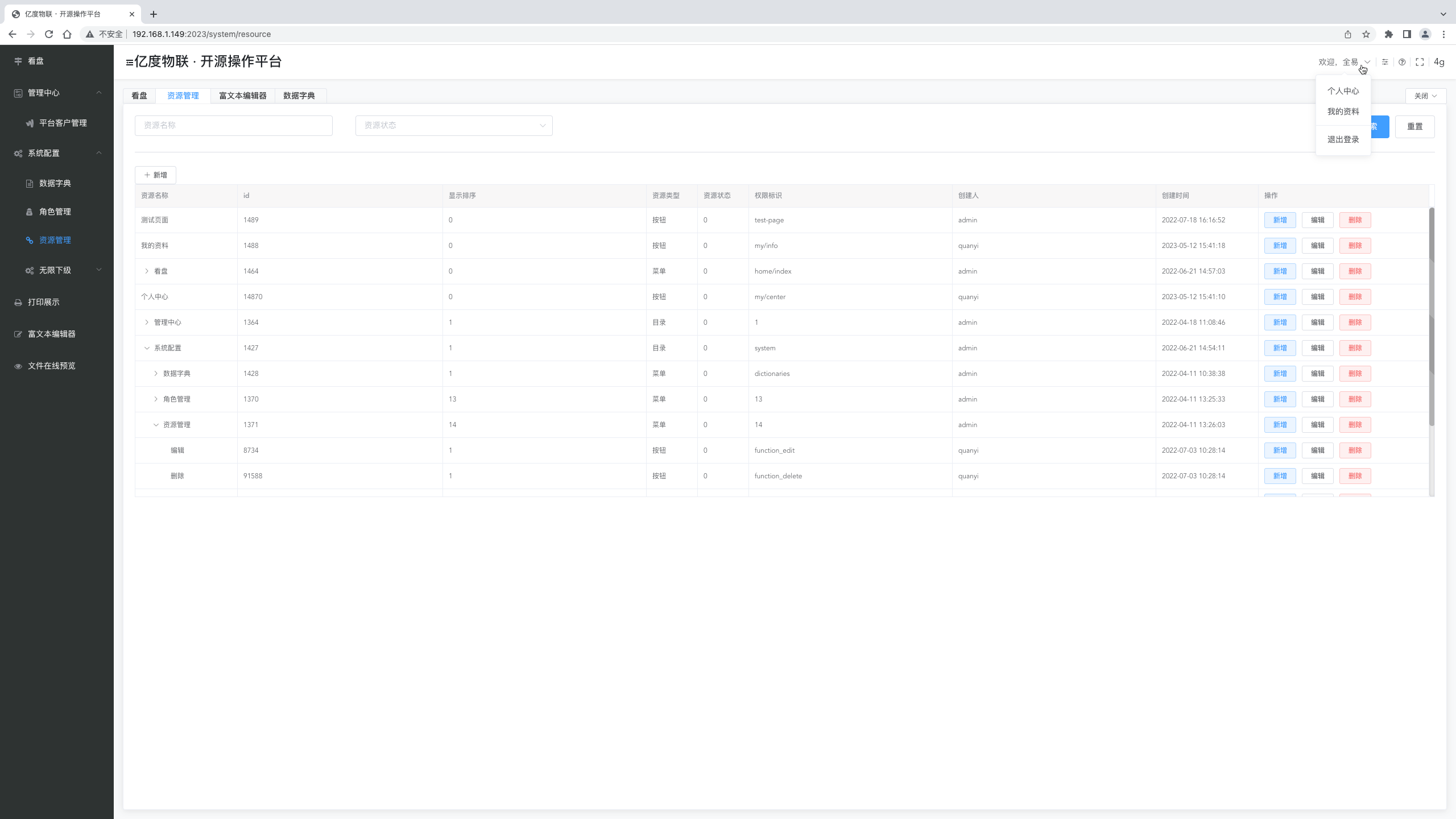
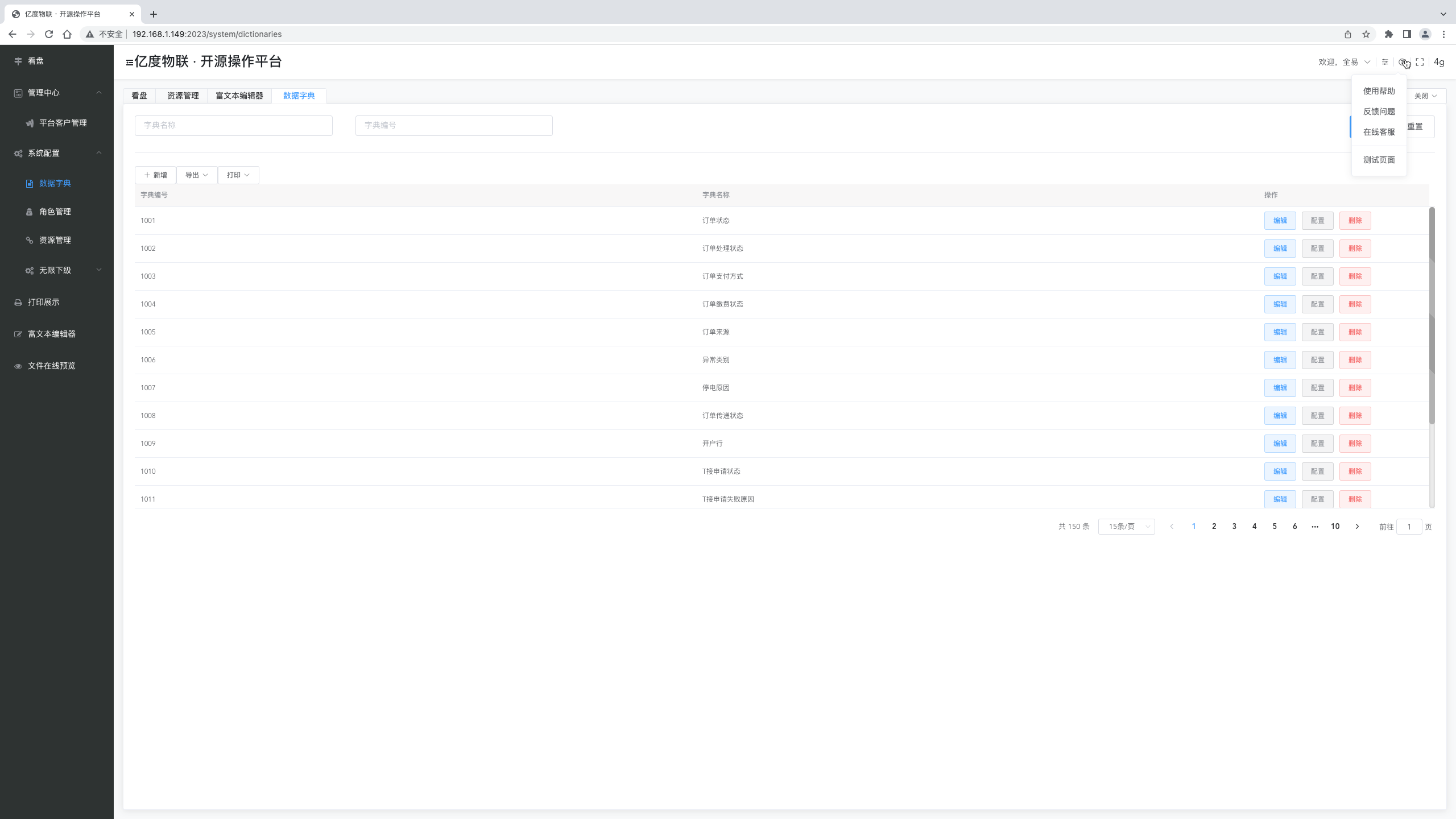
《开源操作平台》使用说明

这是一个轻量级的操作平台前端解决方案，基于[vue2x](https://v2.cn.vuejs.org/v2/guide/)开发框架和热门的 [element-ui](https://element.eleme.cn/#/zh-CN/component/installation)组件库实现。使用了Vue全家桶技术栈、打包Gzip技术，内置了动态路由、权限验证、请求拦截、响应拦截、接口封装，引入了[bootstrap-icons](https://icons.bootcss.com/)、[Font Awesome](https://fontawesome.com.cn/v4/icons)4等图标库，封装了丰富的复制文本、数字千位符、cdn方法、全局监听摁钮的防抖、事件节流等实用小功能。集成了一系列常用好用的小插件，免去技术选型和技术试错的困扰。运用了vue过滤器、vue指令、vue混入、vue组件，将vue的优势发挥的淋漓尽致。最大的优点就是足够轻量，易拓展、好上手、可塑性强。适用多数开发场景





# 项目对比

示例目前市场长使用较多的项目对比：

|  |  |  |  |
| --- | --- | --- | --- |
| 项目 | edo-peoject | [vue-admin-template](https://gitee.com/panjiachen/vue-admin-template) | [jeecg](https://gitee.com/jeecg/ant-design-vue-jeecg) |
| 目录结构 |  |  |  |
| 语法 | 通俗易懂 | 简单 | 复杂 |
| 可塑性 | 极强 | 强 | 弱 |
| 体积 | 微小 | 大 | 超级臃肿 |
| 耦合度 | 纯前端 | 纯前端 | 前端+后端 |
| 上手程度 | 半年从业者即可 | 一年以上 | 四年以上 |

# 技术栈

* 开发框架：[vue2x](https://v2.cn.vuejs.org/v2/guide/)
* 运行环境：[node](https://nodejs.org/zh-cn)16.15.1
* Vue全家桶：[vueX](https://vuex.vuejs.org/zh/)、[vue-router3x](https://v3.router.vuejs.org/zh/installation.html)
* 文件在线预览
* 打印：[printJs](https://printjs.crabbly.com/)
* Excel导入导出：[xlsx](https://www.npmjs.com/package/xlsx)、[xlsx-style](https://www.npmjs.com/package/xlsx-style)
* 富文本编辑器：[vue-quill-editor](https://www.npmjs.com/package/vue-quill-editor)

不用的插件可以卸载以减少项目体积

# 目录结构

├── public # 静态资源（打包不编译）

├ ├── service/demo # 接口演示数据

├ └── index.html # 内容模板

├── src # 源码编辑区

├ ├── assets/ # 静态资源（打包会编译）

├ ├── components/ # 封装好的常用组件

├ ├── config/ # 业务配置目录

├ ├── directives/ # 指令

├ ├── filters/ # 过滤器

├ ├── mixins/ # 混入

├ ├── pages/ # 页面

├ ├── router/ # 路由

├ ├── service/ # 请求

├ ├── store/ # 本地存储

├ ├── utils/ # 小工具

├ ├── App.vue # 页面入口

├ └── main.js # 全局入口

├── .env # 全局环境变量配置

├── .env.development # 开发环境变量配置

├── .env.production # 生产环境变量配置

├── .env.test # 测试环境变量配置

├── babel.config.js # babel-loader 配置

├── jsconfig.json #

├── package-lock.json # 锁定依赖包

├── package.json # 依赖清单

├── vue.config.js # vue脚手架配置

# 安装使用

目前只提供公司内网安装：

# 下载

git clone http://192.168.99.118/pubilc/frontend/init-vue-project

# 进入项目

cd init-vue-project

# 安装依赖（建议不要cnpm安装，会有各种诡异的bug）

npm install

# 如果安装慢，可以通过如下操作解决

npm install --registry=https://registry.npmmirror.com

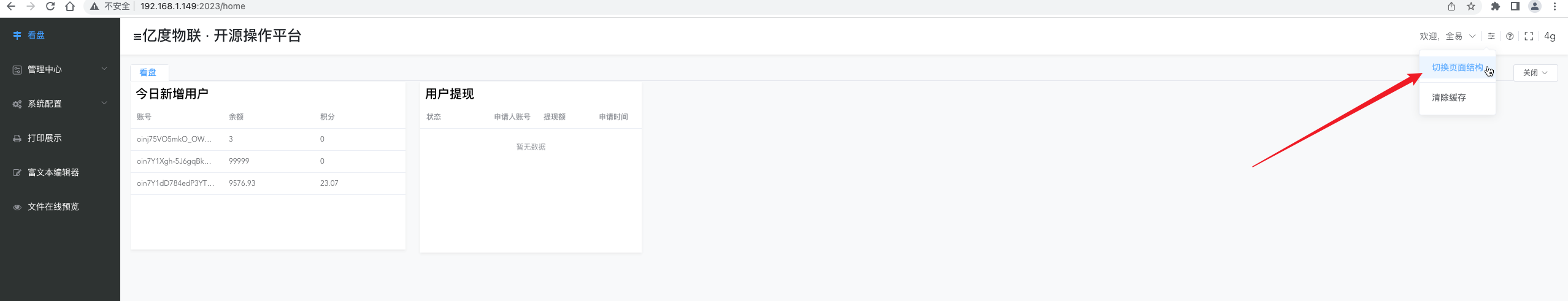
# 本地启动项目

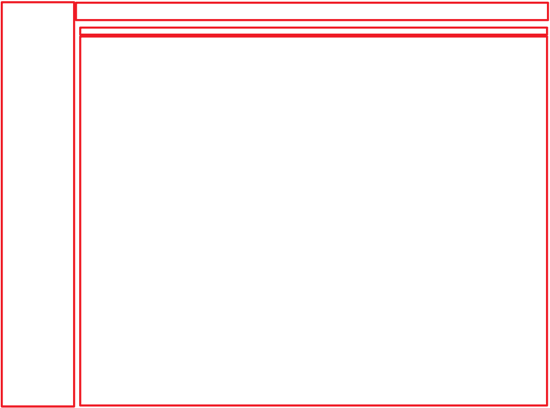
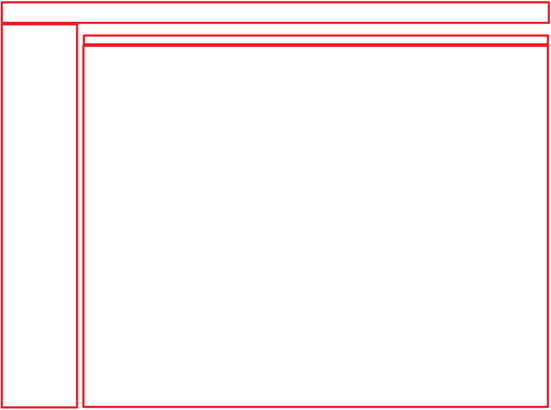
npm run serve

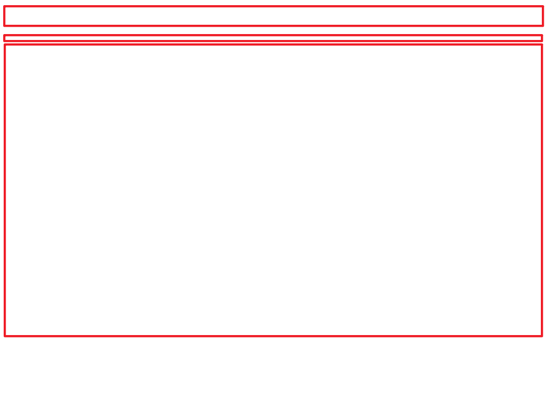
启动完成后会自动打开浏览器访问 http://ip:2023，接下来你可以修改代码进行业务开发了

# 布局说明

提供两种布局选择：

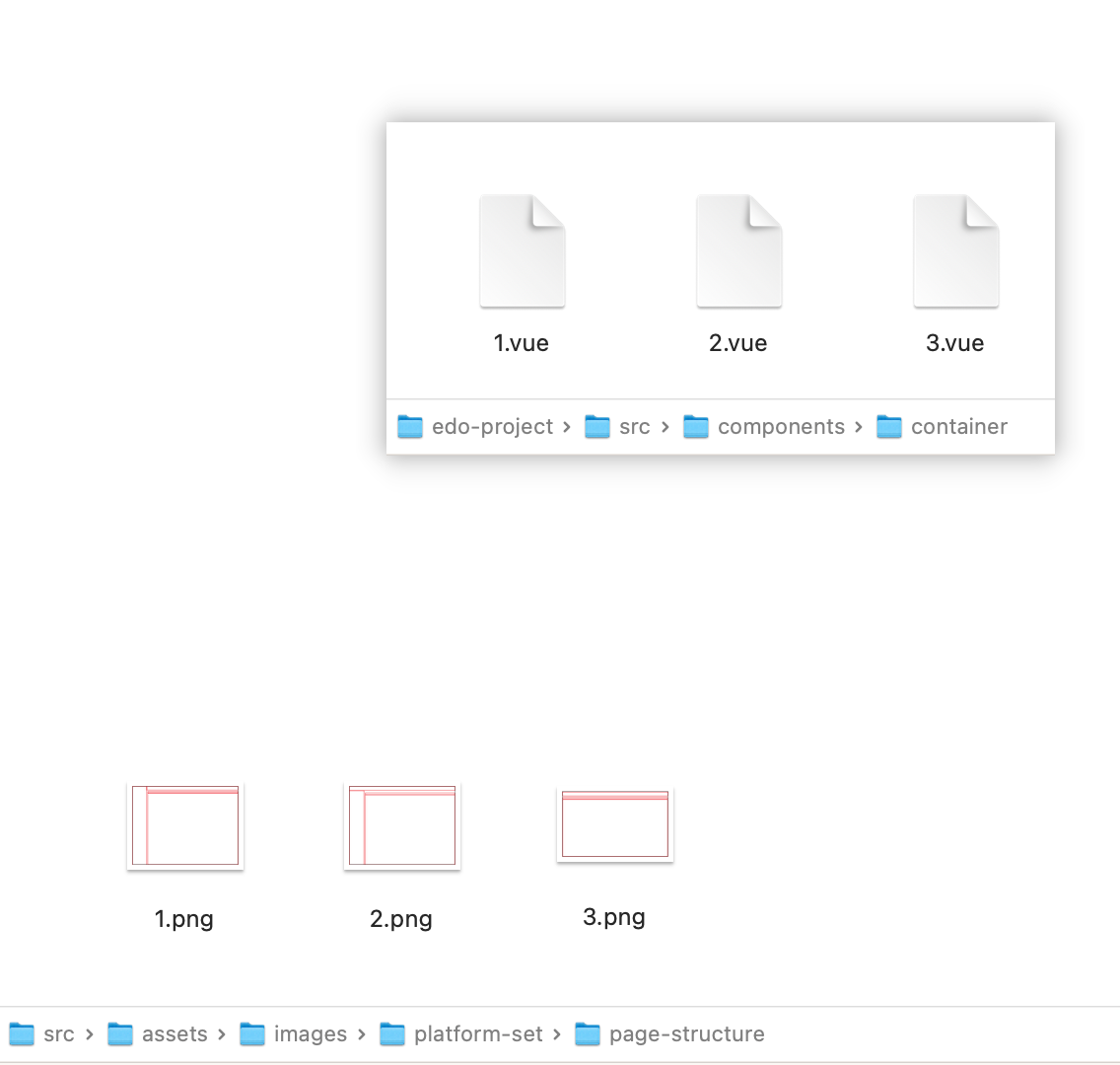




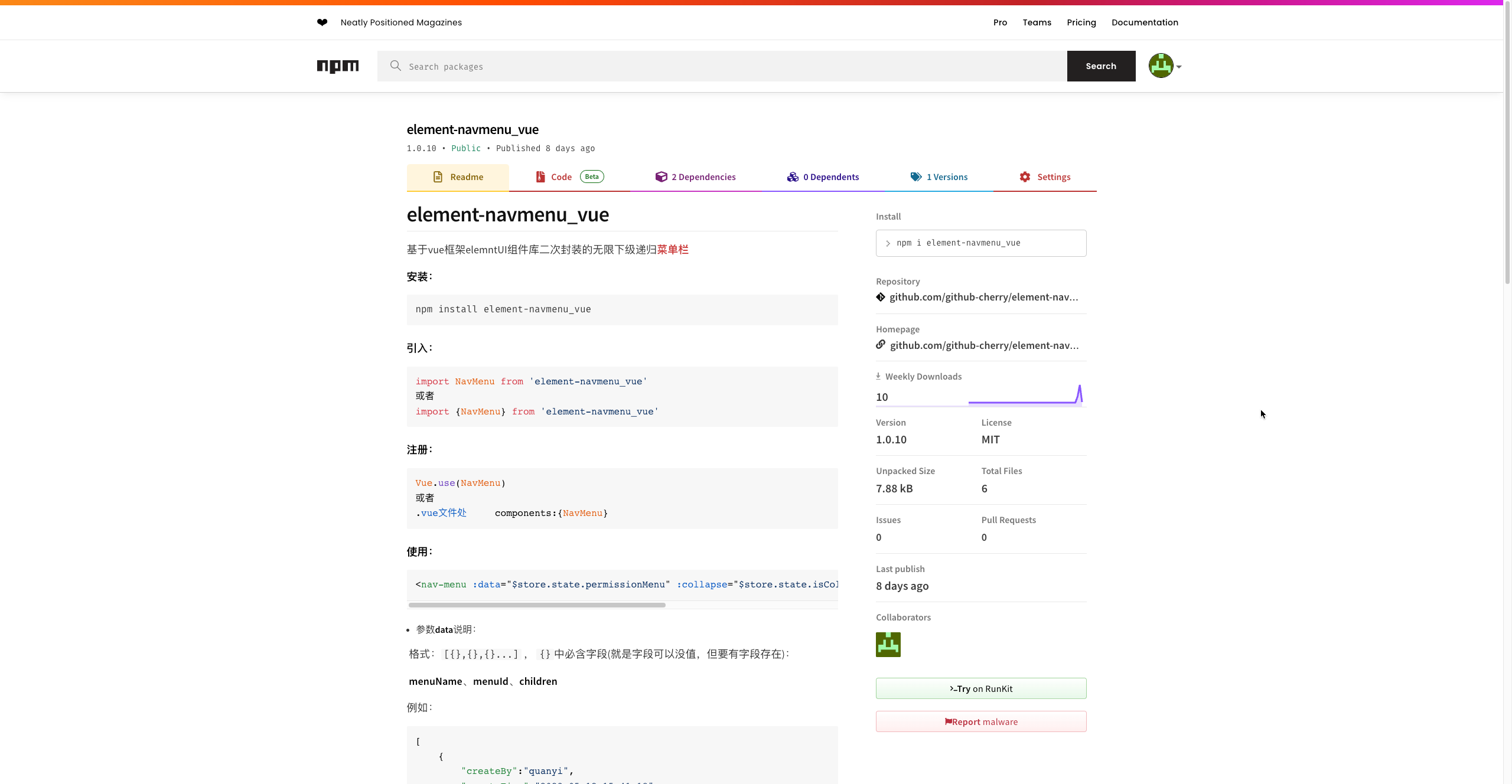
login、404页面没有走container布局。因为业务区域包裹了keep-alive为了缓存router-view，配合页签使用，如不需缓存可自行去除。

可选布局是和已定布局有关联的，@/assets/images/platform-set/page-structure/有几个可选布局，那@/src/components/container/就需要提供几个布局方式。



## 菜单栏

安装的组件[element-navmenu\_vue](https://www.npmjs.com/package/element-navmenu_vue)，按照说明使用即可

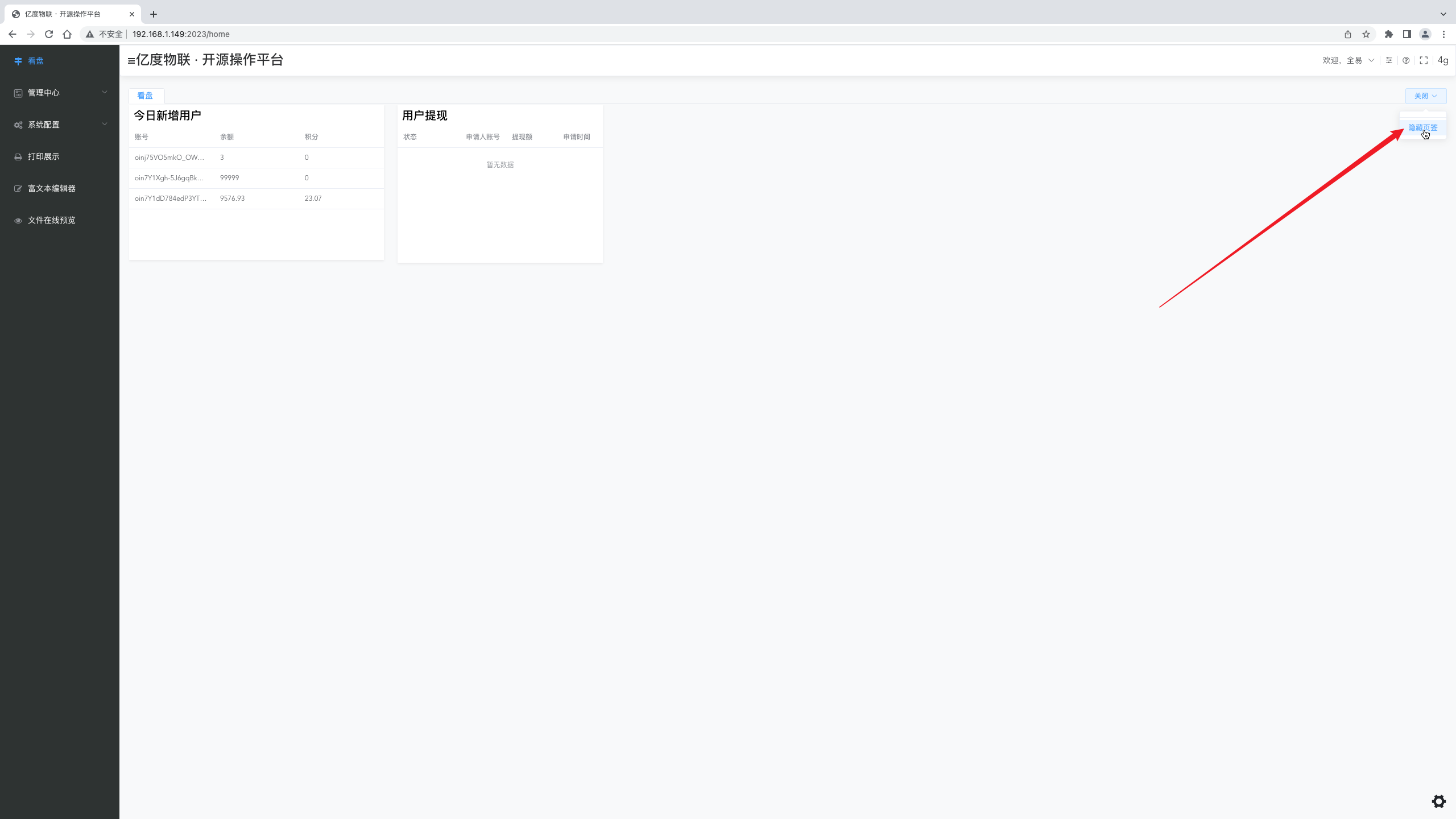


## 顶部条

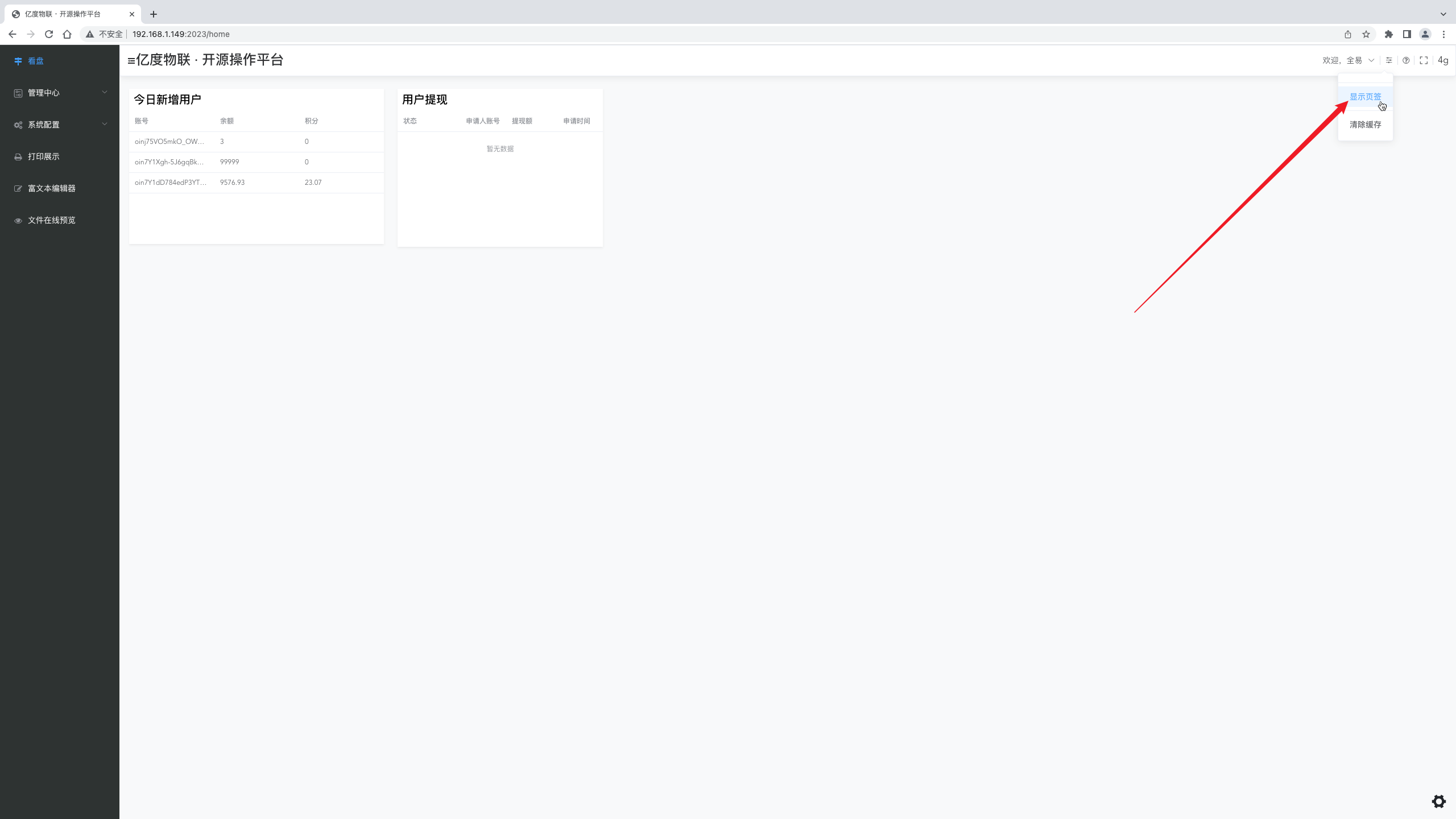
@/src/components/nav.vue 左侧放了应用的名称，右侧是平台的设置和用户信息等

## 页签

@/src/components/opened-page.vue 方便用户查看已打开的页面。右上角可以设置是否显示页签：

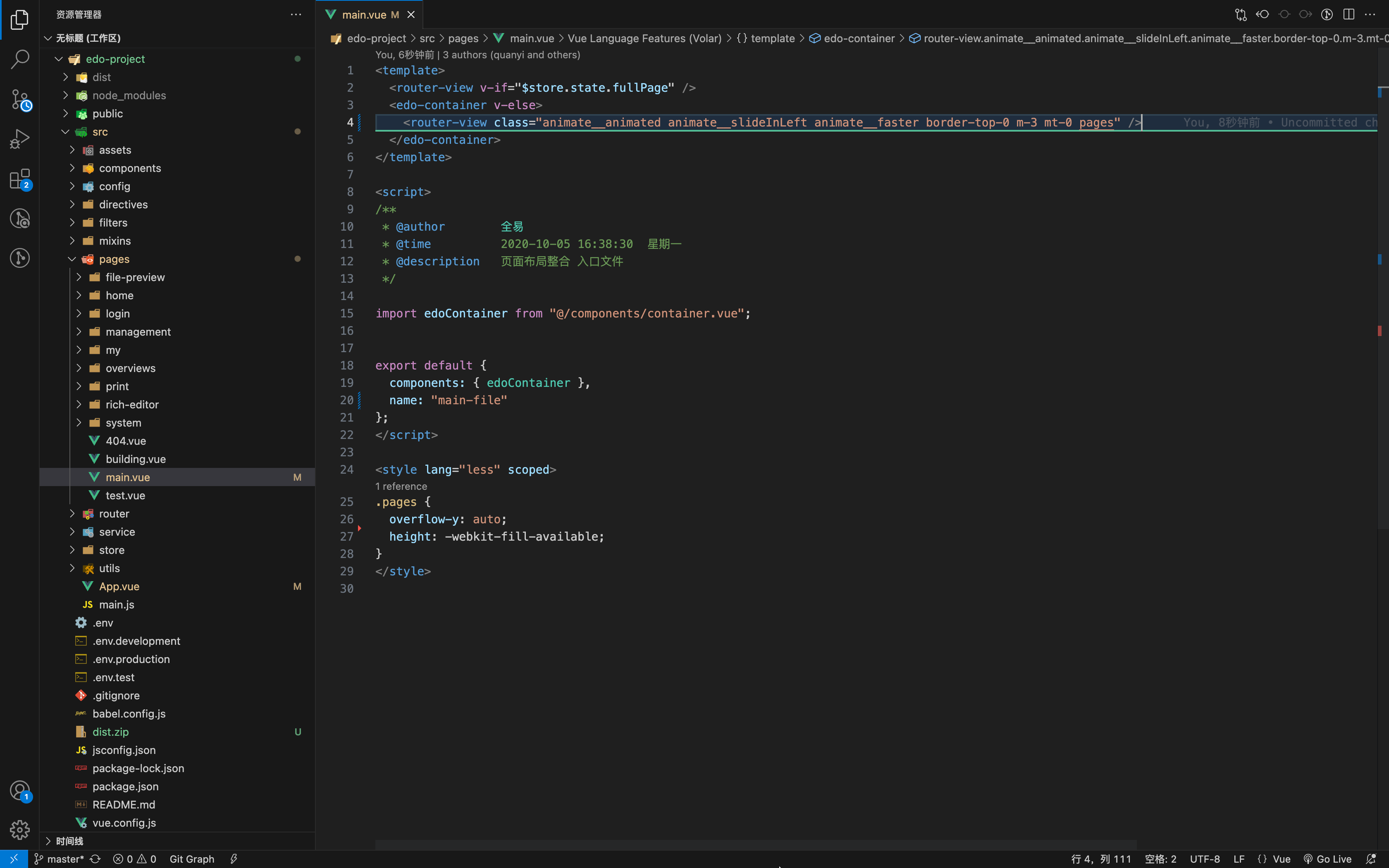


当已隐藏时想再显示，在顶部条可以打开：



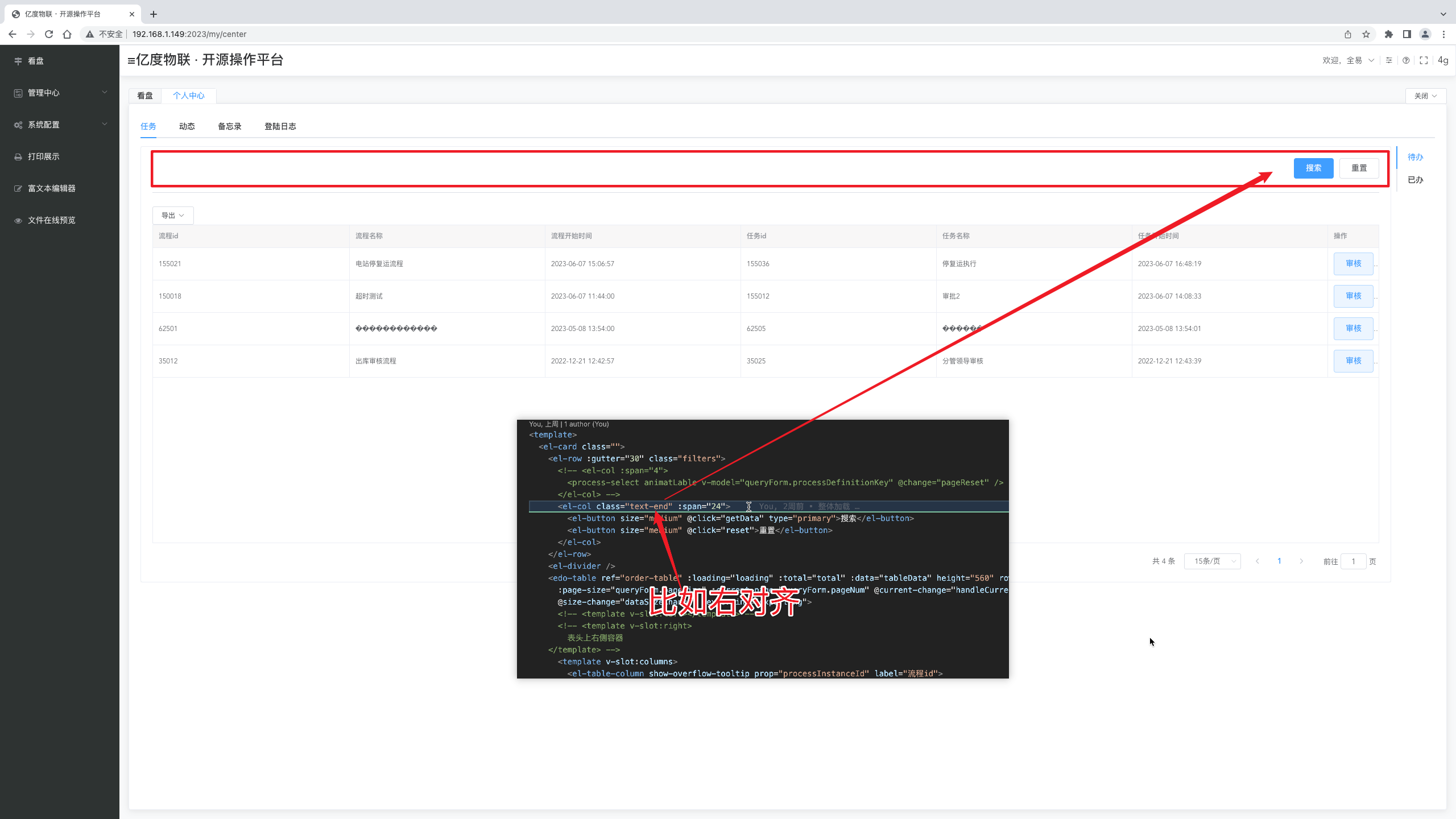
## 业务区域

使用了[animate.css](https://animate.style/)动画库显示页面切换时的动画，如需调整可以在@/src/pages/main.vue调整router-view的class



# 全局样式

引入了[Bootstrap5](https://v5.bootcss.com/docs/getting-started/introduction/)的离线css文件@/src/assets/bootstrap.min.css可以直接使用[官方文档](https://v5.bootcss.com/docs/getting-started/introduction/)的类名即可，尽可能自己少些冗余样式，比如padding、marging、border……还有很多便捷的类名，请自行到官网查看：[https://v5.bootcss.com/docs/getting-started/introduction](https://v5.bootcss.com/docs/getting-started/introduction/)



# 图标

cdn引入了[bootstrap-icons](https://icons.bootcss.com/)、[Font Awesome](https://fontawesome.com.cn/v4/icons)4，直接使用类名即可

# 主题

平台主题样式可以参考[这里](https://blog.csdn.net/qq_42618566/article/details/106643049)，原理就是按照elementUI官方提供的主题选择器确定主题色，下载主题包，放到项目里，在将将import 'element-ui/lib/theme-chalk/index.css'替换为import '@/assets/css/theme/index.css'即可

# 服务交互

所有的接口请求都会过@/service/index.js文件，已配置好了请求拦截、响应拦截

## 请求拦截

会检测需要token参数的接口，从localStorage里取token参数放入接口headers['Authorization']里，没有token就退出登录到/login页面。



此处可根据自己的业务进行改动

## 响应拦截

做了[响应失败](https://www.runoob.com/http/http-status-codes.html)的状态码可视化处理：

switch (error.response.status) {

case 302:

Message.error('302：重定向');

break;

case 400:

Message.error('400：请求错误');

break;

case 401:

Message.error('401：登陆过期');

break;

case 403:

Message.error('403：拒绝访问');

break;

case 404:

Message.error('404：无此接口');

break;

case 408:

Message.error('408：请求超时');

break;

case 500:

Message.error('500：服务器错误');

break;

case 501:

Message.error('501：服务未实现');

break;

case 502:

Message.error('502：网关错误');

break;

case 503:

Message.error('503：服务不可用');

break;

case 504:

Message.error('504：网关超时');

break;

case 505:

Message.error('505：HTTP版本不受支持');

break;

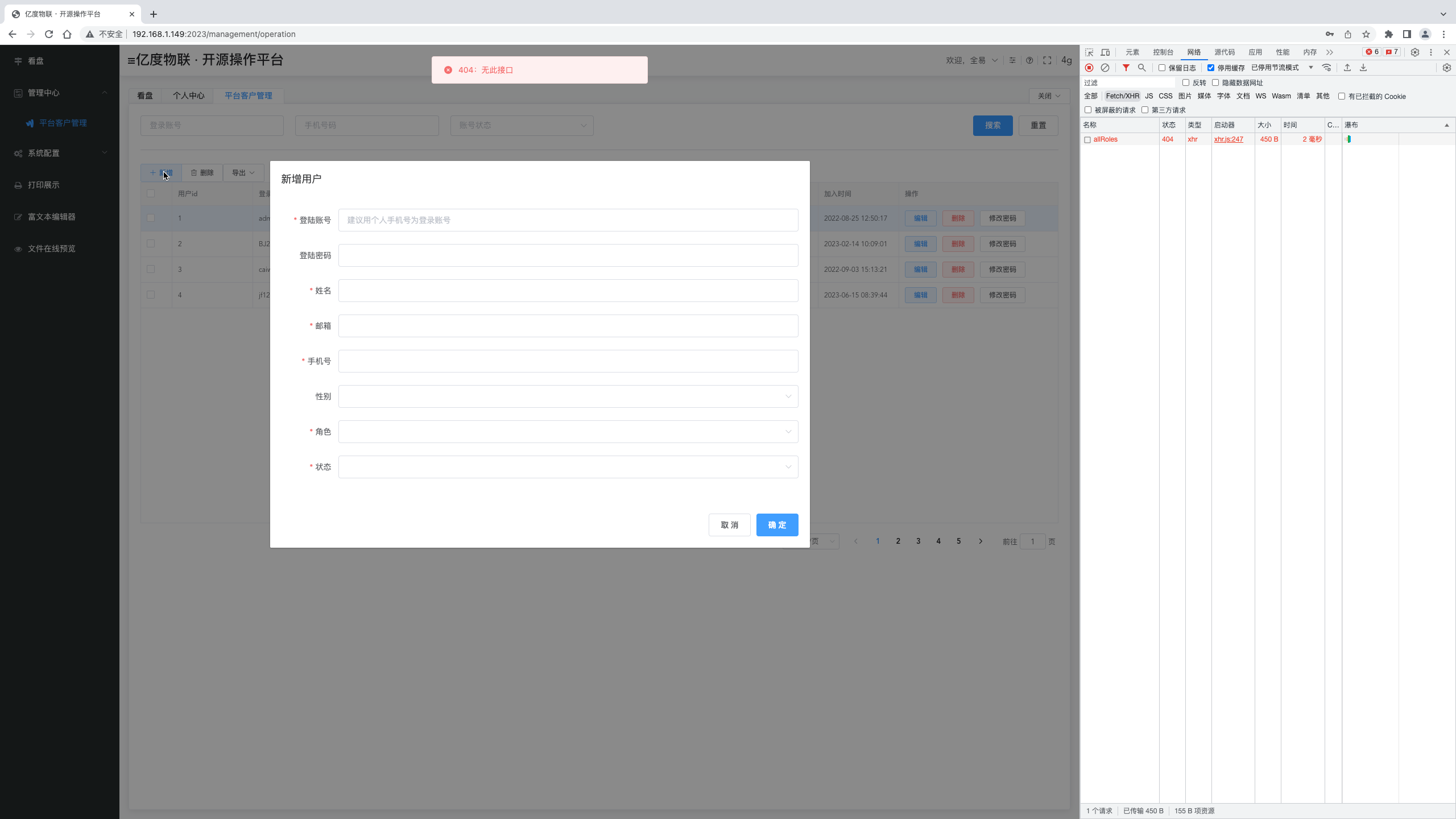
default:

Message.error('服务失败');

break;

}

当未正常响应时会页面顶部会出现警告提示，例如：



对于业务的异常也做了处理，示例的接口返回数据结构是：

{

code: 0,

msg: "",

data:{}

}

业务正常code是0，所以接口响应做了异常的提示

if (response.data.code !== 0) {

Message.error(response.data.msg);

}

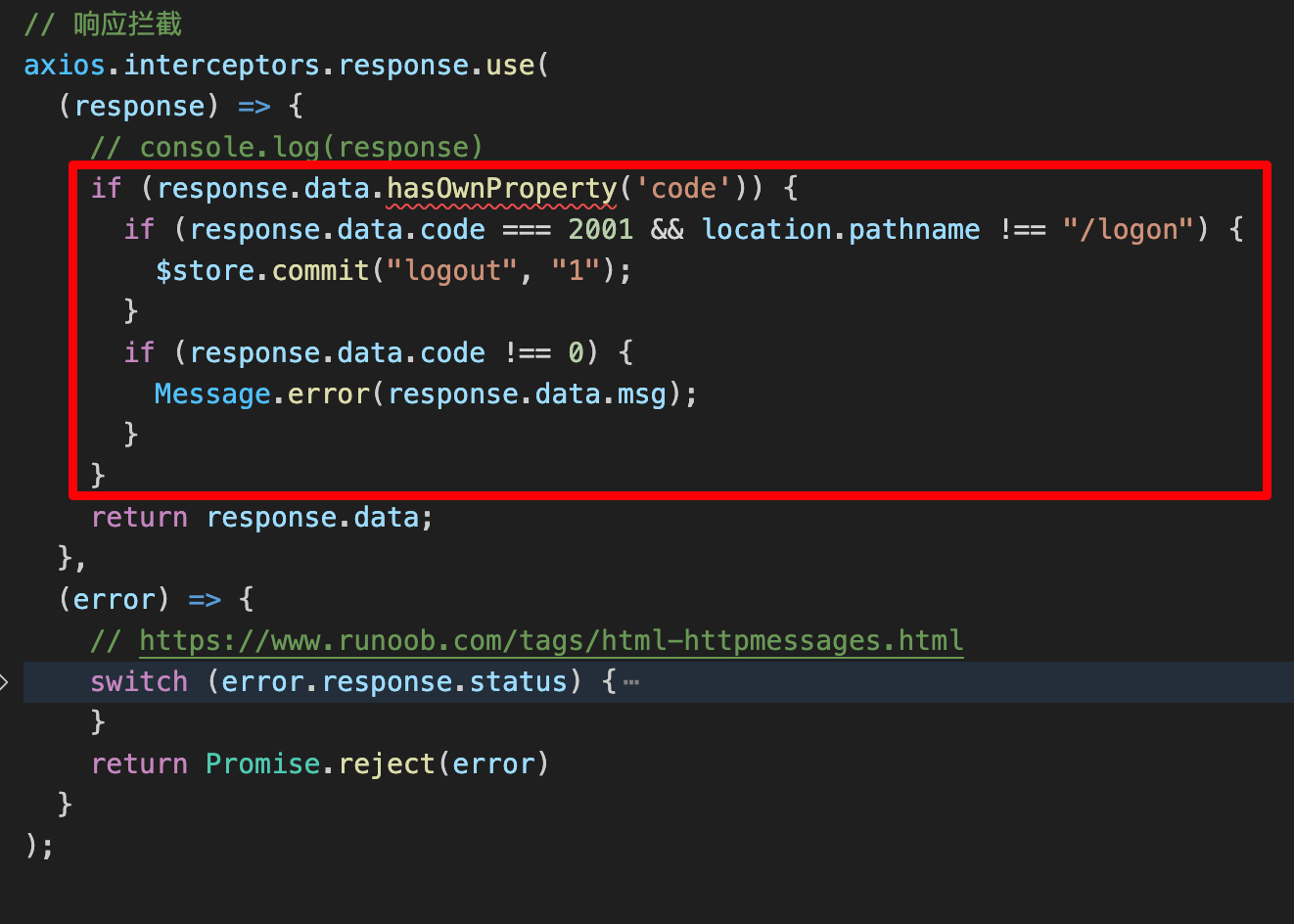
当前业务用户没登录返回的code是2001，所以：

if (response.data.code===2001&&location.pathname!=="/logon"){

$store.commit("logout", "1");

}

这些业务可根据自己的业务调整



## 环境变量

项目是基于 [vue-cli](https://cli.vuejs.org/zh/guide/mode-and-env.html)构建的，所以所有的环境变量配置都是基于[vue-cli](https://cli.vuejs.org/zh/guide/mode-and-env.html)来实现和控制的。

.env # 在所有的环境中被载入

.env.[mode] # 只在指定的模式中被载入

一个环境文件只包含环境变量的“键=值”对：

VUE\_APP\_SECRET=secret

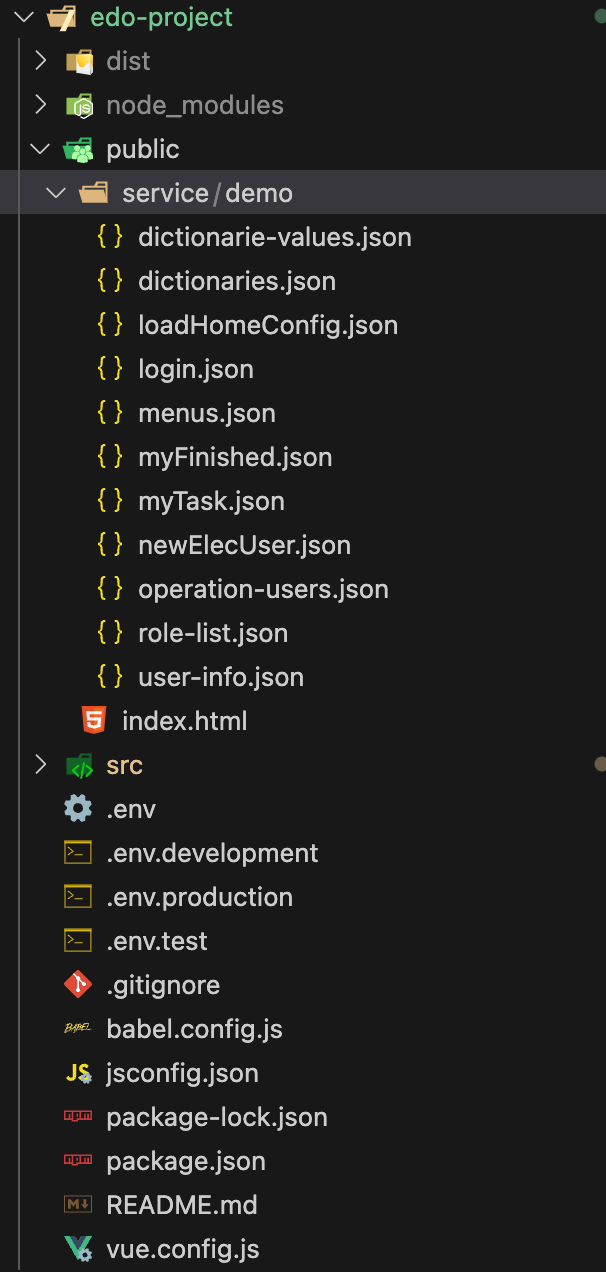
|  |
| --- |
| **注意**  环境变量必须以VUE\_APP\_为开头。如:VUE\_APP\_API、VUE\_APP\_TITLE  你在代码中可以通过如下方式获取:  console.log(process.env.VUE\_APP\_xxxx) |

除了 VUE\_APP\_\* 变量之外，在你的应用代码中始终可用的还有两个特殊的变量：

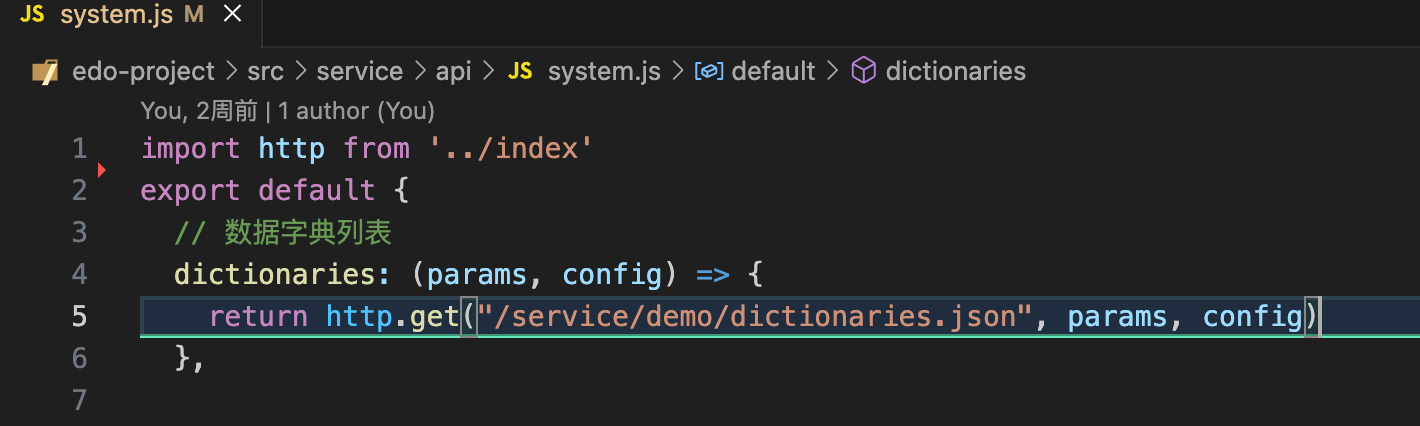
* NODE\_ENV 会是 "development"、"production" 或 "test" 中的一个。具体的值取决于应用运行的模式。
* BASE\_URL 会和 vue.config.js 中的 publicPath 选项相符，即你的应用会部署到的基础路径。

# 演示数据

在public/service/demo/ 弄了几个主要功能的演示数据，在真正开发中删除整个service目录



然后接口请求处改成你业务中用的接口路径，比如数据字典：



@/service/index.js文件的请求基路径记得放开：**axios.defaults.baseURL**



# 登录

@/pages/login.vue文件，分为3步才算登录成功：

1. **登录：**通过返回的token存储到localStorage里，在[请求拦截](#_请求拦截)时业务接口会用到。
2. **获取用户信息：**响应成功后存入stroe
3. **获取路由权限：**该接口返回的数据包含菜单、摁钮等权限，做数据处理后存到stroe里

然后就跳转了

# 路由数据处理

@/store/mutations.js文件setMenu、setRouters方法

## setMenu

处理菜单栏需要显示的数据：排除掉摁钮和全局路由的数据，然后将扁平的数组进行递归循环结构化树形数据。

你也可根据自己的业务处理需要显示在菜单栏的数据。

全局路由指的是没有显示在菜单栏但在平台任何地方可以有入口的路由，比如**个人中心**和**我的资料**



## setRouters

是处理所有合法路由访问权限的数据，包括全局路由、菜单栏的路由。过滤所有没有url的数据，合并全局路由和接口数据，进行去重。动态获取路由指向的资源，同时处理摁钮权限。最后合并路由

@/src/store/mutations.js

// 处理权限路由数据

setRouters(state, data) {

// console.log("路由信息：", data);

// 只保留有url的路由

let permissionPublic = []

const retainURL = data.filter((item) => {

// console.log(item);

if (item.url !== "") {

permissionPublic.push(item.perms)

state.permissionPublic = permissionPublic;

return item

}

});

// console.log("路由数据：", retainURL);

// 存储路由

state.permissionRouters = publicPath;

state.permissionRouters.push(...retainURL.map(item => {

return item.url;

}));

state.permissionRouters = [...new Set(state.permissionRouters)]; // 去重

// 路由结构化

const structuringRoute = retainURL.map(item => {

let component;

try {

component = () => import(`@/pages/${item.remark}.vue`)

} catch (err) {

component = () => import(`@/pages/404.vue`)

}

return {

path: item.url,

name: item.remark.replace(/\//g, "-"),

component,

meta: {

...item,

title: item.menuName,

id: item.menuIdStr,

permissions: item.children.map(it => {

if (it.menuType === "F") {

return {

...it,

title: it.menuName,

id: it.menuIdStr,

}

}

})

}

}

});

// console.log("结构化的路由：", JSON.stringify(structuringRoute));

// 合并路由

$router.addRoute({

path: '/',

name: 'main-file',

component: () => import("@/pages/main.vue"),

redirect: "/home",

children: structuringRoute

});

},

# 路由懒加载

当打包构建应用时，Javascript 包会变得非常大，影响页面加载速度。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

结合 [Vue](https://v2.cn.vuejs.org/v2/guide/components-dynamic-async.html)的异步组件和 [Webpack](https://www.webpackjs.com/guides/code-splitting/)的代码分割功能，轻松实现路由组件的懒加载。如@/src/store/mutations.js的setRouters。通过接口给的数据进行异步获取资源

try {

component = () => import(`@/pages/${item.remark}.vue`)

} catch (err) {

component = () => import(`@/pages/404.vue`)

}

将首屏加载的压力分散到每个页面

# 导航守卫

@/src/router/guards.js

## 全局前置守卫

路由每次发生变化前校验localStorage里有没有token。

* **没有：**就清除平台数据并退出到logon页面
* **有：** 进行校验目标页的路由有没有越权

没有：就清除平台数据并退出到logon页面

有：放行

根据自己的业务可自行调整

// 全局前置守卫

$router.beforeEach((to, from, next) => {

// console.log(to);

// 2/1. 判断登陆情况

const token = localStorage.getItem("token");

if (!token && to.path !== "/logon") {

return store.commit("logout", "1");

}

// 2/2. 判断访问越权

const paths = store.state.permissionRouters;

if (!paths.includes(to.path)) {

return next({ path: '/404' })

}

// 满足以上条件，才能过

next();

})

## 全局后置钩子

页面跳转后执行页签push、设置菜单栏的停留页id、读取本页权限摁钮、区分路由是不是全屏显示

|  |
| --- |
| 全屏：不是传统理解里全屏幕覆盖的意思，而是隐藏菜单栏、顶部条。将[业务区铺](#_布局说明)满浏览器，在平台填写资源数据时menuType字段是0的话，证明该页面是全屏 |

根据自己的业务可自行调整

// 全局后置钩子

$router.afterEach((to, from) => {

// console.log(to, from)

store.commit("setOpenedPages", {

fullPath: to.fullPath,

meta: to.meta,

name: to.name,

path: to.path,

})

store.commit("setDefaultPage", to.meta.id);

store.commit("setPermissionButtons", to.meta.permissions);

store.commit("changeFullPage", to.meta.menuType === "0");

})

# keep-alive

在@/src/components/container/使用了vue提供的**<keep-alive>**组件，配置了exclude和include

* **exclude：**在@/config/exclude-keep-alive-pages.json固定配置
* **include：**在@/src/App.vue监听页签变化，将已打开的页面数据加入includeKeepAlivePages变量进行缓存。用于打开过的页面再次进去就不重新拉取数据。关闭页签就代表不再缓存页面，再次进入便重新经过组件生命周期拉取拉取数据

watch:{

"$store.state.openedPages": {

deep: true, // 深度监听 属性的变化

handler(now) {

// console.log(now) //页签长度发生变化说明缓存也要发生变化

this.includeKeepAlivePages = now.map(item=>{

return item.name

})

}

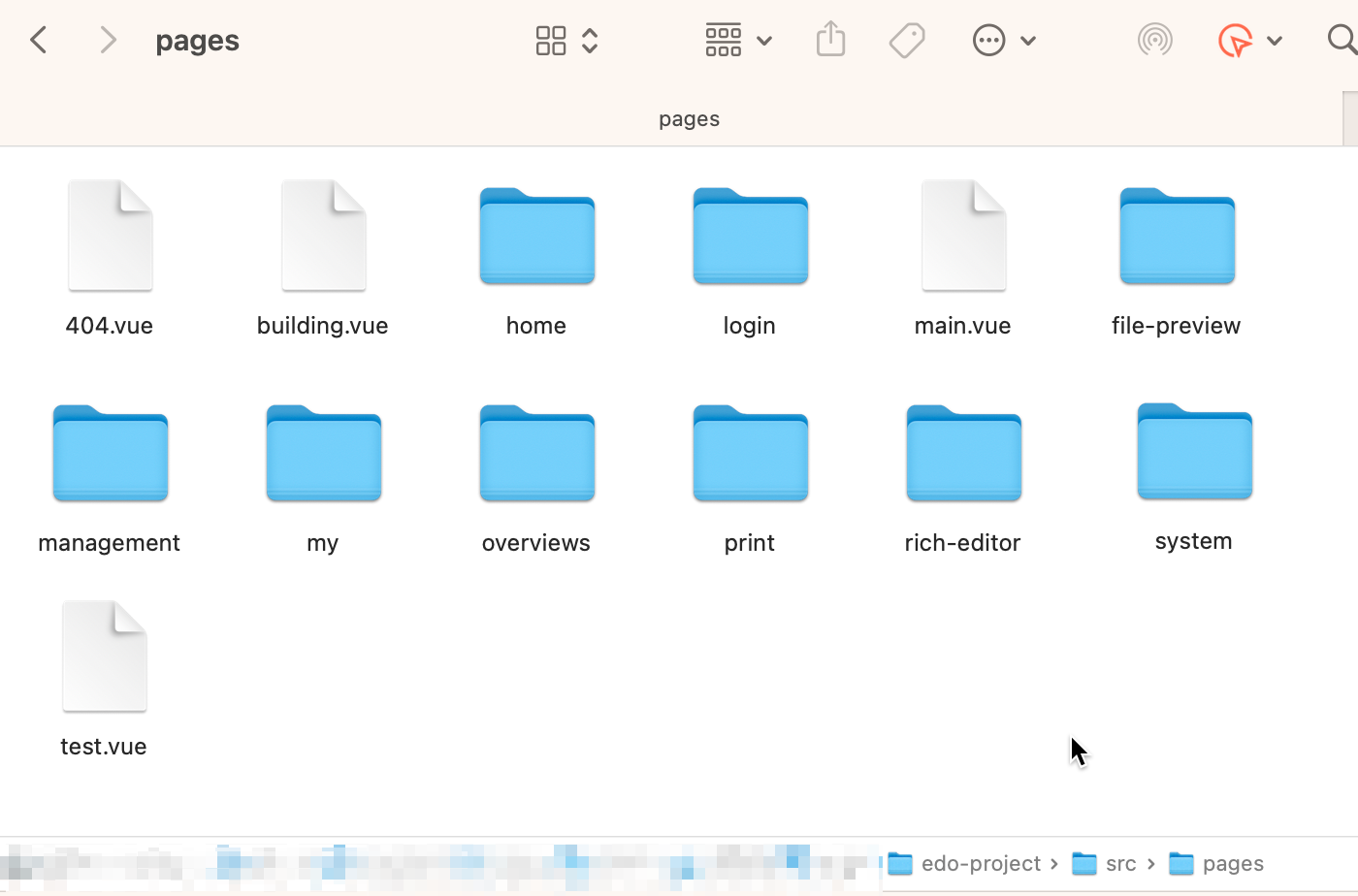
},

},

# 新增资源

## 项目增加资源

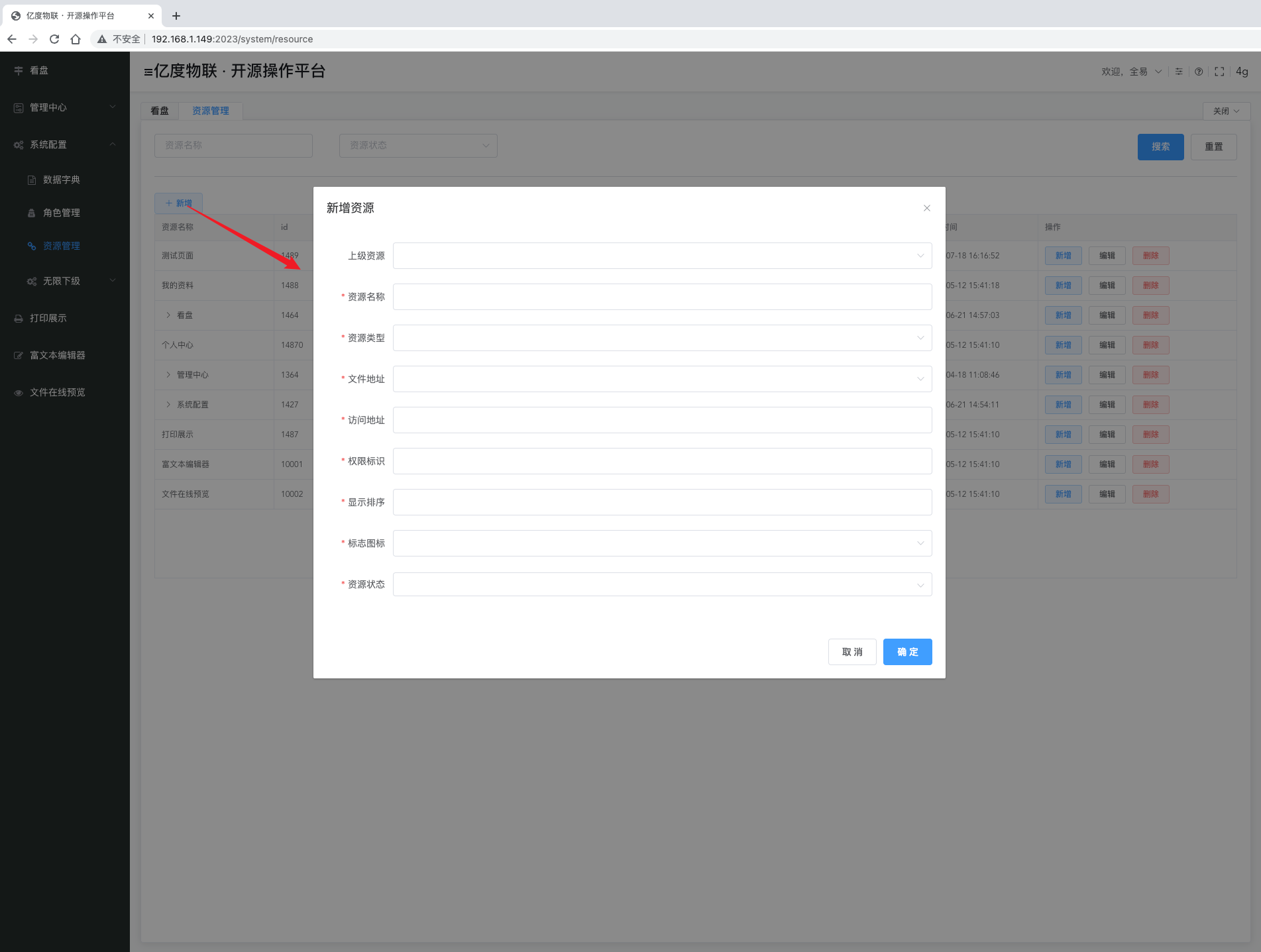
在项目的@/src/pages/里都是浏览器要显示的页面资源



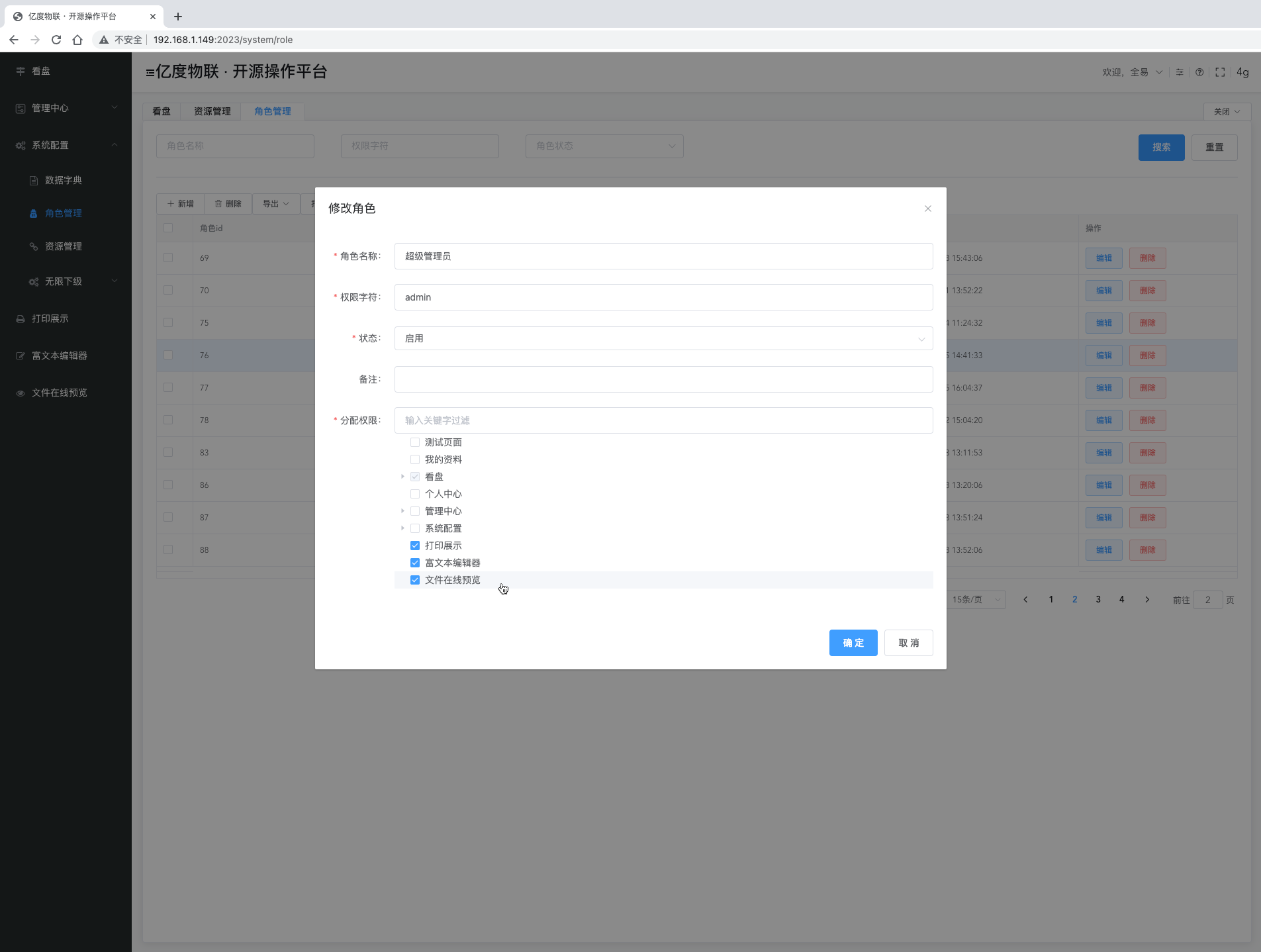
在此目录里增加页面目录和页面文件即可

## 可视化配置资源

项目里有了代码后，在浏览器/system/resource处配置资源：



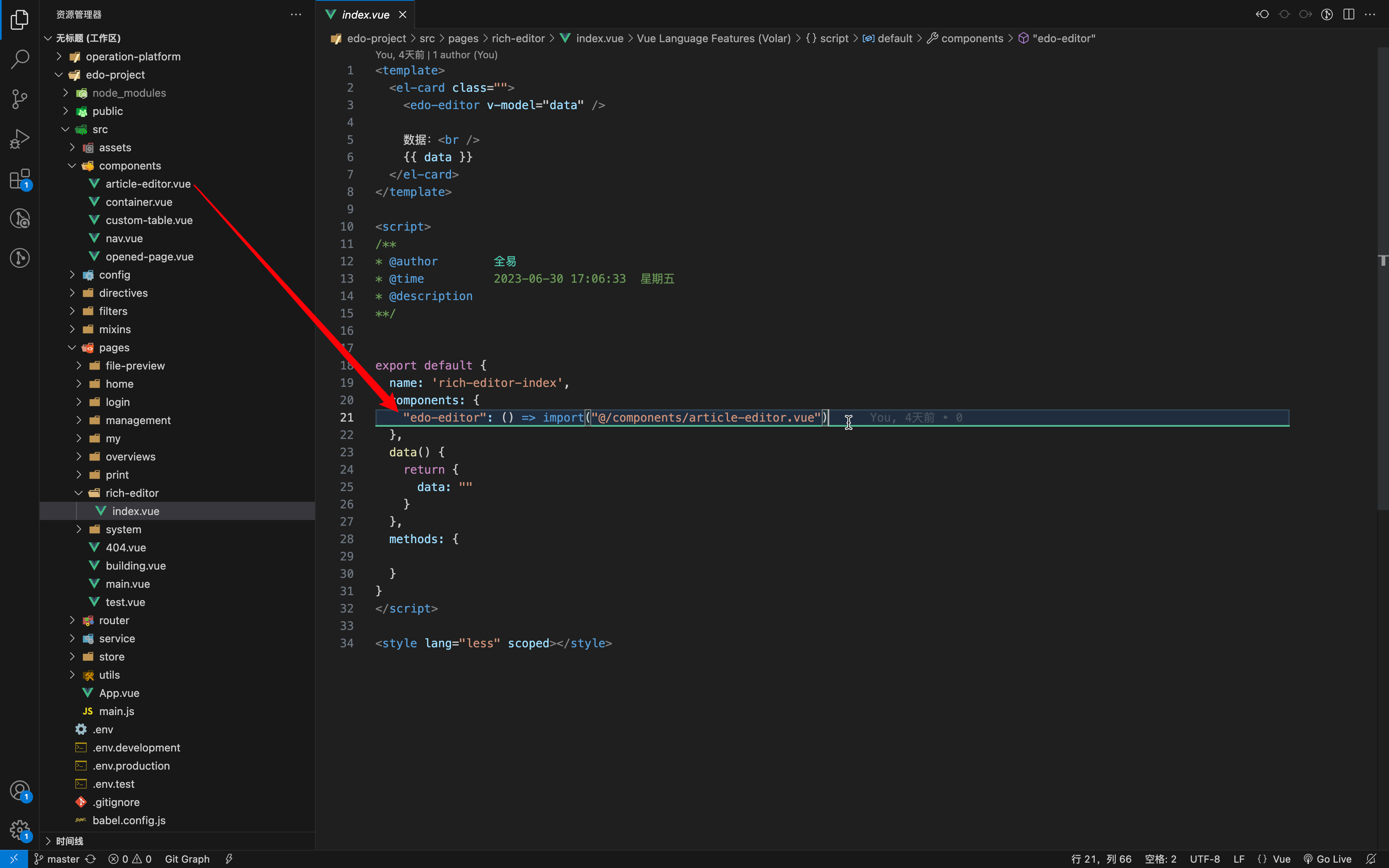
按照要求填写提交数据，需要给谁分配资源，就要确定他的账号是什么角色，在/system/role页面给角色勾选上你刚创建的资源即可：

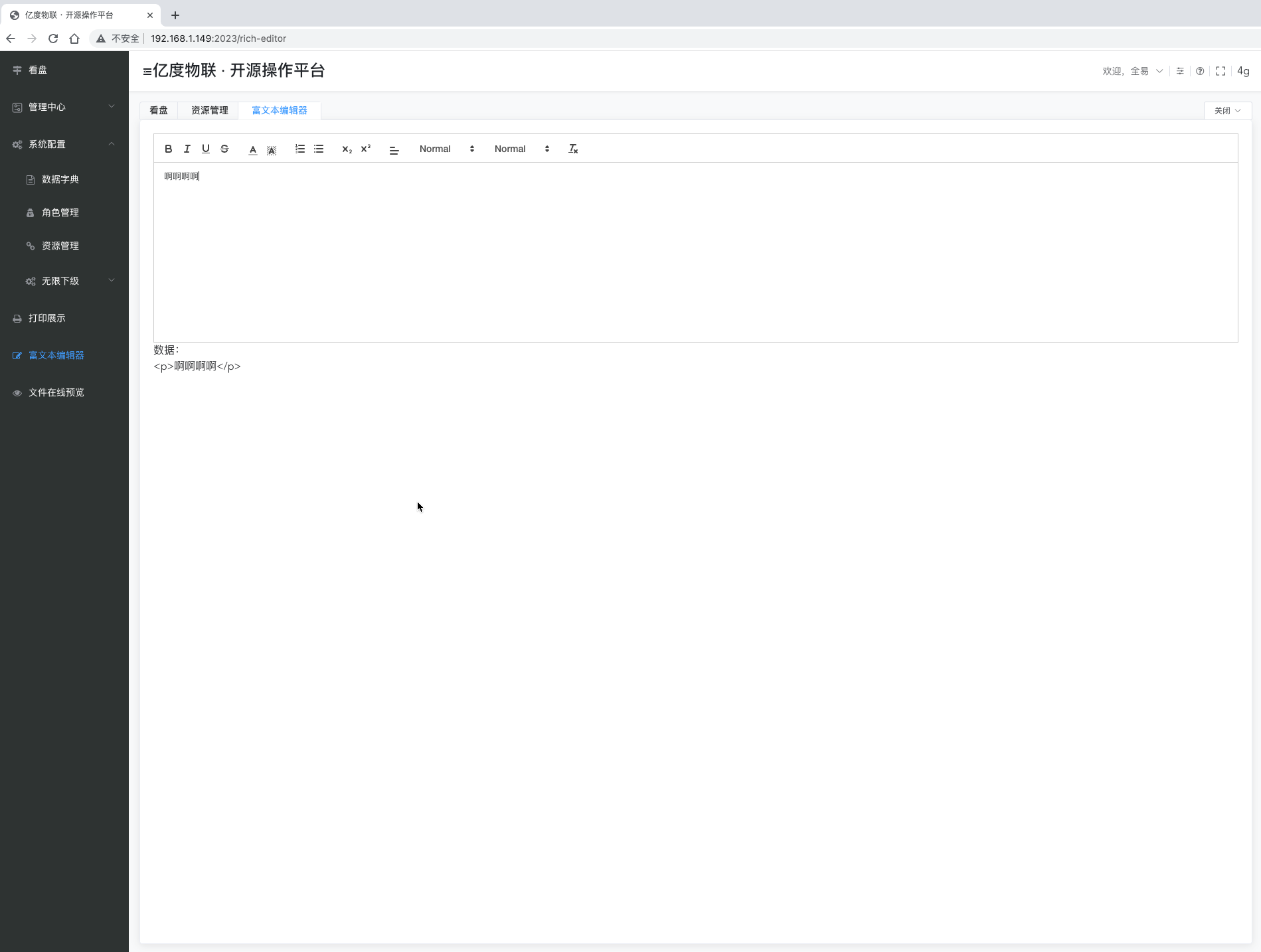


## 组件

* 全局组件

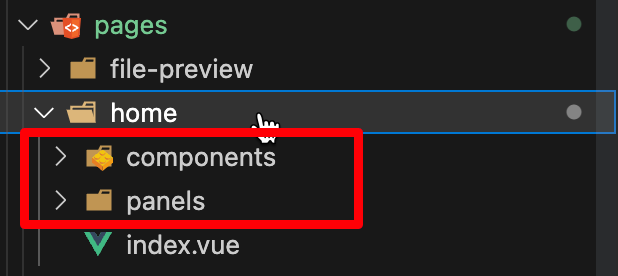
在@/src/components/创建全局的组件后，哪里需要引入即可，比如：



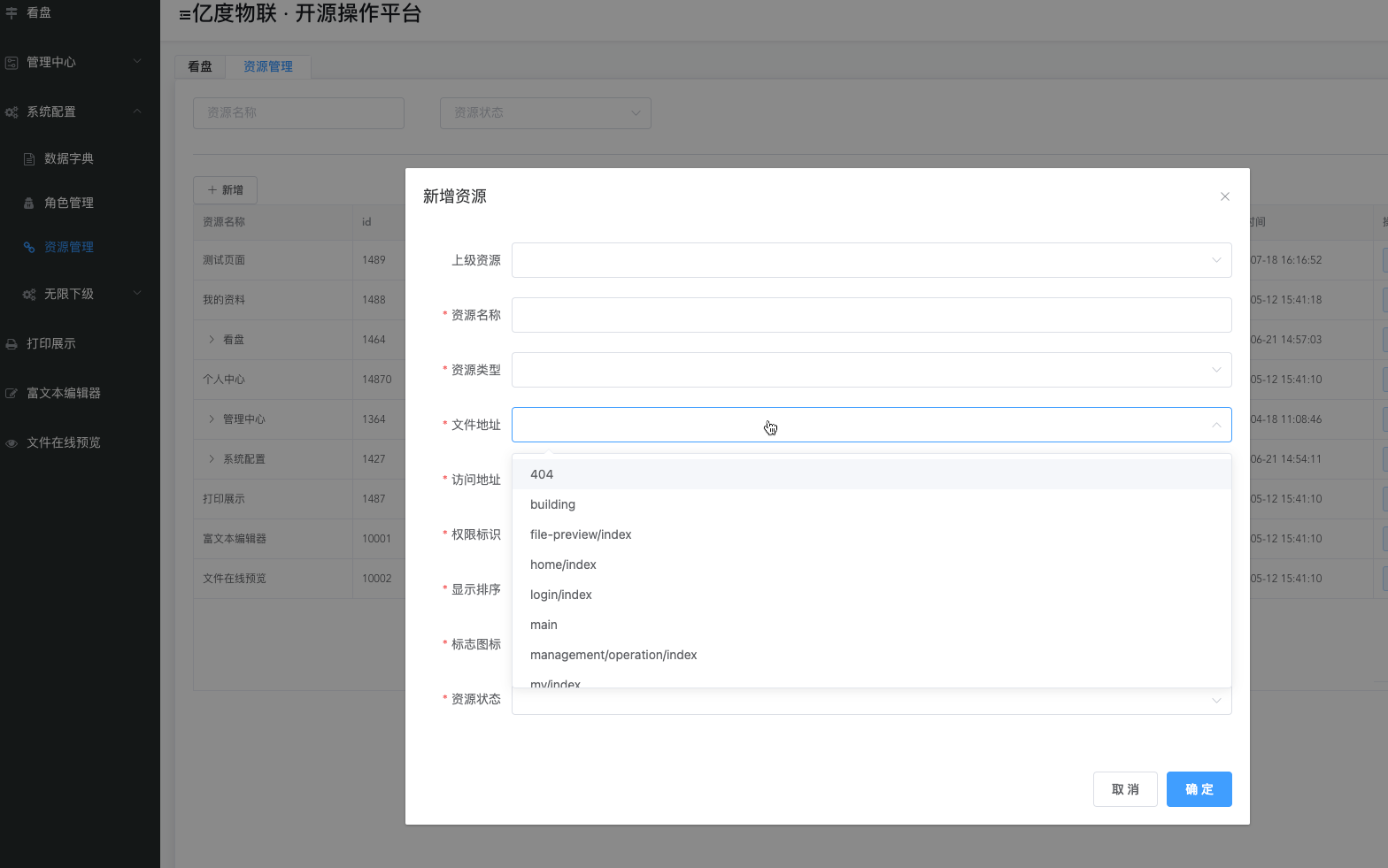


* 局部组件

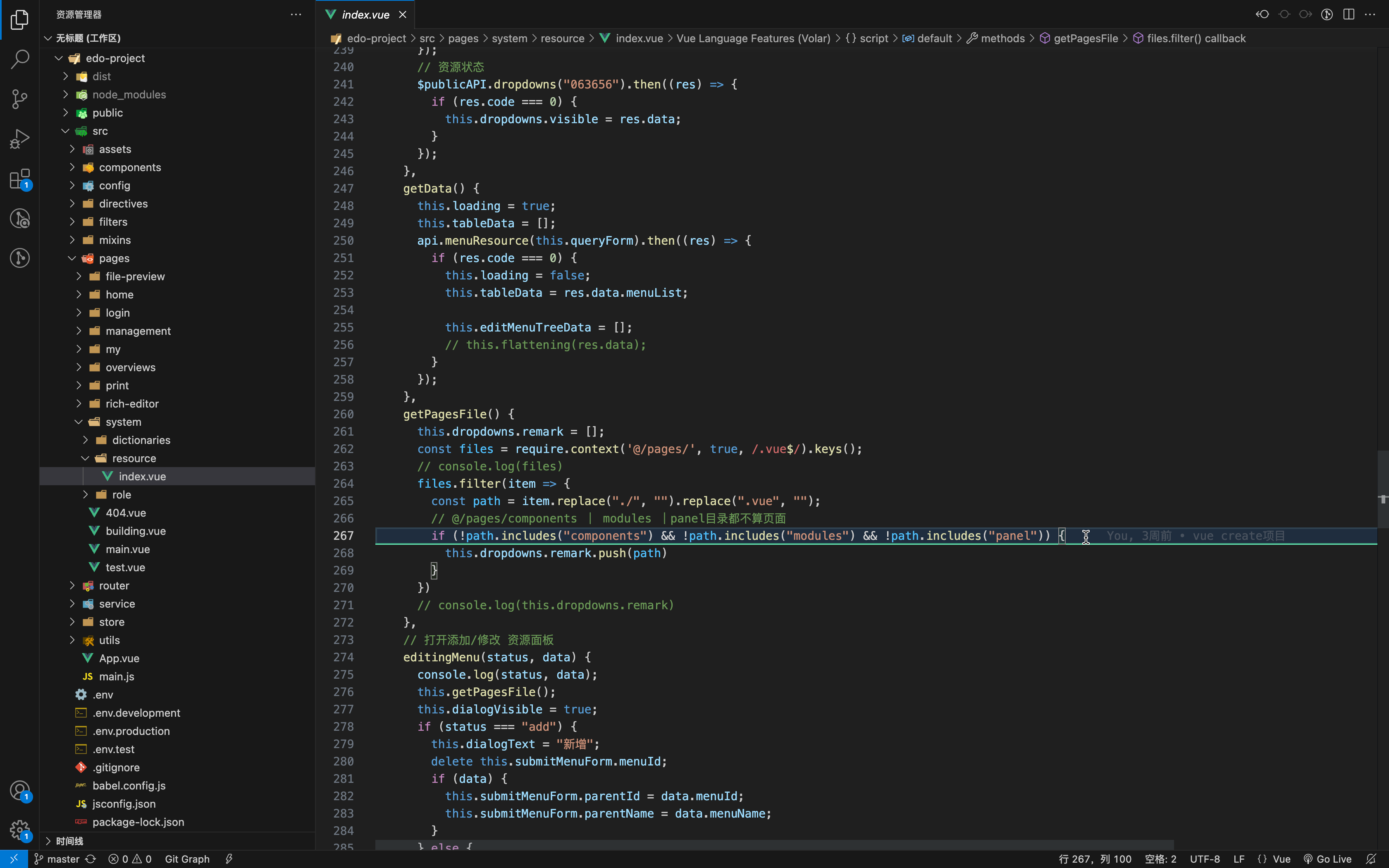
@/src/pages/components/页面目录/components/，比如首页：



页面目录下的components、modules、panel目录代表都是页面的模块，而非页面，在浏览器创建资源时，选择文件地址这个字段有做区分：



getPagesFile()



你也可以在此调整你的业务字段

**请记住拆分组件最大的好处不是公用而是可维护性！**

## 样式

* [全局样式](#_全局样式)

1. 优先使用[Bootstrap5](https://v5.bootcss.com/docs/getting-started/introduction/)的类名
2. 可以自己创建全局样式@/src/assets/css/里可以自己创建全局样式，然后@/src/App.vue文件style处引入

* 局部样式

由于[Bootstrap5](https://v5.bootcss.com/docs/getting-started/introduction/)的离线css文件提供了几乎所有的样式类，自己基本很少有写样式的需求，在页面文件的style写即可：

<template>

<div></div>

</template>

<script>

/\*\*

\* @author 全易

\* @time 2023-07-17 15:43:24 星期一

\* @description

\*\*/

export default {

name: '',

data () {

return {

}

}

}

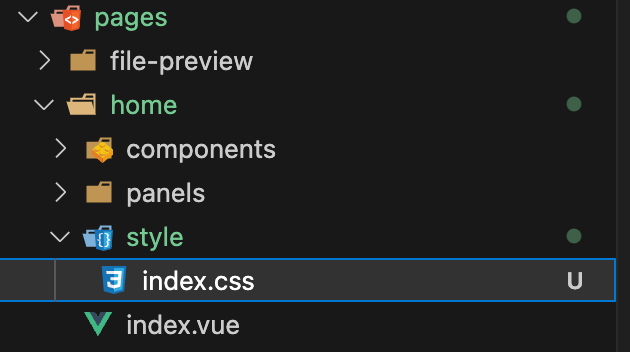
</script>

<style lang="less" scoped>

**在此直接写即可**

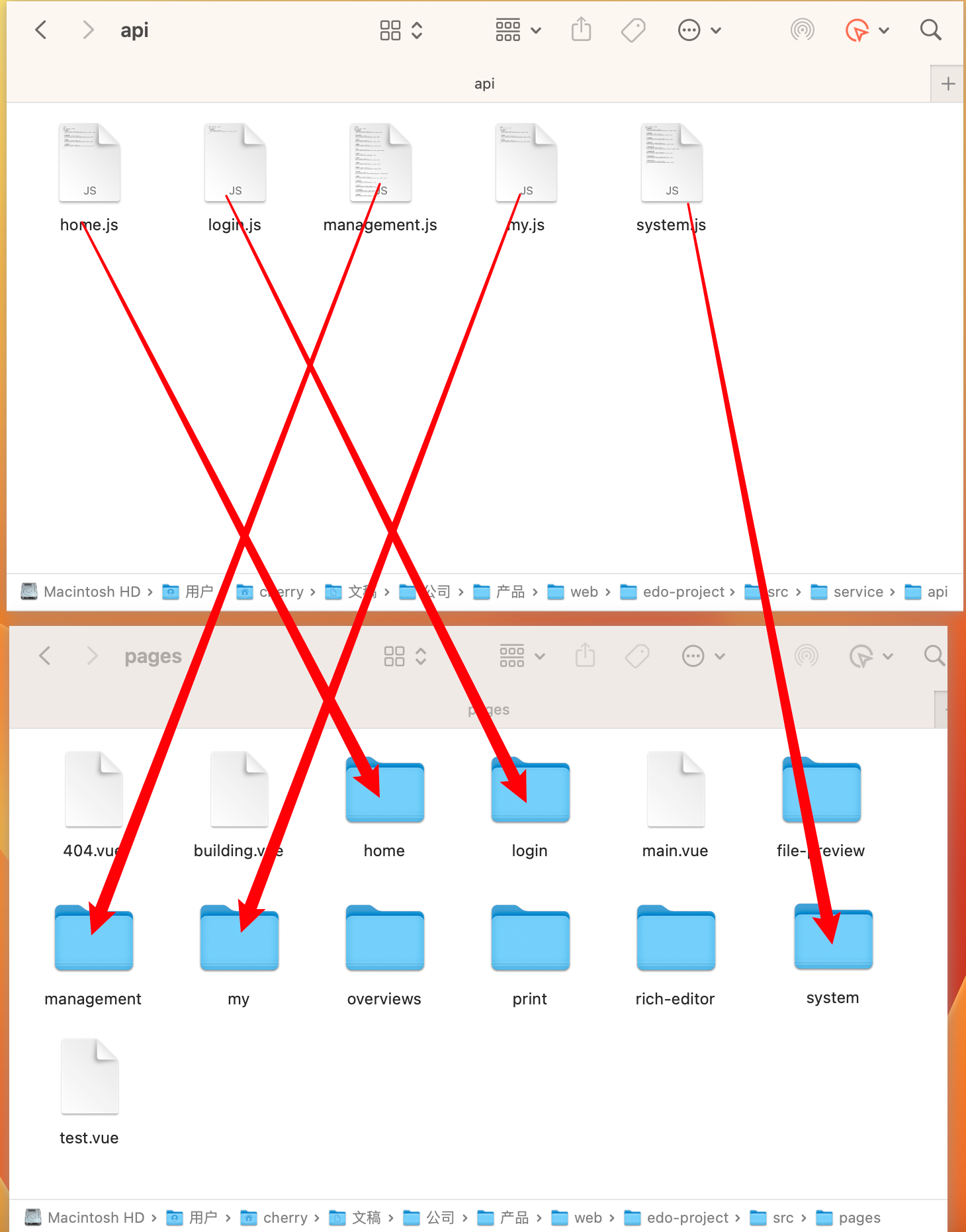
</style>

如果确实还是有很多样式代码，可以在此页面目录增加独立的样式文件，然后引入即可，例如首页：



## 接口

在@/src/service/api/里添加，最好对应@/src/pages/的页面目录，例如：



# 内置的封装

## @/src/utils/cdn.js

**initAll()**执行@/src/config/cdn-css.json和@/src/config/cdn-scripts.json的依赖

**loadScript(url)**：传入url单独加载脚本

**loadStyle(url)**：传入url单独加载样式

## @/src/utils/common.js

封装了一些常用的时间处理函数

## @/src/utils/debounce.js

封装了全局点击事件防抖的防抖处理，不允许持续不停的点击。至少800ms一次

## @/src/utils/export-file.js

搭配xlsx和xlsx-style插件二次封装了导出excel方法，增加了表头的背景色、数据宽度自适应

## @/src/utils/fetch-micro-app.js

接入了micro-app微应用，本项目为主应用。此处示例接入子应用

## @/src/utils/reader-file.js

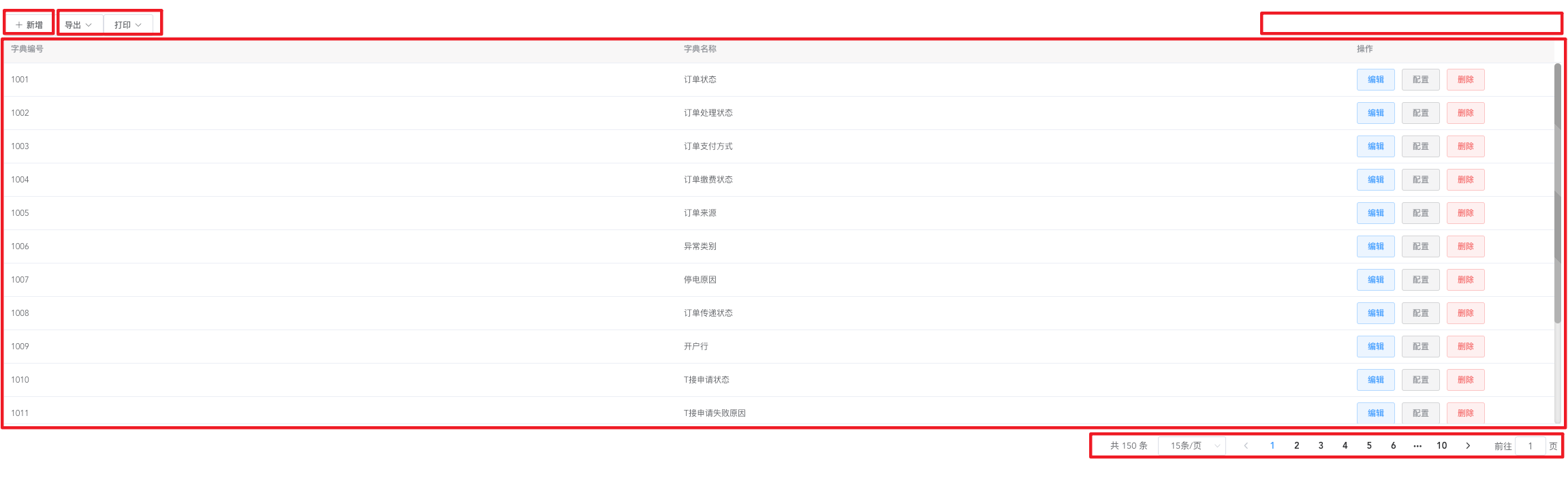
使用xlsx读取excel文件为json数据

## @/src/utils/throttle.js

封装节流的方法

## 组合表格

@/src/components/custom-table.vue，结合了功能区、表格和翻页



功能区：分为左侧和右侧，两侧都预留了插槽可使用处增加摁钮。提供了导出和打印事件

表格区：数据呈现

分页区：执行数据翻页

## 复制文本指令

@/src/directives/index.js

v-copy

## 数字千位符

@/src/filters/index.js

<div>{{ 12345679 | toThousands }}</div>

结果：123,456,789

## 混入

有些方法每个页面都会涉及，那每个页面都写一遍就臃肿了。此处采用混入的方式封装常用方法，植入到每个页面大大降低了冗余代码

* 页码重置

@/src/filters/page-reset.js

比如现在是第5页，你增加个搜索条件后再搜肯定需要还原页码，那么给每个搜索条件增加pageReset方法即可

import pageReset from "@/mixins/page-reset.js"

mixins: [pageReset],

<el-col :span="4">

<el-input clearable size="medium" v-model="queryForm.loginName" placeholder="登录账号" @change="pageReset"

@keyup.enter.native="getData"></el-input>

</el-col>

<el-col :span="4">

<el-input clearable size="medium" v-model="queryForm.phonenumber" placeholder="手机号码" @change="pageReset"

@keyup.enter.native="getData"></el-input>

</el-col>

<el-col :span="4">

<el-select filterable default-first-option clearable size="medium" v-model="queryForm.status" placeholder="账号状态" @change="pageReset">

<el-option v-for="item in dropdowns.status" :key="item.code\_value" :label="item.code\_name"

:value="item.code\_value"></el-option>

</el-select>

</el-col>

* 搜索条件重置

@/src/directives/reset-filter.js

每页的数据在条件过滤后要清除所有的参数并且页码还原

import resetFilter from "@/mixins/reset-filter.js"

mixins: [resetFilter],

<el-button size="medium" @click="reset">重置</el-button>

* 分页方法

@/src/directives/pagination.js

表格的size-change和current-change事件的方法

import pagination from "@/mixins/pagination.js"

mixins: [pagination],

<edo-table :loading="loading" :total="total" :data="tableData" height="600" row-key="userId"

:page-size="queryForm.pageSize" :current-page="queryForm.pageNum" @selection-change="selectingdata"

@size-change="dataSizeChange" @current-change="handleCurrentChange" @exporting="exporting">

</edo-table>

# 风格指南

主要参考vue官方的[风格指南](https://v2.cn.vuejs.org/v2/style-guide" \l "%E5%8D%95%E6%96%87%E4%BB%B6%E7%BB%84%E4%BB%B6%E6%96%87%E4%BB%B6%E5%90%8D%E7%9A%84%E5%A4%A7%E5%B0%8F%E5%86%99%E5%BC%BA%E7%83%88%E6%8E%A8%E8%8D%90)，规范开发使代码易读易维护。

# 性能优化说明

* 配置了Gzip压缩
* 使用模块异步导入
* 使用了webpack splitChunks依赖切片处理
* 部分依赖、样式文件cdn引入
* 图片、字体等静态资源采用服务器而非本地的引入方式
* 剔除臃肿注释，保持代码清爽易读
* 使用依赖时避免图省事在入口文件全局注册，而是在使用的地方按需加载

# 开发准备

1. 删除public/service/demo/演示数据，@/src/service/api/改成你的接口路径。@/src/service/index.js放开请求基路径。
2. 不需要的依赖记得删除，降低包体积。

# 开发建议

安装插件尽可能不要在@/src/main.js引入。保证主包的体积轻量，在需要的地方按需加载，比如[echarts](https://echarts.apache.org/handbook/zh/basics/import/" \l "%E6%8C%89%E9%9C%80%E5%BC%95%E5%85%A5-echarts-%E5%9B%BE%E8%A1%A8%E5%92%8C%E7%BB%84%E4%BB%B6)：

// 按需引入 ECharts 主模块

const echarts = require('echarts/lib/echarts')

// 引入柱状图

require('echarts/lib/chart/bar')

// 引入提示框和标题组件

require('echarts/lib/component/tooltip')

require('echarts/lib/component/title')

而不是图省事整个引入 import \* as echarts from 'echarts';

## 变量名

使用小驼峰命名法

## 方法名

使用小驼峰命名法

## 目录名

使用小写单词用 - 连接的方式

## 文件名

使用小写单词用 - 连接的方式

## 全局注册

无论是全局的组件、指令还是过滤器，尽可能的都不要在@/src/main.js引入，保证入口文件的轻量，坚决使用按需加载的方式。不然打包时都会放入主包，增加首屏加载的负荷。为了保证加载响应速度，不要使入口文件引入过多。

# 更多问题

[在线咨询](https://work.weixin.qq.com/kfid/kfcd894c0a8bc6d7627)