# 前端开发规范

1. [前端开发规范](#%E5%89%8D%E7%AB%AF%E5%BC%80%E5%8F%91%E8%A7%84%E8%8C%83)
   1. [一、概述](#%E4%B8%80%E3%80%81%E6%A6%82%E8%BF%B0)
   2. [二、方针](#%E4%BA%8C%E3%80%81%E6%96%B9%E9%92%88)
   3. [三、环境要求](#%E4%B8%89%E3%80%81%E7%8E%AF%E5%A2%83%E8%A6%81%E6%B1%82)
   4. [四、编码规范](#%E5%9B%9B%E3%80%81%E7%BC%96%E7%A0%81%E8%A7%84%E8%8C%83)
      1. [4.1 HTML / Template 编码规范](#4.1-html-%2F-template-%E7%BC%96%E7%A0%81%E8%A7%84%E8%8C%83)
         1. [语法](#%E8%AF%AD%E6%B3%95)
         2. [HTML5 doctype](#html5-doctype)
         3. [语言属性](#%E8%AF%AD%E8%A8%80%E5%B1%9E%E6%80%A7)
         4. [字符编码](#%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81)
         5. [引入 CSS 和 JavaScript 文件](#%E5%BC%95%E5%85%A5-css-%E5%92%8C-javascript-%E6%96%87%E4%BB%B6)
         6. [减少标签的数量](#%E5%87%8F%E5%B0%91%E6%A0%87%E7%AD%BE%E7%9A%84%E6%95%B0%E9%87%8F)
         7. [属性顺序](#%E5%B1%9E%E6%80%A7%E9%A1%BA%E5%BA%8F)
         8. [语义化](#%E8%AF%AD%E4%B9%89%E5%8C%96)
      2. [4.2 CSS / Less 编码规范](#4.2-css-%2F-less-%E7%BC%96%E7%A0%81%E8%A7%84%E8%8C%83)
         1. [命名](#%E5%91%BD%E5%90%8D)
         2. [语法](#%E8%AF%AD%E6%B3%95-1)
         3. [代码风格](#%E4%BB%A3%E7%A0%81%E9%A3%8E%E6%A0%BC)
         4. [样式兼容性](#%E6%A0%B7%E5%BC%8F%E5%85%BC%E5%AE%B9%E6%80%A7)
         5. [选择器权重（样式覆盖）](#%E9%80%89%E6%8B%A9%E5%99%A8%E6%9D%83%E9%87%8D%EF%BC%88%E6%A0%B7%E5%BC%8F%E8%A6%86%E7%9B%96%EF%BC%89)
         6. [声明简写](#%E5%A3%B0%E6%98%8E%E7%AE%80%E5%86%99)
         7. [CSS 动画](#css-%E5%8A%A8%E7%94%BB)
         8. [声明顺序](#%E5%A3%B0%E6%98%8E%E9%A1%BA%E5%BA%8F)
      3. [4.3 JavaScript 编码规范](#4.3-javascript-%E7%BC%96%E7%A0%81%E8%A7%84%E8%8C%83)
         1. [命名](#%E5%91%BD%E5%90%8D-1)
         2. [语法](#%E8%AF%AD%E6%B3%95-2)
         3. [代码风格](#%E4%BB%A3%E7%A0%81%E9%A3%8E%E6%A0%BC-1)
         4. [数组、对象](#%E6%95%B0%E7%BB%84%E3%80%81%E5%AF%B9%E8%B1%A1)
         5. [使用 null](#%E4%BD%BF%E7%94%A8-null)
         6. [使用 undefined](#%E4%BD%BF%E7%94%A8-undefined)
         7. [文档注释](#%E6%96%87%E6%A1%A3%E6%B3%A8%E9%87%8A)
      4. [4.4 Vue 组件编码规范](#4.4-vue-%E7%BB%84%E4%BB%B6%E7%BC%96%E7%A0%81%E8%A7%84%E8%8C%83)
         1. [命名](#%E5%91%BD%E5%90%8D-2)
         2. [语法](#%E8%AF%AD%E6%B3%95-3)
         3. [代码风格](#%E4%BB%A3%E7%A0%81%E9%A3%8E%E6%A0%BC-2)
         4. [组件 / 实例的选项的顺序](#%E7%BB%84%E4%BB%B6-%2F-%E5%AE%9E%E4%BE%8B%E7%9A%84%E9%80%89%E9%A1%B9%E7%9A%84%E9%A1%BA%E5%BA%8F)
         5. [元素特性的顺序](#%E5%85%83%E7%B4%A0%E7%89%B9%E6%80%A7%E7%9A%84%E9%A1%BA%E5%BA%8F)
         6. [单文件组件的顶级元素的顺序](#%E5%8D%95%E6%96%87%E4%BB%B6%E7%BB%84%E4%BB%B6%E7%9A%84%E9%A1%B6%E7%BA%A7%E5%85%83%E7%B4%A0%E7%9A%84%E9%A1%BA%E5%BA%8F)
         7. [隐性的父子组件通信](#%E9%9A%90%E6%80%A7%E7%9A%84%E7%88%B6%E5%AD%90%E7%BB%84%E4%BB%B6%E9%80%9A%E4%BF%A1)
         8. [非 Flux 的全局状态管理](#%E9%9D%9E-flux-%E7%9A%84%E5%85%A8%E5%B1%80%E7%8A%B6%E6%80%81%E7%AE%A1%E7%90%86)
   5. [五、vue 代码组织规范](#%E4%BA%94%E3%80%81vue-%E4%BB%A3%E7%A0%81%E7%BB%84%E7%BB%87%E8%A7%84%E8%8C%83)

## 一、概述

本规范旨在为前端程序的开发者提供规范化最新的指导，可用于程序员个人编译环境以及研发团队集成环境等场合的代码规范化检查。

不管有多少人共同参与同一项目，一定要确保每一行代码都像是同一个人编写的

## 二、方针

|  |  |  |
| --- | --- | --- |
| 约束等级 | 约束效力 | 强制性 |
| 【强制】 | 违反该项将被认为代码存在严重缺陷 | 前端程序团队必须遵守 |
| 【推荐】 | 违反该项将被认为代码存在轻微缺陷 | 根据具体产品特性的不同，选择性地遵守 |
| 【参考】 | 违反该项可被认为代码存在优化空间 | 从产品持续优化及人员技能提升的角度，参考使用 |

## 三、环境要求

1. [Node.js](https://nodejs.org/) 8.9 或更高版本，你可以使用 [nvm](https://github.com/creationix/nvm) 或 [nvm-windows](https://github.com/coreybutler/nvm-windows) 在一台电脑中管理多个 Node 版本
2. 使用 [Visual Studio Code (VS Code)](http://www.baidu.com/link?url=Sn0aBa011utoLAP3IC799QxEcwe1-mmChrkEchQsAaOJJwwbN8ZMTyf8XF8uDId9) 进行代码编写
3. 代码提交前使用 VS Code 进行格式化（不要格式引入的外部文件）
4. 规定 Tab 大小为 2 个空格，保证在所有环境下获得一致展现// settings.json { "editor.tabSize": 2// ... }
5. 安装插件 [Vetur](https://marketplace.visualstudio.com/items?itemName=octref.vetur)（ Vue开发扩展及 Vue 文件代码格式化）
6. 安装插件 [Prettier - Code formatter](https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode)（ CSS / Less / JS 等其他文件代码格式化；Vetur 的格式化基于此插件实现，固可以在所有文件实现统一的格式化）
7. 安装插件 [Chinese (Simplified) Language Pack for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=MS-CEINTL.vscode-language-pack-zh-hans) （ VS Code 简体中文语言包）
8. 使用 Chrome 浏览器并安装 [Vue.js devtools](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd) 进行调试

## 四、编码规范

### 4.1 HTML / Template 编码规范

#### 语法

1. 缩进使用两个空格代替 Tab
   * 前端代码层级较深，使用短缩进有利于利用屏幕空间，提升效率
   * 使用两个空格代替 Tab 可以保证在所有环境下获得一致展现
2. <!-- not good --><div><div>bar</div></div><!-- good --><div><div>bar</div></div>
3. 嵌套元素应当缩进一次（即两个空格），同层级缩进应保持一致<!-- not good --><div><div>bar</div><div>bar</div></div><!-- good --><div><div>bar</div><div>bar</div></div>
4. 对于属性的定义，使用双引号，不要使用单引号<!-- not good --><inputclass='a'type=text><!-- good --><inputclass="a"type="text">
5. 不要省略可选的结束标签（closing tag）（如</li> 或 </body>）<!-- not good --><h1>h1 text <h2>h2 text <!-- good --><h1>h1 text</h1><h2>h2 text</h2>
6. 特殊符号使用 [HTML 字符实体](http://www.w3school.com.cn/html/html_entities.asp)（实体名称对大小写敏感），常用如下：
7. td / th 要在 tr 里面，li 要在 ul / ol 里面<!-- not good --><table><td>test</td></table><!-- good --><table><tr><td>test</td></tr></table>
8. ul / ol 的直接子元素只能是 li，不能包含其他元素<!-- not good --><ul><span>123</span><li>a</li><li>b</li></ul>
9. 行内元素里面不可使用块级元素<!-- not good --><ahref="../test"><div></div></a><ahref="../test"style="display: block"><div></div></a>
10. 不使用自定义标签，会与 Vue 组件系统的自定义组件冲突
11. 不使用重复属性，重复的属性只会取第一个<!-- error --><inputclass="a"type="text"class="b"><!-- good --><inputclass="a b"type="text">
12. 不要在 https 的链接里写 http 的图片<imgsrc="//static.chimeroi.com/hello-world.jpg">
13. 不要在自闭合（self-closing）元素的尾部添加斜线（ [HTML5 规范](http://dev.w3.org/html5/spec-author-view/syntax.html#syntax-start-tag)中说明这是可选的）<!-- not good --><imgsrc="logo.png"alt /><!-- good --><imgsrc="logo.png"alt>
14. 不使用属性设置样式（img, table等元素）<!-- not good --><imgsrc="test.jpg"altwidth="400"height="300"><!-- good --><imgsrc="test.jpg"style="width:400px;height:300px;">
15. 自定义属性要以 data- 开头 自己添加的非标准的属性要以data-开头，否则[w3c validator](https://link.juejin.im/?target=https%3A%2F%2Fvalidator.w3.org%2F)会认为是不规范的<!-- not good --><divcount="5"></div><!-- good --><divdata-count="5"></div>

#### HTML5 doctype

为每个 HTML 页面添加标准模式（standard mode）的声明，确保在每个浏览器中拥有一致的展现

<!DOCTYPE html><html> ...

#### 语言属性

为每个 HTML 页面根元素添加 [lang 属性](http://w3c.github.io/html/semantics.html#the-html-element)

根据 HTML5 规范：

强烈建议为 html 根元素指定 lang 属性，从而为文档设置正确的语言。这将有助于语音合成工具确定其所应该采用的发音，有助于翻译工具确定其翻译时所应遵守的规则等等。

<htmllang="zh-CN"><!-- ... --></html>

#### 字符编码

通过声明一个明确的字符编码，让浏览器轻松、快速的确定网页内容渲染方式，通常指定为'UTF-8'

<html><head><metacharset="UTF-8"></head> ...

#### 引入 CSS 和 JavaScript 文件

根据 HTML5 规范，在引入 CSS 和 JavaScript 文件时不需要指定 type 属性，因为 text/css 和 text/javascript 分别是它们的默认值

<!-- External CSS --><linkrel="stylesheet"href="code\_guide.css"><!-- In-document CSS --><style> ... </style><!-- External JS --><scriptsrc="code\_guide.js"></script><!-- In-document JS --><script> ... </script>

#### 减少标签的数量

编写 HTML 代码时，尽量避免多余的层级

<!-- not good --><spanclass="avatar"><imgsrc="..."></span><!-- good --><imgclass="avatar"src="...">

#### 属性顺序

属性应该按照特定的顺序出现以保证易读性

1. class
2. id
3. name
4. data-\*
5. src, for, type, href, value , max-length, max, min, pattern
6. placeholder, title, alt
7. aria-\*, role
8. required, readonly, disabled

#### 语义化

尽量遵循 HTML 标准和语义，但是不要以牺牲实用性为代价；任何时候都要尽量使用最少的标签并保持最小的复杂度。

### 4.2 CSS / Less 编码规范

#### 命名

1. 类名使用小写字母，以中划线分隔
2. id 采用驼峰式命名
3. less 中的变量、函数、混合等采用驼峰式命名@mainFontColor:#444; #companyName, .company-name { color: @mainFontColor; }

#### 语法

1. 所有声明语句都应当以分号结尾 最后一条声明语句后面的分号是可选的，但是，如果省略这个分号，你的代码可能更易出错/\* error \*/.selector { font-size: 15px color: red } /\* not good \*/.selector { font-size: 15px; color: red } /\* good \*/.selector { font-size: 15px; color: red; }
2. 避免为 0 值指定单位，例如，用 margin: 0; 代替 margin: 0px;
3. 为选择器中的属性添加双引号，例如，input[type="text"]； [某些情况下是可选的](http://mathiasbynens.be/notes/unquoted-attribute-values#css)，但是，为了代码的一致性，建议都加上双引号/\* not good \*/.selector[type=text] { /\* ... \*/ } /\* good \*/.selector[type="text"] { /\* ... \*/ }
4. 十六进制值应该全部小写，例如，#f3f6fa
5. 不出现空的规则（声明块中没有声明语句）
6. 不要设置太大的 z-index（一个正常的系统的层级关系在 10 以内就能完成）
7. 多写注释，且多使用句子进行描述而不是词语/\* 为了去除输入框和表单点击时的灰色背景 \*/input, form { -webkit-tap-highlight-color: rgba(255, 255, 255, 0); }
8. 不要使用\*选择器
9. 适当使用:before和:after来画页面的一些视觉上的辅助性元素，如三角形、短的分隔线、短竖线等，可以减少页面上没有用的标签
10. 选择器不要超过 4 层（在 Less 中避免嵌套超过 4 层）
11. 用 border: 0; 代替 border: none;
12. 使用简写形式的十六进制值，例如，用 #fff 代替 #ffffff
13. 对于属性值或颜色参数，省略小于 1 的小数前面的 0 （例如，.5 代替 0.5；-.5px 代替 -0.5px）

#### 代码风格

此处大部分工作将由代码格式化工具完成（参见环境要求），一般无需考虑

1. 缩进使用两个空格代替 Tab
2. 为选择器分组时，将单独的选择器单独放在一行/\* not good \*/.selector, .selector-secondary, .selector[type=text] { /\* ... \*/ } /\* good \*/.selector, .selector-secondary, .selector[type="text"] { /\* ... \*/ }
3. 声明块的左花括号前添加一个空格
4. 声明块的右花括号应当单独成行
5. 每条声明语句的 : 后应该插入一个空格
6. 每条样式声明应该独占一行/\* not good \*/.selector { font-size: 15px; color: red; } /\* good \*/.selector { font-size: 15px; color: red; }
7. 对于以逗号分隔的属性值，每个逗号后面都应该插入一个空格（例如，box-shadow，transition）/\* not good \*/.selector { transition: border .2s,color .3s,padding .4s; } /\* good \*/.selector { transition: border .2s, color .3s, padding .4s; }
8. !important前插入一个空格
9. 注释：//后插入一个空格，/\*后插入一个空格，\*/前插入一个空格
10. Less 的操作符，在圆括号中的数学计算表达式的数值、变量和操作符之间均添加一个空格
11. 注释统一用/\* \*/（ Less 中也不要用//）

#### 样式兼容性

1. 当使用一些较新的 CSS3 语法时，应注意添加浏览器前缀（ FAIS 2 打包工具包含 CSS 预处理，固无需考虑此条）
2. 不要使用 input 的 line-height 来做垂直居中 设置 line-height 为一个很高的值会导致 Safari 浏览器的输入光标变得巨大 （与 line-height 等高）/\* not good \*/input { height: 40px; line-height: 40px; } /\* good \*/input { height: 20px; line-height: 20px; padding: 10px0; }

#### 选择器权重（样式覆盖）

权重的基本规则：相同的权重：以后面出现的选择器为最后规则不同的权重，权重值高则生效详细了解权重计算方法

1. 非通用样式使用嵌套方式进行编写，避免影响其他自己不了解样式，造成样式覆盖
2. Vue 中样式谨慎使用 scoped，会影响样式选择器性能，请使用第一点进行特有样式编写
3. 样式需要修改时，尽量找到原样式声明进行修改
4. 无法修改原样式声明时，应通过权重关系，编写权重更高的样式进行覆盖
5. 不使用!important，除非原样式使用内联样式或!important且无法直接修改

#### 声明简写

1. 当你不确定自己写的属性会否影响到其他属性时，应避免使用简写/\* error \*/.element { margin: 0010px; background: red; background: url("image.jpg"); border-radius: 3px3px00; } /\* good \*/.element { margin-bottom: 10px; background-color: red; background-image: url("image.jpg"); border-top-left-radius: 3px; border-top-right-radius: 3px; }
2. 当你确定自己的声明不会影响到其他属性时，请使用简写提升代码简洁性/\* not good \*/.element { padding-top: 10px; padding-right: 20px; padding-bottom: 15px; padding-left: 20px; } /\* good \*/.element { padding: 10px20px15px; }

#### CSS 动画

1. 不要使用 all 属性做动画

使用 transition 做动画的时候不要使用 all 所有属性，在有一些浏览器上面可能会有一些问题，如下：

transition: all 2s linear;

在 Safari 上面可能会有一些奇怪的抖动，正确的做法是要用哪个属性做动画就写哪个，如果有多个就用隔开，如下代码所示：

transition: transform 2s linear, opacity 2s linear;

1. 位移动画使用 transform 替代 position （提升动画性能）
2. 使用 CSS 动画替代 JS 动画

#### 声明顺序

相关的属性声明按以下顺序做分组处理，组之间需要有一个空行

1. **Positioning**（影响其他元素和自身位置相关声明）
2. **Box model**（自身盒模型相关声明）
3. **Typographic**（文本相关声明）
4. **Visual**（自身样式）
5. **Misc**（其他声明）.declaration-order { /\* Positioning \*/position: absolute; top: 0; right: 0; bottom: 0; left: 0; z-index: 100; /\* Box-model \*/display: block; float: right; width: 100px; height: 100px; /\* Typography \*/font: normal 13px"Helvetica Neue", sans-serif; line-height: 1.5; color: #333; text-align: center; /\* Visual \*/background-color: #f5f5f5; border: 1px solid #e5e5e5; border-radius: 3px; /\* Misc \*/opacity: 1; }

### 4.3 JavaScript 编码规范

#### 命名

1. 标准变量采用驼峰式命名（考虑与后台交换数据的情况，对象属性可灵活命名）
2. 常量全大写，用下划线连接
3. 变量名不应过短，要能准确完整地描述该变量所表述的事物
4. 变量名不要使用计算机术语，如 texareaData，应该取和业务相关的名字，如 leaveMsg
5. 变量名的对仗要明确，如 up/down、begin/end、opened/closed、visible/invisible、scource/target
6. 变量名使用正确的语法
7. 使用临时变量时请结合实际需要进行变量命名// not goodlet temp = 10; let leftPosition = currentPosition + temp， topPosition = currentPosition - temp; // goodlet adjustSpace = 10; let leftPosition = currentPosition + adjustSpace， topPosition = currentPosition - adjustSpace;
8. 波尔变量可以结合实际语境使用 done/found/successs/ok/available/complete 等修饰词// goodlet ajaxDone = true, fileFound = false, resourceUpdated = true;
9. 波尔变量名应使用肯定的布尔变量名，不要使用否定的名词，如 notOk、notReady，因为否定的词取反的时候就会比较奇怪，如 if (!notOk)

#### 语法

1. 变量不要先使用后声明
2. 不要声明了变量却不使用
3. 不要在同个作用域下声明同名变量
4. 一个函数作用域中所有的变量声明尽量提到函数首部，可根据代码进行分组，但不允许出现两个连续的变量声明// not goodlet registerForm = null; let question = ""; let calculateResult = 0; // goodlet registerForm = null, question = "", calculateResult = 0;
5. 为了快速知晓变量类型，声明变量时要赋值// not goodlet registerForm, question, calculateResult; // goodlet registerForm = null, question = "", calculateResult = 0;
6. 单一函数的返回值类型要确定（如下无法确定该函数的最终返回类型）// not goodfunctioncalculatePrice(seatCount){ if (seatCount <= 0) { return""; } else { return seatCount \* 79; } }
7. debugger 不要出现在提交的代码里
8. 使用===代替==，!==代替!=（==会自动进行类型转换，可能会出现奇怪的结果）null == undefined//true'' == '0'//false0 == ''//true0 == '0'//true' \t\r\n ' == 0//truenewString("abc") == "abc"//truenewBoolean(true) == true//truetrue == 1//true
9. 使用三目运算代替简单的 if-else// not goodlet seatDiscount = 100; if (seat < 5) { seatDiscount = 90; } elseif(seat < 10) { seatDiscount = 80; } else { seatDiscount = 70; } // goodlet seatDiscount = seat < 5 ? 90 : seat < 10 ? 80 : 70;
10. 使用let定义变量，const定义常量
11. 使用箭头函数取代简单的函数// not goodlet \_this = this; setTimeout(function() { \_this.foo = "bar"; }, 2000); // goodsetTimeout(() =>this.foo = "bar", 2000);
12. 在必要的地方添加非空判断以提高代码的稳健性
13. 将复杂的函数分解成多个子函数，方便维护和复用

#### 代码风格

此处大部分工作将由代码格式化工具完成（参见环境要求），一般无需考虑

1. 缩进使用两个空格代替 Tab
2. 统一使用双引号""（与 Prettier 默认格式化配置持一致）
3. 以下几种情况后需加分号;
   * 变量声明
   * 表达式
   * return
   * throw
   * break
   * continue
   * do-while
4. 以下几种情况**不需要**空格：
   * 对象的属性名后
   * 前缀一元运算符后
   * 后缀一元运算符前
   * 函数调用括号前
   * 无论是函数声明还是函数表达式，'('前不要空格
   * 数组的'['后和']'前
   * 对象的'{'后和'}'前
   * 运算符'('后和')'前
5. 以下几种情况**需要**空格：
   * 二元运算符前后
   * 三元运算符'?:'前后
   * 代码块'{'前
   * 下列关键字前：else, while, catch, finally
   * 下列关键字后：if, else, for, while, do, switch, case, try,catch, finally, with, return, typeof
   * 单行注释'//'后（若单行注释和代码同行，则'//'前也需要），多行注释'\*'后
   * 对象的属性值前
   * for 循环，分号后留有一个空格，前置条件如果有多个，逗号后留一个空格
   * 无论是函数声明还是函数表达式，'{'前一定要有空格
   * 函数的参数之间

#### 数组、对象

1. 对象属性名不需要加引号
2. 对象以缩进的形式书写，不要写在一行
3. 数组中不要存在空元素
4. 不要用for in循环数组
5. 数组、对象最后不要有逗号// not goodlet a = { 'b': 1 }; let a = { b: 1 }; let a = { b: 1, c: 2, }; // goodlet a = { b: 1, c: 2 };

#### 使用 null

**【强制】正确使用 null**

适用场景：初始化一个将来可能被赋值为对象的变量与已经初始化的变量做比较作为一个参数为对象的函数的调用传参作为一个返回对象的函数的返回值

1. 不要用 null 来判断函数调用时有无传参
2. 不要与未初始化的变量做比较// not goodfunctiontest(a, b) { if (b === null) { // not mean b is not supply// ... } } let a; if (a === null) { // ... } // goodlet a = null; if (a === null) { // ... }

#### 使用 undefined

**【强制】正确使用 undefined**

1. 不要给变量赋值 undefined（undefined 本身就表示一个变量未定义）
2. 不要直接使用 undefined 进行变量判断
3. 使用typeof和字符串 'undefined' 对变量进行判断// not goodif (person === undefined) { // ... } // goodif (typeof person === 'undefined') { // ... }

#### 文档注释

各类标签 @param, @method 等请参考 [usejsdoc](http://usejsdoc.org/) 和 [JSDoc Guide](http://yuri4ever.github.io/jsdoc/)；

建议在以下情况下使用：

* 所有常量
* 所有函数
* 所有类

/\*\* \* @func \* @desc 一个带参数的函数 \* @param {string}a - 参数a \* @param {number}b=1 - 参数b默认值为1 \* @param {string}c=1 - 参数c有两种支持的取值</br>1—表示x</br>2—表示xx \* @param {object}d - 参数d为一个对象 \* @param {string}d.e - 参数d的e属性 \* @param {string}d.f - 参数d的f属性 \* @param {object[]}g - 参数g为一个对象数组 \* @param {string}g.h - 参数g数组中一项的h属性 \* @param {string}g.i - 参数g数组中一项的i属性 \* @param {string}[j] - 参数j是一个可选参数 \*/functionfoo(a, b, c, d, g, j) { // ... }

### 4.4 Vue 组件编码规范

#### 命名

1. **组件名**应该始终是多个单词的，根组件 App 除外// not good Vue.component('todo', { // ... }) exportdefault { name: 'Todo', // ... } // good Vue.component('todo-item', { // ... }) exportdefault { name: 'TodoItem', // ... }
2. **单文件组件**的文件名应该要么始终是单词大写开头（ PascalCase ），要么始终是横线连接（ kebab-case ）// not good components/ |- mycomponent.vue components/ |- myComponent.vue // good components/ |- MyComponent.vue components/ |- my-component.vue
3. 应用特定样式和约定的**基础组件** （也就是展示类的、无逻辑的或无状态的组件） 应该全部以一个特定的前缀开头，比如 Base、App 或 V// not good components/ |- MyButton.vue |- VueTable.vue |- Icon.vue // good components/ |- BaseButton.vue |- BaseTable.vue |- BaseIcon.vue components/ |- AppButton.vue |- AppTable.vue |- AppIcon.vue components/ |- VButton.vue |- VTable.vue |- VIcon.vue
4. 只应该拥有单个活跃实例的**单例组件**应该以 The 前缀命名，以示其唯一性单例组件不意味着组件只可用于一个单页面，而是每个页面只使用一次。这些组件永远不接受任何 prop，因为它们是为你的应用定制的，而不是它们在你的应用中的上下文。如果你发现有必要添加 prop，那就表明这实际上是一个可复用的组件，只是目前在每个页面里只使用一次// not good components/ |- Heading.vue |- MySidebar.vue // good components/ |- TheHeading.vue |- TheSidebar.vue
5. 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果一个组件只在某个父组件的场景下有意义，这层关系应该体现在其名字上。因为编辑器通常会按字母顺序组织文件，所以这样做可以把相关联的文件排在一起// not good components/ |- TodoList.vue |- TodoItem.vue |- TodoButton.vue components/ |- SearchSidebar.vue |- NavigationForSearchSidebar.vue //good components/ |- TodoList.vue |- TodoListItem.vue |- TodoListItemButton.vue components/ |- SearchSidebar.vue |- SearchSidebarNavigation.vue
6. 组件名应该以高级别的 （通常是一般化描述的） 单词开头，以描述性的修饰词结尾// not good components/ |- ClearSearchButton.vue |- ExcludeFromSearchInput.vue |- LaunchOnStartupCheckbox.vue |- RunSearchButton.vue |- SearchInput.vue |- TermsCheckbox.vue // good components/ |- SearchButtonClear.vue |- SearchButtonRun.vue |- SearchInputQuery.vue |- SearchInputExcludeGlob.vue |- SettingsCheckboxTerms.vue |- SettingsCheckboxLaunchOnStartup.vue
7. 组件名应该倾向于完整单词而不是缩写// not good components/ |- SdSettings.vue |- UProfOpts.vue // good components/ |- StudentDashboardSettings.vue |- UserProfileOptions.vue

#### 语法

1. 组件的 data 必须是一个函数（除了 new Vue 外的任何地方）// not goodexportdefault { data: { foo: 'bar' } } // goodexportdefault { data () { return { foo: 'bar' } } }
2. prop 的定义应该尽量详细，至少需要指定其类型// not good// 这样做只有开发原型系统时可以接受props: ['status'] // goodprops: { status: String } // betterprops: { status: { type: String, required: true, validator: function (value) { return [ 'syncing', 'synced', 'version-conflict', 'error' ].indexOf(value) !== -1 } } }
3. 为 v-for 设置键值；在组件上总是必须用 key 配合 v-for，以便维护内部组件及其子树的状态
4. 不要把 v-if 和 v-for 同时用在同一个元素上（大部分时候你可以使用计算属性实现）<!-- not good --><ul><liv-for="user in users"v-if="user.isActive":key="user.id" > {{ user.name }} </li></ul><!-- good --><ul><liv-for="user in activeUsers":key="user.id" > {{ user.name }} </li></ul>
5. **自闭合组件**在单文件组件、字符串模板和 JSX 中没有内容的组件应该是自闭合的；但在 DOM 模板里不要这样做<!-- not good --><!-- 在单文件组件、字符串模板和 JSX 中 --><MyComponent></MyComponent><!-- 在 DOM 模板中 --><my-component/><!-- good --><!-- 在单文件组件、字符串模板和 JSX 中 --><MyComponent/><!-- 在 DOM 模板中 --><my-component></my-component>
6. **模版中的组件名**大小写在单文件组件和字符串模板中组件名应该总是 PascalCase 的；但是在 DOM 模板中总是 kebab-case 的<!-- not good --><!-- 在单文件组件和字符串模板中 --><mycomponent/><!-- 在单文件组件和字符串模板中 --><myComponent/><!-- 在 DOM 模板中 --><MyComponent></MyComponent><!-- good --><!-- 在单文件组件和字符串模板中 --><MyComponent/><!-- 在 DOM 模板中 --><my-component></my-component><!-- 在所有地方 --><my-component></my-component>
7. **JS/JSX 中的组件名**应该始终是 PascalCase 的// not good Vue.component('myComponent', { // ... }) import myComponent from'./MyComponent.vue'exportdefault { name: 'myComponent', // ... } exportdefault { name: 'my-component', // ... } // good Vue.component('MyComponent', { // ... }) import MyComponent from'./MyComponent.vue'exportdefault { name: 'MyComponent', // ... }
8. **Prop 名大小写**，在声明 prop 的时候，其命名应该始终使用 camelCase，而在模板和 JSX 中应该始终使用 kebab-case// not goodprops: { 'greeting-text': String } <WelcomeMessage greetingText="hi"/> // good props: { greetingText: String } <WelcomeMessage greeting-text="hi"/>
9. 多个特性的元素应该分多行撰写，每个特性一行（此项 Vetur 插件会自动根据行宽阈值进行自动折行处理，一般无需考虑）<!-- not good --><imgsrc="https://vuejs.org/images/logo.png"alt="Vue Logo"><MyComponentfoo="a"bar="b"baz="c"/><!-- good --><imgsrc="https://vuejs.org/images/logo.png"alt="Vue Logo" ><MyComponentfoo="a"bar="b"baz="c" />
10. 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// not good {{ fullName.split(' ').map(function (word) { return word[0].toUpperCase() + word.slice(1) }).join(' ') }} // good// 在模板中 {{ normalizedFullName }} // 复杂表达式已经移入一个计算属性computed: { normalizedFullName: function () { returnthis.fullName.split(' ').map(function (word) { return word[0].toUpperCase() + word.slice(1) }).join(' ') } }
11. 应该把复杂计算属性分割为尽可能多的更简单的属性// not goodcomputed: { finalPrice: function () { var basePrice = this.manufactureCost / (1 - this.profitMargin) return ( basePrice - basePrice \* (this.discountPercent || 0) ) } } // goodcomputed: { basePrice: function () { returnthis.manufactureCost / (1 - this.profitMargin) }, discount: function () { returnthis.basePrice \* (this.discountPercent || 0) }, finalPrice: function () { returnthis.basePrice - this.discount } }
12. 非空 HTML 特性值应该始终带引号<!-- not good --><inputtype=text><AppSidebar:style={width:sidebarWidth+'px'}><!-- good --><inputtype="text"><AppSidebar:style="{ width: sidebarWidth + 'px' }">
13. 可简写指令需要缩写 （用 : 表示 v-bind: 和用 @ 表示 v-on:)

#### 代码风格

此处大部分工作将由代码格式化工具完成（参见环境要求），一般无需考虑

具体要求见 4.1 ，4.2，4.3 中代码风格部分

#### 组件 / 实例的选项的顺序

组件 / 实例的选项应该有统一的顺序，这是我们推荐的组件选项默认顺序：

1. **副作用** （触发组件外的影响）
   * el
2. **全局感知** （要求组件以外的知识）
   * name
   * parent
3. **组件类型** （更改组件的类型）
   * functional
4. **模板修改器** （改变模板的编译方式）
   * delimiters
   * comments
5. **模板依赖** （模板内使用的资源）
   * components
   * directives
   * filters
6. **组合** （向选项里合并属性）
   * extends
   * mixins
7. **接口** （组件的接口）
   * inheritAttrs
   * model
   * props/propsData
8. **本地状态** （本地的响应式属性）
   * data
   * computed
9. **事件** （通过响应式事件触发的回调）
   * watch
   * 生命周期钩子 （按照它们被调用的顺序）
     + beforeCreate
     + created
     + beforeMount
     + mounted
     + beforeUpdate
     + updated
     + activated
     + deactivated
     + beforeDestroy
     + destroyed
10. **非响应式的属性** （不依赖响应系统的实例属性）
    * methods
11. **渲染** （组件输出的声明式描述）
    * template/render
    * renderError

#### 元素特性的顺序

**【参考】元素 （包括组件） 的特性应该有统一的顺序，这是我们为元素特性推荐的默认顺序：**

1. **定义** （提供组件的选项）
   * is
2. **列表渲染** （创建多个变化的相同元素）
   * v-for
3. **条件渲染** （元素是否渲染 / 显示）
   * v-if
   * v-else-if
   * v-else
   * v-show
   * v-cloak
4. **渲染方式** （改变元素的渲染方式）
   * v-pre
   * v-once
5. **全局感知** （需要超越组件的知识）
   * id
6. **唯一的特性** （需要唯一值的特性）
   * ref
   * key
   * slot
7. **双向绑定** （把绑定和事件结合起来）
   * v-model
8. **其它特性** （所有普通的绑定或未绑定的特性）
9. **事件** （组件事件监听器）
   * v-on
10. **内容** （覆写元素的内容）
    * v-html
    * v-text

#### 单文件组件的顶级元素的顺序

**【强制】单文件组件应该总是按照 <template>、<script> 和 <style> 的标签顺序**

<!-- good --><!-- ComponentA.vue --><template>...</template><style>/\* ... \*/</style><!-- ComponentB.vue --><template>...</template><script>/\* ... \*/</script><style>/\* ... \*/</style>

#### 隐性的父子组件通信

**【强制】应该优先通过 prop 和事件进行父子组件之间的通信，而不是 this.$parent 或改变 prop。**

一个理想的 Vue 应用是 prop 向下传递，事件向上传递的。遵循这一约定会让你的组件更易于理解。然而，在一些边界情况下 prop 的变更或 this.$parent 能够简化两个深度耦合的组件。

问题在于，这种做法在很多简单的场景下可能会更方便。但请当心，不要为了一时方便 （少写代码） 而牺牲数据流向的简洁性 （易于理解）。

#### 非 Flux 的全局状态管理

**【强制】应该优先通过 Vuex 管理全局状态，而不是通过 this.$root 或一个全局事件总线。**

通过 this.$root 和 / 或全局事件总线管理状态在很多简单的情况下都是很方便的，但是并不适用于绝大多数的应用。Vuex 提供的不仅是一个管理状态的中心区域，还是组织、追踪和调试状态变更的好工具。

## 五、vue 代码组织规范

1、通用组件

* 命名 公司名 / 项目名缩写 + 组件名

通用类名的命名规则也可以参照这条

* 语法 单标签
* 引用 标签名、组件名、文件名三者统一

// 不推荐 <template> <my-pager :pageOption='pageOption'></my-pager> </template> <script> import pager from '../../pageCompont.vue' export default { components:{ myPager: pager } } </script> // 同样不推荐 <template> <pager :pageOption='pageOption'></pager> </template> <script> import pager from '../../pageCompont.vue' export default { components:{ pager } } </script> // 推荐 <template> <su-pager :pageOption='pageOption'/> </template> <script> import suPager from '../../suPage.vue' export default { components:{ suPager } } </script>

2、通用对象及方法

* Vuex 存放经常需要改变的量
* localStorage 存放非敏感的简单数据类型或少量复杂数据类型
* Vue.prototype
  + 存放 如 全国地址库（超大量数据）
  + 存放 通用方法（登录、登出、获取验证码等）

3、响应式对象

* data 对象中的对象数量越少越好，这样既便于维护，又方便开发，我们通常将同类型的（如同一个接口）所有变量组合成一个对象存放，这个对象的命名及对象内的 key 值应当与接口一致，如果接口名过长或语义化不强，可以自定义对象名，但对象下的 key 值必须与参数字段一致

// 不推荐 <template> <el-form> <el-form-item label="用户名" prop="name"> <el-input v-model="name"></el-input> </el-form-item> <el-form-item label="手机号" prop="phone"> <el-input v-model="phone"></el-input> </el-form-item> <el-form-item label="密码" prop="password"> <el-input v-model="password"></el-input> </el-form-item> </el-form> </template> <script> export default { data() { return { name: '', phone: '', password: '', } }, methords: { register() { // 此时需要对请求内容进行二次处理 // 一旦字段过多, 代码将会变得很难维护 let data = { userName: this.name, telNumber: this.phone, password: this.password } this.axios .post('/user/register', this.register) .then(res => { console.log(res, 'register') }).catch(err => { console.log(err) }) }, } } </script> // 推荐 <template> <el-form> <el-form-item label="用户名" prop="userName"> <el-input v-model="register.userName"></el-input> </el-form-item> <el-form-item label="手机号" prop="telNumber"> <el-input v-model="register.telNumber"></el-input> </el-form-item> <el-form-item label="密码" prop="password"> <el-input v-model="register.password"></el-input> </el-form-item> </el-form> </template> <script> export default { data() { return { register: {} } }, methords: { register() { // 此时可以直接发送对象, 无需进行任何数据处理 this.axios .post('/user/register', this.register) .then(res => { console.log(res, 'register') }).catch(err => { console.log(err) }) }, } } </script>

* 这种根据接口需求确定 v-model 值 的写法，在字段多而复杂的情况下具有极为明显的优势！这对于前后端的接口联调十分友好，这种优势尤其能够在分页查询，分类查询这类条件查询的接口中体现。

// 示例 <template> <div> <el-form :model="form" inline label-width="80px" ref="form"> <el-form-item label="姓名"> <el-input placeholder="请输入姓名" v-model="queryOption.name"></el-input> </el-form-item> <el-form-item label="电话"> <el-input placeholder="请输入用户手机号" v-model="queryOption.tel"></el-input> </el-form-item> <el-form-item label="客户等级"> <!-- 此处客户等级一般为选择器，本例为简写 --> <el-input placeholder="请输入客户等级" v-model="queryOption.VIPLevel"></el-input> </el-form-item> </el-form> <el-table :data="userList"> <el-table-column label="姓名" prop="name"></el-table-column> <el-table-column label="电话" prop="tel"></el-table-column> <el-table-column label="地址" prop="address"></el-table-column> <el-table-column label="客户等级" prop="VIPLevel"></el-table-column> </el-table> <el-pagination :current-page="queryOption.page" :page-size="queryOption.limit" :page-sizes="[20, 40, 80, 100]" :total="queryOption.total" background layout="total, sizes, prev, pager, next, jumper" ></el-pagination> </div> </template> <script> export default { data() { return { queryOption: {}, userList: [] } }, methods: { queryUserList() { this.axios .get('/queryUserList', this.queryOption) .then(res => { console.log(res) if (res.code == 200) { this.queryOption.page = res.page this.queryOption.total = res.total this.userList = res.records } }) .catch(err => { console.error(err) }) } }, watch: { queryOption: { deep: true, handler() { this.queryUserList() } } }, } </script>

4、关于代码本身

* 链式编程分行写
* 慎用三元表达式，尤其慎用三元表达式的嵌套

// 实例一 // 不建议 let data = this.radio5 == 1 ? this.diagram.outAmountList : this.radio5 == 2 ? this.diagram.orderTonnageList : this.diagram.orderAmountList // 勉强还行 let data if( this.radio5 == 1 ){ data = this.diagram.outAmountList } if( this.radio5 == 2 ){ data = this.diagram.orderTonnageList } if( this.radio5 == 3 ){ data = this.diagram.orderAmountList } // 优秀 let index = this.radio5 - 1, data = [this.diagram.outAmountList, this.diagram.orderTonnageList , this.diagram.orderAmountList][index] // 实例二 // 不推荐 let userName = localStorage.userName ? localStorage.userName : 'admin' // 推荐 let userName = localStorage.userName || 'admin'

* 多重判断，永远将特殊情况放在最前面

// 不推荐 if (res.code == 200) { localStorage.setItem('Variable1', JSON.stringify(res.data.Variable1)) localStorage.setItem('Variable2', res.data.Variable2) localStorage.setItem('Variable3', res.data.Variable3) localStorage.setItem('Variable4', res.data.Variable4) localStorage.setItem('Variable5', res.data.Variable5) console.log('登录成功') } else { if (res.code == 511) { console.log('审核中, 请稍候！') } else if (res.code == 512) { console.log('审核成功, 欢迎登录！') } else if (res.code == 513) { console.log('审核失败, 请重试！') } else if (res.code == 514) { console.log('您的账号已被停用！') } } // 推荐 if (res.code == 511) return console.log('审核中, 请稍候！') if (res.code == 512) return console.log('审核成功, 欢迎登录！') if (res.code == 513) return console.log('审核失败, 请重试！') if (res.code == 514) return console.log('您的账号已被停用！') localStorage.setItem('Variable1', JSON.stringify(res.data.Variable1)) localStorage.setItem('Variable2', res.data.Variable2) localStorage.setItem('Variable3', res.data.Variable3) localStorage.setItem('Variable4', res.data.Variable4) localStorage.setItem('Variable5', res.data.Variable5) console.log('登录成功')

* 复杂或常用的逻辑拆分

// 实例 clearValidate(){ // 重置表单校验 }, resetForm(){ // 重置表单内容 }, getData(){ // 获取新数据 }, init(){ // 页面初始化 clearValidate() resetForm() getData() }

* init函数为数据初始化
  + initData获取页面初始化数据
  + getDetail获取详情页页面需要的详细信息
  + getList获取页面列表信息

export default { mounted() { this.init() }, methods: { //初始化函数 init() { this.initData() this.getDetail() this.getList() }, //获取页面初始化数据 initData() { }, //获取详情页页面需要的详细信息 getDetail() { }, //获取列表页列表信息 getList() { } } }

* 函数定义
  + 增删改-> add,delete,update
  + 提交暂存撤回->submit，storage，withdraw
  + 关闭->close，页面如有多个关闭事件,可以这样命名，closePopup,closeWorld
  + 其他函数采用小驼峰命名，条件允许的情况下请采用动词前缀方式，请保证函数命名语义化明确
  + 构造函数命名必须采用大驼峰命名法，即首字母必须大写

export default { mounted() { }, methods: { add() {}, delete() {}, update() {}, submit() {}, storage() {}, withdraw() {}, //其他函数，例如调用弹窗 openPopup() {}, closePopup() {} } }

* css命名可以尝试BEM