电商后台管理系统

1. 项目描述
2. 项目概述: 电商后台管理商品,客户权限等的项目
3. 功能模块: 登录和登出

用户管理

权限管理

商品管理

订单管理

数据统计

1. 开发模式: 前后端分离的开发模式
2. 技术栈:

前端技术栈:

Vue, vue-router, element-ui, axios, echarts

后端技术栈(了解):

Node.js, express, jwt, mysql, sequlize

1. 项目初始化
2. 安装脚手架
3. 通过vue脚手架创建项目
4. 配置vue路由
5. 配置element-ui插件
6. 配置axios依赖包
7. 初始化git远程仓库
8. 将项目托管到github中
9. 登录和登出功能
10. 登录功能实现

登录业务流程:

1. 输入登录信息
2. 登录前的预验证
3. 调用后台接口进行验证
4. 路由导航守卫控制访问权限
5. 通过验证之后根据后台响应跳转到项目主页

相关技术点

1. http是无状态的
2. 通过cookie在客户端记录状态
3. 通过session在服务端记录状态
4. 通过token方式维持状态

Token原理分析



1. 登录实现步骤
2. 创建登录分支

git checkout -b 分支名

查看分支: git branch

1. 登录组件界面布局
2. 将组件导入路由

import Login from '../components/Login.vue'

1. 创建相应的路由规则
2. 导入axios依赖

import axios from 'axios'

// 配置请求的跟路径

axios.defaults.baseURL = 'http://127.0.0.1:8888/api/private/v1/'

Vue.prototype.$http = axios

1. 添加路由占位符,渲染到app根组件中

    <!-- 路由占位符 -->

    <router-view></router-view>

1. 实现登录相关功能

绑定数据

利用element-ui中的属性与方法

登录前的预验证

根据预验证是否发起请求

调用后台接口进行验证

路由导航守卫控制访问权限

// 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')  强制跳转

  if (to.path === '/login') return next()

  // 获取token

  const tokenStr = window.sessionStorage.getItem('token')

  // 如果没有token强制跳转到登录

  if (!tokenStr) return next('/login')

  next()

})

通过验证之后根据后台响应跳转到项目主页

        this.$router.push('/home')

1. 登出实现步骤

实际上就是销毁token,在其他页面退出按钮的事件处理逻辑中销毁

    logout() {

      window.sessionStorage.clear()

      this.$router.push('/login')

    }

1. 提交代码
2. 主页布局
3. 主页部分分为头部,侧边导航栏,右侧内容区域
4. 路由重定向

登录后在内容区域展示welcome组件

首先创建组件并导入路由模块

为home路由下创建子路由,然后路由重定向到welcome

    {

      path: '/home',

      component: Home,

      redirect: '/welcome',

      children: [{ path: '/welcome', component: Welcome }]

    }

在内容区域添加路由占位符

1. 头部部分分为标示和退出按钮

退出功能的实现逻辑:

    logout() {

      window.sessionStorage.clear()

      this.$router.push('/login')

    }

1. 侧边导航栏分别为: 用户管理,权限管理,商品管理,订单管理,数据统计

导航栏内容是希望打开网页就能自动获取数据,所以用生命周期函数来实现自动获取数据.

实现导航栏布局:

页面结构

获取数据:

Data创建用于渲染的数据

  data () {

    return {

      // 左侧导航栏数据

      menuList: []

    }

  },

创建获取数据方法

    async getMenuList () {

      // 发送请求并获取数据,这里要通过对象解构的方式方便获取数据

      const { data: res } = await this.$http.get('menus')

      if (res.meta.status !== 200) return this.$message.error('获取导航栏参数失败')

      this.menuList = res.data

    }

在created生命周期中引用获取数据方法

  created () {

    // 获取导航列表数据

    this.getMenuList()

  },

渲染数据

 <el-submenu :index="item.id" v-for="item in menuList" :key="item.id">

            <!-- 一级菜单的模板区域 -->

            <template slot="title">

              <!-- 文本 -->

              <span>{{item.authName}}</span>

            </template>

            <!-- 二级菜单 -->

            <el-menu-item :index="subItem" v-for="subItem in item.children" :key="subItem.id">

              <template>

                <!-- 文本 -->

                <span>{{subItem.authName}}</span>

              </template>

            </el-menu-item>

          </el-submenu>

部分功能实现

导航栏只允许一个导航打开,用组件自带属性即可

        <!-- unique-opened表示是否只允许一个导航打开 -->

        <el-menu background-color="#333744" text-color="#fff" active-text-color="#409EFF" :unique-opened="true">

折叠侧边导航栏,组件自带属性

        <!-- collapse控制导航栏是否折叠 -->

        <el-menu background-color="#333744" text-color="#fff" active-text-color="#409EFF" :unique-opened="true" :collapse="isCollapse" :collapse-transition="false">

在data中创建响应的数据

isCollapse: false

通过方法来控制数据改变

    toggleCollapse () {

      this.isCollapse = !this.isCollapse

    }

将方法绑定到事件源

        <div class="toggle-button" @click="toggleCollapse">|||</div>

控制侧边栏一起折叠(导航在侧边栏内,属性只控制导航本身折叠)

      <!-- 判断isCollapse的值来控制整个侧边栏折叠 -->

      <el-aside :width="isCollapse ? '64px' : '200px'">

侧边导航栏的路由链接的改造

开启路由模式,添加router属性值为true即可

        <!-- router表示路由模式，将index变为路由地址点击即可跳转 -->

        <el-menu background-color="#333744" text-color="#fff" active-text-color="#409EFF" :unique-opened="true" :collapse="isCollapse" :collapse-transition="false" router>

保存菜单的激活状态

属性default-active可以控制当前激活的选项

        <!-- router表示路由模式，将index变为路由地址点击即可跳转 -->

        <el-menu background-color="#333744" text-color="#fff" active-text-color="#409EFF" :unique-opened="true" :collapse="isCollapse" :collapse-transition="false" router :default-active="activePath">

当点击选项时，本地储存这个选项的地址，然后使绑定的值等于这个值，同时在created周期中让激活的选项的值等于储存的值，

   saveNavState (activePath) {

      window.sessionStorage.setItem('activePath', activePath)

      this.activePath = activePath

    }

解决点击当前地址控制台报错

// 解决点击当前地址控制台报错

const originalPush = VueRouter.prototype.push

VueRouter.prototype.push = function push (location) {

  return originalPush.call(this, location).catch(err => err)

}

1. 内容区域为当前点击的导航栏相应的内容

1）