电商后台管理系统

此文件不是项目心得,而是学习项目的思路历程

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. 用户列表
2. 用户列表布局:头部面包屑导航,卡片视图区域
3. 面包屑导航布局

    <!-- 面包屑导航区域 -->

    <el-breadcrumb separator-class="el-icon-arrow-right">

      <!-- 为首页添加路由跳转 -->

      <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>

1. 卡片视图区域:

搜索框和添加用户

1. 用户列表区域
2. 用户信息列表布局

获取用户信息

      // 请求参数(获取用户的参数对象)

      queryInfo: {

        query: '',

        // 当前的页数

        pagenum: 1,

        // 当前每页显示多少条数据

        pagesize: 2

      },

      // 具体用户信息

      userList: [],

发送请求并获取用户列表信息

  created () {

    this.getUserList()

  },

方法逻辑

    async getUserList () {

      const { data: res } = await this.$http.get('users', { params: this.queryInfo })

      console.log(res)

      if (res.meta.status !== 200) return this.$message.error('用户列表获取失败')

      this.userList = res.data.users

      this.total = res.data.total

    }

1. 分页区域布局

    <!-- 分页区域 -->

    <!--  @size-change事件是每页多少行改变触发, @current-change当前页面的值发生改变触发 current-page当前展示的是第几页的数据 page-sizes配置每页多少行的选项 page-size当前情况下每页多少行数据 layout打开多少分页功能 total总数-->

    <el-pagination @size-change="handleSizeChange" @current-change="handleCurrentChange" :current-page="queryInfo.pagenum" :page-sizes="[1, 2, 3, 4]" :page-size="queryInfo.pagesize" layout="total, sizes, prev, pager, next, jumper" :total="total">

    </el-pagination>

分页功能实现逻辑

    handleSizeChange (newSize) {

      // 修改数据并重新发送请求

      this.queryInfo.pagesize = newSize

      this.getUserList()

    },

    handleCurrentChange (newPage) {

      this.queryInfo.pagenum = newPage

      this.getUserList()

    }

1. 用户列表功能实现

用户状态功能修改

添加事件

          <el-switch v-model="scope.row.mg\_state" @change="userStateChanged(scope.row)">

          </el-switch>

发送请求修改数据

    async userStateChanged (userinfo) {

      const { data: res } = await this.$http.put(`users/${userinfo.id}/state/${userinfo.mg\_state}`)

      if (res.meta.status !== 200) {

        // 虽然发送请求时失败但是页面状态还是1改变了, 所以要变回去

        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！')

    }

实现搜索功能

         <!-- clearable清空输入框 clear事件在清空时触发-->

          <el-input placeholder="请输入内容" v-model="queryInfo.query" clearable @clear="getUserList">

            <el-button icon="el-icon-search" @click="getUserList" slot="append"></el-button>

          </el-input>

添加用户功能(点击出现添加用户对话框)

布局

为添加用户添加校验规则

实现表单重置

绑定事件

    <el-dialog title="添加用户" :visible.sync="addDialogVisible" width="50%" @close="addDialogClosed">

      <!-- close关闭时触发 -->

事件逻辑

    // 添加用户对话框关闭时

    // resetField重置表单

    addDialogClosed () {

      this.$refs.addFormRef.resetFields()

    }

添加用户前的预校验使用validate校验函数

添加用户发送请求

    addUser () {

      // 预校验

      this.$refs.addFormRef.validate(async valid => {

        if (!valid) {

          // 可以发送请求

          const { data: res } = await this.$http.post('users', this.addForm)

          if (res.meta.status !== 201) return this.$message.error('发送请求失败')

          this.$message.success('添加用户成功')

          // 关闭对话框

          this.addDialogVisible = false

          // 重新获取数据

          this.getUserList()

        }

      })

    }

操作部分按钮功能

修改功能(要求:点击修改按钮弹出对话框,并且有当前点击的选项的数据以及校验功能.)

展示修改功能的对话框,点击修改按钮时就发送请求

    // 显示对话框,同时获取数据

    async showEditDialog (id) {

      // console.log(id)

      const { data: res } = await this.$http.get('users/' + id)

      if (res.meta.status !== 200) {

        return this.$message.error('查询用户信息失败！')

      }

      // 通过绑定数据渲染用户数据

      this.editForm = res.data

      // 显示对话框

      this.editDialogVisible = true

    }

根据id查询用户信息

渲染用户信息

    <!-- 修改用户的对话框 -->

    <el-dialog title="修改用户" :visible.sync="editDialogVisible" width="50%">

      <el-form :model="editForm" :rules="editFormRules" ref="editFormRef" label-width="70px">

        <el-form-item label="用户名">

          <el-input v-model="editForm.username" disabled></el-input>

        </el-form-item>

        <el-form-item label="邮箱" prop="email">

          <el-input v-model="editForm.email"></el-input>

        </el-form-item>

        <el-form-item label="手机" prop="mobile">

          <el-input v-model="editForm.mobile"></el-input>

        </el-form-item>

      </el-form>

      <span slot="footer" class="dialog-footer">

        <el-button @click="editDialogVisible = false">取 消</el-button>

        <el-button type="primary">确 定</el-button>

      </span>

    </el-dialog>

实现用户表单的重置

    // 重置表单

    // 监听关闭事件

    showEditDialogClosed () {

      this.$refs.editFormRef.resetFields()

    }

提交表单的预验证

提交表单完成

    editUser () {

      this.$refs.editFormRef.validate(async valid => {

        if (!valid) return

        const { data: res } = await this.$http.put('users/' + this.editForm.id, { email: this.editForm.email, mobile: this.editForm.mobile })

        if (res.meta.status !== 200) return this.$message.error('修改用户失败')

        // 如果发送成功则数据库修改,只需重新获取用户列表即可

        // 关闭对话框

        this.editDialogVisible = false

        // 刷新数据列表

        this.getUserList()

        // 提示修改成功

        this.$message.success('更新用户信息成功！')

      })

删除用户

显示弹窗

发送请求删除信息