内存管理

简介

低级语言，比如C，有低级的内存管理基元，像malloc(),free()。另一方面，JavaScript的内存基元在变量（对象，字符串等等）创建时分配，然后在他们不再被使用时“自动”释放。后者被称为*垃圾回收*。这个“自动”是混淆并给JavaScript（和其他高级语言）开发者一个错觉：他们可以不用考虑内存管理。

不管什么程序语言，内存生命周期基本一致：

1. 分配你所需要的内存
2. 使用它（读、写）
3. 当它不被使用时释放　ps：和“把大象装冰箱“一个意思

第一二部分过程在所有语言中都很清晰。最后一步在低级语言中很清晰，但是在像JavaScript等高级语言中，最后一步不清晰。

JavaScript的内存分配

变量初始化

为了不让程序员为分配费心，JavaScript在定义变量时完成内存分配。

```
var n = 123;var s = "azerty";
var o = {
  a: 1,
  b: null
};
var a = [1, null, "abra"];function f(a){
  return a + 2;
}someElement.addEventListener('click', function(){
  someElement.style.backgroundColor = 'blue';
}, false);
```

通过函数调用的内存分配

有些函数调用结果是分配对象内存：

```
var d = new Date();
var e = document.createElement('div');
```

有些方法分配新变量或者新对象：

```
var s = "azerty";
```

```
var s2 = s.substr(0, 3);
var a = ["ouais ouais", "nan nan"];
var a2 = ["generation", "nan nan"];
var a3 = a.concat(a2);
```

值的使用

使用值的过程实际上是对分配内存进行读取与写入的操作，这意味着可以写入一个变量或者一个对象的属性值，甚至传递函数的参数。

当内存不再需要使用时释放

大多数内存管理的问题都在这个阶段。在这里最艰难的任务是找到“所分配的内存确实已经不再需要了”。它往往要求开发人员来确定在程序中哪一块内存不再需要并且释放它。

高级语言解释器嵌入了“垃圾回收器”，主要工作是跟踪内存的分配和使用，以便当分配的内存不再使用时，自动释放它。这个过程是一个近似的，因为要知道某块内存是否需要是 无法判定的 (无法被某种算法所解决).

垃圾回收

如上文所述自动寻找是否一些内存“不再需要”的问题是无法判定的。因此，垃圾回收实现只能有限制的解决一般问题。本节将解释必要的概念，了解主要的垃圾回收算法和它们的局限性。

引用

垃圾回收算法主要依赖于*引用*的概念。在内存管理的环境中，一个对象如果有访问另一个对象的权限（隐式或者显式），叫做一个对象引用另一个对象。例如，一个Javascript对象具有对它 原型 的引用（隐式引用）和对它属性的引用（显式引用）。

在这里，“对象”的概念不仅特指Javascript对象，还包括函数作用域（或者全局词法作用域）。

引用计数垃圾收集

这是最简单的垃圾收集算法。此算法把“对象是否不再需要”简化定义为“对象有没有其他对象引用到它”。如果没有引用指向该对象（零引用），对象将被垃圾回收机制回收。

例如

```
var o = {
  a: {
    b:2
  }
};
```

```
var o2 = o;o = 1;
var oa = o2.a;
o2 = "yo";
oa = null;
```

限制：循环引用

这个简单的算法有一个限制，就是如果一个对象引用另一个（形成了循环引用），他们可能“不再需要”了，但是他们不会被回收。

```
function f(){
  var o = {};
  var o2 = {};
  o.a = o2;  o2.a = o;
  return "azerty";
}

f();
```

实际当中的例子

IE 6, 7 对DOM对象进行引用计数回收。对他们来说，一个常见问题就是内存泄露：

```
var div = document.createElement("div");
div.onclick = function(){
  doSomething();
};
```

标记-清除算法

这个算法把“对象是否不再需要”简化定义为“对象是否可以获得”。

这个算法假定设置一个叫做*根*的对象（在Javascript里，根是全局对象）。定期的，垃圾回收器将从根开始，找所有从根开始引用的对象，然后找这些对象引用的对象……从根开始，垃圾回收器将找到所有可以获得的对象和所有不能获得的对象。

这个算法比前一个要好，因为“有零引用的对象”总是不可获得的，但是相反却不一定，参考“循环引用”。

从2012年起，所有现代浏览器都使用了标记-清除垃圾回收算法。所有对JavaScript垃圾回收算法的改进都是基于标记-清除算法的改进，并没有改进标记-清除算法本身和它对“对象是否不再需要”的简化定义。

循环引用不再是问题了

在上面的示例中，函数调用返回之后，两个对象从全局对象出发无法获取。因此，他们将会被垃圾回收器回收。

第二个示例同样，一旦 div 和其事件处理无法从根获取到，他们将会被垃圾回收器回收

。

限制: 对象需要明确的不可获得

尽管这是一个限制，但是很少会被突破，这也就是为什么在现实中很少人会去关心垃圾回收机制。

参考

- IBM article on "Memory leak patterns in JavaScript" (2007)
- Kangax article on how to register event handler and avoid memory leaks (2010)