从前端角度谈 RBAC 权限系统设计

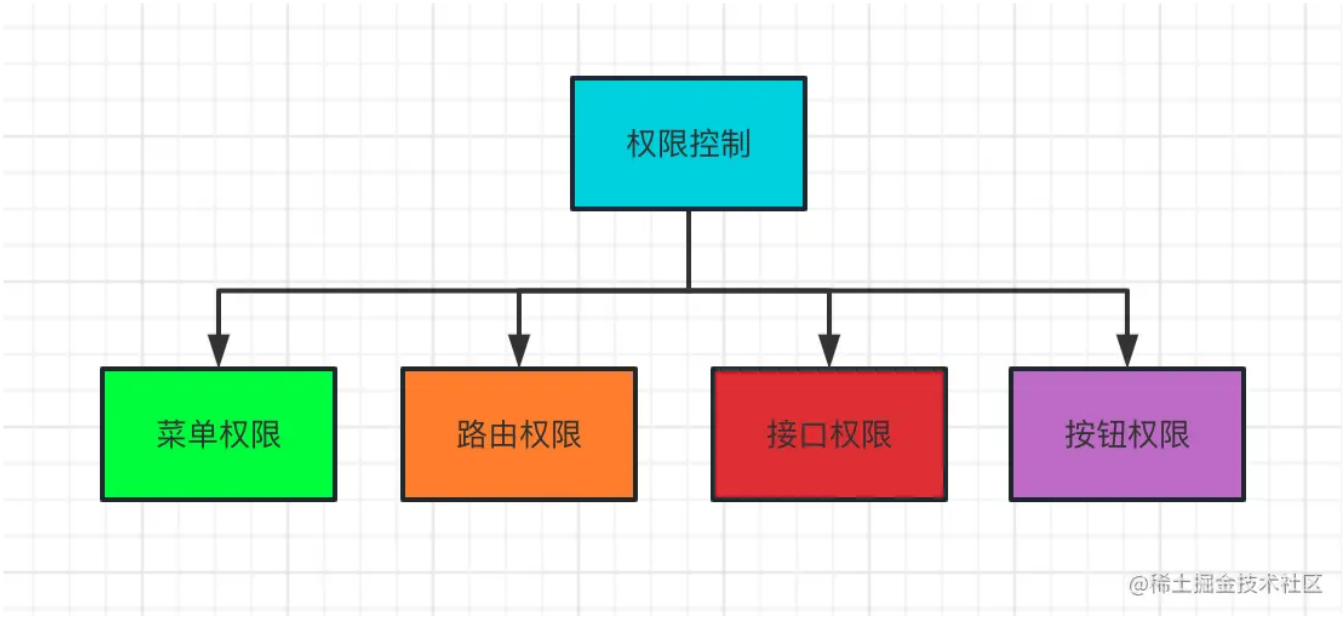
方案设计

在实际业务中，权限系统的设计其实可以做到很复杂，但是我们为了简单起见只保留一些最基本且核心的模块：

- 登录模块：权限平台一般需要靠登录获取用户身份，并通过凭证去请求接口，包括注册功能。
- 系统管理模块：包括用户管理、角色管理、菜单管理（如果菜单是前端控制则可以省略）等功能，是权限系统中的核心部分。

而权限控制其实分为了两部分：**数据权限** 和 **功能权限** 控制。主要从四个方面入手：

- 接口权限：权限系统的最后一道保护屏障，每个权限接口都需要 `token` 。
- 路由权限：通过拦截器阻止越权页面访问，防止用户直接通过 URL 进入页面。
- 菜单权限：通过控制菜单权限来给不同角色的用户展示不同菜单。
- 按钮权限：控制数据的操作，没有权限时隐藏或者按钮置灰禁用。



可以看到权限无非就是对用户的操作和视图控制，一般需要前后端同学共同去配合去做，后端小伙伴更像是守门员，守住最后一道防线，而前端小伙伴则负责在球进门之前给他阻挡掉，从而减少守门员的压力。

菜单路由权限方案

有一个很重要的问题那就是，菜单和路由要怎么管理呢？有的人说放到前端管理，有的人说放到后端管理，我们看看两种常见菜单路由方案（下面以 `Vue` 生态为例）。

方案一：前端管理菜单路由

这种方案主要以 vue-element-admin 这个开源项目为代表的方案：

- 设置两种路由：**公共路由** 和 **权限路由**。同时在路由表的 `meta` 字段中绑定菜单相关信息。
- `vue` 实例化时前创建静态路由，用户登录后根据角色信息筛选出动态路由.
- 在 `vue` 实例化后通过使用 `vue-router` 中 `addRoutes` 将动态路由添加到路由表。
- 根据角色信息过滤路由表生成菜单。

这种方式的优点是不依赖后端，可单独管理，也不需要实现菜单管理这个单独的功能。但也有些缺点：

- 菜单路由耦合，需要在路由里面配置菜单信息，而且路由不一定是菜单，却多了菜单的配置。
- 如果需要改菜单的配置比如图标、文案、排序需要前端编码并重新编译部署。

具体代码实现和细节可以看花裤衩大佬的 这篇文章 。

方案二：前后端配合管理路由菜单

方案一每次都需要前端改动代码修改菜单配置，于是针对这个点可以把菜单路由交给后端管理，比较成熟的代表是 若依后台管理系统 ，它的整体方案和 `vue-element-admin` 差不多，区别在于菜单和路由是前后端一起管理的。

router/index.js

```js
import Vue from 'vue'
import Router from 'vue-router'

Vue.use(Router)

/* Layout */
import Layout from '@/layout'
```

```
/**
 * Note: 路由配置项
 *
 * hidden: true                     // 当设置 true 的时候该路由不会再侧边栏出现 如401，lo
 * alwaysShow: true                 // 当你一个路由下面的 children 声明的路由大于1个时，自
 *                                  // 只有一个时，会将那个子路由当做根路由显示在侧边栏--如
 *                                  // 若你想不管路由下面的 children 声明的个数都显示你的
 *                                  // 你可以设置 alwaysShow: true，这样它就会忽略之前定
 * redirect: noRedirect             // 当设置 noRedirect 的时候该路由在面包屑导航中不可被
 * name:'router-name'               // 设定路由的名字，一定要填写不然使用<keep-alive>时会
 * query: '{"id": 1, "name": "ry"}' // 访问路由的默认传递参数
 * roles: ['admin', 'common']       // 访问路由的角色权限
 * permissions: ['a:a:a', 'b:b:b']  // 访问路由的菜单权限
 * meta : {
    noCache: true                   // 如果设置为true，则不会被 <keep-alive> 缓存(默认
    title: 'title'                  // 设置该路由在侧边栏和面包屑中展示的名字
    icon: 'svg-name'                // 设置该路由的图标，对应路径src/assets/icons/svg
    breadcrumb: false               // 如果设置为false，则不会在breadcrumb面包屑中显示
    activeMenu: '/system/user'      // 当路由设置了该属性，则会高亮相对应的侧边栏。
  }
 */

// 公共路由
export const constantRoutes = [
  {
    path: '/login',
    component: () => import('@/views/login'),
    hidden: true
  },
  {
    path: '/register',
    component: () => import('@/views/register'),
    hidden: true
  },
  {
    path: '/404',
    component: () => import('@/views/error/404'),
    hidden: true
  },
  {
    path: '/401',
    component: () => import('@/views/error/401'),
    hidden: true
  }
]

// 动态路由，基于用户权限动态去加载
export const dynamicRoutes = [
  {
```

```
    path: '/system/user-auth',
    component: Layout,
    hidden: true,
    permissions: ['system:user:edit'],
    children: [
      {
        path: 'role/:userId(\\d+)',
        component: () => import('@/views/system/user/authRole'),
        name: 'AuthRole',
        meta: { title: '分配角色', activeMenu: '/system/user' }
      }
    ]
  },
]

// 防止连续点击多次路由报错
let routerPush = Router.prototype.push;
Router.prototype.push = function push(location) {
  return routerPush.call(this, location).catch(err => err)
}

export default new Router({
  mode: 'history', // 去掉url中的#
  scrollBehavior: () => ({ y: 0 }),
  routes: constantRoutes
})
```

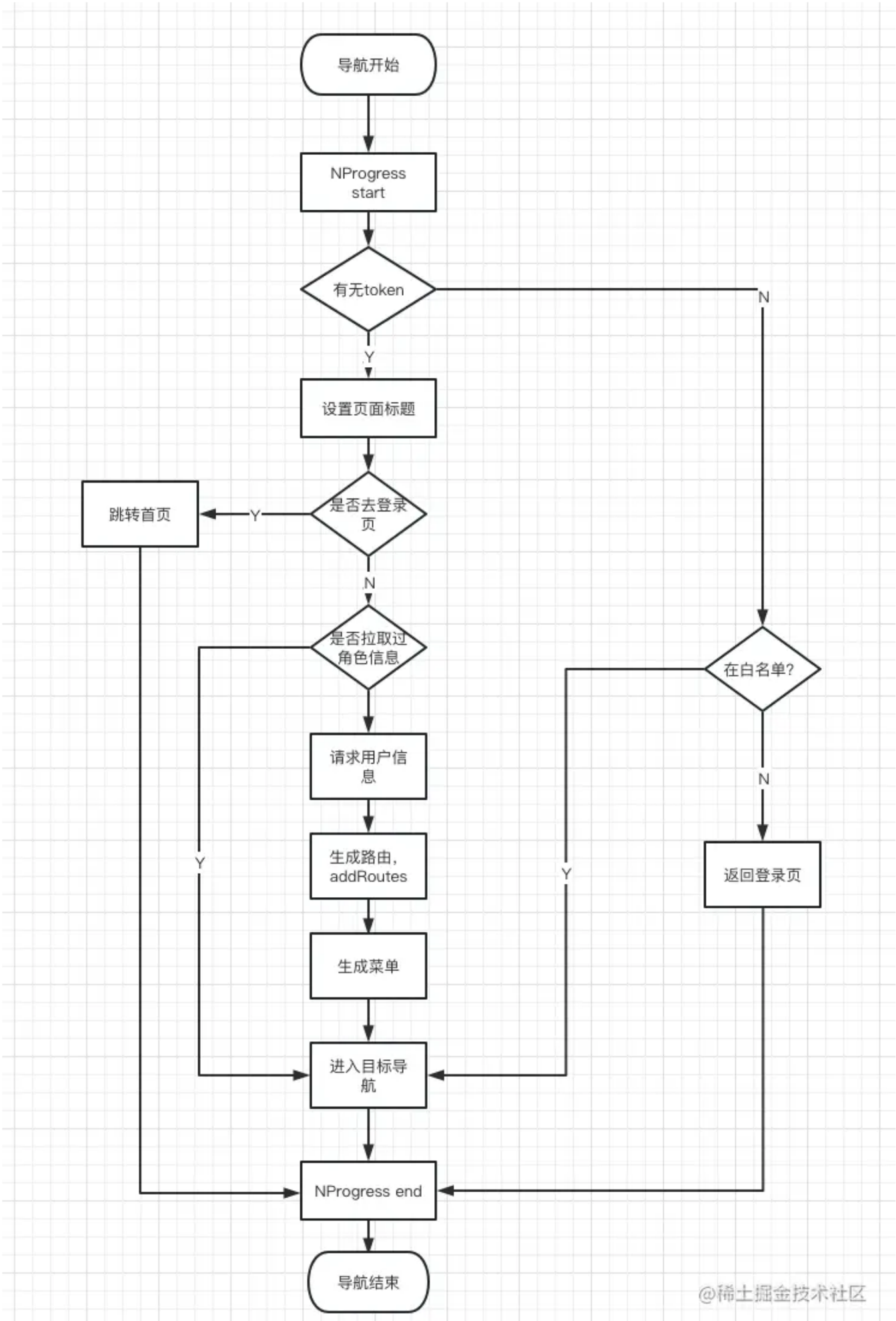
核心拦截器实现和 `vue-element-admin` 差不多：

permission.js

```
js 复制代码
import router from './router'
import store from './store'
import { Message } from 'element-ui'
import NProgress from 'nprogress'
import 'nprogress/nprogress.css'
import { getToken } from '@/utils/auth'
import { isRelogin } from '@/utils/request'

NProgress.configure({ showSpinner: false })

const whiteList = ['/login', '/auth-redirect', '/bind', '/register']

router.beforeEach((to, from, next) => {
  NProgress.start()
  if (getToken()) {
```

```
    to.meta.title && store.dispatch('settings/setTitle', to.meta.title)
    /* has token*/
    if (to.path === '/login') {
      next({ path: '/' })
      NProgress.done()
    } else {
      if (store.getters.roles.length === 0) {
        isRelogin.show = true
        // 判断当前用户是否已拉取完user_info信息
        store.dispatch('GetInfo').then(() => {
          isRelogin.show = false
          store.dispatch('GenerateRoutes').then(accessRoutes => {
            // 根据roles权限生成可访问的路由表
            router.addRoutes(accessRoutes) // 动态添加可访问路由表
            next({ ...to, replace: true }) // hack方法 确保addRoutes已完成
          })
        }).catch(err => {
            store.dispatch('LogOut').then(() => {
              Message.error(err)
              next({ path: '/' })
            })
          })
      } else {
        next()
      }
    }
  } else {
    // 没有token
    if (whiteList.indexOf(to.path) !== -1) {
      // 在免登录白名单，直接进入
      next()
    } else {
      next(`/login?redirect=${to.fullPath}`) // 否则全部重定向到登录页
      NProgress.done()
    }
  }
})

router.afterEach(() => {
  NProgress.done()
})
```

为了方便大家理解这个逻辑给大家画个图：


导航开始
NProgress start
有无token
N
Y
设置页面标题
是否去登录页
Y
跳转首页
N
是否拉取过角色信息
在白名单?
请求用户信息
N
Y
生成路由，addRoutes
Y
返回登录页
生成菜单
进入目标导航
NProgress end
导航结束
@稀土掘金技术社区

用户登录后会请求两个接口数据：

- 用户信息，包括当前用户角色信息、权限信息，核心结构如下：

```json
{
    "msg": "操作成功",
    "code": 200,
    "permissions": [
        "system:user:resetPwd",
        "system:post:list",
        "monitor:operlog:export",
        "monitor:druid:list",
    ],
    "roles": [
        "common"
    ],
    "user": {}
}
```

- 菜单路由信息，当前用户拥有的路由，核心结构如下：

```json
{
  "msg": "操作成功",
  "code": 200,
  "data": [
    {
      "name": "System",
      "path": "/system",
      "hidden": false,
      "redirect": "noRedirect",
      "component": "Layout",
      "alwaysShow": true,
      "meta": {
        "title": "系统管理",
        "icon": "system",
        "noCache": false,
        "link": null
      },
      "children": [
        {
          "name": "Log",
          "path": "log",
          "hidden": false,
          "redirect": "noRedirect",
          "component": "ParentView",
          "alwaysShow": true,
```

```
          "meta": {
            "title": "日志管理",
            "icon": "log",
            "noCache": false,
            "link": null
          },
          "children": [
            {
              "name": "Operlog",
              "path": "operlog",
              "hidden": false,
              "component": "monitor/operlog/index",
              "meta": {
                "title": "操作日志",
                "icon": "form",
                "noCache": false,
                "link": null
              }
            }
          ]
        }
      ]
    }
  ]
}
```

有了这些数据前端需要再对数据做一层数据，把路由表给弄出来，我们都知道路由组件在前端一般都是这样的：

js 复制代码

```
{
    name: "login",
    path: "/login",
    component: () => import("@/views/Login.vue")
}
```

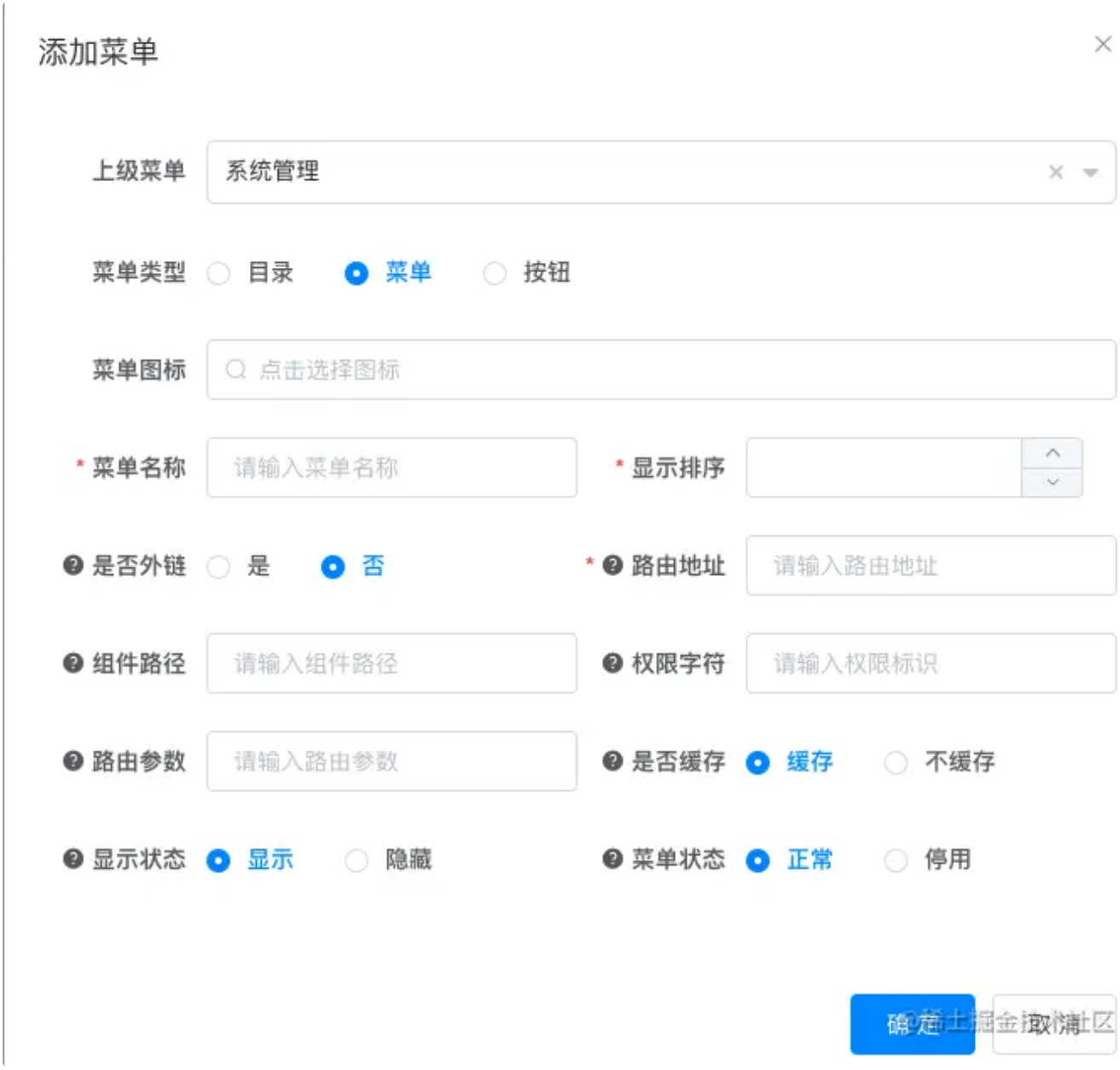
后端是不能直接返回这样的路由的，因为前端代码都是编译后的，已经不认识 `@/views/Login.vue` 了，所以需要前端进行处理：

js 复制代码

```
export const loadView = (view) => {
  if (process.env.NODE_ENV === 'development') {
    return (resolve) => require([`@/views/${view}`], resolve)
  } else {
    // 使用 import 实现生产环境的路由懒加载
```

```
    return () => import(`@/views/${view}`)
  }
}
```

篇幅有限，大家有兴趣可以直接看若依的源码实现：菜单处理逻辑 传送门 。下图就是若依的效果截图：



方案二的优点很明显那就是可以不用部署的方式去修改菜单信息，非常方便，但是也有问题：

- 菜单路由还是耦合
- 需要单独开发一个菜单管理功能，前后端的工作量会多出一块
- 不能随便乱改菜单数据，否则影响整个系统

当然这两个方案可能都不适用，比如还可能存在菜单必须全部展示的情况，不能用的要置灰提示（提示用户充钱才可以用），大家还有其他什么方案可以在评论区进行讨论的哦！只能说不管是软件开发还是方案选型都没有银弹这一说，适合业务的才是好的！

按钮权限方案

按钮权限一般在前端实现，按钮最终下发的操作是对数据的操作所以本质还是接口权限，后端需要兜底。而前端的场景无非就两种：

- 无权限时按钮置灰禁用
- 无权限时按钮直接隐藏

按钮权限稍微麻烦的点在于根据什么去判断权限，角色标识符还是权限点？

这点我在上篇文章 # 浅谈 RBAC 权限模型 有提到过基于资源的权限控制，所以基于权限点去做控制比较好，若依也是这样判断的，但是我也要补充下根据权限点判断的几种情况：

- 按钮操作依赖多个权限点时：满足其一或者全部才可以操作，否则就隐藏或者禁用置灰。
- 按钮操作依赖单个权限点时：满足单个权限点才可以操作，否则就隐藏或者禁用置灰。

基于上述条件我们可以设计一个权限指令 `v-auth` ：

场景一：传入单个权限点：

```html
<button v-auth="'system:user:add'">test</button>
```

场景二：传入多个权限点：

```html
<!-- 必须满足全部权限点 -->
<button v-auth="['system:user:add', 'system:user:edit']">test</button>

<!-- 满足其中一个权限点 -->
<button v-auth.oneOf="['system:user:add', 'system:user:edit']">test</button>
```

指令的实现思路就是判断传入的值与全局的权限点进行对比，满足条件后对 DOM 进行操作（移除、置灰...），下面是若依的实现：

```js
import store from '@/store'

export default {
  inserted(el, binding, vnode) {
    const { value } = binding
    const all_permission = "*:*:*";
    const permissions = store.getters && store.getters.permissions

    if (value && value instanceof Array && value.length > 0) {
      const permissionFlag = value

      const hasPermissions = permissions.some(permission => {
        return all_permission === permission || permissionFlag.includes(permission)
      })

      if (!hasPermissions) {
        el.parentNode && el.parentNode.removeChild(el)
      }
    } else {
      throw new Error(`请设置操作权限标签值`)
    }
  }
}
```

要实现 `oneOf` 的功能只需要对修饰符判断即可，这里就不额外说了，大家有兴趣可以自己实现。

总结

权限系统设计是一个很大的话题，本文提到的只是九牛一毛，比如更为重要的数据库设计、接口设计、系统流程设计都没有提及到。作为前端开发这个角色，本文的一些方案其实能解决很多问题了，但是要深入理解权限系统本身，还是需要去多看一些优秀的开源项目实现的，这样也能培养自己系统的设计能力。