JavaScript中的变量、作用域和内存

# 基本类型和引用类型

5 种基本数据类型：Undefined、Null、Boolean、Number 和String。这5 种基本数据类型是按值访问的，因为可以操作保存在变量中的实际的值。引用类型的值是保存在内存中的对象。与其他语言不同，JavaScript 不允许直接访问内存中的位置，也就是说不能直接操作对象的内存空间。javaScript是通过引用（地址值）来访问对象的内存空间。

* 基本内存值保存在栈内存中，可以直接通过变量名获取。
* 基本变量的复制过程是把值传递的过程。
* 不能给基本类型的值添加属性，这样做无效。添加的属性值为undefined。
* 引用类型的值是对象的地址。引用的复制指向同一个内存空间。
* 函数传递参数的过程都是传值的：基本类型直接传递值，引用类型传递的地址。并且参数传递的时候，参数的值不会改变：基本变量的值不变，引用类型对应的地址不变，但是引用指定的对象可能改变。
* 用于检测引用类型变量的具体引用类型。Ex:alert(person instanceof Object); // 变量person 是Object 吗？

# 执行环境及作用域

## 执行环境

执行环境定义了变量或函数的作用域。每个执行环境都有一个变量环境。在web浏览器中，全局的执行环境是window对象。每个函数都有自己的执行环境。活动对象（activation object）作为变量对象。

作用域链的用途，是保证对执行环境中有权访问的所有变量和函数的有序访问。内部环境可以通过作用域链访问所有的外部环境，但外部环境不能访问内部环境中的任何变量和函数。

标识符解析是沿着作用域链一级一级地搜索标识符的过程。搜索过程始终从作用域链的前端开始，然后逐级地向后回溯，直至找到标识符为止（如果找不到标识符，通常会导致错误发生）。

## 作用域

没有块级作用域：

|  |
| --- |
| if (true) {  var color = "blue";  }  alert(color); //"blue"  函数中的变量作用域为该函数本身。  不声明而直接初始化变量:color=”blue”;会成为全局变量 |

# 内存问题

## 垃圾回收的两种方案

1) 标记清除：当变量进入环境，就将这个变量标记为“进入环境”。而当变量离开环境时，则将其标记为“离开环境”，变量失效。

2) 引用计数：当声明了一个变量并将一个引用类型值赋给该变量时，则这个值的引用次数就是1。如果同一个值又被赋给另一个变量，则该值的引用次数加1。相反，如果包含对这个值引用的变量又取得了另外一个值，则这个值的引用次数减1。当这个值的引用次数变成0 时，则说明没有办法再访问这个值了，因而就可以将其占用的内存空间回收回来。这样，当垃圾收集器下次再运行时，它就会释放那些引用次数为零的值所占用的内存。循环引用的时候会出现问题。

## 性能问题

垃圾收集器是周期性运行的，而且如果为变量分配的内存数量很可观，那么回收工作量也是相当大的。在这种情况下，确定垃圾收集的时间间隔是一个非常重要的问题。

## 管理内存

分配给Web浏览器的可用内存数量通常要比分配给桌面应用程序的少。这样做的目的主要是出于安全方面的考虑，目的是防止运行JavaScript 的网页耗尽全部系统内存而导致系统崩溃。内存限制问题不仅会影响给变量分配内存，同时还会影响调用栈以及在一个线程中能够同时执行的语句数量。为了避免这种情况，我们可以使用解除引用的方式来解决：param=null;// 解除引用的真正作用是让值脱离执行环境，以便垃圾收集器下次运行时将其回收。