# JavaScript 代码规范

# 申明

本文档属于上海数凹科技有限公司，只限于内部使用。如果违反，公司将保留追究泄密的权利。

|  |  |  |
| --- | --- | --- |
| 日期 | 作者 | 备注 |
| 2016-06-27 | Andy.zhou | 创建文档 |
|  |  |  |

目录

[JavaScript 代码规范 1](#_Toc454805209)

[申明 2](#_Toc454805210)

[1 文档概述 5](#_Toc454805211)

[1.1 文档目标 5](#_Toc454805212)

[1.2 编程风格和原则 5](#_Toc454805213)

[1.2.1 编程风格 5](#_Toc454805214)

[1.2.2 规范原则 5](#_Toc454805215)

[1.3 规范分类 6](#_Toc454805216)

[2. JS代码规范 7](#_Toc454805217)

[2.1 JavaScript语言规范 7](#_Toc454805218)

[2.2 JavaScript 编码风格 7](#_Toc454805219)

[2.2.1 行与缩进 7](#_Toc454805220)

[2.2.2括号 9](#_Toc454805221)

[2.2.3变量 9](#_Toc454805222)

[2.2.4字符串 9](#_Toc454805223)

[2.2.5命名规范 9](#_Toc454805224)

[2.3 Douglas Crockford JavaScript规则 10](#_Toc454805225)

[2.3.1大括号的位置 10](#_Toc454805226)

[2.3.2圆括号 11](#_Toc454805227)

[2.3.3 分号 11](#_Toc454805228)

[2.3.4 with语句 11](#_Toc454805229)

[2.3.5 相等和严格相等 11](#_Toc454805230)

[2.3.6 语句的合并 11](#_Toc454805231)

[2.3.7 变量的申明 12](#_Toc454805232)

[2.3.8 全局变量 13](#_Toc454805233)

[2.3.9 new命令 13](#_Toc454805234)

[2.3.10 区块 14](#_Toc454805235)

[2.3.11 自增、自减 14](#_Toc454805236)

[3 Node JS 代码规范 16](#_Toc454805237)

[3.1建议from webnet 16](#_Toc454805238)

[4 JS注释规范 21](#_Toc454805239)

[4.1 File Example 21](#_Toc454805240)

[4.2 Function Example 21](#_Toc454805241)

[参考资料 22](#_Toc454805242)

# 1 文档概述

## 1.1 文档目标

本文档是描述JavaScript的代码规范，规范分为两类，必须follow和建议follow。

## 1.2 编程风格和原则

### 1.2.1 编程风格

**所谓"编程风格"（programming style），指的是编写代码的样式规则。**不同的程序员，往往有不同的编程风格。

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](http://www.ruanyifeng.com/blog/2010/01/12_javascript_syntax_structures_you_should_not_use.html" \t "_blank)的语言尤其重要。

### 1.2.2 规范原则

规范的原则是为了团队更高效的开发和维护高质量的代码。

原则一：质量为先，容易维护；

原则二：逻辑结构清晰、优雅、简洁，易于阅读和维护；

原则三：风格一致，团队成员可以开心的码代码，维护代码，好心情好代码；

原则四：遵循必须的规则，其它的规则尽量follow；

## 1.3 规范分类

|  |  |  |
| --- | --- | --- |
| 词语 | 含义 | 说明 |
| M | Must | 必须遵守 |
| SS | Strong suggest | 强烈建议， |
| S | suggest | 建议，好的代码风格，不过可以考虑自身的习惯 |
| O | other | 其它的，能遵从最好，但更多是习惯问题 |

表1.3 规范分类

# 2. JS代码规范

## 2.1 JavaScript语言规范

* 声明变量时, 必须加上 var 关键字. M
* 尽量减少全局变量的使用. SS
* 语句总是以分号结尾. M
* 不要在块内声明函数. SS
* 标准特性优于非标准特性(如果类库有提供, 优先使用类库中的函数). SS
* 不要封装基本类型. M
* 只在解析序列化串时使用 eval() . SS
* 禁止使用 with . M
* 减少使用 continue 和 break . S
* 仅在函数内使用 this . M
* 使用 Array/Object 直接量, 避免使用 Array/Object 构造器. SS
* 禁止修改内置对象的原型. M

## 2.2 JavaScript 编码风格

### 2.2.1 行与缩进

#### 2.2.1.1语句行

1. 尽可能不要让每行超过 120 个字符; SS
2. 语句必须以分号作为结束符, 不要忽略分号; M

#### 2.2.1.2空格

1. 数值操作符(如, +/-/\*/% 等)两边留空; SS
2. 赋值操作符/等价判断符两边留一空格; SS
3. for 循环条件中, 分号后留一空格; SS
4. 变量声明语句, 数组值, 对象值及函数参数值中的逗号后留一空格; SS
5. 空行不要有空格; S
6. 行尾不要有空格; S
7. 逗号和冒号后一定要跟空格; SS
8. 点号前后不要出现空格; M
9. 空对象和数组不需要填入空格; S
10. 函数名末尾和左括号之间不要出现空格; SS

#### 2.2.1.3空行

1. 逻辑上独立的代码块使用空行分隔; M
2. 文件末尾留 1~2 个空行; M
3. 不要吝啬空行. 尽量使用空行将逻辑相关的代码块分割开, 以提高程序的可读性. SS

#### 2.2.1.4缩进

1. 以 4 个空格为一缩进层次; SS
2. 变量声明:
   * 多个变量声明时, 适当换行表示; SS
   * 参照 var 关键字, 缩进一层次; M
3. 函数参数:
   * 函数参数写在同一行上; SS
   * 传递匿名函数时, 函数体应从调用该函数的左边开始缩进; SS
4. 数组和对象初始化时:
   * 如果初始值不是很长, 尽量保持写在单行上; SS
   * 初始值占用多行时, 缩进一层次; SS
   * 对象中, 比较长的变量/数值, 不要以冒号对齐; SS
5. 二元/三元操作符:
   * 操作符始终跟随前行; SS
   * 实在需要缩进时, 按照上述缩进风格; SS
6. 表达式中的缩进同变量声明时;

### 2.2.2括号

原则: 不要滥用括号, 必要时一定要使用.

1. if/else/while/for 条件表达式必须有小括号; M
2. 语句块必须有大括号; M
3. 一元操作符(如 delete, typeof, void)或在某些关键词 ( 如 return, throw, case, new ) 之后, 不要使用括号; SS

### 2.2.3变量

1. 变量如有较广的作用域, 使用全局变量; 如果是在类中, 可以设计成为一个类的成员; SS
2. 函数体中, 多个局部变量集中在一起声明, 避免分散; S
3. 适当延迟变量的初始化; S

### 2.2.4字符串

1. JS 代码中, 单行字符串使用单引号; M
2. JS 代码中, 多行字符串使用 + 拼接形式, 不要使用 \ 拼接; M
3. HTML 中 Element 属性, 使用双引号; M

### 2.2.5命名规范

原则： \* 尽量避免潜在冲突; \* 精简短小, 见名知意;

1. 普通变量统一使用驼峰形式; SS
2. 常量使用全部大写, 多个单词以下划线分隔; M
3. 枚举量, 同常量; M
4. 私有变量, 属性和方法, 名字以下划线开头; SS
5. 保护变量, 属性和方法, 名字同普通变量名; SS
6. 方法和函数的可选参数, 名字以 opt\_ 开头, 使用 @param 标记说明; SS
7. **方法和函数的参数个数不固定时: SS**
   * 可添加参数 var\_args 为参数个数;
   * 取代使用 arguments;
   * 使用 @param 标记说明;
8. **Getter/Setter 命名: SS**
   * 以 getFoo/setFoo(value) 形式;
   * 布尔类型使用 isFoo()/hasFoo()/canDo()/shouldDO() 也可;
9. **命名空间: SS**
   * 为全局代码使用命名空间, 如 sloth.\*;
   * 外部代码和内部代码使用不同的命名空间;
10. 重命名那些名字很长的变量, 不要在全局范围内创建别名, 而仅在函数块作用域中使用; SS
11. 文件名应全部使用小写字符, 且不包含除 - 和 \_ 外的标点符号; SS
12. 临时的重复变量建议以 i, j, k, ..., 命名; SS

建议阅读google JavaScript 代码规范

<http://docs.kissyui.com/1.4/docs/html/tutorials/style-guide/google-js-style.html>

## 2.3 [Douglas Crockford](http://www.crockford.com/) JavaScript规则

### 2.3.1****大括号的位置****

**规则1：表示区块起首的大括号，不要另起一行。 M**

**理由：**

这条规则的原因是JavaScript可能自动给空行加 ；

### ****2.3.2圆括号****

**规则2：调用函数的时候，函数名与左括号之间没有空格。 SS**

**规则3：函数名与参数序列之间，没有空格。 SS**

**规则4：所有其他语法元素与左括号之间，都有一个空格。SS**

**理由：**

圆括号（parentheses）在Javascript中有两种作用，一种表示调用函数，另一种表示不同的值的组合（grouping）。我们可以用空格，区分这两种不同的括号。

### ****2.3.3 分号****

**规则5：不要省略句末的分号。M**

理由：

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。

### ****2.3.4 with语句****

**规则6：不要使用with语句。M**

理由：with可以减少代码的书写，但是会造成混淆。

### 2.3.5 相等和严格相等

与编译性语言不通，解释性语言的类型转换更加随意和自由。Javascript有两个表示"相等"的运算符："相等"（==）和"严格相等"（===），"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](http://www.2ality.com/2011/06/javascript-equality.html" \t "_blank)。

**规则7：不要使用"相等"（==）运算符，只使用"严格相等"（===）运算符。M or SS**

### ****2.3.6 语句的合并****

**规则8：不要将不同目的的语句，合并成一行。 M**

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;

### 2.3.7 变量的申明

**规则9：所有变量声明都放在函数的头部。 SS**

**规则10：所有函数都在使用之前定义。 M**

**理由：**

Javascript会自动将变量声明"提升"（hoist）到代码块（block）的头部。

　　if (!o) {

　　　　var o = {};

　　}

等同于

　　var o;

　　if (!o) {

　　　　o = {};

　　}

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 ...) {...,}

### 2.3.8 全局变量

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。

**规则11：避免使用全局变量；如果不得不使用，用大写字母表示变量名，比如g\_UPPER\_CASE。SS**

### 2.3.9 new命令

Javascript使用new命令，从建构函数生成一个新对象。

　　var o = new myObject();

这种做法的问题是，一旦你忘了加上new，myObject()内部的this关键字就会指向全局对象，导致所有绑定在this上面的变量，都变成全部变量。

Js在用new关键字的时候，会调用构造函数。对于下一条规则，还是建议为主，使用new还是非常明确的。

**规则12：不要使用new命令，改用**[Object.create()](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/create" \t "_blank)**命令。 S**

如果不得不使用new，为了防止出错，最好在视觉上把建构函数与其他函数区分开来。

因此以下这条是我们必须要遵守的。

**规则13：建构函数的函数名，采用首字母大写（InitialCap）；其他函数名，一律首字母小写。M**

### 2.3.10 区块

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();

因此，

**规则14：总是使用大括号表示区块。 M**

### 2.3.11 自增、自减

自增（++）和自减（--）运算符，放在变量的前面或后面，返回的值不一样，很容易发生错误。

事实上，所有的++运算符都可以用"+= 1"代替。

　　++x

等同于

　　x += 1;

代码变得更清晰了。有一个很可笑的例子，某个JavaScript函数库的源代码中出现了下面的片段：

　　++x;

　　++x;

这个程序员忘了，还有更简单、更合理的写法：

　　x += 2;

因此，

**规则15：不要使用自增（++）和自减（--）运算符，用+=和-=代替。S**

现实中，自增和自减是非常常见的，所以，我们要做的是更加漂亮的使用，而不是不用。

<http://javascript.crockford.com/code.html>

<http://www.ruanyifeng.com/blog/2012/04/javascript_programming_style.html>

# 3 Node JS 代码规范

## 3.1建议from webnet

1. 缩进2 个 space，tab要转换成 2 space. (入乡随俗，这是Node 源码及module 采用的标准) [Eclipse设置](http://ww3.sinaimg.cn/large/6cfc7910jw1dnf44jzellj.jpg). O (这个一般没啥太大关系)
2. 永远用 var 声明变量，不加 var 时，会污染顶层上下文。 M
3. 操作符与操作算子之间要有空格. SS

Right：

var string = 'Foo' + bar;

Wrong:

var string = 'Foo'+bar;

1. 使用string 时，用单引号替代双引号（写JSON 时除外）M

Right：

var foo = 'bar';

var http = require('http');

Wrong：

var foo = "bar";

var http = require("http");

1. 大括号位置。 SS or M

Right：

if (true) {

console.log('winning');

}

Wrong：

if (true)

{

console.log('losing');

}

1. Camel 命名法. SS or M

采用以下规则：

函数和变量：functionNamesLikeThis, variableNamesLikeThis,

类名和枚举类型：ClassNamesLikeThis, EnumNamesLikeThis,

类方法：methodNamesLikeThis

常量：SYMBOLIC\_CONSTANTS\_LIKE\_THIS

Right ：

// var definition

var adminUser = db.query('SELECT \* FROM users ...');

// function definition

function run() {

}

// Class definition

function BankAccount() {

}

Wrong：

var admin\_user = db.query('SELECT \* FROM users ...');

function bankAccount() {

}

1. 文件命名 SS

单词之间使用 \_ (underscore) 来 分割，如果你不想暴露某个文件给用户 ， 你也可以用\_ 来开头

Right :

child\_process.js

string\_decoder.js

\_linklist.js

1. 不使用 const 关键字 虽然V8 和 Mozilla 都支持它，但它不是ECMA 标准，我们用以下方式定义常量：SS (我们一般需要用const)

Right：

var SECOND = 1 \* 1000;

function File() {

}

File.FULL\_PERMISSIONS = 0777;

1. 比较操作 SS or M

有的场景下应该用 "===" 替代 "==" 当你遇到这些符号比较时 ：0 undefined null false true

你应该小心谨慎 比如　' \t\r\n' == 0　比较结果是 true

Right：

var a = 0;

if (a === '') {

console.log('winning');

}

Wrong：

var a = 0;

if (a == '') {

console.log('losing');

}

1. 使用字面表达式，用 '{}' ,'[]' 代替 new Array ，new Object M

这一条是因为{} , [] 可以提高效率

1. 不要使用 string，bool，number 的对象类型，即不要调用 new String ，new Boolean ，new Number。 SS
2. Object ，Array 创建，当有多个元素时，注意分行排列时逗号的位置。 SS

Right：

var a = ['hello', 'world'];

var b = {

good: 'code',

'is generally': 'pretty',

};

Wrong:

var a = [

'hello', 'world'

];

var b = {"good": 'code'

, is generally: 'pretty'

};

1. 避免使用 “with” 与 “eval” M
2. for-in 循环，仅在 object/hash/map 时使用，绝不要对Array 使用 SS

不要把Array 当做关联数组或Object 使用,即你不应该用非数字作为Array 的索引 (Phper 尤其注意这点)

Wrong :

var a = []; // use '{}' instead

a['hello'] = 'shit';

a['foo'] = 'bar';

Right:

var a = {};

a.hello = 'shit';

a.foo = 'bar';

1. Node 的异步回调函数的第一个参数应该是错误指示。 M

function cb(err, data , ...) {...}

这一条必须遵守，是通行的规则，所以得异步回调函数，第一个参数肯定是err

1. 类继承写法，尽管有各种方式来实现继承，但最为推荐的是Node 的标准写法。 SS

function Socket(options) {

// ...

stream.Stream.call(this);

// ...

}

util.inherits(Socket, stream.Stream);

1. 如果你在模块中 exports 一个类，对于此类的私有成员变量，建议加上 "\_" 前缀以示区分。 SS
2. 变量声明时，应该每行声明一个，不应该都写在一行（尽管这被JSLint 所推荐）。SS

Right：

var assert = require('assert');

var fork = require('child\_process').fork;

var net = require('net');

var EventEmitter = require('events').EventEmitter;

Wrong：( Node 源代码已经将此方式全部修正)

var assert = require('assert')

, fork = require('child\_process').fork

, net = require('net')

, EventEmitter = require('events').EventEmitter;

1. 注释规范，采用 [Google 的js 规范](http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml#Comments)。 SS

Right :

/\*\*

\* Queries a Baz for items.

\* @param {number} groupNum Subgroup id to query.

\* @param {string|number|null} term An itemName,

\* or itemId, or null to search everything.

\*/

goog.Baz.prototype.query = function (groupNum, term) {

// ...

};

# 4 JS注释规范

我们选用JS Doc作为JavaScript文档生成工具，那么自然要求我们代码中使用符合JS doc要求的注释方式。

JSDoc是一个根据javascript文件中注释信息，生成JavaScript应用程序或库、模块的API文档 的工具。你可以使用他记录如：命名空间，类，方法，方法参数等。类似JavaDoc和PHPDoc。现在很多编辑器或IDE中还可以通过JSDoc直接或使用插件生成智能提示。从而使开发者很容易了解整个类和其中的属性和方法，并且快速知道如何使用，从而提高开发效率，降低维护成本。

我们对JSdoc的注释要求分为两层，文件层面和函数层面。

<http://usejsdoc.org/>

## 4.1 File Example

/\*\*

 \* @file File description

 \* @author zhouyaowei@dmtec.cn

\* @version 1.2.3

 \*/

## 4.2 Function Example

/\*\*  
     \* Get the red, green, and blue values of a color.  
     \* @function  
     \* @param {string} color - A color, in hexidecimal format.  
     \* @returns {Array.} An array of the red, green, and blue values,  
     \* each ranging from 0 to 255.  
     \*/

# 参考资料

<http://docs.kissyui.com/1.4/docs/html/tutorials/style-guide/google-js-style.html>

<http://docs.kissyui.com/1.4/docs/html/tutorials/style-guide/js-lang-rules.html>

<http://docs.kissyui.com/1.4/docs/html/tutorials/style-guide/js-style-rules.html>

<http://javascript.crockford.com/code.html>

<http://www.ruanyifeng.com/blog/2012/04/javascript_programming_style.html>

<https://github.com/airbnb/javascript>

<https://github.com/windyrobin/iFrame/blob/master/style.md>

<http://usejsdoc.org/index.html>