**工程目录结构**

.

├── build/ # webpack 打包输出路径

│ └── ...

│

├── config/ # webpack 配置文件

│ ├── webpack.dev.conf.js

│ └── webpack.prod.conf.js

│

├── mock/ # 如有统一的接口文档，可自行mock模拟数据

│ └── user.js

│

├── src/ # 资源文件夹，理论上资源文件都应在dependencies中有记录

│ ├── main.js # 应用入口文件

│ ├── vendor.js # 第三方资源 单独抽离打包。

│ ├── App.vue # vue 应用根组件

│ │

│ │

│ ├── components/ # ui 可复用基础组件

│ │ ├── SelectorItem.vue

│ │ └── ...

│ │

│ ├── page/ # 单个页面的组件目录

│ │ ├── workspace/ # 一级导航路径对应的 所有页面文件夹

│ │ │ ├── components/ # 一级路径页面 拆分的组件。

│ │ │ ├── Index.vue # 对应一级路径的页面 ( /workspace/ )

│ │ │ ├── Detail.vue # 对应的子页面( /workspace/detail/:id )

│ │ │ └── ......

│ │ ├── user/

│ │ └── ...

│ │

│ ├── static/ # 静态资源目录

│ │ ├── fonts/ # 字体存放目录

│ │ └── images/ # 图片资源存放目录

│ │

│ ├── script/ # 公共方法js文件

│ │

│ ├── style/ # 可复用样式文件，不可复用的之间写在`.vue`文件中

│ │ ├── common.scss # 全局通用样式，重写UI样式

│ │ └── ......

│ │

│ ├── store/ # 状态管理器，需模块化

│ │ ├── index.js # 组装模块并导出 store 的地方

│ │ └── modules/ # vuex模块目录

│ │ ├── publish.js # 发布模块

│ │ ├── public.js # 公用数据模块

│ │ └── ......

│ │

│ ├── routes/ # 路由配置

│ │

│ └── axios/ # http 库

│ ├── index.js # 配置好的http构造实例

│ └── config.js # 配置请求

│

├── .babelrc # babel 配置文件

├── index.template.html # index.html template 文件

└── package.json # npm 配置文件

**开发规范**

**page目录下文件夹命名**

1. 使用小写字母
2. 使用名词
3. 尽量只有***一个单词***，如 user/ ,

**.vue 文件 规范**

1. 所有通过import 进来的模块/资源，尽量用webpack配置下的alias别名进行定位获取（如：import User from 'components/User' 而不是 import User from '../../components/User'。） import子目录下的资源例外。（PS：）
2. 必须有name值，格式为“ConfigTree” 大驼峰命名格式，与组件名一致。
3. 页面组件尽量命名为： Index.vue, Detail.vue, Edit.vue, Add.vue, List.vue 等之类的形式。
4. 页面组件的name值加上文件夹前缀。（eg: app/Detail.vue 组件的 name值为AppDetail）。
5. vue 方法放置顺序

export default : {

name: '',

components: {},

props: {

},

data: {

},

computed: {

},

methods: {

},

watch: {

},

mounted() {

},

....（其他）

}

1. .vue文件代码不宜太长，太长请拆分成组件。
2. 同理，methods的方法不宜太长，太长请拆分多个方法。
3. 改动复用类组件时，务必考虑其他依赖到该组件的变动，**尽量少改组件暴露的接口**。
4. 只抽离全局都用到的第三方资源，echart之类的只在个别页面用到的，请在其组件内单独加载。
5. 异步请求数据时，应加上isLoading 属性，同时请求错误时 isLoading = false。
6. 使用catch捕获错误时，**必须** console.error(e); 如try catch, Promise.reject , axios.catch 等方法，以免出现调试困难。

**vue-router路由规范**

1. 路由必须有name值
2. 系统内的路由跳转必须以name值为标识（eg: this.$router.push({ name: 'home'})）, **切勿**直接（this.router.push({ path: '/home' })）。因为路径随时会变，name则不会。

**vuex 状态管理器规范**

1. 当应用变得非常复杂时，store 对象就有可能变得相当臃肿，故将需要 store 分割成模块。
2. 为了便于管理，每个模块统一独立成一个文件，全部模块导入到 store/index.js ，进行模块注册后export。

**面向未来编程原则**

1. 以建立仓库的时间为时间节点，选用**最新稳定版本**的 vue 与 element-ui，新版本的vue 与 element往往会有更多新特性，拥抱变化，这对开发更加友好。尽早将新技术引入业务中来，而不是拘泥于旧版本，导致面向过去编程。
2. 确定好第三方库的版本后，**不可将第三方库进行版本升级**。以免产生第三方库不向下兼容导致的bug。
3. 由于开发周期比较长，开发时间内官方文档可能已经更新了多个新的小版本，**查官方文档时留意当前官方文档的版本号**，以免使用了不同版本号的特性导致功能无法工作。

**附录**

* [vue 官方文档](https://cn.vuejs.org/)
* [element-ui 官方文档](http://element.eleme.io/#/zh-CN/component/installation)
* [vue-router 官方文档](https://router.vuejs.org/zh-cn/)
* [vuex 官方文档](https://vuex.vuejs.org/zh-cn/)
* [axios 官方文档](https://github.com/mzabriskie/axios)