NGCRM js编程规范

|  |  |  |  |
| --- | --- | --- | --- |
| 拟制: | 刘效强 | 日期： | 2014-3-12 |
| 审核: |  | 日期： | yyyy-mm-dd |
| 批准: |  | 日期： | yyyy-mm-dd |



Huawei Technologies Co., Ltd.

华为技术有限公司

All rights reserved

版权所有 侵权必究

Revision Record 修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| **Date**  **日期** | **Revision Version**  **修订版本** | **Change Description**  **修改描述** | **Author**  **作者** |
| 2014-3-12 | 0.1 | initial 初稿完成 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[NGCRM js编程规范 1](#_Toc382408432)

[1 前言 4](#_Toc382408433)

[1.1 目的 4](#_Toc382408434)

[2 代码风格 4](#_Toc382408435)

[2.1 命名规范 4](#_Toc382408436)

[**规则1：命名遵从java编程规范中的命名要求，命名清晰有业务含义；js中涉及变量、函数、对象定义、文件等的命名。** 4](#_Toc382408437)

[**规则2：命名避开js保留字、关键字、window属性等；** 5](#_Toc382408438)

[**建议1：变量命名避开框架中的全局变量；** 7](#_Toc382408439)

[**建议2：构造函数名，采用首字母大写（InitialCap）；其他函数名，一律首字母小写；构造函数用名词，其他函数用动词或动名词；** 8](#_Toc382408440)

[2.2 注释规范 8](#_Toc382408441)

[**规则3：遵从java规范中的js注释规范部分；强调关键逻辑增加注释。** 8](#_Toc382408442)

[**规则4：为避免信息泄露，js注释要求使用<%-- --%>；.js文件如果发布时采用压缩方式， 要求使用jsDoc方式注释；** 8](#_Toc382408443)

[2.3 排版规范 9](#_Toc382408444)

[**规则5：排版规范遵从java规范中的要求，缩进、空格、空行、过长换行等。** 9](#_Toc382408445)

[**规则6：表示区块起首的大括号{，不要另起一行；表示结束的大括号}另起一行；** 9](#_Toc382408446)

[**规则7：空格应该用于如下情况：** 9](#_Toc382408447)

[**规则8：不要省略句末的分号。** 10](#_Toc382408448)

[**规则9：换行应选择在操作符和标点符号之后，而不要在变量名、字符串、数字、或')' ']' '++' '--'等符号之后换行。** 10](#_Toc382408449)

[**规则10：总是使用大括号表示区块；即使循环和判断的代码体只有一行。** 11](#_Toc382408450)

[**规则11：不要将不同目的的语句，合并成一行。** 11](#_Toc382408451)

[**规则12：禁止使用with语句。** 12](#_Toc382408452)

[3 编码规范 13](#_Toc382408453)

[3.1 变量 13](#_Toc382408454)

[**规则13：每个变量声明都要加上var关键字。** 13](#_Toc382408455)

[**规则14：避免使用全局变量；如果不得不使用，请参照后边两条建议。** 13](#_Toc382408456)

[**建议3：使用全局变量向专家团申请， 通过管理手段规范全局变量的使用。** 13](#_Toc382408457)

[**建议4：使用全局变量时加上命名空间。** 14](#_Toc382408458)

[**规则15：常量命名用大写，字面间下划线隔开， 如NAMES\_LIKE\_THIS；用关键字var，永远不要使用const。** 14](#_Toc382408459)

[**建议5：用 Array 和 Object 字面量代替 Array 和 Object 构造函数。** 14](#_Toc382408460)

[**规则16：所有变量在使用前声明，所有变量声明都放在函数的头部。** 16](#_Toc382408461)

[3.2 函数 17](#_Toc382408462)

[**规则17：js中的函数设计也要像java语言中的方法要求一样。** 17](#_Toc382408463)

[**规则18：不要在块内声明一个函数。** 17](#_Toc382408464)

[**3.3** **闭包** 18](#_Toc382408465)

[**规则19：一定要记住闭包中局部变量是传引用，不是传值。** 18](#_Toc382408466)

[3.4 其他语言特性 20](#_Toc382408467)

[**规则20：循环尽量使用for(;;)和while()，for in只用在object/map/hash 的遍历。** 20](#_Toc382408468)

[**规则21：慎用eval()方法。** 20](#_Toc382408469)

[**建议6：使用===和!==操作符会更好，==和!=操作符会做类型强制转换。** 20](#_Toc382408470)

[**建议7：需要多次使用表达式，利用局部变量缓存该表达式结果，减少对象查找。** 20](#_Toc382408471)

[**建议8：使用/..../来直接表达一个正则表达式，而不使用new RegExp()形式。** 21](#_Toc382408472)

[**建议9：除非用于类型转化，否则没有必要封装基本类型。** 21](#_Toc382408473)

[3.5 浏览器兼容问题 22](#_Toc382408474)

[**规则22：为了获得最大的可移植性和兼容性，尽量依赖于标准方法。** 22](#_Toc382408475)

[3.6 性能问题 25](#_Toc382408476)

[4 参考资料 26](#_Toc382408477)

# 前言

有人说，编译器的规范叫做“语法规则”（grammar），这是程序员必须遵守的；而编译器忽略的部分，就叫“编程风格”（programming style），这是程序员可以自由选择的。这种说法不完全正确，程序员固然可以自由选择编程风格，但是好的编程风格有助于写出质量更高、错误更少、更易于维护的程序。

所以，有一点应该明确，"编程风格"的选择不应该基于个人爱好、熟悉程度、打字工作量等因素，而要考虑如何尽量使代码清晰易读、减少出错。你选择的，不是你喜欢的风格，而是一种能够清晰表达你的意图的风格。这一点，对于JavaScript这种语法自由度很高、设计不完全成熟的语言尤其重要。

## 目的

目前公司层面和部门层面没有一本合适的javaScript编程规范指导日常开发工作， 而NGCRM领域几个项目都用到它，本规范列出了编写javaScript需要遵守的规范，期望：

1. 统一编程风格；
2. 提高代码的可阅读性；
3. 减少错误产生；
4. 减少性能漏洞；
5. 提高代码可靠性；
6. 减少错误的编码设计；
7. 作为代码检查的依据；

# 代码风格

## 命名规范

### **规则1：命名遵从java编程规范中的命名要求，命名清晰有业务含义；js中涉及变量、函数、对象定义、文件等的命名。**

**说明：**良好的命名能提到代码的可读性和可维护性，能更好的表达意图，避免因命名产生的冲突、错误。

包名：采用域后缀倒置的加上自定义的包名，采用小写字母。

类名：使用类意义完整的英文描述，每个英文单词的首字母使用大写、其余字母使用小写的大小写混合法。

方法名：使用类意义完整的英文描述：第一个单词的字母使用小写、剩余单词首字母大写其余字母小写的大小写混合法。

属性名: 使用意义完整的英文描述：第一个单词的字母使用小写、剩余单词首字母大写其余字母小写的大小写混合法。属性名不能与方法名相同。

常量名: 使用全大写的英文描述，英文单词之间用下划线分隔开。

### **规则2：命名避开js保留字、关键字、window属性等；**

**说明：**

**关键字:** 一个完整的 JavaScript 实现是由以下 3 个不同部分组成的：核心(ECMAScript)、文档对象模型(DOM)、浏览器对象模型(BOM)。其中ECMA-262描述了一组具有特定用途的关键字，这些关键字可用于表示控制语句的开始或结束，或者用于执行特定操作等。按照规则，关键字是语言保留的，不能用作标识符；

|  |  |  |  |
| --- | --- | --- | --- |
| break | do | instanceof | typeof |
| case | else | new | var |
| catch | finally | return | void |
| continue | for | switch | while |
| debugger\* | function | this | with |
| default | if | throw | delete |
| in | try |  |  |

**保留字:** 尽管保留字在这门语言中还没有任何特定的用途，但它们有可能在将来被用作关键字；

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | enum | int | short |
| boolean | export | interface | static |
| byte | extends | long | super |
| char | final | native | synchronized |
| class | float | package | throws |
| const | goto | private | transient |
| debugger | implements | protected | volatile |
| double | import | public | let\* |
| yield\* |  |  |  |

除了上面列出的关键字和保留字，ECMA-262第5版对evel和arguments也施加了限制，在严格模式下，这两个也不能做为标识符或属性名，否则会抛出错误。

**window属性：**window是JS中最大的对象，它描述的是一个浏览器窗口，一般要引用他的属性和方法时，不需要用“window.XXX”这种形式，而是直接使用“XXX”。因此直接用这些属性定义var 变量可能引起不可预见的问题，禁止使用用来命名；

|  |  |
| --- | --- |
| **属性** | **描述** |
| [closed](http://www.w3school.com.cn/jsref/prop_win_closed.asp) | 返回窗口是否已被关闭。 |
| [defaultStatus](http://www.w3school.com.cn/jsref/prop_win_defaultstatus.asp) | 设置或返回窗口状态栏中的默认文本。 |
| [document](http://www.w3school.com.cn/jsref/dom_obj_document.asp) | 对 Document 对象的只读引用。请参阅 [Document 对象](http://www.w3school.com.cn/jsref/dom_obj_document.asp)。 |
| [history](http://www.w3school.com.cn/jsref/dom_obj_history.asp) | 对 History 对象的只读引用。请参数 [History 对象](http://www.w3school.com.cn/jsref/dom_obj_history.asp)。 |
| [innerheight](http://www.w3school.com.cn/jsref/prop_win_innerheight_innerwidth.asp) | 返回窗口的文档显示区的高度。 |
| [innerwidth](http://www.w3school.com.cn/jsref/prop_win_innerheight_innerwidth.asp) | 返回窗口的文档显示区的宽度。 |
| width | 以像素为单位指定窗口的宽度，已被innerWidth取代； |
| height | 以像素为单位指定窗口的高度，已被innerHeight取代； |
| length | 设置或返回窗口中的框架数量。 |
| [location](http://www.w3school.com.cn/jsref/dom_obj_location.asp) | 用于窗口或框架的 Location 对象。请参阅 [Location 对象](http://www.w3school.com.cn/jsref/dom_obj_location.asp)。 |
| [name](http://www.w3school.com.cn/jsref/prop_win_name.asp) | 设置或返回窗口的名称。 |
| [Navigator](http://www.w3school.com.cn/jsref/htmldom/dom_obj_navigator.asp) | 对 Navigator 对象的只读引用。请参数 [Navigator 对象](http://www.w3school.com.cn/jsref/dom_obj_navigator.asp)。 |
| [opener](http://www.w3school.com.cn/jsref/prop_win_opener.asp) | 返回对创建此窗口的窗口的引用。 |
| [outerheight](http://www.w3school.com.cn/jsref/prop_win_outerheight.asp) | 返回窗口的外部高度。 |
| [outerwidth](http://www.w3school.com.cn/jsref/prop_win_outerwidth.asp) | 返回窗口的外部宽度。 |
| pageXOffset | 设置或返回当前页面相对于窗口显示区左上角的 X 位置。 |
| pageYOffset | 设置或返回当前页面相对于窗口显示区左上角的 Y 位置。 |
| parent | 返回父窗口。 |
| [Screen](http://www.w3school.com.cn/jsref/htmldom/dom_obj_screen.asp) | 对 Screen 对象的只读引用。请参数 [Screen 对象](http://www.w3school.com.cn/jsref/dom_obj_screen.asp)。 |
| [self](http://www.w3school.com.cn/jsref/prop_win_self.asp) | 返回对当前窗口的引用。等价于 Window 属性。 |
| [status](http://www.w3school.com.cn/jsref/prop_win_status.asp) | 设置窗口状态栏的文本。 |
| [top](http://www.w3school.com.cn/jsref/prop_win_top.asp) | 返回最顶层的先辈窗口。 |
| window | window 属性等价于 self 属性，它包含了对窗口自身的引用。 |
| * screenLeft * screenTop * screenX * screenY | 只读整数。声明了窗口的左上角在屏幕上的的 x 坐标和 y 坐标。IE、Safari 和 Opera 支持 screenLeft 和 screenTop，而 Firefox 和 Safari 支持 screenX 和 screenY。 |

### **建议1：变量命名避开框架中的全局变量；**

**说明:** 各个项目均使用框架或自己写一些公共的js定义公共变量，在个人文件中变量命名要避开这些名称，避免不必要的错误，例如使用bme3.0变量命名时避开如下全局变量名：

window.PortItem

window.Portlet

window.Layout

window.Popwin

window.jBME

window.MsgBox

window.Timer

window.validator

window.$

window.jQuery

### **建议2：构造函数名，采用首字母大写（InitialCap）；其他函数名，一律首字母小写；构造函数用名词，其他函数用动词或动名词；**

**说明：**js支持自定义对象，但并没有像java语言那样支持类的定义， 定义对象和定义函数都是使用 function关键字， 可用上述建议至少在视觉上区分开构造函数和其他函数。如果只是初步的使用js（例如只是在页面文件中简单使用js控制DOM，完成一些简单逻辑）没必要预定义对象；如果深入使用js（如框架级别的bme3.0中的js）还是建议好好学习一下js如何定义类（或者叫对象预定义）；w3c有相关介绍：http://www.w3school.com.cn/js/pro\_js\_object\_defining.asp

## 注释规范

### **规则3：遵从java规范中的js注释规范部分；强调关键逻辑增加注释。**

**说明**：NGCRM编程规范java部分对js注释已有部分规定不再赘述，日常工作中发现js中的注释不多， 有写阅读困难的代码， 在此强调代码的自注释性、可读性，必要的地方增加注释。

### **规则4：为避免信息泄露，js注释要求使用<%-- --%>；.js文件如果发布时采用压缩方式， 要求使用jsDoc方式注释；**

**说明：**NGCRM领域客户对信息安全很敏感，并且实现方式的注释泄露便于别人找到漏洞攻击；因此页面注释信息禁止能直接通过ie得到；js文件压缩会把注释压缩掉， 且jsDoc类似javaDoc， 是很要的一种注释方式， 并且对js的弱语法性有所补充， 强烈建议压缩js的源文件用此方式注释，可参考google js编程规注释部分。

## 排版规范

### **规则5：排版规范遵从java规范中的要求，缩进、空格、空行、过长换行等。**

### **建议3：表示区块起首的大括号{，不要另起一行；表示结束的大括号}另起一行；**

**说明：**这点有别于java，请关注。

因为JavaScript会自动添加句末的分号，导致一些难以察觉的错误。

1. return
3. {
5. key:value;
7. };

上面的代码的原意，是要返回一个对象，但实际上返回的是undefined，因为JavaScript自动在return语句后面添加了分号。为了避免这一类错误，需要写成下面这样：

1. return {
3. key : value;
5. };

### **规则6：空格应该用于如下情况：**

**1. 关键字后面跟左括号“(”之间应该用一个空格隔开。如if、for、while等**

**2. 函数名和左括号“(”之间不要有空格；但匿名函数的function和左括号“(”之间要有空格。 参数与“（”、“）”均没有空格；**

**3. 所有的二元操作符，除了“.”、“(”和“[”之外，都应该使用一个空格来和操作数隔开。如 a + b、a == b等**

**4. 一元操作符和操作数之间不应该使用空格隔开，除了操作符是一个单词时，如typeof。**

**5. for语句控制部分的每个分号“;”应该在后面跟一个空格。**

**6. 每个逗号“,”后面应该跟一个空格。**

### **规则7：不要省略句末的分号。**

**说明：**分号表示语句的结束。大多数情况下，如果你省略了句尾的分号，JavaScript会自动添加。

var a = 1

等同于

var a = 1;

因此，有人提倡省略句尾的分号。但麻烦的是，如果下一行的第一个字元（token）是下面这五个字符之一，JavaScript将不对上一行句尾添加分号："("、"["、"/"、"+"和"-"。

1. x = y
3. (function (){
5. ...
7. })();

上面的代码等同于

x = y (function (){...})();

并且分号有助于代码阅读，因此不要省略分号。

### **规则8：换行应选择在操作符和标点符号之后，而不要在变量名、字符串、数字、或')' ']' '++' '--'等符号之后换行。**

示例：

<script language="javascript">

var valueA = 1;

var valueB = valueA ///bad

+1;

var valueC = valueB + ///good

valueA;

alert(valueB); //output: valueB=2

alert(valueC);//output: valueC=3

</script>

### **规则9：总是使用大括号表示区块；即使循环和判断的代码体只有一行。**

如果循环和判断的代码体只有一行，JavaScript允许该区块（block）省略大括号。

下面的代码

if (a) b (); c ();

原意可能是

if (a) { b (); c ();}

但是，实际效果是

if (a) { b ();} c ();

### **规则10：不要将不同目的的语句，合并成一行。**

**说明**：有些程序员追求简洁，喜欢合并不同目的的语句。比如，原来的语句是

a = b;

if (a) {...}

他喜欢写成下面这样:

if (a = b) {...}

虽然语句少了一行，但是可读性大打折扣，而且会造成误读，让别人误以为这行代码的意思是：

if （a === b）{...}

另外一种情况是，有些程序员喜欢在同一行中赋值多个变量：

var a = b = 0;

他以为，这行代码等同于

var a = 0, b = 0;

实际上不是，它的真正效果是下面这样：

b = 0;

var a = b;

### **规则11：禁止使用with语句。**

**说明：**with可以减少代码的书写，但是会造成混淆。使你的代码形如踏云，毫无安全感，因为用 with 添加过来的对象可能会对当前局部作用域的属性与方法产生冲突，进而影响到整个环境。

1. with (o) {
3. foo = bar;
5. }

上面的代码，可以有四种运行结果：

1. o.foo = bar;
3. oo.foo = o.bar;
5. foo = bar;
7. foo = o.bar;

这四种结果都可能发生，取决于不同的变量是否有定义。

# 编码规范

## 变量

### **规则12：每个变量声明都要加上var关键字。**

**说明：**如果你不指定关键字var, 该变量就会暴露在全局作用域（window）中，这很可能会覆盖全局作用域中的同名变量，从而引发问题（另外GC也会因此而无法有效回收内存啊），所以务必用var声明变量。

### **规则13：避免使用全局变量；如果不得不使用，请参照后边两条建议。**

**说明：**JavaScript最大的语法缺点，可能就是全局变量对于任何一个代码块，都是可读可写。这对代码的模块化和重复使用，非常不利；能用局部变量的地方不要使用全局变量，局部变量的访问速度更快，安全性更高，能避免一些问题。

**案例：**公共页面上（事后老员工分析该处可以优化，在方法内部声明即可）声明了一个判断验证码验证结果的javascript窗口级变量：var status =1。这个变量与浏览器内置对象同名、同作用域：window.status，结果就是程序重设了该对象。而我们的读卡程序与大唐的读卡页面交互，依赖window.status传递读卡的信息。程序重设了该变量，导致读卡读不到数据，开户业务中止。经分析：该问题是js声明了与IE内置对象同名、同作用域的变量导致的开户失败。

### **建议4：使用全局变量向专家团申请， 通过管理手段规范全局变量的使用。**

**说明：**良好的命名能够避免冲突问题的发生，必须使用的地方经专家把关，能考虑到对整个系统的影响， 有效避免新人因不清楚引入问题。项目组可以规定自定义全局变量全部用大写字面（有点像常量）或者使用同一的前缀等(如:项目名**icrm\_**)。

### **建议5：使用使用全局变量时加上命名空间。**

**说明：**此是电软核编程军规中推荐的方法， 如采用此方法建议项目组需对命名空间做统一规定，避免引入新的冲突；

window.session = {};

session.name = "hhh";

使用命名空间组织类，防止变量或函数名的冲突，不同命名空间中相同变量或函数名不存在冲突，如果命名空间也同名则只能悲剧。

原生js不支持命名空间，可变通实现。实现的想法是建一个对象作为命名空间，将同一个命名空间的变量或函数加入到同一个对象，这样，访问的时候需要引用对象名，即使变量或函数名相同也不会产生冲突。即命名空间就是一个个对象。通过将一个命名空间包含到另一个命名空间的方式，实现命名空间的层次结构。如a.b.c，a为window对象的属性，b为a对象的属性，依此类推。

### **规则14：常量命名用大写，字面间下划线隔开， 如NAMES\_LIKE\_THIS；用关键字var，永远不要使用const。**

**说明：**js语法中没有常量定义，我们只是在需要的时形式上定义常量， 在使用时不要改变其值；可以用@const来标记它是常量，但不要用关键字const声明，const关键字只在firefox下有效， IE不支持。

### **建议6：用 Array 和 Object 字面量代替 Array 和 Object 构造函数。**

**说明：**使用[param,param1....]来直接表达一个数组，而不使用new Array(parm,parm1..)形式；使用{}来直接表达一个对象，而不使用new Object()形式。

数组字面量语法简单，直接，优雅；毕竟，数组就是一个下标从零开始的元素的集合；

没有必要去使用new去调用Array()构造方法，减少代码。

Google规范中给出的解释是：

Array 构造函数会因为传参不当而导致理解错误。

// 长度是 3.

var a1 = new Array(x1, x2, x3);

// 长度是 2.

var a2 = new Array(x1, x2);

// 如果x1是自然数那数组的长度就变成了x1

// 如果x1是数字但不是自然数就会报错

// 不然数组就会有一个元素x1

var a3 = new Array(x1);

// 长度是 0.

var a4 = new Array();

如上，如果传入了一个参数而不是两个，那这数组可能就不是预期的长度了。

为了避免这种情况，用更可读的数组字面量吧。

var a = [x1, x2, x3];

var a2 = [x1, x2];

var a3 = [x1];

var a4 = [];

Object 构造函数没有如上问题，但考虑到可读性和一致性，用字面量更好些。

var o = new Object();

var o2 = new Object();

o2.a = 0;

o2.b = 1;

o2.c = 2;

o2['strange key'] = 3;

应该写成：

var o = {};

var o2 = {

a: 0,

b: 1,

c: 2,

'strange key': 3

};

### **规则15：所有变量在使用前声明，所有变量声明都放在函数的头部。**

**说明：**JavaScript会自动将变量声明"提升"（hoist）到代码块（block）的头部。

1.if (!o) {

2.

3.var o = {};

4.

5.}

等同于

1.var o;

2.

3.if (!o) {

4.

5.o = {};

6.

7.}

为了避免可能出现的问题，不如把变量声明都放在代码块的头部。

for (var i ...) {...}

最好写成：

var i;

for (i ...) {...，}

## 函数[原文](http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml?showone=Constants)

### **规则16：js中的函数设计也要像java语言中的方法要求一样。**

1. **明确函数功能，精确而不是近似实现函数设计；**
2. **函数规模尽量控制在200行内；圈复杂度满足要求；**
3. **一个函数仅完成一件功能，不要设计多用途面面俱到的函数；**
4. **函数名应准确描述函数的功能；**

**设计出高内聚低耦合的函数，提高可维护性。**

### **规则17：不要在块内声明一个函数。**

**说明：**不要写成:

if (x) {

function foo() {}

}

虽然很多 JS 引擎都支持块内声明函数, 但它不属于 ECMAScript 规范 (见 [ECMA-262](http://www.ecma-international.org/publications/standards/Ecma-262.htm), 第13和14条). 各个浏览器糟糕的实现相互不兼容, 有些也与未来 ECMAScript 草案相违背. ECMAScript 只允许在脚本的根语句或函数中声明函数. 如果确实需要在块中定义函数, 建议使用函数表达式来初始化变量:

if (x) {

var foo = function() {}

}

### **建议7：Javascript脚本放在页面的底部，延迟加载。**

说明：放在底部可能会出现JS错误问题，当脚本没加载进来，用户就触发脚本事件，要注意避免此类错误。

## **闭包**

什么叫闭包？指的是词法表示包括不被计算的变量的函数（太拗口， 我也没看明白），翻译过来就是**函数可以使用函数之外定义的变量**（注意这个函数之外的变量可不仅仅是全局变量）。

闭包很有用，经常被忽略，用之要谨慎，记住一句关键的：

### **规则18：一定要记住闭包中局部变量是传引用，不是传值。**

**示例：**

错误代码：

var arr = [];

for ( var i = 0; i < 10; i++) {

arr[i] = function() {

console.log(i);

};

}

arr[3]();

输出：10

正确代码：

var arr = [];

for ( var i = 0; i < 10; i++) {

arr[i] = (function(k) {

return function() {

console.log(k);

};

})(i);

}

arr[3]();

输出：3

**说明：**有一件需要注意的事情，由于闭包会保持一个指向它封闭作用域的指针，所以在给DOM元素附加闭包时，很可能会产生循环引用，进一步的消耗内存，比如下面的代码：

function foo(element, a, b) {

element.onclick = function() { /\* uses a and b \*/ };

}

即便这个闭包函数内部并没有使用 **element**，可它还是保持了对 **element**, a 和 b 的引用，由于 **element** 也保持了对这闭包函数的引用，就导致了循环引用，无法被GC回收。如果遇到了这种情况，可以将代码优化一下：

function foo(element, a, b) {

element.onclick = bar(a, b);

}

function bar(a, b) {

return function() { /\* uses a and b \*/ } //此处的a、b是参数传入，而非闭包了。

}

**注释：**由于IE9之前的版本对JScript对象和DOM对象使用不同的垃圾收集例程，因此闭包在IE的这些版本中会导致如上问题。这个匿名函数作为element元素事件处理程序，形成闭包的状态就会保存对父层函数内活动对象的引用，只要匿名函数存在，element的引用数至少也是1，因此它所占用的内存就永远不会被回收。注意，闭包会引用包含函数的整个活动对象，即使闭包不直接引用element，包含函数的活动对象中也仍然会保存一个引用。因此，有必要把使用完的element变量设置为null，解除对DOM对象的引用，确保正常回收其占用的内存。（取自《Javascript高级程序设计》184P）

## 其他语言特性

### **规则19：循环尽量使用for(;;)和while()，for in只用在object/map/hash 的遍历。**

**说明：**JavaScript中的循环方式有for(;;)、while()、for(in)3种。其中for(in)的效率极差，因为for(in)执行过程中需要查询散列键。for(;;)和while()比较，while循环的效率要优于for(;;)。

### **规则20：慎用eval()方法。**

**说明：**eval() 会让程序执行的比较混乱, 当 eval() 里面包含用户输入的话就更加危险. 可以用其他更佳的, 更清晰, 更安全的方式写你的代码, 所以一般情况下请不要使用 eval()。

### **建议8：使用===和!==操作符会更好，==和!=操作符会做类型强制转换。**

**说明：**JavaScript有两个表示"相等"的运算符："相等"（==）和"严格相等"（===）。

因为"相等"运算符会自动转换变量类型，造成很多意想不到的情况：

0 == ''// true

1 == true // true

2 == true // false

0 == '0' // true

false == 'false' // false

false == '0' // true

" \t\r\n " == 0 // true

### **建议9：需要多次使用表达式，利用局部变量缓存该表达式结果，减少对象查找。**

因为JavaScript的特性，对于类似表达式a.b.c.d.e，需要至少4次查询操作。首先检查a在检查a中的b，如此往下。

### **建议10：使用/..../来直接表达一个正则表达式，而不使用new RegExp()形式。**

**示例：**

var pattern=/^[a-zA-Z]{1,30}$/;

### **建议11：除非用于类型转化，否则没有必要封装基本类型。**

**说明：**Js有5 种原始类型（primitive type），即 Undefined、Null、Boolean、Number 和 String；

对变量或值调用 typeof 运算符将返回下列值之一：

undefined - 如果变量是 Undefined 类型的

boolean - 如果变量是 Boolean 类型的

number - 如果变量是 Number 类型的

string - 如果变量是 String 类型的

object - 如果变量是一种引用类型或 Null 类型的

没有必要封装基本类型，书上说主动 new 基本包装类型（Boolean/Number/String）的人儿永远也成不了高富帅！而且如下情况存在风险：

var x = new Boolean(false);

if (x) {

alert('hi'); // 显示 hi。因为基本包装类型的实例调用typeof会返回"object"，对象么在判断时都会被转换为布尔值true。

}

除非明确用于类型转换， 有时用作 number, string 或 boolean时, 类型的转换会非常实用：

var x = Boolean(0);

if (x) {

alert('hi'); // 不会弹框显示 hi

}

typeof Boolean(0) == 'boolean';

typeof new Boolean(0) == 'object';

## 浏览器兼容问题

### **规则21：为了获得最大的可移植性和兼容性，尽量依赖于标准方法。**

说明：尤其对于开放式的系统，我们不能约束系统使用者的运行环境，要充分考虑兼容性问题，以下js部分浏览器兼容解决办法由电子渠道提供， 更多浏览器兼容（html、样式等）请参考《电子渠道开发规范CheckList》。

|  |  |  |
| --- | --- | --- |
| **常见问题** | **问题描述** | **解决办法** |
| HTML对象获取问题 | FireFox/IE8:document.getElementById("idName"); ie(非IE8):document.idname或者document.getElementById("idName") | 统一使用document.getElementById("idName") 或 jquery:$(selector) |
| const问题 | 说明:Firefox下,可以使用const关键字或var关键字来定义常量; IE下,只能使用var关键字来定义常量. | 统一使用var关键字来定义常量. |
| event.x与event.y问题 | IE下,event对象有x,y属性,但是没有pageX,pageY属性; Firefox下,event对象有pageX,pageY属性,但是没有x,y属性. | 使用mX(mX=event.x?event.x:event.pageX;)来代替IE下的event.x或者Firefox下的event.pageX |
| window.location.href问题 | IE或者Firefox2.0.x下,可以使用window.location或window.location.href; Firefox1.5.x下,只能使用window.location | 使用window.location来代替window.location.href |
| 访问frame对象 | IE:使用window.frameId或者window.frameName来访问这个frame对象.frameId和frameName可以同名。 Firefox:只能使用window.frameName来访问这个frame对象 | 另外，在IE和Firefox中都可以使用window.document.getElementById("frameId")来访问这个frame对象 |
| 切换frame内容 | 在IE和Firefox中都可以使用window.document.getElementById("testFrame").src="xxx.html"或window.frameName.location="xxx.html"来切换frame的内容 | 使用window.frameName或window.document.getElementById("frameId") |
| frame参数回传父窗口 | 如果需要将frame中的参数传回父窗口(注意不是opener,而是parentframe)，可以在frame中使用parent来访问父窗口 | 例如:parent.document.form1.filename.value="Aqing"; |
| 模态和非模态窗口问题 | IE下,可以通过showModalDialog和showModelessDialog打开模态和非模态窗口;Firefox下则不能 | 直接使用window.open(pageURL,name,parameters)方式打开新窗口。 如果需要将子窗口中的参数传递回父窗口,可以在子窗口中使用window.opener来访问父窗口. 如:var parWin=window.opener;parWin.document.getElementById("Aqing").value="Aqing"; |
| firefox与IE的父元素(parentElement)的区别 | IE:obj.parentElement firefox:obj.parentNode | 因为firefox与IE都支持DOM,因此使用obj.parentNode是不错选择,或者使用jquery:$(selector).parent() |
| document.formName.item("itemName")问题 | IE下，可以使用document.formName.item("itemName")或document.formName.elements["elementName"]； Firefox下，只能使用document.formName.elements["elementName"] | 统一使用document.formName.elements["elementName"] |
| 集合类对象问题 | IE下，可以使用()或[]获取集合类对象； Firefox下，只能使用[]获取集合类对象 | 统一使用[]获取集合类对象 |
| 自定义属性问题 | IE下，可以使用获取常规属性的方法来获取自定义属性，也可以使用getAttribute()获取自定义属性；Firefox下，只能使用getAttribute()获取自定义属性。 | 统一通过getAttribute()获取自定义属性, 或使用Jquery: $(selector).attr(); |
| input.type属性问题 | IE下input.type属性为只读；但是Firefox下input.type属性为读写。 | 不修改input.type属性。如果必须要修改，可以先隐藏原来的input，然后在同样的位置再插入一个新的input元素 |
| event.srcElement问题 | IE下，even对象有srcElement属性，但是没有target属性； FF下，even对象有target属性，但是没有srcElement属性。 | 使用srcObj=event.srcElement?event.srcElement:event.target; |
| body载入问题 | Firefox的body对象在body标签没有被浏览器完全读入之前就存在； 而IE的body对象则必须在body标签被浏览器完全读入之后才存在。 | [注]这个问题尚未实际验证，待验证后再来修改。 [注]经验证，IE6、Opera9以及FireFox2中不存在上述问题，单纯的JS脚本可以访问在脚本之前已经载入的所有对象和元素，即使这个元素还没有载入完成。 这个地方最好的解决办法是定义一下body加载完之后在去执行的JS方法或者脚本。 1、$(function(){...}); 2、$(document).ready(){function(){...}}; |
| 事件委托方法 | IE下，使用document.body.onload=inject;其中function inject()在这之前已被实现；在Firefox下，使用document.body.onload=inject(); | 统一使用document.body.onload=newFunction('inject()');或者document.body.onload=function(){/\*这里是代码\*/}, 或使用 jquery: $(function(){}). [注意]Function和function的区别 |
| Table操作问题 | IE、FF以及其它浏览器对于table标签的操作都各不相同，在ie中不允许对table和tr的innerHTML赋值，使用js增加一个tr时，使用appendChild方法也不管用。 | //向table追加一个空行: varrow=otable.insertRow(-1); varcell=document.createElement("td");cell.innerHTML=""; cell.className="XXXX"; row.appendChild(cell); [注]由于俺很少使用JS直接操作表格，这个问题没有遇见过。建议使用JS框架集来操作table，如JQuery $(selector).append() 对table的操作最简单快捷的方式是将可能要分开的行分出来单独做一个表格出来，这样便于操作 |
| 对象宽高赋值问题 | FireFox中类似obj.style.height=imgObj.height的语句无效, 应使用$(selector).height(); | 应使用 $(selector).height(); |
| js获取对象 |  | 建议统一使用 $(selector); |
| js获取对象的高宽 |  | $(selector).width(); $(selector).height();  $(selector).innerWidth(); $(selector).innerHeight();  $(selector).outerWidth(boolean); $(selector).outerHeight(boolean); |
| js获取对象的属性 |  | $(selector).attr(“attribute”) //这里id与name属性比较特殊可以直接将jquery对象转换成dom对象获取,效率更高--$(selector)[0].id, $(selector)[0].name |
| js设置对象的属性 |  | $(selector).attr({  Id:id, Name:name, "class":className, Title:title, ….  "data-"+attribute : ?? //自定义属性前缀为"data-"  ….}); |
| js获取对象的样式 |  | 获取对象的样式 $(selector).css(“background”) |
| js设置对象的样式 |  | $(selector).css({  Position: "absolute", Background: "#333333", MarginLeft: "3px", "z-index": 11234 }) |
| js获取对象的物理位置 |  | var offset = $(selector).offset(); var offLeft = offset.left;//相对页面X轴位移 var offTop = offset.top;//相对页面Y轴位移 var position = $(selector). position(); var positionLeft = position.left;//相对定位的X轴位移 var positionTop = position.top;//相对定位的Y轴位移 可以这样理解： offset可以理解为相对于根目录的坐标位置 position是相对于他父级的坐标位置 |
| js事件源位移 |  | var e = event || window.event; var x = e.pageX; var y = e.pageY; 以上几种位移可以实现任何拖拽,滚动等动画. |

## BOSS规范补充

来源于老的《NGCRM开发手册－编程规范分册（Java）》

* 具有通用功能的js函数不要定义在jsp、usl等页面文件中，首先应该确认框架是否提供了具有该功能的函数，如果没有的话再确认应用公共js文件中是否具有该功能的函数，如果还没有的话，向前台负责人申请在同类js文件中添加该函数。
* 如果页面中js脚本内容很多，应拆分成单独的js文件在页面中引入，以有效利用静态缓存，减小网络流量。
* 不要修改框架提供的js文件，以免升级时被冲掉。
* 对过大的js文件(比如20k以上的)应该用压缩工具压缩后使用。
* JS中注释建议使用<%-- --%>
* 判断是否必要使用js， 能在java实现的建议放java端（具体项目具体分析， bme3.0减少js的使用）。

# 参考资料

**《JavaScript 高级教程》 ---来源于w3c**

**《NGCRM编程手册-编程规范分册（java）》 ---来源NGCRM团队老版编程规范**

**《Google JavaScript 编码规范指南》----来源，网上翻译版**

**《电子渠道开发规范CheckList.xlsx》---来源，电子渠道团队**

**《javascript编程规范.xlsx》 ---来源，内网下载，我司某团队**

**《JavaScript编程规范.doc》—来源，内网下载，我司某团队**

**还有一些同事日常问题的反馈， 感谢提供。**