前端编码规范

本文档定义了编码规范的整体结构，每个具体的项目均有单独的分册。可以点击查看。

> *规范部分采用了RFC2119 规范定义的动词。采用了该标准的文档中，使用的关键字会以 中文 + 括号 包含的关键字英文表示，如 必须 (MUST) 。*

规范列表

- 命名规范
- 内容格式与编码规范
- 项目目录结构规范
- HTML编码规范
- CSS编码规范
- JS编码规范-ES5
- JS编码规范-ESNEXT

refs

- RFC2119

命名规范

简介

该文档主要的设计目标是公司前端项目文件（夹）命名规范化。

编撰

@IFLY_FE

目录

- 简介
- 编撰
- 目录
- 规范详情
  - 编码范围
  - 项目命名
  - 目录命名
  - 文件命名
  - 特例
- Notes
- Refs

规范详情

编码范围

[MUST] 文件（夹）的命名均不允许包含空格。

[MUST] 使用半角符号，也即不允许使用中文命名。

项目命名

[MUST] 全部采用小写形式，以下划线分隔。

示例：`my_project_name`、`x_client`。

目录命名

[MUST] 全部采用小写形式，以下划线分隔。

[MUST] 有复数结构时，采用复数命名法。

[MUST] 简洁且有习惯性缩写的单词必须采用容易理解的缩写。

示例：`pdf_previewer`、`node_modules`、`broker_adapters`、`scripts`、`src`。

文件命名

[MUST] 全部采用小写形式。

[MUST] 以中划线分隔。

解释：为什么要小写。

示例：`html-parser.js`、`error-detector.js`、`index.css`、`logo-cropped.png` 。

特例

[RECOMMENDED] 为了醒目，某些说明文件的文件名，允许使用大写字母，且此时必须使用下划线分隔。

示例：`README.MD`、`LICENSE`、`CODE_OF_CONDUCT.MD`。

NOTES

[RECOMMENDED] 基于遗留项目时，如果项目的命名风格和本规范风格不一致，但是命名清晰，或者有时间、人员等原因无法进行重构时，允许遵循已有风格即可。但是团队风格必须保持一致。

Refs

- **AlloyTeam命名规范**
- **ecomfe spec**
- **yfruan写作规范**
- **soft tab**
- **sideeffect.kr**

内容格式与编码规范

文件编码

[MUST] 所有文件，包括 `.html`、`.css`、 `.js` 文件，全部使用 无BOM头 的 `UTF-8` 编码。

缩进

[MUST] 使用soft-tab。

[MUST] 项目中所有源码使用同样的缩进，使用 `4` 或者 `2` 个 spaces 视项目成员而定，但是必须都保持一致，推荐使用2个spaces。

项目目录结构规范

简介

该文档主要的设计目标是公司前端项目目录结构规范，使容易理解并方便构建与管理。

编撰

@IFLY_FE

目录

- 简介
- 编撰
- 目录
- 规范详情
  - 范围
  - 使用框架
  - 不使用框架
- Refs

约定

本规范中，`xms` 代表根目录。

规范详情

范围

本规范将项目划分为以下两种：

- 业务较少、简单的项目
- 业务多、复杂的项目

基于以上两种项目，本规范规定根据是否使用框架定义以下几种场景中项目目录结构：

- 不使用框架
  - 业务较少、简单的项目
  - 业务多、复杂的项目
- 使用框架

使用框架

[MUST] 使用框架时，按照框架推荐的结构进行划分。

[MUST] 在进行 `components` 等类似文件夹的具体划分时，简单项目直接不继续划分。

[MUST] 在进行 `components` 等类似文件夹的具体划分时，复杂项目需要按照业务来进行划分，划分为 `common` 以及其它特定业务文件夹。

[MUST] 功能模块单元的资源应该内聚。但是业务特定组件的共用资源，应该置于资源文件夹中的 `common` 内。

下面是一个示例：

```
xms
  |- build
  |- config
  |- dist
  |- node_modules
  |- src
    |- assets
      |- logo.png
      |- common
        |- add-btn-prompt.png
    |- components
      |- common
        |- previewer
          |- index.vue
          |- index.css
          |- images
            |- enlarge.png
      |- profile
        |- story
          |- index.vue
          |- index.css
          |- images
            |- send.png
    |- App.vue
    |- main.js
  |- static
  |- test
  |- .bablerc
  |- .eslintrc.js
  |- index.html
  |- package.json
  |- README.md
```

不使用框架

简单项目

根目录划分

[MUST] 在根目录下，目录结构按照 职能 进行划分。

常用的目录有 `src`、`node_modules`、`docs`、`examples`、`test`、`dist`、`benchmarks` 等。

> *这一点其实无论在使用或者不使用框架时，均是这种划分。但是为了防止某些特殊框架的特殊行为，就在该章节定义。*

`src` 目录划分

[RECOMMENDED] 简单项目中，允许将 `src` 直接包含 资源文件，也即 **按照资源类型** 来进行划分目录。

`src` 目录下常见目录有 `scripts`、`templates`、`styles`、`images` 等。

下面是一个简单的例子：

```
xms
  |- src
    |- scripts
      |- util.js
      |- format.js
    |- templates
      |- article.html
      |- profile.html
    |- styles
      |- normalize.css
      |- base.css
      |- index.css
    |- images
      |- add-btn.png
    |- static
      |- logo.ico
    |- index.html
    |- main.js
  |- dist
```

复杂项目

根目录划分

[MUST] 按照简单项目的根目录划分原则。

src 目录划分

[MUST] src 下只包含 业务目录 与 common 目录。

[MUST] 业务公共资源 命名为 common。common 目录作为 业务公共资源 的目录，也视如 业务目录。

下面是一个简单的例子：

```
xms
  |- src
    |- common
    |- biz1
      |- sub
    |- biz2
```

业务目录划分原则

[RECOMMENDED] scripts资源 不按 资源类型 划分目录，scripts资源 直接置于 业务目录 下。即：业务目录 下不允许出现 scripts 目录。

[RECOMMENDED] 除 scripts资源 外的 源文件资源，当资源数量较多时，为方便管理，按 资源类型 划分目录。即：业务目录 下允许出现 styles、tpl、swf、fonts 等目录。

[MUST] 对于一个 业务目录，将业务相关的 源文件资源 都直接置于 业务目录 下。但是业务的共用资源，应该置于 common 内。如：

```
biz1
  |- iamges
    |- add-button.png
  |- index.js
  |- index.css
  |- tpl.html
```

[RECOMMENDED] 业务目录 下 源文件资源 数量较多时，需要考虑是否细致的划分了业务。也即是否应该划分子业务，建立子业务目录。如：

```
biz2
  |- subbiz1
    |- index.js
    |- index.css
    |- tpl.html
  |- subbiz2
```

[RECOMMENDED] 如果确实是一个业务整体，无法划分子业务，将非 `scripts`资源 按 资源类型 划分目录进行管理。

```
biz1
  |- styles
    |- add.css
    |- edit.css
    |- remove.css
  |- images
    |- add.png
  |- templates
    |- add.html
    |- edit.html
    |- remove.html
  |- add.js
  |- edit.js
  |- remove.js
```

划分示例

```
xms
  |- src
    |- common
      |- images
        |- sprites.png
        |- logo.png
        |- conf.js
        |- layout.css
    |- biz1
      |- images
      |- add.png
      |- index.js
      |- tpl.html
      |- index.css
    |- biz2
      |- subbiz1
        |- index.js
        |- index.css
        |- tpl.html
      |- subbiz2
  |- deps
    |- eventer
      |- src
      |- test
    |- inherit
      |- src
      |- test
  |- test
  |- docs
  |- index.html
  |- main.js
  ......
```

Refs

- ecomfe
- tools

HTML编码规范

简介

HTML 作为描述网页结构的超文本标记语言，有着广泛的应用。本文档的目标是使 HTML 代码风格保持一致，容易被理解和被维护。

编撰

@IFLY_FE

目录

- 简介
- 编撰
- 目录
- 规范详情
  - 内容格式与编码规范
  - 缩进
  - 换行
  - 命名
  - 标签
  - 属性
  - DOCTYPE
  - compatible
  - lang
  - CSS 和 JavaScript 引入
  - title
  - favicon
  - 图片
  - form
  - A11Y
  - 多媒体
- Refs

规范详情

内容格式与编码规范

严格依照format。

缩进

[RECOMMENDED] 使用 2 个空格作为一个缩进层级。

备注：对于 `script` 或 `style` 等标签内容中的缩进保持一致。

示例：

```
<style>
  ul {
    padding: 0;
  }
</style>
<ul>
  <li>first</li>
  <li>second</li>
</ul>
<script>
  require([ 'app' ], function (app) {
    app.init();
  });
</script>
```

换行

[MUST] 每行不得超过 120 个字符。

解释：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。

命名

[MUST] class 必须单词全字母小写，单词间以 - 分隔。

[MUST] 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。

示例：

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

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

[MUST] 元素 id 必须保证页面唯一。

解释：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。

[MUST] id 单词全字母小写，单词间以 _ 分隔。同项目必须保持风格一致。

[RECOMMENDED] id、class 命名，在避免冲突并描述清楚的前提下尽可能短。

示例：

```
<!-- good -->
<div id="nav"></div>
<!-- bad -->
<div id="navigation"></div>

<!-- good -->
<p class="comment"></p>
<!-- bad -->
<p class="com"></p>

<!-- good -->
<span class="author"></span>
<!-- bad -->
<span class="red"></span>
```

[MUST] 禁止为了 hook 脚本，创建无样式信息的 class。

解释：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。

[MUST] 同一页面，禁止使用相同的 name 与 id。

解释：IE 浏览器会混淆元素的 id 和 name 属性，document.getElementById 可能获得不期望的元素。所以在对元素的 id 与 name 属性的命名需要非常小心。

一个比较好的实践是，为 id 和 name 使用不同的命名法。

示例：

```
<input name="foo">
<div id="foo"></div>
<script>
  // IE6 将显示 INPUT
  alert(document.getElementById('foo').tagName);
</script>
```

标签

[MUST] 标签名必须使用小写字母。

示例：

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

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

[MUST] 对于无需自闭合的标签，不允许自闭合。

解释：常见无需自闭合标签有 input、br、img、hr 等。

示例：

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

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

[MUST] 对 HTML5 中规定允许省略的闭合标签，不允许省略闭合标签。

解释：对代码体积要求非常严苛的场景，可以例外。

示例：

```
<!-- good -->
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<!-- bad -->
<ul>
  <li>first
  <li>second
</ul>
```

[MUST] 标签使用必须符合标签嵌套规则。

解释：比如 div 不得置于 p 中，tbody 必须置于 table 中。

详细的标签嵌套规则参见HTML DTD中的Elements定义部分。

[建议] HTML 标签的使用应该遵循标签的语义。

解释：下面是常见标签语义

- p - 段落
- h1、h2、h3、h4、h5、h6 - 层级标题
- strong、em - 强调
- ins - 插入
- del - 删除
- abbr - 缩写
- code - 代码标识
- cite - 引述来源作品的标题
- q - 引用
- blockquote - 一段或长篇引用
- ul - 无序列表
- ol - 有序列表
- dl、dt、dd - 定义列表

示例：

```
<!-- good -->
<p>Esprima serves as an important <strong>building block</strong> for some JavaS

<!-- bad -->
<div>Esprima serves as an important <span class="strong">building block</span> f
```

[RECOMMENDED] 在 css 可以实现相同需求的情况下不得使用表格进行布局。

解释：在兼容性允许的情况下应尽量保持语义正确性。对网格对齐和拉伸性有严格要求的场景允许例外，如多列复杂表单。

[RECOMMENDED] 标签的使用应尽量简洁，减少不必要的标签。

示例：

```
<!-- good -->
<img class="avatar" src="image.png">

<!-- bad -->
<span class="avatar">
  <img src="image.png">
</span>
```

属性

[MUST] 属性名必须使用小写字母。

示例：

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

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

[MUST] 属性值必须用双引号包围。

解释：不允许使用单引号，不允许不使用引号。

示例：

```
<!-- good -->
<script src="app.js"></script>

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

[RECOMMENDED] 布尔类型的属性，建议不添加属性值。

示例：

```
<input type="text" disabled>
<input type="checkbox" value="1" checked>
```

[MUST] 自定义属性以 data- 为前缀。

解释：使用前缀有助于区分自定义属性和标准定义的属性。

示例：

```
<ol data-ui-type="Select"></ol>
```

DOCTYPE

[MUST] 使用 HTML5 的 doctype 来启用标准模式，建议使用大写的 DOCTYPE。

示例：

```
<!DOCTYPE html>
```

compatible

[RECOMMENDED] 启用IE Edge模式。

示例：

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

lang

[RECOMMENDED] 在 html 标签上设置正确的 lang 属性。

解释：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：

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

编码格式 charset

[MUST] 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。

解释：见HTML5 Charset能用吗 一文。

示例：

```
<html>
  <head>
    <meta charset="UTF-8">
    ......
  </head>
  <body>
    ......
  </body>
</html>
```

CSS 和 JavaScript 引入

[MUST] 引入 CSS 时必须指明 rel="stylesheet"。

示例：

```
<link rel="stylesheet" href="page.css">
```

[RECOMMENDED] 引入 CSS 和 JavaScript 时无须指明 type 属性。

解释：text/css 和 text/javascript 是 type 的默认值。

[RECOMMENDED] 展现定义放置于外部 CSS 中，行为定义放置于外部 JavaScript 中。

解释：结构-样式-行为的代码分离，对于提高代码的可阅读性和维护性都有好处。

[RECOMMENDED] 在 head 中引入页面需要的所有 CSS 资源。

解释：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。

[RECOMMENDED] JavaScript 应当放在页面末尾，或采用异步加载。

解释：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：

```
<body>
  <!-- a lot of elements -->
  <script src="init.js"></script>
</body>
```

[RECOMMENDED] 移动环境或只针对现代浏览器设计的 Web 应用，如果引用外部资源的 URL 协议部分与页面相同，建议省略协议前缀。

解释：使用 protocol-relative URL 引入 CSS，在 IE7/8 下，会发两次请求。是否使用 protocol-relative URL 应充分考虑页面针对的环境。

示例：

```
<script src="//x.y.com/static/jquery/jquery-1.10.2.min.js"></script>
```

title

[MUST] 页面必须包含 title 标签声明标题。

[MUST] title 必须作为 head 的直接子元素，并紧随 charset 声明之后。

解释：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>
  <meta charset="UTF-8">
  <title>页面标题</title>
</head>
```

favicon

[MUST] 保证 favicon 可访问。

解释：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：

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

图片

[MUST] 禁止 img 的 src 取值为空。延迟加载的图片也要增加默认的 src 。

解释：src 取值为空，会导致部分浏览器重新加载一次当前页面，参考yahoo

[MUST] 禁止为 img 添加不必要的 title 属性。

解释：多余的 title 影响看图体验，并且增加了页面尺寸。

[RECOMMENDED] 为重要图片添加 alt 属性。

解释：可以提高图片加载失败时的用户体验。

[RECOMMENDED] 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。

解释：

1. 产品 logo、用户头像、用户产生的图片等有潜在下载需求的图片，以 img 形式实现，能方便用户下载。
2. 无下载需求的图片，比如：icon、背景、代码使用的图片等，尽可能采用 CSS 背景图实现。

表单

控件标题

[MUST] 有文本标题的控件必须使用 label 标签将其与其标题相关联。

解释：有两种方式：

1. 将控件置于 label 内。
2. label 的 for 属性指向控件的 id。

推荐使用第一种，减少不必要的 id。如果DOM结构不允许直接嵌套，则应使用第二种。

示例：

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

<label for="username">用户名：</label> <input type="textbox" name="username" id="
```

按钮

[MUST] 使用 button 元素时必须指明 type 属性值。

解释：

button 元素的默认 type 可能为 submit （随浏览器），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>
```

[RECOMMENDED] 尽量不要使用按钮类元素的 name 属性。

解释：由于浏览器兼容性问题，使用按钮的 name 属性会带来许多难以发现的问题。具体情况可参考此文。

可访问性 (A11Y)

[RECOMMENDED] 负责主要功能的按钮在 DOM 中的顺序应靠前。

解释：负责主要功能的按钮应相对靠前，以提高可访问性。如果在 CSS 中指定了 float: right 则可能导致视觉上主按钮在前，而 DOM 中主按钮靠后的情况。

示例：

```
<!-- good -->
<style>
.buttons .button-group {
  float: right;
}
</style>

<div class="buttons">
  <div class="button-group">
    <button type="submit">提交</button>
    <button type="button">取消</button>
  </div>
</div>

<!-- bad -->
<style>
.buttons button {
  float: right;
}
</style>

<div class="buttons">
  <button type="button">取消</button>
  <button type="submit">提交</button>
</div>
```

[RECOMMENDED] 当使用JavaScript进行表单提交时，如果条件允许，应使原生提交功能正常工作。

解释：当浏览器 JavaScript 运行错误或关闭 JavaScript 时，提交功能将无法工作。如果正确指定了 form 元素的 action 属性和表单控件的 name 属性时，提交仍可继续进行。

示例：

```
<form action="/login" method="post">
  <p><input name="username" type="text" placeholder="用户名"></p>
  <p><input name="password" type="password" placeholder="密码"></p>
</form>
```

[RECOMMENDED] 在针对移动设备开发的页面时，根据内容类型指定输入框的 type 属性。

解释：根据内容类型指定输入框类型，能获得能友好的输入体验。

示例：

```
<input type="date">
```

多媒体

[RECOMMENDED] 当在现代浏览器中使用 audio 以及 video 标签来播放音频、视频时，应当注意格式。

解释：音频应尽可能覆盖到如下格式：

- MP3
- WAV
- Ogg

视频应尽可能覆盖到如下格式：

- MP4
- WebM
- Ogg

[RECOMMENDED] 在支持 HTML5 的浏览器中优先使用 audio 和 video 标签来定义音视频元素。

[RECOMMENDED] 使用退化到插件的方式来对多浏览器进行支持。

示例：

```
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">
  <source src="audio.ogg" type="audio/ogg">
  <object width="100" height="50" data="audio.mp3">
    <embed width="100" height="50" src="audio.swf">
  </object>
</audio>

<video width="100" height="50" controls>
  <source src="video.mp4" type="video/mp4">
  <source src="video.ogg" type="video/ogg">
  <object width="100" height="50" data="video.mp4">
    <embed width="100" height="50" src="video.swf">
  </object>
</video>
```

[RECOMMENDED] 只在必要的时候开启音视频的自动播放。

[RECOMMENDED] 在 object 标签内部提供指示浏览器不支持该标签的说明。

示例：

```
<object width="100" height="50" data="something.swf">DO NOT SUPPORT THIS TAG</ob
```

Refs

CSS编码规范

简介

CSS 作为网页样式的描述语言，有着广泛的应用。本文档的目标是使 CSS 代码风格保持一致，容易被理解和被维护。

虽然本文档是针对 CSS 设计的，但是在使用各种 CSS 的预编译器(如 less、sass、stylus 等)时，适用的部分也应尽量遵循本文档的约定。

编撰

@IFLY_FE

目录

- 简介
- 编撰
- 目录
- 规范详情
  - 内容格式与编码规范
  - 空格
  - 行长度
  - 选择器
  - 属性
  - 清除浮动
  - !important
  - z-index
  - 值与单位
  - 文本编排
  - 变换与动画
  - 响应式
  - 兼容性
  - Expression
- Refs

规范详情

内容格式与编码规范

严格依照format。

空格

[MUST] 选择器 与 { 之间必须包含空格。

示例：

```
.selector {

}
```

[MUST] 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。

示例：

```
.selector {
  margin: 0;
}
```

[MUST] 列表型属性值 书写在单行时，, 后必须跟一个空格。

示例：

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

行长度

[MUST] 每行不得超过 120 个字符，除非单行不可分割。

解释：常见不可分割的场景为URL超长。

[RECOMMENDED] 对于超长的样式，在样式值的 空格 处或 , 后换行，建议按逻辑分组。

示例：

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

  /* 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))
  );
}
```

选择器

[MUST] 当一条规则包含多个选择器时，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;
}
```

[MUST] >、+、~ 选择器的两边各保留一个空格。

示例：

```
/* 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;
}
```

[MUST] 属性选择器中的值必须用双引号包围。

解释：不允许使用单引号，不允许不使用引号。

示例：

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

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

[MUST] 如无必要，不得为id、class选择器添加类型选择器进行限定。

解释：在性能和维护性上，都有一定的影响。

示例：

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

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

[RECOMMENDED] 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 */
#username input {}
.comment .avatar {}

/* bad */
.page .header .login #username input {}
.comment div * {}
```

属性

[MUST] 属性定义必须另起一行。

示例：

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

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

[MUST] 属性定义后必须以分号结尾。

示例：

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

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

属性

属性定义独占一行

[RECOMMENDED] 在可以使用缩写的情况下，尽量使用属性缩写。

示例：

```
/* good */
.post {
  font: 12px/1.5 arial, sans-serif;
}

/* bad */
.post {
  font-family: arial, sans-serif;
  font-size: 12px;
  line-height: 1.5;
}
```

[RECOMMENDED] 使用 `border` / `margin` / `padding`等缩写时，应注意隐含值对实际数值的影响，确实需要设置多个方向的值时才使用缩写。

解释：`border` / `margin` / `padding` 等缩写会同时设置多个属性的值，容易覆盖不需要覆盖的设定。如某些方向需要继承其他声明的值，则应该分开设置。

示例：

```
article {
  margin: 5px;
  border: 1px solid #999;
}

/* good */
.page {
  margin-right: auto;
  margin-left: auto;
}

.featured {
  border-color: #69c;
}

/* bad */
.page {
  margin: 5px auto;
}

.featured {
  border: 1px solid #69c;
}
```

属性书写顺序

[RECOMMENDED] 同一规则集下的属性在书写时，应按功能进行分组，并以 **Formatting Model（布局方式、位置） > Box Model（尺寸） > Typographic（文本相关） > Visual（视觉效果）** 的顺序书写，以提高代码的可读性。

解释：

- Formatting Model 相关属性包括：`position` / `top` / `right` / `bottom` / `left` / `float` / `display` / `overflow` 等
- Box Model 相关属性包括：`border` / `margin` / `padding` / `width` / `height` 等
- Typographic 相关属性包括：`font` / `line-height` / `text-align` / `word-wrap` 等
- Visual 相关属性包括：`background` / `color` / `transition` / `list-style` 等

另外，如果包含`content`属性，应放在最前面。

示例：

```
.sidebar {
  /* formatting model: positioning schemes / offsets / z-indexes / display / ...
  position: absolute;
  top: 50px;
  left: 0;
  overflow-x: hidden;

  /* box model: sizes / margins / paddings / borders / ...  */
  width: 200px;
  padding: 5px;
  border: 1px solid #ddd;

  /* typographic: font / aligns / text styles / ... */
  font-size: 14px;
  line-height: 20px;

  /* visual: colors / shadows / gradients / ... */
  background: #f5f5f5;
  color: #333;
  -webkit-transition: color 1s;
     -moz-transition: color 1s;
          transition: color 1s;
}
```

清除浮动

[RECOMMENDED] 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。

解释：

触发 BFC 的方式很多，常见的有：

- float 非 none
- position 非 static
- overflow 非 visible

如希望使用更小副作用的清除浮动方法，参见A new micro clearfix hack 一文。

另需注意，对已经触发 BFC 的元素不需要再进行 clearfix。

!important

[RECOMMENDED] 尽量不使用 `!important` 声明。

[RECOMMENDED] 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。

解释：必须注意的是，仅在设计上 确实不允许任何其它场景覆盖样式 时，才使用内联的`!important`样式。通常在第三方环境的应用中使用这种方案。下面的 `z-index` 章节是其中一个特殊场景的典型样例。

z-index

[RECOMMENDED] 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。

解释：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`。

建议每层包含100个 `z-index` 来容纳足够的元素，如果每层元素较多，可以调整这个数值。

[RECOMMENDED] 在可控环境下，期望显示在最上层的元素，`z-index` 指定为 `999999`。

解释：可控环境分成两种，一种是自身产品线环境；还有一种是可能会被其他产品线引用，但是不会被外部第三方的产品引用。

不建议取值为 `2147483647`。以便于自身产品线被其他产品线引用时，当遇到层级覆盖冲突的情况，留出向上调整的空间。

[RECOMMENDED] 在第三方环境下，期望显示在最上层的元素，通过标签内联和 `!important`，将 `z-index` 指定为 `2147483647`。

解释：第三方环境对于开发者来说完全不可控。在第三方环境下的元素，为了保证元素不被其页面其他样式定义覆盖，需要采用此做法。

值与单位

文本

[MUST] 文本内容必须用双引号包围。

解释：文本类型的内容可能在选择器、属性值等内容中。

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: "”";
}
```

数值

[MUST] 当数值为 0 - 1 之间的小数时，省略整数部分的 `0`。

示例：

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

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

url()

[MUST] `url()` 函数中的路径不加引号。

示例：

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

[RECOMMENDED] `url()`函数中的绝对路径可省去协议名。

示例：

```
body {
  background: url(//baidu.com/img/bg.png) no-repeat 0 0;
}
```

长度

[MUST] 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;
}
```

颜色

[MUST] RGB颜色值必须使用十六进制记号形式 `#rrggbb`。不允许使用 `rgb()`。

解释：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。

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);
}
```

[MUST] 颜色值可以缩写时，必须使用缩写形式。

示例：

```
/* good */
.success {
  background-color: #aca;
}

/* bad */
.success {
  background-color: #aaccaa;
}
```

[MUST] 颜色值不允许使用命名色值。

示例：

```
/* good */
.success {
  color: #90ee90;
}

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

[RECOMMENDED] 颜色值中的英文字符采用小写。如不用小写也需要保证同一项目内保持大小写一致。

示例：

```
/* good */
.success {
  background-color: #aca;
  color: #90ee90;
}

/* good */
.success {
  background-color: #ACA;
  color: #90EE90;
}

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

2D 位置

[MUST] 必须同时给出水平和垂直方向的位置。

解释：

2D 位置初始值为 `0% 0%`，但在只有一个方向的值时，另一个方向的值会被解析为 center。为避免理解上的困扰，应同时给出两个方向的值，参见background-position属性值的定义。

示例：

```
/* good */
body {
  background-position: center top; /* 50% 0% */
}

/* bad */
body {
  background-position: top; /* 50% 0% */
}
```

文本编排

字体族

[MUST] `font-family` 属性中的字体族名称应使用字体的英文 `Family Name`，其中如有空格，须放置在引号中。

解释：

所谓英文 Family Name，为字体文件的一个元数据，常见名称如下：

字体	操作系统	Family Name

字体	操作系统	Family Name
宋体 (中易宋体)	Windows	SimSun
黑体 (中易黑体)	Windows	SimHei
微软雅黑	Windows	Microsoft YaHei
微软正黑	Windows	Microsoft JhengHei
华文黑体	Mac/iOS	STHeiti
冬青黑体	Mac/iOS	Hiragino Sans GB
文泉驿正黑	Linux	WenQuanYi Zen Hei
文泉驿微米黑	Linux	WenQuanYi Micro Hei

示例：

```
h1 {
  font-family: "Microsoft YaHei";
}
```

[MUST] `font-family` 按「西文字体在前、中文字体在后」、「效果佳 (质量高/更能满足需求)的字体在前、效果一般的字体在后」的顺序编写，最后必须指定一个通用字体族(`serif` / `sans-serif`)。

解释：更详细说明可参考这篇文章。

示例：

```
/* Display according to platform */
.article {
  font-family: Arial, sans-serif;
}

/* Specific for most platforms */
h1 {
  font-family: "Helvetica Neue", Arial, "Hiragino Sans GB", "WenQuanYi Micro Hei
}
```

[MUST] `font-family` 不区分大小写，但在同一个项目中，同样的 `Family Name` 大小写必须统一。

示例：

```
/* good */
body {
  font-family: Arial, sans-serif;
}

h1 {
  font-family: Arial, "Microsoft YaHei", sans-serif;
}

/* bad */
body {
  font-family: arial, sans-serif;
}

h1 {
  font-family: Arial, "Microsoft YaHei", sans-serif;
}
```

字体风格

[RECOMMENDED] 需要在 Windows 平台显示的中文内容，不要使用除 `normal` 外的 `font-style` 。其他平台也应慎用。

行高

[RECOMMENDED] `line-height` 在定义文本段落时，应使用数值。

解释：

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。

示例：

```
.container {
  line-height: 1.5;
}
```

变换与动画

[MUST] 使用 `transition` 时应指定 `transition-property`。

示例：

```
/* good */
.box {
  transition: color 1s, border-color 1s;
}

/* bad */
.box {
  transition: all 1s;
}
```

[RECOMMENDED] 尽可能在浏览器能高效实现的属性上添加过渡和动画。

解释：见这篇文章，在可能的情况下应选择这样四种变换：

- `transform: translate(npx, npx);`
- `transform: scale(n);`
- `transform: rotate(ndeg);`
- `opacity: 0..1;`

典型的，可以使用 `translate` 来代替 `left` 作为动画属性。

示例：

```
/* good */
.box {
  transition: transform 1s;
}
.box:hover {
  transform: translate(20px); /* move right for 20px */
}

/* bad */
.box {
  left: 0;
  transition: left 1s;
}
.box:hover {
  left: 20px; /* move right for 20px */
}
```

响应式

[MUST] `Media Query` 如果有多个逗号分隔的条件时，应将每个条件放在单独一行中。

示例：

```
@media
(-webkit-min-device-pixel-ratio: 2), /* Webkit-based browsers */
(min--moz-device-pixel-ratio: 2),    /* Older Firefox browsers (prior to Firefox
(min-resolution: 2dppx),             /* The standard way */
(min-resolution: 192dpi) {           /* dppx fallback */
  /* Retina-specific stuff here */
}
```

[RECOMMENDED] 尽可能给出在高分辨率设备(Retina)下效果更佳的样式。

兼容性

属性前缀

[MUST] 带私有前缀的属性由长到短排列，按冒号位置对齐。

解释：标准属性放在最后，按冒号对齐方便阅读，也便于在编辑器内进行多行编辑。

示例：

```
.box {
  -webkit-box-sizing: border-box;
     -moz-box-sizing: border-box;
          box-sizing: border-box;
}
```

Hack

[RECOMMENDED] 需要添加 hack 时应尽可能考虑是否可以采用其他方式解决。

解释：如果能通过合理的 HTML 结构或使用其他的 CSS 定义达到理想的样式，则不应该使用 hack 手段解决问题。通常 hack 会导致维护成本的增加。

[RECOMMENDED] 尽量使用 选择器 hack 处理兼容性，而非 属性 hack。

解释：

尽量使用符合 CSS 语法的 selector hack，可以避免一些第三方库无法识别 hack 语法的问题。

示例：

```
/* IE 7 */
*:first-child + html #header {
  margin-top: 3px;
  padding: 5px;
}

/* IE 6 */
* html #header {
  margin-top: 5px;
  padding: 4px;
}
```

[RECOMMENDED] 尽量使用简单的 属性 hack。

示例：

```
.box {
  _display: inline; /* fix double margin */
  float: left;
  margin-left: 20px;
}

.container {
  overflow: hidden;
  *zoom: 1; /* triggering hasLayout */
}
```

Expression

[MUST] 禁止使用 `Expression`。

Javascript 编码规范

目录

- **基本规则**
  - 文件名
  - 文件编码
  - 特殊字符转义
  - 非ASCII字符
- **格式**
  - 大括号
  - 缩进
  - 分号
  - 空格
  - 注释
  - 换行
- **命名**
- **语言特性**
  - 严格模式
  - 变量声明
  - 使用var申明变量
  - 每行只声明一个变量
  - 变量声明提升
  - 数组
  - 不要使用Array的多参数构造函数
  - 数组元素的换行
  - 数组的尾 逗号
  - Array.prototype.slice方法的使用
  - 对象
  - 使用{}代替Object来声明对象 量面
  - 不要使用保留字作为key
  - 对象属性的换行
  - 对象的尾逗号
  - 仅对包含非法字符的属性名使用引号
  - 访问对象的属性
  - 字符串
  - 使用单引号
  - 不要人为 截断长的字符串
  - 字符串拼接
  - 布尔值
  - 使用===和!==代替==和!=
  - 类型转换
  - 三元表达式
  - 函数
  - 立即执行表达式
  - 不要将参数命名为arguments
  - Function构造器
- **不允许使用的特性或行为**

基本规则

文件名

[MUST]文件名统一采用小写，建议使用意义明确的单词或单词缩写命名，多个单词或缩写之间可以使用横线（-）分隔。

文件编码

[MUST]文件编码统一使用utf-8。

特殊字符转义

[RECOMMENDED]对于特殊的转义序列，比如`\'`，`\"`，`\t`，直接使用这种形式即可，不要使用其对应的数字转义，比如`\u000a`或`\u{a}`这样的形式（都是表示双引号的转义）。

非ASCII字符

[RECOMMENDED]对于非ASCII的字符，可以直接使用实际的Unicode字符或是使用其等价的十六进制Unicode转义，比如希腊字符"μ"，在定义时直接使用`var units='μs'`比使用`var units='\u03bcs'`在可读性上更好。

格式

大括号

[MUST]所有的控制结构都要求使用大括号，比如if，else，for，while以及其他，即使大括号中只有一条语句。

```
// 不推荐
for (let i = 0; i < foo.length; i++) bar(foo[i]);

// 推荐
for (let i = 0; i < foo.length; i++) {
  bar(foo[i]);
}
```

[MUST]对于非空块大括号的使用，遵循以下几条：

- 左大括号前没有换行。
- 左大括号后需要换行。
- 右大括号前需要换行。
- 右大括号如果是用来结束一条语句或函数、类或类中方法，则它后面需要换行，除非它后面跟else，catch，while或者逗号、分号、右括号。

```
function login(name) {
  if (condition(foo)) {
    try {
      something();
    } catch (err) {
      recover();
    }
  }
}
```

[RECOMMENED]对于空块，考虑代码可读性，建议添加相关的注释说明，同时 换行与非空块保持一致。

```
try {
  // ...
} catch (err) {
  // 代码块可为空，但应添加相关注释说明
}
```

缩进

[MUST]使用两个空格代替tab来进行缩进，每开始一个新的代码块时，都在原缩进基础上增加一个缩进级别，当该块结束时，回到之前的缩进级别。

分号

[MUST]每条语句必须以分号结束，分号不得省略。

空格

[RECOMMENDED]除去语言本身的要求之外，空格还出现在以下场景：

- 保留字（if，for，catch等）与其后面出现的左括号之间保留一个空格。
- 保留字（else，catch）与其前面出现的右大括号之间保留一个空格。
- 在任何左大括号前保留一个空格，但有两种场景除外：
  - 如果一个对象字面量作为函数的第一个参数或作为一个数组字面量的第一个元素，那它的左大括号前可以不加空格。
  - 在一些模板中，比如`${message}`，由于模板语言本身的要求，是不能在左大括号前添加空格的。
- 任何二元或三元操作符的前后要添加空格。
- 在逗号或分号之后，比如函数中多个参数之间的逗号以及for循环中的分号。
- 在对象字面量中，冒号之后添加空格。
- 注释标识与注释内容之间添加空格。

注释

[MUST]注释与被注释的代码保持相同的缩进级别，注释的内容部分与注释标识符之间保留一个 空格。

[MUST]对于多行注释。以/**开始，中间部分每行以*开始，最后以*/结束，一般位于类、方法之前，注释要保持对齐。

```
/**
 * ...
 */
function getToken(uid) {
  // ...
}
```

[MUST]对于单行注释。位于所要注释的代码语句之前的一行，以//作为行的起始。在单行注释前保留一个空行，除非该注释为代码块中的第一行。

换行

[MUST]在类（原型）或对象字面量中，几个方法之间添加换行。

[RECOMMENDED]在类或对象字面量中，连续的属性定义之间添加空行，用来创建逻辑分块。

[RECOMMENDED]在具体的方法中，在多条语句之间添加空行，用来创建逻辑分块。

[RECOMMENDED]函数的参数名称如果太长，考虑可读性可将每个参数独占一行。

```
function doSomething(
  veryDescriptiveArgumentNumberOne,
  veryDescriptiveArgumentTwo,
  tableModelEventHandlerProxy,
  artichokeDescriptorAdapterIterator) {
    // ...
}
```

[RECOMMENDED]在像if和while这样的控制结构中，其条件部分如果太长，可以将一条或一组条件单独写在一行（对于复杂或可复用的条件可以考虑封装成更具语义的函数），至于逻辑操作符（&&、||、!）可以放在每行的结尾也可以放在每行的开头。

```
// 不推荐
if ((foo === 123 || bar === 'abc') && doesItLookGoodWhenItBecomesThatLong() && isThisReallyHappening()) {
  thing1();
}

// 推荐
if (
  (foo === 123 || bar === "abc") &&
  doesItLookGoodWhenItBecomesThatLong() &&
  isThisReallyHappening()) {
  thing1();
}
```

命名

[MUST] 命名遵循如下规则：

- 标识符只能由ASCII字符或数字，少数情况下使用"$"符号（比如 声明jQuery选择器选中的对象）。
- 使用一个合理的更具描述性的名称，不用考虑命名过长的问题，相比这个更重要的是让一个新手立刻理解你的代码，不要对项目外的读者使用意义不明确或不熟悉的缩写，也不要为了缩写而删除单词中的某些字母。
- 对象、方法、参数及本地变量都使用首字母小写的驼峰形式。
- 类名使用首字母大写的驼峰形式。

语言特性

严格模式

[RECOMMENDED]推荐使用严格模式，但建议不要对整个脚本使用严格模式，特别是一些使用第三方js库的场景，在代码合并后会因为第三方js库未遵循严格模式而导致报错。如果需要对一个脚本使用严格模式，可以将脚本内容放在一个立即执行的匿名函数中执行。

变量声明

使用var申明变量

[MUST]声明本地变量时，必须使用var，不得省略（否则会声明全局变量，在严格模式下会报错）。

每行只声明一个变量

[MUST]每行仅申明一个变量

```
// 不推荐
var a = 1, b = 2;

// 不推荐，容易产生全局变量
var a = b = 1;

// 推荐
var a = 1;
var b = 2;
var c;
var d;
```

变量声明提升

[RECOMMENDED]变量的声明应放在其作用域的最前面，这可以帮助避免变量声明提升来的问题。

数组

不要使用Array的多参数构造函数

[RECOMMENDED]建议在使用Array构建函数时，不要使用多参数的调用方式，单参数与多参数的 语义是完全不同的。

```
// 得到的是[1,2,3]
var a1 = new Array(1, 2, 3);
// 得到的[undefined,undefined,undefined]
var a2 = new Array(3);
```

数组元素的换行

[RECOMMENDED]数组字面量可以使用如下两种格式，主要还是注重语义分组。在多行形式下，在左中括号之后及右中括号之前都要添加换行。

```
// 一般情况
var num = [1, 2, 3];

// 优化
var items = [
  {
    title:'first'
  },
  {
    title:'second'
  },
  {
    title:'third'
  }
];
```

数组的尾逗号

[MUST]不要数组的最后一个元素之后添加逗号(在IE的某些版本下会报错或导致数组的长度不正确)。

```
// 不推荐
var story = [
  first,
  second,
  third,
];

// 推荐
var story = [
  first,
  second,
  third
];
```

Array.prototype.slice方法的使用

[RECOMMENDED]可以使用slice方法来进行数组的拷贝

```
var desArray = sourceArray.slice();
```

[RECOMMENDED]可以使用Array.prototype.slice来将类数组 转换成数组

```
var args = Array.prototype.slice.call(arguments);
// 或
var args = [].slice.call(arguments);
```

对象

使用{}代替Object来声明对象量面

[RECOMMENDED]使用{}代替Object来创建对象字面量

```
var item = {};
```

不要使用保留字作为key

[MUST]不要使用保留字作为key，在IE的一些旧版本中会报错

```
var obj = {
  // 在IE8下会报错
  for: 'before'
}
```

对象属性的换行

[RECOMMENDED]与数组元素的书写形式类似，对象的属性定义也可 采用如下两种形式。

```
// 一般情况
var point = { x: 0, y: 1 }

// 优化
var point = {
  x: 0,
  y: 1
}
```

对象的尾逗号

[MUST]对象的最后一个 键值对之后不要添加逗号

```
// 不推荐
var user = {
  name: 'Steven',
  age: 18,
  sex: 'male',
}

// 推荐
var user = {
  name: 'Steven',
  age: 18,
  sex: 'male'
}
```

仅对包含非法字符的属性名使用引号

[RECOMMEND]对 象中的属性，一般不使用引号，只有当其中包含非法字符时才使用引号。

```
var good = {
  foo: 3,
  bar: 4,
  'data-blah': 5
};
```

访问对象的属性

[RECOMMENDED]访问对象的属性时，使用.而不要使用中括号的形式，但如果是通过变量的方式来访问属性，则必须使用中括号的形式。

字符串

使用单引号

[MUST]使用单引号代替双引号。

不要人为截断长的字符串

[RECOMMENDED]当字符串超长时，不要人为的进行截取、换行和拼接，这可能会带来问题，比如搜索时可能无法搜到某个关键字，这种情况就交给IDE来进行自动换行。

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

字符串拼接

[RECOMMENDED]当需要动态拼接字符串时，特别是要拼接量较多时，建议使用数组的join方法。

布尔值

使用===和!==代替==和!=

[MUST]==及!=会有一个类型转换的过程，比如2=='2'返回的是true，但使用2==='2'时，返回的就是false，因为它会先进行一个类型的判断，在代码中判断等于或不等于时使用===和!==

类型转换

[MUST]在if的判断条件中，对于布尔类型可以不使用比较操作符来判断，但对于字符串和数字，要显示的使用比较操作符来判断。

```
if (isValid) {
  // ...
}

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

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

三元表达式

[MUST]三元表达式不要嵌套使用。

```
// 不推荐
var foo = maybe1 > maybe2 ? "bar" : value1 > value2 ? "baz" : null;

// 推荐
var maybeNull = value1 > value2 ? 'baz' : null;
var foo = maybe1 > maybe2 ? 'bar' : maybeNull;
```

[RECOMMENDED]应避免在不需要的场合下使用三元表达式。

```
// 不推荐
var foo = a ? a : b;
var bar = c ? true : false;
var baz = c ? false : true;

// 推荐
var foo = a || b;
var bar = !!c;
var baz = !c;
```

[RECOMMENDED]出于可读性考虑，三元表达式中各 部分可以各独占一行。

```
var foo = isLogin()
  ? getUserInfo()
  : 'Anonymous'
```

函数

立即执行表达式

[MUST]对于立即执行表达式要使用括号括起，并在之前添加分号。

```
;(function () {
  console.log('Welcome to the Internet. Please follow me.');
}());
```

不要将参数命名为arguments

[MUST]不要将函数的参数命名为arguments,这会覆盖函数作用域提供的内置同名变量。

Function构造器

[MUST]不要使用Function构造器来创建函数，使用Function构造器来创建函数与eval()的方式类似，会产生漏洞，是不可取的。

不允许使用的特性或行为

[MUST]包括如下场景：

- 不允许使用with
- 不允许使用eval或Function构造函数
- 不允许使用非标准的特性，包括已经被移除的特性，或新的尚未标准化的特性
- 不允许在Boolean，Number，String，Symbol上进行new操作
- 不允许修改内建对象

Javascript 编码规范

目录

- **基本规则**
- **格式**
- **命名**
- **语言特性**
  - 变量声明
  - 使用const和let代替var
  - 按需声明变量，声明之后尽快初始化
  - 数组
  - 数组的尾逗号
  - 解构赋值
  - [RECOMMENDED]使用扩展操作符代替Array.prototype上的一些方法。
  - [RECOMMENDED]使用Array.from方法来将类数组转换成真正的数组
  - 对象
  - 对象的尾逗号
  - 方法和属性的简写
  - 对象的解构赋值
  - 字符串
  - 使用字符串模板
  - 类
  - 构造函数
  - 计算属性
  - 不要重复定义类的成员
  - 函数
  - 使用剩余参数来代替arguments
  - 使用默认参数
  - 默认值参数要放到函数参数列表的末尾
  - 箭头函数
  - 不要直接修改函数的入参
  - 生成器函数
  - 模块
  - 使用模块
  - 模块导入
  - 不要导出可变绑定
  - 优先使用默认导出
  - import语句放在文件的开头
  - 不要在import中使用webpack的loader语法

基本规则

参见ES5规范

格式

参见ES5规范

命名

参见ES5规范

语言特性

变量声明

使用const和let代替var

[MUST]声明本地变量时，使用const和let，不要使用var，同时默认使用const，如果变量需要重新赋值，则使用let代替。

按需声明变量，声明之后尽快初始化

[RECOMMENDED]本地变量不要习惯性的在它们所包含代码块的起始处就进行声明，而应该在接近第一次使用之前进行声明，来减小他们的作用域。

数组

数组的尾逗号

[MUST]在使用多行形式定义数组时，在最后一个成员之后添加逗号。

```
const chapters = [
  chapterOne,
  chapterTwo,
  chapterThree,
]
```

解构赋值

[MUST]数组赋值时可以 通过 解构赋值从一个数组或迭代器中获取多个值，可以配合扩展运算符一起使用。对不需要关注的元素应该要省略。

```
const [a, b, c, ...rest] = [1, 2, 3, 4, 5, 6];
const [x, , y, , z] = [1, 2, 3, 4, 5, 6];
```

[RECOMMENDED]使用扩展操作符代替Array.prototype上的一些方法。

```
// 数组拷贝
const itemsCopy = [...items];

// 数组合并，代替Array.prototype.concat
const itemsConcat = [...arrayOne, ...arrayTwo];
```

[RECOMMENDED]使用Array.from方法来将类数组转换成真正的数组

```
const foo = document.querySelectorAll('.foo');
const nodes = Array.from(foo);
```

对象

对象的尾逗号

[MUST]在使用多行形式定义对象字面时，最后一个键值对之后添加逗号。

```
const components = {
  componentOne,
  componentTwo,
  componentThree,
}
```

方法和属性的简写

[MUST]在定义对象时，可以使用 方法和属性的简写形式。

```
// 非简写
const atom = {
  addValue: function (value) {
    return atom.value + value;
  },
};

const obj = {
  lukeSkywalker: lukeSkywalker,
};

// 简写
const atom = {
  addValue(value) {
    return atom.value + value;
  },
};

const obj = {
  lukeSkywalker,
};
```

这里需要注意，如果使用函数的简写形式或原function的形式，其中this是指所处的对象本身，但如果是使用箭头函数，那this是指向该对象外的作用域

对象的解构赋值

[RECOMMENDED]对象解构赋值与数组的解构赋值类似，不同点在于它是根据key来进行赋值的，与位置没关系。对象的解构赋值也可用于函数的参数，但尽量使用简单的（只是有一个层级），不要使用有复杂的几层嵌套的属性。此外，关于默认值也与数组的解构赋值类似，形式如下：

```
function destructured(ordinary, { num = 1, str = 'some default' } = {}) {
  // ...
}
```

字符串

使用字符串模板

[MUST]当需要动态拼接字符串时，比如字符串中包括一个变量，可以使用字符串模板。

```
function sayHi(name) {
  return `How are you, ${name}?`;
}
```

类

构造函数

[RECOMMENDED]使用class来定义类及类的方法，不要使用旧语法及直接操作prototype。

[MUST]构造函数是可选的，子类的构造函数中，在设置任何字段或访问this之前必须调用super()。

[MUST]在进行类的继承操作时，使用extends。如果没为类指定构造函数，它会默认添加一个，所以在定义子类时，如果构造函数为空或其内容仅仅是调用父类的构造方法（super），则可以直接省略。

```
class Jedi {
  constructor(name) {
    this.name = name;
  }
  getName() {
    return this.name;
  }
}

class SubJedi extends Jedi {
}
```

计算属性

[MUST]只有symbol类型的计算属性可以在类中使用，对于逻辑上支持迭代的类，应当定义[Symbol.iterator]方法。

不要重复定义类的成员

[MUST]重复定义类的成员后会静默选择最后一个定义。

函数

使用剩余参数来代替arguments

[RECOMMENDED]剩余参数在语义和可读性上更好，而且它本身就是一个数组，不像arguments是一个类数组。

使用默认参数

[RECOMMENDED]使用参数的 默认值语法代替在函数体内对参数的操作来进行默认值的处理。

```
// 不推荐
function handleThings(opts) {
  opts = opts || {};
  // ...
}

// 推荐
function handleThings(opts = {}) {
  // ...
}
```

默认值参数要放到函数参数列表的末尾

[MUST]函数的参数列表中，要将默认值参数放在列表的最后部分。

```
// 不推荐
function handleThings(opts = {}, name) {
  // ...
}

// 推荐
function handleThings(name, opts = {}) {
  // ...
}
```

箭头函数

[RECOMMENDED]推荐使用箭头函数来代替function声明式的函数，特别是作为一个嵌套函数时。

```
// 不推荐
[1, 2, 3].map(function (x) {
  const y = x + 1;
  return x * y;
});

// 推荐
[1, 2, 3].map(x => {
  const y = x + 1;
  return x * y;
});
```

[MUST]如果函数的参数只有一个，参数可以省略括号，也可以保留，但代码风格必须保持统一。

[RECOMMENDED]如果函数体只是单语句表达式，可以省略包裹函数体的大括号并使用隐式的return

```
// 不推荐
[1, 2, 3].map(number => {
  const nextNumber = number + 1;
  `A string containing the ${nextNumber}.`;
});

// 推荐，单句表达式，省略大括号并使用隐式return
[1, 2, 3].map(number => `A string containing the ${number}.`);

// 推荐
[1, 2, 3].map((number) => {
  const nextNumber = number + 1;
  return `A string containing the ${nextNumber}.`;
});

// 当返回的是一个对象字面量时，要用括号括起来
[1, 2, 3].map((number, index) => ({
  [index]: number,
}));
```

不要直接修改函数的入参

[MUST]不要直接修改函数的入参，这可能会造成意料之外的副作用，而且可读性不好，可以使用返回值来代替。

生成器函数

[MUST]在原生环境不支持的情况下（浏览器端），不要使用生成器函数，即使你使用了babel之类的工具进行转换。

模块

使用模块

[RECOMMENDED]使用ES6的模块系统，使用import/export导入导出模块（node环境尚不支持此语法，建议保留其自身的模块语法）。

模块导入

[RECOMMENDED]不要使用 通配符的方式进行模块导入，这可以有效的保证你在导出模块时，会有一个默认导出（export default）。

```
// 不推荐
import * as request from './request';

// 推荐
import request from './request';
```

[MUST]从同一个模块引入的内容使用一条import语句。

```
// 不推荐
import foo from 'foo';
// … some other imports … //
import { named1, named2 } from 'foo';

// 推荐
import foo, { named1, named2 } from 'foo';
```

不要导出可变绑定

[MUST]除特殊场景，导出的内容都要求是一个常量引用。

```
// 不推荐
let foo = 3;
export { foo };

// 推荐
const foo = 3;
export { foo };
```

优先使用默认导出

[RECOMMENDED]如果一个模块中只一处导出，则使用默认导出代替具名导出，这样可以代码的可读性和可维护。

import语句放在文件的开头

[MUST]import语句存在声明提升，所以应将模块引入语句集成中在文件的起始部分。