项目主体页面布局设计

12.1 使用 Vue UI 创建项目

创建 Vue 前端工程的另外一种方便有效的方法是使用 Vue UI 图形化可视管理的 Vue 项目管理器，可以运行项目、打包项目，检查等操作。确定已安装如下前置环境：node.js、Vue-CLI 和 WebStorm。使用 Vue UI 创建项目步骤如下：

1. 安装最新的 Vue CLI

```
npm install -g @vue/cli
```

2. 安装成功后，可用 vue -V 命令查看已安装的 Vue CLI 版本

例如，本书安装的 Vue CLI 版本为@vue/cli 4.5.10。

3. 开启 vue ui 服务

使用快捷键 Win+R 打开 cmd 并输入以下命令：`vue ui`，即可在浏览器中打开 Vue 项目管理器，如图所示。

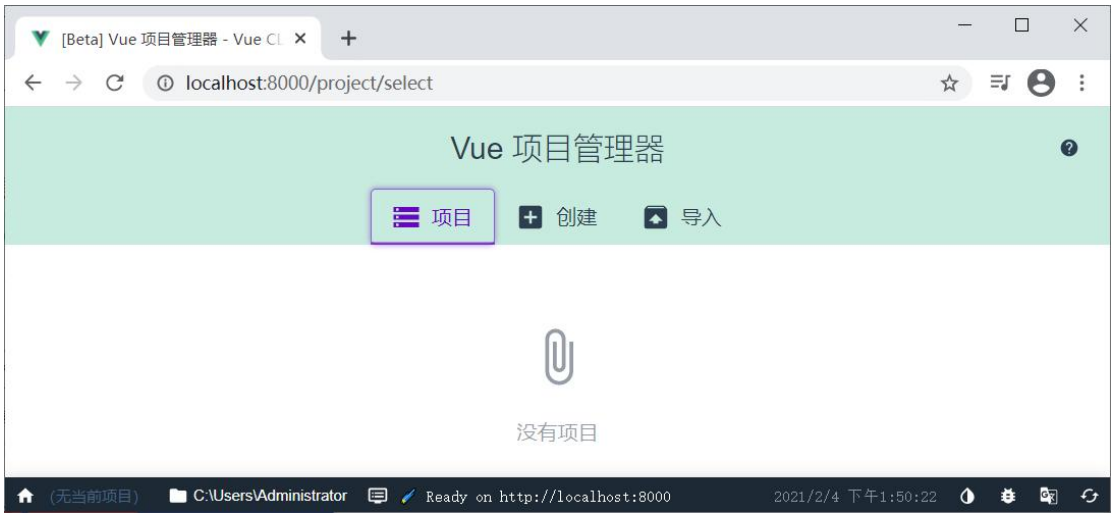

图 12-1 Vue 项目管理器

4. 创建项目

单击“创建”按钮，选择要创建的项目目录，点击“在此创建新项目”按钮，输入项目名称“Vue_hyit”，包管理器选择“npm”，Git 命令填写 init project，点击下一步，如图所示。

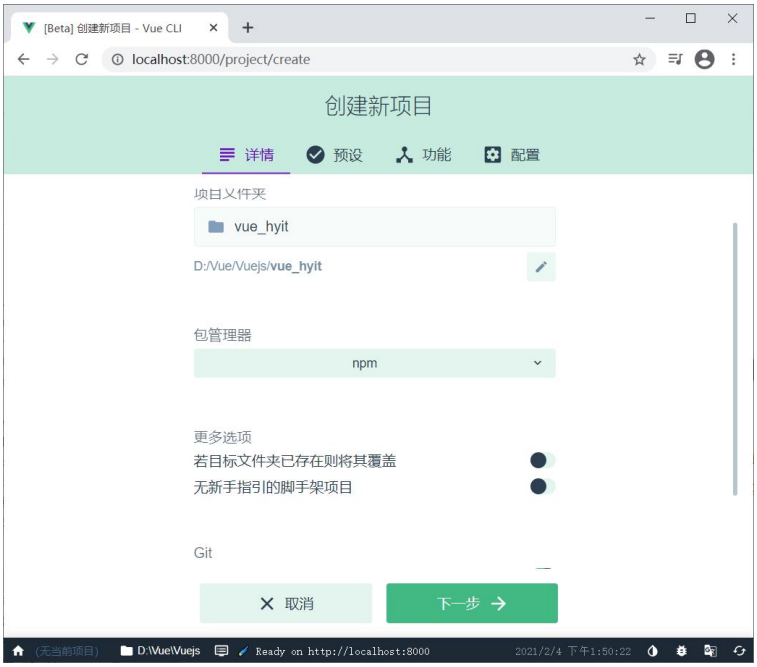

图 12-2 创建项目详情页面

在“预设”面板中选择“手动配置项目”，之后进入“功能”面板，选中其中项目必须的四项：Babel、Router 和使用配置文件，若在项目开发过程之后需要安装其他项，也可以通过安装插件方法增加。在“配置”面板中的设置为默认选项。

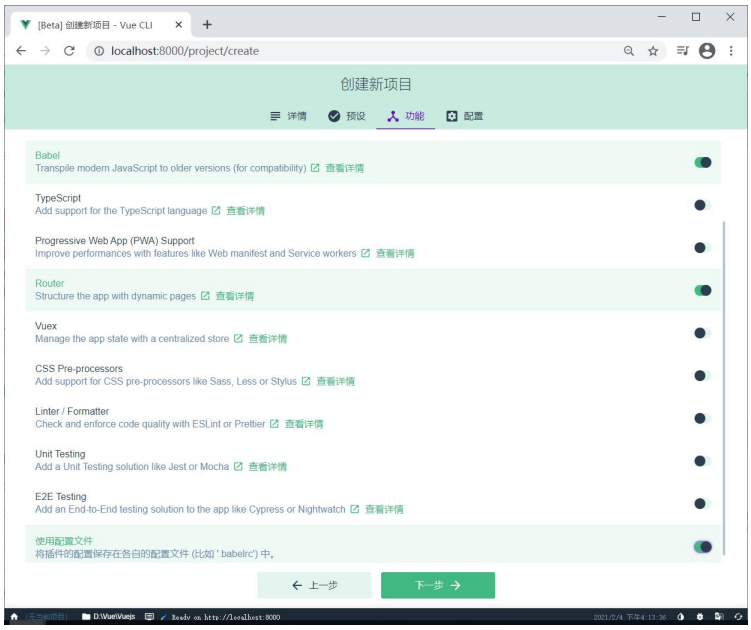

图 12-3 配置功能面板

5. 添加 element-ui 插件

项目创建成功之后，选中左侧导航栏的“插件”选项卡，单击右上角“添加插件”，如图所示，选择“vue-cli-plugin-element”插件，单击右下角“安装 vue-cli-plugin-element”按钮。

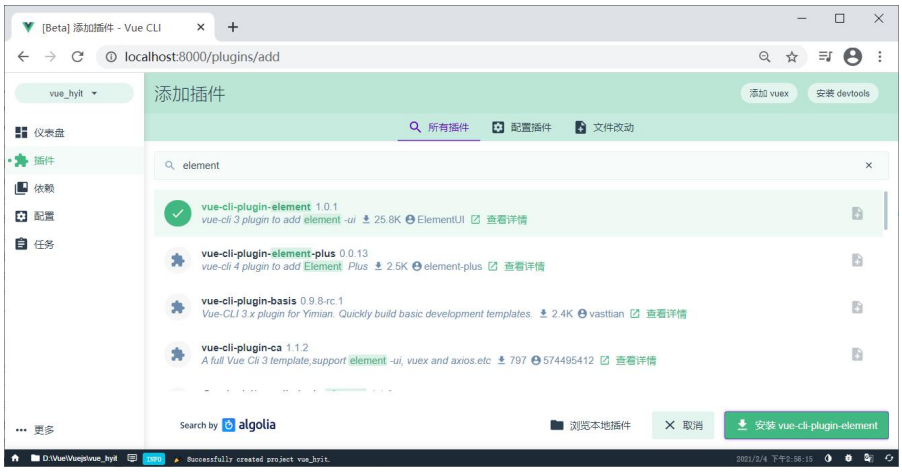

图 12-4 添加插件

插件 element-ui 安装成功之后，按默认方式进行配置即可。在项目的 src→plugins→element.js 文件中自动添加的代码如下，可以自动导入安装的 element 插件。

```
import Vue from 'vue'
import Element from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'
Vue.use(Element)
```

6. 安装网络模块 axios

选中左侧导航栏的“依赖”选项卡，单击右上角“安装依赖”，在运行依赖中查找到 axios 最新版本“axios 0.21.1”，单击右下角“安装 axios”按钮如图所示。

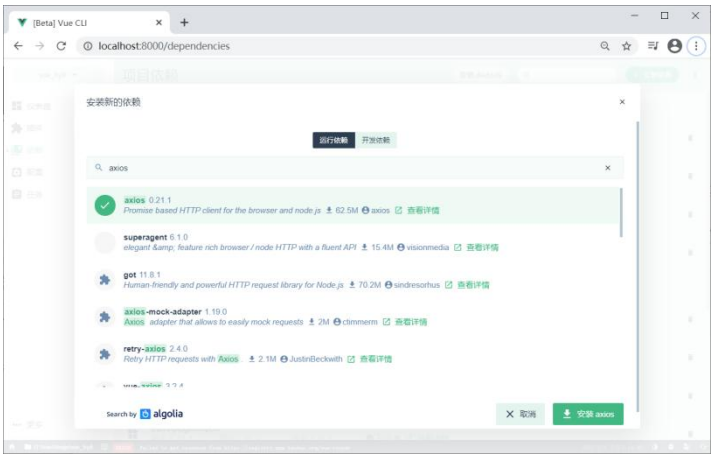

图 12-5 安装 axios 依赖

项目成功创建后，在 src→components 路径下分别添加主页组件 Home.vue 和登录组件 Login.vue，在 src→assets 路径下分别添加 css 和 img 文件夹用于存放样式文件和图片，项目目录如图所示。

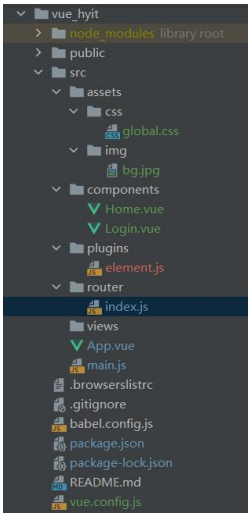

图 12-6 安装 axios 依赖

12.2 项目后台登录页面布局

后台登录效果如图所示。

图 12-7 登录效果图

1. 配置项目入口文件 main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import './plugins/element.js'
import '@/assets/css/global.css'   //导入全局样式文件
import axios from 'axios'          //导入网络请求
//把包 axios 挂载到 Vue 的原型对象上，这样每个 Vue 组件都可以通过 this 直接访问到 $http，从而发起网络请求
Vue.prototype.$http=axios
//配置本地网络请求的根路径
axios.defaults.baseURL='http://127.0.0.1:8888/api/private/v1/'
Vue.config.productionTip = false
new Vue({
```

```
    router,
    render: h => h(App)
}).$mount('#app')
```

2. 配置网络请求

在 src→router→index.js 中配置项目内页面跳转的路由，当前只配置登录页与主页之间的路由关系，index.js 中代码如下：

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Login from '@/components/Login.vue'
import Home from '@/components/Home.vue'

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    redirect: '/login'
  },
  {
    path: '/login',
    component:Login
   },
  {
    path: '/home',
    component:Home
  }
]

const router = new VueRouter({
  routes
})

export default router
```

3. 设计登录组件

应用表单数据对象属性:model="loginForm"进行表单数据的绑定，应用表单验证规则属性:rules="loginFormRules"进行表单数据的预验证；应用表单方法 resetFields 对整个表单进行重置，将所有字段值重置为初始值并移除校验结果，应用表单方法 validate 对整个表单进行校验，参数为一个回调函数。该回调函数会在校验结束后被调用，并传入两个参数：是否校验成功和未通过校验的字段。若不传入回调函数，则会返回一个 promise。登录组件 Login.vue 中代码如下：

```
<template>
  <div class="login_container">
```

```
            <div class="login_box">
                <h1>学科专业一体化建设管理平台</h1>
                <el-form ref="loginFormRef" :model="loginForm" :rules="loginFormRules" label-width="0px" class="login_form">
                    <el-form-item prop="username" class="username">
                        <el-input v-model="loginForm.username" prefix-icon="el-icon-user-solid" class="username"></el-input>
                    </el-form-item>
                    <el-form-item prop="password">
                        <el-input v-model="loginForm.password" prefix-icon="el-icon-lock" type="password" class="password"></el-input>
                    </el-form-item>
                    <el-form-item class="btns">
                        <el-button type="primary" @click="login" class="login">登录</el-button>
                        <el-button type="info" @click="resetLoginForm" class="reset">重置
                        </el-button>
                    </el-form-item>
                </el-form>
            </div>
        </div>
    </template>

    <script>
    export default {
        name: "Login",
        data() {
            return{
                //这是登录表单的数据绑定对象
                loginForm:{
                    username: 'Admin',
                    password: '123456'
                },
                //这是表单的验证规则对象
                loginFormRules:{
                    //验证用户名是否合法
                    username:[
                        { required: true, message: '请输入登录用户名', trigger: 'blur' },
                        { min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur' }
                    ],
                    //验证密码是否合法
                    password:[
                        { required: true, message: '请输入登录密码', trigger: 'blur' },
                        { min: 3, max: 15, message: '长度在 6 到 15 个字符', trigger: 'blur' }
                    ]
```

```
            }
        }
    },
    methods:{
        //重置按钮，重置登录表单
        resetLoginForm() {
            this.$refs.loginFormRef.resetFields();
        },
        login() {
            //根据对登录表单内容的预验证结果确定是否发起网络请求
            this.$refs.loginFormRef.validate(async valid => {
                if (!valid) return;  //校验失败，不发起网络请求
                //校验成功，发起网络请求
                const {data: res}=await this.$http.post("login", this.loginForm);
                //if (res.meta.status!==200) return console.log("登录失败");
                if (res.meta.status!==200) return this.$message.error("登录失败");
                //console.log("登录成功");
                this.$message.success("登录成功，正在跳转...");
                //将登录成功后的 token 保存到客户端的 sessionStorage 中
                window.sessionStorage.setItem("token",res.data.token);
                //通过路由的设置跳转到路由地址项目主页/home
                this.$router.push("/home");
            });
        }
    }
}
</script>

<style scoped>
    .login_container{
        background-color: #d9edf7;
        height: 100%;
        background-image: url("~assets/img/bg.jpg");
        background-size: cover;
        background-position: center;
    }
    .login_box{
        width: 650px;
        height: 400px;
        background-color: #d9edf7;
        border-radius: 3px;
        position: absolute;
        left: 12%;
        top:23%;
```

```
    }
    .btns{
        display: flex;
        justify-content: center;
    }
    .login_form{
        position: absolute;
        bottom: 100px;
        width: 100%;
        padding: 0 20px;
        box-sizing: border-box;
    }
    .username,.password,.login,.reset{
      font-size: 20px;
    }
</style>
```

4. 路由导航控制访问权限

vue-router 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。可以通过全局、单个路由独享或者组件级的植入路由导航过程中。方法 router.beforeEach()一般用于进入页面的限制，例如若没有登录系统就不能进入其他某些页面，只有登录系统之后才有权限查看某些页面，即路由拦截。如果用户没有登录，但是直接通过 URL 访问特定页面，需要重新导航到登录页面。方法 router.beforeEach()的三个参数如下：

to：即将要进入的目标路由对象，这个对象中包含 name、params、meta 等属性；

from：当前导航正要离开的路由，这个对象中包含 name、params、meta 等属性；

next：执行效果依赖 next 方法的调用参数。确保 next 函数在任何给定的导航守卫中都被严格调用一次。

- next(): 导航的状态就是 confirmed (确认的)；
- next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址；
- next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在 router-link 的 to prop 或 router.push 中的选项；
- next(error): (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 router.onError() 注册过的回调。

在 src→router→index.js 中添加路由导航守卫，若没有登录系统，token 值为空时用户无法访问除登录页面之外的页面，代码如下，登录成功后首页如图所示，在 Application 中可以查看保存的 token 值。

```
// 挂载路由导航守卫
router.beforeEach((to, from, next) => {
    // to 将要访问的路径
    // from 代表从哪个路径跳转而来
    // next 是一个函数，表示放行
    // next()  放行     next('/login')  表示强制跳转
```

```
    if (to.path === '/login') return next()
    const tokenStr = window.sessionStorage.getItem('token')      // getItem 获取 token
    if (!tokenStr) return next('/login')
    next()
})
export default router
```

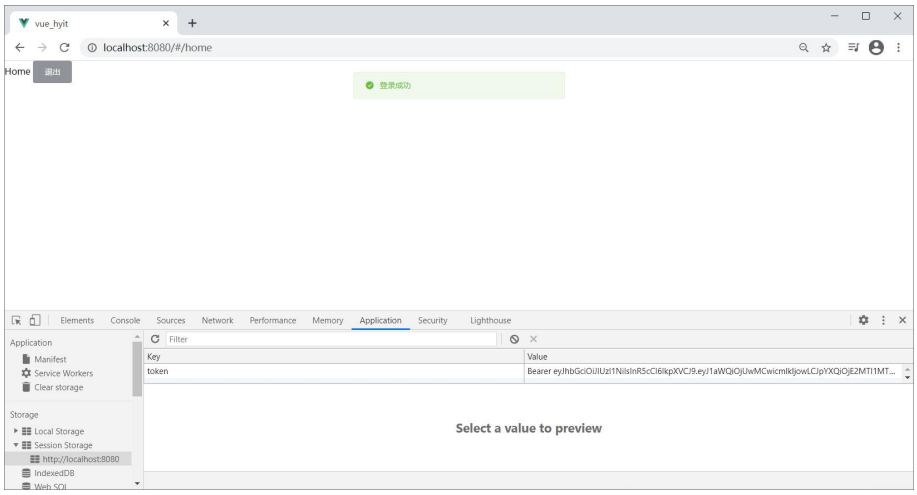

图 12-8 登录成功后首页

5. 实现退出系统功能

基于 token 的方式实现系统退出比较简单，应用 clear()方法销毁本地 token 即可。这样，后续的请求就不会携带 token，必须重新登录生成一个新的 token 之后才可以访问页面。在系统首页 Home.vue 组件中新增退出按钮实现系统退出，代码如下。

```
<template>
    <div>
        Home
        <el-button type="info" @click="logout">退出</el-button>
    </div>
</template>

<script>
    export default {
    name: "Home",
        methods: {
            logout() {
                window.sessionStorage.clear()
                this.$router.push('/login')
            }
        }
    }
</script>
```

12.3 项目后台主页布局

12.3.1 后台主页整体布局

应用 ElementUI 中的 Container 布局容器提供的页面主体布局样例选择一种布局方式，在首页 Home.vue 中添加页面布局代码：

```
<template>
  <el-container>
    <!--页面头部区域-→
    <el-header>Header
      <el-button type="info" @click="logout">退出</el-button>
    </el-header>
    <!--  页面主体区域-→
    <el-container>
      <!--侧边栏-→
      <el-aside width="200px">Aside</el-aside>
      <!--页面右侧主体内容-→
      <el-container>
        <el-main>Main</el-main>
      </el-container>
    </el-container>
    <!--页面底部区域-→
    <el-footer>Footer</el-footer>
  </el-container>
</template>
```

为了便于在界面设计过程中观察页面中每部分的布局，暂时为页面中每个区域设置背景色，如图所示，在系统首页 src→components→Home.vue 组件中的布局代码如下：

```
<template>
  <el-container>
    <!--页面头部区域-→
    <el-header>Header
      <el-button type="info" @click="logout">退出</el-button>
    </el-header>
    <!--  页面主体区域-→
    <el-container>
      <!--侧边栏-→
      <el-aside width="200px">Aside</el-aside>
      <!--页面右侧主体内容-→
      <el-container>
        <el-main>Main</el-main>
      </el-container>
    </el-container>
    <!--页面底部区域-→
    <el-footer>Footer</el-footer>
```

```
    </el-container>
</template>

<style scoped>
    .el-header{background-color: #1e9fff;}
    .el-aside{background-color: #5fb878;}
    .el-main{background-color: #dee1e6;}
    .el-footer{background-color: #44495d;}
</style>
```

图 12-9 系统首页布局

通过浏览器的 Elements 选项卡，可以查看到类 el-container 高度没有设置高度全屏，因此设置最外层的 el-container 高度为 100%，系统首页布局如图所示。

```
<template>
    <el-container class="home-container">
        <!--页面头部区域-→
        ...
    </el-container>
</template>

<style scoped>
    .home-container{height: 100%;}
    ...
</style>
```

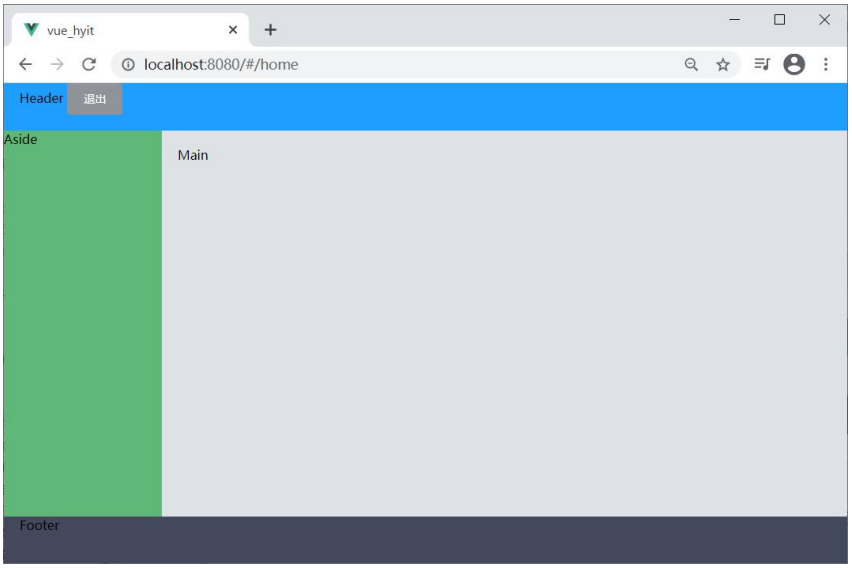

图 12-10 系统首页布局

12.3.2 系统首页 header 部分布局

系统首页 header 部分布局主要涉及样式设计，在系统首页 src→components→Home.vue 组件中页面头部区域的布局代码如下，系统首页 header 部分布局如图所示。

```
<!--页面头部区域-->
    <el-header>
      <div>
        <img src="~assets/img/hyit_logo.png" height="95px">
        <span>学科专业一体化建设管理平台</span>
      </div>
      <el-button type="info" @click="logout">退出</el-button>
    </el-header>

<style scoped>
  .home-container{height: 100%;}
  .el-header{
    background-color: #1e9fff;
    display: flex;
    justify-content: space-between;
    align-items: center;
    color: #fff;
    font-size: 40px;
    height: 100px !important;
  }
  .el-header div
  {
    display: flex;
    align-items: center;
  }
```

```
    .el-header div span
    {
       margin-left: 20px;
    }
    .el-aside{background-color: #5fb878;}
    .el-main{background-color: #dee1e6;}
    .el-footer{background-color: #44495d;}
</style>
```

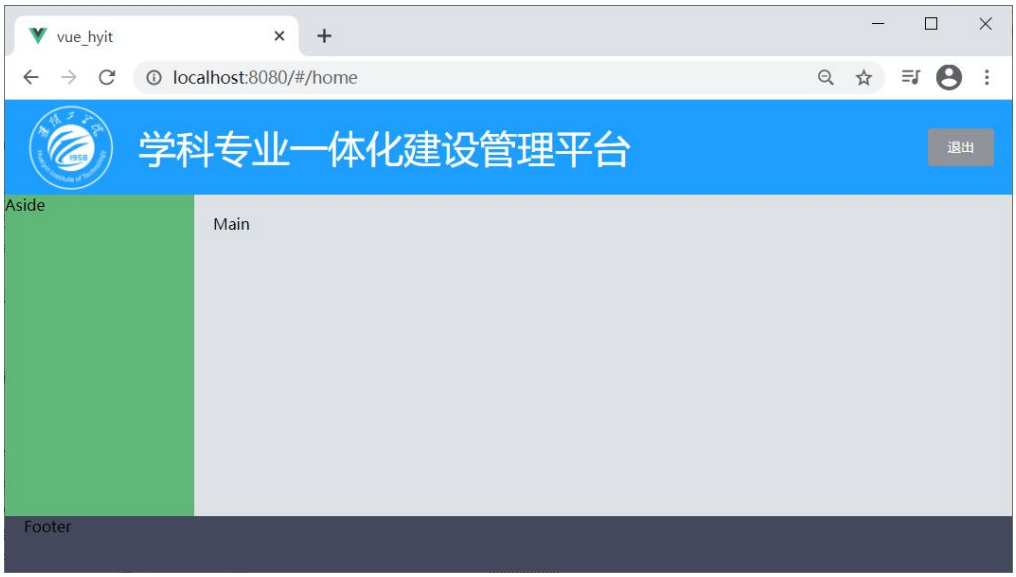

图 12-11 系统首页 header 部分布局

12.3.3 首页左侧 Aside 菜单栏布局

1. 左侧导航栏布局

ElementUI 提供了为网站提供导航功能的菜单：NavMenu 导航菜单。通过 el-menu-item-group 组件可以实现菜单进行分组，分组名可以通过 title 属性直接设定，也可以通过具名 slot 设定。应用 ElementUI 提供的自定义颜色模板设计系统首页左侧导航栏，由于导航数据需要从后台获取，因此只设计一个一级菜单和一个二级菜单，Home.vue 组件侧边栏部分设计如下：

```
<!--侧边栏-->
        <el-aside width="200px">
          <el-menu
              background-color="#545c64"
              text-color="#fff"
              active-text-color="#ffd04b">
            <!--一级菜单-->
            <el-submenu index="1">
              <template slot="title">
                <i class="el-icon-location"></i>
                <span>一级菜单</span>
              </template>
```

```
                <!--二级菜单-->
                <el-menu-item index="1-4">
                  <template slot="title">
                    <i class="el-icon-location"></i>
                    <span>二级菜单</span>
                  </template>
                </el-menu-item>
              </el-submenu>
            </el-menu>
          </el-aside>
```

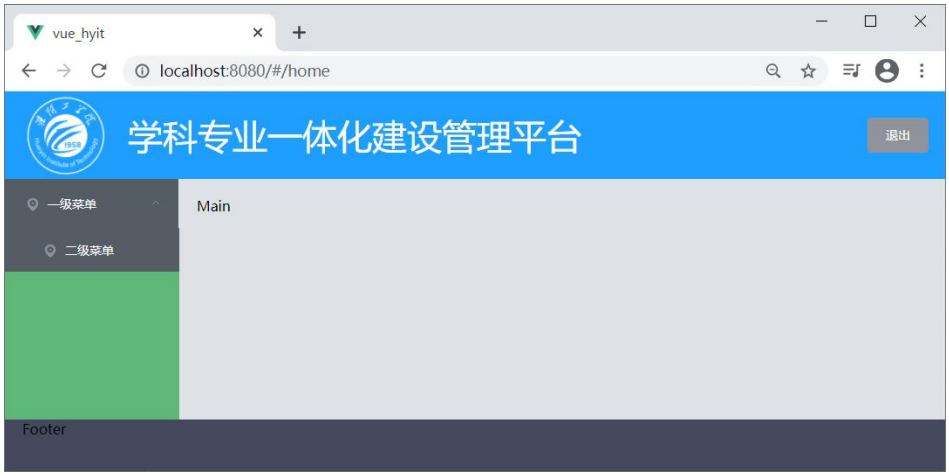

图 12-12 导航栏设计页面

2. 通过接口获取菜单数据

在项目入口文件 main.js 中通过 axios 请求拦截器添加 token，保证拥有获取数据的权限，在文件 main.js 中添加如下代码：

```
// axios 请求拦截
axios.interceptors.request.use(config => {
  // 为请求头对象，添加 Token 验证的 Authorization 字段
  config.headers.Authorization = window.sessionStorage.getItem('token')
  return config
})
```

3. 发起网络请求获取菜单数据

在 Home.vue 组件中通过 v-for 双层循环分别进行一级菜单和二级菜单的渲染，通过路由相关属性启用菜单的路由功能，Home.vue 中新增代码如下。

```
<script>
  export default {
  name: "Home",
    data() {
      return {
        // 左侧菜单数据
        menulist: []
      }
    },
```

```
        created() {
            this.getMenuList()
        },
        methods: {
            logout() {
                window.sessionStorage.clear()
                this.$router.push('/login')
            },
            // 获取所有的菜单
            async getMenuList() {
                const { data: res } = await this.$http.get('menus')
                if (res.meta.status !== 200) return this.$message.error(res.meta.msg)
                this.menulist = res.data
                console.log(res)
            }
        }
    }
</script>
```

其中，NavMenu 组件的 Menu 属性 unique-opened 可以设置是否只保持一个子菜单的展开；属性 router 用于开启菜单的路由链接，即是否使用 vue-router 的模式，启用该模式会在激活导航时以 index 作为 path 进行路由跳转。

```
<el-aside width="200px">
            <el-menu
                background-color="#545c64"
                text-color="#fff"
                active-text-color="#ffd04b" unique-opened router>
              <!--一级菜单-→
              <el-submenu :index="item.id + ' '" v-for="item in menulist" :key="item.id">
                <template slot="title">
                  <i class="el-icon-user-solid"></i>
                  <span>{{item.authName}}</span>
                </template>
                <!--二级菜单-→
                <el-menu-item :index="'/' + subItem.path" v-for="subItem in
item.children" :key="subItem.id">
                  <template slot="title">
                    <i class="el-icon-menu"></i>
                    <span>{{subItem.authName}}</span>
                  </template>
                </el-menu-item>
              </el-submenu>
            </el-menu>
          </el-aside>
```

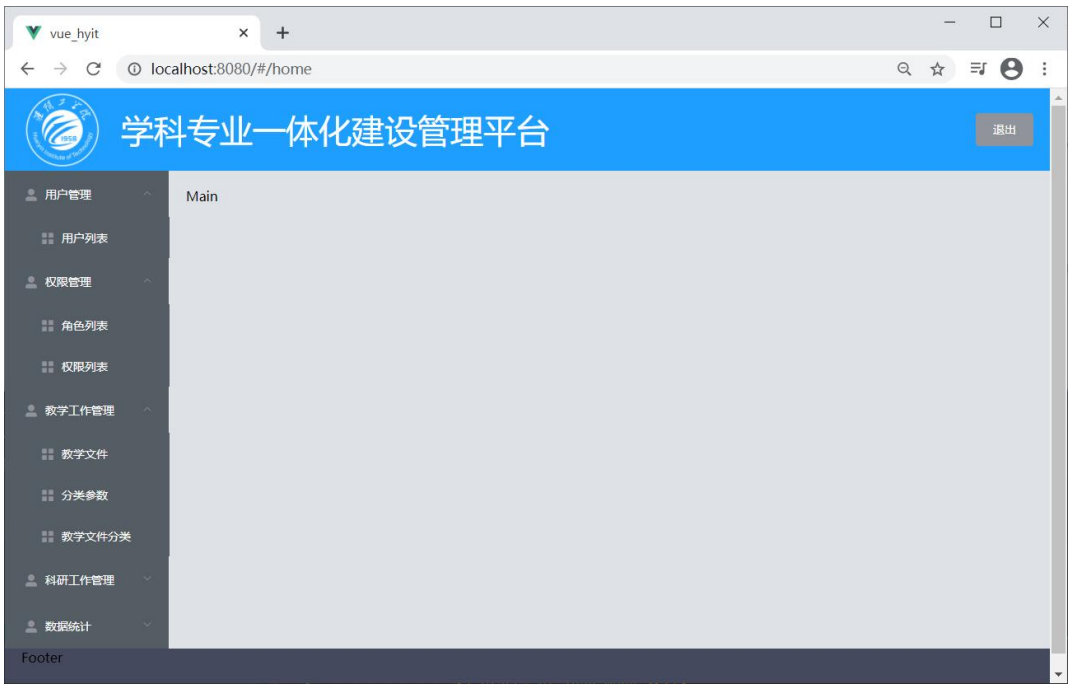

图 12-13 左侧导航栏页面

设置.el-menu 右侧边框值为 none，消除左侧导航栏边框参差不齐的情况。

```
.el-menu{
    border-right: none;
}
```

12.3.4 用户列表管理设计

1. 路由设置

新建组件 src→components→user→Users.vue 用于用户管理，在路由配置文件 src→router→index.js 中设置 home 子页面的路由信息。

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Login from '@/components/Login.vue'
import Home from '@/components/Home.vue'
import Users from '@/components/user/Users.vue'

Vue.use(VueRouter)

const routes = [
    {
        path: '/',
        redirect: '/login'
    },
    {
        path: '/login',
        component:Login
    },
```

```
    {
      path: '/home',
      component: Home,
      redirect: '/users',
      children: [
          { path: '/users', component: Users }
          ]
    }
]
```

在 src→components→Home.vue 组件中增加路由占位符，从而显示用户管理组件，如图所示。

```
<!--页面右侧主体内容-→
        <el-container>
          <el-main>
            <router-view></router-view>
          </el-main>
        </el-container>
```

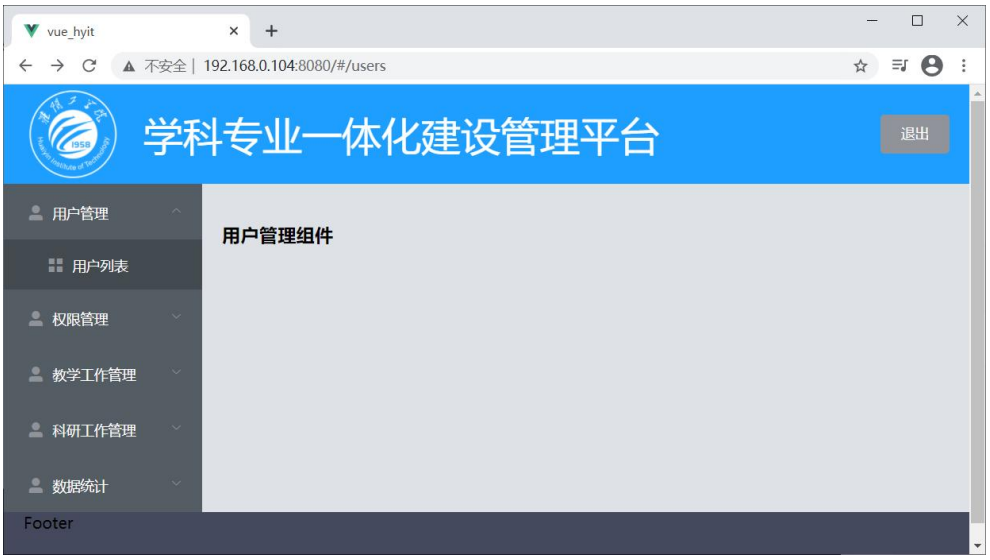

图 12-14 用户管理页面

2. 用户管理页面导航设计

ElementUI 提供的 Breadcrumb 面包屑组件显示当前页面的路径，快速返回之前的任意页面。通过设置 separator-class 可使用相应的 iconfont 作为分隔符，注意这将使 separator 设置失效。在组件 src→components→user→Users.vue 中添加面包屑组件代码：

```
<div>
  <el-breadcrumb separator-class="el-icon-arrow-right">
    <el-breadcrumb-item :to="{ path: '/home' }">首页</el-breadcrumb-item>
    <el-breadcrumb-item>用户管理</el-breadcrumb-item>
    <el-breadcrumb-item>用户列表</el-breadcrumb-item>
  </el-breadcrumb>
</div>
```

3. 用户列表设计

ElementUI 提供的 Card 卡片组件，可以将用户列表信息聚合在卡片容器中展示。应用<el-row>组件布局用户列表内的布局，Row 组件提供 gutter 属性指定每一栏之间的间隔，默认间隔为 0。

Table 表格用于展示多条结构类似的数据，可对数据进行排序、筛选、对比或其他自定义操作，这里用于显示用户列表组件，当 el-table 元素中注入 data 对象数组后，在 el-table-column 中用 prop 属性来对应对象中的键名即可填入数据，用 label 属性来定义表格的列名，使用 width 属性定义列宽。

应用 Switch 开关组件设计用户状态，Switch 开关表示两种相互对立的状态间的切换，一般用于触发状态的开和关。绑定 v-model 到一个 Boolean 类型的变量。可以使用 active-color 属性与 inactive-color 属性来设置开关的背景色。

Pagination 分页组件用于当数据量过多时，使用分页分解数据。设置 layout，表示需要显示的内容，用逗号分隔，布局元素会依次显示。prev 表示上一页，next 为下一页，pager 表示页码列表，除此以外还提供了 jumper 和 total，size 和特殊的布局符号→，→后的元素会靠右显示，jumper 表示跳页元素，total 表示总条目数，size 用于设置每页显示的页码数量。size-change 和 current-change 事件用于处理页码大小和当前页变动时候触发的事件。page-sizes 接受一个整型数组，数组元素为展示的选择每页项目显示个数的选项，[100, 200, 300, 400]表示四个选项，每页显示 100 个、200 个、300 个或者 400 个。

4. 弹出对话框设计

ElementUI 提供的 Dialog 对话框，在保留当前页面状态的情况下，告知用户并承载相关操作。Dialog 弹出一个对话框，适合需要定制性更大的场景。Dialog 组件的内容可以是任意的，可以是表格或表单。before-close 仅当用户通过点击关闭图标或遮罩关闭 Dialog 时起效。如果在 footer 具名 slot 里添加了用于关闭 Dialog 的按钮，那么可以在按钮的点击回调函数里加入 before-close 的相关逻辑。Dialog 对话框的 visible 属性是否显示 Dialog，支持.sync 修饰符。

在组件 src→components→user→Users.vue 中添加用户列表布局代码：

```
<el-card>
    <!-- 搜索与添加区域 -->
    <el-row :gutter="20">
      <el-col :span="8">
      //使用 clearable 属性可以设置为一个可清空的输入框
      //在点击由 clearable 属性生成的清空按钮时触发 clear 事件
        <el-input placeholder="请输入内容" v-model="queryInfo.query" clearable
          @clear="getUserList">
          <el-button slot="append" icon="el-icon-search" @click="getUserList">
          </el-button>
        </el-input>
      </el-col>
      <el-col :span="4">
        <el-button type="primary" @click="addDialogVisible = true">添加用户
        </el-button>
      </el-col>
    </el-row>
    <!-- 用户列表区域 -->
```

```
            <el-table :data="userlist" border stripe>
              <el-table-column type="index"></el-table-column>//添加索引列
              <el-table-column label="姓名" prop="username"></el-table-column>
              <el-table-column label="邮箱" prop="email"></el-table-column>
              <el-table-column label="电话" prop="mobile"></el-table-column>
              <el-table-column label="角色" prop="role_name"></el-table-column>
              <el-table-column label="状态">  //设计状态列
                <template slot-scope="scope">
                  <el-switch v-model="scope.row.mg_state"
                    @change="userStateChanged(scope.row)">
                  </el-switch>
                </template>
              </el-table-column>
              <el-table-column label="操作" width="180px">
                <template slot-scope="scope">
                  <!-- 修改按钮 -->
                  <el-button type="primary" icon="el-icon-edit" size="mini"></el-button>
                  <!-- 删除按钮 -->
                  <el-button type="danger" icon="el-icon-delete" size="mini"></el-button>
                  <!-- 分配角色按钮 -->
                  <el-tooltip effect="dark" content="分配角色"
                      placement="top" :enterable="false">
                    <el-button type="warning" icon="el-icon-setting" size="mini"></el-button>
                  </el-tooltip>
                </template>
              </el-table-column>
            </el-table>
            <!-- 分页区域 -->
            <el-pagination @size-change="handleSizeChange"
    @current-change="handleCurrentChange" :current-page="queryInfo.pagenum"
    :page-sizes="[1, 2, 5, 10]" :page-size="queryInfo.pagesize"
    layout="total, sizes, prev, pager, next, jumper" :total="total">
            </el-pagination>
          </el-card>

    <!-- 添加用户的对话框 -->
          <el-dialog title="添加用户" :visible.sync="addDialogVisible" width="50%"
@close="addDialogClosed">
            <!-- 内容主体区域 -->
            <el-form :model="addForm" :rules="addFormRules"
    ref="addFormRef" label-width="70px">
              <el-form-item label="用户名" prop="username">
                <el-input v-model="addForm.username"></el-input>
              </el-form-item>
```

```
            <el-form-item label="密码" prop="password">
              <el-input v-model="addForm.password"></el-input>
            </el-form-item>
            <el-form-item label="邮箱" prop="email">
              <el-input v-model="addForm.email"></el-input>
            </el-form-item>
            <el-form-item label="手机" prop="mobile">
              <el-input v-model="addForm.mobile"></el-input>
            </el-form-item>
          </el-form>
          <!-- 对话框的底部区域 -->
          <span slot="footer" class="dialog-footer">
            <el-button @click="addDialogVisible = false">取 消</el-button>
            <el-button type="primary" @click="addUser">确 定</el-button>
          </span>
        </el-dialog>
```

若组件中绑定的事件有传入事件名称字符串/字符串参数，这里用反单引号，是模板字符串（template string），增强版的字符串，用反引号（`）标识。它可以当作普通字符串使用，也可以用来定义多行字符串，或者在字符串中嵌入变量。

```
export default {
  name: "Users",
  data() {
    // 验证邮箱的规则
    var checkEmail = (rule, value, cb) => {
      // 验证邮箱的正则表达式
      const regEmail = /^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(\.[a-zA-Z0-9_-])+/
      if (regEmail.test(value)) {
        // 合法的邮箱
        return cb()
      }
      cb(new Error('请输入合法的邮箱'))
    }
    // 验证手机号的规则
    var checkMobile = (rule, value, cb) => {
      // 验证手机号的正则表达式
      const regMobile =
        /^(0|86|17951)?(13[0-9]|15[012356789]|17[678]|18[0-9]|14[57])[0-9]{8}$/
      if (regMobile.test(value)) {
        return cb()
      }
      cb(new Error('请输入合法的手机号'))
    }
    return {
      // 获取用户列表的参数对象
```

```
queryInfo: {
  query: '',
  // 当前的页数
  pagenum: 1,
  // 当前每页显示多少条数据
  pagesize: 2
},
userlist: [],
total: 0,
// 控制添加用户对话框的显示与隐藏
addDialogVisible: false,
// 添加用户的表单数据
addForm: {
  username: '',
  password: '',
  email: '',
  mobile: ''
},
// 添加表单的验证规则对象
addFormRules: {
  username: [
    { required: true, message: '请输入用户名', trigger: 'blur' },
    {
      min: 3,
      max: 10,
      message: '用户名的长度在 3~10 个字符之间',
      trigger: 'blur'
    }
  ],
  password: [
    { required: true, message: '请输入密码', trigger: 'blur' },
    {
      min: 6,
      max: 15,
      message: '用户名的长度在 6~15 个字符之间',
      trigger: 'blur'
    }
  ],
  email: [
    { required: true, message: '请输入邮箱', trigger: 'blur' },
    { validator: checkEmail, trigger: 'blur' }
  ],
  mobile: [
    { required: true, message: '请输入手机号', trigger: 'blur' },
```

```
                { validator: checkMobile, trigger: 'blur' }
            ]
        }
    }
},
created() {     //获取用户列表
    this.getUserList()
},
methods: {
    async getUserList() {
        const { data: res } = await this.$http.get('users', {
            params: this.queryInfo
        })
        if (res.meta.status !== 200) {
            return this.$message.error('获取用户列表失败！')
        }
        this.userlist = res.data.users
        this.total = res.data.total
        console.log(res)
    },
    // 监听 pagesize 改变的事件
    handleSizeChange(newSize) {
        // console.log(newSize)
        this.queryInfo.pagesize = newSize
        this.getUserList()
    },
    // 监听 页码值 改变的事件
    handleCurrentChange(newPage) {
        console.log(newPage)
        this.queryInfo.pagenum = newPage
        this.getUserList()
    },
    // 监听 switch 开关状态的改变
    async userStateChanged(userinfo) {
        console.log(userinfo)
        const { data: res } = await this.$http.put(
            `users/${userinfo.id}/state/${userinfo.mg_state}`
        )
        if (res.meta.status !== 200) {
            userinfo.mg_state = !userinfo.mg_state
            return this.$message.error('更新用户状态失败！')
        }
        this.$message.success('更新用户状态成功！')
    },
```

```
        // 监听添加用户对话框的关闭事件
        addDialogClosed() {
            this.$refs.addFormRef.resetFields()
        },
        // 点击按钮，添加新用户
        addUser() {
            this.$refs.addFormRef.validate(async valid => {
                if (!valid) return
                // 可以发起添加用户的网络请求
                const { data: res } = await this.$http.post('users', this.addForm)
                if (res.meta.status !== 201) {
                    this.$message.error('添加用户失败！')
                }
                this.$message.success('添加用户成功！')
                // 隐藏添加用户的对话框
                this.addDialogVisible = false
                // 重新获取用户列表数据
                this.getUserList()
            })
        }
    }
}
```

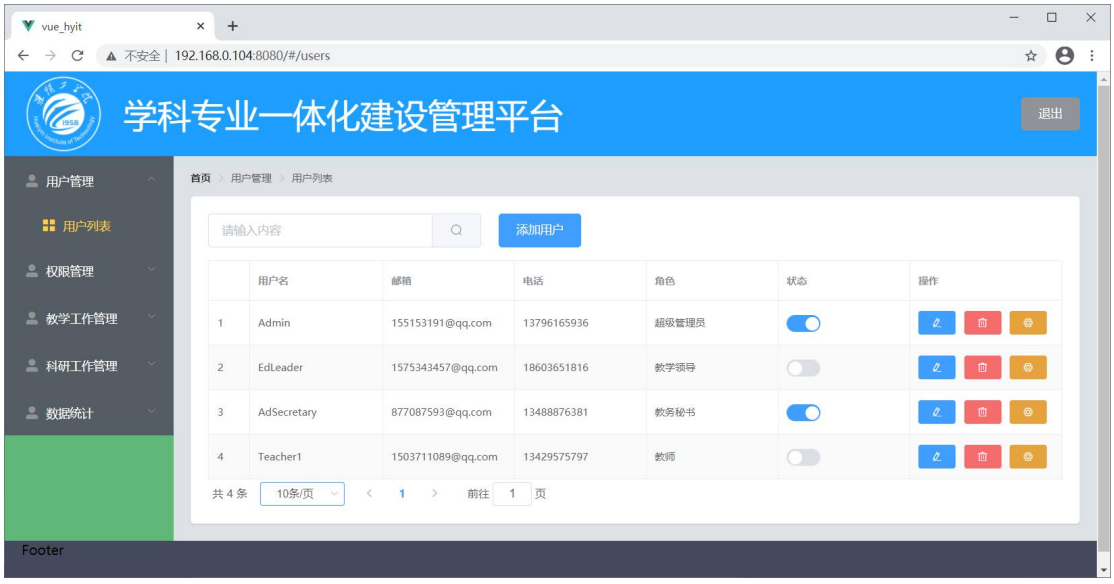

图 12-15 用户管理页面

最后，可以将 header 右侧的“退出”按钮改为用户的弹出式菜单形式的系统管理，其中的弹出式菜单使用 Element UI 组件中的“Dropdown 下拉菜单”组件，弹出式子菜单左侧使用图标组件，代码如下，如图 12-12 所示。

```
<!--用户登录展示的头像-->
        <div class="right_box">
            <el-dropdown>
```

```
                <img src="../assets/img/user.jpeg">
                <!--下拉菜单-→
                <el-dropdown-menu slot="dropdown">
                  <el-dropdown-item icon="el-icon-house">返回首页</el-dropdown-item>
                  <el-dropdown-item icon="el-icon-ship">消息通知</el-dropdown-item>
                  <el-dropdown-item icon="el-icon-user">个人中心</el-dropdown-item>
                  <el-dropdown-item icon="el-icon-switch-button">退出登录
</el-dropdown-item>
                </el-dropdown-menu>
              </el-dropdown>
            </div>
```

图 12-16 系统首页 header 部分弹出式菜单布局