**前端开发规范v1.0**

# 前言

随着团队人数的增加，每个人的代码编写喜好不同，代码风格也迥然不同。如果有一个大家的统一的愿意遵守的代码规范，肯定事半功倍，提高效率，避免代码Review和重构。

坚持好的代码风格规范，从你我做起。

# 命名规范

## 项目命名

全部采用小写方式， 以下划线分隔。

例：my\_project\_name

## 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。

例：pages, assets, directives, components, mixins, utils

## javaScript 文件命名

参照项目命名规则。

例：account\_model.js

## css, less, scss文件命名

参照项目命名规则。

例：retina\_sprites.less

## HTML文件命名

参照项目命名规则。

例：error\_report.html

#### 如果使用Vue或者React技术栈，组件Component命名

所有组件名字需要首字母大写，然后驼峰格式

例：CalendarList.vue

# HTML

## 语法

* 缩进使用soft tab（4个空格）；
* 嵌套的节点应该缩进；
* 在属性上，使用双引号，不要使用单引号；
* 属性名全小写，用中划线做分隔符；
* 不要在自动闭合标签结尾处使用斜线（[HTML5 规范](http://dev.w3.org/html5/spec-author-view/syntax.html" \l "syntax-start-tag" \t "https://juejin.im/post/_blank) 指出他们是可选的）；
* 不要忽略可选的关闭标签;

<!DOCTYPE html>

<html>

    <head>

        <title>Page title</title>

    </head>

    <body>

        <img src="images/company\_logo.png" alt="Company" />

        <h1 class="hello-world">Hello, world!</h1>

    </body>

</html>

## HTML5 doctype

在页面开头使用这个简单地doctype来启用标准模式，使其在每个浏览器中尽可能一致的展现；虽然doctype不区分大小写，但是按照惯例，doctype大写 （[关于html属性，大写还是小写](http://stackoverflow.com/questions/15594877/is-there-any-benefits-to-use-uppercase-or-lowercase-letters-with-html5-tagname" \t "https://juejin.im/post/_blank)）。

**<!DOCTYPE html>**

<html>

    ...

</html>

## lang属性

根据HTML5规范：

应在html标签上加上lang属性。这会给语音工具和翻译工具帮助，告诉它们应当怎么去发音和翻译。

更多关于 lang 属性的说明[在这里](http://www.w3.org/html/wg/drafts/html/master/semantics.html" \l "the-html-element" \t "https://juejin.im/post/_blank)；

在sitepoint上可以查到[语言列表](http://reference.sitepoint.com/html/lang-codes" \t "https://juejin.im/post/_blank)；

但sitepoint只是给出了语言的大类，例如中文只给出了zh，但是没有区分香港，台湾，大陆。而微软给出了一份更加[详细的语言列表](http://msdn.microsoft.com/en-us/library/ms533052(v=vs.85).aspx" \t "https://juejin.im/post/_blank)，其中细分了zh-cn, zh-hk, zh-tw。

**<!DOCTYPE html>**

<html lang="en-us">

...

</html>

## 字符编码

通过声明一个明确的字符编码，让浏览器轻松、快速的确定适合网页内容的渲染方式，通常指定为'UTF-8'。

<!DOCTYPE html>

<html>

    <head>

        <meta charset="UTF-8" />

    </head>

    ...

</html>

## IE兼容模式

用 `` 标签可以指定页面应该用什么版本的IE来渲染；

不同doctype在不同浏览器下会触发不同的渲染模式。

<!DOCTYPE html>

<html>

    <head>

        <meta http-equiv="X-UA-Compatible" content="IE=Edge" />

    </head>

    ...

</html>

## 引入CSS, JS

根据HTML5规范, 通常在引入CSS和JS时不需要指明 type，因为 text/css和 text/javascript 分别是他们的默认值。

说到js和css的位置，大家应该都知道js放在下面，css放在上面。但是，如果你的项目只需要兼容ie10+或者只是在移动端访问，那么可以使用HTML5的新属性async，将脚本文件放在<head>内，兼容老旧浏览器(IE9-)时：脚本引用写在 body 结束标签之前，并带上 async 属性。这虽然在老旧浏览器中不会异步加载脚本，但它只阻塞了 body 结束标签之前的 DOM 解析，这就大大降低了其阻塞影响。

而在现代浏览器中：

脚本将在 DOM 解析器发现 body 尾部的 script 标签才进行加载，此时加载属于异步加载，不会阻塞 CSSOM（但其执行仍发生在 CSSOM 之后）。

综上所述，

所有浏览器中推荐:

<html>

    <head>

        <link rel="stylesheet" href="main.css" />

    </head>

    <body>

        <!-- body goes here -->

        <script src="main.js" async></script>

    </body>

</html>

只兼容现代浏览器推荐:

<html>

    <head>

        <link rel="stylesheet" href="main.css" />

        <script src="main.js" async></script>

    </head>

    <body>

        <!-- body goes here -->

    </body>

</html>

**HTML5 规范链接**

[使用link](http://www.w3.org/TR/2011/WD-html5-20110525/semantics.html" \l "the-link-element" \t "https://juejin.im/post/_blank) 、[使用style](http://www.w3.org/TR/2011/WD-html5-20110525/semantics.html" \l "the-style-element" \t "https://juejin.im/post/_blank) 、[使用script](http://www.w3.org/TR/2011/WD-html5-20110525/scripting-1.html" \l "the-script-element" \t "https://juejin.im/post/_blank)

<!-- External CSS -->

<link rel="stylesheet" href="code\_guide.css" />

<!-- In-document CSS -->

<style>

    ...;

</style>

<!-- External JS -->

<script src="code\_guide.js"></script>

<!-- In-document JS -->

<script>

    ...

</script>

## 属性顺序

属性应该按照特定的顺序出现以保证易读性；

class

id

name

data-\*

src, for, type, href, value , max-length, max, min, pattern

placeholder, title, alt

aria-\*, role

required, readonly, disabled

class是为高可复用组件设计的，所以应处在第一位；

id 具体且应该尽量少使用，所以将它放在第二位。

<a class="..." id="..." data-modal="toggle" href="#">Example link</a>

<input class="form-control" type="text">

<img src="..." alt="...">

## boolean属性

boolean属性指不需要声明取值的属性，XHTML需要每个属性声明取值，但是HTML5并不需要；

更多内容可以参考 [WhatWG section on boolean attributes](http://www.whatwg.org/specs/web-apps/current-work/multipage/common-microsyntaxes.html" \l "boolean-attributes" \t "https://juejin.im/post/_blank)：

boolean属性的存在表示取值为true，不存在则表示取值为false。

<input type="text" disabled />

<input type="checkbox" value="1" checked />

<select>

    <option value="1" selected>1</option>

</select>

## JS生成标签

在JS文件中生成标签让内容变得更难查找，更难编辑，性能更差。应该尽量避免这种情况的出现。

## 减少标签数量

在编写HTML代码时，需要尽量避免多余的父节点；

很多时候，需要通过迭代和重构来使HTML变得更少。

<!-- 不建议这么做 -->

<span class="avatar">

    <img src="..." />

</span>

<!-- 建议这么做 -->

<img class="avatar" src="..." />

## 实用高于完美

尽量遵循HTML标准和语义，但是不应该以浪费实用性作为代价；

任何时候都要用尽量小的复杂度和尽量少的标签来解决问题。

# css、less、scss

## 缩进

使用soft tab（4个空格）

Less

.element {

position: absolute;

top: 10px;

left: 10px;

border-radius: 10px;

width: 50px;

height: 50px;

}

## 分号

每个属性声明末尾都要加分号。

Less

.element {

width: 20px;

height: 20px;

background-color: red;

}

## 空格

以下几种情况不需要空格：

* 属性名后
* 多个规则的分隔符','前
* !important '!'后
* 属性值中'('后和')'前
* 行末不要有多余的空格

以下几种情况需要空格：

* 属性值前
* 选择器'>', '+', '~'前后
* '{'前
* !important '!'前
* @else 前后
* 属性值中的','后
* 注释'/*'后和'*/'前

/\* not good \*/

.element{

    color: red!important;

    background-color: rgba(0, 0, 0, 0.5);

}

/\* good \*/

.element {

    color: red !important;

    background-color: rgba(0, 0, 0, 0.5);

}

/\* not good \*/

.element,.dialog{

    ...;

}

/\* good \*/

.element,

.dialog {

}

/\* not good \*/

.element>.dialog {

    ...;

}

/\* good \*/

.element > .dialog {

    ...;

}

/\* not good \*/

.element{

    ...;

}

/\* good \*/

.element {

    ...;

}

/\* not good \*/

@debug: true;

header {

    background-color: (yellow) when(@debug = true);

}

/\* good \*/

header {

    background-color: (yellow) when (@debug = true);

}

## 空行

以下几种情况需要空行：

* 文件最后保留一个空行
* '}'后最好跟一个空行，包括scss中嵌套的规则
* 属性之间需要适当的空行，具体见[属性声明顺序](http://alloyteam.github.io/CodeGuide/" \l "css-declaration-order" \t "https://juejin.im/post/_blank)

/\* not good \*/

.element {

    ...;

}

.dialog {

    color: red;

    &:after {

        ...;

    }

}

/\* good \*/

.element {

    ...;

}

.dialog {

    color: red;

    &:after {

        ...;

    }

}

## 注释

注释统一用'/\* \*/'（scss中也不要用'//'），具体参照右边的写法；

缩进与下一行代码保持一致；

可位于一个代码行的末尾，与代码间隔一个空格。

Less

/\* Modal header \*/

.modal-header {

...

}

/\*

\* Modal header

\*/

.modal-header {

...

}

.modal-header {

/\* 50px \*/

width: 50px;

color: red; /\* color red \*/

}

## 引号

最外层统一使用双引号；

url的内容要用引号；

属性选择器中的属性值需要引号。

Less

.element:after {

content: "";

background-image: url("logo.png");

}

**li**[data-type="single"] {

...

}

## 命名

类名使用小写字母，以中划线分隔

id采用驼峰式命名

less中的变量、函数以中划线分隔命名

Less

/\* class \*/

.element-content {

...

}

/\* id \*/

**#myDialog** {

...

}

/\* 变量 \*/

@color-black: #000;

/\* mixins \*/

.my-mixin() {

color: black;

}

## 属性声明顺序

相关的属性声明按右边的顺序做分组处理，组之间需要有一个空行。

.declaration-order {

display: block;

float: right;

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

z-index: 100;

border: 1px solid #e5e5e5;

border-radius: 3px;

width: 100px;

height: 100px;

font: normal 13px "Helvetica Neue", sans-serif;

line-height: 1.5;

text-align: center;

color: #333;

background-color: #f5f5f5;

opacity: 1;

}

书写顺序前后为：

（1）定位属性：position display float left top right bottom overflow clear z-index

（2）自身属性：width height padding border margin background

（3）文字样式：font-family font-size font-style font-weight font-varient color

（4）文本属性text-align vertical-align text-wrap text-transform text-indent text-decoration letter-spacing word-spacing white-space text-overflow

**目的**：减少浏览器reflow（回流），提升浏览器渲染dom的性能

原理：浏览器的渲染流程为：

1、解析html构建dom树，解析css构建css树：将html解析成树形的数据结构，将css解析成树形的数据结构

2、构建render树：DOM树和CSS树合并之后形成的render树。

3、布局render树：有了render树，浏览器已经知道那些网页中有哪些节点，各个节点的css定义和以及它们的从属关系，从而计算出每个节点在屏幕中的位置。

4、绘制render树：按照计算出来的规则，通过显卡把内容画在屏幕上。

css样式解析到显示至浏览器屏幕上就发生在234步骤，可见浏览器并不是一获取到css样式就立马开始解析而是根据css样式的书写顺序将之按照dom树的结构分布render样式，完成第2步，然后开始遍历每个树结点的css样式进行解析，此时的css样式的遍历顺序完全是按照之前的书写顺序。在解析过程中，一旦浏览器发现某个元素的定位变化影响DOM，则需要重新渲染。

正确的书写顺序：

Less

.demo{

width: 100px;

    height: 100px;

    background-color: red ;

    position: absolute;

}

当浏览器解析到position的时候突然发现该元素是绝对定位元素需要脱离文档流，而之前却是按照普通元素进行解析的，所以不得不重新渲染，解除该元素在文档中所占位置，然而由于该元素的占位发生变化，其他元素也可能会受到它回流的影响而重新排位。最终导致③步骤花费的时间太久而影响到④步骤的显示，影响了用户体验。

## 颜色

颜色16进制用小写字母；

Less

/\* not good \*/

.element {

    color: #ABCDEF;

    background-color: #001122;

}

/\* good \*/

.element {

    color: #abcdef;

    background-color: #001122;

}

## 属性简写

属性简写需要你非常清楚属性值的正确顺序，而且在大多数情况下并不需要设置属性简写中包含的所有值，所以建议尽量分开声明会更加清晰；

margin 和 padding 相反，需要使用简写；

常见的属性简写包括：

font

background

transition

Animation

/\* not good \*/

.element {

    transition: opacity 1s linear 2s;

}

/\* good \*/

.element {

    transition-delay: 2s;

    transition-timing-function: linear;

    transition-duration: 1s;

    transition-property: opacity;

}

## 媒体查询

尽量将媒体查询的规则靠近与他们相关的规则，不要将他们一起放到一个独立的样式文件中，或者丢在文档的最底部，这样做只会让大家以后更容易忘记他们。

.element {

    ...;

}

.element-avatar {

    ...;

}

@media (min-width: 480px) {

    .element {

        ...;

    }

    .element-avatar {

        ...;

    }

}

## Less相关

每个模块应该有一个单独的less, 然后每个最外层的父类 className 应该写在第一位，所有子Node的样式都写在里面，这样是为了避免命名冲突。比如

Less

//out: false

.parent-name {

    .child-name {

        ...;

    }

    ...;

}

* @import 引入的文件不需要结尾的'.less'
* LESS嵌套最多不能超过5层；
* 不允许有空的规则；
* 元素选择器用小写字母；
* 去掉小数点前面的0；
* 去掉数字中不必要的小数点和末尾的0；
* 属性值'0'后面不要加单位；
* 同个属性不同前缀的写法需要在垂直方向保持对齐，具体参照右边的写法；
* 无前缀的标准属性应该写在有前缀的属性后面；
* 不要在同个规则里出现重复的属性，如果重复的属性是连续的则没关系；
* 不要在一个文件里出现两个相同的规则；
* 用 border: 0; 代替 border: none;；
* CSS选择器不要超过3层；
* 尽量少用'\*'选择器。

/\* not good \*/

.element {

    width: 0px;

}

/\* good \*/

.element {

    width: 0;

}

/\* not good \*/

.element {

    border-radius: 3px;

    -webkit-border-radius: 3px;

    -moz-border-radius: 3px;

    background: linear-gradient(to bottom, #fff 0, #eee 100%);

    background: -webkit-linear-gradient(top, #fff 0, #eee 100%);

    background: -moz-linear-gradient(top, #fff 0, #eee 100%);

}

/\* good \*/

.element {

    -webkit-border-radius: 3px;

    -moz-border-radius: 3px;

    border-radius: 3px;

    background: -webkit-linear-gradient(top, #fff 0, #eee 100%);

    background: -moz-linear-gradient(top, #fff 0, #eee 100%);

    background: linear-gradient(to bottom, #fff 0, #eee 100%);

}

/\* not good \*/

.element {

    color: rgb(0, 0, 0);

    width: 50px;

    color: rgba(0, 0, 0, 0.5);

}

/\* good \*/

.element {

    color: rgb(0, 0, 0);

    color: rgba(0, 0, 0, 0.5);

}

# JavaScript

## 缩进

使用soft tab（4个空格）。

var x = 1,

    y = 1;

if (x < y) {

    x += 10;

} else {

    x += 1;

}

## 分号

以下几种情况后需加分号：

* 变量声明
* 表达式
* return
* throw
* break
* continue
* do-while

Javascript

/\* var declaration \*/

**var** x = 1;

/\* expression statement \*/

x++;

/\* do-while \*/

**do** {

x++;

} **while** (x < 10);

## 空格

以下几种情况不需要空格：

* 对象的属性名后
* 前缀一元运算符后
* 后缀一元运算符前
* 函数调用括号前
* 无论是函数声明还是函数表达式，'('前不要空格
* 数组的'['后和']'前
* 对象的'{'后和'}'前
* 运算符'('后和')'前

以下几种情况需要空格：

* 二元运算符前后
* 三元运算符'?:'前后
* 代码块'{'前
* 下列关键字前：else, while, catch, finally
* 下列关键字后：if, else, for, while, do, switch, case, try,catch, finally, with, return, typeof
* 单行注释'//'后（若单行注释和代码同行，则'//'前也需要），多行注释'\*'后
* 对象的属性值前
* for循环，分号后留有一个空格，前置条件如果有多个，逗号后留一个空格
* 无论是函数声明还是函数表达式，'{'前一定要有空格
* 函数的参数之间

Javascript

// not good

**var** a = {

b :1

};

// good

**var** a = {

b: 1

};

// not good

++ x;

y ++;

z = x?1:2;

// good

++x;

y++;

z = x ? 1 : 2;

// not good

**var** a = [ 1, 2 ];

// good

**var** a = [1, 2];

// not good

**var** a = ( 1+2 )\*3;

// good

**var** a = (1 + 2) \* 3;

// no space before '(', one space before '{', one space between function parameters

**var** doSomething = **function**(a, b, c) {

// do something

};

// no space before '('

doSomething(item);

// not good

**for**(i=0;i<6;i++){

x++;

}

// good

**for** (i = 0; i < 6; i++) {

x++;

}

## 空行

以下几种情况需要空行：

* 变量声明后（当变量声明在代码块的最后一行时，则无需空行）
* 注释前（当注释在代码块的第一行时，则无需空行）
* 代码块后（在函数调用、数组、对象中则无需空行）
* 文件最后保留一个空行

Javascript

// need blank line after variable declaration

var x = 1;

// not need blank line when variable declaration is last expression in the current block

if (x >= 1) {

    var y = x + 1;

}

var a = 2;

// need blank line before line comment

a++;

function b() {

    // not need blank line when comment is first line of block

    return a;

}

// need blank line after blocks

for (var i = 0; i < 2; i++) {

    if (true) {

        return false;

    }

    continue;

}

var obj = {

    foo: function() {

        return 1;

    },

    bar: function() {

        return 2;

    }

};

// not need blank line when in argument list, array, object

func(

    2,

    function() {

        a++;

    },

    3

);

var foo = [

    2,

    function() {

        a++;

    },

    3

];

var foo = {

    a: 2,

    b: function() {

        a++;

    },

    c: 3

};

## 换行

换行的地方，行末必须有','或者运算符；

以下几种情况不需要换行：

* 下列关键字后：else, catch, finally
* 代码块'{'前
* 以下几种情况需要换行：
* 代码块'{'后和'}'前
* 变量赋值后

Javascript

// not good

var a = {

    b: 1

    , c: 2

    };

    x = y

    ? 1 : 2;

    // good

    var a = {

    b: 1,

    c: 2

    };

    x = y ? 1 : 2;

    x = y ?

    1 : 2;

    // no need line break with 'else', 'catch', 'finally'

    if (condition) {

    ...

    } else {

    ...

    }

    try {

    ...

    } catch (e) {

    ...

    } finally {

    ...

    }

    // not good

    function test()

    {

    ...

    }

    // good

    function test() {

    ...

    }

    // not good

    var a, foo = 7, b,

    c, bar = 8;

    // good

    var a,

    foo = 7,

    b, c, bar = 8;

## 单行注释

双斜线后，必须跟一个空格；

缩进与下一行代码保持一致；

可位于一个代码行的末尾，与代码间隔一个空格。

Javascript

**if** (condition) {

// if you made it here, then all security checks passed

allowed();

}

**var** zhangsan = 'zhangsan'; // one space after code

## 多行注释

最少三行, '\*'后跟一个空格，具体参照右边的写法；

建议在以下情况下使用：

* 难于理解的代码段
* 可能存在错误的代码段
* 浏览器特殊的HACK代码
* 业务逻辑强相关的代码

Javascript

/\*

\* one space after '\*'

\*/

**var** x = 1;

## 文档注释

各类标签@param, @method等请参考[usejsdoc](http://usejsdoc.org/" \t "https://juejin.im/post/_blank)和[JSDoc Guide](http://yuri4ever.github.io/jsdoc/" \t "https://juejin.im/post/_blank)；

建议在以下情况下使用：

所有常量

所有函数

所有类

/\*\*

\* @func

\* @desc 一个带参数的函数

\* @param {string} a - 参数a

\* @param {number} b=1 - 参数b默认值为1

\* @param {string} c=1 - 参数c有两种支持的取值</br>1—表示x</br>2—表示xx

\* @param {object} d - 参数d为一个对象

\* @param {string} d.e - 参数d的e属性

\* @param {string} d.f - 参数d的f属性

\* @param {object[]} g - 参数g为一个对象数组

\* @param {string} g.h - 参数g数组中一项的h属性

\* @param {string} g.i - 参数g数组中一项的i属性

\* @param {string} [j] - 参数j是一个可选参数

\*/

**function** foo(a, b, c, d, g, j) {

...

}

## 引号

最外层统一使用单引号。

Javascript

// not good

**var** x = "test";

// good

**var** y = 'foo',

z = '<div id="test"></div>';

## 变量命名

* 标准变量采用驼峰式命名（除了对象的属性外，主要是考虑到cgi返回的数据）
* 'ID'在变量名中全大写
* 'URL'在变量名中全大写
* 'Android'在变量名中大写第一个字母
* 'iOS'在变量名中小写第一个，大写后两个字母
* 常量全大写，用下划线连接
* 构造函数，大写第一个字母
* jquery对象必须以'$'开头命名

Javascript

**var** thisIsMyName;

**var** goodID;

**var** reportURL;

**var** AndroidVersion;

**var** iOSVersion;

**var** MAX\_COUNT = 10;

**function** **Person**(name) {

**this**.name = name;

}

// not good

**var** body = $('body');

// good

**var** $body = $('body');

## 变量声明

### 函数

一个函数作用域中所有的变量声明尽量提到函数首部，用一个var 声明，不允许出现两个连续的var声明。

Javascript

function doSomethingWithItems(items) {

    // use one var

    var value = 10,

        result = value + 10,

        i,

        len;

    for (i = 0, len = items.length; i < len; i++) {

        result += 10;

    }

}

* 无论是函数声明还是函数表达式，'('前不要空格，但'{'前一定要有空格；
* 函数调用括号前不需要空格；
* 立即执行函数外必须包一层括号；
* 不要给inline function命名；
* 参数之间用', '分隔，注意逗号后有一个空格。

Javascript

// no space before '(', but one space before'{'

**var** doSomething = **function**(item) {

// do something

};

**function** **doSomething**(item) {

// do something

}

// not good

doSomething (item);

// good

doSomething(item);

// requires parentheses around immediately invoked function expressions

(**function**() {

**return** 1;

})();

// not good

[1, 2].forEach(**function** **x**() {

...

});

// good

[1, 2].forEach(**function**() {

...

});

// not good

**var** a = [1, 2, **function** **a**() {

...

}];

// good

**var** a = [1, 2, **function**() {

...

}];

// use ', ' between function parameters

**var** doSomething = **function**(a, b, c) {

// do something

};

### 数组、对象

对象属性名不需要加引号；

对象以缩进的形式书写，不要写在一行；

数组、对象最后不要有逗号。

Javascript

// not good

**var** a = {

'b': 1

};

**var** a = {b: 1};

// good

**var** a = {

b: 1,

c: 2

};

### 括号

下列关键字后必须有大括号（即使代码块的内容只有一行）：if, else,for, while, do, switch, try, catch, finally, with。

Javascript

// not good

**if** (condition)

doSomething();

// good

**if** (condition) {

doSomething();

}

复制代码

### null

适用场景：

初始化一个将来可能被赋值为对象的变量

与已经初始化的变量做比较

作为一个参数为对象的函数的调用传参

作为一个返回对象的函数的返回值

不适用场景：

不要用null来判断函数调用时有无传参

不要与未初始化的变量做比较

javascript// not good

**function** **test**(a, b) {

**if** (b === null) {

// not mean b is not supply

...

}

}

var a;

if (a === null) {

...

}

// good

**var** a = null;

**if** (a === null) {

...

}

### undefined

永远不要直接使用undefined进行变量判断；

使用typeof和字符串'undefined'对变量进行判断。

javascript// not good

**if** (person === undefined) {

...

}

// good

if (typeof person === 'undefined') {

...

}

### 杂项

不要混用tab和space；

不要在一处使用多个tab或space；

换行符统一用'LF'；

对上下文this的引用只能使用'that', 'self'其中一个来命名；

行尾不要有空白字符；

switch的falling through和no default的情况一定要有注释特别说明；

不允许有空的代码块。

javascript// not good

**var** a = 1;

**function** **Person**() {

// not good

**var** me = **this**;

// good

**var** that = **this**;

// good

**var** self = **this**;

}

// good

**switch** (condition) {

**case** 1:

**case** 2:

...

break;

**case** 3:

...

// why fall through

**case** 4

...

break;

// why no default

}

// not good with empty block

**if** (condition) {

}

插件推荐： EditorConfig for VS Code 生成.editorconfig文件

root = true

[\*]

indent\_style = space

indent\_size = 4

charset = utf-8

trim\_trailing\_whitespace = true

insert\_final\_newline = true

[\*.md]

trim\_trailing\_whitespace = false