首先假设我们有如下的一个扩展方法：

```
    public static void LockExec<T>(this T obj, Action<T> action) where T : class
    {
        lock (obj)
        {
            action(obj);
        }
    }
```

我们用这个扩展方法写下了如下的代码：

```
    static void TestInsert(bool canInsert, string insertValue)
    {
        var list1 = new List<string>();
        list1.LockExec(
                    (list) =>
                    {
                        if (canInsert && !list.Contains(insertValue))
                        {
                            list.Add(insertValue);
                        }
                    });
    }
```

在这个例子中的Lambda表达式中从表面看来只有list是Action<T>传进来的变量，canInsert，insertValue对于Lamba表达式来时都是外部变量，为什么这个表达式还能通过编译且正确运行呢？其实这是c#编译器帮我们做的一个工作，c#编译器将上面的代码编译成下边的代码，编译器为我们生成了一个内部类，来保存Lambda表达式引用的外部变量值，并将这个内部类的实例方法传递给Action<T>，这样这个Action<T>对象的Target和Mothed属性就都有值了，而且这个Action<T>的方法调用将采用实例方法调用，也就是说会有this指针。

```
static void TestInsert(bool canInsert, string insertValue)
{
    var list1 = new List<string>();
    var innerClass = new InnerClass { InnerProperty1 = canInsert, InnerProperty2 = ins
    list1.LockExec(innerClass.InnerMothed);
}

internal class InnerClass
{
    public bool InnerProperty1
    {
        get; set;
    }
```

```
        public string InnerProperty2
        {
            get; set;
        }

        public void InnerMothed(List<string> list)
        {
            if (InnerProperty1 && !list.Contains(InnerProperty2))
            {
                list.Add(InnerProperty2);
            }
        }
    }
```