**[js命名规范](https://www.cnblogs.com/Hsong/p/9016950.html)**

ECMAScript 规范中标识符采用驼峰大小写格式，驼峰命名法由小(大)写字母开始，后续每个单词首字母都大写。根据首字母是否大写，分为两种方式：

1. Pascal Case 大驼峰式命名法：首字母大写。eg：StudentInfo、UserInfo、ProductInfo
2. Camel Case 小驼峰式命名法：首字母小写。eg：studentInfo、userInfo、productInfo

标识符，则包括变量、函数名、类名、属性名和函数或类的参数，每个命名方法又略有不同，下面详细解释一下：

**1.项目命名**

全部采用小写方式， 以下划线分隔。

示例：my\_project\_name

**2.目录命名**

参照项目命名规则；有复数结构时，要采用复数命名法。

示例：scripts, styles, images, data\_models

**3.JS文件命名**

1. 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 {

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);