前端开发演进之路

从大后端到前端组件化

大后端时代

MVC

优势

- 后端数据直接渲染模板，速度快
- 页面可以直接执行代码逻辑
- 后端程序员可以无缝开发

缺点

- 页面嵌入代码逻辑，复用和结构调整需考虑变量相关
- 不易测试前端逻辑
- (多后端)同时开发(同一功能)效率低下

分离后端逻辑

数据API时代

特点

- 后端只负责渲染页面
- 不再使用后端模板
- 前后端通过数据API交互

优势

- 前端独立
- 协作更加高效

缺点

- 前端页面富含js逻辑，各种dom操作
- 不够工程化

前端框架兴起

前端MVC时代

特点

- 前端路由
- 逻辑由前端控制

优势

- 前端工程化
- 可测试
- 减少页面随处js逻辑的情况
- 前端可以更加专注

缺点

- 代码冗余
- 复用率不够高

前端工程化

前端组件化时代

特点

- MVVM
- 数据驱动
- 前端功能模块组件化
- 前端架构设计更清晰
- UI与逻辑解耦

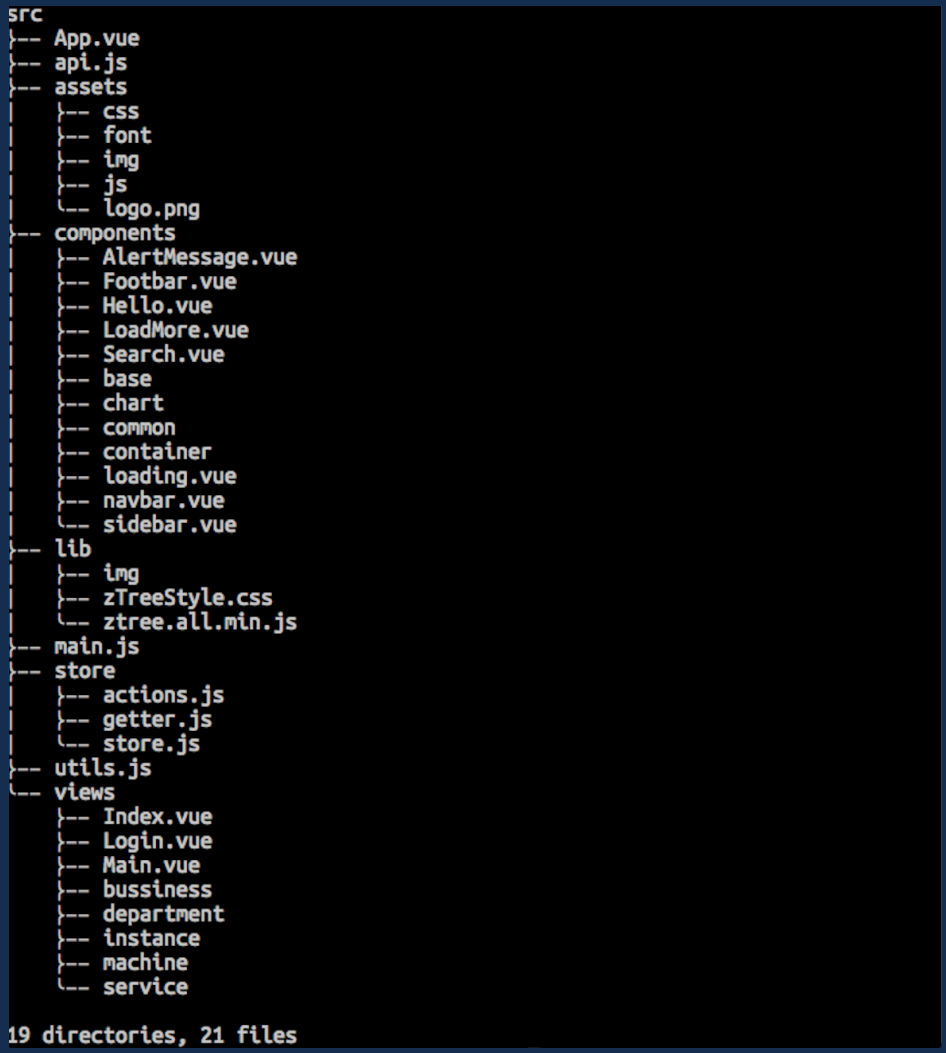
优势

- 复用率高
- Web Application
- 工程化程度较高

缺点

- 对工程师要求较高
- 对前端生态需要熟悉

代码结构


src
├── App.vue
├── api.js
├── assets
│ ├── css
│ ├── font
│ ├── img
│ ├── js
│ └── logo.png
├── components
│ ├── AlertMessage.vue
│ ├── Footbar.vue
│ ├── Hello.vue
│ ├── LoadMore.vue
│ ├── Search.vue
│ ├── base
│ ├── chart
│ ├── common
│ ├── container
│ ├── loading.vue
│ ├── navbar.vue
│ └── sidebar.vue
├── lib
│ ├── img
│ ├── zTreeStyle.css
│ └── ztree.all.min.js
├── main.js
├── store
│ ├── actions.js
│ ├── getter.js
│ └── store.js
├── utils.js
└── views
 ├── Index.vue
 ├── Login.vue
 ├── Main.vue
 ├── bussiness
 ├── department
 ├── instance
 ├── machine
 └── service
19 directories, 21 files

总结

- 后端MVC
- 前端MVC
- 前端MVVM
- 组件化

Thanks