JavaScript 编码规范

> 对当前规范有更好的建议，请到 issue (http://gitlab.alibaba-inc.com/specs/style-guide/issues) 页面反馈给我们，谢谢！

基本规范

缩进

统一使用两空格缩进

理由：结合 80 字符的限制 + javascript 的异步嵌套，如果使用四个空格需要面临不断换行的问题

引号

统一使用单引号

编码

统一使用 utf-8 编码

单行字符长度限制

单行代码字符长度尽量不超过 80 个字符

理由：代码更加清晰，便于阅读

严格模式

推荐总是在 js 文件头部声明使用严格模式，这样可以使你的代码更加健壮，参考 MDN 严格模式 (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode)

```
'use strict'

// ...
```

建议使用字面量来代替构造函数

```
// bad
var arr = new Array(1, 2, 3);
var obj = new Object();

// good
var arr = [1, 2, 3];
var obj = {};
```

代码规范

对象

- 不要使用保留字 reserved words (https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Reserved_Words) 作为对象属性，IE8 下不支持。参考 (https://github.com/airbnb/javascript/issues/61)

```
// bad
var superman = {
  class: 'superhero'
};

// good
var superman = {
  className: 'superhero'
};
```

数组

- 使用 Array#push 为数组添加元素

```
var someStack = [];

// bad
someStack[someStack.length] = 'test';

// good
someStack.push('test');
```

字符串

- 超过80个字符的字符串应该使用字符串连接换行

```
// bad
var errorMessage = 'This is a super long error that

// good
var errorMessage = 'This is a super long error that
  'was thrown because of Batman.' +
  'fast.';
```

函数

- 不要在非函数块（if, while 等）里声明函数，如有需要，可以把函数赋值给一个变量。

```
// bad
if (currentUser) {

  function test() {
    console.log('Nope.');
  }

  test();
}

// good
if (currentUser) {

  var test = function() {
    console.log('Yup.');
  };

  test();
}
```

- 绝对不要把参数命名为 `arguments`，这将会覆盖 `arguments` 对象.

```
// bad
function nope(name, options, arguments) {
  // ...stuff...
}

// good
function yup(name, options, args) {
  // ...stuff...
}
```

属性

- 访问对象的属性时，尽可能使用 `.`

```
var superman = {
  age: 12
};

// bad
var age = superman['age'];

// good
var age = superman.age;
```

变量

- 总是使用 var 来声明变量，如果不这么做将导致产生全局变量，我们要避免污染全局命名空间。

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

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

- 推荐使用多个 var 声明多个变量。理由：方便增删；多 var 可以更好的帮助养成良好的习惯，而不发生遗漏 var 而导致全局变量的风险。

```
// bad
var first = 1,
  second = 2,
  third = 3;

// good
var first = 1;
var second = 2;
var third = 3;
```

条件表达式和等号

- 尽量使用 === 和 !==
- 条件表达式的强制类型转换遵循以下规则：

    - **对象** 被计算为 **true**
    - **Undefined** 被计算为 **false**
    - **Null** 被计算为 **false**
    - **布尔值** 被计算为 **布尔的值**
    - **数字** 如果是 **+0, -0, or NaN** 被计算为 **false** , 否则为 **true**
    - **字符串** 如果是空字符串 `''` 则被计算为 **false**, 否则为 **true**
    - **0**, 值为 0 时，要特别注意

- 更多信息请阅读 Truth Equality and JavaScript (http://javascriptweblog.wordpress.com/2011/02/07/truth-equality-and-javascript/#more-2108).

块

- 不要省略花括号

```
// bad
if (test)
  return false;

// bad
if (test) return false;

// good
if (test) {
  return false;
}
```

- 花括号后应该换行

```
// bad
if (test) { return false; }

// good
function() {
  return false;
}
```

空白

- 操作符之间需要有空格

```
// bad
var a=1+2;

// good
var a = 1 + 2;
```

- 函数空格

```
// bad
function add(a,b) {      // 多个参数时逗号后要加空格
}
function add(a, b){      // 花括号前需要加空格
}
function add( a, b ) {   // 参数两边需要加空格
}
function add (a, b) {    // 函数名和参数括号前不需要加空
}

// good
function add(a, b) {
}
```

- 块级代码 (if, else, for 等) 基本与函数保持一致

```
// bad
if(isSuccess){
}
//bad
if ( isSuccess ) {
}

// good
if (isSuccess) {
}
```

- 对象

```
// bad
var base = {
  width : 100,
  height:200
};

// good
var base = {
  width: 100,
  height: 200
};
```

- key, value 之间只需要一个空格，因为这样更加自然与易读。

```
// bad
{
  a           : 'short',
  looooongname: 'long'
}

// bad
{
  a:            'short',
  looooongname: 'long'
}

// good
{
  a: 'short',
  looooongname: 'long'
}
```

- 避免使用过长的链式调用；调用比较长时建议使用换行和缩进来组织结构

```
// bad
$('#items').find('.selected').highlight().end().find

// good
$('#items')
  .find('.selected')
    .highlight()
    .end()
  .find('.open')
    .updateCount();
```

逗号

- 逗号后置

```
// bad
var options = {
    name: '马云'
  , gender: '男'
  , age: 1024
};

// good
var options = {
  name: '马云',
  gender: '男',
  age: 1024
};
```

- 不要加多余的逗号，这可能会在 IE 下引起错误，同时如果多一个逗号某些 ES3 的实现会导致数组长度加1。

```
// bad
var hero = {
  firstName: 'Kevin',
  lastName: 'Flynn',
};

var heroes = [
  'Batman',
  'Superman',
];

// good
var hero = {
  firstName: 'Kevin',
  lastName: 'Flynn'
};

var heroes = [
  'Batman',
  'Superman'
];
```

类型转换

- 在语句的开始执行类型转换
- 对数字使用 `parseInt` 并且总是带上类型转换的基数，备注： `parseFloat` 方法没有基数参数

```
var inputValue = '4';

// bad
var val = + inputValue;

// bad
var val = parseInt(inputValue);

// good
var val = Number(inputValue);

// good
var val = parseInt(inputValue, 10);
```

- 布尔值:

```
var age = 0;

// bad
var hasAge = new Boolean(age);

// good
var hasAge = !!age;
```

命名规范

原则

所有的命名必须有语义，所以永远不要使用单个字符命名。

变量

- 变量声明必须使用 `var`，命名规范统一为**小驼峰命名法**：即第一个单词首字母小写，其余单词的首字母大写。

```
// dirty
eventType = 'click';

// bad
var eventtype = 'click';

// bad
var EventType = 'click';

// good
var eventType = 'click';
```

- 对象字面量命名使用小驼峰

```
// bad
var Base = {
  isOnline: function() {
    // ...
  }
};
return Base;

// good
var base = {
  isOnline: function() {
    // ...
  }
};
return base;
```

常量

- 常量的命名使用全大写字母，多单词使用下划线分开，例如：

```
// good
var WELCOME_TEXT = 'Hello World';
```

构造函数/类

- 构造函数/类使用大驼峰命名，即所有单词首字母大写。如下示例：

```
// good
function MenuButton() {
}

MenuButton.prototype.show = function () {
};
```

函数名

- 普通函数名使用小驼峰，尽可能的选用动词作为名字。

布尔值

- 尽量使用表示状态的词汇来命名布尔值：可以是形容词、副词，或充当状语成分的词组。这种情况下，不要添加诸如 `is`，`has`，`can` 的前缀，因为那样看上去像是命名一个方法。

```
// good
this.available = true;
this.isAvailable = function () {
  return this.available;
};

// bad
this.isAvailable = true;
```

- 如果无法通过上一种情况来命名，比如描述的目标是一个名词，那么可以添加诸如 `is`，`has`，`can` 的前缀。

```
// good
this.hasName = true;

// bad
this.name = true;
```

- 使用「表示肯定的」词汇来命名布尔值，而不是「表示否定的」。因为这样更易读，并且 **具有更好的兼容性**：因为如果此变量未定义，那么其默认值为 `undefined`，等价于 `false`。

```
// good
this.available = false;

// bad
this.unavailable = true;
```

配置项

- 组件配置项属性名使用小驼峰

```
// bad
new Compoment({
  event_type: 'click'
});

// good
new Compoment({
  eventType: 'click'
});
```

私有属性前面加下划线 _

```
// bad
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';

// good
this._firstName = 'Panda';
```

注释规范

原则

- 不要为了注释而注释，多余的注释等价于冗余的代码
- 如有必要，注释尽量详尽

注释的目的是：提高代码的可读性，从而提高代码的可维护性。

单行注释

单行注释使用 `//`，一般面向的是语句或者简单逻辑的代码块 (if, for, function)

- 应独立于一行，不要追加在某条语句的后面

```
// bad
var active = true;  // is current tab

// good
// is current tab
var active = true;
```

- 在单行注释前添加一个空行，便于阅读区分

```
// bad
function getType() {
  console.log('fetching type...');
  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}

// good
function getType() {
  console.log('fetching type...');

  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}
```

多行注释:

一般情况下，多行注释用于函数/对象/文件。

- 多行注释使用 `/** */` 或 `/* */`, 不推荐使用多行的 `//` 来替代 `/**/`.

```
// bad
// it's foo
// but not bar
function foo() {
}

// good
// it's foo, but not bar
function foo() {
}

// good
/*
 * it's foo, but not bar
 */
function foo() {
}

// good
/**
 * it's foo, but not bar
 */
function foo() {
}
```

对于公共的类/库/组件等代码，面向较多的使用者，推荐使用更为完善的注释规范：

- 函数功能说明，**必选**
- 参考文档链接，可选
- 示例，可选
- 参数说明，可选
- 返回值说明，有则必选
- api 类型(public/private), 可选

如下，是一个良好风格的示例：

```
/**
 * 发送旺旺/邮件的方法
 *
 * Doc: http://api.fed.taobao.net/mc
 *
 * Example:
 *   message(1, '马云', 'Hello')
 *     .then(function(result) {
 *       // 成功逻辑处理
 *     })
 *     .catch(function(err) {
 *       // 错误逻辑处理
 *     });
 *
 * @param {Number}   type     0-旺旺 1-邮件
 * @param {String}   name     旺旺昵称
 * @param {String}   content  消息内容
 * @return {Promise}
 * @api public
 */
function message(type, name, content) {

  // some code
  return new Promise(function(resolve, reject) {

  });

};
```

文件注释

文件注释，即声明在文件头部，描述文件的元信息。文件注释的基本规则：

- 文件描述，**必选**

```
/**
 * 这个文件的作用是什么
 */
```

协议注释

对于引用的外部框架/库，以及我们自己写的开源库，头部会包含一些开源协议声明的注释，对于这部分的注释，我们是**不希望也不能**将之压缩，这时候需要使用 `/*! */` 的注释方式，如下：

```
/*!
 * jQuery JavaScript Library v2.1.4
 * http://jquery.com/
 *
 * Includes Sizzle.js
 * http://sizzlejs.com/
 *
 * Copyright 2005, 2014 jQuery Foundation, Inc. and oth
 * Released under the MIT license
 * http://jquery.org/license
 *
 * Date: 2015-04-28T16:01Z
 */
```

标记注释

平时的写代码的过程，可能会遇到某个地方是个隐藏的 bug, 因为某种原因还没法修复，或者是方法还有一些需要完成的功能，这时候我们需要加上相应的注释告知未来的自己或者是合作者，这里常用的有三种标签：

- `// TODO:` 之后需要做的一些事
- `// FIXME:` 存在 bug, 最好能修复掉
- `// XXX:` 存在致命的 bug, 看到务必修复

如下为示例代码：

```
// TODO 需要考虑参数个数不确定的情况
funcion add(a, b) {
    return a + b;
}

function render(data) {
    // FIXME status 不为 1 的情况没有考虑
    if (data.status === 1) {
        // ...
    }
}

// XXX 异常会导致系统挂掉
JSON.parse(data);
```

参考资料

- Airbnb JavaScript Style Guide (https://github.com/airbnb/javascript)
- aralejs: JavaScript 注释规范 (https://github.com/aralejs/aralejs.org/wiki/JavaScript-%E6%B3%A8%E9%87%8A%E8%A7%84%E8%8C%83)
- Popular Coding Convention on Github (http://sideeffect.kr/popularconvention#javascript)