JDC 前端 JavaScript 代码规范

本规范的目标是使 JavaScript 代码风格保持一致，有助于写出质量高、错误少、易于维护及团队协作的程序。

标准委员会（fe-codestyle@jd.com）参与规则制定人员：

- 罗均波 （luojunbo@jd.com）
- 刘威 （liuwei1@jd.com）
- 陈晓春 （chenxiaochun@jd.com）
- 巫燿恒 （wuyaoheng@jd.com）
- 王少星 （wangshaoxing@jd.com）
- 冯伟平 （fengweiping@jd.com）
- 佟恩 （tongen@jd.com）
- 李春来 （lichunlai@jd.com）
- 石建国 （bjshijianguo@jd.com）
- 王戈 （wangge1@jd.com）
- 周涛 （yfzhoutao@jd.com）

内容列表

文件

- JavaScript 文件使用无 BOM 的 UTF-8 编码。

> 为什么? UTF-8 编码具有更广泛的适应性。BOM 在使用程序或工具处理文件时可能造成不必要的干扰，比如Mac系统中的换行符问题。

缩进层次

- 缩进层级：使用 4 个空格， 不允许使用 2 个空格或 tab。

> switch 下的 case 和 default 必须增加一个缩进层级。

```
// bad
function fn() {
··var name;
}
// good
function fn() {
····var name;
}
```

```
// bad
switch (vs) {
case '1':
    // do...
    break;
case '2':
    // do...
    break;
default:
    // do...
```

```
}

// good
switch (vs) {
    case '1':
        // do...
        break;
    case '2':
        // do...
        break;
    default:
        // do...
}
```

空格

- 关键字 `if`、 `else`、 `switch`、 `case`、 `for`、 `while` 和 `:` 后加 1 个空格。

```
// bad
if(a > b) {

}
// good
if (a > b) {

}

// bad
for(var i = 0; i < 5; i++) {
    // ...
}
// good
for (var i = 0; i < 5; i++) {
    // ...
}
```

- 赋值运算符 `=` 两边各加 1 个空格。

```
// bad
name='John';

// good
name = 'John';
```

- 两元运算符（`==`, `>`, `<`等）两边各加 1 个空格。

```
// bad
if (a==b) {

}

// good
if (a == b) {

}
```

- 函数参数之间加 1 个空格。

```
// bad
function ajax(url,param,success) {
    // ...
}

// good
function ajax(url, param, success) {
    // ...
}
```

- 一元运算符（`++`, `--`）前后不加空格。

```
// bad
```

```
++ sum;
total --;

// good
++sum;
total--;
```

- 左大括号前加 1 个空格。

```
// bad
function ajax(url, param, success){
    // ...
}
if (a > b){

}

// good
function ajax(url, param, success) {
    // ...
}
if (a > b) {

}
```

- 数组成员间 `,` 后加 1 个空格。

```
// bad
var arr = [1,2,3,4];

// good
var arr = [1, 2, 3, 4];
```

- 小括号作为函数调用时 `函数名` 与 `左小括号` 之间不加空格。

```
// bad
func ();

// good
func();
```

- 小括号作为强制运算符时左括号前加 1 个空格。

```
// bad
return(x + y);

// good
return (x + y);
```

- 起首的大括号跟在关键字的后面，且大括号前留有 1 个空格。

```
// bad
if (a > b)
{
    // ...
}
// good
if (a > b) {
    // ...
}

// bad
dog.set('attr',{
    age: '1 year',
    breed: 'Bernese Mountain Dog'
});
// good
dog.set('attr', {
    age: '1 year',
    breed: 'Bernese Mountain Dog'
```

```
});
```

换行

- 每个独立语句结束后必须换行。

```
// bad
user.setCode(1); user.setName('admin'); user.setPwd('123456');

// good
user.setCode(1);
user.setName('admin');
user.setPwd('123456');
```

- 行的长度不应超过80个字符，超过后需要手动将一行拆成两行。下一行增加两级缩进（8个空格）。

```
// bad
doSomething(argument1, argument2, argument3, argument4, argument5);
doSomething(argument1, argument2, argument3, argument4,
    argument5);

// good
doSomething(argument1, argument2, argument3, argument4,
        argument5);
```

- 空行，函数之间、函数的局部变量和第一条语句之间加空行。

```
// bad
function fetch() {
}
function getUser {
}

// good
function fetch() {
}

function getUser {
}
```

> [建议] 把不同业务逻辑的语句使用空行隔开，更易阅读。

语句

- 语句末尾始终使用分号，不要依赖于引擎隐式插入（Automatic Semicolon Insertion，ASI）。

> `function`、`for`、`if`、`switch`、`try`、`while` 语句块末尾不需要加分号。

```
// bad
function fn() {
    // ...
};
// good
function fn() {
    // ...
}

// bad
for (var i = 0; i < 5; i++) {
    // ...
};
// good
for (var i = 0; i < 5; i++) {
    // ...
}
```

- 花括号对齐方式，将花括号放置在第一句代码的末尾。

```
// bad
if (boo)
{
    // do...
}
else
{
    // do...
}

// good
if (boo) {
    // do...
} else {
    // do...
}
```

- 所有的语句块，即使只有一行，也不得省略块 {...}。

```
// bad
if (condition) callFunc();
if (condition)
    callFunc();

// good
if (condition) {
    callFunc();
}
```

变量

- 始终使用 `var` 来声明变量，否则会产生全局变量，避免污染全局命名空间。

```
// bad
superPower = new SuperPower();

// good
var superPower = new SuperPower();
```

- 命名采用驼峰式，第一个单词小写，之后的单词首字母大写。

```
// bad
var nickname = 'John';
var nick_name = 'John';

// good
var nickName = 'John';
```

- 常量名全部用大写字母，多个单词使用下划线 `_` 连接。

```
// bad
var pi = 3.1415926;
// good
var PI = 3.1415926;

// bad
var maxCount = 100;
// good
var MAX_COUNT = 100;
```

> 常量：指数学(圆周率/自然对数)，物理领域（光速/质子磁矩/元电荷），计算机领域（HTTP状态码/URL），这些变量声明后不再改变。

- 类（构造器）名首字母大写，用名词。

```
// bad
function person(name, age) {
    this.name = name;
    this.age = age;
```

```
}

// good
function Person(name, age) {
    this.name = name;
    this.age = age;
}
```

- 避免单个字符名，变量名应具有描述意义（计数器 `i,j,k` 除外）。

```
// bad
function q() {
    // ...stuff...
}

// good
function query() {
    // ..stuff..
}
```

- 声明且赋值时使用多个 `var` 及换行。

```
// bad
var items = getItems(),
    goSportsTeam = true,
    dragonball = 'z';

// good
var items = getItems();
var goSportsTeam = true;
var dragonball = 'z';
```

- 只声明不赋值时可以使用单个 `var` 。

```
// bad
var top;
var left;
var width;
var height;

// good
var top, left, width, height;
```

- 分组变量声明，变量多且有明显相关性时。

```
// bad
var x, y, z, width, height;

// good
var x, y, z;
var width, height;
```

- 就近声明，用时声明，更自然的在视角范围内。

> 为什么？ 当函数行数较多时，如果把变量声明全部放在顶部，
> 阅读或修改时还需要滚动到顶部，甚为不便。

```
// bad
var i, len;
var goSportsTeam = true;
var dragonball = 'z';

// ...

for (i = 0; i < len; i++) {
    // ...
}

// good
var goSportsTeam = true;
```

```
var dragonball = 'z';

// ...

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

- 声明时变量之间存在依赖关系时，最后声明未赋值的变量，当你想引用之前已赋值变量的时候很有用。

```
// bad
var width;
var height;
var $div = $('#nav');

// good
var $div = $('#nav');
var width = $div.width();
var height = $div.height();
```

- 变量应先声明后使用，避免 `变量提升` 现象产生。

```
// bad
function fn() {
    doStaff(width, height);
    var width = $div.width() || 0;
    var height = $div.height() || 0;
}

// good
function fn() {
    var width = $div.width() || 0;
    var height = $div.height() || 0;
    doStaff(width, height);
}
```

- 使用前缀，不采用后缀。常见前缀参考

前缀	类型	示例
is, can, has	Boolean	isPass
reg	RegExp	regNum
get	Getter	getName
set	Setter	setName
i, j, k	iterator	
el	HTMLElement	elNav
$	jQuery Object	$nav

返回列表

函数

- 函数的长度控制在 50 行以内。

> 为什么？将过多的逻辑单元混在一个大函数中，难以维护。一个清晰易懂的函数应该完成单一的逻辑单元。复杂的操作应进一步分解，通过函数的调用来体现流程。
>
> 不可分割的特定算法逻辑允许例外。

- 函数的参数控制在 5 个以内，过多参数会导致维护难度增大。

> 为什么？ 有些函数的参数并非作为算法的输入，而是对算法的

> 某些分支条件判断之用，此类参数建议通过一个JS对象 options 传递。
>
> 某些情况下，如使用 AMD/CMD Loader 的 require 加载多个模块时，其 callback 可能会存在较多参数，因此对函数参数的个数不做强制限制。

返回列表

注释

- 在单行注释符后留一个空格。

```
// bad
//this is comment

// good
// this is comment
```

- 在多行注释的结束符前留一个空格，使星号对齐。

```
// bad
/*
 * this is comment
*/

// good
/*
 * this is comment
 */
```

- 文档或API使用双星号 `/**` 开始，用 `@` 符号表示属性（方法名/参数/返回值等）

> @ 的常见属性： class, constructor, method, param, return 等

```
/**
 * @method merge
 * @param  {Object} 被合并的对象，一个或多个
 * @return {Object} 返回新的合并后的对象
 *
 */
Y.merge = function() {
    // ...
    return dest;
}
```

返回列表

其它

- 定义对象时将逗号放后面。

```
// bad
var hero = {
    firstName: 'Bob'
  , lastName: 'Parr'
  , heroName: 'Mr. Incredible'
  , superPower: 'strength'
};

// good
var hero = {
  firstName: 'Bob',
  lastName: 'Parr',
  heroName: 'Mr. Incredible',
  superPower: 'strength'
};
```

- 使用单引号定义字符串。

```
// bad
var name = "Capt. Janeway";

// good
var name = 'Capt. Janeway';
```

- 超长字符串采用加号 `+`， 多行显示，换行后加两级缩进。

```
// bad
var errorMessage = 'This is a super error that was thrown because of
Batman. When you stop how Batman had anything to do with this, you
would get nowhere fast.';

// bad
var errorMessage = 'This is a super error that was thrown because \
of Batman. When you stop how Batman had anything to do \
with this, you would get nowhere fast.';

// good
var errorMessage = 'This is a super error that was thrown because '
        + 'of Batman. When you stop how Batman had anything to do '
        + 'with this, you would get nowhere fast.';
```

- HTML属性使用双引号，即单引号在外层，双引号在内层。

```
// bad
var html = "<a href='http://www.jd.com'>京东</a>";
// good
var html = '<a href="http://www.jd.com">京东</a>';
```

- 大量 HTML 的拼接使用前置加号

```
 var theadHTML = ''
        +'<thead>'
        +   '<tr class="controls"><th colspan="7">'
        +       '<span class="selectMonth">'
        +           '<select class="month"></select>'
        +       '</span>'
        +       '<span class="slecteYear">'
        +           '<select class="year"></select>'
        +       '</span>'
        +   '</th></tr>'
        +   '<tr class="days">'
        +       '<th class="J-sun">日</th>'
        +       '<th class="J-mon">一</th>'
        +       '<th class="J-tue">二</th>'
        +       '<th class="J-wed">三</th>'
        +       '<th class="J-thu">四</th>'
        +       '<th class="J-fri">五</th>'
        +       '<th class="J-sat">六</th>'
        +   '</tr>'
        +'</thead>'
```

- 需要引用闭包外层的上下文 `this` 时，使用标识符 `me`、`self`、 `that`， 不使用 `_this`。

```
// bad
function outer() {
    var _this = this;
    function inner() {
        _this.search();
    }
}

// good
function outer() {
    var me = this;
    function inner() {
        me.search();
```

```
            me.search();
        }
    }
```

返回列表