闭包实例；