1. 命名规范
2. 项目命名

全部采用小写方式， 以中划线分隔。

推荐：mall-management-system

不推荐：mall\_management-system / mallManagementSystem

1. 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。

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

不推荐： script / style / demo\_scripts / demoStyles / imgs / docs

注：components 中的组件目录，使用 kebab-case 命名

如：head-search / page-loading / authorized / notice-icon

views中的组件目录，使用 kebab-case 命名

1. JS、CSS、SCSS、PNG 文件命名

全部采用小写方式， 以中划线分隔

推荐： render-dom.js / signup.css / company-logo.png

不推荐： renderDom.js

（注：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）

1. 组件规范
2. 组件名为多个单词

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。

推荐：

export default {

name: 'TodoItem'

// ...

};

不推荐：

export default {

name: 'Todo',

// ...

}

export default {

name: 'todo-item',

// ...

}

1. 组件文件名为 pascal-case 格式

如：head-search.vue

1. 基础组件文件名为 base 开头，使用完整单词而不是缩写

如：

components/

|- base-button.vue

|- base-table.vue

|- base-icon.vue

1. 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

如：

components/

|- todo-list.vue

|- todo-list-item.vue

|- todo-list-item-button.vue

|- user-profile-options.vue （完整单词）

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

如：

<!-- 在单文件组件、字符串模板和 JSX 中 -->

<TodoList />

<Row><table :column="data"/></Row>

1. 组件的 data 必须是一个函数。
2. 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

如：

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

}

}

1. 为组件样式设置作用域

<!-- 使用 `scoped` 特性 -->

<style scoped>

.btn-close {

background-color: red;

}

</style>

1. 如果特性元素较多，应该主动换行

如：

<TodoList

:number=”2”

:row=”4”

@click=”getName”

/>

1. 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。

1. 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:)

1. 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先template，然后script，最后style。

推荐：

<template>...</template>

<script>...</script>

<style>...</style>

不推荐：

<template>...</template>

<style>...</style>

<script>...</script>

1. 必须为 v-for 设置键值 key
2. 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。

1. 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

1. Router规范
2. 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。

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、children属性命名规范采用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）

注：name属性命名采用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）

1. 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：

{

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')

}

}

1. 目录规范
2. 目录整体结构

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  
 |-- utils 公共方法目录

|-- themes 自定义样式主题

|-- styles 公共样式、全局样式。

|-- 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

tests 单元测试、组件测试目录

（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。

如：

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)

}

1. constants 目录

此目录存放项目所有常量，常量定义要用大写

如：

export const EMPLOYEE\_STATUS = {

NORMAL: {

value: 1,

desc: '正常'

},

DISABLED: {

value: 1,

desc: '禁用'

}

}

1. router 与 store 目录

业务逻辑时候，两个目录需要根据业务拆分目录，切记不要把所有router或store堆在一个文件。目录名和views中的目录保持一致。

1. 关于注释（**很重要！**）

模块、变量、方法声明必须要加注释！

业务逻辑复杂的代码块也要加上必要的注释！

1. 关于垃圾代码

垃圾代码、无用代码、用于测试debugger的代码在提交前一定要删掉！