

前端代码规范

Front Standard Guide

前端 JS 项目开发规范

规范的目的是为了编写高质量的代码，让你的团队成员每天得心心情都是愉悦的，大家在一起是快乐的。

引自《阿里规约》的开头片段：

----现代软件架构的复杂性需要协同开发完成，如何高效地协同呢？无规矩不成方圆，无规范难以协同，比如，制订交通法规表面上是要限制行车权，实际上是保障公众的人身安全，试想如果没有限速，没有红绿灯，谁还敢上路行驶。对软件来说，适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率，降低沟通成本。代码的字里行间流淌的是软件系统的血液，质量的提升是尽可能少踩坑，杜绝踩重复的坑，切实提升系统稳定性，码出质量。

一、编程规约

(一)命名规范

1.1.1 项目名称

全部采用小写方式，以中划线分隔。

正例：`mall-management-system`

反例：`mall_management-system` / `mallManagementSystem`

1.1.2 目录命名

全部采用小写方式，以中划线分隔，有复数结构时，要采用复数命名法，缩写不用复数

正例：`scripts` / `styles` / `components` / `images` / `utils` / `layouts` / `demo-styles` / `demo-scripts` / `img` / `doc`

反例：`script` / `style` / `demo_scripts` / `demoStyles` / `imgs` / `docs`

【特殊】VUE 的项目中的 `components` 中的组件目录，使用 `kebab-case` 命名

正例：`head-search` / `page-loading` / `authorized` / `notice-icon`

反例：`HeadSearch` / `PageLoading`

【特殊】VUE 的项目中的除 `components` 组件目录外的所有目录也使用 `kebab-case` 命名

正例：`page-one` / `shopping-car` / `user-management`

反例：`ShoppingCar` / `UserManagement`

1.1.3 JS、CSS、SCSS、HTML、PNG 文件命名

全部采用小写方式，以中划线分隔

正例：`render-dom.js / signup.css / index.html / company-logo.png`

反例：`renderDom.js / UserManagement.html`

1.1.4 命名严谨性

代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用

正例：`henan / luoyang / rmb` 等国际通用的名称，可视同英文。

反例：`DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3`

杜绝完全不规范的缩写，避免望文不知义：

反例：AbstractClass“缩写”命名成 AbsClass；condition“缩写”命名成 condi，此类随意缩写严重降低了代码的可阅读性。

(二)HTML 规范（Vue Template 同样适用）

1.2.1 HTML 类型

推荐使用 HTML5 的文档类型申明：。

（建议使用 text/html 格式的 HTML。避免使用 XHTML。XHTML 以及它的属性，比如 application/xhtml+xml 在浏览器中的应用支持与优化空间都十分有限）。

- 规定字符编码
- IE 兼容模式
- 规定字符编码
- doctype 大写

正例：

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
    <meta charset="UTF-8" />
    <title>Page title</title>
  </head>
  <body>
    
  </body>
</html>
```

1.2.2 缩进

缩进使用 2 个空格（一个 tab）

嵌套的节点应该缩进。

1.2.3 分块注释

在每一个块状元素，列表元素和表格元素后，加上一对 HTML 注释。注释格式

<!-- 英文 中文 start -->

<!-- 英文 中文 end -->

正例：

```
<body>
  <!-- header 头部 start -->
  <header>
    <div class="container">
      <a href="#">
        <!-- 图片会把a标签给撑开，所以不用设置a标签的大小 -->
        
      </a>
    </div>
  </header>
  <!-- header 头部 end -->
</body>
```

1.2.4 语义化标签

HTML5 中新增很多语义化标签，所以优先使用语义化标签，避免一个页面都是 div 或者 p 标签

正例

```
<header></header>
<footer></footer>
```

反例

```
<div>
  <p></p>
</div>
```

1.2.5 引号

使用双引号(" ")而不是单引号(' ')。

正例： ``

反例： ``

(三) CSS 规范

1.3.1 命名

- 类名使用小写字母，以中划线分隔
- id 采用驼峰式命名
- scss 中的变量、函数、混合、placeholder 采用驼峰式命名

ID 和 class 的名称总是使用可以反应元素目的和用途的名称，或其他通用的名称，代替表象和晦涩难懂的名称

不推荐：

```
.fw-800 {  
  font-weight: 800;  
}  
  
.red {  
  color: red;  
}
```

推荐:

```
.heavy {  
  font-weight: 800;  
}  
  
.important {  
  color: red;  
}
```

1.3.2 选择器

1)css 选择器中避免使用标签名

从结构、表现、行为分离的原则来看，应该尽量避免 css 中出现 HTML 标签，并且在 css 选择器中出现标签名会存在潜在的问题。

2)很多前端开发人员写选择器链的时候不使用 直接子选择器（注：直接子选择器和后代选择器的区别）。有时，这可能会导致疼痛的设计问题并且有时候可能会很耗性能。然而，在任何情况下，这是一个非常不好的做法。如果你不写很通用的，需要匹配到 DOM 末端的选择器，你应该总是考虑直接子选择器。

不推荐:

```
.content .title {  
  font-size: 2rem;  
}
```

推荐:

```
.content > .title {  
  font-size: 2rem;  
}
```

1.3.3 尽量使用缩写属性

不推荐:

```
border-top-style: none;  
font-family: palatino, georgia, serif;  
font-size: 100%;  
line-height: 1.6;  
padding-bottom: 2em;  
padding-left: 1em;  
padding-right: 1em;  
padding-top: 0;
```

推荐:

```
border-top: 0;  
font: 100%/1.6 palatino, georgia, serif;  
padding: 0 1em 2em;
```

1.3.4 每个选择器及属性独占一行

不推荐:

```
button{  
  width:100px;height:50px;color:#fff;background:#00a0e9;  
}
```

推荐:

```
button{  
  width:100px;  
  height:50px;  
  color:#fff;  
  background:#00a0e9;  
}
```

1.3.5 省略 0 后面的单位

不推荐:

```
div{  
  padding-bottom: 0px;  
  margin: 0em;  
}
```

推荐:

```
div{  
  padding-bottom: 0;  
  margin: 0;  
}
```

1.3.6 避免使用 ID 选择器及全局标签选择器防止污染全局样式

不推荐:

```
#header{  
  padding-bottom: 0px;  
  margin: 0em;  
}
```

推荐:

```
.header{
  padding-bottom: 0px;
  margin: 0em;
}
```

1.3.7 CSS书写顺序

1. 位置属性(position, top, right, z-index, display, float等)
2. 大小(width, height, padding, margin)
3. 文字系列(font, line-height, letter-spacing, color- text-align等)
4. 背景(background, border等)
5. 其他(animation, transition等)

1.3.8 CSS书写规范

1. 使用CSS缩写属性
2. 去掉小数点前的“0”
3. 简写命名
4. 16进制颜色代码缩写
5. 连字符CSS选择器命名规范
 1. 长名称或词组可以使用中横线来为选择器命名。
 2. 不建议使用“_”下划线来命名CSS选择器
6. 不要随意使用Id
7. 为选择器添加状态前缀

1.3.9 CSS命名规范（规则）

1. 头：header
2. 内容：content/container
3. 尾：footer
4. 导航：nav
5. 侧栏：sidebar
6. 栏目：column
7. 页面外围控制整体佈局宽度：wrapper
8. 左右中：left right center
9. 登录条：loginbar
10. 标志：logo
11. 广告：banner
12. 页面主体：main
13. 热点：hot
14. 新闻：news
15. 下载：download
16. 子导航：subnav
17. 菜单：menu
18. 子菜单：submenu
19. 搜索：search
20. 友情链接：friendlink
21. 页脚：footer
22. 版权：copyright
23. 滚动：scroll
24. 内容：content

- 25. 标签: tags
- 26. 文章列表: list
- 27. 提示信息: msg
- 28. 小技巧: tips
- 29. 栏目标题: title
- 30. 加入: joinus
- 31. 指南: guide
- 32. 服务: service
- 33. 注册: regsiter
- 34. 状态: status
- 35. 投票: vote
- 36. 合作伙伴: partner

1.3.10 注释的写法

```
/* Header */  
内容区  
/* End Header */
```

1.3.11 Id的命名

1. 页面结构

- 容器: container
- 页头: header
- 内容: content/container
- 页面主体: main
- 页尾: footer
- 导航: nav
- 侧栏: sidebar
- 栏目: column
- 页面外围控制整体佈局宽度: wrapper
- 左右中: left right center

2. 导航

- 导航: nav
- 主导航: mainnav
- 子导航: subnav
- 顶导航: topnav
- 边导航: sidebar
- 左导航: leftsidebar
- 右导航: rightsidebar
- 菜单: menu
- 子菜单: submenu
- 标题: title
- 摘要: summary

3. 功能

- 标志: logo
- 广告: banner
- 登陆: login
- 登录条: loginbar
- 注册: register

- 搜索: search
- 功能区: shop
- 标题: title
- 加入: joinus
- 状态: status
- 按钮: btn
- 滚动: scroll
- 标籤页: tab
- 文章列表: list
- 提示信息: msg
- 当前的: current
- 小技巧: tips
- 图标: icon
- 注释: note
- 指南: guild
- 服务: service
- 热点: hot
- 新闻: news
- 下载: download
- 投票: vote
- 合作伙伴: partner
- 友情链接: link
- 版权: copyright
- 注意事项:
 1. 一律小写;
 2. 尽量用英文;
 3. 不加中横和下划线;
 4. 尽量不缩写, 除非一看就明白的单词。

4. CSS样式表文件命名

- 主要的 master.css
- 模块 module.css
- 基本共用 base.css
- 布局、版面 layout.css
- 主题 themes.css
- 专栏 columns.css
- 文字 font.css
- 表单 forms.css
- 补丁 mend.css
- 打印 print.css

(四) LESS 规范

1.4.1 代码组织

1)将公共 less 文件放置在 style/less/common 文件夹

例:// color.less,common.less

2)按以下顺序组织

- 1、@import;
- 2、变量声明;
- 3、样式声明;

```
@import "mixins/size.less";

@default-text-color: #333;

.page {
  width: 960px;
  margin: 0 auto;
}
```

1.4.2 避免嵌套层级过多

将嵌套深度限制在 3 级。对于超过 4 级的嵌套，给予重新评估。这可以避免出现过于详实的 CSS 选择器。

避免大量的嵌套规则。当可读性受到影响时，将之打断。推荐避免出现多于 20 行的嵌套规则出现

不推荐：

```
.main{
  .title{
    .name{
      color:#fff
    }
  }
}
```

推荐：

```
.main-title{
  .name{
    color:#fff
  }
}
```

(五) Javascript 规范

1.5.1 命名

1) 采用小写驼峰命名 lowerCamelCase，代码中的命名均不能以下划线，也不能以下划线或美元符号结束

反例： `_name / name_ / name$`

2) 方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从驼峰形式。

正例： `localValue / getHttpMessage() / inputUserId`

****其中 method 方法命名必须是 动词 或者 动词+名词 形式****

正例： `saveShopCarData / openShopCarInfoDialog`

反例： `save / open / show / go`

****特此说明，增删查改，详情统一使用如下 5 个单词，不得使用其他（目的是为了统一各个端）****

`add / update / delete / detail / get`

附： 函数方法常用的动词：

get 获取/set 设置，
add 增加/remove 删除
create 创建/destroy 移除
start 启动/stop 停止
open 打开/close 关闭，
read 读取/write 写入
load 载入/save 保存，
create 创建/destroy 销毁
begin 开始/end 结束，
backup 备份/restore 恢复
import 导入/export 导出，
split 分割/merge 合并
inject 注入/extract 提取，
attach 附着/detach 脱离
bind 绑定/separate 分离，
view 查看/browse 浏览
edit 编辑/modify 修改，
select 选取/mark 标记
copy 复制/paste 粘贴，
undo 撤销/redo 重做
insert 插入/delete 移除，
add 加入/append 添加
clean 清理/clear 清除，
index 索引/sort 排序
find 查找/search 搜索，
increase 增加/decrease 减少
play 播放/pause 暂停，
launch 启动/run 运行
compile 编译/execute 执行，
debug 调试/trace 跟踪
observe 观察/listen 监听，
build 构建/publish 发布
input 输入/output 输出，
encode 编码/decode 解码
encrypt 加密/decrypt 解密，
compress 压缩/decompress 解压缩
pack 打包/unpack 解包，

parse 解析/emit 生成
connect 连接/disconnect 断开,
send 发送/receive 接收
download 下载/upload 上传,
refresh 刷新/synchronize 同步
update 更新/revert 复原,
lock 锁定/unlock 解锁
check out 签出/check in 签入,
submit 提交/commit 交付
push 推/pull 拉,
expand 展开/collapse 折叠
begin 起始/end 结束,
start 开始/finish 完成
enter 进入/exit 退出,
abort 放弃/quit 离开
obsolete 废弃/depreciate 废旧,
collect 收集/aggregate 聚集

3) 常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。

正例： `MAX_STOCK_COUNT`

反例： `MAX_COUNT`

1.5.2 代码格式

1) 使用 2 个空格进行缩进

正例：

```
if (x < y) {  
    x += 10;  
} else {  
    x += 1;  
}
```

2) 不同逻辑、不同语义、不同业务的代码之间插入一个空行分隔开来以提升可读性。

说明：任何情形，没有必要插入多个空行进行隔开。

1.5.3 字符串

统一使用单引号(')，不使用双引号(")。这在创建 HTML 字符串非常有好处：

正例：

```
let str = 'foo';  
let testDiv = '<div id="test"></div>';
```

反例：

```
let str = 'foo';  
let testDiv = "<div id='test'></div>";
```

1.5.4 对象声明

1)使用字面值创建对象

正例: `let user = {};`

反例: `let user = new Object();`

2) 使用字面量来代替对象构造器

正例:

```
var user = {  
  age: 0,  
  name: 1,  
  city: 3  
};
```

反例:

```
var user = new Object();  
user.age = 0;  
user.name = 0;  
user.city = 0;
```

1.5.5 使用 ES6,7

必须优先使用 ES6,7 中新增的语法糖和函数。这将简化你的程序，并让你的代码更加灵活和可复用。

必须强制使用 ES6, ES7 的新语法，比如箭头函数、await/async，解构，let，for...of 等等

1.5.6 括号

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

正例:

```
if (condition) {  
  doSomething();  
}
```

反例:

```
if (condition) doSomething();
```

1.5.7 undefined 判断

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。

正例:

```
if (typeof person === 'undefined') {  
  ...  
}
```

反例：

```
if (person === undefined) {  
  ...  
}
```

1.5.8 条件判断和循环最多三层

条件判断能使用三目运算符和逻辑运算符解决的，就不要使用条件判断，但是谨记不要写太长的三目运算符。如果超过 3 层请抽成函数，并写清楚注释。

1.5.9 this 的转换命名

对上下文 this 的引用只能使用'self'来命名

1.5.10 慎用 console.log

因 console.log 大量使用会有性能问题，所以在非 webpack 项目中谨慎使用 log 功能

二、Vue 项目规范

(一) Vue 编码基础

vue 项目规范以 Vue 官方规范 (<https://cn.vuejs.org/v2/style-guide/>) 中的 A 规范为基础，在其上面进行项目开发，故所有代码均遵守该规范。

请仔仔细细阅读 Vue 官方规范，切记，此为第一步。

2.1.1. 组件规范

1) 组件名为多个单词。

组件名应该始终是由多个单词组成（大于等于 2），且命名规范为 KebabCase 格式。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。

正例：

```
export default {  
  name: 'TodoItem'  
  // ...  
};
```

反例：

```
export default {  
  name: 'Todo',  
  // ...  
}  
export default {  
  name: 'todo-item',  
  // ...  
}
```

2) 组件文件名为 pascal-case 格式

正例：

```
components/  
|- my-component.vue
```

反例：

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

3) 基础组件文件名为 base 开头，使用完整单词而不是缩写。

正例：

```
components/  
|- base-button.vue  
|- base-table.vue  
|- base-icon.vue
```

反例：

```
components/  
|- MyButton.vue  
|- VueTable.vue  
|- Icon.vue
```

4) 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

正例：

```
components/  
|- todo-list.vue  
|- todo-list-item.vue  
|- todo-list-item-button.vue  
|- user-profile-options.vue （完整单词）
```

反例：

```
components/  
|- TodoList.vue  
|- TodoItem.vue  
|- TodoButton.vue  
|- UProfOpts.vue （使用了缩写）
```

5) 在 Template 模版中使用组件，应使用 PascalCase 模式，并且使用自闭合组件。

正例：

```
<!-- 在单文件组件、字符串模板和 JSX 中 -->  
<MyComponent />  
<Row><table :column="data"/></Row>
```

反例:

```
<my-component /> <row><table :column="data"/></row>
```

6) 组件的 data 必须是一个函数

当在组件中使用 data 属性的时候 (除了 new Vue 外的任何地方), 它的值必须是返回一个对象的函数。因为如果直接是一个对象的话, 子组件之间的属性值会互相影响。

正例:

```
export default {
  data () {
    return {
      name: 'jack'
    }
  }
}
```

反例:

```
export default {
  data: {
    name: 'jack'
  }
}
```

7) Prop 定义应该尽量详细

- 必须使用 camelCase 驼峰命名
- 必须指定类型
- 必须加上注释, 表明其含义
- 必须加上 required 或者 default, 两者二选其一
- 如果有业务需要, 必须加上 validator 验证

正例:

```
props: {
  // 组件状态, 用于控制组件的颜色
  status: {
    type: String,
    required: true,
    validator: function (value) {
      return [
        'succ',
        'info',
        'error'
      ].indexOf(value) !== -1
    }
  },
  // 用户级别, 用于显示皇冠个数
  userLevel: {
    type: String,
    required: true
  }
}
```

8) 为组件样式设置作用域

正例:

```
<template>
  <button class="btn btn-close">X</button>
</template>

<!-- 使用 `scoped` 特性 -->
<style scoped>
  .btn-close {
    background-color: red;
  }
</style>
```

反例:

```
<template>
  <button class="btn btn-close">X</button>
</template>
<!-- 没有使用 `scoped` 特性 -->
<style>
  .btn-close {
    background-color: red;
  }
</style>
```

9) 如果特性元素较多，应该主动换行。

正例:

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

反例:

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

2.1.2. 模板中使用简单的表达式

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。

正例:


```

<template>
  <p>{{ normalizedFullName }}</p>
</template>

// 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(' ')
  }
}

```

反例:

```

<template>
  <p>
    {{
      fullName.split(' ').map(function (word) {
        return word[0].toUpperCase() + word.slice(1)
      }).join(' ')
    }}
  </p>
</template>

```

2.1.3 指令都使用缩写形式

指令推荐都使用缩写形式, (用 : 表示 v-bind: 、用 @ 表示 v-on: 和用 # 表示 v-slot:)

正例:

```

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

```

反例:

```

<input
  v-on:input="onInput"
  @focus="onFocus"
>

```

2.1.4 标签顺序保持一致

单文件组件应该总是让标签顺序保持为`

正例:

```

<template>...</template>
<script>...</script>
<style>...</style>

```

反例:

```
<template>...</template>
<style>...</style>
<script>...</script>
```

2.1.5 必须为 v-for 设置键值 key

2.1.6 v-show 与 v-if 选择

如果运行时，需要非常频繁地切换，使用 v-show；如果在运行时，条件很少改变，使用 v-if。

2.1.7 script 标签内部结构顺序

components > props > data > computed > watch > filter > 钩子函数（钩子函数按其执行顺序） > methods

2.1.8 Vue Router 规范

1) 页面跳转数据传递使用路由参数

页面跳转，例如 A 页面跳转到 B 页面，需要将 A 页面的数据传递到 B 页面，推荐使用 路由参数进行传参，而不是将需要传递的数据保存 vuex，然后在 B 页面取出 vuex 的数据，因为如果在 B 页面刷新会导致 vuex 数据丢失，导致 B 页面无法正常显示数据。

正例：

```
let id = '123';
this.$router.push({ name: 'userCenter', query: { id: id } });
```

2) 使用路由懒加载（延迟加载）机制

```
{
  path: '/uploadAttachment',
  name: 'uploadAttachment',
  meta: {
    title: '上传附件'
  },
  component: () => import('@/view/components/uploadAttachment/index.vue')
},
```

3) router 中的命名规范

path、childrenPoints 命名规范采用 kebab-case 命名规范（尽量 vue 文件的目录结构保持一致，因为目录、文件名都是 kebab-case，这样很方便找到对应的文件）

name 命名规范采用 kebabCase 命名规范且和 component 组件名保持一致！（因为要保持 keep-alive 特性，keep-alive 按照 component 的 name 进行缓存，所以两者必须高度保持一致）

```
// 动态加载
export const reload = [
  {
    path: '/reload',
    name: 'reload',
    component: Main,
    meta: {
      title: '动态加载',
      icon: 'icon iconfont'
    }
  }
]
```

```

    },

    children: [
      {
        path: '/reload/smart-reload-list',
        name: 'SmartReloadList',
        meta: {
          title: 'SmartReload',
          childrenPoints: [
            {
              title: '查询',
              name: 'smart-reload-search'
            },
            {
              title: '执行reload',
              name: 'smart-reload-update'
            },
            {
              title: '查看执行结果',
              name: 'smart-reload-result'
            }
          ]
        },
        component: () =>
          import('@views/reload/smart-reload/smart-reload-list.vue')
      }
    ]
  }
];

```

4) router 中的 path 命名规范

path 除了采用 `kebab-case` 命名规范以外，必须以 / 开头，即使是 children 里的 path 也要以 / 开头。如下示例

-

目的：

经常有这样的场景：某个页面有问题，要立刻找到这个 vue 文件，如果不用以 / 开头，path 为 parent 和 children 组成的，可能经常需要在 router 文件里搜索多次才能找到，而如果以 / 开头，则能立刻搜索到对应的组件

-

```

{
  path: '/file',
  name: 'File',
  component: Main,
  meta: {
    title: '文件服务',
    icon: 'ios-cloud-upload'
  },
  children: [
    {
      path: '/file/file-list',
      name: 'FileList',
      component: () => import('@views/file/file-list.vue')
    }
  ]
}

```

```

    },
    {
      path: '/file/file-add',
      name: 'FileAdd',
      component: () => import('@/views/file/file-add.vue')
    },
    {
      path: '/file/file-update',
      name: 'FileUpdate',
      component: () => import('@/views/file/file-update.vue')
    }
  ]
}

```

(二) Vue 项目目录规范

2.2.1 基础

vue 项目中的所有命名一定要与后端命名统一。

比如权限：后端 privilege, 前端无论 router, store, api 等都必须使用 privilege 单词！

2.2.2 使用 Vue-cli 脚手架

使用 vue-cli3 来初始化项目，项目名按照上面的命名规范。

2.2.3 目录说明

目录名按照上面的命名规范，其中 components 组件用大写驼峰，其余除 components 组件目录外的所有目录均使用 kebab-case 命名。

src	源码目录
-- api	所有api接口
-- assets	静态资源, images, icons, styles等
-- components	公用组件
-- config	配置信息
-- constants	常量信息, 项目所有Enum, 全局常量等
-- directives	自定义指令
-- filters	过滤器, 全局工具
-- datas	模拟数据, 临时存放
-- lib	外部引用的插件存放及修改文件
-- mock	模拟接口, 临时存放
-- plugins	插件, 全局使用
-- router	路由, 统一管理
-- store	vuex, 统一管理
-- themes	自定义样式主题
-- views	视图目录
-- role	role模块名
-- -- role-list.vue	role列表页面
-- -- role-add.vue	role新建页面
-- -- role-update.vue	role更新页面
-- -- index.less	role模块样式
-- -- components	role模块通用组件文件夹
-- employee	employee模块

1) api 目录

- 文件、变量命名要与后端保持一致。
- 此目录对应后端 API 接口，按照后端一个 controller 一个 api.js 文件。若项目较大时，可以按照业务划分子目录，并与后端保持一致。
- api 中的方法名字要与后端 api url 尽量保持语义高度一致性。
- 对于 api 中的每个方法要添加注释，注释与后端 swagger 文档保持一致。

正例：

后端 url: EmployeeController.java

```
/employee/add  
/employee/delete/{id}  
/employee/update
```

前端: employee.js

```
// 添加员工  
addEmployee: (data) => {  
  return postAxios('/employee/add', data)  
},  
// 更新员工信息  
updateEmployee: (data) => {  
  return postAxios('/employee/update', data)  
},  
// 删除员工  
deleteEmployee: (employeeId) => {  
  return postAxios('/employee/delete/' + employeeId)  
},
```

2) assets 目录

assets 为静态资源，里面存放 images, styles, icons 等静态资源，静态资源命名格式为 kebab-case

```
| assets  
|-- icons  
|-- images  
|   |-- background-color.png  
|   |-- upload-header.png  
|-- styles
```

3) components 目录

此目录应按照组件进行目录划分，目录命名为 KebabCase，组件命名规则也为 KebabCase

```
| components  
|-- error-log  
|   |-- index.vue  
|   |-- index.less  
|-- markdown-editor  
|   |-- index.vue  
|   |-- index.js  
|-- kebab-case
```

4) constants 目录

此目录存放项目所有常量，如果常量在 vue 中使用，请使用 vue-enum 插件(<https://www.npmjs.com/package/vue-enum>)

目录结构：

```
| constants
|-- index.js
|-- role.js
|-- employee.js
```

例子：employee.js

```
export const EMPLOYEE_STATUS = {
  NORMAL: {
    value: 1,
    desc: '正常'
  },
  DISABLED: {
    value: 1,
    desc: '禁用'
  },
  DELETED: {
    value: 2,
    desc: '已删除'
  }
};

export const EMPLOYEE_ACCOUNT_TYPE = {
  QQ: {
    value: 1,
    desc: 'QQ登录'
  },
  WECHAT: {
    value: 2,
    desc: '微信登录'
  },
  DINGDING: {
    value: 3,
    desc: '钉钉登录'
  },
  USERNAME: {
    value: 4,
    desc: '用户名密码登录'
  }
};

export default {
  EMPLOYEE_STATUS,
  EMPLOYEE_ACCOUNT_TYPE
};
```

5) router 与 store 目录

这两个目录一定要将业务进行拆分，不能放到一个 js 文件里。

router 尽量按照 views 中的结构保持一致

store 按照业务进行拆分不同的 js 文件

6) views 目录

- 命名要与后端、router、api 等保持一致
- components 中组件要使用 PascalCase 规则

-- views	视图目录
-- role	role模块名
-- role-list.vue	role列表页面
-- role-add.vue	role新建页面
-- role-update.vue	role更新页面
-- index.less	role模块样式
-- components	role模块通用组件文件夹
-- role-header.vue	role头部组件
-- role-modal.vue	role弹出框组件
-- employee	employee模块
-- behavior-log	行为日志log模块
-- code-generator	代码生成器模块

2.2.4 注释说明

整理必须加注释的地方

- 公共组件使用说明
- api 目录的接口 js 文件必须加注释
- store 中的 state, mutation, action 等必须加注释
- vue 文件中的 template 必须加注释，若文件较大添加 start end 注释
- vue 文件的 methods，每个 method 必须添加注释
- vue 文件的 data，非常见单词要加注释

2.2.5 其他

1) 尽量不要手动操作 DOM

因使用 vue 框架，所以在项目开发中尽量使用 vue 的数据驱动更新 DOM，尽量（不到万不得已）不要手动操作 DOM，包括：增删改 dom 元素、以及更改样式、添加事件等。

2) 删除无用代码

因使用了 git/svn 等代码版本工具，对于无用代码必须及时删除，例如：一些调试的 console 语句、无用的弃用功能代码。

3) 默认约定

- 使用带的变量 (this) 表示函数内部私有变量
- 使用带\$的同名变量 (\$name = name) 进行缓存
- 使用带的函数 (getData) 表示与网络请求相关函数
- 但是在Vue为插件、混入等不考虑作为对外公共 API 的自定义私有 property中，请使用\$两个结合来定义私有属性，以确保不会和Vue本身产生冲突 (Vue中使用_前缀来定义其自身的私有 property[update(),_mountComponent()], 使用\$前缀暴露给用户的一个特殊的实例 property[\$refs、\$bus])

