**JavaScript**

**编码规范**

目录

[1 背景 5](#_Toc385318760)

[2 兼容性要求 5](#_Toc385318761)

[3 术语定义 5](#_Toc385318762)

[4 JavaScript 语言规范 5](#_Toc385318763)

[4.1 方法定义 5](#_Toc385318764)

[4.2 闭包 5](#_Toc385318765)

[4.3 eval() 6](#_Toc385318766)

[4.4 with() {} 6](#_Toc385318767)

[4.5 this 6](#_Toc385318768)

[4.6 块内函数和块内变量声明 7](#_Toc385318769)

[4.7 分号 8](#_Toc385318770)

[4.8 变量 8](#_Toc385318771)

[4.9 封装基本类型 9](#_Toc385318772)

[4.10 修改内置对象的原型 10](#_Toc385318773)

[4.11 IE下的条件编译 10](#_Toc385318774)

[4.12 多行字符串 10](#_Toc385318775)

[4.13 自定义 toString() 方法 11](#_Toc385318776)

[4.14 HTML与JS分离 11](#_Toc385318777)

[4.15 使用”use strict” 11](#_Toc385318778)

[4.16 对象 12](#_Toc385318779)

[4.17 相等操作符 12](#_Toc385318780)

[4.18 多级原型结构 12](#_Toc385318781)

[4.19 for-in 循环 12](#_Toc385318782)

[4.20 数组 13](#_Toc385318783)

[5 JavaScript 编码风格 14](#_Toc385318784)

[5.1 命名 14](#_Toc385318785)

[5.1.1 属性和方法 14](#_Toc385318786)

[5.1.2 常量 14](#_Toc385318787)

[5.1.3 Getters 和 Setters 命名 14](#_Toc385318788)

[5.1.4 为全局代码使用命名空间 14](#_Toc385318789)

[5.2 代码格式化 14](#_Toc385318790)

[5.2.1 缩进 14](#_Toc385318791)

[5.2.2 数组和对象的初始化 15](#_Toc385318792)

[5.2.3 传递匿名函数 15](#_Toc385318793)

[5.2.4 空格 15](#_Toc385318794)

[5.2.5 空行 16](#_Toc385318795)

[5.2.6 二元和三元操作符 16](#_Toc385318796)

[5.3 括号 16](#_Toc385318797)

[5.3.1 小括号 16](#_Toc385318798)

[5.3.2 大括号 16](#_Toc385318799)

[5.3.3 引号 16](#_Toc385318800)

[5.4 陷阱和小技巧 16](#_Toc385318801)

[5.4.1 parseInt 16](#_Toc385318802)

[5.4.2 布尔表达式 17](#_Toc385318803)

[5.4.3 DOM节点ID 17](#_Toc385318804)

[5.4.4 遍历NodeList 17](#_Toc385318805)

[5.4.5 遗漏的参数 18](#_Toc385318806)

[5.4.6 HTML id 冲突 18](#_Toc385318807)

[6 内存编程规范 18](#_Toc385318808)

[6.1 DOM 18](#_Toc385318809)

[原则 2.1 在满足业务特性需求的情况下，减少DOM对象的动态创建和删除 18](#_Toc385318810)

[规则2.1 禁止在循环、事件操作回调中使用文本+变量拼接方式使用jQuery创建DOM 19](#_Toc385318811)

[建议2.1 删除Iframe时推荐使用innerHTML=“”的方式删除。 20](#_Toc385318812)

[6.2 事件监听 21](#_Toc385318813)

[原则3.1 正确使用DOM Event Level 2/3标准进行事件的绑定和注销 21](#_Toc385318814)

[规则3.1 页面卸载时要注销绑定的事件监听。 21](#_Toc385318815)

[规则3.2 动态删除的DOM对象在其销毁时要注销绑定的事件监听。 22](#_Toc385318816)

[规则3.3 禁止直接在HTML标签上嵌入JS事件回调代码。 23](#_Toc385318817)

[规则3.4 业务事件在业务销毁时注销 23](#_Toc385318818)

[6.3 定时器和延时器 23](#_Toc385318819)

[规则4.1 页面卸载时、页面局部刷新调整时，要注销相应的定时器和延时器。 23](#_Toc385318820)

[6.4 其他 24](#_Toc385318821)

[规则5.1 不在JS代码中使用console.log输出。 24](#_Toc385318822)

[规则5.2 使用完数组要将数组长度置空。 24](#_Toc385318823)

[规则5.3业务中禁止使用$digest()做数据更新。 24](#_Toc385318824)

[7 标准特性 24](#_Toc385318825)

[8 国际化规范 24](#_Toc385318826)

[9 web目录结构 24](#_Toc385318827)

# 背景

JavaScript 是一种客户端脚本语言，这份指南列出了编写 JavaScript 时需要遵守的规则。

本编程规范基于Google的JavaScript编程规范制定，并针对iEMP增加了若干增强要求。增强性要求使用以下格式表示。

# 兼容性要求

所有特性必须支持IE8以上、Firefox3.6以上,Chrome三种浏览器

# 术语定义

**原则：**编程时必须坚持的指导思想。

**规则：**编程时必须遵守的约定。

**建议：**编程时必须加以考虑的约定。

**说明：**对此原则/规则/建议进行必要的解释。

**错误示例：**对此原则/规则/建议从反面给出例子。

**推荐做法：**对此原则/规则/建议从正面给出例子。

**延伸阅读材料：**建议进一步阅读的参考材料。

# JavaScript 语言规范

## 方法定义

**原则：**定义类时，尽量在原型下定义方法，在构造方法内定义属性，从而最大程度发挥JS引擎优化机制。

**说明**：使用原型可以降低内存占用，提高运行效率。

**错误示例**：

function Foo () {

this.bar = "";

this.eat = function () {};

}

这样做会导致在每次实例化Foo对象时都新建一个eat方法。

**推荐做法**：

function Foo() {

this.bar = "";

}

Foo.prototype.eat = function () {};

## 闭包

**原则：**使用闭包时，应避免构成循环引用，防止内存泄露。

**说明：**闭包会持有其外部作用域（包括局部变量、参数及方法）的引用，当闭包本身又被作用域成员（常见于DOM对象）持有时便构成循环引用。

**错误示例：**

function Foo(element, a, b){

element.onclick = function () {

// 这里使用a、b

};

}

典型的闭包场景是定义内部方法，例中内部方法持有element、a和b的引用，但同时element又持有了内部方法。

**推荐做法一，不需要带参数：**

function Foo(element) {

element.onclick = bar;

}

function bar() {

}

**推荐做法二，带参数：**

function Foo(element, a, b) {

element.onclick = bar(a, b);

}

function bar(a, b) {

return function {

// 这里使用a、b

};

}

## eval()

**原则**：有替代方案时禁止使用eval方法。

**说明1**：eval()接收一个参数s，如果s不是字符串，则直接返回s。否则执行s语句。如果s语句执行结果是一个值，则返回此值，否则返回undefined。这样会让程序执行的比较混乱。当 eval() 里面包含用户输入的话存在安全风险，可以用其它更佳的, 更清晰, 更安全的方式写你的代码。

示例：

var code1='"a" + 2'; //表达式

var code2='{a:2}'; //语句

alert(eval(code1)); //->'a2'

alert(eval(code2)); //->undefined

alert(eval('(' + code2 + ')')); //->[object Object]

**说明2**：在微软的MSDN中（[http://support.microsoft.com/kb/2572253/zh-cn）已经确认了IE9](http://support.microsoft.com/kb/2572253/zh-cn）已经描述了IE8的兼容和IE9)浏览器下，使用eval()函数反复来构造对象从大 JavaScript 对象符号 (JSON) 字符串时，浏览器占用的内存使用量意外增加。

**建议方案**：在需要使用eval()函数反复来构造对象从大 JavaScript 对象符号 (JSON) 字符串时请使用JSON的parse方法来分析 JSON 字符串。例如：在解析返回的数据为：

**错误示例：**

var feed = '{"name": "Alice","id": 31502} ';

var userInfo = eval(feed);

var id = userInfo['id'];

**推荐做法：**

var feed = '{"name": "Alice","id": 31502} ';

var userInfo = JSON.parse(feed);

var id = userInfo['id'];

## with() {}

**原则**：禁止使用

**说明**：使用 with 让你的代码在语义上变得不清晰. 因为 with 的对象, 可能会与局部变量产生冲突, 从而改变你程序原本的用义.

**错误示例**：

var foo = {x: 5};

with (foo) {

var x = 3;

return x; // x = 5

}

.

## this

**原则**：仅在对象构造器, 方法, 闭包中使用this.

**说明**：

1. 在Javascript里面，this指针代表的是执行当前代码的对象的所有者。this 的语义很特别.

* 全局对象(大多数情况下)
* 调用者的作用域(使用 eval时)
* DOM 树中的节点(添加事件处理函数时)
* 新创建的对象(使用一个构造器)
* 其他对象(如果函数被 call() 或 apply())

1. 使用this时很容易出错, 所以只有在下面两个情况时才能使用：

* 在构造器中
* 对象的方法(包括创建的闭包)中

**推荐做法：**

Foo.prototype.foo = function () {

this.test();//当前

function callback(evt) {

alert(this);

}

$(document).on('click', this, callback);//click on事件的定义，this作为callback参数

callback();// this指向windows

callback = $.proxy(callback, this); // this为当前执行的Foo.prototype对象

callback(); // 打印 Foo.prototype对象

callback.call(this) // call方法this绑定到callback，打印出Foo.prototype

};

## 块内函数和块内变量声明

**原则：**块内函数必须使用函数表达式声明，块内变量不能与函数内的其他变量同名；

**说明：**ECMAScript不支持Block Scope，导致所有的Control Flow 都不是独立的作用域，其内部声明的变量或者函数都是在其父函数或者脚本的作用域中，导致块内函数或者变量的声明会存在覆盖现象，会造成代码逻辑错误；

**错误示例 一， 块内变量声明：**

// 大括号不是个作用域，输出为0

var acall = function (name) {

var x = 2; {var x = 0;} console.log(x);}()

// if、while、for内部块不是作用域，输出为0

var bcall = function (name) {

var x = 2; if(true){var x = 0;} console.log(x);}()

// try内部块不是作用域，输出为0

var ecall = function (name) {var x = 2; try{var x = 0;}catch(e){} console.log(x);}()

**错误示例 二， 块内函数声明：**

function bar(name) {

if (name === "hotel") {

// 1、定义一个foo函数，其作用域不是if代码块，而是bar函数作用域。

function foo() {console.log("hotel foo A");}

} else {

// 2、再重复定义一次foo函数，覆盖上面if条件分支下的foo函数定义。

function foo() {console.log("hotel foo 2");}

}

foo && foo();

}

bar("hotel"); // 输出结果总是显示"hotel foo 2"

**推荐做法一， 块内变量声明：**

var acall = function (name) {

var x = 2; {var y = 0;} console.log(x);}()

var bcall = function (name) {var x = 2; if(true){var y = 0;} console.log(x);}()

var ecall = function (name) {var x = 2; try{var y = 0;}catch(e){} console.log(x);}()；

// 上述代码输出的都是2；

**推荐做法二， 块内函数声明：**

function bar(name) {

var foo;

if (name == "hotel") {

foo = function() {console.log("hotel foo 1");};

}else {

foo = function () {console.log("hotel foo 2");}

}

foo && foo();

}

bar("hotel"); // 正确输出"hotel foo 1"

**延伸阅读材料：**

[**https://developer.mozilla.org/zh-CN/docs/JavaScript/Reference/Functions\_and\_function\_scope**](https://developer.mozilla.org/zh-CN/docs/JavaScript/Reference/Functions_and_function_scope)

[**https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Statements**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Statements)

[**http://javascriptweblog.wordpress.com/2010/07/06/function-declarations-vs-function-expressions/**](http://javascriptweblog.wordpress.com/2010/07/06/function-declarations-vs-function-expressions/)

## 分号

**规则：**每句代码后必须加";"

**说明：**遗漏分号有时会出现很奇怪的结果, 特别是当多个文件压缩成一个时，所以确保语句以分号结束.

错误示例：

a = b // 没有分号，会连着下一行一起解析，变成调用b(fn)();

(function(){

//....

})() // 自执行函数

推荐做法：

a = b; // 赋值

(function(){

//....

})(); // 自执行函数

## 变量

**规则：**声明变量必须加上 var 关键字.

**说明：**如果没有写 var, 变量就会暴露在全局上下文中, 这样很可能会和现有变量冲突

**错误示例：**

function a() {

url = 'http://rnd-iemp.huawei.com/'; // url会暴露到window上

}

a();

console.log(url) // 可以访问到，值为http://rnd-iemp.huawei.com/

**推荐做法**

var url = 'http://map.baidu.com/detail';

## 封装基本类型

**规则：**使用js字面量而不是封装基本类

**说明：**javascript中的基本类型对象只是值，不包含其封装对象的方法和属性等操作，在不需要使用属性和方法的时候不需要使用其封装类型。封装类型的使用有如下的几个注意事项：

1、封装类型Boolean的语义问题：在用作条件表达式中时，封装类型Boolean的语义与Object相同，结果都是true，而不是基本类型的true和false；

**错误示例：**

var x = new Boolean(false);

if (x) { // will be true

alert('hi'); // Shows 'hi'.

}

**推荐做法：**

var x = false;

if (x) {

alert(‘hi’);

}

2、Array的构造器的语义问题：Array的构造器行为与参数传入的个数和值关联，导致存在语义的不确定性；

**错误示例：**

// 长度为 3.

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

// 如果x1是自然数，会创建一个长度为x1的数组

// 如果x1是一个数字，但是非自然数，会抛异常

// 否则会创建长度为1的数组，值为x1

var a2 = new Array(x1);

// 长度为 0.

var a3 = new Array();

**推荐做法：**为了避免这些歧义, 我们应该使用更易读的直接量来声明.

var a1 = [x1, x2, x3];

var a2 = [x1];

var a3 = [];

3、Object类型：鉴于可读性和一致性考虑, 建议也使用直接量进行声明。

**错误示例：**

var o = new Object();

var o2 = new Object();

o2.a = 0;

o2.b = 1;

o2.c = 2;

o2['strange key'] = 3;

**推荐做法：**

var o = {};

var o2 = {

a: 0,

b: 1,

c: 2,

'strange key': 3

};

## 修改内置对象的原型

**规则：**禁止修改内置对象的原型。

**说明：**内置对象作为一套公共接口，具有约定俗成的行为方式，修改其原型，可能破坏接口语义。

**错误示例：**

Array.prototype.indexOf = function () {return -1}

// somewhere else

var o = [1, 1, 1, 1, 1, 2, 1, 1, 1];

alert(o.indexOf(2)); // 返回-1

## IE下的条件编译

**规则：**禁止IE下的条件编译。

**说明：**在IE下使用 @cc\_on 语句或使用 @if 或 @set 语句可以激活条件编译。尽管可以通过注释来兼容IE以外的浏览器，但它妨碍自动化工具的执行, 因为在运行时, 它们会改变 JavaScript 语法树。

**错误示例：**

var f = function () {

/\*@cc\_on @\*/

/\*@if (@\_jscript\_version >= 4)

alert("JavaScript version 4 or better");

@else @\*/

alert("Conditional compilation not supported by this scripting engine.");

/\*@end @\*/

};

## 多行字符串

**规则：** 禁止使用’\’的方式做长字符串的拼接。

**说明：**

1. 用’\’做拼接长字符串不是ECMAScript的规范，而且存在浏览器兼容性问题。
2. 使用字符串拼接在js压缩后会优化成单个字符串，不存在性能问题。

**错误示例：**

var myString = ‘first line \

second line \

third line.’;

// 这种写法得到的字符串第2、3行中会包含前面的空格。

**正确示例：**

var myString = ‘first line \n’+

‘second line \n’+

‘third line.’;

## 自定义 toString() 方法

**规则：**自定义的 toString() 方法应该总是成功调用且不要抛异常.

**说明：**可自定义 toString() 方法, 但确保你的实现方法满足: (1) 总是成功； (2) 没有其他负面影响。 如果不满足这两个条件, 那么可能会导致严重的问题, 比如, 如果 toString() 调用了包含 assert 的函数, assert 输出导致失败, 一些隐式的对象转字符串过程在调用toString() 时也就会出现错误。

## HTML与JS分离

**建议：**除了特殊业务需要（如要求极致高性能），都应该在html中通过外部js方式引入js代码

**说明：**html与js分离更便于代码维护，而且可以提高js的重用性

**错误示例：**

<head>

<script>

‘this is some business code...’;

</script>

</head>

**推荐做法：**

<body>

<script src=”/portal/themes/default/framework/js/framework.min.js”></script>

</body>

## 使用”use strict”

**建议：**在函数内部使用”use strict”

**说明：**

1、”use strict” 为ECMAscript 5新添加的一种js运行方式，主要有以下几个好处

- 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

- 消除代码运行的一些不安全之处，保证代码运行的安全；

- 提高编译器效率，增加运行速度；

- 为未来新版本的Javascript做好铺垫。

2、”use strict” 对js提出了更严格的语法，如果代码中使用"use strict"开启了严格模式,则下面的情况都会在脚本运行之前抛出[SyntaxError](https://developer.mozilla.org/zh-CN/docs/Core_JavaScript_1.5_Guide/SyntaxError)异常:

- 八进制语法:var n = 023和var s = "\047"

- [with](https://developer.mozilla.org/zh-CN/docs/JavaScript/Reference/Statements/with)语句

- 使用[delete](https://developer.mozilla.org/zh-CN/docs/JavaScript/Reference/Operators/delete)删除一个变量名(而不是属性名):delete myVariable

- 使用eval或arguments作为变量名或函数名

- 使用未来保留字(也许会在ECMAScript 6中使用):implements, interface, let, package, private, protected, public, static,和yield作为变量名或函数名

- 在语句块中使用函数声明:if(a<b){ function f(){} }

- 其他错误

对象子面量中使用两个相同的属性名:{a: 1, b: 3, a: 7}

函数形参中使用两个相同的参数名:function f(a, b, b){}

**示例：**

function demo() {

"use strict";

mistypedVariable = 17; // 抛出 ReferenceError，严格模式下不允许通过该方式定义全局变量

}

**延伸阅读材料：**<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions_and_function_scope/Strict_mode>

## 对象

**建议：**在访问某个外部对象或外部对象内的方法及属性时，必须先判断该对象是否为空，或该对象是否包含调用的方法及属性。

**说明：**如果是对外接口，必须遵守该规则。

**错误示例：**

function print(err) {

addToSysLog(err.msg); // err可能是undefined

}

**推荐做法：**

function print(err) {

if (err) {

addToSysLog(err.msg);

}

}

## 相等操作符

**建议：**做判断时使用 === 而不是 ==

**说明：**

1. js中做变量做判断会自动做类型转换，当类型确定时使用全等可以提高效率。
2. 注意：比较window对象时建议使用==。IE下window对象的判断比较特殊

window === top; // false

window === window.window // false

window == top; // true

window == window.window // true

**错误示例：**

if (name == ‘lucy’) {

jump();

}

**推荐做法：**

if (name === ‘lucy’) {

jump();

}

**延伸阅读材料：**<http://stackoverflow.com/questions/4850978/ie-bug-window-top-false>

## 多级原型结构

**建议:**在使用原型prototype实现继承时，尽量使用现有稳定且轻量的库方法而非自行实现。

**说明**：在实现继承甚至多层继承时，对原型的操作复杂性将显著会上升，使用现有稳定的库方法可避免不必要的编码失误。多级原型继承实现可参考:

**延伸阅读材料：**<http://www.amirharel.com/2010/06/11/implementing-multiple-inheritance-in-javascript/.>

## for-in 循环

**建议：** 使用for-in循环必须结合实际具体应用场景正确使用hasOwnProperty方法

**说明：**

1. for-in主要用于遍历对象的属性，但不包括对象的内置成员（如：toString、valueOf）。
2. 如果重写了Object的内置属性，使用for-in遍历在不同的浏览器上结果是不同的。
3. 如果往原生对象上加方法或者属性，会被遍历出来。如果不是业务需要遍历prototype内的属性，都要使用hasOwnProperty方法来提高代码的健壮度。
4. 遍历数组禁止使用for-in做遍历，请用普通的for循环做遍历

**示例：**

// A模块业务在代码某处重写了bind方法

Object.prototype.bind = function() {}

// B模块业务代码

var student = {

name: 'Lilei',

age: 18

};

for (var i in student) {

console.log(i);

}

// 打印 name age bind， 其中bind并不是预期属性，有可能导致代码出错

for (var i in student) {

if (student.hasOwnProperty(i)) {

console.log(i);

}

}

// 打印 name age。 使用hasOwnProperty可以使结果明确，使代码健壮

var obj = {

name: ‘king’,

toString: function(){}

};

for(var key in obj){

//1、IE6/7/8 下key值为：name

//2、IE9/Firefox/Chrome/Opera/Safari下key值为：name, toString

}

// 如非特殊场景，建议不要在实例上重写对象内置方法，避免浏览器兼容问题

遍历数组用最普通的 for 循环.

for(var i=0, len = arr.length;i < len; i++){

alert(arr[i]);

}

## 数组

**建议：** 尽量避免给数组添加属性

**说明：**在js中，数组也是对象，可以往数组上添加属性，但为了处理方便和避免出错，数组只用来存储有序（即索引连续）的一组数据。

// 用Array 定义数组

var myhash = new Array();

myhash[″key1″] = ″val1″;

myhash[″key2″] = ″val2″;

myhash[0] = ″222″;

for(var i in myhash) {

// i的值为 key1 key2 0

}

myhash.length //数组长度为1

# JavaScript 编码风格

## 命名

### 属性和方法

**规则：**

1、文件或类中的私有属性, 变量和方法名应该以下划线 "\_" 开头；

2、构造器函数必须大写字母开头，非构造器函数必须小写字母开头；

3、常量定义全部大写；

* iEMP增强规范要求：
* 函数名遵循驼峰风格；
* 变量名遵循驼峰风格,第一个字母小写；

**推荐做法：**

普通函数：\_myFunction()，calculateArea() , getFirstName()，

构造器函数：MyConstructor()

属性名：\_firstName, favoriteBands, oldCompanyName

全局不可变常量：var PI = 3.14,MAX\_WIDTH = 800;

### 常量

**规则：**常量的形式如: NAMES\_LIKE\_THIS, 即使用大写字符, 并用下划线分隔。

**错误示例：**

// 错误的命名

var hotleGetUrl = 'http://map.baidu.com/detail',

placeType = 'hotel';

**推荐做法：**

// 正确的命名

var HOTEL\_GET\_URL = 'http://map.baidu.com/detail',

PLACE\_TYPE = 'hotel';

### Getters 和 Setters 命名

**规则：**Getters 和 Setters 并不是必要的. 但只要使用了, 就请将 getters 命名成 getFoo() 形式, 将 setters 命名成 setFoo(value) 形式。对于布尔类型的 getters, 使用 isFoo() 。

**推荐做法：**

普通参数的Getter/Setter方法定义：getFoo()，setFoo(value)

Boolean类型的Getter/Setter方法定义：isReady()，setReady(value)

### 为全局代码使用命名空间

**规则：**在全局作用域上, 使用一个唯一的, 与工程/库相关的名字作为前缀标识.

**推荐做法：**

比如, 你的工程是 " eview", 那么命名空间前缀可取为 eview.\*.

var eview = {};

eview.sleep = function() {

...

};

## 代码格式化

### 缩进

**规则：**统一采取4空格缩进。

**说明：**不要依赖编辑器的默认tab行为。使用四空格缩进，或者预先设置编辑器的tab键为4空格。函数体、循环语句，条件语句、Switch语句，对象中属性字符定义都必须遵循。

**推荐做法:**

someValue = this.foo(

shortArg,

'Some really long string arg - this is a pretty common case, actually.',

shorty2,

this.bar());

### 数组和对象的初始化

**建议：**如果初始值不是很长, 就保持写在单行上。

**推荐做法：**

var arr = [1, 2, 3];

var obj = {a: 1, b: 2, c: 3};

### 传递匿名函数

**规则：**如果参数中有匿名函数, 函数体从调用该函数的左边开始缩进4个空格, 而不是从 function 这个关键字开始，这让匿名函数更加易读。

**错误示例：**

var names = items.map(function(item) {

return item.name;

});

**推荐做法：**

var names = items.map(function(item) {

return item.name;

});

### 空格

**规则：**

// 循环语句中分号后带个空格：

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

// 数组初始化时逗号后带个空格：

var a = [1, 2, 3]

// 同一行对象属性之间带个空格：

var o = {a: 1, b: 2};

// 函数调用多个参数之间带个空格：

myFunc(a, b, c)

// 函数声明时函数大括号之前带个空格：

function myFunc() {}

// 匿名函数function后面带个空格：

var fn = function () {}

// 操作符前后带个空格：

+, -, \*, =, <, >, <=, >=, ===, !==, &&, ||, +=

### 空行

**规则：**使用空行来划分一组逻辑上相关联的代码片段。

**推荐用法：**

doSomethingTo(x);

doSomethingElseTo(x);

andThen(x);

nowDoSomethingWith(y);

andNowWith(z);

### 二元和三元操作符

**规则：**操作符始终跟随着前行,如果一行实在放不下,按照上述的缩进风格换行。

**说明：**操作符始终跟随着前行, 这样就不用顾虑分号的隐式插入问题。

**推荐用法：**

var x = a ? b : c;

// 换行时‘？’跟随在前行

var y = a ?

longButSimpleOperandB : longButSimpleOperandC;

// 换行时‘:’跟随在前行

var z = a ?

moreComplicatedB :

moreComplicatedC;

## 括号

### 小括号

**规则：**不要滥用小括号, 只在必要的时候使用它。

**说明：**对于一元操作符(如delete, typeof 和 void ), 或是在某些关键词(如 return, throw, case, new )之后, 不要使用括号。

**推荐做法：**

if (typeof obj === ‘object’)

### 大括号

**规则：**大括号开始紧随语句，不独立一行；if 、 while 、 do 、for 语句必须有大括号，即使只有一行代码

**推荐做法：**

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

console.log(i);

}

### 引号

**建议：** 使用单引号优于双引号

**说明：** 单引号优于双引号 . 当你创建一个包含 HTML 代码的字符串时就知道它的好处了.

var msg = 'This is some HTML';

## 陷阱和小技巧

### parseInt

**原则：**parseInt方法必须指定基数。

**说明**：如果不指定parseInt的第二个参数，如果前缀为“0x”，按16进制转换；对前缀为“0”的字符串，在chrome和firefox下，默认使用10进制转换，但在IE浏览器下按照8进制转换。**错误示例**：

parseInt('011') //chrome下返回11；IE下返回9

**推荐做法**：

parseInt('011'， 10)

### 布尔表达式

**建议：**在需要判断某变量不是0，空串或者false时，使用if(x)。

**说明：**JavaScript是弱类型的，在布尔表达式中，会转换类型为Boolean类型进行判断。

0

-0

null

“” 空串

false

undefined

NaN

对于上面7个值，if(x)返回false。

‘0’ 字符串0

[] 空数组

{} 空对象

对于上面3个值，返回的是true。

**错误示例**：

if (x != null) {...} //尽量不使用，可能判断条件不完整

**推荐做法**：

if (x) {...}

### DOM节点ID

**规则：**DOM节点的ID命名使用\_、英文字母和数字，避免使用. < >等其他特殊字符

**说明：**DOM节点的ID常会用作CSS等的选择器，特殊字符可能会与选择器的语法冲突。

**错误示例**：

<div id="eview.div"></div>

### 遍历NodeList

**建议：**遍历NodeList时，缓存NodeList的length。

**说明**：NodeList是通过给节点迭代器加一个过滤器来实现的，这表示获取他的属性，如length的时间复杂度为O(n)，通过length来遍历整个列表需要O(n^2)。

**错误示例**：

var paragraphs = document.getElementsByTagName('p');

for (var i = 0; i < paragraphs.length; i++) {

doSomething(paragraphs[i]);

}

**推荐做法**：

var paragraphs = document.getElementsByTagName('p');

for (var i = 0, paragraph; paragraph = paragraphs[i]; i++) {

doSomething(paragraph);

}

**或者：**

var parentNode = document.getElementById('foo');

for (var child = parentNode.firstChild; child; child = child.nextSibling){

doSomething(child);

}

### 遗漏的参数

**建议：**函数的可选参数或者新增参数设置默认值

**说明：**如果你需要在已经被调用的函数中增加一个参数来处理一个特殊情况下的调用，请给这个函数中的这个参数设置默认值，以防万一在众多脚本中的众多调用中的一个忘记更新。

**推荐做法**：

// country是新增的参数

function addressFunction(country){

country = country || “US”; //如果没有传入country,给出默认值“US”

//剩下代码

}

### HTML id 冲突

**建议：**尽量不要使用Html中的Id作为JavaScript的变量名。

**说明**：在JavaScript中函数和属性共享同一个名字空间。所以，当在HTML中的一个id和函数或属性有相同的名字时，你会得到难以跟踪的逻辑错误。

**错误示例**：

<html>

<head></head>

<body>

<ul>

<li id="length">1</li>

<li id="thisLength">2</li>

<li id="thatLength">3</li>

</ul>

</body>

<script>

var listitems = document.getElementsByTagName('li');

var liCount = listitems.length; //IE下返回的是<li id="length">1</li>这个节点而不是所有<li>的数量

var thisLength = document.getElementById('thisLength');

thatLength = document.getElementById('thatLength');

//IE下会出现“对象不支持此属性和方法”的错误，IE8 beta2下首次加载页面会出错，刷新页面则不会

//在IE中thisLength和thatLength直接表示以其为id值的DOM节点，

//所以赋值时会出错，当有var声明时，IE会把其当着变量，这个时候就正常了。

</script>

</html>

# 内存编程规范

## DOM

### 原则 2.1 在满足业务特性需求的情况下，减少DOM对象的动态创建和删除

**说明**：DOM对象的动态创建和删除会导致浏览器频繁的分配和释放内存，不同的浏览器在DOM对象的垃圾回收上采用的垃圾回收算法不同，导致DOM对象的动态创建和删除所使用的内存完全不同，

减小DOM对象的动态创建和删除，降低内存的频繁操作，不仅可以有效减少内存泄漏的大小，也有利于提升运行效率。

### 规则2.1 禁止在循环、事件操作回调中使用文本+变量拼接方式使用jQuery创建DOM

**说明**：jQuery 在内部实现中为了提高频繁的DOM创建的性能，会使用[DocumentFragment](#_W3C:DOM-Level-3-Core)缓存用户创建的DOM，在重复创建的过程中使用缓存的DOM对象clone创建DOM，从而提高DOM创建的效率，

但是这一优化，仅适用于用户频繁创建的DOM内容没有变化的情况，对于内容不断变化的话，在页面没有卸载的情况下长时间运行的时候会有严重的内存问题。

涉及的jQuery 函数如下：

$ 函数

$.before\$.after 函数

$.append\$.prepend 函数

常见的几种DOM创建方式：

（1）单个DOM对象：

大家通常用字符串文本与JS变量来拼接一个HTML代码，然后使用jQuery创建DOM对象，该方式功能完整，代码简洁，但是却有一个由于[jQuery bug](http://bugs.jquery.com/ticket/10682) 引发的内存泄漏问题（该问题在jQuery 1.9.1版本中才解决）。因此，禁止在循环代码以及事件回调函数中使用该方式创建DOM。

错误示例：

$('#btn').on('click', function() {

var p = $('#hostElement');

var content = getValue()；// 获取显示文本信息

var $span = $('<span>' + content + '</span>');// 拼接HTML文本，创建DOM对象。

/\*

从别处获取显示内容，值可能相同也可能不同，不同的content会造成在jQuery.fragments变量中保存了一个key 为'<span>' + content + '</span>'，value 为1 的键值对，当该key再次出现时，jQuery.fragments该键值对就会变成 （'<span>' + content + '</span>'，DocumentFragment对象）这样的键值对，在循环或者事件回调中使用这种方式后，便会造成Query.fragments 会积累越来越多的 DocumentFragment对象，造成在页面不F5刷新的情况下，内存上涨无法被回收。

\*/

p.append($span);

});

推荐做法：

$('#btn').on('click', function() {

var p = $('#hostElement');

var content = getValue()；// 获取显示文本信息

var $span = $('<span></span>').html(content); //创建SPAN对象，设置显示文本

/\*

该方式下，jQuery会不会生成[DocumentFragment](#_W3C:DOM-Level-3-Core)对象，也不会缓存DOM对象在其内部变量中，因此重复、频繁使用时不会造成因为DOM泄漏产生的内存上涨问题

\*/

p.append($span);

});

（2）HTML代码模板：

在代码中通常有较多的HTML代码模板， 通过使用模板引擎编译后再传入模板参数得到html代码字符串。在通过jQuery生成DOM时，由于jQuery的内部变量jQuery.fragments对象中会保存生成的[DocumentFragment](#_W3C:DOM-Level-3-Core)对象，当长时间持续运行的时候，会逐渐累积，且HTML模板通常是包含较多的DOM 节点，所以占用的内存会比单个DOM要多很多。因此，不能使用该方式直接创建DOM，要做调整使用。

具体方法：先将html字符串使用innerHTML或者jQuery的html方法设置到一个临时的DOM下，再将该DOM下所有孩子移动到目标位置。

错误示例：

//使用HandleBars 将html模板编译为函数。

var template = compile('<span>{{content}}</span>');

$('#btn').on('click', function() {

var p = $('#hostElement');

//获取动态显示的内容。

var i = getValue()；

// 使用handlebar 编译的函数生成 html 代码。

var html = template({content:i})；

// 使用jQuery生成DOM对象。

var $span = $(html);

p.append($span);

});

推荐做法：

//使用HandleBars 将html模板编译为函数。

var template = compile('<span>{{content}}</span>');

$('#btn').on('click', function() {

//获取一个临时DOM对象容器并隐藏，将生成的DOM对象先放置与该DOM节点下，再移出。

var docTemp = $('#tempcontainer').hide();

//获取动态显示的内容。

var i = getValue();

// 使用handlebar 编译的函数生成 html 代码。

var htmlitem = template({content:i});

// 将HTML代码放置于临时DOM节点下。

docTemp.html(htmlitem);

// 将临时DOM节点下所有子DOM对象移出至目标DOM对象。

var $span = docTemp.children();

p.append($span);

});

### 建议2.1 删除Iframe时推荐使用innerHTML=“”的方式删除。

**说明：**iframe在使用removeChild方法，在IE8某些版本中无法触发unload事件的产生，导致iframe中unload事件监听和回调函数不能正常执行，从而导致iframe中内存泄漏。Iframe适用于第三方集成，文件异步上传等业务，非此类业务时，建议使用普通HTML方式构建，而不使用iframe。

错误示例：

function \_removeIframes(container) {

$('iframe', container.document).each(function() {

try{

var win = this.window || this.contentWindow;

if (win) {

\_removeIframes(win);

// 1、可能因为跨域造成权限不够，无法访问而抛出异常的问题

// 2、\_removeIframes 通常在iframe的unload事件回调中处理，但是当多个业务注册了unload事件监听后，这里使用document.write('')会导致后续其他unload事件监听函数执行抛出异常，原因为，document 已经被清空，但是可能一些组件代码中还持有DOM对象的引用，那么这时访问这些DOM对象的属性时会发生拒绝访问错误。

win.document.write('');

win.document.close();

if (this.parentNode) {

this.parentNode.removeChild(this);

}

}

}catch(e){

}

});

};

推荐做法：

var temp = document.createElement('div');

$(temp).hide();

//DOM追加的先后顺序在IE8、IE9、firefox、chrome上其内存和性能表现没有差异，但是如果被其他操作将两个操作隔离开来的话，将会导致reflow的增加，从而影响性能表现。

temp.appendChild(ifr);

document.appendChild(temp);

temp.innerHTML='';

document.removeChild(temp);

## 事件监听

### 原则3.1 正确使用DOM Event Level 2/3标准进行事件的绑定和注销

说明：采用非标准的事件绑定方式，容易受浏览器的特殊实现的影响，造成内存无法释放问题。在iEMP中由于使用jQuery组件进行事件操作，jQuery组件会自动选择浏览器支持的最高事件模型来绑定事件和注销事件。

DOM Event Level 2 规格： <http://www.w3.org/TR/DOM-Level-2-Events/>

DOM Event Level 3 规格： <http://www.w3.org/TR/DOM-Level-3-Events/>

### 规则3.1 页面卸载时要注销绑定的事件监听。

说明：通常事件监听都是与DOM对象挂钩的，当事件监听没有被注销时，可能会造成JS对象和DOM对象之间存在引用关系，当JS对象或者DOM对象没有释放时，也会造成另外一方无法释放，尤其在大量使用闭包的情况下，情况会比较复杂，容易引入事件回调闭包造成JS对象和DOM的循环引用，使用闭包造成的循环引用在IE、firefox上会造成内存泄漏，原因是由于IE和Firefox都使用引用计数方式的[垃圾回收算法](#_Eric_Lippert:_How)。

错误示例：

$(function() {

var btn = $('<button id="btn">toggle target</button>');

btn.appendTo($(document.body));

btn.on('click', function onButtonClick() {

$('#target').toggle();

});

btn = null;

});

//没有事件的注销动作，在IE或者firefox上会造成循环引用 button 这个DOM对象 和 onButtonClick 这个闭包的循环引用。

推荐做法：

$(function() {

var btn = $('<button id="btn">toggle target</button>');

btn.appendTo($(document.body));

btn.on('click', function() {

$('#target').toggle();

});

btn = null;

});

$(window).on('unload', function() {

//添加销毁动作，并注销事件。

$('#btn').off();

});

### 规则3.2 动态删除的DOM对象在其销毁时要注销绑定的事件监听。

说明：通常事件监听都是与DOM对象挂钩的，当事件监听没有被注销时，可能会造成JS对象和DOM对象之间存在引用关系，当JS对象或者DOM对象没有释放时，也会造成另外一方无法释放，尤其在大量使用闭包的情况下，情况会比较复杂，容易引入事件回调闭包造成JS对象和DOM的循环引用，闭包循环引用在IE、firefox上会造成内存泄漏，原因是由于IE和Firefox都使用引用计数方式的垃圾回收算法。

错误示例：

$('#btn\_create').on('click', function() {

var btn = $('<button id="btn">toggle target</button>');

btn.appendTo($(document.body));

btn.on('click', function() {

$('#target').toggle();

});

btn = null;

});

推荐做法：

$('#btn\_create').on('click', function() {

var btn = $('<button id="btn">toggle target</button>');

btn.appendTo($(document.body));

btn.on('click', function() {

$('#target').toggle();

});

btn = null;

});

//添加动态销毁动作，并注销事件。

$('#btn\_destroy').on('click', function() {

$('#btn').off().remove();

});

### 规则3.3 禁止直接在HTML标签上嵌入JS事件回调代码。

说明：直接将JS字符串作为DOM 的事件处理逻辑，容易引发IE8版本下无法释放内存的问题。虽然BME中在动态创建的HTML代码中存在该类型的事件回调函数，并没有发现严重问题，但是禁止在iEMP的业务业务代码中直接使用该方式注册事件函数。

错误示例：

<div id='container' onclick='showmessage();'></div>

function showmessage() {

alert('Not supported');

}

推荐做法：

<div id='container'></div>

…

JS 文件中：

$('#container').on('click', function() {

alert(this.innerHTML);

});

### 规则3.4 业务事件在业务销毁时注销

错误示例：

//注册API事件回调

$.addEventListener('com.huawei.oms.fm.event.alarmSound', modifySound);

推荐做法：

//注册API事件回调

$.addEventListener('com.huawei.oms.fm.event.alarmSound', modifySound);

//在销毁时注销业务事件

$.removeEventListener('com.huawei.oms.fm.event.alarmSound', modifySound)

## 定时器和延时器

### 规则4.1 页面卸载时、页面局部刷新调整时，要注销相应的定时器和延时器。

说明：定时器和延时器容易造成JS 对DOM的引用，在JS对象本身没有被释放的情况下容易引起DOM对象的泄漏。

在页面局部刷新时必须清理响应的定时器，否则会造成多个定时器同时执行的业务问题和每个定时器对应闭包所引用的内存无法释放的问题。

错误代码：

var intervalID = setInterval(fn, timeout);

正确代码：

var intervalID = setInterval(fn, timeout);

…

//清理函数中要对定时器进行销毁

if (intervalID) {

clearInterval(intervalID);

}

## 其他

### 规则5.1 不在JS代码中使用console.log输出。

说明：console 输出会导致浏览器内存上涨，在IE下，只要打开过F12，console就会被激活，后面即使关闭F12工具也无法去激活console，内存依然会占用。

### 规则5.2 使用完数组要将数组长度置空。

说明：JS是通过引用传参，需要在接口中或者模块中明确数组对象的生命周期责任，否则可能会造成一处已经至NULL，而别处仍然持有引用，导致内存没有释放。

### 规则5.3业务中禁止使用$digest()做数据更新。

说明： $digest()方法为angular的接口方法，但是存在内存泄漏问题，需要修改为接口$apply(exp)。通过正确接口触发正确的scope生命周期管理逻辑，来保证AngularJS自身对象管理的正确性。

# 标准特性

**原则：**当同时存在标准特性与非标准特性时，优先使用标准特性

说明：参考附件[重要的浏览器差异.xlsx](file:///C:\Users\f00187484\Downloads\重要的浏览器差异.xlsx)

# 国际化规范

每个模块前台和后台资源文件分离，前台HTML只加载前台的资源文件。

所有给前台的ROA接口返回值都必须带国际化信息

# web目录结构

***Webroot***

themes--------------------------------------------------主题

default---------------------------------------------主题一：默认主题

commonJs ---------------------------包括所有的公共类的集合

{module Name}---------------------前台框架模型/业务模块

css---------------------- css集合

images----------------图片集合

demo

test

xxxx.js

xxxx.js

components-----------------------组件部分集合

basicComponent------------基础组件集合

css--------------------- css集合

images----------------图片集合

demo------------------基础组建的demo集合（包含html）

test---------------------基础组件单元测试

xxx.js------------------基础组件类（每个组件对应一个类）

yyy.js

zzzcomponent-------------复杂，特制组件（单独组件分开）

不是所有的复杂组件杂合在一起，

每个复杂组件一个单独的文件目录

css---------------------- css集合

images----------------图片集合

demo -----------------当前组件的实例目录

test--------------------当前组件的单元测试目录

xxx.js -----------------当前组件的js文件

yyy.js

*打包后的文件目录结构*

Webroot

themes--------------------------------------------------主题

default---------------------------------------------主题一：默认主题

Skins

Green

{module Name}----------------前台框架模型/业务模块

css

images

components-----------------------组件部分集合

basicComponent------------基础组件集合

css----------------------css集合

images----------------图片集合

zzzcomponent-------------

Css -----------------当前组件的私有css目录

Images

Demo

Demo.html

blue

commonJs ---------------------------包括所有的公共类的集合

common.min.js

{module Name}-----------------------前台框架模型/业务模块

xxxx.js

components-----------------------组件部分集合

basicComponent------------基础组件集合

basic.min.js

components -------------复杂，特制组件（单独组件分开）

不是所有的复杂组件杂合在一起，

每个复杂组件一个单独的文件目录

charts.min.js