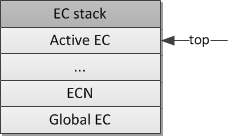
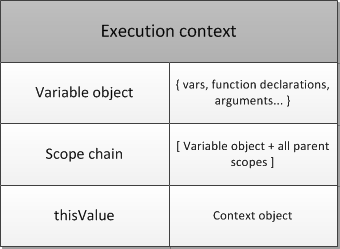
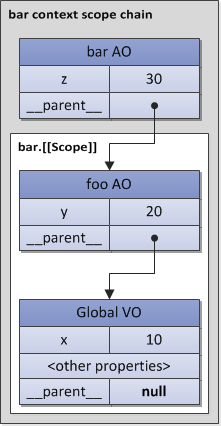
1. JS只有3中代码。Global、function、eval。每一种代码的执行都要依赖自生的执行上下文。
2. 所有JS程序的执行都可以看作是一个执行上下文堆栈（EC Stack）。堆栈的顶部就是正在执行的上下文。  
   
3. 一段程序开始时，首先会进入全局执行上下文。
4. 执行上下文可以理解为一个对象。包括变量对象，this指针，以及作用域链。
5. 
6. 变量对象：他是上下文关联的特殊对象，用于存储被定义在上下文中的变量（variables）和函数声明（function declarations）。  
   当函数被激活（而不是创建）时，活动对象（activation object）就被创建了。
7. 作用域链：一个对象列表（list of objects），用于检索上下文代码中出现的标识符。
8. 作用域链=活动对象+[[Scope]]
9. This是和执行上下文息息相关的一个对象。可以被称为上下文对象。  
   this是执行上下文的一个属性，而不是某个变量对象的属性。很重要，this是没有一个类似搜寻变量的过程。当你在代码中使用了this，this是直接从执行的上下文中获得的，而不会从作用域链中搜寻。
10. 变量对象  
    进入执行上下文时，VO已经包括：  
    1.函数的所有形参，arguments  
    2.所有的函数声明  
    3.所有的变量声明  
    注意：这个函数本身的声明不保存在自己的VO中，而是在父上下文的VO中，之所以能调用也是因为闭包。
11. \_parent\_  
    有些浏览器实现了\_parent\_这个函数的属性,指向的是父级上下文的VO。
12. [[Scope]]  
    [[Scope]]是所有父变量对象的层级链，处于当权函数上下文之上，在函数创建时就存于其中。永远永远。与作用域链相比，[[scope]]是一个属性而不是上下文。  
    作用域链=AO/VO+[[scope]]
13. 闭包  
    闭包的根源就在函数的[[scope]]上。同一个上下文中共用一个[[scope]]属性。  
    Function（）构造器创建的函数，其[[scope]]只包含全局对象。  
    闭包：  
    1.理论上将，所有的函数都是闭包。因为他们在创建的时候就已经保存上层上下文的数据。  
    2.实践角度，以下函数才是闭包：  
     1.及时创建他的上下文销毁，他依然存在（如内部函数从父函数返回）  
     2.在代码中使用了自由变量
14. 闭包的实际作用  
    应用的作用域不会被垃圾回收处理掉，但是不合理的使用闭包是会造成内存消耗的。  
    封装一个私有变量。