**品骏前端框架设计文档**

版本 v0.7

研发中心前端框架组

2018.5.30

**修订历史记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 说明 | 作者 |
| 2018.5.13 | v0.1 | 草稿 | 翁祝平 |
| 2018.5.18 | v0.2 | 添加项目结构 | 翁祝平 |
| 2018.5.21 | v0.3 | 添加打包部署、适配移动端等内容 | 翁祝平 |
| 2018.5.22 | v0.4 | 添加代码规范 | 翁祝平 |
| 2018.5.23 | v0.5 | 添加主题和国际化 | 曾利婷 |
| 2018.5.25 | v0.6 | 添加单元测试  添加样式规范  添加自研发组件说明 | 翁祝平  曾利婷  何家伟 |
| 2018.5.30 | v0.7 | 评审后优化部分细节 | 翁祝平 |

目录

[一、说明 1](#_Toc16349)

[1.1、背景 1](#_Toc8457)

[1.2、内容简介 1](#_Toc20620)

[1.3、参考文献 2](#_Toc22815)

[二、 框架搭建 2](#_Toc2432)

[2.1、安装nvm管理node.js版本 2](#_Toc11202)

[2.2、nrm设置镜像 2](#_Toc13754)

[2.3、ox-cli代码生成工具 1](#_Toc25801)

[2.3.1、ox-cli安装 1](#_Toc21291)

[2.3.2、ox-cli生成框架 1](#_Toc11786)

[2.3.3、ox-cli生成模块代码 1](#_Toc8526)

[2.4、品骏npm私服 1](#_Toc6667)

[三、 项目结构 2](#_Toc23972)

[3.1、目录结构 2](#_Toc22808)

[3.2、数据管理 3](#_Toc25838)

[3.2.1、Vuex 3](#_Toc5128)

[3.2.2、数据缓存 3](#_Toc26456)

[3.3、路由 4](#_Toc573)

[3.3.1、vue-router 4](#_Toc3836)

[3.3.2、标签页管理 4](#_Toc15529)

[3.3.3、参数传递 5](#_Toc13145)

[3.4、权限管理 5](#_Toc7635)

[四、组件 5](#_Toc18874)

[4.1、自研发组件 5](#_Toc16180)

[4.2、第三方组件 6](#_Toc419)

[4.2.1、element-ui 6](#_Toc26224)

[4.2.2、echarts 6](#_Toc13746)

[4.2.3、icon-font 6](#_Toc2337)

[4.3、第三方工具类 6](#_Toc23715)

[4.3.1、axios 6](#_Toc1119)

[4.3.2、async/await 6](#_Toc17237)

[四、 适配移动端 6](#_Toc19753)

[5.1、弱适配 7](#_Toc24570)

[5.1.1、设置meta 7](#_Toc18768)

[5.1.2、图片设置宽度百分比 7](#_Toc20450)

[5.1.3、媒体查询 7](#_Toc15519)

[5.2、强适配 7](#_Toc23678)

[六、主题和国际化 8](#_Toc21582)

[6.1、主题切换 8](#_Toc2441)

[6.2、国际化 9](#_Toc27395)

[七、 打包部署 9](#_Toc32129)

[7.1、打包多页面 10](#_Toc12786)

[7.2、懒加载 10](#_Toc24273)

[7.2.1、页面文件懒加载 10](#_Toc4053)

[7.2.2、图片懒加载 11](#_Toc21335)

[7.3、压缩、混淆 11](#_Toc19670)

[7.4、分离样式文件 12](#_Toc32055)

[7.5、打包资源文件 12](#_Toc6823)

[7.6、hash解决CDN强缓存 13](#_Toc26386)

[八、测试 13](#_Toc16973)

[8.1、单元测试（Karma + Mocha） 13](#_Toc8983)

[8.1.1、测试框架配置 13](#_Toc19830)

[8.1.2、测试代码 14](#_Toc15248)

[8.1.3、vue-test-utils 14](#_Toc28497)

[8.2、自动化功能测试（Nightwatch） 15](#_Toc18640)

[8.3、兼容性测试 15](#_Toc20794)

[九、安全和性能 15](#_Toc24504)

[9.1、数据传输 15](#_Toc18877)

[9.1.1、HTTPS 15](#_Toc25892)

[9.1.2、请求超时时间 15](#_Toc30908)

[9.1.3、请求防篡改 15](#_Toc30208)

[9.2、服务器认证 15](#_Toc22973)

[9.2.1、服务器根据session识别用户 15](#_Toc20701)

[9.2.2、服务器根据下发token识别用户 15](#_Toc9640)

[9.2.3、认证方式配置 16](#_Toc21810)

[9.3、安全 16](#_Toc26498)

[9.3.1、防XSS攻击 16](#_Toc3510)

[9.4、性能 16](#_Toc995)

[十、 代码规范 16](#_Toc9841)

[10.1、eslint配置和vscode设置 16](#_Toc12602)

[10.2、例外 18](#_Toc29816)

[十一、样式规范 19](#_Toc3870)

[11.1、样式规则 19](#_Toc9335)

[11.1.1、类class命名规则 19](#_Toc15679)

[11.1.2、变量命名定义 19](#_Toc3684)

[11.1.3、mixins和function名定义 19](#_Toc5804)

[11.1.4、样式目录结构和文件名定义规范说明 20](#_Toc24608)

[11.1.5、补充 20](#_Toc7310)

[十二、开发工具 21](#_Toc7633)

[12.1、Visual Studio Code 21](#_Toc13200)

[12.2、Chrome调试 22](#_Toc11820)

[十三、补充 23](#_Toc12668)

[13.1、防止重复提交请求 23](#_Toc6678)

[13.2、页面快捷键 23](#_Toc25819)

[13.3、开发过程中保持第三方组件版本更新 23](#_Toc8768)

# 一、说明

## 1.1、背景

随着计算机硬件的不断提升和网络的全面提速，互联网技术近几年飞速发展。Node.js的推出给web前端开发领域带来了革命性的爆发。web前端不再只是简单展示文字信息和图片，越来越多的逻辑向前端转移，以此减少服务器的压力。几年前的jquery已经不足以应对大型的前端系统，angularjs、react、vue等一批框架应运而生。这些mvvm结构的框架让复杂的前端工程模块化、组件化，响应式数据驱动大大提高了前端渲染的能力和速度。前端技术也不仅局限于页面的开发，现在甚至能够用前端技术体系开发服务端、手机应用、PC客户端，而且比原生的技术更方便高效。

我们品骏技术部现存不少系统的前端页面是基于angularjs 1.x版本。angularjs 1.x框架是现代化前端开发技术的先行者，为前端开发者带来了全新的视野，被后面的很多框架所借鉴。但不可否认，angularjs 1.x过于臃肿，性能较差不适用于移动端。框架的开发者Google已经推翻了1.x，重新设计了新的angularjs，现在最新的版本是angularjs 6。

经过了不少的调研，各方面的考虑，和项目的实践，我们选择了vue.js作为我们新的前端开发框架。vue.js性能强大，方便易用，在国内有众多的拥趸。阿里巴巴是国内vue.js的主要推行者。vue框架有能力支撑我们品骏集团庞大的业务系统的前端开发，能满足未来几年品骏飞速发展带来的复杂需求，为5g网络新时代做好准备。

## 1.2、内容简介

本文档主要讲述以下几个方面：

1. 框架搭建：介绍安装node.js、npm，连接公司私服，使用cli工具快速生成基本框架。
2. 项目结构：使用vuex作为数据中心，管理系统的全局数据（如用户信息），管理打开的菜单、标签页等状态。通过vue-router来管理前端页面的跳转，前端的路由不再经过服务端，前端和后端只是通过ajax交换json格式数据。vuex和vue-router能更好模块化前端工程，化繁为简，提高组件复用率。
3. 组件：框架提供了根据品骏业务需求自主研发的组件（如多功能表格、嵌套表格的下拉菜单等），并推荐使用一些常用、强大的第三方组件。
4. 平台自适应：主要适配PC端和移动端，分为弱适应和强适应。弱适应要求在代码上根据百分比、rem等相对长度单位在一定程度上适应各分辨率，在较小或较大显示屏上打开页面不至于错乱。强适应则是根据不同的平台跳转对应的页面，从用户体验上适配PC端和移动端。
5. 主题和国际化：略
6. 打包部署：通过webpack打包，将代码压缩、混淆，较少生产环境代码体积。通过分块打包并懒加载，提高页面加载速度。为资源添加hash码来避免浏览器缓存。通过Jenkins自动打包部署提高发布效率，减少失误风险。
7. 测试：包括单元测试、自动化功能测试、浏览器兼容测试等。
8. 安全和性能：防xss跨域脚本攻击
9. 代码规范：推荐使用eslint的standard模板，统一代码规范。这样有利于团队开发代码合并和review。
10. 开发工具简介和推荐配置：推荐使用vscode，并建议使用常用的插件和IDE设置。简单介绍了chrome调试方法。

## 1.3、参考文献

VUE教程 <https://cn.vuejs.org/v2/guide/>

vue-router教程 <https://router.vuejs.org/zh-cn/>

vuex教程 <https://vuex.vuejs.org/zh-cn/>

ES6入门 <http://es6.ruanyifeng.com/>

饿了么前端框架官网 [http://element-cn.eleme.io/#/zh-CN](http://element-cn.eleme.io/" \l "/zh-CN)

Mint-UI移动版官网 [http://mint-ui.github.io/#!/zh-cn](http://mint-ui.github.io/" \l "!/zh-cn)

Flex布局 <http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html?utm_source=tuicool>

菜鸟教程 <http://www.runoob.com/>

# 框架搭建

# 2.1、安装nvm管理node.js版本

nvm是node.js的沙盒，通过nvm可以安装多个node.js版本并根据需要切换，在不同版本的node.js下使用npm安装的组件互不影响。nvm下载地址：<https://github.com/coreybutler/nvm-windows/releases>

nvm常用命令：

* nvm list或npm list installed 查看已安装的版本
* nvm list available 查看网络可以安装的版本
* nvm arch 查看当前系统的位数和当前node.js的位数
* nvm on 打开node.js版本控制
* nvm off 关闭node.js版本控制
* nvm proxy [url] 查看和设置代理
* nvm node\_mirror [url] 查看和设置node镜像，淘宝镜像https://npm.taobao.org/mirrors/node/
* nvm npm\_mirror [url] 查看和设置npm镜像，淘宝镜像https://npm.taobao.org/mirrors/node/
* nvm install <version> [arch] 安装指定版本
* nvm uninstall <version> 卸载指定版本
* nvm use <version> [arch] 切换到指定版本
* nvm version 查看当前nvm版本

注意，使用nvm安装node.js的时候，可能会因为路径太深，超出windows路径字符数限制而安装不成功。请尽量把nvm的安装路径设置在短的路径。

# 2.2、nrm设置镜像

我们使用nrm来作为npm registry管理工具，通过npm install nrm -g全局安装。运行nrm -h或nrm help可以查看nrm的所有命令。运行nrm ls查看所有的镜像列表：

* npm ---- https://registry.npmjs.org/
* cnpm --- http://r.cnpmjs.org/
* taobao - https://registry.npm.taobao.org/
* nj ----- https://registry.nodejitsu.com/
* rednpm - http://registry.mirror.cqupt.edu.cn/
* npmMirror https://skimdb.npmjs.com/registry/
* edunpm - http://registry.enpmjs.org/

我们选择淘宝镜像：nrm use taobao

## 2.3、ox-cli代码生成工具

### 2.3.1、ox-cli安装

运行npm install ox-cli -g全局安装。

### 2.3.2、ox-cli生成框架

ox-cli init [template] <project name>

e.g. ox-cli init simple myproject

ox框架模板有以下几种：

* simple 提供基本的目录结构和依赖
* standard 提供简单的目录结构和依赖，包括自动化测试、代码规范配置、网络和安全工具包等
* full提供完整的目录结构和依赖，包括全套自研发组件和推荐的第三方组件

### 2.3.3、ox-cli生成模块代码

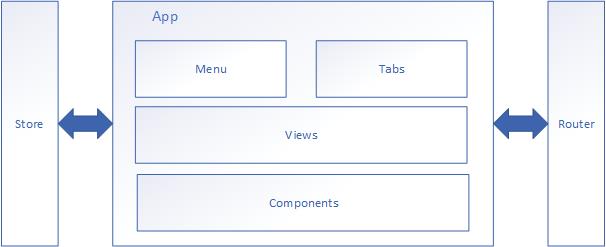
通过ox-cli generate -n [name] -t [template]生成页面模板。e.g. ox-cli generate -n demo -t tms-query。其中-n是生成模板组件的name和文件名，-t是生成的模板的。目前模板库有3中，后续会逐渐丰富。

* default：默认的模板，可以不用指定-t，e.g. ox-cli generate -n demo
* tms-query：TMS项目查询页面的模板
* tms-operate：TMS项目操作页面的模板

## 2.4、品骏npm私服

N/A

# 项目结构



## 3.1、目录结构

│ index.html 主页

│ package.json 项目配置文件，包括项目信息、依赖插件等

│ README.md 项目的说明文件

│

├─build 打包配置

│ config.js 开发人员根据需求更改config配置，不要去修改webpack.xxx.conf.js

│ webpack.base.conf.js 基本打包配置

│ webpack.dev.conf.js 开发环境打包配置

│ webpack.prod.conf.js 生产环境打包配置

│

├─dist 打包生成的文件，直接部署到服务器，自动生成

├─node\_modules 组件安装目录，自动生成

├─src 源代码

│ │ App.vue 入口vue组件，注入到index.html运行

│ │ main.js 入口js

│ │

│ ├─assets 资源目录，放图片、音频、字体等

│ ├─components 自定义组件目录

│ ├─config 系统配置

│ │ index.js

│ │ http.js 全局配置axios

│ │

│ ├─router 路由配置vue-router

│ │ index.js

│ │

│ ├─store 数据中心配置vuex

│ │ index.js 数据中心的总入口

│ │ └─[module name] 把数据中心划分为多个模块

│ │ index.js 模块的入口

│ │ mutations.js 把mutations独立出来，避免index太复杂

│ │ actions.js 把actions独立出来

│ │

│ ├─utils 自定义工具方法

│ │ index.js

│ └─views 用于显示的页面组件

│ └─demo

│ Demo.vue

│

└─static 不通过webpack打包管理的资源，直接在index.html引入

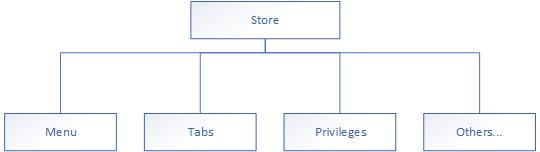
## 3.2、数据管理

### 3.2.1、Vuex

框架使用vuex作为数据中心，存放全局性的数据信息和控制状态。通过在main.js里配置好store，在任何vue组件内可以方便获取数据，而不需要经过层层的prop传递。修改数据则需要通过commit和dispatch的方式向vuex发送命令，避免在各个组件修改数据而导致混乱。vue组件对vuex的数据一样具有响应式。

参考：<https://vuex.vuejs.org/zh-cn/>

Vuex在框架中一个重要的角色就是用来保存菜单结构、已打开的标签页，并管理是否打开新标签页或打开已有标签页，以及当前打开菜单项和当前标签页的关系。只需在vue组件的任何地方向vuex发出open tab（暂定）的命令。



把store划分了模块后，建议为模块加上namespaced: true的属性。这样可以避免mutations和actions的命名冲突。使用了命名空间后，指令前面也要加上模块的名字，e.g. this.$store.dispatch(‘tabs/open’)。

### 3.2.2、数据缓存

1）localStorage可以一直保存数据，除非用户清楚了浏览器的localStorage。localStorage有较大的容量（在chrome浏览器有5M，不同的浏览器容量不同），可以存放比较多的信息，而Cookies只有4K。可以通过localStorage来保存一些用户个性化的设置，如主题颜色等。

* 主题
* 语言

1. 存在sessionStorage的数据在关闭浏览器或标签页后清除，可以用于保存如用户资料等。如果没有再发起网络请求。以此减少请求次数，提高页面加载速度，提升用户体验，减轻服务器负担。

p.s. Vuex用来在内存中存放数据，和页面做响应式交互，但是一刷新页面Vuex的内容会丢失。所以部分刷新后需要保留的数据可以放在sessionStorage里面。

* 用户资料

3）Cookies具有时效，可以用来保存用户登录信息，实现记住登录信息，一段时间内免登录。Cookies的大小限制是4Kb。

* 登录信息

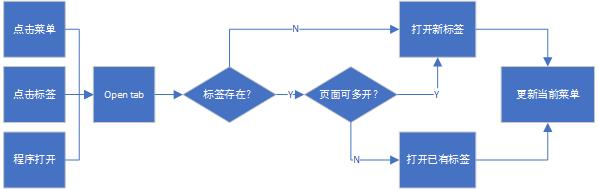
## 3.3、路由

### 3.3.1、vue-router

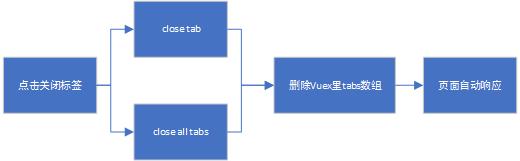
用于登录页面、主页、说明页等的跳转。

### 3.3.2、标签页管理

Vuex中的tabs模块管理已打开的标签列表，当点击菜单或程序向vuex提交open tab命令时创建一个新的组件对象，和标签页的其他信息一起插入tabs数组，然后更新vuex中menu模块的当前菜单属性。



当点击标签页的关闭按钮，或是关闭全部页面选项时，程序向vuex提交close tab或close all tabs命令。vuex的mutation对应方法从tabs列表中删除该tab元素，页面通过响应式关闭了标签页，标签页的vue对象也被自动回收。



### 3.3.3、参数传递

向vuex发送open tab命令时带上参数对象，vuex将组件对象和参数对象一起保存在tabs的列表。页面渲染标签组件时从tabs列表获取组件对象和参数，将参数作为params属性传入组件。这里要求从外部获取参数的页面组件需要支持params属性

<el-tabs v-model="currentTab" type="card" closable @tab-remove="removeTab" @tab-click="clickTab">

<el-tab-pane v-for="item in $store.state.tabs.list" :key="item.name" :label="item.title" :name="item.name">

<component :is="item.component" :ref="item.name" **:params="item.params"** />

</el-tab-pane>

</el-tabs>

## 3.4、权限管理

用户登录成功后系统读取程序中完整的目录树，再根据服务端返回的权限列表，过滤没有权限的模块，将可访问的目录树存放在vuex的menu模块中。页面响应式渲染出菜单。



# 四、组件

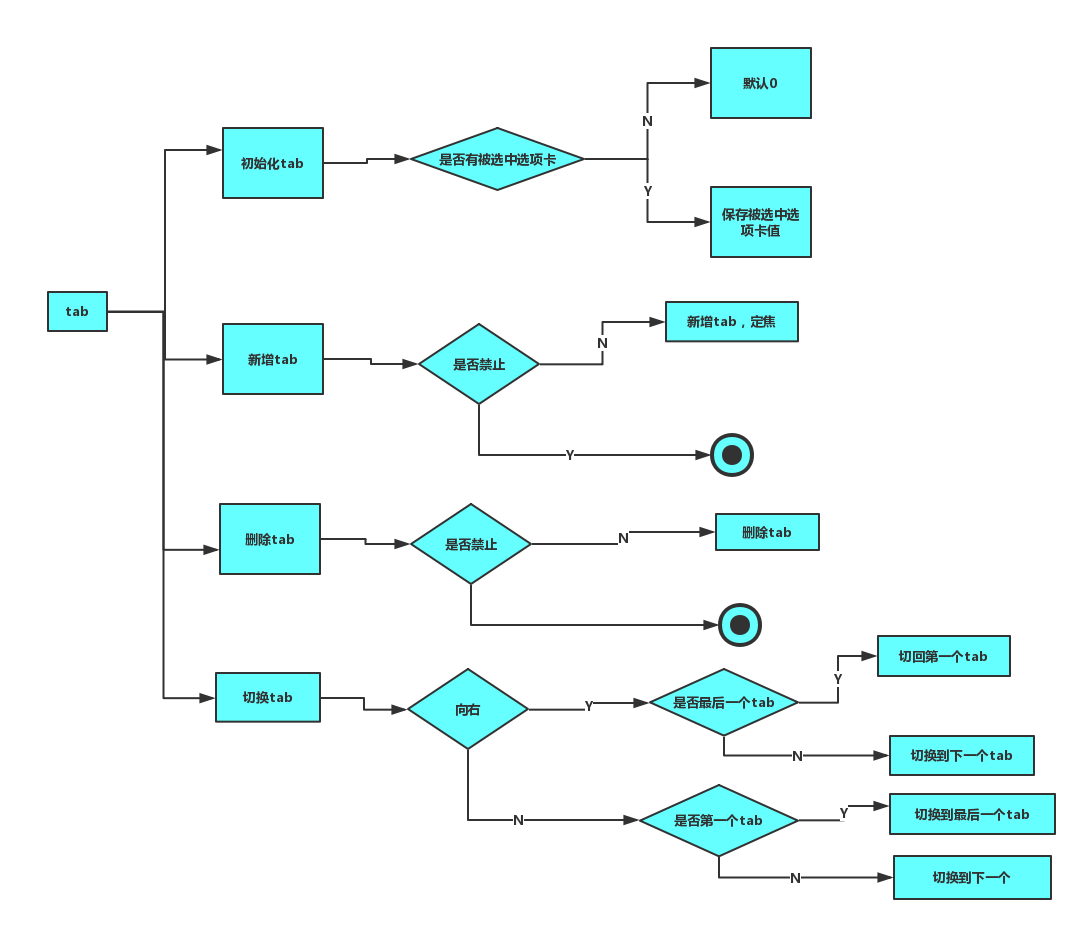
## 4.1、自研发组件

### 4.1.1 ox-tab 标签页

常用于平级区域大块内容的的收纳和展现或分隔内容上有关联但属于不同类别的数据集合；

配置说明

1 活动图



2 属性说明

1）、tab属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **属性** | **说明** | **类型** | **可选值** | **默认值** |
| closable | 标签是否可关闭 | Boolean | True/false | true |
| activeName | 绑定值，选中选项卡的name | String | --- | 第一个选项卡的name |
| isShowBtn | 是否显示操作按钮 | Boolean |  | true |
| before-leave | 切换之前的钩子，若返回false或promise且被reject，则阻止切换 | function | --- | --- |

2）、tab事件名称

|  |  |  |
| --- | --- | --- |
| **事件名称** | **说明** | **回调参数** |
| tab-click | tab被选中时触发 | 被选中的tab标签实例 |
| tab-remove | tab 被移除后触发 | 被删除的标签的tab |
| tab-add | tab 新增后触发 | --- |
| tab-remove-all | tab 全部关闭时触发 | ---- |
| tab-close-other | 保存当前tab 时触发 | ---- |

3）、tab-pane 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **说明** | **类型** | **可选值** | **默认值** |
| label | 选项卡说明 | string | --- | ---- |
| disable | 是否禁用 | Boolean | true/false | false |
| name | 与选项卡activeName 对应的标识符 | string |  | 该选项卡在选项卡列表中的顺序值，如第一个选项卡则为'1' |
| closable | 标签是否可关闭 | Boolean | true/false | true |

### 4.1.2 ox-menu 菜单

ox-menu是高度封装组件，为页面和功能提供导航功能的菜单列表，用户只需创建ox-menu标签，然后把配置和数据源添加进去，就可以生成一个基本的表格。

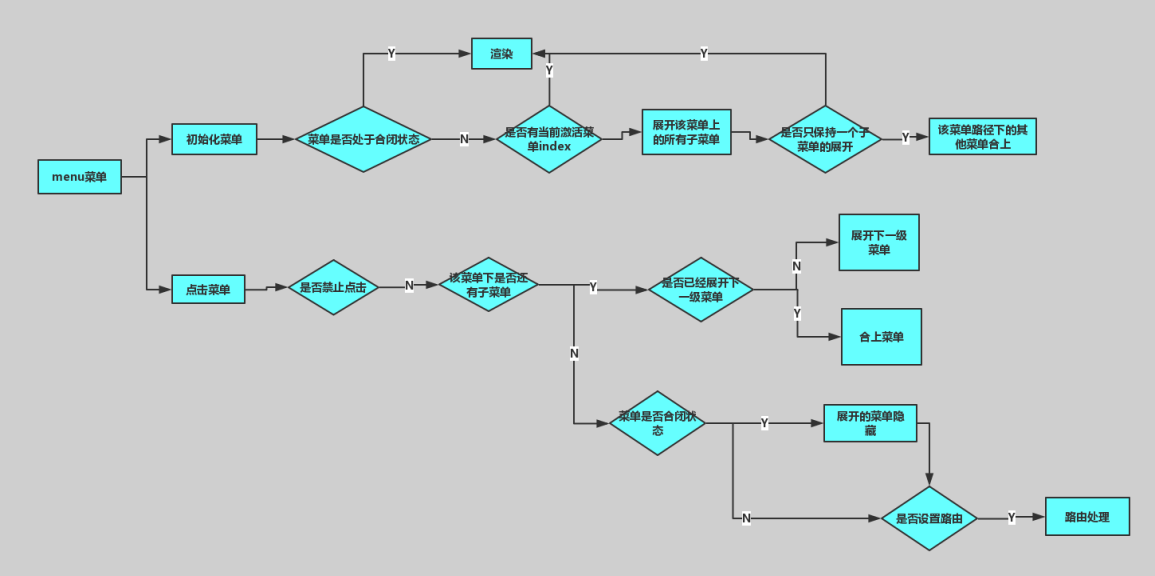
<ox-menu :menuConfig=”menuConfig”:menuTree=“menuTree”/>

</ox-menu>

ox-menu组件需要引入公共样式，一下列举几个可能用到的公共样式，以实现全局样式的联动：

* menu-item-color：菜单项字体颜色
* menu-item-fill：菜单项背景颜色
* menu-item-hover-fill：鼠标经过时菜单项背景颜色

1. **活动图**



1. **配置说明**

|  |  |  |  |
| --- | --- | --- | --- |
| **属性** | **类型** | **默认值** | **说明** |
| menuTree | Array | -- | 菜单数据源，详细见下menuTree配置说明 |
| menuConfig | object | -- | 菜单基础配置，详细见下menuConfig配置说明 |
| collapse | boolean | false | 是否水平折叠收起菜单 |

**menuTree配置说明**

|  |  |  |  |
| --- | --- | --- | --- |
| **属性** | **类型** | **默认值** | **说明** |
| title | string | -- | 菜单项名 |
| key | string | -- | 菜单项唯一标识 |
| icon | string | -- | 菜单项图标 |
| component | string | -- | 菜单项触发的组件 |
| submenu | object | -- | 菜单项的子菜单 |
| path | string | -- | 启用vue-router模式时设置该path以index作为路由跳转 |

**menuCofig配置说明**

|  |  |  |  |
| --- | --- | --- | --- |
| ***属性*** | ***类型*** | ***默认值*** | ***说明*** |
| backgroundColor | string | #fff | 菜单的背景色（仅支持 hex 格式） |
| textColor | string | #303133 | 菜单的文字颜色（仅支持 hex 格式） |
| activeTextColor | string | #ffd04b | 当前激活菜单的文字颜色（仅支持 hex 格式） |
| defaultOpeneds | array | -- | 当前打开的 sub-menu 的 index 的数组 |
| defaultActive | string | 1 | 当前激活菜单的 index |
| showTimeout | number | 300 | 展开 sub-menu 的延时 |
| hideTimeout | number | 300 | 收起sub-menu 的延时 |
| collapseTransition | boolean | true | 是否开启折叠动画 |
| router | Boolean | false | 是否使用 vue-router 的模式，启用该模式会在激活导航时以 index 作为 path 进行路由跳转 |
| uniqueOpened | boolean | true | 是否只保持一个子菜单的展开 |
| isHorizontalBar | boolean | true | 菜单内容是否设置水平滚动条 |
| isVerticalBar | Boolean | true | 菜单内容是否设置垂直滚动条 |

**oxmenu事件**

|  |  |  |
| --- | --- | --- |
| **事件** | **说明** | **返回值** |
| select | 菜单激活触发 | index: 选中菜单项的 index, indexPath: 选中菜单项的 index path |
| open | sub-menu 展开的回调 | index: 打开的 sub-menu 的 index，indexPath: 打开的 sub-menu 的 index path |
| close | sub-menu 收起的触发 | index: 收起的 sub-menu 的 index， indexPath: 收起的 sub-menu 的 index path |

### 4.1.3 ox-scrollbar

**属性配置**

|  |  |  |  |
| --- | --- | --- | --- |
| ***属性*** | ***类型*** | ***默认值*** | ***说明*** |
| wrapClass | string | {} | 自定义设置ox-scrollbar\_\_wrap 的class样式 |
| wrapStyle | string/array | {} | 自定义设置ox-scrollbar\_\_wrap 的style样式 |
| viewClass | string | {} | 自定义设置ox-scrollbar\_\_view 的class样式 |
| viewStyle | array/array | {} | 自定义设置ox-scrollbar\_\_view的style样式 |
| noresize | boolean | false | 窗口或容器尺寸大小不会变化时设置为true，提升性能 |
| isVerticalBar | boolean | true | 是否显示垂直方向上的滚动条 |
| isHorizontalBar | boolean | true | 是否显示水平方向上的滚动条 |
| native | boolean | false | 是否使用原生滚动条，isVerticalBar和isHorizontalBar同样适用 |

详见《组件使用说明.doc》



## 4.2、第三方组件

### 4.2.1、element-ui

N/A

### 4.2.2、echarts

N/A

### 4.2.3、icon-font

N/A

## 4.3、第三方工具类

### 4.3.1、axios

参考 <https://www.jianshu.com/p/df464b26ae58>

### 4.3.2、async/await

N/A

# 适配移动端

系统以PC端页面为主，兼容移动端显示。兼容方式分为弱适配和强适配。弱适配主要是通过百分比的长度单位，以及媒体查询等方式，做到在移动端打开PC版页面格式不错乱。强适配则是开发另外移动端版本页面，采用适合移动端的表单组件和交互方式，针对移动端增删某些功能。服务端或页面根据浏览器平台选择跳转到对应版本。

## 5.1、弱适配

### 5.1.1、设置meta

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

* width=device-width 表示宽度是设备屏幕的宽度；
* initial-scale=1 表示初始的缩放比例；
* shrink-to-fit=no 自动适应手机屏幕的宽度。

### 5.1.2、图片设置宽度百分比

<img src... stype=”width: 100%” />

（补图）

### 5.1.3、媒体查询

使用CSS3的媒体查询，当屏幕分辨率小于（max-width）或大于（min-width）某个阀值时应用另外一套样式。

@media screen and (max-width: 992px) {

.some-block {

display: none;

}

}

参考资料：<http://www.runoob.com/cssref/css3-pr-mediaquery.html>

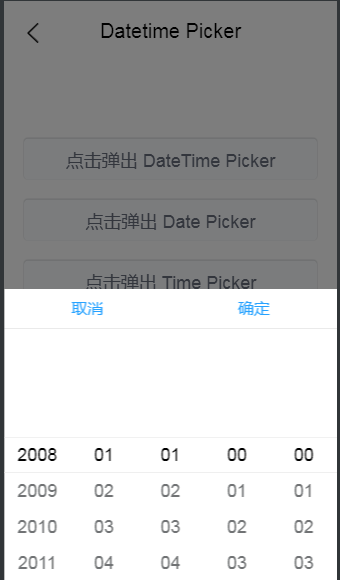
## 5.2、强适配

强适配要求为移动端开发另外一套页面，从根本上提升移动端用户体验，而又不会因为要兼顾PC端和移动端混入很多判断的代码影响逻辑。

PC版的日历：



移动版的日历：



移动版的UI框架参考Mint-UI：[http://mint-ui.github.io/#!/zh-cn](http://mint-ui.github.io/" \l "!/zh-cn)

**前端根据浏览器平台类型跳转：**

N/A

# 六、主题和国际化

## 6.1、主题切换

主题切换是为尽量满足不同用户的审美需求和避免对单一主题产生审美疲劳，主要针对系统背景颜色和字体颜色的几种换肤切换；

设计方案：

不同主题定义分别抽离到对应的主题文件，对引入不同主题文件的多个样式主入口文件分别打包生成不同主题的css文件

实际应用情况如下：

1. 如系统选定为某个单一主题，可通过link标签或@import引用某个css主题文件，从用户体验上来说，建议采用link标签，因为@import方式是装载完整个html文件再去装载css文件，当文件较大而网速较慢的情况下，import加载容易出现明显的闪烁；
2. 如系统有主题切换的需求，则采用link标签，js控制触发主题切换按钮时动态改变link的href值；
3. 如系统需要根据节日或天气自动切换主题，可通过接口获取天气信息或获取指定日期的节假日信息后，自定义判断条件，在页面加载时link引入不同的主题文件；
4. 如果现有的主题仍不能满足需求，可通过暴露变量和新增样式文件，直接更改主题变量实现自定义（同iview），说明如下：
5. 首先在项目中先建一个目录，比如 my-theme，然后在 my-theme 下建立一个 scss 文件 index.scss，并写入如下内容

@import '~ox/src/styles/index.scss'；

$--color-primary: #409EFF !default；

1. 然后在入口main.js导入这个scss文件

import '../my-theme/index.scss';

## 6.2、国际化

采用vue-i18n国际化，满足中英文切换；

1. 安装 npm install –save vue-i18n;
2. Src目录下创建多语言的子文件夹lang，采用多文件管理不同的语言，lang下新建zh-CN.js和en-US.js模块，自定义本地的翻译信息，对应的语言信息保存为不同的json对象；
3. 设置语言，在主入口main.js中引入

Import VueI18n from ‘vue-i18n’；

Vue.use(VueI18n)；

Vue.config.lang = localStorage.lang || navigator.language||’zh-CN’；

(优先级顺序：localstorage用户的语言选择🡪浏览器语言🡪默认中文)

1. 多语言配置

Import zhLocale from ‘./lang/zh-CN’；

Import enLocale from ‘./lang/en-US’；

Vue.locale(‘zh-CN’,zhLocale)；

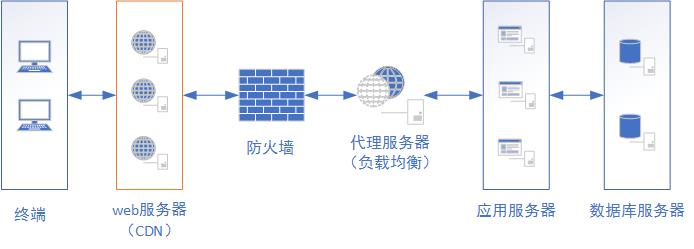
Vue.locale(‘en-US’,enLocale)；

6） 通过vue-i18n的$t(‘ ’)传入key获取对应版本的翻译信息；

5） 通过改变Vue.config.lang值切换语言，切换时把值存储在localStorage

# 打包部署

运行npm run build打包，生成的结果文件放在了dist目录。可以通过Jenkins部署到web服务器。部署的过程先拷贝带hash码的css、js、图片等静态文件，最后拷贝覆盖index.html，避免在发布过程有用户访问而导致加载不到正确资源。



## 7.1、打包多页面

现代化前端开发一般推荐单页面应用，但很多时候我们也需要用到多页面，比如让PC版和移动版在同一个工程里面，共用部分逻辑和组件。打包多页面需要在webpack的配置文件中设置多个入口，output的文件也以[name]变量作为文件名。然后在HtmlWebpackPlugin中配置需要注入的代码块。

{

entry: {

**main: ‘./src/main.js’,**

**other: ‘./src/other.js’**

},

output: {

path: path.join(\_\_dirname, ‘dist’),

filename: ‘js/**[name]**.js’

},

...

plugins: {

new HtmlWebpackPlugin({

filename: ‘index.html’,

template: ‘./index.html’,

**chunks: [‘main’],**

hash: true

}),

new HtmlWebpackPlugin({

filename: ‘other.html’,

template: ‘./other.html’,

**chunks: [‘other’],**

hash: true

}),

}

}

CommonsChunkPlugin是否能让页面自动注入用到的公共模块，而不引入没用到的公共模块？如果没办法自动，在入口处配置pc-common和mobile-common，在对应的js文件中引入需要的模块，通过配置CommonsChunkPlugin让它们打包成两个模块文件，然后HtmlWebpackPlugin的chunks属性中显示引入。（未验证）

## 7.2、懒加载

### 7.2.1、页面文件懒加载

vue的页面组件异步引入有多种方式，我们选用ES6推荐的动态import。动态import需要引入babel-plugin-syntax-dynamic-import组件，在.babelrc中配置：

{

...

“plugins”: [... “syntax-dynamic-import”],

...

}

我们需要在config/menu.js配置每一个页面的时候显式import对应的页面组件，webpack打包的时候才能把组件打包成chunk模块文件，并在用到的时候才去加载。

{

title: '说明',

key: 'direction',

**component: () => import(/\* webpackChunkName: "pages" \*/ '@/views/direction/Direction'),**

icon: 'el-icon-news'

}

### 7.2.2、图片懒加载

使用vue-lazyload组件来实现图片懒加载，当图片元素出现在显示区域时，才去请求图片文件。这样可以提高页面的加载速度。

参考：<https://www.npmjs.com/package/vue-lazyload>

## 7.3、压缩、混淆

发布到生产服务器的版本需要使用UglifyJsPlugin插件来压缩和混淆代码，减少代码文件大小，提高加载速度，且避免源代码泄露。

plugins: [

new UglifyJsPlugin()

]

参考：<https://www.npmjs.com/package/uglifyjs-webpack-plugin>

## 7.4、分离样式文件

ExtractTextPlugin可以把vue代码中的样式抽取成单独的样式文件。

用法一：

plugins: [

new ExtractTextPlugin({

filename: ‘css/[name].[contenthash].css’,

allChunks: true

})

]

用法二：

module: {

rules: [

{

test: /\.css$/,

use: ExtractTextPlugin.extract({

fallback: ‘style-loader’,

use: ‘css-loader’

})

}

]

}

参考：<https://www.npmjs.com/package/extract-text-webpack-plugin>

## 7.5、打包资源文件

通过url-loader加载资源（图片、音频、字体等），可以指定打包的方式。如下面代码，limit: 10000表示不超过10000bytes的文件将以base 64的方式被打包进js文件中，如果超过10000bytes，则存放到img文件夹下，并添加7位的hash code。

module: {

rules: {

test: /\.(png|jpe?g|gif|svg)(\?.\*)?$/,

loader: ‘url-loader’,

options: {

limit: 10000,

name: ‘img/[name].[hash:7].[ext]’

}

}

}

## 7.6、hash解决CDN强缓存

上述的几个插件都可以指定配置生成文件的hash code格式，通过在文件名后面加hash code，可以避免客户端浏览器缓存而没有应用到最新的代码文件。

# 八、测试

## 8.1、单元测试（Karma + Mocha）

目前主流的vue上的单元测试方法有几种，我们选择Karma + Mocha这一套。Mocha是单元测试框架，断言工具采用Chai，Karma是运行在浏览器的单元测试运行器。虽然在前端做单元测试不是必要的，但我们还是建议在开发功能前先写好测试用例，用测试去驱动开发，并在开发完成或修改了功能代码后用测试区验证。

### 8.1.1、测试框架配置

**1）安装插件**

* karma
* karma-chrome-launcher 调用chrome，在chrome运行测试用例
* karma-mocha
* karma-sourcemap-loader
* karma-spec-reporter 在控制台输出测试结果
* karma-webpack 读取webpack设置来编译项目文件
* karma-coverage 生成覆盖率报告
* karma-chai 运行器对chai的支持，同时需要chai
* chai 断言库
* mocha
* babel-plugin-istanbul 根据source-map计算覆盖率

**2）定义测试脚本**

package.json

{

“scripts”: {

“test”: “karma start --single-run”

}

}

karma start默认读取同目录下的karma.conf.js配置文件，如果不用默认路径，可以另外指明：

karma start test/unit/karma.conf.js

**3）配置karma.conf.js**

var webpackConfig = require('../../build/webpack.base.conf.js')

module.exports = function (config) {

config.set({

browsers: ['Chrome'],

frameworks: ['mocha', ‘chai’],

reporters: ['spec', 'coverage'],

files: ['specs/\*\*/\*.spec.js'],

preprocessors: {

'\*\*/\*.spec.js': ['webpack', 'sourcemap']

},

webpack: webpackConfig,

coverageReporter: {

dir: './coverage',

reporters: [

{ type: 'lcov', subdir: '.' },

{ type: 'text-summary' }

]

}

})

}

相比Jest、mocha-webpack等运行在node.js的测试框架，karma的好处是运行在真实浏览器的，并可以通过安装多个浏览器launcher，如karma-phantomjs-launcher、karma-firefox-launcher，让测试代码同时在多个浏览器运行，在一定程度上可以测试浏览器兼容性。

其中PhantomJS是一个只有内核、没有界面的浏览器。可以首选PhantomJS运行测试用例，提高测试运行速度。

### 8.1.2、测试代码

**1）目录结构**

test

└─unit

│ .eslintrc 配置测试代码的eslint规则

│ karma.conf.js 测试运行器配置

│

└─specs 存放单元测试代码，文件名一般是测试的目标文件名加上spec.js

xxx.spec.js

**2）代码**

参考 <http://www.ruanyifeng.com/blog/2015/12/a-mocha-tutorial-of-examples.html>

### 8.1.3、vue-test-utils

vue-test-utils是vue官方推出的，为方便对vue组件做单元测试的工具库。可以配合任何测试框架使用。参考 <https://vue-test-utils.vuejs.org/zh/>

## 8.2、自动化功能测试（Nightwatch）

N/A

## 8.3、兼容性测试

N/A

# 九、安全和性能

## 9.1、数据传输

### 9.1.1、HTTPS

N/A

### 9.1.2、请求超时时间

具体设置参考4.3.1

### 9.1.3、请求防篡改

N/A

## 9.2、服务器认证

### 9.2.1、服务器根据session识别用户

N/A

### 9.2.2、服务器根据下发token识别用户

N/A

### 9.2.3、认证方式配置

N/A

## 9.3、安全

### 9.3.1、防XSS攻击

N/A

## 9.4、性能

N/A

# 代码规范

框架使用eslint的javascript标准模板作为代码规范。规范的具体内容参考：

<https://github.com/standard/standard/blob/master/docs/RULES-zhcn.md>

## 10.1、eslint配置和vscode设置

**1）安装eslint插件作为开发依赖**

“devDependencies”: {

...

"babel-eslint": "^8.2.1",

"eslint": "^4.15.0",

"eslint-config-standard": "^10.2.1",

"eslint-friendly-formatter": "^3.0.0",

"eslint-loader": "^1.7.1",

"eslint-plugin-import": "^2.7.0",

"eslint-plugin-node": "^5.2.0",

"eslint-plugin-promise": "^3.4.0",

"eslint-plugin-standard": "^3.0.1",

"eslint-plugin-vue": "^4.0.0",

}

**2）webpack配置eslint（开发模式）**

module: [

rules: [{

test: /\.(js|vue)$/,

loader: 'eslint-loader',

enforce: 'pre',

include: [resolve('src'), resolve('test')],

options: {

formatter: require('eslint-friendly-formatter'),

emitWarning: true

}

}

...

]

**3）在项目根目录添加eslint配置文件**

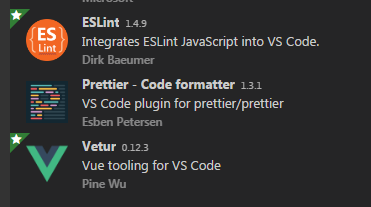
* .editorconfig 编辑器配置
* .eslintignore 忽略的文件，只对js有效
* .eslintrc.js 检查规则

具体内容详见代码，参考：<https://blog.csdn.net/IT_HLM/article/details/78776630>

以上1、2、3点的配置使用ox-cli生成项目框架已经包含，用户无需另外配置。

这样在开发模式运行前就会通过eslint检查代码规范，如果不符合规范就会报错。然后我们配置一下vscode，在写代码的时候让IDE提示，并自动帮我们按照eslint的设置格式化。

**4）vscode安装格式化插件**



**5）vscode用户设置**

{

"eslint.enable": true,

"eslint.validate": [

"javascript",

{

"language": "vue",

"autoFix": true

}

],

"eslint.autoFixOnSave": true,

"prettier.eslintIntegration": true,

"vetur.format.defaultFormatter.html": "js-beautify-html",

"vetur.format.defaultFormatter.css": "prettier",

"vetur.format.defaultFormatter.js": "prettier",

"workbench.editor.enablePreview": false,

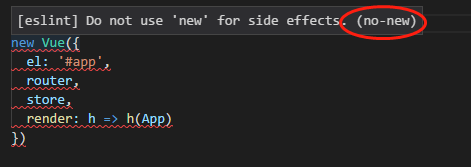
"editor.tabSize": 2,

}

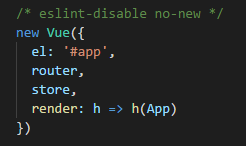
## 10.2、例外

有时候在个别地方不严格遵守eslint的规范会让代码更灵活，这时我们可以通过注释的方式告诉eslint这段代码是例外，不用遵守某个规范。

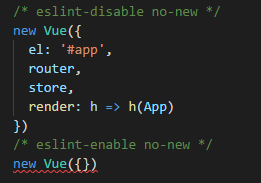
e.g. 如在man.js入口的new Vue，不符合eslint的no-new规范：



我们可以通过注释告诉eslint这段代码不用遵守no-new，这样在vscode和编译运行就不会报错了



eslint-disable注释是把之后所有的代码都忽略eslint检查，应该在要忽略检查的代码块后再加上eslint-enable，重新打开检查。



注释里不指明eslint的规范项的话，就禁用了eslint的所有检查 e.g. /\* eslint-disable \*/。但是还是建议大家尽量遵守eslint规范。

# 十一、样式规范

## 11.1、样式规则

### 11.1.1、类class命名规则

为达到一定的命名统一，避免样式污染，采用连接符区分BEM（B-Block，E-Element，M-Modifier）三类元素来命名class，命名规则统一写在config.scss，以便统一修改，详细定义规则如下：

1、命名空间定义：ox，后面接组件名关键词，如ox-button，ox-tooltip，ox-card，ox-table

2、元素E连接符：\_\_，如\_\_title，\_\_content，\_\_header，\_\_footer（即组件被解剖的各个部分的关键词）

3、修饰M连接符：--，如—success， --default，--small，--large（即对组件不同意义或不同规格的关键字描述）

4、状态修饰符：is-，如is-active，is-checked，is-top（即对组件状态的一般描述）

5、图标命名：ox-ico，如ox-ico-circle，ox-ico-arrow-left（即对图标含义的关键字描述）

故Class类命名，一般地：

1. 命名空间定义+组件名+元素连接符：ox-buttoon\_\_title，ox-table\_\_row，ox-table\_\_header；
2. 命名空间定义+组件名+修饰连接符：ox-button--small，ox-alert--success；
3. 状态类名：如果是通用的可单独使用如is-fit，is-selected，如果是针对某个组件的状态描述，采用后代选择器，如.ox-table\_\_row .is-checked；
4. 图标类名：ox-ico，ox-ico-circle，ox-ico-arrow-left。

### 11.1.2、变量命名定义

变量定义统一写在var.scss文件，主要分通用变量定义和组件变量定义：

1. 通用变量定义
2. color颜色：定义几个主色调和辅色调，通过color函数衍生不同色阶混合的颜色；
3. 字体规格（size：base、lg、sm；weight：normal、light、bold）
4. 屏幕尺寸（xs（<768px）、sm（≥768px）、md（≥992px）、lg（≥1200px）、xl（≥1600）），；

2、组件变量定义是针对特定组件的属性定义，比如table的边框颜色、按钮字体样式、table当前行被选中的背景和字体样式、table鼠标hover时的行样式等等。

变量定义规则：

1、采用与class类似的定义规范，$--组件名-（组件元素或状态修饰）-属性-辅助修饰，

如$--table-body-bg，$--table-text-color，$--table-row-hover-bg

1. 每个组件变量定义前严格要求注释，定义书写在当前组件注释之后和下一个组件注释之前的空间块

3、所有变量定义统一美元符$对齐，不采用冒号或变量值对齐

### 11.1.3、mixins和function名定义

复用性强的代码块可采用混合和函数名，定义无须前缀，要求见名知义，建议采用 属性名-属性值或属性名配合传参的方式，如float-left，scroll-bar。

### 11.1.4、样式目录结构和文件名定义规范说明

─style 样式目录

├─css //编译后css

└─sass //sass样式

│ fonts //oxui字体资源

│ oxui.scss //sass样式主入口

│ oxui\_theme\_blue.scss //蓝色主题主入口

│ oxui\_theme\_green.scss //绿色主题主入口

├─oxstyle

│ │ index.scss //oxui样式主入口

│ │

│ ├─common //oxui基础公共样式目录

│ │ utilities.scss //自定义常用class

│ │ var.scss //oxui变量定义

│ │

│ ├─components //oxui组件样式目录

│ │ button.scss //oxui组件样式文件

│ │ table.scss

│ ├─mixins //oxui混合目录

│ │ config.scss // oxui连接符和前缀配置

│ │ function.scss //oxui函数定义

│ │ mixins.scss //oxui混合定义

│ │ utils.scss //自定义常用混合

│ │ \_table.scss //如果公共混合定义不能满足，可新增组件混合定义文件

│ │

│ └─page //oxui页面目录

│ │ login.scss //oxui登陆页面样式

│ │ error.scss //oxui 404/500error页面

│ └─theme //oxui主题目录

│ theme\_blue.scss //oxui蓝色主题样式

│ theme.green.scss //oxui 绿色主题样式

### 11.1.5、补充

为着重于需求定制化开发，一部分组件样式源于Element的按需加载，一部分组件基于需求做部分重写或完全重写。

1. 按需加载方式
2. 安装npm install babel-plugin-component –D
3. 将.babelrc修改为

{

"presets": [["es2015", { "modules": false }]],

"plugins": [

[

"component",

{

"libraryName": "element-ui",

"styleLibraryName": "theme-chalk"

}

]

]

}

1. main.js引入所需的部分组件

import { message } from “element-ui”;

vue.use(message);

通过这种方式可以实现“引入的组件对应的样式才有加载进来”，达到减少项目体积的目的

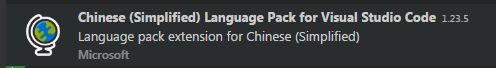
1. 样式重写
2. 按照12.1.1-12.1.4样式规范定义

# 十二、开发工具

## 12.1、Visual Studio Code

VSCode是微软开发的跨平台编辑器，运行速度非常快。可以通过安装插件的方式支持vue。

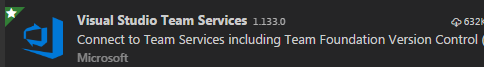
**1）安装官方提供的中文包**



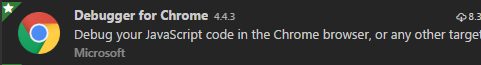
1. **代码格式化设置**

参考“十、代码规范”

**3）代码管理（git）**



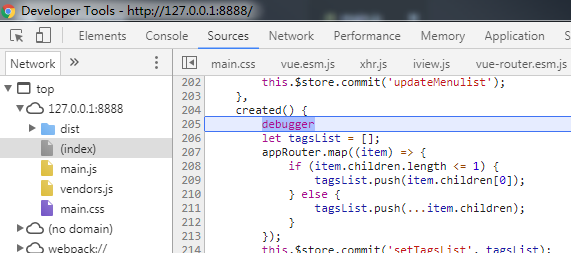
**4）调试**



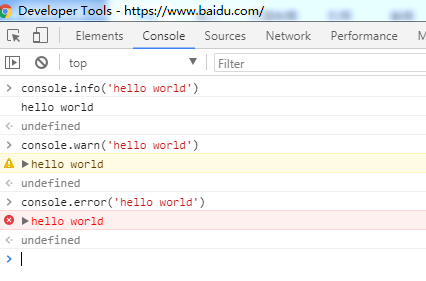
## 12.2、Chrome调试

Chrome浏览器可以很方便的调试前端页面代码，按F12或Ctrl + Shft + I调出开发者工具。这里可以查看日志输入、调试页面布局样式、设置断点调试javascript代码、监视页面性能、查看网络请求、查看本地数据缓存等功能。使用Chrome调试是前端开发者的必修课。这里介绍几项用法：

1. 在代码中输入debugger，Chrome运行到这里会断点（需开启source-map模式）



1. 除了console.log，还有console.error、console.warn



# 十三、补充

## 13.1、防止重复提交请求

N/A

## 13.2、页面快捷键

N/A

## 13.3、开发过程中保持第三方组件版本更新

npm通过package.json来管理依赖包的版本，其中版本号默认带有前缀，用于指定版本依赖的范围。依赖包的前缀有一下几种：

* ^ 匹配最新的大版本依赖包，如^1.2.3会匹配所有1.x.x的版本，包括1.3.0，不包括2.0.0
* ~ 匹配最新的小版本依赖包，如~1.2.3会匹配所有1.2.x的版本，不包括1.3.0
* \* 匹配最新的依赖包
* 没有前缀，匹配具体的版本依赖包

使用npm安装后自动生成package-lock.json文件，package-lock.json锁定安装时的包的版本号，并且需要上传到git，以保证其他人在npm install时大家的依赖能保证一致。

在开发过程要注意第三方包是否有新的版本，特别是一些不稳定的第三方包，获取最新的修复版本。常用的依赖包管理命令如下：

* npm ls <package> [-g]查看当前包的版本
* npm info <package> 查看远程包版本（或npm view <package> version[s]）
* npm update <package> --save 更新包版本到最新版
* npm install <package>@<version> 安装指定版本的包