前端开发规范

一、编程规约

（一）命名规范

1.项目命名

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

2.目录命名

全部采用 **camelCase** 方式，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 / layout/ demoStyles

3. JS、CSS、SCSS、HTML、PNG 文件命名

全部采用 **camelCase** 方式

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]

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. 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

2.缩进

缩进使用 2 个空格（一个 tab），嵌套的节点应该缩进。

3.分块注释

在每一个块状元素，列表元素和表格元素，加上 HTML 注释。注释格式

<!-- 英文或者中文 >

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

5.多个属性的 html 元素规范

多个特性的元素，占据一行过多过长时，应该分多行撰写，每个特性一行。(增强更易读)

```
<v-button
foo="foo"
bar="bar"
```

```
4.  baz="baz"
5.  box="box"
6.  >
7.  </v-button>
```

（三） CSS 规范

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：

```
1.  .fw-800 {
2.    font-weight: 800;
3.  }
4.  .red {
5.    color: red;
6.  }
```

推荐：

```
1.  .heavy {
2.    font-weight: 800;
3.  }
```

2.选择器

css 选择器中尽量避免使用标签名

3.尽量使用缩写属性

不推荐：

```
1.  padding-bottom: 2px;
2.  padding-left: 1px;
3.  padding-right: 1px;
```

```
4.  padding-top: 0;
5.  background-color:#f4f4f4;
6.  background-image: url(img.png);
7.  background-repeat: no-repeat;
8.  background-position: center;
9.  background-size: cover;
```

推荐：

```
1.  padding: 0 1px 2px;
2.  background: #f4f4f4 url(img.png) no-repeat center/cover;
```

（四）LESS/SCSS 规范

1.代码组织

按以下顺序组织，并用空行间隔

✓ @import

✓ 变量声明

✓ 样式声明

```
1.  @import "mixins/size.less";
2.
3.  @defaultTextColor: #333;
4.
5.  .page {
6.    width: 960px;
7.  }
```

（五） Javascript 规范

1.命名

1）采用小写驼峰命名 **camelCase**，代码中的命名均不能以下划线开头，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 **camelCase** 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+名词** 形式

正例：saveShopCarData /openShopCarInfoDialog

反例：save / open / show / go

增删查改，详情统一使用如下 5 个单词

add / update / delete / detail / get

校验类型函数命名请遵照以下命名规则：

can 判断是否可执行某个动作 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

函数方法常用的动词：

get 获取/set 设置/add 增加/remove 删除/create 创建/destory 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

2.代码格式

1）使用 2 个空格进行缩进

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。

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>";
```

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();

5.使用 ES6,7

强烈建议使用 ES6, ES7 的新语法，比如箭头函数、await/async，解构，let ，const，for…of 等等，这将简化你的程序，并让你的代码更加灵活和可复用。

6.括号

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

7.条件判断和循环最多三层

条件判断能使用三元表达式和逻辑运算符（短路）解决的，就不要使用条件判断，但是谨记不要写太长的三元表达式。如果超过 3 层请抽成函数，并写清楚注释。

二、Vue 项目规范

（一）Vue 编码基础

1.组件规范

1）组件名为多个单词

尽可能的为每个组件命名

组件名应该始终是多个单词组成（大于等于 2），且命名规范为 **CamelCase** 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）在注册组件的时候，全部使用 **CamelCase** 格式。

```
import MyComponent from './my-component.vue'
export default {
    components:{MyComponent}
 }
```

3）在 template 模版中使用组件，应使用 **camel-case** 模式，并且使用自闭合组件。

```
<my-component></my-component>
```

4）组件内部 import 顺序

避免出现混乱在一起的 import 排列顺序，各个类型之间用空行间隔

```
import G6Editor from '@antv/g6-editor' // 外部依赖和组件
import { mapGetters } from 'vuex'
import PermissionSet from './permission-set'

import { getToken } from 'utils'// 工具和常量
import { treeData, roleList } from './constants'

import { getUserToken } from 'api' // api 接口
import { getUserListByRoleId } from  'api'
```

5）script 标签内部 class api 结构顺序

① name

② components

③ props

④ data

⑤ computed

⑥ watch

⑦ filter

⑧ 钩子函数（钩子函数按其执行顺序）

⑨ methods

6）props 定义应该尽量详细

✓ 提供默认值

✓ 使用 type 属性校验类型

```
props: {
    greetingText:{
        type: String,
        required: true,
        default:''
    },
    userInfo:{
        type: Object,
        required: true,
        default:()=>({})
    }
}
```

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复用。

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>
```

3.必须为 v-for 设置键值 key

4.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。

5.Vue Router 规范

1）path 命名规范采用

命名规范（尽量于 vue 文件的目录结构保持一致，因为目录、文件名都是 **camelCase** ，这样很方便找到对应的文件）

2）name 命名规范采用 CamelCase

命名规范且和 component 组件名 name 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）

```
const routes = [
  {
    path: '/dataSource',  // 数据源
    component: DataSource,
    name: 'DataSource'
  },
  {
    path: '/dataSet',      // 数据集
    component:DataSet ,
    name: 'DataSet'
  },
];
```

（二） Vue 项目目录规范

1.使用 vue-cli 脚手架

使用 vue-cli3 来初始化项目。

2.目录说明

```
src                                源码目录
|-- api                            所有 api 接口
|-- assets                         静态资源，images
|-- components                     公用组件
|-- config                         配置信息
|-- constants                      常量信息，全局常量等
|-- directives                     自定义指令
|-- icons                          icon(svg)
|-- plugins                        插件，全局使用
|-- router                         路由，统一管理
|-- store                          vuex，统一管理
|-- styles                         全局公共样式
|-- views                          视图目录
|   |-- dataSet                          数据集模块
|   |-- |-- index.vue                         数据集主页面
|   |-- |-- components                        数据集模块组件
|   |-- dataSource                       数据源模块
```

1）api 目录

此目录对应后端 API 接口，请按照后端 controller 来划分子目录，一个 controller 对应一个目录。

✓ api 中的方法名字要与后端 api 的 url 保持语义高度一致性

✓ 对于 api 中的每个方法要添加注释（使用模块、接口用途）

2）assets 目录

assets 为静态资源，里面主要存放 images 等静态资源，命名格式为

images 目录下可以根据业务划分子目录，与 views 目录结构保持一致

```
assets
|-- images
|--|-- dataSet
|--|--|-- dataSet.png
|--|-- dataSource
|--|--|-- dataSource.png
```

3）icons 目录

```
icons
|-- svg      将所有 svg 文件放在该目录下
|-- index.js
```

关于 icons 字体图标外部引入

由于 UI 组件内部 iconfont 数量很少，当不能满足页面要求时，可以使用 svg 形式，具体使用方式如下

1.安装 **svg-sprite-loader**　　**npm install svg-sprite-loader -D**

2.修改 **webpack** 配置　使用 **svg-sprite-loader** 解析 **svg** 文件

```
module.exports = {
  chainWebpack: config => {
    config.module.rule('svg')
      .exclude.add(resolve('src/icons'))
      .end()
    // 设置 svg-sprite-loader 处理 icons 目录中的 svg
    config.module
      .rule('icons')
      .test(/\.svg$/)
      .include.add(resolve('src/icons'))
      .end()
```

```
      .use('svg-sprite-loader')
      .loader('svg-sprite-loader')
      .options({ symbolId: 'icon-[name]' })
      .end()
  }
};
```

3.在 **components** 目录下建立 **SvgIcon** 全局组件，在 **icons/index.js** 全局注册

```
components  // 注册 svg 全局组件
|-- SvgIcon
|--|-- index.vue
```

SvgIcon/index.vue

```
<template>
  <svg :class="svgClass" aria-hidden="true">
    <use :xlink:href="iconName"></use>
  </svg>
</template>

<script>
export default {
  name: 'SvgIcon',
  props: {
    iconClass: {
      type: String,
      required: true
    },
    className: {
      type: String
    }
  },
  computed: {
    iconName() {
      return `#icon-${this.iconClass}`
    },
    svgClass() {
      if (this.className) {
        return 'svg-icon ' + this.className
      } else {
        return 'svg-icon'
      }
    }
  }
```

```
}
</script>
```

icons/index.js

```
import Vue from 'vue'
import SvgIcon from '@/components/SvgIcon'// svg 组件

// register globally
Vue.component('svg-icon', SvgIcon)

const requireAll = requireContext => requireContext.keys().map(requireContext)
const req = require.context('./svg', false, /\.svg$/)
requireAll(req)
```

4.使用

```
<svg-icon  icon-class="file"></svg-icon>  // icon-class 为 svg 文件名
```

5.在阿里巴巴 **Iconfont** 图标库以 **svg** 形式下载需要的图标

https://www.iconfont.cn/home/index

4）components 目录

此目录下应为基础公共组件，按照组件功能进行目录划分

公共组件文件夹命名方式请采用 **CamelCase** 方式

```
components
|-- SvgIcon
|--|-- index.vue
```

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）utils 目录

```
utils
|-- index.js
|-- auth.js                                    本地存储（cookie，localStorage）
```

```
|-- data.js                                  数据处理
|-- date.js                                  日期处理（建议使用 day.js）
|-- validate.js                              校验信息（建议使用 validator.js）
|-- fetch.js                                 网络请求（axios）
...
```

7）styles 目录

```
styles
|-- index.sass                               统一管理全局样式
|-- variables.sass                           全局颜色变量
|-- dialog.sass                              弹窗样式
|-- table.sass                               表格样式
...
```

8）views 目录

对页面拆分后的组件功能比较复杂，可以建立文件夹，方便对组件进行继续拆分

```
|-- views                                    视图目录
|   |-- dataSet                              数据集模块
|   |-- |-- index.vue                            数据集主页面（对应一个 router）
|   |-- |-- components                           dataSet 组件文件夹（如果是独立的子页面请放在外面）
|   |-- |-- |-- dataSetMenulist                     数据集菜单列表
|   |-- |-- |-- |-- index.vue
|   |-- |-- |-- dataSetLineage                  数据集数据沿袭
|   |-- |-- |-- |-- index.vue
|   |-- |-- |-- dataSetLog                      数据集操作记录
|   |-- |-- |-- |-- index.vue
|   |-- |-- |-- dataSetView                     数据集数据预览
|   |-- |-- |-- |-- index.vue
|   |-- |-- |-- |-- viewTable.vue               （根据业务复杂程度对子组件继续进行二次拆分）
|   |-- |-- |-- |-- viewForm.vue
|   |-- |-- |--  dataSetVersion                 数据集版本列表
|   |-- |-- |-- |-- index.vue
```

如果拆分后的组件功能比较简单，展示类组件，比如弹窗类，可以不必建立文件夹，直接使用.vue 文件

```
|-- views                              视图目录
|   |-- dataSet                        数据集模块
|   |-- |-- index.vue                      数据集主页面（对应一个 router）
|   |-- |-- components                     dataSet 组件文件夹（如果是独立的子页面请放在外面）
|   |-- |-- |-- dataSetMenulist.vue            数据集菜单列表
|   |-- |-- |-- dataSetLineage.vue             数据集数据沿袭
|   |-- |-- |-- dataSetLog.vue                 数据集操作记录
|   |-- |-- |-- dataSetView.vue                数据集数据预览
|   |-- |-- |-- dataSetVersion.vue             数据集版本列表
```

3.注释说明

需要加注释的地方

- ✓ 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
- ✓ 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
- ✓ 多重 if 判断语句

（三） Vue 项目打包

关于 vue-cli 3.0 打包后删除 console.log

1. 使用 babel-plugin-transform-remove-console 插件

npm install babel-plugin-transform-remove-console -D

2. 在 babel.config.js 中配置

```
const plugins = ["@vue/babel-plugin-transform-vue-jsx"];
// 生产环境移除 console
if(process.env.NODE_ENV === 'production') {
  plugins.push("transform-remove-console")
}
```

```
module.exports = {
  plugins,
  presets: [
    '@vue/cli-plugin-babel/preset'
  ]
}
```

三、常用插件库

工具函数 loadash
校验插件 validator.js
网络请求 axios
日期格式化 day.js
复制粘贴插件 clipboard
前端加密解密工具 crypto-js
Cookie js-cookie
Vue 拖拽插件 vuedraggable
网页保存为 pdf 工具 html2canvas
富文本编辑器 vue-Quill-Editor tinymce
右键菜单 vue-contextmenu
数字滚动 vue-count-to
前端导出 Excel xlsx
进度条 nprogress