
JavaScript编码风格指南

> 引用于《编写可维护的JavaScript》附录A

1、缩进

1 缩进

每一行的层级由4个空格组成，避免使用制表符缩进。

2、行的长度

每行长度不超过80个字符，如果超过，应当在一个运算符（逗号，加号等）后换行。下一行应当增加两级缩进（8个空格）。

```
//好的写法
doSomething(arguments1, arguments2, arguments3, arguments4,
    arguments5)
```

3、运算符的行间距

二元运算符前后必须使用一个空格来保持表达式的整洁。操作符包括赋值运算符和逻辑运算符。

```
var found = (values[0] === item);

if(found && (count > 10));

for(var i = 0; i < count; i++)
```

4、括号间距

当使用括号时，紧接左括号之后和紧接右括号之前不应该有空格。

5、对象直接量

对象直接量应当使用如下格式：

- 起始左花括号应当同表达式保持一行
- 每个属性的名值对应当保持一个缩进，第一个属性应当在左花括号后另起一行
- 每个属性的名值对应当使用不含引号的属性名，其后紧跟一个冒号（之前不含空格），而后是值
- 倘若属性值是函数类型，函数体应当在属性名之下另起一行，而且其前后就均保留一个空格
- 一组相关的属性前后可以插入空行以提升代码的可读性
- 结束的右花括号应当独占一行

```
var obj = {

  key1: value1,

    key2: value2,

    func: function() {

    },

    key3: value3
};
```

当对象字面量作为函数参数时，如果值是变量，其实花括号应当同函数名在同一行，所有其实先前列出的规则同样适用

```
//好的写法
doSomething({
  key1: value1,
    key2: value2
})
//不好的写法
doSomething({key1: value1,key2: value2})
```

6、注释

频繁地使用注释有助于他人理解你的代码，如下情况应当使用注释：

- 代码晦涩难懂
- 可能被误认为错误的代码
- 必要但并不明显的针对特定浏览器的代码
- 对于对象、方法或者属性，生成文档是有必要的（使用恰当的文档注释）

6.1 单行注释

单行注释应当用来说明一行或者一组相关的代码。单行注释可能有如下三种使用方式：

- 独占一行的注释，用来解释下一行代码
- 在代码行的尾部注释，用来解释它之前的代码
- 多行，用来注释掉一个代码块

```
//好的写法
if(condition){

  //如果代码执行到这里，则表明通过了所有的安全检查
  allowed()
}

//不好的写法:注释前没有空行
if(condition){
  //如果代码执行到这里，则表明通过了所有的安全检查
  allowed()
}
//不好的写法:错误的缩进
if(condition){

//如果代码执行到这里，则表明通过了所有的安全检查
  allowed()
}
```

对于行尾的情况，代码和注释间至少一个空格

```
//好的写法
var a = 10; // 这里是注释
//不好的写法
var a = 10;// 这里是注释
```

6.2 多行注释

多行注释应当在代码需要更多的文字去解释的时候使用。每个多行注释应当有如下三行

1. 首行仅仅包含/*注释开始，该行不能有其他文件
2. 接下来的行*开头并保持左对齐。这些行可以有文字描述
3. 最后一行以*/结束并与先前行对齐。也不应再有其他文字

多行注释应当保持同它描述的代码一样的缩进，后续每个*后加一个空格

```
//好的写法
if (condition) {

 /*
 * 这里是说明
 * 这也是
 * 打字真他妈累
 * 不好好看，我掐死你们
 */
   allowed();
}
```

7、变量声明

所有变量在使用前都应当事先定义，使用一个var。

```
//好的写法
var a = 1,
 b = 2,
 c = 3,
 d = 4;
//不好的写法
var a = 1;
var b = 2;
var c = 3;
var d = 4;
//不好的写法
var a = 1,
 b = 2,
 c = 3,
 d = 4;
```

未初始化值得变量在后面

```
//好的写法
var a = 1,
  b,
  c;
// 不好的写法
var b,
  c,
  a = 1;
```

8 、函数声明

函数应当在使用前提前定义，不是作为对象的方法的函数，应当使用函数声明的格式（function声明）

```
// 好的写法
function fn(arg1, arg2) {

}
// 不好的写法：不恰当的空格
function fn (arg1, arg2){

}
// 不好的写法：花括号位置不对
function fn(arg1, arg2)
{

}
```

函数内的函数声明，应当在var声明后，立即定义。

```
//好的写法
function outer() {

  var count = 10,
    name = 'fengyu',
    age = 18,
    empty;

  function inner() {

  }
}
```

匿名函数的自执行

```
// 推荐使用
(function (){}());

// 其他的都不推荐。
```

9、命名

变量和函数命名时要小心。命名应仅限于数字字母字符。某些情况可以使用下划线。最好不好用美元符号('$')和反斜杠('\')

变量命名采用驼峰式，首字母小写，每个单词首字母大写，变量名的第一个单词应当是一个名词（而非动词）以避免和函数混淆，不要再变量名中使用下划线。

```
// 好的写法
var accountNumber = "1234567";
// 不好的写法：大写字母开头
var AccountNumber = "1234567";
// 不好的写法：动词开头
var getAccountNumber = "1234567";
//不好的写法：使用下划线
var account_Number = "1234567";
```

函数名也使用驼峰，第一个单词应当是动词（而非名词），也不要使用下划线

```
// 好的写法
function doSomething() {

}
// 不好的写法:首字母大写
function DoSomething() {

}
// 不好的写法:名词
function car() {

}
// 不好的写法：下划线
function do_something() {

}
```

构造函数才能首字母大写，名称应当非动词开头

```
// 好的写法
function MyObject() {

}
// 不好的写法：首字母小写
function myObject() {

}
// 不好的写法：动词开头
function getMyObject() {

}
```

常量（值不会被改变的变量）命名应当是全部大写，不同单词用下划线隔开

```
// 好的写法
var TOTAL_COUNT = 10;
// 其他都是不好的写法
```

对象的属性同变量的命名规则相同。对象的方法同函数的命名规则相同，如果属性或者方法是私有（不希望别人访问），应当在之前加一个下划线

```
// 好的写法
var object = {
  _count: 10,

  _getCount: function (){
    return this._count;
    }
}
```

10、严格模式

严格模式应当仅限在函数内部使用，千万不要在全局使用。

```
// 不好的写法：全局使用严格模式
"use strict";

function doSomething() {
  //code
};

// 好的写法
function doSomething() {
    "use strict";

    //code
};
```

如果你期望在多个函数里使用严格模式而不需要多次声明“use strict”，可以使用立即执行的函数。

```
(function (){
   "use strict";

  function doSomething() {

  };

  function doSomethingElse() {

  };
}());
```

11、赋值

当给变量赋值时，如果右侧时含有比较语句的表达式，需要使用圆括号包裹。

```
// 好的写法
var flag = (i < count);

//不好的写法
var flag = i < count;
```

12、等号运算符

使用===（严格等于）和！==（严格不相等）代替==（相等）和!=（不等）来避免弱类型转换错误。

```
// 好的写法
var same = (a === b);

//不好的写法
var same = (a == b);
```

13、三元运算符

三元运算符应当仅仅用在条件赋值语句中，而不要作为if的替代品。

```
// 好的写法
var value = condition ? value1 : valule2;

// 不好的写法
condition ? doSomething() : doSomethingElse();
```

14、语句

14.1 简单语句

每行最多只包含一条语句。所有简单的语句都应该以分号结束

```
// 好的写法
count++;
a = b;

// 不好的写法
count++; a = b;
```

14.2 返回语句

返回语句当返回一个值得时候不应当使用圆括号包裹，除非在某些情况下这么做可以让返回值更容易理解，例如：

```
return;
return fengyu.length();
return (length > 18 ? length : 18);
```

14.3 复合语句

复合语句时大括号括起来得语句列表。

- 括起来得语句应当较符合语句多缩进一个层级
- 开始得大括号应当在符合语句所在行的末尾；结束的大括号应当独占一行且同复合语句的开始保持一样的缩进
- 当语句是控制结构的一部分时，诸如if或者for语句，所有的语句都需要用大括号括起来，也包括单个语句。这个约定是的我们更加方便地添加语句而不用担心忘记加括号而引起bug。
- 像if一样的语句的开始的关键词，其后应该紧跟一个空格，起始大括号应当在空格之后

14.3.1 if语句

if语句应当是下面的格式

```
if (condition) {
    statements
}

if (condition) {
    statements1
} else {
    statements2
}

if (condition1) {
    statements1
} else if (condition2) {
    statements2
} else {
    statements3
```

```
}
```

绝不允许在if中省略花括号

```
// 好的写法
if (condition) {
    doSomething();
}
//不好的写法：不恰当的空格
if(condition){
    doSomething();
}
// 不好的写法：省略花括号
if(condition)
    doSomething();
// 不好的写法：代码在一行
if (condition) {doSomething();}
// 不好的写法：在一行还省略花括号
if (condition) doSomething();
```

14.3.2 for语句

for类型的语句应当是下面的格式

```
for (initialization; condition; Step) {
    statements
}

for (variable in object) {
    statements
}
```

for语句的初始化部分不应当有变量声明

```
// 好的写法
var i,
  len;
for (i = 0, len = 10; i < len; i++) {
    //code
}

// 不好的写法
for (var i = 0, len = 10; i < len; i++) {
    //code
}
for (var prop in object) {
    //code
}
```

14.3.3 while语句

while类的语句应当是下面格式

```
while (condition) {
  //code
}
```

14.3.4 do...while语句

do类语句应当是下面格式

```
do {
  //code
} while (condition);
```

14.3.5 switch语句

格式如下：

```
#除了第一个case之外，包括default在内的每一个case之前都应当有一个空行
switch (expression) {
    case exp1:
      //code
      break;

    case exp2:
      //code
      break;

    default:
      //code
      break;
}
```

如果一个switch不包含default的情况，应当用注释代替

```
switch (expression) {
    case exp1:
      //code
      break;

    case exp2:
      //code
      break;

    // 没有default
}
```

15、留白

在逻辑相关的代码块之间添加空行可以提高代码的可读性。

两行空行仅限在如下情况中使用

- 在不同的源代码文件之间
- 在类和接口定义之间

单行空行仅限在如下情况中使用

- 方法之间
- 方法中局部变量和第一行语句中间
- 多行或者单行注释之前
- 方法中逻辑代码块之间以提升代码的可读性

空格应当在如下情况中使用

- 关键词后跟括号的情况应当用空格隔开
- 参数列表中逗号之后应当留一个空格
- 所有的除了点（.）之外的二元运算符，其操作数都应当用空格隔开。单目运算符的操作数事件不应该用空白隔开，诸如一元减号，递增(++)，递减(--)
- for语句中的表达式之间应当用空格隔开

16、需要避免的

- 切勿使用像String一类的原始包装类型创建新的对象。
- 避免使用eval()
- 避免使用with语句（其实该语句已经被废除了，所以老夫就不教你们了）

关于命名

变量：必须采用小驼峰式命名法。

命名规范：前缀应当是名词。(函数的名字前缀为动词，以此区分变量和函数)

命名建议：尽量在变量名字中体现所属类型，如:length、count等表示数字类型；而包含name、title表示为字符串类型。

```
// 好的命名方式
let maxCount = 10;
let tableTitle = 'LoginTable';
// 不好的命名方式
let setCount = 10;
let getTitle = 'LoginTable';
```

1. 常量：必须采用**全大写**的命名，且**单词以_分割**，常量通常用于ajax请求url，和一些不会改变的数据

命名规范：使用大写字母和下划线来组合命名，下划线用以分割单词。

```
const MAX_COUNT = 10;
const URL = 'http://www.foreverz.com';
```

1. 函数

- 命名方法：小驼峰式命名法。
- 命名规范：前缀应当为动词。
- 命名建议：可使用常见动词约定

动词	含义	返回值
can	判断是否可执行某个动作(权限)	函数返回一个布尔值。true：可执行；false：不可执行
has	判断是否含有某个值	函数返回一个布尔值。true：含有此值；false：不含有此值
is	判断是否为某个值	函数返回一个布尔值。true：为某个值；false：不为某个值
get	获取某个值	函数返回一个非布尔值
set	设置某个值	无返回值、返回是否设置成功或者返回链式对象
load	加载某些数据	无返回值或者返回是否加载完成的结果

```
// 是否可阅读
function canRead(): boolean {
  return true;
}
// 获取名称
function getName(): string {
  return this.name;
}
```

1. 类 & 构造函数

命名方法：大驼峰式命名法，首字母大写。

命名规范：前缀为名称。

示例：

```
class Person {
  public name: string;
  constructor(name) {
    this.name = name;
  }
}
const person = new Person('mevyn');
```

1. 类的成员

类的成员包含：

- 公共属性和方法：跟变量和函数的命名一样。
- 私有属性和方法：前缀为_(下划线)，后面跟公共属性和方法一样的命名方式。

示例：

```
class Person {
  private _name: string;
  constructor() { }
  // 公共方法
  getName() {
    return this._name;
  }
  // 公共方法
  setName(name) {
    this._name = name;
  }
}
const person = new Person();
person.setName('mervyn');
person.getName(); // ->mervyn
```

4. 注释规范

js 支持三种不同类型的注释：行内注释、单行注释和多行注释：

1. 行内注释

- 说明：行内注释以两个斜线开始，以行尾结束。
- 语法：code // 这是行内注释
- 使用方式：//(双斜线)与代码之间保留一个空格，并且//(双斜线)与注释文字之间保留一个空格。

命名建议：

// 用来显示一个解释的评论
// -> 用来显示表达式的结果，
// >用来显示 console 的输出结果，

示例：

```
function test() { // 测试函数
  console.log('Hello World!'); // >Hello World!
  return 3 + 2; // ->5
}
```

1. 单行注释

- 说明：单行注释以两个斜线开始，以行尾结束。
- 语法：// 这是单行注释
- 使用方式：单独一行：//(双斜线)与注释文字之间保留一个空格。

示例：

```
// 调用了一个函数；1)单独在一行
setTitle();
```

1. 多行注释

- 说明：以 /* 开头， */ 结尾
- 语法：/* 注释说明 */
- 使用方法：若开始/*和结束*/都在一行，推荐采用单行注释。若至少三行注释时，第一行为/*，*最后行为*/，其他行以*开始，并且注释文字与*保留一个空格。

示例:

```
/*
* 代码执行到这里后会调用setTitle()函数
* setTitle()：设置title的值
*/
setTitle();
```

1. 函数(方法)注释

- 说明：函数(方法)注释也是多行注释的一种，但是包含了特殊的注释要求，参照JSDoc
- 语法：

```
/**
* 函数说明
* @关键字
*/
```

常用注释关键字：(只列出一部分，并不是全部)

注释名	语法	含义	示例
@param	@param 参数名 {参数类型} 描述信息	描述参数的信息	@param name {String} 传入名称
@return	@return {返回类型} 描述信息	描述返回值的信息	@return {Boolean} true:可执行;false:不可执行
@author	@author 作者信息 [附属信息：如邮箱、日期]	描述此函数作者的信息	@author 张三 2015/07/21
@version	@version XX.XX.XX	描述此函数的版本号	@version 1.0.3
@example	@example 示例代码	演示函数的使用	@example setTitle('测试')

```
/**
```

```
* 合并Grid的行
* @param grid {Ext.Grid.Panel} 需要合并的Grid
* @param cols {Array} 需要合并列的Index(序号)数组；从0开始计数，序号也包含。
* @param isAllSome {Boolean}： 是否2个tr的cols必须完成一样才能进行合并。true：完成一样；false(默认)：不完全一样
* @return void
* @author polk6 2015/07/21
* @example
* _____________                    _____________
* | 年龄 | 姓名 |                   | 年龄 | 姓名 |
* ----------------    mergeCells(grid,[0])  ----------------
* | 18  | 张三 |          =>        |     | 张三 |
* ----------------                  - 18  ---------
* | 18  | 王五 |                    |     | 王五 |
* ----------------                  ----------------
*/
function mergeCells(grid: Ext.Grid.Panel, cols: Number[], isAllSome: boolean = false) {
  // Do Something
}
```

- 不要保存 this 的引用。使用 Function#bind。

```
// bad
function () {
  var self = this;
  return function () {
    console.log(self);
  };
}

// bad
function () {
  var that = this;
  return function () {
    console.log(that);
  };
}

// bad
function () {
  var _this = this;
  return function () {
    console.log(_this);
  };
}

// good
function () {
  return function () {
    console.log(this);
  }.bind(this);

}
```

- 给函数命名。这在做堆栈轨迹时很有帮助。

```
// bad
var log = function (msg) {
  console.log(msg);
};

// good
var log = function log(msg) {
  console.log(msg);
};
```

- 如果你的文件导出一个类，你的文件名应该与类名完全相同。

```
// file contents
class CheckBox {
  // ...
}
module.exports = CheckBox;

// in some other file
// bad
var CheckBox = require('./checkBox');

// bad
var CheckBox = require('./check_box');

// good
var CheckBox = require('./CheckBox');
```

5.模块

- 模块应该以 ! 开始。这样确保了当一个不好的模块忘记包含最后的分号时，在合并代码到生产环境后不会产生错误。详细说明
- 文件应该以驼峰式命名，并放在同名的文件夹里，且与导出的名字一致。
- 增加一个名为 noConflict() 的方法来设置导出的模块为前一个版本并返回它。
- 永远在模块顶部声明 'use strict';。

```
// fancyInput/fancyInput.js

!function (global) {
  'use strict';

  var previousFancyInput = global.FancyInput;

  function FancyInput(options) {
    this.options = options || {};

  }
```

```
  FancyInput.noConflict = function noConflict() {
    global.FancyInput = previousFancyInput;
    return FancyInput;
  };

  global.FancyInput = FancyInput;
}(this);
```

构造函数

- 给对象原型分配方法，而不是使用一个新对象覆盖原型。覆盖原型将导致继承出现问题：重设原型将覆盖原有原型！

```
function Jedi() {
  console.log('new jedi');
}

// bad
Jedi.prototype = {
  fight: function fight() {
    console.log('fighting');
  },

  block: function block() {
    console.log('blocking');
  }
};

// good
Jedi.prototype.fight = function fight() {
  console.log('fighting');
};

Jedi.prototype.block = function block() {
  console.log('blocking');
};
```

- 方法可以返回 this 来实现方法链式使用。

```
// bad
Jedi.prototype.jump = function jump() {
  this.jumping = true;
  return true;
};

Jedi.prototype.setHeight = function setHeight(height) {
  this.height = height;
};
```

```
var luke = new Jedi();
luke.jump(); // => true
luke.setHeight(20); // => undefined

// good
Jedi.prototype.jump = function jump() {
  this.jumping = true;
  return this;
};

Jedi.prototype.setHeight = function setHeight(height) {
  this.height = height;
  return this;
};

var luke = new Jedi();

luke.jump()
  .setHeight(20);
```

写在最后：

风格这种东西，有很多，就拿缩进来说：

- jQuery核心风格指南明确规定使用制表符缩进
- Dauglas Crockford的JavaScript代码规范规定使用4个空格字符缩进
- SproutCore风格指南规定使用2个空格缩进
- Google的Javascript风格指南规定使用2个空格缩进
- Dojo编程风格指南规定使用制表符缩进

所以啊，风格很多，大概的了解一下，一切以你所在的团队风格为准。