史上最全 Vue 前端代码风格指南

mp.weixin.qq.com/s/2GLfEHXDTCOD1q-etlGMEQ

前端新世界 2021-09-14 08:30

喜欢就关注我们吧

> 作者 | 卡喵妹
> https://juejin.cn/post/6987349513836953607

一、命名规范

市面上常用的命名规范：

- `camelCase`（小驼峰式命名法 —— 首字母小写）
- `PascalCase`（大驼峰式命名法 —— 首字母大写）
- `kebab-case`（短横线连接式）
- `Snake`（下划线连接式）

1.1 项目文件命名

1.1.1 项目名

全部采用小写方式， 以**短横线**分隔。例： `my-project-name` 。

1.1.2 目录名

参照项目命名规则，有复数结构时，要采用复数命名法。例：docs、assets、components、directives、mixins、utils、views。

```
my-project-name/
|- BuildScript    // 流水线部署文件目录
|- docs           // 项目的细化文档目录（可选）
|- nginx          // 部署在容器上前端项目 nginx 代理文件目录
|- node_modules   // 下载的依赖包
|- public         // 静态页面目录
    |- index.html // 项目入口
|- src            // 源码目录
    |- api        // http 请求目录
    |- assets     // 静态资源目录，这里的资源会被wabpack构建
        |- icon   // icon 存放目录
        |- img    // 图片存放目录
        |- js     // 公共 js 文件目录
        |- scss   // 公共样式 scss 存放目录
            |- frame.scss   // 入口文件
            |- global.scss  // 公共样式
            |- reset.scss   // 重置样式
    |- components     // 组件
    |- plugins        // 插件
    |- router         // 路由
    |- routes         // 详细的路由拆分目录（可选）
        |- index.js
    |- store          // 全局状态管理
    |- utils          // 工具存放目录
        |- request.js // 公共请求工具
    |- views          // 页面存放目录
    |- App.vue        // 根组件
    |- main.js        // 入口文件
    |- tests          // 测试用例
    |- .browserslistrc// 浏览器兼容配置文件
    |- .editorconfig  // 编辑器配置文件
    |- .eslintignore  // eslint 忽略规则
    |- .eslintrc.js   // eslint 规则
    |- .gitignore     // git 忽略规则
    |- babel.config.js // babel 规则
    |- Dockerfile // Docker 部署文件
    |- jest.config.js
    |- package-lock.json
    |- package.json // 依赖
    |- README.md // 项目 README
    |- vue.config.js // webpack 配置
```

1.1.3 图像文件名

全部采用小写方式， 优先选择单个单词命名，多个单词命名以**下划线**分隔。

```
banner_sina.gif
menu_aboutus.gif
menutitle_news.gif
logo_police.gif
logo_national.gif
pic_people.jpg
pic_TV.jpg
```

1.1.4 HTML 文件名

全部采用小写方式， 优先选择单个单词命名，多个单词命名以**下划线**分隔。

```
|- error_report.html
|- success_report.html
```

1.1.5 CSS 文件名

全部采用小写方式， 优先选择单个单词命名，多个单词命名以**短横线**分隔。

```
|- normalize.less
|- base.less
|- date-picker.scss
|- input-number.scss
```

1.1.6 JavaScript 文件名

全部采用小写方式， 优先选择单个单词命名，多个单词命名以**短横线**分隔。

```
|- index.js
|- plugin.js
|- util.js
|- date-util.js
|- account-model.js
|- collapse-transition.js
```

> 上述规则可以快速记忆为“静态文件下划线，编译文件短横线”。

1.2 Vue 组件命名

1.2.1 单文件组件名

文件扩展名为 `.vue` 的 `single-file components` (单文件组件)。单文件组件名应该始终是**单词大写开头** (PascalCase)。

```
components/
|- MyComponent.vue
```

1.2.2 单例组件名

只拥有单个活跃实例的组件应该以 `The` 前缀命名，以示其唯一性。

这不意味着组件只可用于一个单页面，而是_每个页面_只使用一次。这些组件永远不接受任何 prop，因为它们是为你的应用定制的。如果你发现有必要添加 prop，那就表明这实际上是一个可复用的组件，_只是目前_在每个页面里只使用一次。

比如，头部和侧边栏组件几乎在每个页面都会使用，不接受 prop，该组件是专门为该应用所定制的。

```
components/
|- TheHeading.vue
|- TheSidebar.vue
```

1.2.3 基础组件名

> 基础组件：不包含业务，独立、具体功能的基础组件，比如**日期选择器**、**模态框**等。这类组件作为项目的基础控件，会被大量使用，因此组件的 API 进行过高强度的抽象，可以通过不同配置实现不同的功能。

应用特定样式和约定的基础组件(也就是展示类的、无逻辑的或无状态、不掺杂业务逻辑的组件) 应该全部以一个特定的前缀开头 —— Base。**基础组件在一个页面内可使用多次，在不同页面内也可复用，是高可复用组件。**

```
components/
|- BaseButton.vue
|- BaseTable.vue
|- BaseIcon.vue
```

1.2.4 业务组件

> 业务组件：它不像基础组件只包含某个功能，而是在业务中被多个页面复用的（具有可复用性），它与基础组件的区别是，业务组件只在当前项目中会用到，不具有通用性，而且会包含一些业务，比如数据请求；而基础组件不含业务，在任何项目中都可以使用，功能单一，比如一个具有数据校验功能的输入框。

掺杂了复杂业务的组件（拥有自身 `data`、`prop` 的相关处理）即业务组件应该以 `Custom` 前缀命名。业务组件在一个页面内比如：某个页面内有一个卡片列表，而样式和逻辑跟业务紧密相关的卡片就是业务组件。

```
components/
|- CustomCard.vue
```

1.2.5 紧密耦合的组件名

和父组件紧密耦合的子组件应该以父组件名作为前缀命名。 因为编辑器通常会按字母顺序组织文件，所以这样做可以把相关联的文件排在一起。

```
components/
|- TodoList.vue
|- TodoListItem.vue
|- TodoListItemButton.vue
```

1.2.6 组件名中单词顺序

组件名应该以高级别的 (通常是一般化描述的) 单词开头，以描述性的修饰词结尾。 因为编辑器通常会按字母顺序组织文件，所以现在组件之间的重要关系一目了然。如下组件主要是用于搜索和设置功能。

```
components/
|- SearchButtonClear.vue
|- SearchButtonRun.vue
|- SearchInputQuery.vue
|- SearchInputExcludeGlob.vue
|- SettingsCheckboxTerms.vue
|- SettingsCheckboxLaunchOnStartup.vue
```

还有另一种多级目录的方式，把所有的搜索组件放到“search”目录，把所有的设置组件放到“settings”目录。我们只推荐在非常大型 (如有 100+ 个组件) 的应用下才考虑这么做，因为在多级目录间找来找去，要比在单个 components 目录下滚动查找要花费更多的精力。

1.2.7 完整单词的组件名

组件名应该倾向于而不是缩写。 编辑器中的自动补全已经让书写长命名的代价非常之低了，而其带来的明确性却是非常宝贵的。不常用的缩写尤其应该避免。

```
components/
|- StudentDashboardSettings.vue
|- UserProfileOptions.vue
```

1.3 代码参数命名

1.3.1 name

组件名应该始终是多个单词，应该始终是 PascalCase 的。 根组件 App 以及 `<transition>`、`<component>` 之类的 Vue 内置组件除外。这样做可以避免跟现有的以及未来的 HTML 元素相冲突，因为所有的 HTML 元素名称都是单个单词的。

```
export default {
  name: 'ToDoList',
  // ...
}
```

1.3.2 prop

在声明 prop 的时候，其命名应该始终使用 camelCase，而在模板和 JSX 中应该始终使用 kebab-case。我们单纯的遵循每个语言的约定，在 JavaScript 中更自然的是 camelCase。而在 HTML 中则是 kebab-case。

```
<WelcomeMessage greeting-text="hi"/>
```

```
export default {
  name: 'MyComponent',
  // ...
  props: {
    greetingText: {
      type: String,
      required: true,
      validator: function (value) {
        return ['syncing', 'synced',].indexOf(value) !== -1
      }
    }
  }
}
```

1.3.3 router

Vue Router Path 命名采用 kebab-case 格式。 用 Snake（如：`/user_info`）或 camelCase（如：`/userInfo`)的单词会被当成一个单词，搜索引擎无法区分语义。

```
// bad
{
path: '/user_info', // user_info 当成一个单词
name: 'UserInfo',
component: UserInfo,
meta: {
title: ' - 用户',
desc: ''
  }
},

                // good
{
  path: '/user-info', // 能解析成 user info
  name: 'UserInfo',
  component: UserInfo,
  meta: {
    title: ' - 用户',
    desc: ''
  }
},
```

1.3.4 模板中组件

对于绝大多数项目来说，在单文件组件和字符串模板中组件名应该总是 PascalCase 的，但是在 DOM 模板中总是 kebab-case 的。

```
<!-- 在单文件组件和字符串模板中 -->
<MyComponent/>

 <!-- 在 DOM 模板中 -->
<my-component></my-component>
```

1.3.5 自闭合组件

在单文件组件、字符串模板和 JSX 中没有内容的组件应该是自闭合的——但在 DOM 模板里永远不要这样做。

```
<!-- 在单文件组件和字符串模板中 -->
<MyComponent/>

<!-- 在所有地方 -->
<my-component></my-component>
```

1.3.6 变量

- 命名方法：camelCase
- 命名规范：类型 + 对象描述或属性的方式

```
// bad
var getTitle = "LoginTable"

// good
let tableTitle = "LoginTable"
let mySchool = "我的学校"
```

1.3.7 常量

- 命名方法：全部大写下划线分割
- 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://test.host.com'
```

1.3.8 方法

- 命名方法：camelCase
- 命名规范：统一使用动词或者动词 + 名词形式

```
// 1、普通情况下，使用动词 + 名词形式
// bad
go、nextPage、show、open、login

// good
jumpPage、openCarInfoDialog

// 2、请求数据方法，以 data 结尾
// bad
takeData、confirmData、getList、postForm

// good
getListData、postFormData

// 3、单个动词的情况
init、refresh
```

动词	含义	返回值
can	判断是否可执行某个动作(权)	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

1.3.9 自定义事件

自定义事件应始终使用 kebab-case 的事件名。

不同于组件和 prop，事件名不存在任何自动化的大小写转换。而是触发的事件名需要完全匹配监听这个事件所用的名称。

```
this.$emit('my-event')


<MyComponent @my-event="handleDoSomething" />
```

不同于组件和 prop，事件名不会被用作一个 JavaScript 变量名或 property 名，所以就没有理由使用 camelCase 或 PascalCase 了。并且 `v-on` 事件监听器在 DOM 模板中会被自动转换为全小写 (因为 HTML 是大小写不敏感的)，所以 `v-on:myEvent` 将会变成 `v-on:myevent` ——导致 `myEvent` 不可能被监听到。

原生事件参考列表[1]

由原生事件可以发现其使用方式如下：

```
<div
  @blur="toggleHeaderFocus"
  @focus="toggleHeaderFocus"
  @click="toggleMenu"
  @keydown.esc="handleKeydown"
  @keydown.enter="handleKeydown"
  @keydown.up.prevent="handleKeydown"
  @keydown.down.prevent="handleKeydown"
  @keydown.tab="handleKeydown"
  @keydown.delete="handleKeydown"
  @mouseenter="hasMouseHoverHead = true"
  @mouseleave="hasMouseHoverHead = false">
</div>
```

而为了区分_原生事件_和_自定义事件_在 Vue 中的使用，建议除了多单词事件名使用 kebab-case 的情况下，命名还需遵守为 `on` + **动词** 的形式，如下：

```
<!-- 父组件 -->
<div
  @on-search="handleSearch"
  @on-clear="handleClear"
  @on-clickoutside="handleClickOutside">
</div>


// 子组件
export default {
  methods: {
    handleTriggerItem () {
      this.$emit('on-clear')
    }
  }
}
```

1.3.10 事件方法

- 命名方法：camelCase
- 命名规范：handle + 名称（可选）+ 动词

```
<template>
  <div
    @click.native.stop="handleItemClick()"
    @mouseenter.native.stop="handleItemHover()">
  </div>
<

/template>

<script>

export default {
  methods: {
    handleItemClick () {
      //...
    },
    handleItemHover () {
      //...
    }
  }
}
</script>
```

二、代码规范

2.1 Vue

2.1.1 代码结构

```
<template>
<div id="my-component">
<DemoComponent />
</div>
</template>



<script>


import DemoComponent from'../components/DemoComponent'

 export default {
  name: 'MyComponent',
  components: {
    DemoComponent
  },
  mixins: [],
  props: {},
  data () {
    return {}
  },
  computed: {},
  watch: {}
  created () {},
  mounted () {},
  destroyed () {},
  methods: {},
}
</script>

<style lang="scss" scoped>
#my-component {
}
</style>
```

2.1.2 data

组件的 `data` 必须是一个函数。

```
// In a .vue file
export default {
  data () {
    return {
      foo: 'bar'
    }
  }
}
```

2.1.3 prop

Prop 定义应该尽量详细。

```
export default {
  props: {
    status: {
      type: String,
      required: true,
      validator: function (value) {
        return [
          'syncing',
          'synced',
          'version-conflict',
          'error'
        ].indexOf(value) !== -1
      }
    }
  }
}
```

2.1.4 computed

应该把复杂计算属性分割为尽可能多的更简单的属性。 小的、专注的计算属性减少了信息使用时的假设性限制，所以需求变更时也用不着那么多重构了。

```
// bad
computed: {
price: function () {
var basePrice = this.manufactureCost / (1 - this.profitMargin)
return (
      basePrice -
      basePrice * (this.discountPercent || 0)
    )
  }
}

          // good
computed: {
  basePrice: function () {
    return this.manufactureCost / (1 - this.profitMargin)
  },
  discount: function () {
    return this.basePrice * (this.discountPercent || 0)
  },
  finalPrice: function () {
    return this.basePrice - this.discount
  }
}
```

2.1.5 为 `v-for` 设置键值

在组件上必须用 `key` 搭配 `v-for` ，以便维护内部组件及其子树的状态。甚至在元素上维护可预测的行为，比如动画中的对象固化 \(object constancy\)[2]。

```
<ul>
  <li
    v-for="todo in todos"
    :key="todo.id">
      {{ todo.text }}
  </li>
</ul>
```

2.1.6 `v-if` 和 `v-for` 互斥

永远不要把 `v-if` 和 `v-for` 同时用在同一个元素上。

```
<!-- bad：控制台报错 -->
<ul>
  <li
    v-for="user in users"
    v-if="shouldShowUsers"
    :key="user.id">
      {{ user.name }}
  </li>
</ul>
```

一般我们在两种常见的情况下会倾向于这样做：

> 为了过滤一个列表中的项目 (比如 `v-for="user in users" v-if="user.isActive"`)。在这种情形下，请将 `users` 替换为一个计算属性 (比如 `activeUsers`)，让其返回过滤后的列表。

```
computed: {
  activeUsers: function () {
    return this.users.filter((user) => {
      return user.isActive
    })
  }
}
```

```
<ul>
  <li
    v-for="user in activeUsers"
    :key="user.id">
      {{ user.name }}
  </li>
</ul>
```

> 为了避免渲染本应该被隐藏的列表 (比如 `v-for="user in users" v-if="shouldShowUsers"`)。这种情形下，请将 `v-if` 移动至容器元素上 (比如 `ul`，`ol`)。

```
<!-- bad -->
<ul>
<li
    v-for="user in users"
    v-if="shouldShowUsers"
    :key="user.id">
      {{ user.name }}
</li>
</ul>

    <!-- good -->
<ul v-if="shouldShowUsers">
  <li
    v-for="user in users"
    :key="user.id">
      {{ user.name }}
  </li>
</ul>
```

2.1.7 多个 attribute 的元素

多个 attribute 的元素应该分多行撰写，每个 attribute 一行。

```
<!-- bad -->
<img src="https://vuejs.org/images/logo.png" alt="Vue Logo">
<MyComponent foo="a" bar="b" baz="c"/>



<!-- good -->
<img
  src="https://vuejs.org/images/logo.png"
  alt="Vue Logo">

<MyComponent
  foo="a"
  bar="b"
  baz="c"/>
```

2.1.8 模板中简单的表达式

组件模板应该只包含简单的表达式，复杂的表达式则应该重构为计算属性或方法。

复杂表达式会让你的模板变得不那么声明式。我们应该尽量描述应该出现的**是什么**，而非**如何**计算那个值。而且计算属性和方法使得代码可以重用。

```
// bad
{{
  fullName.split(' ').map((word) => {
    return word[0].toUpperCase() + word.slice(1)
  }).join(' ')
}}
```

更好的做法：

```
<!-- 在模板中 -->
{{ normalizedFullName }}


// 复杂表达式已经移入一个计算属性
computed: {
  normalizedFullName: function () {
    return this.fullName.split(' ').map(function (word) {
      return word[0].toUpperCase() + word.slice(1)
    }).join(' ')
  }
}
```

2.1.9 带引号的 attribute 值

非空 HTML 特性值应该始终带双引号。

```
<!-- bad -->
<input type=text>
<AppSidebar :style={width:sidebarWidth+'px'}>


<!-- good -->
<input type="text">
<AppSidebar :style="{ width: sidebarWidth + 'px' }">
```

2.1.10 指令缩写

- 用 `:` 表示 `v-bind:`
- 用 `@` 表示 `v-on:`
- 用 `#` 表示 `v-slot:`

```
<input
  :value="newTodoText"
  :placeholder="newTodoInstructions">

<input
  @input="onInput"
  @focus="onFocus">

<template #header>
<h1>Here might be a page title</h1>
</template>

  <template #footer>
  <p>Here's some contact info</p>
</template>
```

2.2 HTML

2.2.1 文件模板

HTML5 文件模板：

```
<!DOCTYPE html>
  <html lang="zh-CN">
  <head>
    <meta charset="UTF-8">
    <title>HTML5标准模版</title>
  </head>
  <body>
  </body>
</html>
```

移动端：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="UTF-8">
<meta name="viewport"
        content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
scalable=no, shrink-to-fit=no">
<meta name="format-detection" content="telephone=no">
<title>移动端HTML模版</title>



<!-- S DNS预解析 -->
<link rel="dns-prefetch" href="">
<!-- E DNS预解析 -->



<!-- S 线上样式页面片，开发请直接取消注释引用 -->
<!-- #include virtual="" -->
<!-- E 线上样式页面片 -->



<!-- S 本地调试，根据开发模式选择调试方式，请开发删除 -->
<link rel="stylesheet" href="css/index.css">
<!-- /本地调试方式 -->



<link rel="stylesheet" href="http://srcPath/index.css">
<!-- /开发机调试方式 -->
<!-- E 本地调试 -->

        </head>
<body>
</body>
</html>
```

PC 端：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="UTF-8">
<meta name="keywords" content="your keywords">
<meta name="description" content="your description">
<meta name="author" content="author,email address">
<meta name="robots" content="index,follow">
<meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">
<meta name="renderer" content="ie-stand">
<title>PC端HTML模版</title>



<!-- S DNS预解析 -->
<link rel="dns-prefetch" href="">
<!-- E DNS预解析 -->



<!-- S 线上样式页面片，开发请直接取消注释引用 -->
<!-- #include virtual="" -->
<!-- E 线上样式页面片 -->



<!-- S 本地调试，根据开发模式选择调试方式，请开发删除 -->
<link rel="stylesheet" href="css/index.css">
<!-- /本地调试方式 -->

            <link rel="stylesheet" href="http://srcPath/index.css">
    <!-- /开发机调试方式 -->
    <!-- E 本地调试 -->
</head>
<body>
</body>
</html>
```

2.2.2 元素及标签闭合

HTML 元素共有以下5种：

- 空元素：area、base、br、col、command、embed、hr、img、input、keygen、link、meta、param、source、track、wbr
- 原始文本元素：script、style
- RCDATA 元素：textarea、title
- 外来元素：来自 MathML 命名空间和 SVG 命名空间的元素
- 常规元素：其他 HTML 允许的元素都称为常规元素

为了能让浏览器更好的解析代码以及能让代码具有更好的可读性，有如下约定：

- 所有具有开始标签和结束标签的元素都要写上起止标签，某些允许省略开始标签或和束标签的元素亦都要写上。
- 空元素标签都不加 “/” 字符。

```
<!-- good -->
<div>
<h1>我是h1标题</h1>
<p>我是一段文字，我有始有终，浏览器能正确解析</p>
</div>



<br data-tomark-pass>

<!-- bad -->
<div>
<h1>我是h1标题</h1>
<p>我是一段文字，我有始无终，浏览器亦能正确解析
</div>

        <br/>
```

2.2.3 代码嵌套

元素嵌套规范，每个块状元素独立一行，内联元素可选。

```
<!-- good -->
<div>
<h1></h1>
<p></p>
</div>
<p><span></span><span></span></p>

         <!-- bad -->
<div>
    <h1></h1><p></p>
</div>
<p>
    <span></span>
    <span></span>
</p>
```

段落元素与标题元素只能嵌套内联元素。

```
<!-- good -->
<h1><span></span></h1>
<p><span></span><span></span></p>

<!-- bad -->
<h1><div></div></h1>
<p><div></div><div></div></p>
```

2.3 CSS

2.3.1 样式文件

样式文件必须写上 `@charset` 规则，并且一定要在样式文件的第一行首个字符位置开始写，编码名用 `“UTF-8”`。

推荐：

```
@charset "UTF-8";
.jdc {}
```

不推荐：

```
/* @charset规则不在文件首行首个字符开始 */
@charset"UTF-8";
.jdc {}



/* @charset规则没有用小写 */
@CHARSET"UTF-8";
.jdc {}

 /* 无@charset规则 */
.jdc {}
```

2.3.2 代码格式化

样式书写一般有两种：一种是紧凑格式 （Compact），一种是展开格式（Expanded）。

推荐：展开格式（Expanded）

```
.jdc {
  display: block;
  width: 50px;
}
```

不推荐：紧凑格式 （Compact）

```
.jdc { display: block; width: 50px;}
```

2.3.3 代码大小写

样式选择器，属性名，属性值关键字全部使用小写字母书写，属性字符串允许使用大小写。

推荐：

```
.jdc {
  display: block;
}
```

不推荐：

```
.JDC {
  DISPLAY: BLOCK;
}
```

2.3.4 代码易读性

1. 左括号与类名之间一个空格，冒号与属性值之间一个空格。

   推荐：

```
.jdc {
  width: 100%;
}
```

   不推荐：

```
.jdc{
  width:100%;
}
```

2. 逗号分隔的取值，逗号之后一个空格。

   推荐：

```
.jdc {
  box-shadow: 1px 1px 1px #333, 2px 2px 2px #ccc;
}
```

   不推荐：

```
.jdc {
  box-shadow: 1px 1px 1px #333,2px 2px 2px #ccc;
}
```

3. 为单个 CSS 选择器或新声明开启新行。

   推荐：

```
.jdc, .jdc_logo, .jdc_hd {
color: #ff0;
}

  .nav{
  color: #fff;
}
```

   不推荐：

```
.jdc, .jdc_logo, .jdc_hd {
  color: #ff0;
}.nav{
  color: #fff;
}
```

4. 颜色值 `rgb()` `rgba()` `hsl()` `hsla()` `rect()` 中不需有空格，且取值不要带有不必要的 0。

   推荐：

```
.jdc {
  color: rgba(255,255,255,.5);
}
```

   不推荐：

```
.jdc {
  color: rgba( 255, 255, 255, 0.5 );
}
```

5. 属性值十六进制数值能用简写的尽量用简写。

   推荐：

```
.jdc {
  color: #fff;
}
```

   不推荐：

```
.jdc {
  color: #ffffff;
}
```

6. 不要为 `0` 指明单位。

   推荐：

```
.jdc {
  margin: 0 10px;
}
```

   不推荐：

```
.jdc {
  margin: 0px 10px;
}
```

2.3.5 属性值引号

CSS 属性值需要用到引号时，统一使用单引号。

推荐：

```
.jdc {
  font-family: 'Hiragino Sans GB';
}
```

不推荐：

```
.jdc {
  font-family: "Hiragino Sans GB";
}
```

2.3.6 属性书写建议

建议遵循以下顺序：

1. 布局定位属性：display / position / float / clear / visibility / overflow
2. 自身属性：width / height / margin / padding / border / background
3. 文本属性：color / font / text-decoration / text-align / vertical-align / white- space / break-word
4. 其他属性（CSS3）：content / cursor / border-radius / box-shadow / text-shadow / background: linear-gradient ...

```
.jdc {
  display: block;
  position: relative;
  float: left;
  width: 100px;
  height: 100px;
  margin: 0 10px;
  padding: 20px 0;
  font-family: Arial, 'Helvetica Neue', Helvetica, sans-serif;
  color: #333;
  background: rgba(0,0,0,.5);
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -o-border-radius: 10px;
  -ms-border-radius: 10px;
  border-radius: 10px;
}
```

3.3.7 CSS3 浏览器私有前缀

CSS3 浏览器私有前缀在前，标准前缀在后。

```
.jdc {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -o-border-radius: 10px;
  -ms-border-radius: 10px;
  border-radius: 10px;
}
```

2.4 JavaScript

2.4.1 单行代码块

在单行代码块中使用空格。

不推荐：

```
function foo () {return true}
if (foo) {bar = 0}
```

推荐：

```
function foo () { return true }
if (foo) { bar = 0 }
```

2.4.2 大括号风格

在编程过程中，大括号风格与缩进风格紧密联系，用来描述大括号相对代码块位置的方法有很多。在 JavaScript 中，主要有三种风格，如下：

【推荐】One True Brace Style

```
if (foo) {
  bar()
} else {
  baz()
}
```

Stroustrup

```
if (foo) {
  bar()
}
else {
  baz()
}
```

Allman

```
if (foo)
{
  bar()
}
else
{
  baz()
}
```

2.4.3 代码中的空格

1. 逗号前后的空格可以提高代码的可读性，团队约定在逗号后面使用空格，逗号前面不加空格。

   推荐：

```
var foo = 1, bar = 2
```

   不推荐：

```
var foo = 1,bar = 2

var foo = 1 , bar = 2

var foo = 1 ,bar = 2
```

2. 对象字面量的键和值之间不能存在空格，且要求对象字面量的冒号和值之间存在一个空格。

   推荐：

```
var obj = { 'foo': 'haha' }
```

   不推荐：

```
var obj = { 'foo' : 'haha' }
```

3. 代码块前要添加空格。

   推荐：

```
if (a) {
  b()
}

function a () {}
```

   不推荐：

```
if (a){
  b()
}

function a (){}
```

4. 函数声明括号前要加空格。

   推荐：

```
function func (x) {
  // ...
}
```

   不推荐：

```
function func(x) {
  // ...
}
```

5. 在函数调用时，禁止使用空格。

   推荐：

```
fn()
```

   不推荐：

```
fn ()

fn
()
```

6. 在操作符前后都需要添加空格。

   推荐：

```
var sum = 1 + 2
```

   不推荐：

```
var sum = 1+2
```

三、注释规范

注释的目的：

提高代码的可读性，从而提高代码的可维护性

注释的原则：

- **如无必要，勿增注释 (As short as possible)**
- **如有必要，尽量详尽 (As long as necessary)**

3.1 HTML 文件注释

3.1.1 单行注释

一般用于简单的描述，如某些状态描述、属性描述等。

注释内容前后各一个空格字符，注释位于要注释代码的上面，单独占一行。

推荐：

```
<!-- Comment Text -->
<div>...</div>
```

不推荐

```
<div>...</div><!-- Comment Text -->

<div><!-- Comment Text -->
  ...
</div>
```

3.1.2 模块注释

一般用于描述模块的名称以及模块开始与结束的位置。

注释内容前后各一个空格字符， `<!-- S Comment Text \-->` 表示模块开始， `<!-- E Comment Text \-->` 表示模块结束，模块与模块之间相隔一行。

推荐：

```
<!-- S Comment Text A -->
<div class="mod_a">
  ...
</div>
<!-- E Comment Text A -->

  <!-- S Comment Text B -->
<div class="mod_b">
  ...
</div>
<!-- E Comment Text B -->
```

不推荐

```
<!-- S Comment Text A -->
<div class="mod_a">
  ...
</div>
<!-- E Comment Text A -->
<!-- S Comment Text B -->
<div class="mod_b">
  ...
</div>
<!-- E Comment Text B -->
```

3.1.3 嵌套模块注释

当模块注释内再出现模块注释的时候，为了突出主要模块，嵌套模块不再使用。

```
<!-- S Comment Text -->
<!-- E Comment Text -->
```

而改用

```
<!-- /Comment Text -->
```

注释写在模块结尾标签底部，单独一行。

```
<!-- S Comment Text A -->
<div class="mod_a">


<div class="mod_b">
        ...
</div>
<!-- /mod_b -->


<div class="mod_c">
     ...
</div>
<!-- /mod_c -->

          </div>
<!-- E Comment Text A -->
```

3.2 CSS 文件注释

3.2.1 单行注释

注释内容第一个字符和最后一个字符都是一个空格字符，单独占一行，行与行之间相隔一行。

推荐：

```
/* Comment Text */
.jdc {}

 /* Comment Text */
.jdc {}
```

不推荐：

```
/*Comment Text*/
.jdc {
display: block;
}

  .jdc {
  display: block;/*Comment Text*/
}
```

3.2.2 模块注释

注释内容第一个字符和最后一个字符都是一个空格字符，`/*` 与 模块信息描述占一行，多个横线分隔符 `-` 与 `*/` 占一行，行与行之间相隔两行。

推荐：

```
/* Module A
---------------------------------------------------------------- */
.mod_a {}

/* Module B
---------------------------------------------------------------- */
.mod_b {}
```

不推荐：

```
/* Module A ------------------------------------------------------ */
.mod_a {}
/* Module B ------------------------------------------------------ */
.mod_b {}
```

3.2.3 文件注释

在样式文件编码声明 `@charset` 语句下面注明页面名称、作者、创建日期等信息。

```
@charset "UTF-8";
/**
 * @desc File Info
 * @author Author Name
 * @date 2015-10-10
 */
```

3.3 JavaScript 文件注释

3.3.1 单行注释

单行注释使用 `//`，注释应单独一行写在被注释对象的上方，不要追加在某条语句的后面。

推荐：

```
// is current tab
const active = true
```

不推荐：

```
const active = true // is current tab
```

注释行的上方需要有一个空行（**除非注释行上方是一个块的顶部**），以增加可读性。

推荐：

```
function getType () {
console.log('fetching type...')

      // set the default type to 'no type'
  const type = this.type || 'no type'
  return type
}
```

```
// 注释行上面是一个块的顶部时不需要空行
function getType () {
  // set the default type to 'no type'
  const type = this.type || 'no type'
  return type
}
```

不推荐：

```
function getType () {
  console.log('fetching type...')
  // set the default type to 'no type'
  const type = this.type || 'no type'
  return type
}
```

3.3.2 多行注释

多行注释使用 `/** ... */`，而不是多行的 `//`。

推荐：

```
/**
 * make() returns a new element
 * based on the passed-in tag name
 */
function make (tag) {
// ...

    return element
}
```

不推荐：

```
// make() returns a new element
// based on the passed in tag name
function make (tag) {
// ...

    return element
}
```

3.3.3 注释空格

注释内容和注释符之间需要有一个空格，以增加可读性。eslint: `spaced-comment`。

推荐：

```
// is current tab
const active = true

/**
 * make() returns a new element
 * based on the passed-in tag name
 */
function make(tag) {
// ...

    return element
}
```

不推荐：

```
//is current tab
const active = true

/**
 *make() returns a new element
 *based on the passed-in tag name
 */
function make(tag) {
// ...

    return element
}
```

3.3.4 特殊标记

有时我们发现某个可能的 bug，但因为一些原因还没法修复；或者某个地方还有一些待完成的功能，这时我们需要使用相应的特殊标记注释来告知未来的自己或合作者。常用的特殊标记有两种：

- `// FIXME`：说明问题是什么

  `// TODO`：说明还要做什么或者问题的解决方案

```
class Calculator extends Abacus {
constructor () {
super ()



// FIXME: shouldn’t use a global here
      total = 0

      // TODO: total should be configurable by an options param
      this.total = 0
  }
}
```

3.3.5 文档类注释

文档类注释，如函数、类、文件、事件等；都使用 jsdoc 规范。

```
/**
 * Book类，代表一个书本.
 * @constructor
 * @param {string} title - 书本的标题.
 * @param {string} author - 书本的作者.
 */
function Book (title, author) {
this.title = title
this.author = author
}

    Book.prototype = {
  /**
   * 获取书本的标题
   * @returns {string|*}
   */
  getTitle: function () {
    return this.title
  },
  /**
   * 设置书本的页数
   * @param pageNum {number} 页数
   */
  setPageNum: function (pageNum) {
    this.pageNum=pageNum
  }
}
```

3.3.6 注释工具

`ESLint` 是当下最流行的 JS 代码检查工具，`ESLint` 中有一些注释相关的规则，用户可选择开启：

- `valid-jsdoc`
- `require-jsdoc`
- `no-warning-comments`
- `capitalized-comments`
- `line-comment-position`
- `lines-around-comment`
- `multiline-comment-style`
- `no-inline-comments`
- `spaced-comment`

四、其它

- 缩进换行请使用两个空格。

- 大型团队多人协作项目推荐 JavaScript 代码末尾加分号。
- 小型个人创新练手项目可尝试使用 JavaScript 代码末尾不加分号的风格，更加清爽简练。

（文本完）

每日分享前端插件干货，欢迎关注！

前端新世界

分享 JS / CSS 技术教程；Vue、React、jQuery等前端开发组件

543篇原创内容

公众号

点赞和在看就是最大的支持❤️