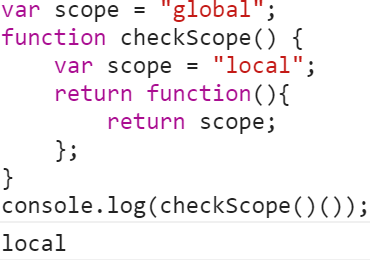
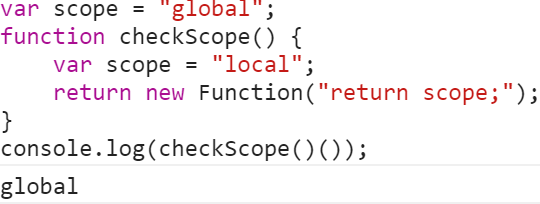
**第十章 JS作用域及执行上下文**

**JS作用域及其特点**

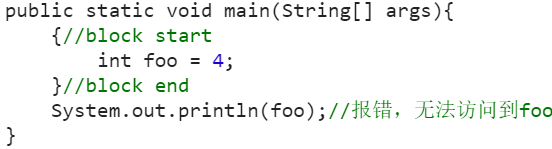
**什么是作用域**- 作用域就是变量与函数的可访问范围（变量生效的区域范围，即在何处可以被访问到）  
- 作用域控制着变量与函数的可见性和生命周期，它也是根据名称查找变量的一套规则

**JS作用域特点（词法作用域）**- JS采用的是词法作用域（静态性），这种静态结构决定了一个变量的作用域  
- 词法作用域不会被函数从哪里调用等因素影响，与调用形式无关（体现了静态性）

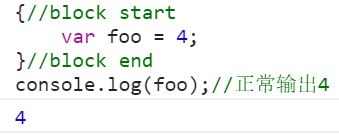
- 通过new Function创建的函数对象不一定遵从静态词法作用域  
- 对比下边两个例子（通过不同形式定义的函数对象，访问到的scope的区别）

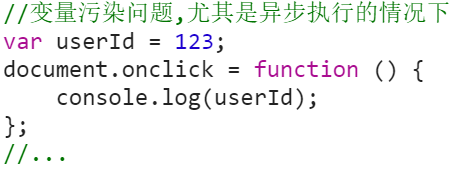
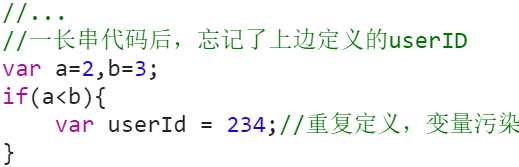
* 大多数语言都有块级作用域  
  **- 变量“存活”在最近的代码块中，比如Java中**



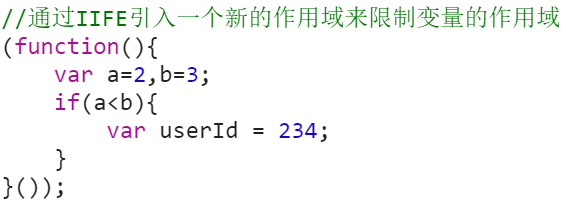
**JS（ES5）采用的是函数级作用域，没有块作用域**



**无块作用域的问题（变量污染、变量共享问题）**

解决方案IIFE（更多内容参见IIFE部分）



**JS执行上下文与调用栈（Call Stack）**

**执行上下文（context，举例生活中的上下文环境）**- 执行上下文指代码执行时的上下文环境（包括局部变量、相关的函数、相关自由变量等）  
- JS运行时会产生多个执行上下文，处于活动状态的执行上下文环境只有一个

断点：实例中执行到断点时，就有一个当前断点所对应的执行上下文（对应当前执行环境）

**调用栈（Call Stack）**- 代码执行时JS引擎会以栈的方式来处理和追踪函数调用（函数调用栈 Call Stack）  
- 栈底对应的是全局上下文环境，而栈顶对应的是当前正在执行的上下文环境

**作用域链与执行上下文**

* 执行时，当前执行上下文，对应一个作用域链环境来管理和解析变量和函数（动态性）  
  - 变量查找按照由内到外的顺序（遵循词法作用域），直到完成查找，若未查询到则报错  
  - 当函数执行结束，运行期上下文被销毁，此作用域链环境也随之被释放

**环境：变量的管理**- 当程序运行到变量所在的作用域时，变量被创建，此时需要一个存储的空间  
- JS中提供存储空间的数据结构被称为环境，每个函数都有自己的执行环境  
- 每个执行环境都有一个与之关联的变量对象，环境中所有变量和函数都保存在此对象中  
- Web浏览器中，全局执行环境为window对象

**作用域链（在 ECMA262 中的解释，涉及到内部属性）**- 任何执行上下文时刻的作用域，都是由作用域链 (scope chain) 来实现。 在一个函数被定义的时候，会将它定义时候的 scope chain 链接到这个函数对象的[[scope]]属性。 在一个函数对象被调用的时候，会创建一个活动对象 (也就是一个对象，然后对于每一个函数的形参，都命名为该活动对象的命名属性，然后将这个活动对象做为此时的作用域链 (scope chain) 最前端， 并将这个函数对象的 [[scope]] 加入到 scope chain 中