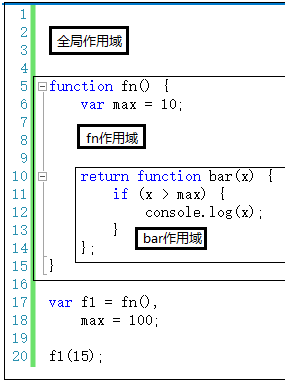
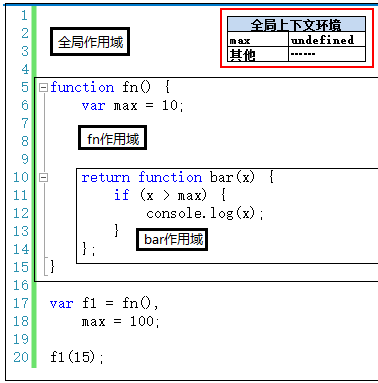
**深入理解javascript原型和闭包（15）——闭包**

<http://www.cnblogs.com/wangfupeng1988/p/3994065.html>

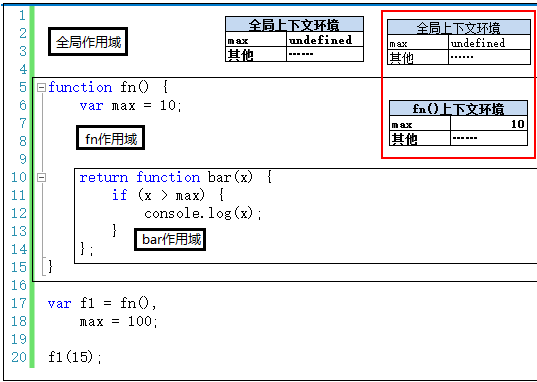
分析一下下面的代码：



第一步，代码执行前生成全局上下文环境，并在执行时对其中的变量进行赋值。此时全局上下文环境是活动状态。



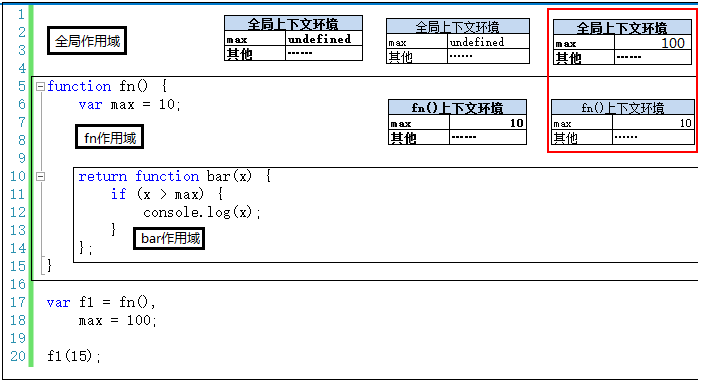
第二步，执行第17行代码时，调用fn()，产生fn()执行上下文环境，压栈，并设置为活动状态。



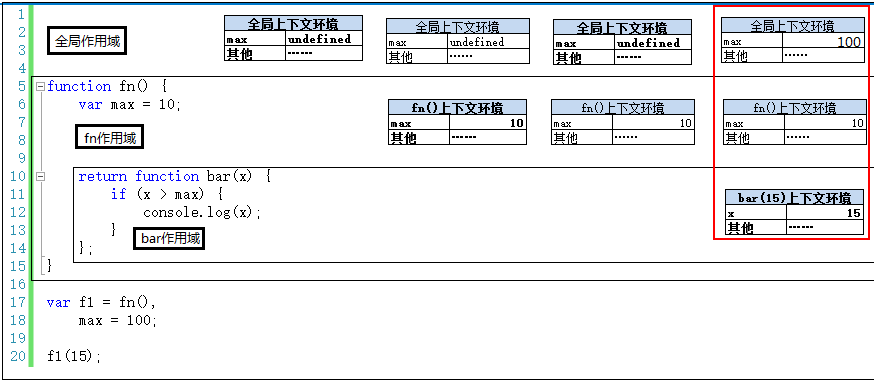
第三步，执行完第17行，fn()调用完成。按理说应该销毁掉fn()的执行上下文环境，但是这里不能这么做。注意，重点来了：因为执行fn()时，返回的是一个函数。函数的特别之处在于可以创建一个独立的作用域。而正巧合的是，返回的这个函数体中，还有一个自由变量max要引用fn作用域下的fn()上下文环境中的max。因此，这个max不能被销毁，销毁了之后bar函数中的max就找不到值了。

因此，这里的fn()上下文环境不能被销毁，还依然存在与执行上下文栈中。

——即，执行到第18行时，全局上下文环境将变为活动状态，但是fn()上下文环境依然会在执行上下文栈中。另外，执行完第18行，全局上下文环境中的max被赋值为100。如下图：



第四步，执行到第20行，执行f1(15)，即执行bar(15)，创建bar(15)上下文环境，并将其设置为活动状态。



执行bar(15)时，max是自由变量，需要向创建bar函数的作用域中查找，找到了max的值为10。这个过程在作用域链一节已经讲过。

这里的重点就在于，创建bar函数是在执行fn()时创建的。fn()早就执行结束了，但是fn()执行上下文环境还存在与栈中，因此bar(15)时，max可以查找到。如果fn()上下文环境销毁了，那么max就找不到了。

使用闭包会增加内容开销，现在很明显了吧！

第五步，执行完20行就是上下文环境的销毁过程，这里就不再赘述了。