JS的垃圾回收机制很简单，找出不再使用的变量，然后释放掉其占用的内存。但是这个过程不是实时的。因为其开销太大，所以垃圾回收器会按照固定的时间间隔周期性的执行。

不再使用的变量当然只能是全局变量，因为全局变量那你随时有可能会要引用，全局变量的生命周期知道页面卸载才会结束。

局部变量只有在函数执行的时候才会存在，在这个过程中才会在栈或者堆上分配内存，，注意闭包。

1. 标记清除  
   垃圾回收器会在运行的时候给所有的变量都加上标记。然后去掉在执行环境中的变量，以及环境中引用的变量（也就是闭包）。在此之后还有标记的就是要删除的变量了。原因是环境中已经无法再引用这些变量了。  
   其实就是执行上下文中的变量对象中的对象，以及scope chain中的变量。
2. 引用计数  
   声明一个 变量并赋给一个引用类型值给该变量时，这个值的引用次数加1.如果该值又被覆盖另一个变量则又加1.相反，如果包含这个值引用的变量又取得了另一个值，那么这个值的引用次数减1.当这个值的引用次数为0的时候，就把这个值收回来。

引用计数的问题  
循环引用。A是B的引用，B是A的引用。  
function fn() {

 var a = {};

 var b = {};

 a.pro = b;

 b.pro = a;

}

fn();  
标记清除没有问题，fn执行完毕后，两个变量离开环境了，会被清除。但是引用计数会发现计数都是2，没办法清除。  
如果设置document.getElementById(id).onclick=function(){  
}  
那么click方法自然会引用外部变量。手工断开js对象于DOM的连接，赋值为null。

1. GC的缺陷是：GC工作的时候，停止相应其他工作。应该避免GC照成长时间的停止相应。
2. 操作不当引起的内存泄露：  
   1.意外的全局变量：因为全局变量是不可回收的，使用严格模式避免。  
   2.闭包：将时间处理函数定义在外部，解除闭包  
   3.没有清理的DOM元素的引用  
   4.被遗忘的定时器或者回调：因为定时器中可能会有这个DOM的引用，及时你这个DOM删除了，但是内存在还是有的啊，要手动删除  
   5.子元素存在引用引起的内存泄露：