从零开发企业级中后台项目(实战篇)

🍉 这是 Vue 实战内容的开篇,本文将带着大家完成一个企业级中后台项目的搭建。

源码地址: https://git.yummy.tech/common/admin-demo

首先,笔者倡导的是拥抱变化,所以本文所有内容,都会采用最新且生产可用的技术栈来进行企业级 中后台的搭建。在此之前,推荐大家先看 日 从零开发企业级中后台项目(前置知识)这篇文章,并 对文章提及的技术栈有一定的了解。

本项目技术栈包括:

- · Vue 3
- Vuex 4.x
- Vue-router 4.x
- Vite

一、项目初始化

Vue 项目初始化最快、最稳妥的方式无非两种:

- · vue-cli
- · vite

当然也可以直接使用社区比较优秀的模板如:

无颜 (李鑫祥) 8187

- · vue-vben-admin
- · antd-vue-pro

本章主要介绍基于 Vite 快速初始化项目的方式,先来简单了解下。 Vite 是尤雨溪又一力作,号称下 一代开发构建工具,由于其支持原生 ES 模块导入方法,所以它允许快速提供代码,使得开发效率大 大提高。 如果读者好奇为什么要选 Vite ,可以看看官方文档给出的答案。

1.1 搭建项目

第一步,通过命令行搭建项目:

```
Bash

1 # 在终端执行下面命令

2 yarn create @vitejs/app
```

执行效果如图:

```
filee — lee@leedeMacBook — ~ — -zsh — 89×30
> yarn create @vitejs/app
yarn create evitejs/app
yarn create v1.19.1

[1/4] ■ Resolving packages...

[2/4] ➡ Fetching packages...

[3/4] Ø Linking dependencies...
warning "create-umi > sylvanas > Qumijs/fabric > Qtypescript-eslint/eslint-pluginQ1.13.0"
has incorrect peer dependency "Otypescript-eslint/parserO^1.9.0".
warning "create-umi > sylvanas > Oumijs/fabric > eslint-config-airbnbO17.1.1" has incorre
ct peer dependency "eslint-plugin-react@^7.14.2".
        Building fresh packages...
success Installed "@vitejs/create-app@2.4.3" with binaries:
        - create-app
        - cva
Project name: ... vite-project
 Select a framework: > vue
 ✓ Select a variant: > vue
Scaffolding project in /Users/lee/vite-project...
Done. Now run:
   cd vite-project
   yarn
   yarn dev

    Done in 21.46s.

                                                           无额 (李鑫祥) 8187
```

项目整体目录如图:



通过 Vite 命令行创建的项目 package.json 如下:

```
JSON
 1 {
 2
      "version": "0.0.0",
 3
      "scripts": {
 4
        "dev": "vite",
       "build": "vite build",
 5
 6
       "serve": "vite preview"
 7
      },
      "dependencies": {
      "vue": "^3.0.5"
 9
10
      },
      "devDependencies": {
11
        "@vitejs/plugin-vue": "^1.2.3",
12
13
        "@vue/compiler-sfc": "^3.0.5",
        "vite": "^2.3.7"
14
      }
15
16 }
```

1.2 启动项目

安装依赖并启动项目:

```
Bash

1 # 安装依赖
2 yarn
3 # 启动开发服务器
4 yarn dev
```

启动成功之后可以看到如下页面:



Hello Vue 3 + Vite

Vite Documentation | Vue 3 Documentation

count is: 0

Edit components/HelloWorld.vue to test hot module replacement.

二、项目配置

在开始开发前,可以做一些基础的配置来提升开发效率。

2.1 配置别名

避免在项目中使用路径时,写一长串路径字符,可设置常用别名如:

- · @ 代表 src
- · ~ 代表根目录下的别名

在 vite.config.js 下添加 resolve.alias , 代码如下:

```
JavaScript
    import { defineConfig } from "vite";
    import vue from "@vitejs/plugin-vue";
 2
 3
   // https://vitejs.dev/config/
 4
    export default defineConfig({
 5
      plugins: [vue()],
 6
 7
      resolve: {
        alias: {
 8
           "~": path.resolve(__dirname, "./"),
 9
           "@": path.resolve(__dirname, "src"),
10
11
        },
12
      },
13 });
```

2.2 代码片段

我们在开发过程中,通常会沉淀非常多的业务代码片段,提高开发效率的同时,这部分代码还可以作为后续低代码平台的场景提供者。在如下地方添加:

```
✓ MY-ADMIN

                 中の計却
                                .vscode > {} vue.code-snippets > ...
                                       lixinxiang, 4 days ago | 1 author (lixinxiang)
 🥀 Login.jsx
 🗸 🥰 .vscode
    {} vue.code-snippets
                                              "import { defineComponent } from 'vue';",
 > 🖷 docs
                                             "export default defineComponent({",
 > mock
                                              " setup() {",
 > node_modules
                                                   return () => <div>$1</div>;",
 > 🅫 public
   .editorconfig
   .gitignore
                                            "description": "Create <script> block"
   prettierrc ...
   index.html
                                            "prefix": "clg",
   J& jsconfig.json
                                            "body": ["console.log(\"[] $1:\",$1)"],
   package.json
                                            "description": "Create <script> block"
   vite.config.js
   yarn.lock
```

Vscode 还有很多小技巧,但本章的重点是结合项目的配置,一些项目标准的配置,如 Eslint、 Prettier 等不属于本章的内容,感兴趣的同学可以查阅相关文章进行了解。

2.3 Vite JSX 设置

由于笔者比较推荐使用 React,所以推荐使用 JSX 这种更为灵活的方式进行组件和业务开发。读者可以结合自身情况自行选择。

安装 plugin-vue-jsx:

```
Bash

1 yarn add -D @vitejs/plugin-vue-jsx@next
```

在 vite.config.js 中设置:

```
JavaScript

1 import vueJsx from "@vitejs/plugin-vue-jsx";
2 import { defineConfig } from "vite";
3
4 export default defineConfig({
5 plugins: [vueJsx()],
6 });
```

添加 JSX 之后可以直接 import 使用,无需注册:

```
JavaScript
    import { defineComponent } from "vue";
 2
   import ProLayout from "@ant-design-vue/pro-layout";
 3
 4
   export default defineComponent({
 5
      setup() {
       return () => (
 6
 7
          <ProLayout>
            <router-view />
 8
         </ProLayout>
 9
       );
10
      },
11
12 });
```

单文件的编码方式想必很多读者都非常熟悉了,而本文秉承拥抱变化的初衷,所以后续教程均使用 JSX 语法。

三、安装路由

中后台项目作为单页应用,路由是必不可少的。下面我们基于 Vue-Router 来进行简单的配置讲解。

第一步,新增 Vue-Router 依赖:

```
Bash

1 yarn add vue-router@next
```

第二步,新建 src/router/index.js ,并添加如下代码:

```
JavaScript
   import { createRouter, createWebHashHistory } from "vue-router";
 2
 3 import Home from "@/views/Home.vue";
 4 import Dashboard from "@/views/Dashboard.vue";
 5
 6 // 参考: https://next.router.vuejs.org/zh/
 7 const router = createRouter({
     history: createWebHashHistory(),
 8
      routes: [
 9
10
       {
11
          path: "/",
          name: "/",
12
13
       component: Home,
       },
14
        {
15
         path: "/dashboard",
16
         name: "dashboard",
17
18
         component: Dashboard,
19
       },
20
      ],
21 });
22
23 export default router;
```

第三步,在 src/main.js 中添加如下代码:

```
JavaScript

1 import { createApp } from "vue";
2 import App from "./App.vue";
3 import router from "./router";
4
5 const app = createApp(App);
6 app.use(router);
7 app.mount("#app");
```

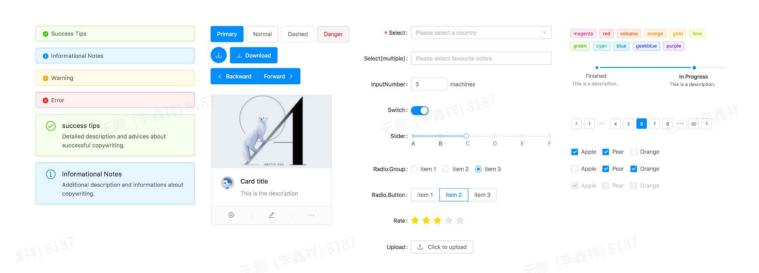
在 src/views 下创建 Home、Dashboard 组件文件,就可以开始联调看相应的效果了。

四、安装 UI 库

对于企业级中后台产品来说,目前社区比较推荐使用 Ant Design Vue 这一套。

4.1 Ant Design Vue 简介

Ant Design Vue 是使用 Vue 实现的遵循 Ant Design 设计规范的高质量 UI 组件库,用于开发和服务于企业级中后台产品。除了优秀的开发支持,还具备完整的 设计资源包。团队可以基于这些资源构建属于自己的产品设计体系。



4.2 如何使用

第一步,安装 ant-design-vue:

Bash 1 yarn add -D ant-design-vue@next

第二步, Antd 本身使用的是 less, 故推荐安装 less 作为样式的:

```
Bash

1 yarn add -D less
```

第三步, src/main.js 中引入:

```
JavaScript

1 // Vue 3.0 中 全局引入,也可以按需 https://2x.antdv.com/docs/vue/introduce-cn
2 import { createApp } from "vue";
3 import Antd from "ant-design-vue";
4 import "ant-design-vue/dist/antd.less";
5
6 const app = createApp(App);
7 app.use(Antd);
```

第四步,在 vite.config.js 开启 less:

```
JavaScript
 1 import { defineConfig } from "vite";
 2
 3 export default defineConfig({
 4
      css: {
        preprocessorOptions: {
 5
          less: {
 6
             javascriptEnabled: true,
 7
 8
          },
       },
 9
10
      },
11 });
```

第五步(可选),在 vite.config.js 设置每个组件自动引入 antd 变量:

首先新建 src/themes/var.less 文件:

1 @import "ant-design-vue/dist/antd.less";// 这里可以覆盖 antd 变量

这里的内容主要涉及主题定制,更多信息参考 antd 主题定制。另外,所有样式变量可以在 这里 找到。

注意:理论上,所有设计 Design Token 的值都应该从变量中获取,以便最大强度地进行主题定制。 避免后续,手动替换值得尴尬局面。

接着修改 vite.config.js 配置文件:

```
JavaScript
 1 import { defineConfig } from "vite";
 2 function resolve(dir) {
    return path.resolve(__dirname, ".", dir);
 4 }
 5
 6 export default defineConfig({
7
     css: {
       preprocessorOptions: {
 8
 9
         modifyVars: {
           hack: `true; @import "${resolve("./src/themes/var.less")}";`,
10
11
         },
     },
12
14 });
```

添加最终代码后的最终效果如图:



注意:引入主题修改配置会使得 vite 启动速度大幅度降低(教程为提升开发体验,暂时移除该配置)。如果对主题定制没有特别的要求,可以移除 less 变量修改相关配置,并将视觉呈现放置统一的自定义 UI 组件库中处理。

五、基础布局

对于中后台管理系统来说,通常会有基础布局,可以使用自研或者参考诸如 pro-layout 这类的开源库(涵盖了大量业务场景)。下面简单介绍与基础布局相关的知识点:

第一步,新建 | src/layouts/BasicLayout.jsx | (这里的后缀在 vite 中不建议忽略):

(李鑫祥) 810

一顿 (李鑫祥) 8187

```
JavaScript
    import { defineComponent } from "vue";
 2 import { useRouter } from "vue-router";
 3 import { getMenuData } from "@ant-design-vue/pro-layout";
 4 import ProLayout from "@ant-design-vue/pro-layout";
 5
 6 export default defineComponent({
 7
      setup() {
        const { getRoutes } = useRouter();
 8
 9
        const routes = getRoutes();
        const { menuData } = getMenuData(routes); // 通过路由配置获取菜单数据
10
11
12
        return () => (
13
          <ProLayout>
            <router-view />
14
          </ProLayout>
15
16
       );
         前 (李鑫祥) 8187
17
      },
18 });
```

注意: 为避免纯黏贴代码导致教程冗余,示例代码非完整可运行代码。

第二步,配置路由参数如下:

(李鑫祥) 810

无颜(

一顿 (李鑫祥) 8187

干颜 (李鑫符

```
JavaScript
    import { createRouter, createWebHashHistory } from "vue-router";
 2 import { BasicLayout } from "@/layouts";
 3
   const router = createRouter({
 4
 5
      routes: [
        {
 6
 7
          path: "/",
          name: "index",
 8
          meta: { title: "home" },
 9
          redirect: "/dashboard",
10
          component: BasicLayout,
11
12
          children: [
            {
13
              path: "/dashboard",
14
15
              name: "dashboard",
              meta: { title: "Dashboard", icon: "SmileOutlined" },
16
           // 动态引入路由文件,起到代码分割的作用(懒加载)
17
              component: () =>
18
                import(/* webpackChunkName: "dashboard" */
19
    "@/views/Dashboard.vue"),
            },
20
21
          ],
22
       },
23
      ],
24 });
25
```

大概效果如图:

26 export default router;

一顿 (李鑫祥) 8187

0187

(李鑫祥) 8181

无颜 (李鑫祥



这样,一个基础的左右布局的模板就完成了,后续只需要在如有里面简单关联即可,并且可以在 layouts 目录下新增更多的布局模板供路由使用。

六、路由权限

权限对于中后台系统来说也是必不可少的,路由的权限通常可以通过两种方式来实现:

- · 前端在本地配置好路由表,根据服务端返回的权限标识来动态将路由表添加到 Vue-router 配置中。
- · 通过服务端来返回对应的路由表, 动态添加带配置中。

考虑到前端分离,服务解耦,首选前端本地维护路由表的方式,并且这种方式维护成本最低、且最灵活。(服务端无需配置 Component 和 Icon 等信息)。

6.1 前端配置

根据是否需要权限控制,可以把路由分为两种:

· staticRoutes:静态路由,不受权限控制 · dynamicRoutes:动态路由,受权限控制

首先,我们要做的就是配置本地动态路由(dynamicRoutes)用于后续根据权限生成路由:

JavaScript 1 import { BasicLayout } from "@/layouts"; 2 3 /** 4 * 根据权限生成的路由 5 * @type { *[] } 6 */ 7 const dynamicRoutes = [8 path: "/", 9 name: "index", 10 meta: { title: "home" }, 11 12 redirect: "/dashboard", 13 component: BasicLayout, children: [14 15 path: "/dashboard", 16 name: "dashboard", 17 meta: { title: "Dashboard", icon: "SmileOutlined", permission:'' }, 18 component: () => 19 import(/* webpackChunkName: "dashboard" */ "@/views/Dashboard.vue"), 20 21 },], 22 23 }, 24], 25 26 export default dynamicRoutes

有了动态路由表,就可结合 Vue-router 4.0 的 动态路由 在应用程序运行的时候添加或删除路由。提供了下列 API 来实现路由动态配置(查看更多 API):

```
router.addRoute()
router.removeRoute()
router.hasRoute()
router.getRoutes()
```

另外,对于权限的控制需要实现在无论进入哪个路由,都可以根据权限生成路由表,这里需要用到 导航守卫,伪代码如下:

JavaScript import router from "./router"; 2 router.beforeEach(() => { 3 // 用全局数据中的角色来标识是否已经根据角色生成路由表 (如 vuex 中的 store) 4 if (!store.roles) { 5 // 发送请求获取 roles const { roles } = request("/role"); 7 8 // 根据角色生成路由表 9 const addRouters = generateRoutes(roles); 10 11 12 // 动态添加可访问路由表 router.addRoutes(addRouters); 13 14 } 15 });

最后来说一下 generateRoutes 生成路由的方式:

- · 根据 dynamicRoutes 中的每个路由 meta 信息中的权限标识与服务端请求的返回的标识进行过滤
- · 通常服务端返回标识的方式有两种,一种是方式:返回角色 roles,这时 meta 中的 permission 通常为数组。另一种方式是:直接返回唯一约定好的权限标识。这种比较暴力,但相对灵活。所以具体如何写 generateRoutes 的逻辑,视具体服务端逻辑

此外,为提升研发以及交互体验通常会用到这些库:

- · vuex 用于管理全局数据
- · axios 用于发送请求
- · NProgress 用于提升页面加载体验的进度条

注意:后续结合这些库的方式请参考源码,这里就不罗列代码了。

七、mock 服务

在前后端分离的项目,通常会接入 mock 服务。前端可以和服务端约定好数据格式,在本地 mock 对应的数据。另外 mock 数据有利于边界情况测试。接下来,我们结合 vite 接入 mock 服务(结合 登录场景)。

7.1 基础配置

接入 mock 服务需要用到下列依赖,读者可提前了解一下:

- · mockis 用于 mock 数据
- · vite-plugin-mock 用于启动 vite mock 服务
- · axios 用于发送请求
- · ahooks-vue 用于结合 Vue 维护请求数据

第一步,安装依赖:

```
Bash

1 # 安装 mock 数据生成库

2 yarn add -D mockjs

3

4 # 安装 vite mock 服务插件

5 yarn add -D vite-plugin-mock

6

7 # 安装 axios 请求

8 yarn add axios

9

10 # 安装 ahooks-vue 常用 vue hook

11 yarn add ahooks-vue
```

第二步,配置 vite.config.js:

```
JavaScript
 1 import { defineConfig } from "vite";
 2 import { viteMockServe } from "vite-plugin-mock";
 3
 4 // https://vitejs.dev/config/
 5 export default defineConfig({
     plugins: [
 6
 7
       viteMockServe({
          // default
 8
 9
         mockPath: "mock",
         localEnabled: true,
10
11
       }),
12 ],
13 });
```

CoffeeScript 1 export default [2 3 url: "/api/user/login", response: ({ body }) => { 4 5 console.log("[mock response]", body); 7 return { code: 0, 8 message: "ok", 9 data: null, 10 11 }; 12 }, 13 }, 14];

如果需要添加更灵活的数据可以参考mock 规则。下面结合登录场景讲解,如何使用 mock 服务。

7.2 接口服务

首先新建 helpers/request.js 用于项目输出统一的 axios 实例用于发送请求:

无颜 (李鑫祥) 810

- 短 (李鑫祥) 8187

李鑫代

-- 師 (李鑫祥) 8187

干颜 (李鑫祥

```
JavaScript
 1 import axios from "axios";
 2
 3 // 创建 axios 实例
 4 const request = axios.create({
 5 // API 请求的默认前缀
     baseURL: "/",
 6
     timeout: 6000, // 请求超时时间
 7
 8 });
 9
10 // 异常拦截处理器
11 const errorHandler = (error) => {
12
     if (error.response) {
     console.log("[axios] error.response:", error.response);
13
14
      }
     return Promise.reject(error);
15
16 };
17
18 // 请求拦截器
19 request.interceptors.request.use((config) => {
     console.log("[axios] config:", config);
20
    return config;
21
22 }, errorHandler);
23
24 // 响应拦截器
25 request.interceptors.response.use((response) => {
26
    console.log("[axios] response:", response);
     return response.data;
27
28 }, errorHandler);
29
```

创建 services/user.js 文件:

30 export default request;

祥) 8187

-- 5 (李鑫祥) 8187

干颜 (李鑫符

```
JavaScript
 1 import { request } from "@/helpers";
 2
 3 const userApi = {
 4 Login: "/api/user/login",
 5 };
 6
   export function login(data) {
 7
      return request({
 9
        url: userApi.Login,
        method: "post",
10
        data,
11
12
      });
13 }
```

添加登录页面(重点关注发送请求部分的逻辑,更多细节参看源码):

```
JavaScript
 1 import { defineComponent, reactive } from "vue";
 2 import { Button, Form, Input, Typography } from "ant-design-vue";
 3 import { useAxios } from "ahooks-vue";
 4
 5 import { login } from "@/services/user";
 6
 7
   export default defineComponent({
 8
      setup() {
        // 通过 useAxios 来建立请求机制
 9
        const { run, loading } = useAxios(login, { manual: true });
10
11
12
        const state = reactive({ username: "", password: "" });
13
14
        const handleFinish = async (values) => {
          console.log(values);
15
          const { code } = await run(values);
16
          if (code === 200) {
17
            router.replace("/");
18
19
          }
20
        };
21
22
        return () => (
23
          <Form
24
            model={state}
25
             onFinish={handleFinish}
```

```
onFinishFailed={handleFinishFailed}
26
27
28
            <Form.Item name="username">
              <Input vModel={[state.username, "value"]} placeholder="Username" />
29
            </Form.Item>
30
            <Form.Item name="password">
31
32
              <Input
                vModel={[state.password, "value"]}
33
34
                type="password"
35
                placeholder="Password"
              />
36
37
            </Form.Item>
            <Button
38
              htmlType="submit"
39
              block
40
              type="primary"
41
              disabled={state.username === "" || state.password === ""}
42
              loading={loading.value}
43
44
45
            Login
46
            </Button>
47
          </Form>
       );
48
49
      },
50 });
```

八、登录功能

我们先来看一下常规的登录场景:

场景一:

- · 访问登录页面,登录之后跳转首页,同时记录 token 且在后续的每个请求头上都带上 token 信息
- · 退出登录,清空 token 记录

场景二:

· 未登录的情况下,访问非登录页,重定向到登录页

根据这些场景可以知道需要做什么,接着来完成如下功能:

- · 登录页、其他页面的路由配置
- · mock 登录、用户信息等接口
- ·接口统一鉴权,无权限则重定向到登录页

- · 登录页、首页 UI 实现 (看源码)
- · 用户信息(如 token、name)、路由表数据全局维护(vuex)
- · 根据权限动态生成路由表

8.1 路由配置

我们在路由权限讲了如何配置动态路由,这里来简单配置下。首先登录页面和 404 这类属于不需要 权限的路由,我们约定为 staticRoutes。代码如下:

```
JavaScript
 1 import { UserLayout } from "@/layouts";
 2
 3 /**
 4 * 不需要权限的路由
 5 * @type { *[] }
 6
   */
 7 const staticRoutes = [
 8
        path: "/user",
 9
        component: UserLayout,
10
        redirect: "/user/login",
11
12
        hidden: true,
        children: [
13
           path: "login",
14
15
            name: "login",
16
            component: () =>
17
              import(/* webpackChunkName: "user" */ "@/views/user/Login.jsx"),
18
19
         },
        ],
20
      }, (李鑫洋)818
21
22
23
      {
24
        path: "/404",
        component: () =>
25
          import(/* webpackChunkName: "fail" */ "@/views/exception/404.jsx"),
26
27
      },
28 ];
29
30 export default staticRoutes;
```

```
JavaScript
 1 import { BasicLayout } from "@/layouts";
 2
 3 /**
 4 * 根据权限生成的路由
 5 * @type { *[] }
 6 */
 7 const dynamicRoutes = [
 8
        path: "/",
 9
        name: "index",
10
        meta: { title: "home" },
11
        redirect: "/dashboard",
12
13
        component: BasicLayout,
        children: [
14
15
            path: "/dashboard",
16
            name: "dashboard",
17
            meta: {
18
             title: "Dashboard",
19
             icon: "SmileOutlined",
20
21
            },
            component: () =>
22
              import(/* webpackChunkName: "dashboard" */ "@/views/Dashboard.vue"),
23
24
         },
25
       ],
26
      },
27
        path: "/:catchAll(.*)",
28
       redirect: "/404",
29
        hidden: true,
30
```

8.2 模拟数据

31

32]; 33

},

34 export default dynamicRoutes;

登录的时候,需要匹配账号和密码,并且密码需要加密。为了避免大量地贴代码,伪代码实现如下:

```
JavaScript
```

```
1 import md5 from "md5";
2
3 const fakeUserList = [
     id: "4291d7da9005377ec9aec4a71ea837f",
4
 5
       username: "admin",
7
       password: md5("admin"),
       token: "fakeToken1",
8
      name: "admin",
9
                                                      无额 (李鑫祥) 8187
       avatar: "https://q1.qlogo.cn/g?b=qq&nk=339449197&s=640",
10
       permissions: ["admin"],
11
12
    },
13 ];
14
   export default [
15
16
17
       url: "/api/user/login",
       method: "post",
18
       response: ({ body }) => {
19
        // body 请求体
20
        const { username, password } = body;
21
22
     return "你需要 mock 的任何数据,这里 mock 通常返回的是从 fakeUserList 匹配的用
23
   户数据";
24
   },
25
    },
26 ];
```

8.3 请求接口权限校验

得益于 axios 的请求拦截器功,我们可以很轻松的实现接口的拦截并做响应的处理。

一顿 (李鑫祥) 8187

```
JavaScript
 1 import axios from "axios";
 2 import storage from "./storage";
 3 import { ACCESS_TOKEN } from "@/store/mutation-types";
 4
 5 // 创建 axios 实例
 6 const request = axios.create({
     // API 请求的默认前缀
 7
      baseURL: "/",
 8
      timeout: 6000, // 请求超时时间
 9
10 });
11
12
   // 异常拦截处理器
13 const errorHandler = (error) => {
      if (error.response) {
14
        console.log("[axios] error.response:", error.response);
15
      }
16
      return Promise.reject(error);
   };
18
19
   // 请求拦截器
20
   request.interceptors.request.use((config) => {
21
      const token = storage.getItem(ACCESS_TOKEN);
22
      if (token) {
23
24
       // 如果 localstorage 中存有 token,则在请求中带上
        config.headers["authorization"] = token;
25
      }
26
      return config;
27
   }, errorHandler);
28
29
   // 响应拦截器
30
   request.interceptors.response.use((response) => {
31
32
     return response.data;
   }, errorHandler);
33
34
   export default request;
35
```

8.4 用户信息全局维护

在具体实现登录功能之前,我们可以先来设计一下登录相关的全局数据维护,下面使用 vuex 来实现用户信息和路由权限表的 store:

入口文件:

```
JavaScript
    import Vuex from "vuex";
 2
 3 import permission from "./modules/permission"; // 权限表
 4 import user from "./modules/user"; // 用户信息
 5 import getters from "./getters";
 6
 7
   const store = new Vuex.Store({
      modules: {
 8
        permission,
 9
        user,
10
      },
11
      getters, // 相当于 vue 的 computed, 方便取值
12
13
   });
14
15 export default store;
```

用户信息:

```
JavaScript
 1 import storage from "@/helpers/storage";
 2 import { login, logout, getInfo } from "@/services";
    import { ACCESS_TOKEN } from "@/store/mutation-types";
 3
 4
 5
    const user = {
      state: { _______8
 6
         token: "",
 7
        name: "",
 8
 9
         avatar: "",
        permissions: [],
10
        info: {},
11
12
      },
13
14
      mutations: {
         SET_TOKEN: (state, token) => {
15
           state.token = token;
16
         },
17
         SET_NAME: (state, { name }) => {
18
19
          state.name = name;
20
        },
21
         SET_AVATAR: (state, avatar) => {
22
          state.avatar = avatar;
```

```
23
       },
       SET_PERMISSIONS: (state, permissions) => {
24
25
         state.permissions = permissions;
26
       },
       SET_INFO: (state, info) => {
27
28
        state.info = info;
      },
29
     },
30
31
     actions: {
32
       // 登录
33
       Login({ commit }, userInfo) {
34
         return new Promise((resolve, reject) => {
35
36
          // 1、请求登录接口 login
          // 2、commit("SET_TOKEN", data.token);
37
38
        });
       },
39
40
41
       // 获取用户信息
      GetInfo({ commit }) {
42
         return new Promise((resolve, reject) => {
43
          // 1、请求获取用户信息 getInfo
44
          // 2、commit("SET_PERMISSIONS", data.permissions);
45
          // commit("SET_INFO", data);
46
         });
47
48
       },
49
       // 登出
50
51
       Logout({ commit, state }) {
         return new Promise((resolve) => {
52
          // 1、请求退出接口 logout
53
          // 2. commit("SET_TOKEN", "");
54
           // commit("SET PERMISSIONS", []);
55
        });
56
57
      },
58
    },
59 };
61 export default user;
```

路由权限表:

无额 (李鑫祥) 8187

```
JavaScript
 1 import { dynamicRoutes, staticRoutes } from "@/router/routes";
 2
 3 function hasPermission(permission, route) {
                                         无颜 (李鑫祥) 8187
 4 // 判断是否有权限
 5 }
 6
 7 function filterAsyncRouter(routes, permissions) {
     // 过滤有权限的路由
 9
   }
10
   const permission = {
11
12
      state: {
        routers: staticRoutes,
13
14
        addRouters: [],
15
      },
      mutations: {
16
17
        SET ROUTERS: (state, routers) => {
         state.addRouters = routers;
18
          state.routers = staticRoutes.concat(routers);
19
       },
20
21
      },
      actions: {
22
23
        GenerateRoutes({ commit }, data) {
24
          return new Promise((resolve) => {
            const { permissions } = data;
25
            const accessedRouters = filterAsyncRouter(dynamicRoutes, permissions);
26
            commit("SET_ROUTERS", accessedRouters);
27
            resolve(accessedRouters);
28
29
         });
        },
30
31
      },
32 };
33
34 export default permission;
```

8.5 路由动态生成

无论用户是否登录,访问任何一个路由做的其中一件事就是请求用户信息,并且在应用的整个生命周期都可以方便的获取。

也就是需要有一个地方统一校验当前用户的权限和可访问的路由表,这里需要用到路由拦截器,大概的实现逻辑如下:

```
JavaScript
 1 import router from "@/router";
 2 import store from "@/store";
 3 import storage from "./helpers/storage";
 4 import { ACCESS_TOKEN } from "./store/mutation-types";
 5
 6 const loginRoutePath = "/user/login";
 7 const whiteList = ["login", "register", "registerResult"];
 8
 9 // 路由守卫: https://router.vuejs.org/zh/guide/advanced/navigation-guards.html
   router.beforeEach((to, from, next) => {
10
      const token = storage.getItem(ACCESS_TOKEN);
11
12
      if (token) {
13
14
       if (to.path === loginRoutePath) {
       next({ path: "/" });
15
16
         return;
17
        }
18
19
        if (store.getters.permissions.length === 0) {
20
         // 1. store.dispatch("GetInfo")
          // 2、store.dispatch("GenerateRoute")
21
22
          // 3. addRouters.forEach((route) => router.addRoute(route));
        } else {
23
24
          next();
        }
25
      } else {
26
       if (whiteList.includes(to.name)) {
27
       // 在免登录名单,直接进入
28
29
         next();
        } else {
30
          next({ path: loginRoutePath, query: { redirect: to.fullPath } });
31
32
        }
33
     }
34 });
```

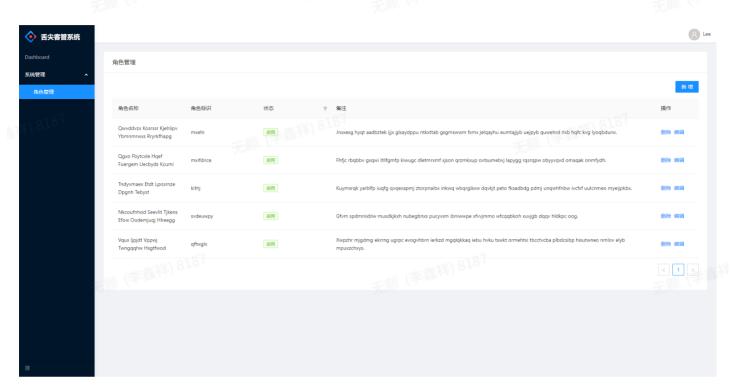
九、CURD

中后台常见的场景业务通常包括增删改查,即:

· 列表的展示

- ·新增列表项
- 编辑列表项
- ·删除列表项

下面带着大家简单实现一个基本角色管理,效果如图:



9.1 项目配置

常规的业务开发需要经过这几步:

- ·路由配置
- · 接口配置
- · mock 数据

路由配置:

目录组织以及路由配置如下图:

```
src > router > routes > JS dynamicRoutes.js > [@] dynamicRoutes > & children
✓ MY-ADMIN
 > 📹 .template
                                             path: "/dashboard",
name: "dashboard",
 > 💋 .vscode
 > 💼 docs
 JS _util.js
    JS user.js
 > node_modules
 > 🗱 public
                                             path: "/system",
 ∨ 📹 src
  > 🔞 assets
  > 🥦 helpers
                                               title: "系统管理",
  > 👨 layouts
  v 📹 router
                                             component: RouterView,
   JS dynamicRoutes.js M
                                                 path: "/system/role",
name: "role",
      JS index.js
      JS staticRoutes.js
     JS index.js
                                                   title: "角色管理",
  JS index.js
     JS user.js
  > ii store
  > plant themes
  },
   > = exception
   U
                                        redirect: "/404",
       index.jsx
   > user
     V Dashboard.vue
                                      无颜 (李鑫祥) 816
                                                                                           无额 (李鑫祥) 818
```

接口配置:

接口满足 RESTful 规范,配置如下:

干颜 (李鑫祥) 8187

- 師 (李鑫祥) 818

一顿 (李鑫祥) 8187

· (李鑫祥) 818 (

干颜 (李鑫符

```
JavaScript
   import { request } from "@/helpers";
 2
 3 const userApi = {
 4 systemRole: "/api/system/role",
 5
   };
 6
 7
   export function getRole(data) {
      return request({
 9
        url: userApi.systemRole,
        method: "get",
10
        data,
11
12
     });
13 }
14
   export function addRole(data) {
15
      return request({
16
17
        url: userApi.systemRole,
        method: "post",
18
19
        data,
      });
20
   }
21
22
23 export function editRole(data) {
24
     return request({
        url: userApi.systemRole,
25
        method: "put",
26
        data,
27
28
      });
29 }
30
31 export function removeRole(data) {
32
      return request({
        url: userApi.systemRole,
33
        method: "delete",
34
        data,
35
36
      });
37 }
```

请求 URL 固定,通过方法来区分功能。如 GET、POST、PUT、DELETE 分别表示:查、新增、修改、删除。

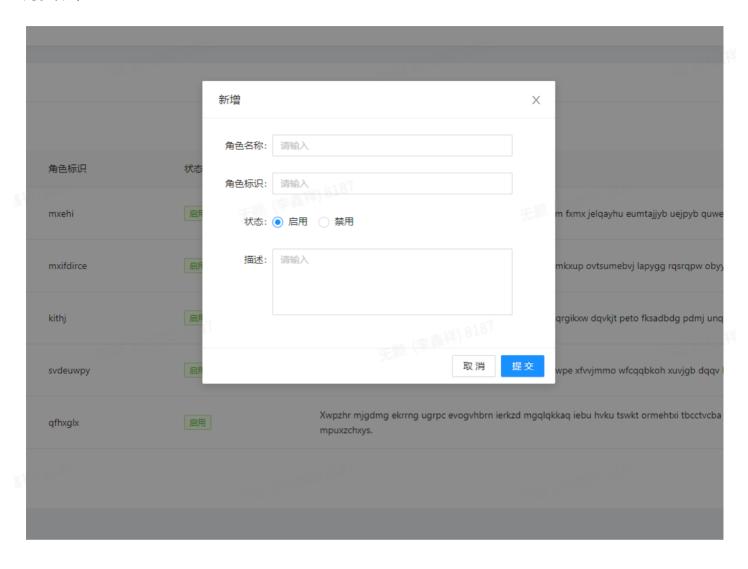
Mock 数据:

这里会用到 mockis 用于模拟数据。

```
JavaScript
 1 import Mock from "mockjs";
 2 import { success, checkToken } from "./_util";
 3
   // http://mockjs.com/examples.html#Array
 4
 5 let { data } = Mock.mock({
      "data|5-10": [
 6
 7
        {
          id: "@guid",
          name: "@title",
 9
10
          code: "@word",
          status: 1,
11
          desc: "@sentence",
12
13
       },
         面 (李鑫祥) 8187
14
      ],
15
   });
16
17
    export default [
18
19
        url: "/api/system/role",
        method: "post", // geh. put. delete
20
        response: (request) => {
21
22
          const { body } = request;
23
          // 新增
24
25
          data.push({ id: Mock.mock("@guid"), ...body });
26
         // 修改
27
          data = data.map((item) => {
28
            if (item.id === body.id) {
29
30
              return { ...item, ...body };
            }
31
32
            return item;
33
          });
34
35
          // 删除
          data = data.filter((item) => {
36
            return item.id !== body.id;
37
          });
38
      }, 微 (李慈祥) 8181
39
40
   ];
41
```

9.2 业务开发

对于页面的布局,这里就不进行详细的说明。主要说一下,这里新增、编辑是采用弹窗的形式来实现 的。如下:



表单主要用于数据收集,所有的业务逻辑在入口文件处理,核心逻辑如下(为方便理解,仅保留关键逻辑):

```
TypeScript
   import { defineComponent, onMounted, reactive, ref } from "vue";
 2 import { useAxios } from "ahooks-vue";
 3 import { addRole, editRole, getRole, removeRole } from "@/services/role";
   import FormModal from "./FormModal";
 5
   export default defineComponent({
 6
 7
      setup() {
        // 这里采用 hook 的方式来维护请求状态,参
 8
    见: https://dewfall123.github.io/ahooks-vue/zh/use-axios/
        const { loading, data, run } = useAxios(getRole, { manual: true });
 9
        const { run: add } = useAxios(addRole, { manual: true });
10
     const { run: edit } = useAxios(editRole, { manual: true });
```

```
const { run: remove } = useAxios(removeRole, { manual: true });
12
13
       const visible = ref(false);
14
15
       // 通过 type 来区分是新增还是修改,这里可以用魔法字符串或枚举代替,具体看团队规范
16
17
        const type = ref("edit");
18
       const handleAdd = () => {
19
         visible.value = true;
20
         type.value = "add";
21
22
       };
23
       const handleEdit = (record) => {
24
25
         visible.value = true;
         type.value = "edit";
26
27
       };
28
       const handleRemove = async (record) => {
29
         await remove({ id: record.id });
30
       await run();
31
32
       };
33
34
       const handleCancel = () => {
         visible.value = false;
35
       };
36
37
       // 表单提交逻辑
38
        const handel0k = async (values) => {
39
         if (type.value === "add") {
40
           await add(values);
41
42
          }
43
         if (type.value === "edit") {
44
           await edit({ id: currentData.id, ...values });
45
         }
46
47
          run();
48
49
         visible.value = false;
50
51
       };
52
       onMounted(() => {
53
          run();
54
        });
55
56
        return () => (
57
          <Card title="角色管理">
58
```

```
<Space direction="vertical" size="middle" style="width:100%">
59
              <Row type="flex" justify="end">
60
                <Col>
61
                  <Button type="primary" onClick={handleAdd}>
62
                    新增
63
64
                  </Button>
                </Col>
65
              </Row>
66
              <Table
67
                rowKey="id"
68
69
                columns={columns}
                dataSource={data.value}
70
                loading={loading.value}
71
              />
72
              <FormModal
73
                title={type.value === "add" ? "新增" : "编辑"}
74
                visible={visible.value}
75
                on0k={handel0k}
76
                onCancel={handleCancel}
77
           />
78
79
            </Space>
          </Card>
80
        );
81
82
      },
   });
83
```

这里仅仅简单的 CURD 示例,更多细节可以查看 源码。

李鑫祥)⁸¹⁰

一顿 (李鑫祥) 8187

干颜 (李鑫符