# 前端开发规范

目录

[前端开发规范 1](#_Toc12052)

[（一）前言及简介 2](#_Toc1913)

[（二）环境搭建 2](#_Toc6029)

[2.1 安装依赖 2](#_Toc4468)

[2.1.1 生产环境依赖 2](#_Toc906)

[2.1.2 开发环境依赖 3](#_Toc13144)

[2.2 webpack配置 3](#_Toc24941)

[2.2.1 运行端口 3](#_Toc28575)

[2.2.2 代理 3](#_Toc18663)

[（三）项目规范 3](#_Toc2541)

[3.1 项目技术构成 3](#_Toc5872)

[3.2 项目结构规范 4](#_Toc7059)

[3.2.1 项目结构的核心思想 4](#_Toc30419)

[3.2.2 项目目录 4](#_Toc9482)

[3.3 资源命名 5](#_Toc10287)

[3.3.1 目录命名 5](#_Toc19540)

[3.3.2 页面命名 5](#_Toc4694)

[3.3.3 组件命名 6](#_Toc14189)

[3.3.4 图片命名 6](#_Toc30594)

[3.4 资源使用 7](#_Toc11051)

[3.4.1 组件使用规范 7](#_Toc28327)

[3.4.2 图片使用规范 8](#_Toc6722)

[3.5 项目路由 8](#_Toc16627)

[3.5.1 路由命名 8](#_Toc27976)

[3.5.3 多级路由 9](#_Toc28734)

[3.5.4 路由权限 9](#_Toc21481)

[（四）编码规范 9](#_Toc5618)

[4.1 HTML规范 9](#_Toc12460)

[4.1.1 文件编码 9](#_Toc1584)

[4.1.2 外链资源URL协议 10](#_Toc31738)

[4.1.3 标签中的属性顺序 11](#_Toc16714)

[4.1.4 语义化 11](#_Toc8262)

[4.1.5 引号规范 12](#_Toc19900)

[4.1.6 注释规范 12](#_Toc8326)

[4.2 CSS规范 12](#_Toc10067)

[4.2.1 css初始化及公用样式 12](#_Toc9052)

[4.2.2 id class命名规则 18](#_Toc5153)

[4.2.3 属性 18](#_Toc4256)

[4.2.4 媒体查询 19](#_Toc30313)

[4.2.5 注释规范 19](#_Toc7213)

[4.2.6 less规范 19](#_Toc10931)

[4.2.7 其他规范 20](#_Toc12316)

[4.3 JavaScript编码规范 20](#_Toc15262)

[4.3.1 缩进 20](#_Toc31189)

[4.3.2 空格 20](#_Toc16686)

[4.3.3 空行 21](#_Toc14487)

[4.3.4 引号 21](#_Toc12204)

[4.3.5 模板字符串 21](#_Toc8560)

[4.3.6 命名规范 22](#_Toc14922)

[4.3.7 二元三元操作符 23](#_Toc3000)

[4.3.8 && 和 || 23](#_Toc20526)

[4.3.9 声明规范 24](#_Toc29314)

[4.3.10 分号 24](#_Toc11920)

[4.3.11 注释规范 24](#_Toc31412)

[4.4 ESlint语法检测 25](#_Toc6415)

[（五） 代码提交规范 25](#_Toc27287)

[（六）版本管理 26](#_Toc4459)

[（七）部署 26](#_Toc10008)

# （一）前言及简介

没有规矩，不成方圆，对于团队来说，有一套完整的开发规范可以减少团队协作成本和维护成本，让代码阅读起来更容易。通过规范化代码，使代码统一化具有更好的可读性，也减少了后期代码的维护以及团队协作沟通的成本。

# （二）环境搭建

## 2.1 安装依赖

项目中所需要的依赖都会在package.json文件中展示。

安装依赖需要使用npm命令，可去[npm官方文档](https://link.juejin.cn?target=https://www.npmjs.cn/getting-started/installing-node/" \o "https://www.npmjs.cn/getting-started/installing-node/" \t "_blank)使用及学习。

### 2.1.1 生产环境依赖

生产环境需要的依赖安装在属性dependencies之下。

可以使用命令：

npm install xxxx --save 复制代码

不可安装生产环境无需用到的依赖。

### 2.1.2 开发环境依赖

开发环境需要的依赖安装在属性devDependencies之下。

可以使用命令：

npm install xxxx --save-dev 复制代码

## 2.2 webpack配置

### 2.2.1 运行端口

项目开发环境下的运行端口为：

port: 9

### 2.2.2 代理

如果项目中需要使用其他系统请求，涉及跨域问题，可按照如下修改代理配置。

proxy: {  
 [process.env.VUE\_APP\_BASE\_API ]: {  
 target: 'http://localhost:3000',  
 changeOrigin: true,  
 pathRewrite: {  
 '^/api:':''  
 }  
 }  
}

# （三）项目规范

## 3.1 项目技术构成

本项目由vue.js技术框架搭建，配合webpack打包。

[vue.js官方文档](https://link.juejin.cn?target=https://cn.vuejs.org/v2/guide/) 学习及使用。

[webpack官方文档](https://link.juejin.cn?target=https://www.webpackjs.com/) 学习及使用。

## 3.2 项目结构规范

### 3.2.1 项目结构的核心思想

* 业务功能模块的相关代码都集中在一块，方便移动和删除
* 实现关注点分离，方便开发、调试、维护、编写、查阅、理解代码

### 3.2.2 项目目录

**├── vue.config.js/** **# webpack 配置文件**

**├── config/ # 与项目构建相关的常用的配置选项**

**│ ├── index.js # 主配置文件**

**│ ├── dev.env.js # 开发环境变量**

**│ ├── prod.env.js # 生产环境变量**

**│ └── test.env.js # 测试环境变量**

**├── public/ # 纯静态资源**

**├── node\_modules/ # 构建配置相关**

**├── src/ # 源代码**

**│ ├── api/ # 请求接口文件**

**│ ├── assets/ # 静态资源**

**│ ├── components/ # 全局公共组件**

**│ │ ├── base/ # 基本组件**

**│ │ ├── common/ # 共用的全局组件**

**│ │ ├── temp/ # 模板组件**

**│ │ └── UItemp/ # UI组件**

**│ ├── css/ # 全局样式**

**│ ├── pages/ # 单页页面**

**│ ├── router/ # 路由模块**

**│ ├── store/ # 全局 store管理**

**│ ├── services/ # 全局网络模块**

**│ ├── views/ # vue路由组件**

**│ ├── styles/ # 全局公用样式**

**│ ├── App.vue # 入口根组件**

**│ ├── main.js # 入口文件**

**├── .babelrc # babel 的配置文件**

**├── .editorconfig # 编辑器的配置文件；**

**├── .eslintrc.js # eslint 的配置文件**

**├── .eslintignore # eslint 的忽略规则**

**├── .gitignore # git的忽略配置文件**

**├── .postcssrc.js # postcss 的配置文件**

**├── index.html # HTML模板**

**├── package.json # npm包配置文件**

**└── README.md # 项目信息文档复制代码**

## 3.3 资源命名

### 3.3.1 ****目录命名****

* 能直观的感受当前目录文件的作用
* 以小驼峰方式命名
* 特殊缩写名称可大写开头

│ ├── pages/

│ ├── components/

│ │ ├── UItemp/

### 3.3.2 ****页面命名****

* 能直观的感受当前文件的作用
* 以小驼峰方式命名

│ │ │ ├── login.vue

│ │ │ ├── changePhone.vue

### 3.3.3 ****组件命名****

* 能直观的感受当前组件的用途
* 组件命名始终是多个单词的，避免跟html元素或标签冲突
* 要么大写开头，要么始终用横线链接（这里项目大写开头）

// 反例

components/

|- MyButton.vue

|- VueTable.vue

|- Icon.vue

// 正例

components/

|- BaseButton.vue

|- BaseTable.vue

|- BaseIcon.vue复制代码

### 3.3.4 ****图片命名****

* 图片文件夹一般遵从页面或者模块命名,如：login/）
* 图片不可随意命名，且严禁使用0，1，等数字直接命名图片。
* 图片命名可遵循：用途+描述，多个单词用下划线隔开，如：login\_icon.png,pwd\_icon.png
* 10k以下图片建议放置assets/img下（webpack打包时直接转为base64嵌入）
* 大图且不常更换的图片放置public/img下
* 可用css编写的样式严禁使用图片
* 国际化图片，后缀使用简体-cn,英文-en,繁体-tw

│ ├── assets/

│ │ ├── img/

│ │ │ ├── common/

│ │ │ │ ├── default\_avatar.png

│ │ │ ├── login/

│ │ │ │ ├── login1.png

│ │ │ │ ├── login\_icon-en.png

│ │ │ │ ├── login\_icon-cn.png

│ │ │ │ ├── login\_icon-tw.png

│ │ │ ├── userInfo/

## 3.4 资源使用

### 3.4.1 ****组件使用规范****

* 使用时以v-开头
* 命名遵循组件命名规范
* 推荐使用单标签闭合

<v-BaseButton :data="data"/><script>

export default{

components:{

"v-BaseButton":BaseButton

}

}

</script>

### 3.4.2 ****图片使用规范****

项目下有两个文件夹放置图片地址src/assets/img,public/img

webpack打包时会直接copy public/img下的图片，所以，当你的图片放置public/img下时，为防止图片打包重复，需要遵从以下写法：

* public/img**下的图片调用时禁止使用require**

因为通过require调用的图片会经过webpack打包，生成hash后缀，同`public/img`下图片重复

let img1 = require('../public/img/xxxx')//错误写法

let img2 ='../public/img/xxxx'//正确写法复制代码

* public/img**下的图片不可直接嵌入img的src中**

因为img标签src中的图片也会经过webpack打包

<!--错误写法--><img src="../public/img/xxx">

<!--正确写法--><img :src="`../public/img/xxx`">

* src/assets/img**下的图片必须使用require写法，否则会报错找不到图片**

<!--正确写法--><img src="../asset/img/xxx">

<!--错误写法--><img :src="`../asset/img/xxx`">

let img1 = require('../asset/img/xxxx')//正确写法

let img2 ='../asset/img/xxxx'//错误写法复制代码

图片这样区分，防止图片重复打包，增加项目体积

## 3.5 项目路由

### 3.5.1 路由命名

* 普通路由(非动态多级)命名，可以直接使用页面组件的命名。

{

path: '/login',

name: 'login',

component(resolve) {

require.ensure(['../pages/login/login.vue'], () => {

resolve(require('../pages/login/login.vue'));

});

}

}

* 动态多级路由，遵循：用途或作用或功能。

/user/personal/infomaition

/user/company/infomaition

### 3.5.3 多级路由

* 嵌套不可超过三层。

### 3.5.4 路由权限

#### 3.5.4.1 ****登录权限****

一般页面无需登录可查看，重要页面需要登录权限。

登录权限字段在路由meta标签中确定。

meta: {

requireAuth: true // 为true则是需要登录权限

}

#### 3.5.4.2 管理员权限

管理员拥有的权限：

* 显示管理入口，进入后台管理。
* 导航结构展示不一样。
* 与普通用户相同的路径，会展示不一样的内容。

根据不同的权限，渲染不同的组件或页面。

# （四）编码规范

## 4.1 HTML规范

### 4.1.1 ****文件编码****

* 开头声明文档类型
* 使用语言属性
  + 中文："zh-Hans"
  + 简体中文："zh-cmn-Hans"
  + 繁体中文："zh-cmn-Hant"
  + 英文："en"
* 使用不带BOM的UTF-8编码
* 在HTML中指定编码<meta charset="utf-8">
* SEO优化
* viewport是否可缩放(为移动设备优化)
* favicon：如果未指定，大多数浏览器会请求根目录下的favicon，为避免404错误，两种解决方法，一是在根目录放置favicon.ico文件，二是使用link指定favicon文件

<!DOCTYPE html><html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<!-- SEO -->

<title>Style Guide</title>

<meta name="keywords" content="your keywords">

<meta name="description" content="your description">

<meta name="author" content="author,email address">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="shortcut icon" href="path/to/favicon.ico">

</head></html>

### 4.1.2 ****外链资源URL协议****

省略外链资源（图片及其它媒体资源）URL 中的 http / https 协议，使 URL 成为相对地址，避免Mixed Content 问题，减小文件字节数。Mixed Content：http和https混用造成的问题。其它协议（ftp 等）的 URL 不省略。

<!-- 推荐 -->

<script src="//www.xxx.cn/statics/js/autotrack.js"></script>

<!-- 不推荐 -->

<script src="http://www.xxx.cn/statics/js/autotrack.js"></script>

/\* 推荐 \*/.example {

background: url(//www.google.com/images/example);

}

/\* 不推荐 \*/.example {

background: url(http://www.google.com/images/example);

}

### 4.1.3 ****标签中的属性顺序****

* class(class 是最高的复用设计，应该放在最前)
* id name(id 应尽量少用)
* data-\\\* 自定义属性（属性名称全小写用-做连接）
* src (资源文件)
* placeholder title alt (提示)
* required readonly disable(辅助)

### 4.1.4 ****语义化****

语义的HTML结构，有助于机器搜索，保证代码可读性。

常见的标签语义

| **标签** | **语义** |
| --- | --- |
| <p> | 段落 |
| <h1,2,3> | 标题 |
| <ul> | 无序列表 |
| <ol> | 有序列表 |
| <blockquote> | 大段引用 |
| <cite> | 一般引用 |
| <b> | 为样式加粗 |
| <strong> | 为强调加粗 |
| <i> | 为样式倾斜 |
| <em> | 为强调倾斜 |
| <code> | 代码标识 |
| …… | …… |

例如：

* h1 大标题，一般用于banner背景，一个页面有且只有一个
* h2 章节标题，可以有多个
* h3 章节内的文章标题
* h4,h5,h6 小标题/副标题
* p 段落

├── h1---文章大标题

│ ├── h2--- 这里是一个节点

│ │ ├──h3--- 节点内的文章标题

│ │ │ ├── h4,h5,h6--- 副标题/作者等信息

│ │ │ ├── p---段落

│ ├── h2--- 这里是一个节点

│ │ ├──h3--- 节点内的文章标题

│ │ │ ├── h4,h5,h6--- 副标题/作者等信息

│ │ │ ├── p---段落

### 4.1.5 ****引号规范****

* 属性值使用双引号包裹

<div :data="data" data-src="some"></div>

### 4.1.6 ****注释规范****

<!-- 头部 --><view class="header">

<!-- LOGO -->

<image class="logo"></image>

<!-- /LOGO -->

<!-- 详情 -->

<view class="detail"> </view>

<!-- /详情 --></view><!-- /头部 -->

## 4.2 CSS规范

### 4.2.1 ****css初始化及公用样式****

/\* css 初始化 \*/

/\* margin padding 初始化为0 \*/

body, h1, h2, h3, h4, h5, h6, hr, p, blockquote, dl, dt, dd, ul, ol, li, pre, form, fieldset, legend, button, input, textarea, th, td {

margin: 0;

padding: 0;

}

body, button, input, select, textarea {

font: 12px/1.5tahoma, arial, \5b8b\4f53;

}

h1, h2, h3, h4, h5, h6 { font-size: 100%;}address, cite, dfn, em, var { font-style: normal;}

code, kbd, pre, samp {font-family: couriernew, courier, monospace;}

small {font-size: 12px;}

ul, ol {list-style: none;}

a {text-decoration: none;}

a:hover {text-decoration: none;}

sup {vertical-align: text-top;}

sub {vertical-align: text-bottom;}

legend {color: #000;}

fieldset, img {border: 0;}

button, input, select, textarea {font-size: 100%;}

table {

border-collapse: collapse;

border-spacing: 0;

}

html, body {font-size: 10px !important;}

flex布局共用样式类，字体大小样式类，margin,padding类：

/\* flex布局共用样式类，字体大小样式类，margin,padding

(如有需求，可自行添加) \*/

.yl-flex{

dispaly:flex;

}

.flex-center-center{

justify-content: center;

justify-items: center;

align-items: center

}

.flex-start-center{

justify-content: start;

justify-items: center;

align-items: center

}

.flex-end-center{

justify-content: end;

justify-items: center;

align-items: center

}

.flex-wrap{

flex-wrap: wrap;

}

.flex-column{

flex-direction:column;

}

.flex-space-around{

justify-content: space-around;

}

/\* 文本 \*/.tl {text-align: left;}

.tr { text-align: right;}

.tc { text-align: center;}

/\* 按钮 \*/.btn {

display: inline-block;

}

.btn-success {

background: @success;

}

.btn-error {

background: @error;

}/\* 按钮禁用 \*/.disabled {

outline: 0 none;

cursor: default !important;

opacity: .4;

filter: alpha(opacity=40);

-ms-pointer-events: none;

pointer-events: none;

}

/\* 浮动与清除浮动 \*/.fl {

float: left;

}

.fr {

float: right;

}

.fix {

\*zoom: 1;

}

.fix:after {

display: table;

content: '';

clear: both;

}

/\* 显示 \*/ .dn {

display: none;

}

.di {

display: inline;

}

.db {

display: block;

}

.dib {

display: inline-block;

}

.dt {

display: table;

}

/\* 定位 \*/ .pr {

position: relative;

}

### 4.2.2 id class****命名规则****

* 首先根据内容命名，如：nav,header
* 内容中的子元素使用-链接，名称一律小写，如：card-item
* 修饰类（易变的）使用--链接，如：card-item--warning
* 若无内容，结合行为进行辅助，如：box-shawder
* 不影响语义的情况下，可缩写，如：img-box,btn
* 避免广告拦截词汇：ad,gg,banner,guagngao

### 4.2.3 ****属性****

**属性顺序**

* 位置属性(position,top,right,z-index,display,flaot)
* 大小(width,height,padding,margin)
* 文字(font,line-height,letter-spacing, color)
* 背景 (background,border)
* 其他(animation,transition)

**属性简写**

属性简写需要你非常清楚属性值的正确顺序，而且在大多数情况下并不需要设置属性简写中包含的所有值，所以建议尽量分开声明会更加清晰；

/\* not good \*/.element {

transition: opacity 1s linear 2s;

}

/\* good \*/.element {

transition-delay: 2s;

transition-timing-function: linear;

transition-duration: 1s;

transition-property: opacity;

}

margin 和 padding 相反，需要使用简写；

### 4.2.4 ****媒体查询****

尽量将媒体查询的规则靠近与他们相关的规则，不要将他们一起放到一个独立的样式文件中，或者丢在文档的最底部，这样做只会让大家以后更容易忘记他们。

.element-selected { ... }

@media (max-width: 768px) {

.element-selected { ... }

}

### 4.2.5 ****注释规范****

* 组件块和子组件之间的注释

/\*组件块\*/

/\* 子组件块\*/

.selector {

padding: 15px;

margin-bottom: 15px;

}

/\* 子组件块\*/

.selector-secondary {

display: block; /\* 注释\*/

}

* 单行注释/\*后跟空格

/\* xxx \*/

### 4.2.6 ****less规范****

* 组织顺序
  + - @import
    - 变量声明
    - 样式声明
* 避免嵌套过多，将嵌套深度限制在3级，超过需重新评估

@import "mixins/size.less";

@default-text-color: #333;

.page {

width: 960px;

margin: 0 auto;

}

### 4.2.7 ****其他规范****

* 去掉小数点后面的0，如：rgba(0,0,0,0.5)=>rgba(0,0,0,.5)
* 颜色代码尽量简写，如：#fff
* 尽量少用元素选择器，元素选择器和 ID、Class 混合使用也违反关注分离原则。如果HTML标签修改了，就要再去修改 CSS 代码，不利于后期维护。

/\* 不推荐 \*/

.red {}.box\_green {}.page .header .login #username input {}ul#example {}

/\* 推荐 \*/

#nav {}.box-video {}#username input {}#example {}

## 4.3 JavaScript编码规范

### 4.3.1 ****缩进****

html,css,js缩进一致，使用4个空格。

### 4.3.2 ****空格****

* 二元运算符前后
* 三元运算符前后
* 代码块{前
* 关键字前：else, while, catch, finally
* 关键字后：if, else, for, while, do, switch, case, catch, finally, with, return
* 单行注释//后，多行注释/\*后
* 对象的属性值前
* for 循环
* 函数的参数之间
* 运算符前后
* 函数声明，函数表达式(前不要空格，)后空格

### 4.3.3 ****空行****

* 变量声明后
* 注释前
* 代码块后
* 文件最后保留一个空行

let string = 'my'

let obj = {

'name': 'yang'

}

{

// do something

}

/\* 正例 \*/

if {

} else {

}

### 4.3.4 ****引号****

* 最外层统一使用单引号。

let str = 'my name is'

### 4.3.5 ****模板字符串****

* 需要拼接，尽量少用+，多使用模板字符串

let str1 = 'yang'

let string = `hello ${str1}`

### 4.3.6 ****命名规范****

* 变量命名：小驼峰命名
* 参数名：小驼峰命名
* 函数名：小驼峰命名
* 方法/属性名：小驼峰命名
* 类名开头大写
* 私有属性、变量和方法以下划线 \_ 开头。
* 常量名：全部使用大写+下划线
* 由多个单词组成的缩写词，在命名中，根据当前命名法和出现的位置，所有字母的大小写与首字母的大小写保持一致。
* 语义
  + 变量应当使用名词，尽量符合当时语义
  + boolean类型应当使用 is , has 等开头
  + 点击事件命名方式：tap + onClick()

let loadingModules = {};

const HTML\_ENTITY = {};

function stringFormat(theBells) {}

function insertHTML(element, html) {}

function Engine(options) {}

### 4.3.7 ****二元三元操作符****

* 可用于简单的if替代
* 操作符始终写在前一行，避免产生预想不到的问题

// 不推荐 var x = a ? b : c;

// 推荐 var y = a ?

longButSimpleOperandB : longButSimpleOperandC;

var z = a ?

moreComplicatedB :

moreComplicatedC;

### 4.3.8 ****&& 和 ||****

二元布尔操作符是可短路的, 只有在必要时才会计算到最后一项。

// 不推荐

function foo(opt\_win) {

var win;

if (opt\_win) {

win = opt\_win;

} else {

win = window;

}

// ...

}

if (node) {

if (node.kids) {

if (node.kids[index]) {

foo(node.kids[index]);

}

}

}

// 推荐

function foo(opt\_win) {

var win = opt\_win || window;

// ...

}

var kid = node && node.kids && node.kids[index];

if (kid) {

foo(kid);

}

### 4.3.9 ****声明规范****

* 尽量使用 let 声明变量，少用 var 。

### 4.3.10 ****分号****

以下情况需要加分号

* 表达式
* return
* throw
* break
* continue
* do-while

### 4.3.11 ****注释规范****

* 单行注释，独占一行，//后跟空格
* 多行注释，/\*后跟空格
* 函数/方法注释
  + - 注释必须包含函数声明，有参数和返回值时必须注释标识
    - 参数和返回类型必须包含类型信息和说明
    - 当函数是内部函数，外部不可访问时，可使用@inner标识

// 注释

/\* xxxx

xxxx

xxxxx

/\*

/\*\*

\* 函数描述

\* @param {string} p1 参数说明

\* @param {string} p2 参数2的说明，比较长

\* 那就换行了.

\* @param {number=} p3 参数3的说明（可选）

\* @return {Object} 返回值描述

/\*function foo(p1, p2, p3) {

return {

p1: p1,

p2: p2

}

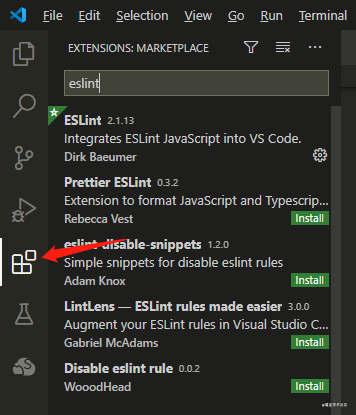
}

## 4.4 规范化工具

### 4.4.1 ESlint

**ESLint最初是由Nicholas C. Zakas 于2013年6月创建的开源项目。它的目标是提供一个插件化的javascript代码检测工具，现价段已经成为了最为主流的 JavaScript Lint 检查工具。**

1. 插件安装



1. 初始化配置 .eslintrc 文件

{

"parserOptions": {

"ecmaVersion": 2021

},

"env": {

"es6": true,

},

"parser": "babel-eslint",

"extends": "airbnb-base",

}

1. 配置

选择 File ->Preference-> Settings（如果装了中文插件包应该是 文件 -> 选项 -> 设置），搜索 eslint，点击 Edit in setting.json。

将以下选项添加到配置文件实现自动检测

"editor.codeActionsOnSave": {

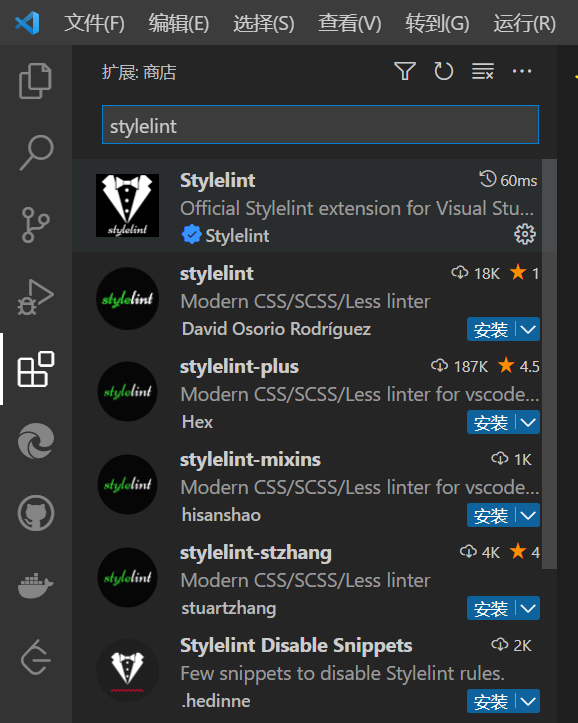
"source.fixAll": true,

},

### 4.4.2 stylelint

**stylelint 是一个开源的、用于检查 CSS 代码格式的开源工具**

1. 安装插件



1. 配置

在根目录下新建一个 .stylelintrc.json 文件，并输入以下内容：

{

"extends": "stylelint-config-standard"

}

### 4.4.3 prettier

**自动格式化代码的工具**

1. 安装



1. 配置

项目根目录下创建.prettierrc.js文件

{

"printWidth": 100, //每行到多少长度开始折行

"tabWidth": 2,

"singleQuote": true, //单引号

"trailingComma": "none", //数组、对象最后一个元素的尾逗号

"bracketSpacing": true, //花括号前后空格

"jsxBracketSameLine": true, //使多行JSX元素最后一行末尾的 > 单独一行

"parser": "babel", //指定使用哪一种解析器

"semi": true, //是否在行尾加分号

"useTabs": true, //使用tab（制表符）缩进而非空格

"arrowParens": "avoid", //只有一个参数的箭头函数的参数是否带圆括号（默认avoid不带）

"jsxSingleQuote": true, //在JSX中使用单引号

"htmlWhitespaceSensitivity": "ignore", //为 HTML 文件定义全局空格敏感度

"quoteProps": "as-needed" //自定义引号配置

}

# （五）代码提交规范

一个项目都是由很多人一起合作，然而每个人的开发习惯，提交格式并不统一，这对于统计bug 和 需求有一定困难。

git 规范一般包括两点

1. 分支管理规范
2. git commit 规范

## 5.1 分支管理规范

一般项目分主分支（master）和其他分支。

当有团队成员要开发新功能或改 BUG 时，就从 master 分支开一个新的分支。例如项目要从客户端渲染改成服务端渲染，就开一个分支叫 SSR，开发完了再合并回 master 分支。

如果要改一个重大的 BUG，也可以从 master 分支开一个新分支，并用 BUG 号命名。

# 新建分支并切换到新分支

git checkout -b test

# 切换回主分支，合并新分支

git checkout master

git merge test

注意，在将一个新分支合并回 master 分支时，如果新分支中有一些意义不明确的 commit，建议先对它们进行合并（使用 git rebase）。合并后，再将新分支合并回 master 分支。

## 5.2 git commit 规范

git 在每次提交时，都需要填写 commit message。

commit message 就是对你这次的代码提交进行一个简单的说明，好的提交说明可以让人一眼就明白这次代码提交做了什么。

git commit -m 'this is a test'

### 5.2.1 commit message 的格式

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

我们可以发现，commit message 分为三个部分(使用空行分割):

1. 标题行（subject）: 必填, 描述主要修改类型和内容。
2. 主题内容（body）: 描述为什么修改, 做了什么样的修改, 以及开发的思路等等。
3. 页脚注释（footer）: 可以写注释，放 BUG 号的链接。

### 5.2.2 commit 的类型

* feat: 新功能、新特性
* fix: 修改 bug
* perf: 更改代码，以提高性能
* 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）
* docs: 文档修改
* style: 代码格式修改, 注意不是 css 修改（例如分号修改）
* test: 测试用例新增、修改
* build: 影响项目构建或依赖项修改
* revert: 恢复上一次提交
* ci: 持续集成相关文件修改
* chore: 其他修改（不在上述类型中的修改）
* release: 发布新版本
* workflow: 工作流相关文件修改

#### 示例fix（修复BUG）

每次 git commit 最好加上范围描述。

例如这次 BUG 修复影响到全局，可以加个 global。如果影响的是某个目录或某个功能，可以加上该目录的路径，或者对应的功能名称。

// 示例1

fix(global):修复checkbox不能复选的问题

// 示例2 下面圆括号里的 common 为通用管理的名称

fix(common): 修复字体过小的BUG，将通用管理下所有页面的默认字体大小修改为 14px

// 示例3

fix(test): value.length -> values.length