前端规范

# 通用规范

## 缩进

使用4个空格作为代码缩进

## 命名

使用具有详细语义的名称，不要使用简单语义

<!-- good -->

<div class="sidebar"></div>

<!-- bad -->

<div class="left"></div>

<!-- good -->

const userList = new Array();

<!-- bad -->

const arr = new Array();

# JS规范

## 命名规范

1. 变量、函数、类使用驼峰式命名
2. 类名首字母使用大写字母
3. 不要使用前置和后置下划线
4. 缩写要么使用全大写要么使用全小写

## 字符串

1. 字符串拼接使用字符串模版`${}`
2. 禁止使用eval
3. 尽量少使用反斜杠“\”

## 解构赋值

解构保存了这些属性的临时值/引用

用对象的解构赋值来获取和使用对象某个或多个属性值

// bad

function getFullName(user) {

const firstName = user.firstName;

const lastName = user.lastName;

return `${firstName} ${lastName}`;

}

// good

function getFullName(user) {

const { firstName, lastName } = user;

return `${firstName} ${lastName}`;

}

// best

function getFullName({ firstName, lastName }) {

return `${firstName} ${lastName}`;

}

## 箭头函数

回调使用箭头函数（上下文统一，不需要自定义保存this，代码更简洁）

[1, 2, 3].map((number) => {

const y = x + 1;

return x \* y;

})

函数体由一个没有副作用的表达式，删除大括号和return（更简洁）

[1, 2, 3].map((number) => number \* 2)

参数总是放在圆括号里（保持代码清晰）

// bad

[1, 2, 3].map(x => x \* x);

// good

[1, 2, 3].map((x) => x \* x);

## 类

使用class声明，不使用prototype(使用class语义更清晰)

// bad

function Queue(contents = []) {

this.queue = [...contents];

}

Queue.prototype.pop = function () {

const value = this.queue[0];

this.queue.splice(0, 1);

return value;

};

// good

class Queue {

constructor(contents = []) {

this.queue = [...contents];

}

pop() {

const value = this.queue[0];

this.queue.splice(0, 1);

return value;

}

}

使用extends进行继承（简单，内置方法）

## 变量

每个变量都用一个 const 或 let，一行声明一个变量，尽量避免使用var

没有重复赋值的变量统一使用const

业务变量提取成枚举常量

if (user.role === 'master') {

// TODO:  
}

const SYS\_ROLE = {

"MASTER": "master",

"NORMAL": "normal"

}

if (user.role === SYS\_ROLE.MASTER) {

// TODO:

}

## 运算符

相等和不等使用 === 或 !== 而不是 == 或 !=

简单三元表达式放在一行，JSX中?和: 放在一行的开始

## 注释

多注释，特别是业务复杂的逻辑点一定要注释

写具体业务之前先写注释

函数和类使用/\*\* ...\*/进行注释，html使用<!-- -->注释，其他的使用//注释// 后面带一个空格

# css规范

## 命名规范

1. 全小写
2. 分词使用中杠-

/\* bad \*/

.loginBtn

/\* bad \*/

.Loginbtn

/\* good \*/

.login-btn

## 空格

1. 使用4个空格作为代码缩进
2. 选择器与{之间一个空格
3. 属性和:之间不能有空格，和值之间留一个空格
4. +, > , ~ 选择器两遍各留一个空格

.selector {

margin: 0;

padding: 0;

}

/\* 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;

}

## 选择器

1. 多选择器并列的需换行书写

/\* good \*/

.post,

.page,

.comment {

line-height: 1.5;

}

/\* bad \*/

.post, .page, .comment {

line-height: 1.5;

}

# Vue规范

## 引号

js代码使用单引号，JSX使用双引号、vue的template使用双引号

## 命名规范

自定义组件使用大写驼峰式命名

自定义组件需要具名name属性

## v-for

性能考虑循环必须加上key，通常使用id作为key

## style

业务页面的style标签加上scoped

# Git规范

## 分支

分支使用多分支

M域：

1. 线上分支（稳定分支）： master/common\_dev
2. 开发/测试分支： develop/test（只有测试环境时取其一即可）
3. 需求分支： feature\_jira号\_任务描述\_日期

例如feature\_IMIP718\_k8s\_2024018.

1. 发布分支：当测试分支test达到发布计划的要求时，建立分支release\_版本号\_日期。例如release\_v9.7.0\_2024018.
2. 线上修复分支：hotfix\_jira号\_任务描述\_日期

例如hotfix\_IMIP690\_k8s\_2024018

注意： 3， 4， 5 种情况， 任务上线后请记得删除分支

B域：

master（线上分支/测试分支）

develop(开发分支)

使用git-flow，对新增功能和bug修复使用feature分支，命名feature/功能-姓名或fix/功能-bug号

## 提交

commit message：

标题（必填）格式：[type]: [version] [title]

OCT:feat: 新功能、新特性

OCT:fix: 修改 bug

OCT:perf: 更改代码，以提高性能（在不影响代码内部行为的前提下，对程序性能进行优化）

OCT:refactor: 代码重构（重构，在不影响代码内部行为、功能下的代码修改）

OCT:docs: 文档修改

OCT:style: 代码格式修改, 注意不是 css 修改（例如分号修改）

OCT:test: 测试用例新增、修改

OCT:build: 影响项目构建或依赖项修改

OCT:revert: 恢复上一次提交

OCT:ci: 持续集成相关文件修改

OCT:chore: 其他修改（不在上述类型中的修改）

OCT:release: 发布新版本

例子：

OCT:feat: v0.1.0 添加login页面

OCT:refactor: v0.1.1 xxx功能前后端联调

OCT:perf: v0.1.0 优化内存泄漏问题

OCT:fix: v0.1.0 修复无法正确显示列表问题（复杂内容写在body中）

1、造成问题的原因

2、修复过程

OCT:build: v0.1.0构建代码，触发流水线

OCT:chore: v0.1.0 更新xx依赖至vxx.xx.xx版本

## 安装Husky

Husky 是一个用于前端开发的 Node.js 工具，它主要用于在 Git 提交钩子（Git Hooks）中运行脚本。Git 提交钩子是在 Git 版本控制系统的特定事件发生时执行的自定义脚本，例如在代码提交前执行一些**检查或验证**。

yarn add husky

修改package.json

{

"scripts": {

...

"prepare": "husky",

"postinstall": "shx rm -rf .git/hooks && shx ln -s ../.husky .git/hooks" // 支持GUI报错问题

}

}

初始化

npx prepare

在项目目录下新建./husky/\_目录，初始化脚本

npx husky init

在.husky目录下创建一个测试pre-commit文件

可以在文件中添加node脚本

在pre-commit中添加eslint校验

#!/bin/bash

npx lint-staged

## 安装lint-staged

lint-staged可以只检查修改的文件而不会扫描项目所有文件，只针对git stage中的文件进行修改

yarn add lint-staged

修改package.json添加lint-staged配置信息

{

"lint-staged": {

"\*.{js,cjs,jsx,vue,ts,tsx,html,vue,css,sass,less}": "eslint --cache --fix"

}

}

此刻 执行git commit -m "xxxx"， 就会触发 lint-staged 脚本对文件进行自动修复

## 提交内容校验

### 安装 commitizen 和 commitlint

yarn add commitizen @commitlint/cli @commitlint/config-conventional -D

修改package.json

{

"scripts": {

...

"cz": "cz",

}

}

添加配置文件commitlint.config.cjs

module.exports = {

extends: ['@commitlint/config-conventional'],

rules: {

'type-enum': [

2,

'always',

['build', 'chore', 'ci', 'docs', 'feat', 'fix', 'perf', 'refactor', 'revert', 'style', 'test', 'types'],

],

},

};

# H5/移动端适配规范

1. 视口设置：在HTML头部中设置合适的viewport，保证页面在不同设备上都能正确显示页面布局。
2. 采用Flex布局，并熟练掌握flex-direction、justify-content、align-items、flex-wrap等属性。
3. 字体大小：字体大小应使用相对大小（em），而不是绝对大小（px）

例如，body的字体大小可以设置为正常大小100%（16px），h1的字体大小为默认大小的1.5倍（24px），small元素的字体大小为默认大小的0.875倍（14px）。

响应式设计：使用媒体查询根据不同的屏幕尺寸调整样式，以确保页面在各种设备上都能正确显示

# 依赖包

在package.json中

代码使用的依赖包放在dependencies中，构建相关的放在devDependencies中

version和当前业务版本一致，方便运维人员部署后验证

# 前后端接口规范

## 普通接口

请求头：统一使用json格式

{

"Content-Type": "application/json"

"AUTH(自定义)": "TOKEN"

}

请求体：使用payload（JSON）

返回体(不分页)：

{

code: 200|403|404|500,

message: "错误信息",

data: []|{}

}

返回体(分页)：

{

code: 200|403|404|500,

message: "错误信息",

data: [],

total: 100,

pageNum: 1,

pageSize: 10

}

## 下载文件接口

请求头：统一使用流格式格式

{

"Content-Type": "application/octet-stream"

"AUTH(自定义)": "TOKEN"

}

返回头：

{

"x-file": "fileName.docx(转义后)"

}

# 代码规范校验

代码规范使用eslint进行校验，可配置vscode的插件方便显示错误和自动修复

## 安装插件

安装eslint2.4.4或以上版本

// vscode的setting.json配置修改添加如下配置

// 保存自动格式化

"editor.formatOnType": false, // required

"editor.formatOnPaste": true, // optional

"editor.formatOnSave": true, // optional

// 支持的文件类型

"eslint.validate": [

"javascript",

"javascriptreact",

"typescript",

"typescriptreact",

"vue"

],

"eslint.format.enable": true,

"[typescriptreact]": {

"editor.defaultFormatter": "dbaeumer.vscode-eslint"

},

"[typescript]": {

"editor.defaultFormatter": "dbaeumer.vscode-eslint"

}

## 安装依赖(vue3 + ts)

yarn add eslint eslint-config-standard-with-typescript eslint-plugin-import eslint-plugin-n eslint-plugin-promise eslint-plugin-vue @typescript-eslint/eslint-plugin -D

注意：

需要禁用prettier插件，与eslint会有冲突

## ESLINT配置

根目录新建.eslintrc.js文件

module.exports = {

env: {

browser: true,

es2021: true,

node: true

},

extends: ['standard-with-typescript', 'eslint:recommended', 'plugin:vue/vue3-essential'],

overrides: [

{

env: {

node: true

},

files: ['.eslintrc.{js,cjs}'],

parserOptions: {

sourceType: 'script'

}

}

],

parser: 'vue-eslint-parser',

parserOptions: {

parser: '@typescript-eslint/parser',

ecmaVersion: 'latest',

sourceType: 'module',

ecmaFeatures: {

jsx: true

},

extraFileExtensions: ['.vue'],

project: ['./tsconfig.json', 'src/components/HeaderNav.vue']

},

plugins: ['vue'],

rules: {

'no-console': process.env.NODE\_ENV === 'production' ? 'warn' : 'off',

'no-debugger': process.env.NODE\_ENV === 'production' ? 'warn' : 'off',

'space-before-function-paren': 1, // 函数定义时括号前面要有空格

'no-unused-vars': 'off', // 不能有声明后未被使用的变量或参数

'comma-dangle': ['error', 'only-multiline'],

'@typescript-eslint/space-before-function-paren': 1,

'@typescript-eslint/no-unused-vars': 'off',

'@typescript-eslint/indent': ['error', 4],

'@typescript-eslint/semi': ['error', 'always'],

'@typescript-eslint/explicit-function-return-type': 'off',

'@typescript-eslint/prefer-nullish-coalescing': 'off',

'@typescript-eslint/strict-boolean-expressions': 'off',

'@typescript-eslint/no-unsafe-argument': 'off',

'@typescript-eslint/no-explicit-any': 'off',

'@typescript-eslint/no-namespace': 'off',

// 关闭使用前必须定义

'@typescript-eslint/no-use-before-define': 'off',

// 空接口

'@typescript-eslint/no-empty-interface': 'off',

// 不能使用for in

'@typescript-eslint/no-for-in-array': 'off',

// 强制加await

'@typescript-eslint/no-floating-promises': 'off',

// 可以使用断言

'@typescript-eslint/consistent-type-assertions': ['error', { assertionStyle: 'as' }],

// 强制在单行元素的内容前后加换行符

'vue/singleline-html-element-content-newline': 'off',

// 该规则强制在多行元素的内容前后换行

'vue/multiline-html-element-content-newline': 'off',

// 强制属性默认值default

'vue/require-valid-default-prop': 'off',

// 组件名称需要多个单词

'vue/multi-word-component-names': 'off',

// 强制驼峰法命名

camelcase: [

2,

{

properties: 'always'

}

],

// 关闭箭头函数不能有返回值

'no-return-assign': 'off',

// JSX 属性中一致使用双引号或单引号

'jsx-quotes': [2, 'prefer-double'],

// 对象字面量中冒号的前后空格

'key-spacing': [

2,

{

beforeColon: false,

afterColon: true

}

],

// 在关键字前后强制使用一致的空格

'keyword-spacing': [

2,

{

before: true,

after: true

}

],

// 在调用没有参数的构造函数时强制或禁止使用圆括号

'new-parens': 2,

// 不允许“数组”构造函数

'no-array-constructor': 2,

// 不允许使用“arguments.caller”或“arguments.callee”

'no-caller': 2,

// 在条件表达式中禁止赋值操作符

'no-cond-assign': 2,

// 禁止对const变量重新赋值

'no-const-assign': 2,

// 在“函数”定义中禁止重复参数

'no-dupe-args': 2,

// 禁止使用' eval()

'no-eval': 2,

// 不允许多行空格

'no-multi-spaces': 2,

// 禁止使用' \_\_proto\_\_ '属性

'no-proto': 2,

// 不允许逗号操作符

'no-sequences': 2,

// 要求或不允许函数标识符与其调用之间有空格+

'func-call-spacing': 2,

// 在构造函数中调用super()之前禁止使用' this ' / ' super '

'no-this-before-super': 2,

// 禁止行尾有空格

'no-trailing-spaces': 2,

// 在' return '， ' throw '， ' continue '和' break '语句之后出现不可到达的代码，进行警告

'no-unreachable': 1

}

};

添加ignore, 创建.eslintignore

node\_modules

dist

\*.md

vite.config.ts