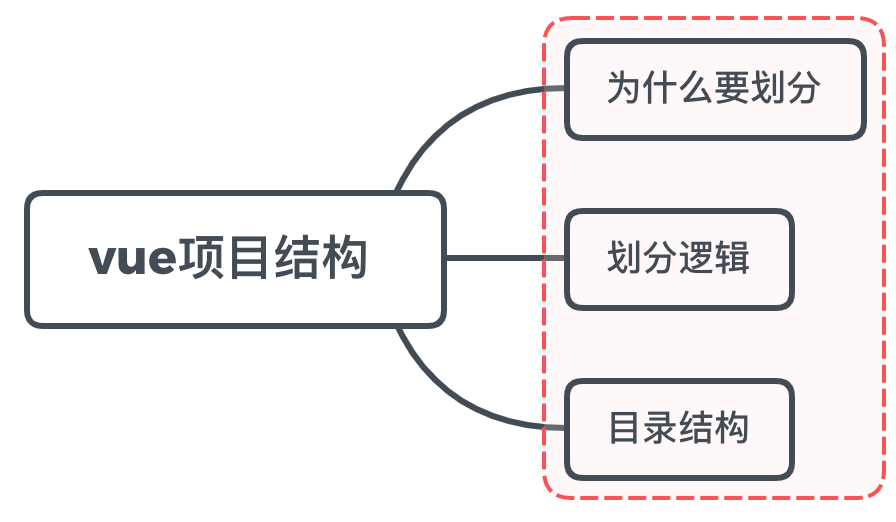
# 面试官：说下你的vue项目的目录结构，如果是大型项目你该怎么划分结构和划分组件呢？



## 一、为什么要划分

使用vue构建项目，项目结构清晰会提高开发效率，熟悉项目的各种配置同样会让开发效率更高

在划分项目结构的时候，需要遵循一些基本的原则：

* 文件夹和文件夹内部文件的语义一致性
* 单一入口/出口
* 就近原则，紧耦合的文件应该放到一起，且应以相对路径引用
* 公共的文件应该以绝对路径的方式从根目录引用
* /src 外的文件不应该被引入

### 文件夹和文件夹内部文件的语义一致性

我们的目录结构都会有一个文件夹是按照路由模块来划分的，如pages文件夹，这个文件夹里面应该包含我们项目所有的路由模块，并且仅应该包含路由模块，而不应该有别的其他的非路由模块的文件夹

这样做的好处在于一眼就从 pages文件夹看出这个项目的路由有哪些

### 单一入口/出口

举个例子，在pages文件夹里面存在一个seller文件夹，这时候seller 文件夹应该作为一个独立的模块由外部引入，并且 seller/index.js 应该作为外部引入 seller 模块的唯一入口

// 错误用法  
import sellerReducer from 'src/pages/seller/reducer'  
  
// 正确用法  
import { reducer as sellerReducer } from 'src/pages/seller'

这样做的好处在于，无论你的模块文件夹内部有多乱，外部引用的时候，都是从一个入口文件引入，这样就很好的实现了隔离，如果后续有重构需求，你就会发现这种方式的优点

### 就近原则，紧耦合的文件应该放到一起，且应以相对路径引用

使用相对路径可以保证模块内部的独立性

// 正确用法  
import styles from './index.module.scss'  
// 错误用法  
import styles from 'src/pages/seller/index.module.scss'

举个例子

假设我们现在的 seller 目录是在 src/pages/seller，如果我们后续发生了路由变更，需要加一个层级，变成 src/pages/user/seller。

如果我们采用第一种相对路径的方式，那就可以直接将整个文件夹拖过去就好，seller 文件夹内部不需要做任何变更。

但是如果我们采用第二种绝对路径的方式，移动文件夹的同时，还需要对每个 import 的路径做修改

### 公共的文件应该以绝对路径的方式从根目录引用

公共指的是多个路由模块共用，如一些公共的组件，我们可以放在src/components下

在使用到的页面中，采用绝对路径的形式引用

// 错误用法  
import Input from '../../components/input'  
// 正确用法  
import Input from 'src/components/input'

同样的，如果我们需要对文件夹结构进行调整。将 /src/components/input 变成 /src/components/new/input，如果使用绝对路径，只需要全局搜索替换

再加上绝对路径有全局的语义，相对路径有独立模块的语义

### /src 外的文件不应该被引入

vue-cli脚手架已经帮我们做了相关的约束了，正常我们的前端项目都会有个src文件夹，里面放着所有的项目需要的资源，js,css, png, svg 等等。src 外会放一些项目配置，依赖，环境等文件

这样的好处是方便划分项目代码文件和配置文件

## 二、目录结构

单页面目录结构

project  
│ .browserslistrc  
│ .env.production  
│ .eslintrc.js  
│ .gitignore  
│ babel.config.js  
│ package-lock.json  
│ package.json  
│ README.md  
│ vue.config.js  
│ yarn-error.log  
│ yarn.lock  
│  
├─public  
│ favicon.ico  
│ index.html  
│  
|-- src  
 |-- components  
 |-- input  
 |-- index.js  
 |-- index.module.scss  
 |-- pages  
 |-- seller  
 |-- components  
 |-- input  
 |-- index.js  
 |-- index.module.scss  
 |-- reducer.js  
 |-- saga.js  
 |-- index.js  
 |-- index.module.scss  
 |-- buyer  
 |-- index.js  
 |-- index.js

多页面目录结构

my-vue-test:.  
│ .browserslistrc  
│ .env.production  
│ .eslintrc.js  
│ .gitignore  
│ babel.config.js  
│ package-lock.json  
│ package.json  
│ README.md  
│ vue.config.js  
│ yarn-error.log  
│ yarn.lock  
│  
├─public  
│ favicon.ico  
│ index.html  
│  
└─src  
 ├─apis //接口文件根据页面或实例模块化  
 │ index.js  
 │ login.js  
 │  
 ├─components //全局公共组件  
 │ └─header  
 │ index.less  
 │ index.vue  
 │  
 ├─config //配置（环境变量配置不同passid等）  
 │ env.js  
 │ index.js  
 │  
 ├─contant //常量  
 │ index.js  
 │  
 ├─images //图片  
 │ logo.png  
 │  
 ├─pages //多页面vue项目，不同的实例  
 │ ├─index //主实例  
 │ │ │ index.js  
 │ │ │ index.vue  
 │ │ │ main.js  
 │ │ │ router.js  
 │ │ │ store.js  
 │ │ │  
 │ │ ├─components //业务组件  
 │ │ └─pages //此实例中的各个路由  
 │ │ ├─amenu  
 │ │ │ index.vue  
 │ │ │  
 │ │ └─bmenu  
 │ │ index.vue  
 │ │  
 │ └─login //另一个实例  
 │ index.js  
 │ index.vue  
 │ main.js  
 │  
 ├─scripts //包含各种常用配置，工具函数  
 │ │ map.js  
 │ │  
 │ └─utils  
 │ helper.js  
 │  
 ├─store //vuex仓库  
 │ │ index.js  
 │ │  
 │ ├─index  
 │ │ actions.js  
 │ │ getters.js  
 │ │ index.js  
 │ │ mutation-types.js  
 │ │ mutations.js  
 │ │ state.js  
 │ │  
 │ └─user  
 │ actions.js  
 │ getters.js  
 │ index.js  
 │ mutation-types.js  
 │ mutations.js  
 │ state.js  
 │  
 └─styles //样式统一配置  
 │ components.less  
 │  
 ├─animation  
 │ index.less  
 │ slide.less  
 │  
 ├─base  
 │ index.less  
 │ style.less  
 │ var.less  
 │ widget.less  
 │  
 └─common  
 index.less  
 reset.less  
 style.less  
 transition.less

### 小结

项目的目录结构很重要，因为目录结构能体现很多东西，怎么规划目录结构可能每个人有自己的理解，但是按照一定的规范去进行目录的设计，能让项目整个架构看起来更为简洁，更加易用

## 参考文献

* https://juejin.cn/post/6844904129186234381#heading-0
* https://zhuanlan.zhihu.com/p/89693668