协同数据
Collaboration Data System

协同开发框架

前端开发规范

浙江协同数据系统有限公司
+86 571 81951353
杭州市文三路 90 号东部软件园科技大厦 9 层
www.zjcds.com postmaster@zjcds.com

目录

引言

1.1 编写目的

本文档的目标是使前端开发人员能够按照统一规范进行编写，使其风格保持一致，便于理解和维护。

1.2 使用范围

公司内部所有技术部门前端开发人员及公司定制开发系统供应商的前端开发人员。

第 1 部分 CSS 编写规范

1.1 代码风格

1.1.1 缩进，空格

【强制】使用 2 个空格做为一个缩进层级，不允许使用 4 个空格 或 tab 字符。

【强制】选择器与 { 之间必须包含空格。

【强制】属性名与之后的 : 之间不允许包含空格， : 与属性值之间必须包含空格。

【强制】列表型属性值书写在单行时，后必须跟一个空格。

【示例】

```
.selector {
  margin: 0;
  padding: 0;
  font-family: Arial, sans-serif;
}
```

1.1.2 行长度

【强制】每行不得超过 120 个字符，除非单行不可分割。

【示例】

```
.selector {
  /* 不同属性值按逻辑分组 */
  background:
    transparent url(aVeryVeryVeryLongUrlIsPlacedHere)
    no-repeat 0 0;
}
```

```
.selector {
  /* 可重复多次的属性，每次重复一行 */
  background-image:
    url(aVeryVeryVeryLongUrlIsPlacedHere)
    url(anotherVeryVeryVeryLongUrlIsPlacedHere);

  /* 类似函数的属性值可以根据函数调用的缩进进行 */
  background-image: -webkit-gradient(
    linear,
    left bottom,
    left top,
    color-stop(0.04, rgb(88,94,124)),
    color-stop(0.52, rgb(115,123,162))
  );
}
```

1.1.3 选择器

【强制】当一个 rule 包含多个 selector 时，每个选择器声明必须独占一行。

【示例】

```
/* good */
.post,
.page,
.comment {
  line-height: 1.5;
}

/* bad */
.post, .page, .comment {
  line-height: 1.5;
}
```

1.1.4 有符号的空格

【强制】>、+、~ 选择器的两边各保留一个空格。

【示例】

```css
/* good */
main > nav {
  padding: 10px;
}

label + input {
  margin-left: 5px;
}

input:checked ~ button {
  background-color: #69C;
}

/* bad */
main>nav {
  padding: 10px;
}

label+input {
  margin-left: 5px;
}

input:checked~button {
  background-color: #69C;
}
```

1.1.5 属性选择器

【强制】属性选择器中的值必须用双引号包围。

【示例】

```css
/* good */
article[character="juliet"] {
  font-family: "Vivien Leigh", victoria, female;
}
```

```
/* bad */
article[character='juliet'] {
  font-family: "Vivien Leigh", victoria, female;
}
```

1.1.6 属性定义

【强制】属性定义必须另起一行。

【强制】属性定义后必须以分号结尾。

【示例】

```
/* good */
.selector {
  margin: 0;
  padding: 0;
}

/* bad */
.selector { margin: 0; padding: 0; }

/* good */
.selector {
  margin: 0;
}

/* bad */
.selector {
  margin: 0
}
```

1.2 通用

1.2.1 选择器

【强制】如无必要，不得为 id、class 选择器添加类型选择器进行限定。（在性能

和维护性上，都有一定的影响。）

【建议】选择器的嵌套层级应不大于 3 级，位置靠后的限定条件应尽可能精确。

【示例】

```
/* good */
#error,
.danger-message {
  color: #c00;
}

/* bad */
dialog#error,
p.danger-message {
  color: #c00;
}

/* good */
#username input {
  font-size: 12px;
}
.comment .avatar {
  font-size: 12px;
}

/* bad */
.page .header .login #username input {
  font-size: 12px;
}
.comment div * {
  font-size: 12px;
}
```

1.2.2 属性缩写

【建议】在可以使用缩写的情况下，尽量使用属性缩写。

1.2.3 清除浮动

【建议】当元素需要撑起高度以包含内部的浮动元素时，通过对伪类设置 clear 或触发 BFC 的方式进行 clearfix。尽量不使用增加空标签的方式。

触发 BFC 的方式很多，常见的有：

1. float 非 none
2. position 非 static
3. overflow 非 visible

另需注意，对已经触发 BFC 的元素不需要再进行 clearfix。

1.2.4 !important 声明

【建议】尽量不使用 !important 声明。当需要强制指定样式且不允许任何场景覆盖时，通过标签内联和 !important 定义样式。

1.2.5 z-index 设置

【建议】 将 z-index 进行分层，对文档流外绝对定位元素的视觉层级关系进行管理。同层的多个元素，如多个由用户输入触发的 Dialog，在该层级内使用相同的 z-index 或递增 z-index。

【建议】在可控环境下，期望显示在最上层的元素，z-index 指定为 999999。

1.3 值与单位

1.3.1 文本

【强制】文本内容必须用双引号包围。

【示例】

```
/* good */
html[lang|="zh"] q:before {
  font-family: "Microsoft YaHei", sans-serif;
  content: "“";
}

html[lang|="zh"] q:after {
  font-family: "Microsoft YaHei", sans-serif;
  content: "”";
}

/* bad */
html[lang|=zh] q:before {
  font-family: 'Microsoft YaHei', sans-serif;
  content: '“';
}


html[lang|=zh] q:after {
  font-family: "Microsoft YaHei", sans-serif;
  content: "”";
}
```

1.3.2 数值

【强制】当数值为 `0 – 1` 之间的小数时，省略整数部分的 `0`。

【示例】

```
/* bad */
panel {
  opacity: 0.8;
}
```

```
/* good */
panel {
  opacity: .8;
}
```

1.3.3 url 设置

【强制】url() 函数中的路径不加引号。

【示例】

```
body {
  background: url(bg.png);
}
```

1.3.4 长度

【强制】长度为 0 时须省略单位。（也只有长度单位可省）。

【示例】

```
/* good */
body {
  padding: 0 5px;
}

/* bad */
body {
  padding: 0px 5px;
}
```

1.3.5 颜色

【强制】RGB 颜色值必须使用十六进制记号形式 #rrggbb。不允许使用 rgb()。

（注：带有 alpha 的颜色信息可以使用 rgba()。使用 rgba() 时每个逗号后必须保留一个空格。）

【强制】颜色值不允许使用命名色值。

【建议】颜色值中的英文字符采用小写。

【示例】

```
/* good */
.success {
  box-shadow: 0 0 2px rgba(0, 128, 0, .3);
  border-color: #008000;
}

/* bad */
.success {
  box-shadow: 0 0 2px rgba(0,128,0,.3);
  border-color: rgb(0, 128, 0);
}

/* bad */
.success {
  color: lightgreen;
}

/* bad */
.success {
  background-color: #ACA;
  color: #90ee90;
}
```

1.4 文本编排

1.4.1 字号

【强制】需要在 Windows 平台显示的中文内容，其字号应不小于 12px。（注：由于 Windows 的字体渲染机制，小于 12px 的文字显示效果极差、难以辨认。）

1.4.2 字重

【强制】font-weight 属性必须使用数值方式描述。（注： CSS 的字重分 100 – 900 共九档，但目前受字体本身质量和浏览器的限制，实际上支持 400 和 700 两档，分别等价于关键词 normal 和 bold。 浏览器本身使用一系列启发式规则来进行匹配，在 <700 时一般匹配字体的 Regular 字重，>=700 时匹配 Bold 字重。但已有浏览器开始支持 =600 时匹配 Semibold 字重（见此表），故使用数值描述增加了灵活性，也更简短。）

【示例】

```
/* good */
h1 {
  font-weight: 700;
}

/* bad */
h1 {
  font-weight: bold;
}
```

1.4.3 行高

【建议】line-height 在定义文本段落时，应使用数值。（注： 将 line-height 设置为数值，浏览器会基于当前元素设置的 font-size 进行再次计算。在不同字号的文本段落组合中，能达到较为舒适的行间间隔效果，避免在每个设置了 font-size 都需要设置 line-height。 当 line-height 用于控制垂直居中时，还是应该设置成与容器高度一致。）

第 2 部分 html 编写规范

2.1 代码风格

2.1.1 缩进，空格

【强制】使用 2 个空格做为一个缩进层级，不允许使用 4 个空格 或 tab 字符。（注：对于非 HTML 标签之间的缩进，比如 script 或 style 标签内容缩进，与 script 或 style 标签的缩进同级。）

【示例】

```
<style>
  /* 样式内容的第一级缩进与所属的 style 标签对齐 */
  ul {
    padding: 0;
  }
</style>
<ul>
  <li>first</li>
  <li>second</li>
</ul>
<script>
// 脚本代码的第一级缩进与所属的 script 标签对齐
require(['app'], function (app) {
  app.init();
});
</script>
```

2.1.2 行长度

【建议】每行不得超过 120 个字符，除非单行不可分割。（注：过长的代码不容易阅读与维护。但是考虑到 HTML 的特殊性，不做硬性要求。）

2.1.3 命名

【强制】class 驼峰命名。

【强制】class 必须代表相应模块或部件的内容或功能，不得以样式信息进行命名。

【强制】元素 id 必须保证页面唯一。(注：同一个页面中，不同的元素包含相同的 id，不符合 id 的属性含义。并且使用 document.getElementById 时可能导致难以追查的问题)

【强制】 禁止为了 hook 脚本，创建无样式信息的 class。(注：不允许 class 只用于让 JavaScript 选择某些元素，class 应该具有明确的语义和样式。否则容易导致 CSS class 泛滥。使用 id、属性选择作为 hook 是更好的方式。)

【示例】

```
<!-- good -->
<div class="sideBar"></div>

<!-- bad -->
<div class="left"></div>
```

2.1.4 标签

【强制】 标签名必须使用小写字母。

【强制】 对于无需自闭合的标签，不允许自闭合。(注：常见无需自闭合标签有 input、br、img、hr 等。)

【强制】 标签使用必须符合标签嵌套规则。(注：比如 div 不得置于 p 中，tbody 必须置于 table 中。)

【建议】 HTML 标签的使用应该遵循标签的语义。

【建议】 标签的使用应尽量简洁，减少不必要的标签。

【示例】

```
<!-- good -->
<p>Hello StyleGuide!</p>

<!-- bad -->
<P>Hello StyleGuide!</P>

<!-- good -->
<input type="text" name="title">

<!-- bad -->
<input type="text" name="title" />
```

2.1.5 属性

【强制】 属性名必须使用小写字母。

【强制】 属性值必须用双引号包围。（注：不允许使用单引号，不允许不使用引号。）

【示例】

```
<!-- good -->
<table cellspacing="0">...</table>

<!-- bad -->
<table cellSpacing="0">...</table>

<!-- bad -->
<script src='esl.js'></script>
<script src=esl.js></script>
```

2.1.6 属性定义

【强制】属性定义必须另起一行。

【强制】属性定义后必须以分号结尾。

【示例】

```
<style>
/* good */
.selector {
  margin: 0;
  padding: 0;
}

/* bad */
.selector { margin: 0; padding: 0; }

/* good */
.selector {
  margin: 0;
}

/* bad */
.selector {
  margin: 0
}
</style>
```

2.2 3.通用

2.2.1 DOCTYPE

【强制】 使用 HTML5 的 doctype 来启用标准模式，建议使用大写的 DOCTYPE。

【建议】 在 html 标签上设置正确的 lang 属性。（注：有助于提高页面的可访问性，如：让语音合成工具确定其所应该采用的发音，令翻译工具确定其翻译语言等。）

【示例】

```
<!DOCTYPE html>
<html lang="zh-CN">
```

2.2.2 编码

【强制】页面必须使用精简形式，明确指定字符编码。指定字符编码的 meta 必须是 head 的第一个直接子元素。

【示例】

```
<html lang="zh-CN">
    <head>
      <meta charset="UTF-8">
      <link rel="stylesheet" href="page.css">
      <title>标题</title>
    </head>
    <body>
      <!-- a lot of elements -->
      <script src="init-behavior.js"></script>
  </body>
</html>
```

2.2.3 CSS 和 JavaScript 引入

【强制】 引入 CSS 时必须指明 rel="stylesheet"。

【建议】 引入 CSS 和 JavaScript 时无须指明 type 属性。(注: text/css 和 text/javascript 是 type 的默认值。)

【建议】 在 head 中引入页面需要的所有 CSS 资源。(注：在页面渲染的过程中，新的 CSS 可能导致元素的样式重新计算和绘制，页面闪烁。)

【建议】 JavaScript 应当放在页面末尾，或采用异步加载。(注：将 script 放在页面中间将阻断页面的渲染。出于性能方面的考虑，如非必要，请遵守此条建议。)

【示例】

```
<html lang="zh-CN">
    <head>
      <meta charset="UTF-8">
      <link rel="stylesheet" href="page.css">
      <title>标题</title>
    </head>
    <body>
      <!-- a lot of elements -->
      <script src="init-behavior.js"></script>
  </body>
</html>
```

2.3 head

2.3.1 网页标题

【强制】页面必须包含 title 标签声明标题。（注：title 中如果包含 ASCII 之外的字符，浏览器需要知道字符编码类型才能进行解码，否则可能导致乱码。）

【示例】

```
<head>
  <title>标题</title>
</head>
```

2.3.2 网站图标

【强制】保证 favicon 可访问。在未指定 favicon 时，大多数浏览器会请求 Web Server 根目录下的 favicon.ico 。为了保证 favicon 可访问，避免 404，必须遵循以下两种方法之一：

1. 在 Web Server 根目录放置 favicon.ico 文件。
2. 使用 link 指定 favicon。

【示例】

```
<head>
  <link rel="shortcut icon" href="path/to/favicon.ico">
</head>
```

2.3.3 viewport

【建议】若页面欲对移动设备友好，需指定页面的 viewport。

viewport meta tag 可以设置可视区域的宽度和初始缩放大小，避免在移动设备上出现页面展示不正常。

比如，在页面宽度小于 980px 时，若需 iOS 设备友好，应当设置 viewport 的 width 值来适应你的页面宽度。同时因为不同移动设备分辨率不同，在设置时，应当使用 device-width 和 device-height 变量。

2.4 图片

【强制】 禁止 img 的 src 取值为空。延迟加载的图片也要增加默认的 src。(注: src 取值为空，会导致部分浏览器重新加载一次当前页面)

【建议】 避免为 img 添加不必要的 title 属性。(注: 多余的 title 影响看图体验，并且增加了页面尺寸。)

【建议】 为重要图片添加 alt 属性。(注: 可以提高图片加载失败时的用户体验。)

【建议】 添加 width 和 height 属性，以避免页面抖动。

【建议】 有下载需求的图片采用 img 标签实现，无下载需求的图片采用 CSS 背景图实现。

2.5 表单

2.5.1 控件标题

【强制】 有文本标题的控件必须使用 label 标签将其与其标题相关联。

【示例】

```
<label><input type="checkbox" name="confirm" value="on"> 我已确认上述条款</label>
```

2.5.2 按钮

【强制】使用 button 元素时必须指明 type 属性值（注：button 元素的默认 type 为 submit，如果被置于 form 元素中，点击后将导致表单提交。为显示区分其作用方便理解，必须给出 type 属性。）

【示例】

```
<button type="submit">提交</button>
<button type="button">取消</button>
```

第 3 部分 javascript 编写规范

3.1 代码风格

3.1.1 文件

【建议】在文件结尾处，保留一个空行。

3.1.2 结构

3.1.2.1 缩进

【强制】使用 2 个空格做为一个缩进层级，不允许使用 4 个空格 或 tab 字符。

【强制】switch 下的 case 和 default 必须增加一个缩进层级。

【示例】

```
// good
switch (variable) {

  case '1':
    // do...
    break;

  case '2':
    // do...
    break;

  default:
    // do...

}
```

```
// bad
switch (variable) {

case '1':
  // do...
  break;

case '2':
  // do...
  break;

default:
  // do...

}
```

3.1.2.2 空格

【强制】二元运算符两侧必须有一个空格，一元运算符与操作对象之间不允许有空格。

【示例】

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

while (condition) {
}

function funcName() {
}
```

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

while (condition){
}

function funcName(){
}
```

【强制】用作代码块起始的左花括号 { 前必须有一个空格。

【示例】

```
// good
if (condition) {
  // ...
}

while (condition) {
  // ...
}

(function () {
})();

// bad
if(condition) {
  // ...
}

while(condition) {
  // ...
}

(function() {
})();
```

【强制】if / else / for / while / function / switch / do / try /

catch / finally 关键字后，必须有一个空格。

【示例】

```
// good
if (condition) {
  // ...
}

while (condition) {
  // ...
}

(function () {
})();

// bad
if(condition) {
  // ...
}

while(condition) {
  // ...
}

(function() {
})();
```

【强制】在对象创建时，属性中的 : 之后必须有空格，: 之前不允许有空格。

【示例】

```
// good
let obj = {
  a: 1,
  b: 2,
  c: 3
};
```

```
// bad
var obj = {
  a : 1,
  b:2,
  c :3
};
```

【强制】函数声明、具名函数表达式、函数调用中，函数名和 (之间不允许有空格。

【示例】

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

var fnName = function fnName() {
};

funcName();
```

```
// bad
function fnName () {
}

var funcName = function funcName () {
};

funcName ();
fnName();
```

【强制】, 和 ; 前不允许有空格。如果不位于行尾，, 和 ; 后必须跟一个空格。

【示例】

```
// good
callFunc(a, b);

// bad
callFunc(a , b) ;
```

【强制】在函数调用、函数声明、括号表达式、属性访问、if / for / while / switch / catch 等语句中，() 和 [] 内紧贴括号部分不允许有空格。

【示例】

```
// good
callFunc(param1, param2, param3);

save(this.list[this.indexes[i]]);

needIncream && (variable += increament);

if (num > list.length) {
  // ...
}

while (len--) {
  // ...
}
```

```
// bad
callFunc( param1, param2, param3 );

save( this.list[ this.indexes[ i ] ] );

needIncreament && ( variable += increament );

if ( num > list.length ) {
  // ...
}

while ( len-- ) {
  // ...
}
```

【强制】单行声明的数组与对象，如果包含元素，{} 和 [] 内紧贴括号部分不允许包含空格

【示例】

```
// good
var arr1 = [];
var arr2 = [1, 2, 3];
var obj1 = {};
var obj2 = {name: 'obj'};
var obj3 = {
  name: 'obj',
  age: 20,
  sex: 1
};

// bad
var arr1 = [ ];
var arr2 = [ 1, 2, 3 ];
var obj1 = { };
var obj2 = { name: 'obj' };
var obj3 = {name: 'obj', age: 20, sex: 1};
```

【强制】行尾不得有多余的空格。

3.1.2.3 换行

【强制】每个独立语句结束后必须换行。

【强制】每行不得超过 120 个字符。(注： 超长的不可分割的代码允许例外，比如复杂的正则表达式。长字符串不在例外之列。)

【强制】运算符处换行时，运算符必须在新行的行首。

【示例】

```
// good
if (user.isAuthenticated()
    && user.isInRole('admin')
    && user.hasAuthority('add-admin')
    || user.hasAuthority('delete-admin')
) {
    // Code
}

var result = number1 + number2 + number3
    + number4 + number5;


// bad
if (user.isAuthenticated() &&
    user.isInRole('admin') &&
    user.hasAuthority('add-admin') ||
    user.hasAuthority('delete-admin')) {
    // Code
}

var result1 = number1 + number2 + number3 +
    number4 + number5;
```

【强制】在函数声明、函数表达式、函数调用、对象创建、数组创建、for 语句等场景中，不允许在 ，或 ； 前换行。

【示例】

```
// good
var obj = {
  a: 1,
  b: 2,
  c: 3
};

foo(
  aVeryVeryLongArgument,
  anotherVeryLongArgument,
  callback
);
```

```
// bad
var obj = {
  a: 1
  , b: 2
  , c: 3
};

foo(
  aVeryVeryLongArgument
  , anotherVeryLongArgument
  , callback
);
```

【建议】不同行为或逻辑的语句集，使用空行隔开，更易阅读。

【示例】

```
// 仅为按逻辑换行的示例，不代表setStyle的最优实现
function setStyle(element, property, value) {
  if (element == null) {
    return;
  }

  element.style[property] = value;
}
```

【建议】在语句的行长度超过 120 时，根据逻辑条件合理缩进。

【建议】对于 if...else...、try...catch...finally 等语句,推荐使用在 } 号后添加一个换行 的风格，使代码层次结构更清晰，阅读性更好。

【示例】

```
if (condition) {
  // some statements;
}
else {
  // some statements;
}
```

```
try {
  // some statements;
}
catch (ex) {
  // some statements;
}
```

3.1.2.4 语句

【强制】不得省略语句结束的分号。

【强制】在 if / else / for / do / while 语句中，即使只有一行，也不得省略块 {...}。

【示例】

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

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

【强制】函数定义结束不允许添加分号。

【示例】

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

```
// bad
function funcName() {
  // ...
}

// 如果是函数表达式，分号是不允许省略的。
var funcName1 = function () {
}
```

3.1.3 命名

【强制】变量 使用驼峰命名法。

【强制】常量 使用 全部字母大写，单词间下划线分隔 的命名方式。

【强制】函数 使用驼峰命名法。

【强制】函数的参数 使用驼峰命名法。

【示例】

```
// 变量
var loadingModules = {};
// 常量
var HTML_ENTITY = 333;
// 函数
function stringFormat() {
}
// 函数参数
function hear(theBells) {
  console.log(theBells)
  // ...
}
```

【强制】枚举变量 使用 Pascal 命名法，枚举的属性 使用 全部字母大写，单词间下划线分隔的命名方式。

【建议】函数名 使用 动宾短语。

【建议】boolean 类型的变量使用 is 或 has 开头。

【示例】

```
// 枚举变量命名
var TargetState = {
  READING: 1,
  READED: 2,
  APPLIED: 3,
  READY: 4
};
// 函数命名
function getStyle() {
}

// boolean类型命名
var isReady = false;
var hasMoreCommands = false;
```

3.1.4 注释

3.1.4.1 单行注释

【强制】必须独占一行。// 后跟一个空格，缩进与下一行被注释说明的代码一致。

3.1.4.2 多行注释

【建议】避免使用 / * ... */ 这样的多行注释。有多行注释内容时，使用多个单行注释。

3.1.4.3 文档化注释

【强制】为了便于代码阅读和自文档化，以下内容必须包含以 /**...*/ 形式的块注释中。

【强制】文档注释前必须空一行。

1. 文件
2. namespace
3. 类
4. 函数或方法
5. 类属性
6. 事件
7. 全局变量
8. 常量
9. AMD 模块

3.1.4.4 文件注释

【强制】文件顶部必须包含文件注释，用 @file 标识文件说明。

【建议】 文件注释中可以用 @author 标识开发者信息。

【示例】

```
/**
 * @file Describe the file
 * @author author-name(mail-name@domain.com)
 *         author-name2(mail-name2@domain.com)
 */
```

3.1.4.5 函数/方法注释

【强制】函数/方法注释必须包含函数说明，有参数和返回值时必须使用注释标识。

【强制】参数和返回值注释必须包含类型信息，且不允许省略参数的说明。

【示例】

```
/**
 * 函数描述
 *
 * @param {string} p1 参数1的说明
 * @param {string} p2 参数2的说明，比较长
 *     那就换行了.
 * @param {number=} p3 参数3的说明（可选）
 * @return {Object} 返回值描述
 */
function foo(p1, p2, p3) {
    var p4 = p3 || 10;
    return {
        p1: p1,
        p2: p2,
        p4: p4,
    };
}
```

3.1.4.6 细节注释

【强制】有时我们会使用一些特殊标记进行说明。特殊标记必须使用单行注释的形式。下面列举了一些常用标记：

1. TODO：有功能待实现。此时需要对将要实现的功能进行简单说明。

2. FIXME：该处代码运行没问题，但可能由于时间赶或者其他原因，需要修正。此时需要对如何修正进行简单说明。

3. HACK：为修正某些问题而写的不太好或者使用了某些诡异手段的代码。此时需要对思路或诡异手段进行描述。

4. XXX：该处存在陷阱。此时需要对陷阱进行描述。

3.2 语言特性

3.2.1 变量

【强制】变量、函数在使用前必须先定义。（注： 不通过 var 定义变量将导致变量污染全局环境。原则上不建议使用全局变量，）

【示例】

```
// good
var name = 'MyName';

// bad
name = 'MyName';
```

【强制】每个 var 只能声明一个变量。（注：一个 var 声明多个变量，容易导致较长的行长度，并且在修改时容易造成逗号和分号的混淆。）

【示例】

```
// good
var hangModules = [];
var missModules = [];
var visited = {};

// bad
var hangModules1 = [],
    missModules1 = [],
    visited1 = {};
```

【强制】变量必须 即用即声明，不得在函数或其它形式的代码块起始位置统一声明所有变量。（注： 变量声明与使用的距离越远，出现的跨度越大，代码的阅读与维护成本越高。虽然 JavaScript 的变量是函数作用域，还是应该根据编程中的意图，缩小变量出现的距离空间。）

【示例】

```
// good
function kv2List(source) {
  var list = [];

  for (var key in source) {
    if (source.hasOwnProperty(key)) {
      var item = {
        k: key,
        v: source[key]
      };

      list.push(item);
    }
  }

  return list;
}

// bad
function kv2List(source) {
  var list = [];
  var key;
  var item;

  for (key in source) {
    if (source.hasOwnProperty(key)) {
      item = {
        k: key,
        v: source[key]
      };

      list.push(item);
    }
  }

  return list;
}
```

3.2.2 条件

【强制】在 Equality Expression 中使用类型严格的 ===。仅当判断 null 或 undefined 时，允许使用 == null。（注： 使用 === 可以避免等于判断中隐式的类型转换。）

【示例】

```
// good
if (age === 30) {
  // ......
}

// bad
if (age == 30) {
  // ......
}
```

【建议】尽可能使用简洁的表达式。

【示例】

```
// 字符串为空

// good
if (!name) {
  // ......
}

// bad
if (name === '') {
  // ......
}
```

```
// 字符串非空

// good
if (name) {
  // ......
}

// bad
if (name !== '') {
  // ......
}
```

```
// 数组非空

// good
if (collection.length) {
  // ......
}

// bad
if (collection.length > 0) {
  // ......
}
```

```
// 布尔不成立

// good
if (!notTrue) {
  // ......
}

// bad
if (notTrue === false) {
  // ......
}
```

```
// null 或 undefined

// good
if (noValue == null) {
  // ......
}

// bad
if (noValue === null || typeof noValue === 'undefined') {
  // ......
}
```

【建议】对于相同变量或表达式的多值条件，用 switch 代替 if。

【示例】

```
// good
switch (typeof variable) {
  case 'object':
    // ......
    break;
  case 'number':
  case 'boolean':
  case 'string':
    // ......
    break;
}

// bad
var type = typeof variable;
if (type === 'object') {
    // ......
}
else if (type === 'number' || type === 'boolean' || type === 'string') {
    // ......
}
```

【建议】如果函数或全局中的 else 块后没有任何语句，可以删除 else。

【示例】

```
// good
function getName() {
  if (name) {
    return name;
  }

  return 'unnamed';
}

// bad
function getName() {
  if (name) {
    return name;
  }
  else {
    return 'unnamed';
  }
}
```

3.3 循环

【建议】不要在循环体中包含函数表达式，事先将函数提取到循环体外。（注： 循环体中的函数表达式，运行过程中会生成循环次数个函数对象。）

【示例】

```
// good
function clicker() {
  // ......
}

for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  addListener(element, 'click', clicker);
}
```

```
// bad
for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  addListener(element, 'click', function () {});
}
```

【建议】对循环内多次使用的不变值，在循环外用变量缓存。

【示例】

```
// good
var width = wrap.offsetWidth + 'px';
for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  element.style.width = width;
  // ......
}


// bad
for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  element.style.width = wrap.offsetWidth + 'px';
  // ......
}
```

【建议】对有序集合进行遍历时，缓存 length。（注： 虽然现代浏览器都对数组长度进行了缓存，但对于一些宿主对象和老旧浏览器的数组对象，在每次 length 访问时会动态计算元素个数，此时缓存 length 能有效提高程序性能。）

【示例】

```
// good
for (var i = 0, len = elements.length; i < len; i++) {
  var element = elements[i];
  // ......
}
```

3.4 类型

3.4.1 类型检测

【建议】类型检测优先使用 typeof。对象类型检测使用 instanceof。null 或 undefined 的检测使用 == null。

【示例】

```
// string
typeof variable === 'string'

// number
typeof variable === 'number'

// boolean
typeof variable === 'boolean'

// Function
typeof variable === 'function'

// Object
typeof variable === 'object'

// RegExp
variable instanceof RegExp

// Array
variable instanceof Array

// null
variable === null

// null or undefined
variable == null

// undefined
typeof variable === 'undefined'
```

3.4.2 类型转换

【建议】转换成 string 时，使用 + ''。

【示例】

```
// good
num + '';

// bad
new String(num);
num.toString();
String(num);
```

【建议】转换成 number 时，通常使用 +。

【示例】

```
// good
+str;

// bad
Number(str);
```

【建议】string 转换成 number，要转换的字符串结尾包含非数字并期望忽略时，使用 parseInt。

【示例】

```
var width = '200px';
parseInt(width, 10);
```

【强制】使用 parseInt 时，必须指定进制。

【示例】

```
// good
parseInt(str, 10);

// bad
parseInt(str);
```

3.5 字符串

【强制】字符串开头和结束使用单引号 '。(注： 实际使用中，字符串经常用来拼接 HTML。为方便 HTML 中包含双引号而不需要转义写法。)

【示例】

```
var str = '我是一个字符串';
var html = '<div class="cls">拼接HTML可以省去双引号转义</div>';
```

3.6 对象

【强制】使用对象字面量 {} 创建新 Object。

【示例】

```
// good
var obj = {};

// bad
var obj = new Object();
```

【建议】for in 遍历对象时，使用 hasOwnProperty 过滤掉原型中的属性。

【示例】

```
var newInfo = {}
for (var key in info) {
  if(info.hasOwnProperty(key)){
    newInfo[key] = info[key];
  }
}
```

3.7 数组

【强制】使用数组字面量 [] 创建新数组，除非想要创建的是指定长度的数组。

【示例】

```
// good
var arr = [];

// bad
var arr = new Array();
```

【强制】遍历数组不使用 for in（注：数组对象可能存在数字以外的属性，这种情况下 for in 不会得到正确结果。）

【示例】

```
var arr = ['a', 'b', 'c'];

// 这里仅作演示，实际中应使用 Object 类型
arr.other = 'other things';

// 正确的遍历方式
for (var i = 0, len = arr.length; i < len; i++) {
  console.log(i);
}

// 错误的遍历方式
for (var i in arr) {
  console.log(i);
}
```

3.8 函数

3.8.1 函数长度

【建议】一个函数的长度控制在 50 行以内。（注：将过多的逻辑单元混在一个大函数中，易导致难以维护。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抽取，通过函数的调用来体现流程。）

3.8.2 参数设计

【建议】一个函数的参数控制在 6 个以内，过多参数会导致维护难度增大。

【建议】 通过 options 参数传递非数据输入型参数。

附录：文档变更信息

文档基本信息					
文档编号	协同技术规范-前端编码规范-yyyymmrr（发布日期）				
文档密级	NB	发布日期			
文档变更记录					
版本号	**变更章节、内容说明**	**变更日期**	**编制人**	**审核人**	**批准人**
V1.0	初版	2020-03-04			