# JavaScript内存管理

## 函数执行环境

每个执行函数都由自己对应的函数执行环境。函数执行环境包含函数自身定义得变量和函数有权访问的其他数据。

每一个执行环境都对应着一个由解释器使用的变量对象。全局执行环境是最外围的执行环境。在web浏览器中全局执行环境变量为window。因此所有的全局变量和全局函数都是window对象的属性和方法。

当执行流进入一个函数时，函数对应的执行环境会压入执行函数栈。在函数执行完毕后，该函数的执行环境就会弹出。并将控制权返还给下一个执行环境。

当进入执行环境时，便开始分配内存，创建作用域链。当退出执行环境时便开始垃圾回收。

## 作用域链

当代码在执行环境中进行执行时，会创建执行环境对应的作用域链。

作用域链的作用是使得函数所有权访问的变量和函数进行有序的访问。

作用域链的规则是函数自身定义的变量位于最前端，其次是父级函数中的变量，再把再高级的函数变量放在其次，这样一直延续到全局环境。当数据需要进行变量查找时，也会按照作用域链的顺序进行查找，首先查找本地的变量。再查找父级。一直到全局。查到便停止。如果一直没有找到，则抛出undefined。

## 内存分配以及垃圾回收

### 内存分配

Javascript中包含两种变量类型：原始值和引用值。

原始值指null，undefined，number，string，boolean等。

引用值指object，function，Array等。

原始值在栈中分配内存。引用值在堆中分配内存。

|  |
| --- |
| 图 1 |

如图1所示。可以明确的发现。定义的对象数据被分配在堆空间中。并且栈空间中有对象的变量引用。而基本数据类型都是分配再栈空间。

栈的读写速度高于堆。仅次于CPU寄存器。但是只能存储数据大小已知或者生存周期确定。缺乏灵活性。

堆的优势是可以动态的分配内存空间。由于垃圾回收机制会不定期的回收堆中的垃圾资源，所以生存周期也可以是未知的。

相对于引用类型数据，原始类型数据所占用的内存空间小。将其放在堆中查找会浪费时间。而引用类型数据放入栈中会影响栈的读写效率。而且数组和对象会不断扩展，放入堆中正合适。

### 垃圾回收

垃圾回收机制会回收不再使用的变量的内存空间。不是实时的，会定期进行回收。

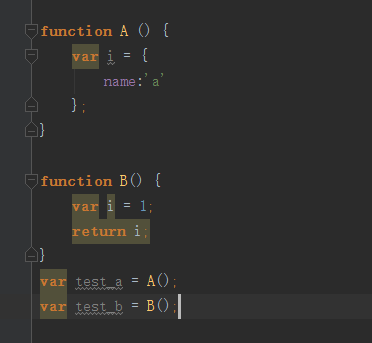


图 2

在方法A运行时，会为i分配空间，当A调用完毕后，对象A将会被回收。

在方法B运行时，会为i分配空间，当B调用完毕后返回了i的引用给test\_b。故外界引用了B的内部变量。所以对象B不会被回收。

#### 回收策略

##### 标记清除

该策略会给内存中的变量添加标记。去除环境（个人理解为全局环境）中的变量和环境中变量引用的变量（闭包）外。还标记的变量就是需要回收的变量。大部分浏览器都采用这种策略。

##### 引用计数

该策略会追踪每个值所被引用的次数，当有变量引用该值时，则引用次数加1。当变量变更引用对象时，则该值引用次数减1。当引用次数为0时则清除该值。

该策略无法清除循环引用的垃圾资源。会导致内存泄漏。如图3所示。a.prop引用了b。b.prop引用饿了a。当test调用完毕后，a和b的引用次数都不会是0。也都不会被回收，造成内存泄漏。

|  |
| --- |
| 图 3 |