js常见的内存泄漏及解决方法总汇

时间: 2017-12-06 阅读: 1090 标签: 内存

什么是内存泄漏?

内存泄漏指由于疏忽或错误造成程序未能释放已经不再使用的内存。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，导致在释放该段内存之前就失去了对该段内存的控制，从而造成了内存的浪费。

内存泄漏通常情况下只能由获得程序源代码的程序员才能分析出来。然而，有不少人习惯于把任何不需要的内存使用的增加描述为内存泄漏，即使严格意义上来说这是不准确的。

JS垃圾收集机制：

js具有自动回收垃圾的机制，即执行环境会负责管理程序执行中使用的内存。在C和C++等其他语言中，开发者的需要手动跟踪管理内存的使用情况。在编写js代码时候，开发人员不用再关心内存使用的问题，所需内存的分配 以及无用的回收完全实现了自动管理。

Js中最常用的垃圾收集方式是标记清除(mark-and-sweep)。当变量进入环境（例如，在函数中声明一个变量）时，就将这个变量标记为"进入环境"。从逻辑上讲，永远不能释放进入环境的变量所占的内存，因为只要执行流进入相应的环境，就可能用到它们。而当变量离开环境时，这将其 标记为"离开环境"。

> 这篇文章主要讲解JavaScript常见的内存泄漏及解决办法!

1.意外的全局变量：

Js处理未定义变量的方式比较宽松：未定义的变量会在全局对象创建一个新变量。在浏览器中，全局对象是window。

```
function foo(arg) {
    bar = "this is a hidden global variable"; //等同于window.bar="this is a hidden global
    this.bar2= "potential accidental global";//这里的this 指向了全局对象（window）,等同于
}
```

解决方法：在JavaScript程序中添加，开启严格模式'use strict'，可以有效地避免上述问题。

注意：那些用来临时存储大量数据的全局变量，确保在处理完这些数据后将其设置为null或重新赋值。与全局变量相关的增加内存消耗的一个主因是缓存。缓存数据是为了重用，缓

存必须有一个大小上限才有用。高内存消耗导致缓存突破上限，因为缓 存内容无法被回收。

2.循环引用

在js的内存管理环境中，对象 A 如果有访问对象 B 的权限，叫做对象 A 引用对象 B。引用计数的策略是将“对象是否不再需要”简化成“对象有没有其他对象引用到它”，如果没有对象引用这个对象，那么这个对象将会被回收 。

```
let obj1 = { a: 1 }; // 一个对象（称之为 A）被创建，赋值给 obj1，A 的引用个数为 1
let obj2 = obj1; // A 的引用个数变为 2

obj1 = 0; // A 的引用个数变为 1
obj2 = 0; // A 的引用个数变为 0，此时对象 A 就可以被垃圾回收了
```

但是引用计数有个最大的问题： 循环引用。

```
function func() {
    let obj1 = {};
    let obj2 = {};

    obj1.a = obj2; // obj1 引用 obj2
    obj2.a = obj1; // obj2 引用 obj1
}
```

当函数 func 执行结束后，返回值为 undefined，所以整个函数以及内部的变量都应该被回收，但根据引用计数方法，obj1 和 obj2 的引用次数都不为 0，所以他们不会被回收。要解决循环引用的问题，最好是在不使用它们的时候手工将它们设为空。上面的例子可以这么做：

```
obj1 = null;
obj2 = null;
```

3.被遗忘的计时器和回调函数

```
let someResource = getData();
setInterval(() => {
    const node = document.getElementById('Node');
    if(node) {
        node.innerHTML = JSON.stringify(someResource));
    }
}, 1000);
```

上面的例子中，我们每隔一秒就将得到的数据放入到文档节点中去。但在 setInterval 没有结束前，回调函数里的变量以及回调函数本身都无法被回收。那什么才叫结束呢？就是调用了 clearInterval。如果回调函数内没有做什么事情，并且也没有被 clear 掉的话，就会造成内存泄漏。不仅如此，如果回调函数没有被回收，那么回调函数内依赖的变量也没法被

回收。上面的例子中，someResource 就没法被回收。同样的，setTiemout 也会有同样的问题。所以，当不需要 interval 或者 timeout 时，最好调用 clearInterval 或者 clearTimeout。

4.DOM泄漏

在Js中对DOM操作是非常耗时的。因为JavaScript/ECMAScript引擎独立于渲染引擎，而DOM是位于渲染引擎，相互访问需要消耗一定的资源。 而IE的DOM回收机制便是采用引用计数的，以下主要针对IE而言的。

a.没有清理的DOM元素引用

```
var refA = document.getElementById('refA');
document.body.removeChild(refA);
 // #refA不能回收，因为存在变量refA对它的引用。将其对#refA引用释放，但还是无法回收#refA。
```

解决办法：refA = null;

b.给DOM对象添加的属性是一个对象的引用:

```
var MyObject = {};
document.getElementById('myDiv').myProp = MyObject;
```

解决方法:
在window.onunload事件中写上: document.getElementById('myDiv').myProp = null;

c.DOM对象与JS对象相互引用：

```
function Encapsulator(element) {
this.elementReference = element;
element.myProp = this;
}
new Encapsulator(document.getElementById('myDiv'));
```

解决方法： 在onunload事件中写上: document.getElementById('myDiv').myProp = null;

d.给DOM对象用attachEvent绑定事件：

```
function doClick() {}
element.attachEvent("onclick", doClick);
```

解决方法： 在onunload事件中写上: element.detachEvent('onclick', doClick);

e.从外到内执行appendChild。这时即使调用removeChild也无法释放:

```
var parentDiv = document.createElement("div");
var childDiv = document.createElement("div");
document.body.appendChild(parentDiv);
parentDiv.appendChild(childDiv);
```

解决方法： 从内到外执行appendChild:

```
var parentDiv = document.createElement("div");
var childDiv = document.createElement("div");
parentDiv.appendChild(childDiv);
document.body.appendChild(parentDiv);
```

5.js的闭包

闭包在IE6下会造成内存泄漏，但是现在已经无须考虑了。值得注意的是闭包本身不会造成内存泄漏，但闭包过多很容易导致内存泄漏。闭包会造成对象引用的生命周期脱离当前函数的上下文，如果闭包如果使用不当，可以导致环形引用（circular reference），类似于死锁，只能避免，无法发生之后解决，即使有垃圾回收也还是会内存泄露。

6.console

控制台日志记录对总体内存配置文件的影响可能是许多开发人员都未想到的极其重大的问题。记录错误的对象可以将大量数据保留在内存中。注意，这也适用于：

1）、 在用户键入 JavaScript 时，在控制台中的一个交互式会话期间记录的对象。

2）、由 console.log 和 console.dir 方法记录的对象。

结束语

本文主要介绍了几种常见的内存泄露。在开发过程，需要我们特别留意，随着浏览器的升级，目前大部分 大部分浏览器都已改进了内存清理功能。

上一页: JavaScript函数创建的细节

下一页: js不同数据类型下的toString()与t...

php中的内存管理