JavaScript 内存管理与垃圾回收机制

1. 为什么要进行垃圾回收?

因为程序中存在很多数据，这些数据在内存中占据一定的空间 。在程序运行中 ，一些没有用的数据（这些数据可以称为垃圾）还会在内存中占据空间。如果不进行垃圾回收的话，随着程序的运行，可用的内存越来越小 ，必然会带来程序性能的下降，造成卡、慢甚至系统异常。

2. js 垃圾回收机制

Javascript 具有自动垃圾回收机制，也就是说，执行环境会负责管理代码执行过程中使用的内存。

原理：垃圾收集器会定期（周期性）找出那些不在继续使用的变量，然后释放其内存。

3. 标记清除

js 中最常用的垃圾回收方式就是标记清除。当变量进入环境时，例如，在函数中声明一个变量，就将这个变量标记为“进入环境”。此时的变量在函数执行过程中一直存在，直到函数结束后，将变量标记为“离开环境”，变量就被回收了。（JavaScript 中全局变量的在浏览器卸载页面才会被销）

如下例子：

```
function count() {
  var num=0;
  num++
  console.log(num);
}

count(); //1
count(); //1
```

定义一个函数 count ，函数 count 被调用了两次结果 num 都输出 1 ，说明当第一次执行之后函数里面的变量 num 会被回收。然后在第二次调用时又重新声明变量 num 并初始化为 0,再进行计算输出结果 1。

4.引用计数（不常用）

引用计数的含义是跟踪记录每个值被引用的次数。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。如下：

```
var obj={
    name:"xiaoming",
    age:23
}; //引用次数为1

var person=obj; //引用次数为2
```

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，当垃圾回收器下次再运行时，它就会释放那些引用次数为 0 的值所占用的内存。如下：

```
var obj={
    name:"xiaoming",
    age:23
}; //引用次数为1

var person=obj; //引用次数为2
person={};//引用次数减1 ,剩余引用次数为1
obj={};//引用次数减1 ,剩余引用次数为0
```

Netscape Navigator3 是最早使用引用计数策略的浏览器，但很快它就遇

到一个严重的问题：循环引用。循环引用指的是对象 a 中包含一个指向对象 b 的指针，而对象 b 中也包含一个指向对象 a 的引用。

```
function fn(){
    var a={}; //引用次数为1
    var b={}; //引用次数为1

    a.pro=b; //引用次数为2
    b.pro=a; //引用次数为2
}

fn();
```

以上代码 a 和 b 的引用次数都是 2，fn()执行完毕后，两个对象都已经离开环境，在标记清除方式下是没有问题的，但是在引用计数策略下，因为 a 和 b 的引用次数不为 0，所以不会被垃圾回收器回收内存，如果 fn 函数被大量调用，就会造成内存泄露。在 IE7 与 IE8 上，内存直线上升。

IE 中有一部分对象并不是原生 js 对象。例如，其内存泄露 DOM 和 BOM 中的对象就是使用 C++以 COM 对象的形式实现的，而 COM 对象的垃圾回收机制采用的就是引用计数策略。因此，即使 IE 的 js 引擎采用标记清除策略来实现，但 js 访问的 COM 对象依然是基于引用计数策略的。换句话说，只要在 IE 中涉及 COM 对象，就会存在循环引用的问题。

```
var obj={};
var elem=document.getElementById("box");
elem.someAttr=obj;
obj.someProperty=elem;
```

如上例子中一个 dom 对象和一个原生 js 对象之间循环引用，即使 DOM 从页面中移除，它也永远不会被回收。最简单的解决办法就是手动解除引用：

```
elem.someAttr=null;
obj.someProperty=null;
```

5.内存管理

计算机分配给 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。因此，确保占用最少的内存可以让页面获得更好的性能。而优化内存占用的最佳方式，就是为执行中的代码只保存必要的数据。一旦数据不再有用，最好通过将其值设置为 null 来释放其引用——这个 做法叫做解除引用（dereferencing）。如下:

```
function personObj(name){
    var person = new Object();
    person.name = name;
    return person;
}
var student = personObj("Sunshine");
console.log(student.name)
// 手动解除student 的引用
student = null;
```

这一做法适用于大多数全局变量和全局对象的属性。局部变量会在它们离开执行环境时自动被解除引用