**[JavaScript内存回收机制深入解读](http://www.cnblogs.com/jenry/archive/2011/04/20/2021920.html)**

JavaScript语言是一门优秀的脚本语言.其中包含脚本语言的灵活性外还拥有许多高级语言的特性.例如充许构建和实例化一个对象,垃圾回

收机制(GC:Garbage   
Collecation).通常我们使用new创建对象,GC负责回收对象占用内存区域.因此了解GC,可以加深对JavaScript垃圾回收

机制的理解。

**1.用局部变量和全局变量解释GC**

GC在回收内存时,首先会判断该对象是否被其它对象引用.在确定没有其它对象引用便释放该对象内存区域.因此如何确定对象不再被引用是

GC的关键所在.

1. <script>
2. **function** aa(){
3. **this**.rr = "弹窗";
4. }
5. **function** bb(){
6. **this**.rr = "弹窗";
7. }
9. **var** b1;
10. **function** cc(){
11. **var** a1 = **new** aa();
12. b1 = **new** bb();
13. **return** b1;
14. }
16. cc();
17. alert(b1.rr)
18. </script>

如上代码中，执行完cc()后a1被回收了,此后我们可以通过b1.rr弹出文字窗口.在一些基础书籍中解释为:a1为局部变量,b1是全局变量.局部

变量执行完后会被GC回收.但不全是这样,如下代码:

1. <script>
2. **function** aa(){
3. **this**.rr = "弹窗";
4. }
5. **function** bb(){
6. **this**.rr = "弹窗";
7. }
9. **function** cc(){
10. **var** a1 = **new** aa();
11. **var** b1 = **new** bb();
12. **return** b1;
13. }
15. **var** b1 = cc();
16. alert(b1.rr);
17. </script>

此时cc函数中的   
a1,b1都是局部变量,但仍然会弹出文字窗口.说明b1并没有被GC回收.因此JavaScript中局部变量不是所有时候都被GC回收的.

**2.抽象理解GC**

GC回收机制还需要近一步了解。在此时引入几个概念:双向链表,作用域链,活动对象(为了方便理解，简化了原文的概念

[http://softbbs.pconline.com.cn/9497825.html]) , 其中双向链表描述复杂对象的上下层级关系.   
作用域链与活动对象分别是双向链表

中的某个节点.以函数cc为例变量层级关系为:

1. window<=>cc<=>a1<=>rr
2. <=>b1<=>rr

(原文有详细解释)在执行cc()方法时，内存中变量的引用关系如上图,文字解释如下:

window的活动对象包括cc,假设window是顶级对象(因为运行中不会被回收)

cc的活动对象包括a1和b1,其作用域链是window

a1的活动对象包括rr,其作用域链是cc

b1的活动对象包括rr,其作用域链是cc

执行cc()时，cc的执行环境会创建一个活动对象和一个作用域链.其局部变量a1,b1都会挂在cc的活动对象中.当cc()执行完毕后，执行环境

会尝试回收活动对象占用的内存.但因局部变量b1 通过return   
b1,为其增加了一条作用域链:window<=>b1<=>rr,所以GC停止对b1回收.

因此如果想将一个局部变量/函数提升为全局的，为其增加一条作用域链就OK了。

同时控制好对象的作用域链也变得重要了.因作用域链会意外导致GC无法回收目标对象.例如:

1. <SCRIPT LANGUAGE="JavaScript">
2. <!--
3. //猫
4. **function** cat(name){
5. **var** zhuren ;
6. **this**.name = name;
8. //设置主人
9. **this**.addZhuRen = **function**(zr){
10. zhuren = zr;
11. }
13. **this**.getZhuRen = **function**(){
14. **return** zhuren;
15. }
16. }
18. //主人
19. **function** zhuren(name){
20. **this**.name = name;
21. }
23. //创建主人:
24. **var** zr = **new** zhuren("zhangsan");
25. //创建猫
26. **var** cat1 = **new** cat("asan");
27. //设置该猫的主人
28. cat1.addZhuRen(zr);
29. //释放主人
30. zr = **null** ;
31. //此处还存在对主人对象的引用
32. alert(cat1.getZhuRen().name)
33. //-->
34. </SCRIPT>