# Vue电商项目

# 登录界面

## 项目概述

### 1、电商项目基本业务概述

根据不同的应用场景，电商系统一般都提供了PC端、移动APP、移动Web、微信小程序、PC后台管理等多种终端访问形式。他们都通过服务器连接数据库。

### 电商后台管理系统的功能

电商后台管理系统用于管理用户账号、商品分类、商品信息、订单、数据统计等业务功能。

用户登录、退出登录、用户管理、权限管理(角色列表，权限列表)、商品管理(商品列表，分类管理，参数管理)、订单管理、数据统计

### 电商后台管理系统的开发模式(前后端分离)

电商后台管理系统整体采用前后端分离的开发模式，其中前端项目是基于Vue技术栈的SPA项目 single page applicaton

用户->前端项目(SPA)->后端项目->数据库

### 电商后台管理系统的技术选型

前端项目技术栈

Vue、Vue-router、Element-UI、Axios、Echarts

后端项目技术栈

Node.js、Express、Jwt(状态保持工具 类是于session的登录状态)、Mysql、Sequlize(操作数据的框架)了解

## 项目初始化

### 1、创建码云账号并配置SSH公钥

1. 安装Vue脚手架
2. 通过Vue脚手架创建项目
3. 配置Vue路由
4. 配置Element-UI组件库
5. 配置axios库
6. 初始化git远程仓库
7. 将本地项目托管到github或码云中

cmd vue ui

可视化创建项目

创建到指定目录 输入项目名称vue\_shop 勾取初始化git仓库(建议)init project

下一步 手动配置项目下一步 安装功能 Babel Router Linter/Formatter 使用配置文件

下一步Use history mode for router 不打开 hash模式兼容性更强些

点击select下拉框 ESL int + Standard config

Pick additional lint features:

选择Lint on save

创建项目

保存为新预设

预设名:vue\_shop\_preset 保存预设并创建项目

选择插件 右上角添加插件 vue-cli-plugin-element找到1.0.1版本

选中右下角点击安装

配置插件 将Fully import(默认)改为import on demand(将完整导入改为按需导入)

点击完成安装

选择依赖

安装依赖 axios 0.18.0选择 运行依赖 点击安装axios

gitee.com 登录账号 点击头像 设置 选择SSH公钥 点击怎样生成SSH公钥

cmd输入

ssh-keygen -t ed25519 -C "xxxxx@xxxxx.com"

连按三次回车

找到C:\Users\许高峰\.ssh里的id\_ed25519.pub

打开复制到gitee里面 名称imac

ssh -T git@gitee.com

### 将本地项目托管到项目中

新建仓库 名称vue\_shop 不适用Readme文件初始化这个仓库

cmd中

git config --global user.name "许高峰"

git config --global user.email "2456864054@qq.com"

在powershell终端中

没有仓库

mkdir vue\_shop

cd vue\_shop

git init

touch README.md

有仓库

git add .

git commit -m "first commit"

git remote add origin https://gitee.com/x2456864054/vue\_shop.git

git push -u origin "master"

### 后台项目环境安装配置

安装MySQL数据库 phpStudy 停止Apache 其他选项菜单 服务管理器 apache停止

D:\vs code html\mysql\mydb.sql

D:\vs code html\project\电商项目实战\3.vue-项目实战day1\素材\vue\_api\_server\mydb.sql

点击MySQL管理器 MySQL导入和导出 密码root 选择还原的文件mydb.sql 数据库名mydb

检验：其他选项菜单 打开MySQL工具 打开数据库目录 文件夹有mydb 打开有对应的文件

安装node.js环境

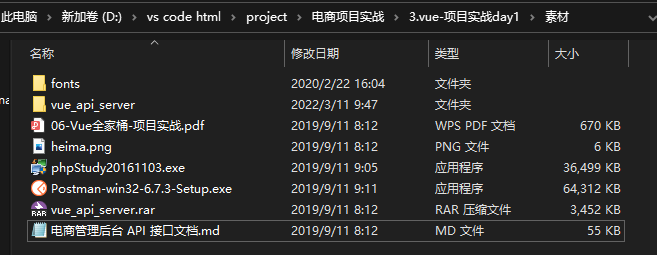
配置项目相关信息

启动项目

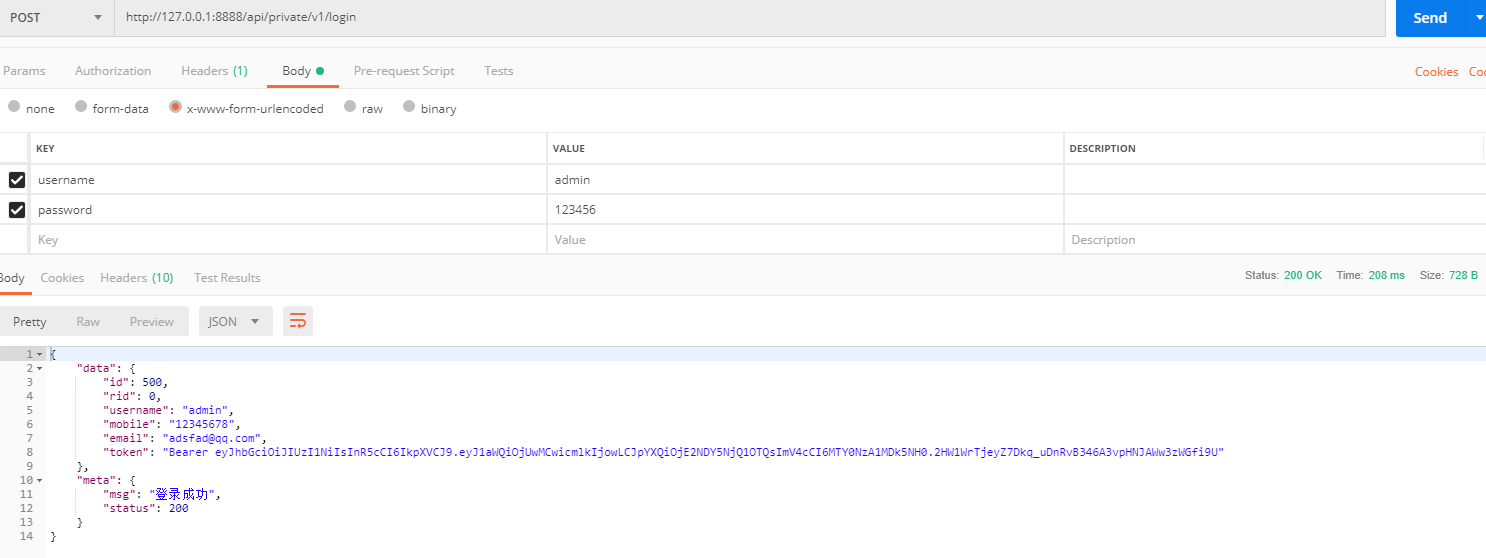
使用Postman测试后台项目接口是否正常

工具PowerShell D:\vs code html\project\电商项目实战\3.vue-项目实战day1\素材\vue\_api\_server目录上输入PowerShell 然后npm install

cls清屏 执行node .\app.js



找到API接口文档 用工具Postman进行检测接口



在dos命令窗口中，如果想切换到其他盘符，比如想切换到e盘根目录，可以在提示符后输入e:

使用cd /d e:命令也可以切换盘符

使用cd /d e:/data可以切换到指定盘符的指定目录中

cd .. 返沪上一级目录

cd \ 返回根目录

解决问题

phpstudy导入数据库无反应

使用[phpstudy](https://so.csdn.net/so/search?q=phpstudy&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_43416014/article/details/_blank)，在用MySQL管理导入sql文件，点击导入键没有响应，但是导出键能响应  
有大佬能帮我解一下问题吗，已经找了一天网页搜索，没有相关的问题，很难受。  
解决方法：  
之前我的sql文件是在F盘文件夹里，我把sql文件拉到了桌面，再重新选择要还原的文件，再点击导出就成功了

方法

## 登录退出-分析登录过程以及token的原理

### 登录概述

登录业务流程

1. 在冷路页面输入用户名和密码
2. 调用后台接口进行验证
3. 通过验证后，根据后台的响应状态跳转到项目主页

### 登录业务的相关技术点

http是无状态的

通过cookie在客户端记录状态

通过session在服务器端记录状态

通过token方式维护状态 前端和服务器存在跨域问题

### 登录-token原理分析

客户端登录页面输入用户名和密码登录，服务器验证通过后生成该用户的token并返回，客户端存储该token，客户端后续所有的请求都携带该token发送请求，服务器端验证token是否通过

1. 登录功能的实现

登录页面的布局

通过Element-UI组件实现布局

el-form el-form-item el-input el-button 字体图标

Vscode中git status

git checkout -b login创建login子分支

git brance 查看项目所有的分支

login分支和master分支并列

梳理项目结构

进入可视化面板，在任务中 点击serve 点击运行命令 启动app

### 渲染Login组件并实现路由重定向

创建Login.vue组件 并且在router.js中进行组件懒加载

可以在vue.config.js中设置别名

### 设置背景色并在屏幕中央绘制登录盒子

<style *lang*="less" *scoped*>

还要安装less依赖

在可视化工具及中找到依赖 安装依赖 安装到开发依赖中 less-loader和less

在可视化工具中任务停止重新运行下

在assets\css文件下创建global.css文件 在main.js中导入

### 绘制默认头像

### 绘制登录表单区域

[https://element.eleme.cn/#/zh-CN/](https://element.eleme.cn/" \l "/zh-CN/)

找到组件Form中的 Form表单 显示代码

<el-form ref="form" :model="form" label-width="80px">

<el-form-item label="活动名称">

<el-input v-model="form.name"></el-input>

</el-form-item>

</el-form>

创建plugins文件 element.js 导入组件

import {

  Button,

  Form,

  FormItem,

  Input,

} from 'element-ui'

Vue.use(Button)

Vue.use(Form)

Vue.use(FormItem)

Vue.use(Input)

找到组件Basic中的 Button按钮 显示代码

<el-button type="primary">主要按钮</el-button>

<el-button type="info">信息按钮</el-button>

*.btns* {

  display: flex;

  justify-content: flex-end;

}

弹性布局 横轴尾部对齐

### 绘制带icon的input输入框

找到组件Form中的 input输入框 显示代码

找到带 icon 的输入框

用属性的方式进行添加

前面的图标

<el-input placeholder="请输入内容"

prefix-icon="el-icon-search"

v-model="input2">

</el-input>

找到Basic中的icon图标

如果没有可以去阿里图标库找

D:\vs code html\project\电商项目实战\3.vue-项目实战day1\素材\fonts

找到iconfont.css 打开demo\_fontclass.html

第一步：引入项目下面生成的fontclass代码：

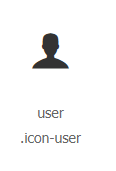
<link rel="stylesheet" type="text/css" href="./iconfont.css">

第二步：挑选相应图标并获取类名，应用于页面：

<i class="iconfont icon-xxx"></i>

将fonts复制到assets文件下

在main.js文件导入字体图标



### 实现表单的数据绑定

:model="ruleForm"

找到Form中的Form表单下的表单验证

 <el-form *ref*="loginFormRef" :*model*="loginForm" :*rules*="loginFormRules" *label-width*="0px" *class*="login\_form">

        <!-- 用户名 -->

        <el-form-item *prop*="username">

          <el-input *v-model*="loginForm.username" *prefix-icon*="iconfont icon-user"></el-input>

        </el-form-item>

</el-form>

<el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px" class="demo-ruleForm">

<el-form-item label="活动名称" prop="name">

<el-input v-model="ruleForm.name"></el-input>

</el-form-item>

rules: { name: [ { required: true, message: '请输入活动名称', trigger: 'blur' }, { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' } ],}

      loginFormRules: {

        // 验证用户名是否合法

        username: [

          { required: true, message: '请输入登录名称', 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: 6, max: 15, message: '长度在 6 到 15 个字符', trigger: 'blur' }

        ]

      }

### 实现表单的重置功能

找到Form中的Form下的Form Methods

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

ref="loginFormRef"

拿到表单的实例对象

<el-button *type*="info" @*click*="resetLoginForm">重置</el-button>

 resetLoginForm() {

      // console.log(this);

*this*.$refs.loginFormRef.resetFields()

    },

重置之后不会清空 而是返回默认用户名和密码 v-mode双向绑定的

### 实现登录前表单数据的预验证

找到Form中的Form下的Form Methods

validate 对整个表单进行校验的方法，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 Function(callback: Function(boolean,object))

login() {

*this*.$refs.loginFormRef.validate(*valid* => {

console.log(valid);

});

}

### 配置axios发起登录请求

在main.js中导入

import axios from 'axios'

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

Vue.prototype.$http = axios

**接口文档1.1**

运行phpStudy的MySQL启动

在D:\vs code html\project\电商项目实战\3.vue-项目实战day1\素材\vue\_api\_server中powershell工具打开node .\app.js运行api接口

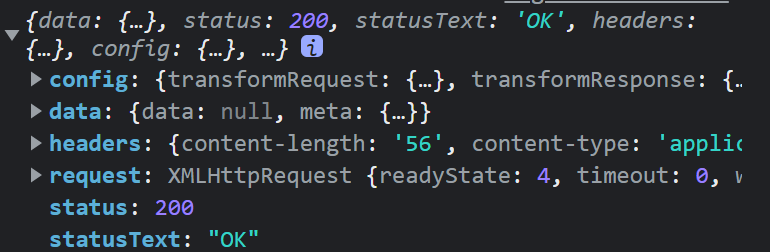
如果某个方法的返回值是promise我们可以用async和await来简化promise操作

await只能用在async的方法中

const result = await *this*.$http.post('login', *this*.loginForm)

console.log(result)

**接口文档1.2.1**



对result进行结构只需要data属性

 const { *data*: res } = await *this*.$http.post('login', *this*.loginForm)

将data重命名为res

console.log(res)



if (res.meta.status !== 200) return *this*.$message.error('登录失败！')

*this*.$message.success('登录成功')

判断登录成功和失败

### 配置message全局弹框组件

找到Message消息提示

在element.js导入Message

进行全局挂载

*Vue*.prototype.$message = Message

*this*.$message.error('登录失败！')

### 完善登录之后的操作

1. 将登录成功之后的token，保存到客户端的sessionStorage中
2. 项目中除了登录之外的其他API接口，必须在登录之后才能访问
3. token只应用在当前网站打开期间生效，所以将token保存在sessionStorage中

window.sessionStorage.setItem('token', res.data.token)

1. 通过编程式导航跳转到后台主页，路由地址是/home

*this*.$router.push('/home')

创建Home.vue组件

在router.js中导入Home组件 配置映射关系

### 路由导航守卫控制页面访问权限

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界面

在router.js中

// 挂载路由导航守卫

//为路由对象，添加beforeEach 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

//如果用户访问的是登录页，直接放行

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

  // 获取token

//从sessionStorage 中获取到 保存的token值

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

//如果没有token 强制跳转到登录页

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

  next()

})

### 退出功能实现原理

基于token的访问实现退出比较简单，只需要销毁本地的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Home.vue中

清空token

window.sessionStorage.clear()

跳转到登录页

this.$router.push(‘/login’)

logout() {

      window.sessionStorage.clear()

*this*.$router.push('/login')

    },

### 处理项目中的ESLint语法报错问题

点击输出

创建.prettierrc文件

{

  "semi": false,//不会额外加分号

  "singleQuote": true,//启用单引号这种格式

  "printWidth": 200 //

}

格式化文档快捷键 Ctrl+Alt+L

进入vue可视化项目里面进行管理

在D:\vs code html\project\电商项目实战\9.vue-项目实战day7\code进行cmd

然后进行vue ui进入可视化页面 解决可视化工具不能切换磁盘的问题

点解任务 运行 点击输出

怎么解决ESLint警告

找到.eslintrc.js 打开

在rules:{

'space-before-function-paren': 0

}

### 修改element-ui组件的按需导入形式

### 将本地代码提交代码到码云中

在终端中git status

git add .

git commit -m “完成了登录功能” 提交到本地服务

git branch

将login分支的代码合并更新到master主分支中

git checkout master 切换到主分支上

git branch

git merge login合并login分支

git push 推送到码云端服务

码云上只记录了master主分支，没有记录login分支

怎么将本地的login分支推送到码云上进行保存呢

git checkout login

git branch

git push -u origin login

## 主页布局

### 实现基本的主页布局

找到Basic的 Container布局容器 第四个布局

<el-container>

<el-header>Header</el-header>

<el-container>

<el-aside width="200px">Aside</el-aside>

<el-main>Main</el-main>

</el-container>

</el-container>

在element.js中进行导入注册组件

### 美化主页的header区域

进行flex布局

*.el-header* {

  background-color: #373d41;

  display: flex;

  justify-content: space-between;

  padding-left: 0;

  align-items: center;

  color: #fff;

  font-size: 20px;

  > div {

    display: flex;

    align-items: center;

    span {

      margin-left: 15px;

    }

  }

}

作用于同一级盒子进行布局

### 实现导航菜单的基本结构

菜单分为二级，并且可以折叠

<el-menu>

<el-submenu>

这个template是一级菜单的内容模板

<i class=””></i>

<span>一级菜单</span>

在一级菜单中，可以嵌套二级菜单

<el-menu-item>

<i class=””></i>

<span>二级菜单</span>

</el-menu-item>

</el-submenu>

</el-menu>

找到NavMenu导航菜单

 <el-menu

*background-color*="#545c64"

*text-color*="#fff"

*active-text-color*="#ffd04b">

      <el-submenu *index*="1">

        <template *slot*="title">

          <i *class*="el-icon-location"></i>

          <span>导航一</span>

        </template>

       <el-menu-item *index*="1-4-1">

<template *slot*="title">

          <i *class*="el-icon-location"></i>

          <span>导航一</span>

        </template>

</el-menu-item>

      </el-submenu>

    </el-menu>

在element.js进行导入注册组件

### 通过axios拦截器添加token验证

**保证拥有获取数据的权限**

axios.interceptors.request.use(*config* => {

  // console.log(config)

  NProgress.start()

*config*.headers.Authorization = window.sessionStorage.getItem('token')

  // 在最后必须 return config

  return *config*

})

// 在 response 拦截器中，隐藏进度条 NProgress.done()

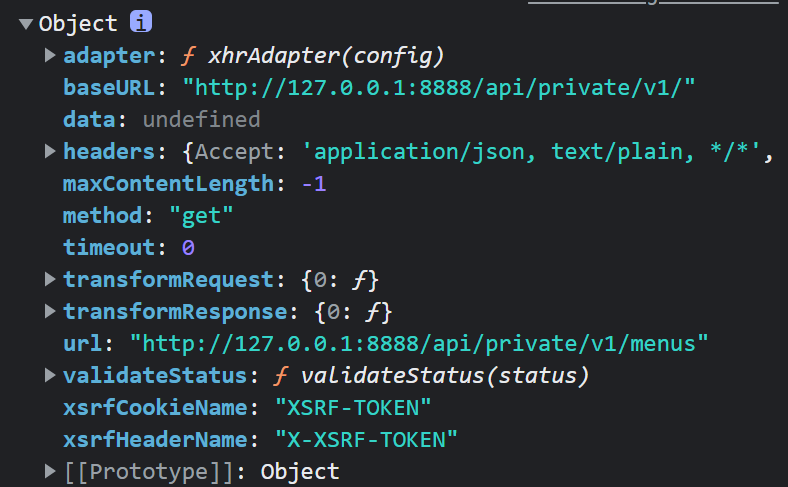
axios.interceptors.response.use(*config* => {

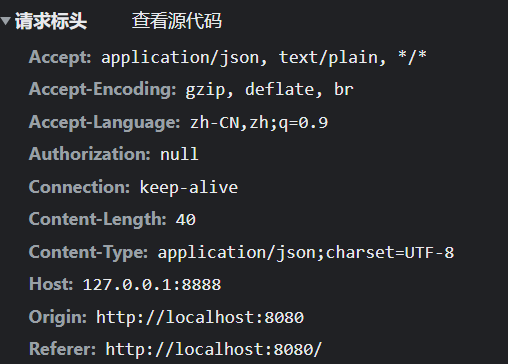
  NProgress.done()

  return *config*

})

打印config





### 获取左侧菜单数据

在Home.vue中

 created() {

*this*.getMenuList()

}

在methods中

async getMenuList() {

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

      if (res.meta.status !== 200) return *this*.$message.error(res.meta.msg)

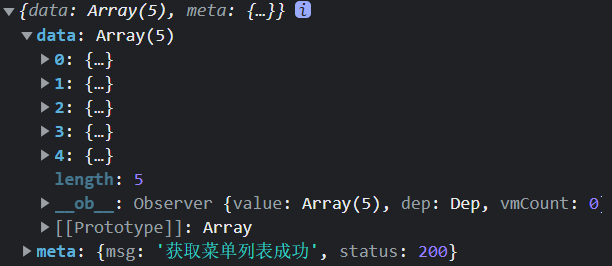
*this*.menulist = res.data

      console.log(res)

    },

**接口文档1.4.2**

打印res



### 通过双层for循环渲染左侧菜单

一级菜单渲染

<el-submenu :*index*="item.id + ''" *v-for*="item in menulist" :*key*="item.id">

:index只接受一个字符串，只作用于当前菜单

二级菜单渲染

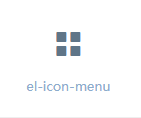
<el-menu-item :*index*="'/' + subItem.path" *v-for*="subItem in item.children" :*key*="subItem.id" @*click*="saveNavState('/' + subItem.path)">

### 选中项设置字体颜色并添加分类图标

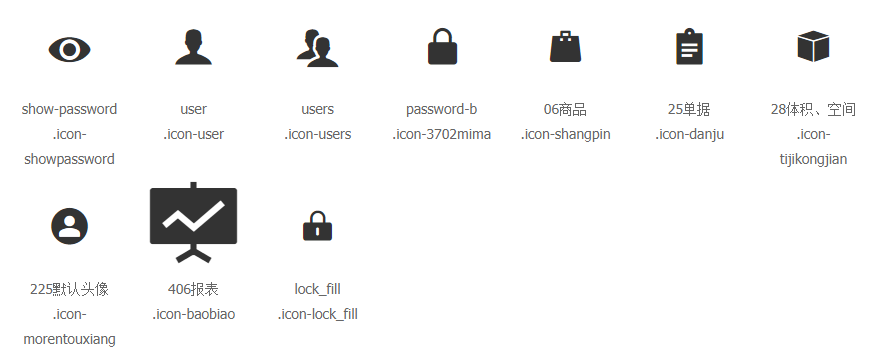
 <el-menu *background-color*="#333744" *text-color*="#fff" *active-text-color*="#409EFF" *unique-opened* :*collapse*="isCollapse" :*collapse-transition*="false" *router* :*default-active*="activePath">

active-text-color="#409EFF" 来点击设置字体颜色

找到icon图标



<i *class*="el-icon-menu"></i>



### 定义字体的对象 每循环一次展示一次图标

在data里面

iconsObj: {

        '125': 'iconfont icon-user',

        '103': 'iconfont icon-tijikongjian',

        '101': 'iconfont icon-shangpin',

        '102': 'iconfont icon-danju',

        '145': 'iconfont icon-baobiao'

      },

<i :*class*="iconsObj[item.id]"></i>

*.iconfont* {

  margin-right: 10px;

}

### 每次只能打开一个菜单项并解决边框问题

**其他菜单项关闭**

unique-opened 是否只保持一个子菜单的展开 boolean false

<el-menu unique-opened></el-menu>

*.el-aside* {

  background-color: #333744;

*.el-menu* {

    border-right: none;

  }

}

### 实现侧边栏的折叠与展开效果

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

letter-spacing: 0.2em;

使|||距离增加0.2em

toggleCollapse() {

*this*.isCollapse = !*this*.isCollapse

    },

collapse 是否水平折叠收起菜单（仅在 mode 为 vertical 时可用） boolean false

<el-menu :collapse=”true”></el-menu>

关闭动画效果

collapse-transition 是否开启折叠动画 boolean true

<el-menu :collapse-transition=”false”></el-menu>

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

### 实现首页路由的重定向效果

登录之后/home改为/welcome

创建Welcome.vue组件 在router.js里面导入 映射关系嵌套在home的子路由中

{

      path: '/home',

      component: Home,

      redirect: '/welcome',

      children: [

        { path: '/welcome', component: Welcome },}

在Home.vue中

  <el-main>

        <!-- 路由占位符 -->

        <router-view></router-view>

      </el-main>

### 实现侧边栏路由链接的改造

router 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 boolean false

<el-menu :router=”true”></el-menu> 简写<el-menu router></el-menu>

<el-menu-item :*index*="'/' + subItem.path"></el-menu-item>

# 用户管理

## 用户列表

### 通过路由的形式展示用户列表组件

在components/users/Users.vue

在router.js进行导入 挂载组件

在sessionStorage中保存左侧菜单的激活状态

default-active 当前激活菜单的 index string

在Home.vue中

<el-menu :default-active=”/users”></el-menu>

在二级菜单中

<el-menu-item>

@*click*="saveNavState('/' + subItem.path)

</el-menu-item>

 // 保存链接的激活状态

    saveNavState(*activePath*) {

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

*this*.activePath = *activePath //点击其他菜单后再点击回来也处于激活状态*

    }

<el-menu>

:*default-active*="activePath"

</el-menu>

 created() {

*this*.getMenuList()

*this*.activePath = window.sessionStorage.getItem('activePath')

  },

### 绘制用户列表组件的基础布局结构

找到Navigation中的Breadcrumb面包屑

 <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>

在element.js中按需导入

找到Others中的Card卡片

在element.js中按需导入

<el-card>

</el-card>

在global.css中 设置element-ui框架的样式

*.el-breadcrumb* {

  margin-bottom: 15px;

  font-size: 12px;

}

*.el-card* {

  box-shadow: 0 1px 1px rgba(0, 0, 0, 0.15) !important;

}

找到Form中的input输入框 找到Basic中的Layout布局 在element.js导入 注册

复合型输入框

 <el-row :*gutter*="20">

        <el-col :*span*="8">

          <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>

：gutter增加了输入框和按钮之间的距离(列和列之间的间隙) :span=”8”可以设置输入框的长度

### 获取用户列表数据

 // 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

      },

  async getUserList() {

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

        params: *this*.queryInfo

      })

}

**接口文档1.3.1**

打印res



### 使用el-table组件渲染基本的用户列表

找到Data中的table表格

 <el-table :*data*="userlist" *border* *stripe*>

//为表格添加索引列

        <el-table-column *type*="index"></el-table-column>

<el-table-column *label*="姓名" *prop*="username"></el-table-column>

</el-table>

在element.js中按需导入 注册

border加边框 stripe隔行变色

prop=”username”等价于userList数组里面的属性username

*.el-table* {

  margin-top: 15px;

  font-size: 12px;

}

### 自定义状态列的显示效果

 <el-table-column *label*="状态">

          <template *slot-scope*="scope">

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

          </template>

        </el-table-column>

slot-scope=”scope” <el-table-column>中的prop属性失效

scope.row可以获取该列表<el-table>的数据

找到Form中的switch开关 在element.js中按需导入 注册

在methods中

 // 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！')

    },

**接口文档1.3.3**

### 通过作用域插槽渲染操作列

 <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" @*click*="showEditDialog(scope.row.id)"></el-button>

            <!-- 删除按钮 -->

            <el-button *type*="danger" *icon*="el-icon-delete" *size*="mini" @*click*="removeUserById(scope.row.id)"></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" @*click*="setRole(scope.row)"></el-button>

            </el-tooltip>

          </template>

        </el-table-column>

找到Basic中的buttone按钮的图标按钮

type修改按钮类型颜色 icon修改按钮形状 size设置按钮大小

找到Others中的tooltip 在element.js按需导入 注册

effect设置提示颜色 placement提示信息位置 enterable鼠标到提示信息上隐藏false是布尔值要进行数据邦定

### 实现分页效果

找到Data中的Pagination分页 找到完整功能

 <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>

在element.js中进行按需导入 注册

:page-size 一页展示几条数据

@size-change 设置一页展示几条数据

@current-change 前往第几页显示当前页数

layout展示组件的功能 total共多少条 sizes每页多少条数据

// 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()

    },

改变了数据得重新调用this.getUserList()

console.log(newSize)



console.log(newPage)



*.el-pagination* {

  margin-top: 15px;

}

### 修改用户状态

找到Form中的switch开关 @change事件

 // 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！')

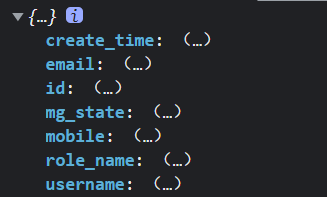
      }

      this.$message.success('更新用户状态成功！')

    },

打印userinfo

**接口文档1.3.3**



### 实现搜索功能

<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>

找到Form中的input输入框 可清空 clearable 清空按钮 @clear清空可以显示全部信息

## 添加用户

### 渲染添加用户的对话框

找到others中的Dialog对话框

在elements.js中按需导入 注册

 <!-- 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>

      <!-- 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>

:visible.sync窗口的属性绑定控制窗口的显示和隐藏

<el-button *type*="primary" @*click*="addDialogVisible = true">添加用户</el-button>

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

### 渲染添加用户的表单 表单验证

:mode=””绑定表单数据 :rules=””验证数据规则

// 添加用户的表单数据

      addForm: {

        username: '',

        password: '',

        email: '',

        mobile: ''

      },

 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'

          }

        ],

  email: [{ required: true, message: '请输入邮箱', trigger: 'blur' }, { validator: checkEmail, trigger: 'blur' }],

}

v-mode绑定的数据都会存储到:mode里面

### 自定义邮箱和手机号的校验规则

找到Form表单下的自定义校验规则

 // 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'))

    }

### 实现添加表单的重置操作

添加用户时候输入到一般取消重新进入 之前输入的信息重置 不再显示

<el-dialog @close=””>

 // 监听添加用户对话框的关闭事件

    addDialogClosed() {

      this.$refs.addFormRef.resetFields()

    },

### 实现添加用户前的表单预校验

<el-button @click=””>确定</el-button>

 // 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！')

        }

**接口文档1.3.2**

        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

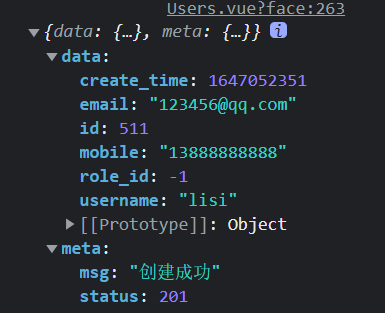
        this.getUserList()

      })

    },

打印valid是一个布尔值

### 调用API接口完成添加用户的操作



## 修改用户

### 修改用户的对话框

修改按钮里

加入@*click*="showEditDialog(scope.row.id)

 // 展示编辑用户的对话框

    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！')

      }

**接口文档1.3.4**

      this.editForm = res.data

      this.editDialogVisible = true

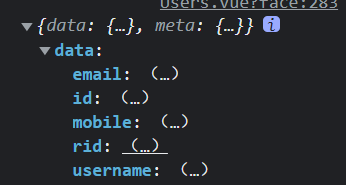
    },

### 根据id查询对应的用户信息

打印id



打印res



### 渲染修改用户的表单

找到Form中的Form表单的表单验证

用户名不需要有校验规则 所以不用prop disabled禁用修改

进行校验规则

:mode :rules

### 实现修改用户表单的重置操作

el-dialog中@close

 // 监听修改用户对话框的关闭事件

    editDialogClosed() {

      this.$refs.editFormRef.resetFields()

    },

### 完成提交修改之前的表单预验证

@clicke 确定 ref validate

// 修改用户信息并提交

    editUserInfo() {

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

        if (!*valid*) return

        // 发起修改用户信息的数据请求

        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('更新用户信息失败！')

        }

**接口文档1.3.5**

        // 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！')

      })

    },

### 提交表单完成用户信息的修改

## 删除用户

### 弹框询问用户是否确认删除数据

找到Notice中的MessageBox弹框的确认消息

在element.js进行按需导入 不需要注册 需要全局挂载

Vue.prototype.$confirm = MessageBox.confirm

找到删除按钮绑定删除事件@click=”removeUserById(scope.row.id)”

 // 根据Id删除对应的用户信息

    async removeUserById(*id*) {

      // 弹框询问用户是否删除数据

//返回值是Promise 可以用async await来优化

      const confirmResult = await this.$confirm('此操作将永久删除该用户, 是否继续?', '提示', {

        confirmButtonText: '确定',

        cancelButtonText: '取消',

        type: 'warning'

      }).catch(*err* => *err*)

      // 如果用户确认删除，则返回值为字符串 confirm

      // 如果用户取消了删除，则返回值为字符串 cancel

      // console.log(confirmResult)

      if (confirmResult !== 'confirm') {

        return this.$message.info('已取消删除')

      }

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

**接口文档1.3.6**

      if (res.meta.status !== 200) {

        return this.$message.error('删除用户失败！')

      }

      this.$message.success('删除用户成功！')

      this.getUserList()

    },

箭头函数只有一行代码时，可以省略{}和return

### 调用API完成删除用户的操作

### 创建user子分支并推送到码云仓库

git branch

git checkout -b user

git branch

git status

git add .

git status

git commit -m “完成用户列表功能的开发”

git status

git branch

git push -u origin user

git branch

git checkout -b master

git merge user

git push

### 新建分支-创建rights子分支并推送到码云

git brance

git checkout -b rights

git brance

git push -u orgin rights

# 权限管理

## 权限列表

### 通过路由展示权限列表组件

在components下创建power文件Right.vue

在router.js中进行导入 挂载在/home下的子路由上

### 绘制面包屑导航和卡片视图

### 调用API获取权限列表的数据

 // 获取权限列表

    async getRightsList() {

      const { *data*: res } = await *this*.$http.get('rights/list')

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取权限列表失败！')

      }

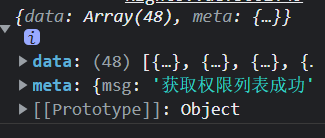
*this*.rightsList = res.data

      console.log(*this*.rightsList)

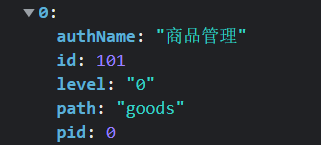
    }

打印res

**接口文档1.4.1**



### 渲染权限列表UI结构



找到Data中的Tag标签 使用模板作用域插槽

在element.js中按需导入 注册

 <el-table-column *label*="权限等级" *prop*="level">

          <template *slot-scope*="scope">

            <el-tag *v-if*="scope.row.level === '0'">一级</el-tag>

            <el-tag *type*="success" *v-else-if*="scope.row.level === '1'">二级</el-tag>

            <el-tag *type*="warning" *v-else*>三级</el-tag>

          </template>

        </el-table-column>

### 介绍 用户-角色-权限三者之间的关系

权限管理业务分析

通过权限管理模块控制不同的用户可以进行哪些操作，具体可以通过角色的方式进行控制，即每个用户分配一个特定的角色，角色包括不同的功能权限

## 角色列表

### 通过路由展示角色列表组件

在power文件创建Roles.vue

在router.js中导入 通过路由加载 挂载到/home子路由上

### 绘制基本布局结构并获取列表数据

放置面包屑导航

卡片视图

添加角色按钮

<!-- 添加角色按钮区域 -->

      <el-row>

        <el-col>

          <el-button *type*="primary">添加角色</el-button>

        </el-col>

      </el-row>

角色列表权限展示

 // 获取所有角色的列表

    async getRolesList() {

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

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取角色列表失败！')

      }

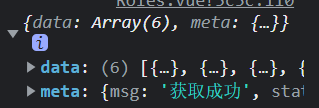
**接口文档1.5.1**

*this*.rolelist = res.data

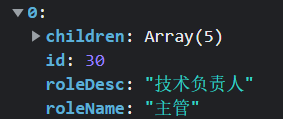
      console.log(*this*.rolelist)

    },

打印res



打印res.data



### 渲染角色列表数据

<el-table>

<el-table-column>

操作

作用域插槽

找到Basic中的Button按钮 带图标按钮

 <template *slot-scope*="scope">

            <el-button *size*="mini" *type*="primary" *icon*="el-icon-edit">编辑</el-button>

            <el-button *size*="mini" *type*="danger" *icon*="el-icon-delete">删除</el-button>

            <el-button *size*="mini" *type*="warning" *icon*="el-icon-setting" @*click*="showSetRightDialog(scope.row)">分配权限</el-button>

          </template>

展开列

<el-table-column *type*="expand">

### 说明角色列表需要完成的功能模块

分析角色下权限渲染的实现思路

<template slot-scope=”scope”>

<pre>{{scope.row}}</pre>

</template>

{

"id": 30,

"roleName": "主管",

"roleDesc": "技术负责人",

"children": [

{

"id": 101,

"authName": "商品管理",

"path": "goods",

"children": [

{

"id": 104,

"authName": "商品列表",

"path": "goods",

"children": [

{

"id": 116,

"authName": "商品修改",

"path": "goods"

}

]

},

### 通过第一层for循环渲染一级权限

### 美化一级权限的UI结构

### 通过第二层for循环渲染二级权限

### 通过三层for循环渲染三级权限

### 点击删除权限按钮弹出确认提示框

### 完成删除角色下指定权限的功能

 <template *slot-scope*="scope">

            <el-row :*class*="['bdbottom', i1 === 0 ? 'bdtop' : '', 'vcenter']" *v-for*="(item1, i1) in scope.row.children" :*key*="item1.id">

              <!-- 渲染一级权限 -->

              <el-col :*span*="5">

                <el-tag *closable* @*close*="removeRightById(scope.row, item1.id)">{{ item1.authName }}</el-tag>

                <i *class*="el-icon-caret-right"></i>

              </el-col>

              <!-- 渲染二级和三级权限 -->

              <el-col :*span*="19">

                <!-- 通过 for 循环 嵌套渲染二级权限 -->

                <el-row :*class*="[i2 === 0 ? '' : 'bdtop', 'vcenter']" *v-for*="(item2, i2) in item1.children" :*key*="item2.id">

                  <el-col :*span*="6">

                    <el-tag *type*="success" *closable* @*close*="removeRightById(scope.row, item2.id)">{{ item2.authName }}</el-tag>

                    <i *class*="el-icon-caret-right"></i>

                  </el-col>

                  <el-col :*span*="18">

                    <el-tag *type*="warning" *v-for*="item3 in item2.children" :*key*="item3.id" *closable* @*close*="removeRightById(scope.row, item3.id)">{{ item3.authName }}</el-tag>

                  </el-col>

                </el-row>

              </el-col>

            </el-row>

            <!-- <pre>

              {{scope.row}}

            </pre> -->

          </template>

在该组件style里面设置样式

*.el-tag* {

  margin: 7px;

}

*.bdtop* {

  border-top: 1px solid #eee;

}

*.bdbottom* {

  border-bottom: 1px solid #eee;

}

找到Basic中的icon图标

:span属性总共是24列

在全局global.css中设置样式

*#app* {

  height: 100%;

  margin: 0;

  padding: 0;

  min-width: 1366px;

}

解决左右拉动窗口 样式发生变化设置了min-width属性

*.vcenter* {

  display: flex;

  align-items: center;

}

纵向居中对齐

找到Data中tag标签中的可移除标签

添加closable属性 @close事件

// 根据Id删除对应的权限

    async removeRightById(*role*, *rightId*) {

      // 弹框提示用户是否要删除

      const confirmResult = await *this*.$confirm('此操作将永久删除该文件, 是否继续?', '提示', {

        confirmButtonText: '确定',

        cancelButtonText: '取消',

        type: 'warning'

      }).catch(*err* => *err*)

      if (confirmResult !== 'confirm') {

        return *this*.$message.info('取消了删除！')

      }

      const { *data*: res } = await *this*.$http.delete(`roles/${*role*.id}/rights/${*rightId*}`)

      if (res.meta.status !== 200) {

        return *this*.$message.error('删除权限失败！')

      }

**接口文档1.5.7**

      // this.getRolesList()

*role*.children = res.data

    },

找到notice中的MessageBox弹框的确认消息框

//this.getRolesList()会进行页面的完整渲染 删除之后会渲染页面关闭主管下的权限列表 所以不需要重新渲染

## 分配权限

1.5.2添加角色

1.5.3根据id查询角色

1.5.4编辑提交角色

1.5.5删除角色

### 弹出分配权限对话框并请求权限数据

在分配权限按钮添加事件@click

找到Others中的Dialog对话框

 <!-- 分配权限的对话框 -->

    <el-dialog *title*="分配权限" :*visible*.*sync*="setRightDialogVisible" *width*="50%" @*close*="setRightDialogClosed">

      <!-- 树形控件 -->

      <el-tree :*data*="rightslist" :*props*="treeProps" *show-checkbox* *node-key*="id" *default-expand-all* :*default-checked-keys*="defKeys" *ref*="treeRef"></el-tree>

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

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

        <el-button *type*="primary" @*click*="allotRights">确 定</el-button>

      </span>

    </el-dialog>

 // 展示分配权限的对话框

    async showSetRightDialog(*role*) {

*this*.roleId = *role*.id

      // 获取所有权限的数据

      const { *data*: res } = await *this*.$http.get('rights/tree')

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取权限数据失败！')

      }

**接口文档1.4.1用了两次 一次list 一次tree**

      // 把获取到的权限数据保存到 data 中

*this*.rightslist = res.data

      console.log(*this*.rightslist)

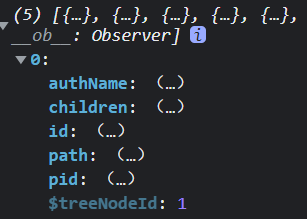
      // 递归获取三级节点的Id

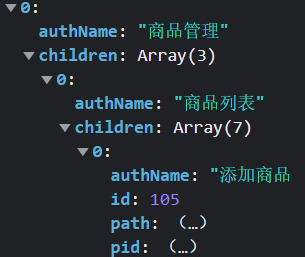
*this*.getLeafKeys(*role*, *this*.defKeys)

*this*.setRightDialogVisible = true

    },

打印res.data





### 初步配置并使用el-tree树形控件

找到Data中的Tree树形控件的基本用法

<el-tree :*data*="rightslist" :*props*="treeProps" *show-checkbox* *node-key*="id" *default-expand-all* :*default-checked-keys*="defKeys" *ref*="treeRef"></el-tree>

:props通过父子嵌套为整棵树绑定数据字段

 treeProps: {

        label: 'authName',

        children: 'children'

      },

在element.js按需导入 注册

### 优化树形控件的展示效果

找到Data中的Tree树形控件的Attributes

show-checkbox可以添加复选框

node-key 每个树节点用来作为唯一标识的属性，整棵树应该是唯一的 String

默认为折叠起来的

default-expand-all 是否默认展开所有节点 boolean false

### 分析已有权限默认勾选的实现思路

default-expanded-keys 默认展开的节点的 key 的数组 array

 // 默认选中的节点Id值数组

      defKeys: [],

将当前角色的三级权限的所有id都添加到defKeys数组中

### 加载当前角色已有的权限

// 通过递归的形式，获取角色下所有三级权限的id，并保存到 defKeys 数组中

    getLeafKeys(*node*, *arr*) {

      // 如果当前 node 节点不包含 children 属性，则是三级节点

      if (!*node*.children) {

        return *arr*.push(*node*.id)

      }

*node*.children.forEach(*item* => *this*.getLeafKeys(*item*, *arr*))

    },

 async showSetRightDialog(*role*) {

     // 递归获取三级节点的Id

*this*.getLeafKeys(*role*, *this*.defKeys)

}

### 在关闭对话框时重置defKeys数组

给el-dialog添加@*close*="setRightDialogClosed"

 // 监听分配权限对话框的关闭事件

    setRightDialogClosed() {

*this*.defKeys = []

    },

### 调用API完成分配权限的功能

找到Data中的Tree树形控件中的方法

getCheckedKeys 若节点可被选择（即 show-checkbox 为 true），则返回目前被选中的节点的 key 所组成的数组 (leafOnly) 接收一个 boolean 类型的参数，若为 true 则仅返回被选中的叶子节点的keys，默认值为 false

getHalfCheckedNodes若节点可被选择（即 show-checkbox 为 true），则返回目前半选中的节点所组成的数组

给确定按钮添加事件allotRights()

 // 点击为角色分配权限

    async allotRights() {

      const keys = [...*this*.$refs.treeRef.getCheckedKeys(), ...*this*.$refs.treeRef.getHalfCheckedKeys()]

      const idStr = keys.join(',')

      const { *data*: res } = await *this*.$http.post(`roles/${*this*.roleId}/rights`, { rids: idStr })

**接口文档1.5.6**

      if (res.meta.status !== 200) {

        return *this*.$message.error('分配权限失败！')

      }

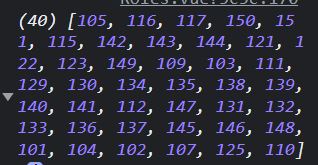
*this*.$message.success('分配权限成功！')

*this*.getRolesList()

*this*.setRightDialogVisible = false

    }

打印keys



 // 当前即将分配权限的角色id

      roleId: ''

// 展示分配权限的对话框

    async showSetRightDialog(*role*) {

*this*.roleId = *role*.id}

打开分配权限的对话框时，保存分配权限角色的id

### 渲染分配用户管理角色的对话框并请求角色列表数据

 <!-- 分配角色的对话框 -->

    <el-dialog *title*="分配角色" :*visible*.*sync*="setRoleDialogVisible" *width*="50%" @*close*="setRoleDialogClosed">

      <div>

        <p>当前的用户：{{ userInfo.username }}</p>

        <p>当前的角色：{{ userInfo.role\_name }}</p>

        <p>

          分配新角色：

          <el-select *v-model*="selectedRoleId" *placeholder*="请选择">

            <el-option *v-for*="item in rolesList" :*key*="item.id" :*label*="item.roleName" :*value*="item.id"> </el-option>

          </el-select>

        </p>

      </div>

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

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

        <el-button *type*="primary" @*click*="saveRoleInfo">确 定</el-button>

      </span>

    </el-dialog>

 // 展示分配角色的对话框

    async setRole(*userInfo*) {

      this.userInfo = *userInfo*

      // 在展示对话框之前，获取所有角色的列表

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

      if (res.meta.status !== 200) {

        return this.$message.error('获取角色列表失败！')

      }

**接口1.5.1用了两次 一次在权限管理角色列表展示 一次用户管理用户列表**

      this.rolesList = res.data

      this.setRoleDialogVisible = true

    },

### 渲染角色列表的select下拉菜单

找到Form中的Select选择器

:lable是你看到的值

:value是你选中的值

选中的值保存到v-mode中的属性值里面

在element.js中按需导入 注册

el-select el-option

### 完成分配角色的功能

在确定按钮中绑定事件

// 点击按钮，分配角色

    async saveRoleInfo() {

      if (!this.selectedRoleId) {

        return this.$message.error('请选择要分配的角色！')

      }

      const { data: res } = await *this*.$http.put(`users/${*this*.userInfo.id}/role`, {

        rid: *this*.selectedRoleId

      })

**接口1.3.7**

      if (res.meta.status !== 200) {

        return this.$message.error('更新角色失败！')

      }

      this.$message.success('更新角色成功！')

      this.getUserList()

      this.setRoleDialogVisible = false

    },

重置dialog @close进行重置

// 监听分配角色对话框的关闭事件

    setRoleDialogClosed() {

      this.selectedRoleId = ''

      this.userInfo = {}

    }

### 提交本地代码到Git仓库并推送到码云

git branch

git add .

git commit -m “完成了权限功能的开发”

git push

git checked master

git meger rights

git branch

git push

# 商品管理

## 商品分类

### 创建goods\_cate子分支并push到码云中

git branch

git checkout -b goods\_cate

git branch

git push -u origin goods\_cate

### 通过路由加载商品分类组件

创建components下的goods的Cate.vue组件

在router.js中导入 映射路径和组件

### 绘制商品分类组件的基本页面布局

面包屑导航

卡片视图区域

卡片添加分类按钮

表格

分页区域

### 调用API获取商品分裂列表数据

**接口文档1.6.1**

// 获取商品分类数据

    async getCateList() {

      const { *data*: res } = await *this*.$http.get('categories', {

        params: *this*.querInfo

      })

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取商品分类失败！')

      }

      console.log(res.data)

      // 把数据列表，赋值给 catelist

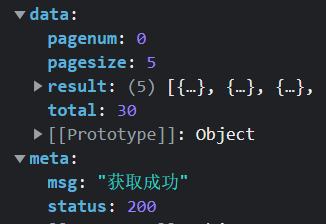
*this*.catelist = res.data.result

      // 为总数据条数赋值

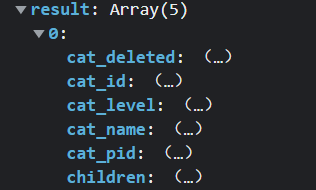
*this*.total = res.data.total

    },

打印res



打印res.data.result



### 初步使用vue-table-with-tree-grid

打开vue可视化面板找到依赖 安装依赖按钮 运行依赖 搜索vue-table-with-tree-grid 0.2.4查看详情 安装

找到main.js

import TreeTable from 'vue-table-with-tree-grid'

Vue.component('tree-table', TreeTable)

查看详情 找到example

columns 表格各列的配置 Array [] Columns Configs里面包含

属性labe 列标题名称 String

prop对应列内容的属性名

type列类型，可选值有”template”自定义模板

template 判断为”template”自定义模板时，对应的作用域插槽

selection-type是否为多选类型表格 Boolean false

expand-type 是否为展开行类型表格 Boolean false

show-index是否显示数据索引 Boolean false

index-text数据索引名称 String ‘序号’

border是否显示纵向边框

show-row-hover鼠标悬停时，是否高亮当前行

columns: [

        {

          label: '分类名称',

          prop: 'cat\_name'

        },

 <tree-table *class*="treeTable" :*data*="catelist" :*columns*="columns" :*selection-type*="false" :*expand-type*="false" *show-index* *index-text*="#" *border* :*show-row-hover*="false">

</tree-table>

### 使用自定义模板列渲染表格数据

**接口文档1.6.1**

 {

          label: '是否有效',

          // 表示，将当前列定义为模板列

          type: 'template',

          // 表示当前这一列使用模板名称

          template: 'isok'

        },

 <!-- 是否有效 -->

        <template *slot*="isok" *slot-scope*="scope">

          <i *class*="el-icon-success" *v-if*="scope.row.cat\_deleted === false" *style*="color: lightgreen;"></i>

          <i *class*="el-icon-error" *v-else* *style*="color: red;"></i>

        </template>

找到Basic中的Icon图标

### 渲染排序和操作对应的UI结构

 {

          label: '排序',

          // 表示，将当前列定义为模板列

          type: 'template',

          // 表示当前这一列使用模板名称

          template: 'order'

        },

      <!-- 排序 -->

        <template *slot*="order" *slot-scope*="scope">

          <el-tag *size*="mini" *v-if*="scope.row.cat\_level === 0">一级</el-tag>

          <el-tag *type*="success" *size*="mini" *v-else-if*="scope.row.cat\_level === 1">二级</el-tag>

          <el-tag *type*="warning" *size*="mini" *v-else*>三级</el-tag>

        </template>

找到Basic中的Button按钮找到搜索代码

### 实现分页功能

找到Data的Pagination分页找到附加功能(功能最完整的代码)

 <!-- 分页区域 -->

      <el-pagination

        @*size-change*="handleSizeChange"

        @*current-change*="handleCurrentChange"

        :*current-page*="querInfo.pagenum"

        :*page-sizes*="[3, 5, 10, 15]"

        :*page-size*="querInfo.pagesize"

*layout*="total, sizes, prev, pager, next, jumper"

        :*total*="total"

      >

      </el-pagination>

 // 监听 pagesize 改变

    handleSizeChange(*newSize*) {

*this*.querInfo.pagesize = *newSize*

*this*.getCateList()

    },

    // 监听 pagenum 改变

    handleCurrentChange(*newPage*) {

*this*.querInfo.pagenum = *newPage*

*this*.getCateList()

    },

*.treeTable* {

  margin-top: 15px;

}

### 渲染添加分类的对话框和表单

找到Others的Dialog对话框

<!-- 添加分类的对话框 -->

    <el-dialog *title*="添加分类" :*visible*.*sync*="addCateDialogVisible" *width*="50%" @*close*="addCateDialogClosed">

      <!-- 添加分类的表单 -->

      <el-form :*model*="addCateForm" :*rules*="addCateFormRules" *ref*="addCateFormRef" *label-width*="100px">

        <el-form-item *label*="分类名称：" *prop*="cat\_name">

          <el-input *v-model*="addCateForm.cat\_name"></el-input>

        </el-form-item>

        <el-form-item *label*="父级分类：">

          <!-- options 用来指定数据源 -->

          <!-- props 用来指定配置对象 -->

          <el-cascader *expand-trigger*="hover" :*options*="parentCateList" :*props*="cascaderProps" *v-model*="selectedKeys" @*change*="parentCateChanged" *clearable* *change-on-select*> </el-cascader>

        </el-form-item>

      </el-form>

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

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

        <el-button *type*="primary" @*click*="addCate">确 定</el-button>

      </span>

    </el-dialog>

 // 控制添加分类对话框的显示与隐藏

      addCateDialogVisible: false,

 // 点击按钮，展示添加分类的对话框

    showAddCateDialog() {

      // 先获取父级分类的数据列表

*this*.getParentCateList()

      // 再展示出对话框

*this*.addCateDialogVisible = true

    },

找到Form中的Form表单组件表单验证

  // 添加分类的表单数据对象

      addCateForm: {

        // 将要添加的分类的名称

        cat\_name: '',

        // 父级分类的Id

        cat\_pid: 0,

        // 分类的等级，默认要添加的是1级分类

        cat\_level: 0

      },

**接口文档1.6.2**

### 获取父级分类数据列表

**接口文档1.6.1**

 // 获取父级分类的数据列表

    async getParentCateList() {

      const { *data*: res } = await *this*.$http.get('categories', {

        params: { type: 2 }

      })

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取父级分类数据失败！')

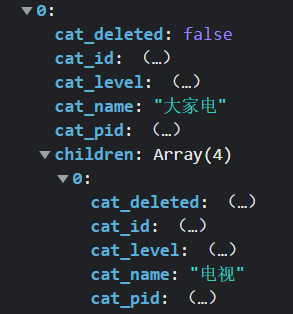
      }

      console.log(res.data)

*this*.parentCateList = res.data

    },

打印res.data



### 渲染级联选择器

找到Form中的Cascader级联选择器用hover选项

在element.js中导入 注册

 <el-form-item *label*="父级分类：">

          <!-- options 用来指定数据源 -->

          <!-- props 用来指定配置对象 -->

          <el-cascader *expand-trigger*="hover" :*options*="parentCateList" :*props*="cascaderProps" *v-model*="selectedKeys" @*change*="parentCateChanged" *clearable* *change-on-select*> </el-cascader>

        </el-form-item>

 // 指定级联选择器的配置对象

      cascaderProps: {

        value: 'cat\_id',

        label: 'cat\_name',

        children: 'children'

      },

 // 选中的父级分类的Id数组

      selectedKeys: []

 // 选择项发生变化触发这个函数

    parentCateChanged() {

      console.log(*this*.selectedKeys)

      // 如果 selectedKeys 数组中的 length 大于0，证明选中的父级分类

      // 反之，就说明没有选中任何父级分类

      if (*this*.selectedKeys.length > 0) {

        // 父级分类的Id

*this*.addCateForm.cat\_pid = *this*.selectedKeys[*this*.selectedKeys.length - 1]

        // 为当前分类的等级赋值

*this*.addCateForm.cat\_level = *this*.selectedKeys.length

      } else {

        // 父级分类的Id

*this*.addCateForm.cat\_pid = 0

        // 为当前分类的等级赋值

*this*.addCateForm.cat\_level = 0

      }

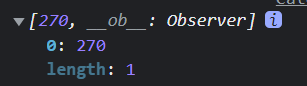
    },

*.el-cascader* {

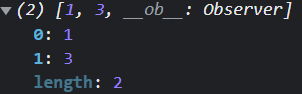
  width: 100%;

}

打印selectedKeys 添加到二级菜单里面



添加到三级菜单里面



Clearable 是否支持清空选项 boolean false

Change-on-select用来点击一级选项 是否允许选择任意一级的选项 boolean false

也可以使用checkStrictly 是否严格的遵守父子节点不互相关联 boolean false

### 根据父分类的辩护处理表单中的数据

// 点击按钮，添加新的分类

    addCate() {

*this*.$refs.addCateFormRef.validate(async *valid* => {

        if (!*valid*) return

        const { *data*: res } = await *this*.$http.post('categories', *this*.addCateForm)

        if (res.meta.status !== 201) {

          return *this*.$message.error('添加分类失败！')

        }

*this*.$message.success('添加分类成功！')

*this*.getCateList()

*this*.addCateDialogVisible = false

      })

    },

### 在对话框的close事件中重置表单数据

  // 监听对话框的关闭事件，重置表单数据

    addCateDialogClosed() {

*this*.$refs.addCateFormRef.resetFields()

*this*.selectedKeys = []

*this*.addCateForm.cat\_level = 0

*this*.addCateForm.cat\_pid = 0

    }

### 完成添加分类的操作

**接口文档1.6.2**

### 将goods\_cate分支的代码提交到git仓库中

git branch

git status

git add .

git status

git commit -m “完成了分类功能的开发”

git status

git push

git branch

git checkout master

git merge goods\_cate

git push

### 创建goods\_params分支

清空cls

git branch

git checkout -b goods\_params

git branch

git push -u orgin goods\_params

## 分类参数

### 介绍分类参数功能的作用

参数管理概述

商品参数用于显示商品的固定的特征信息，可以通过电商平台详情页面直观的看到。

分为动态参数 静态属性

### 通过路由加载分类参数组件页面

在goods下面创建Params.vue组件

在router.js中进行导入 在home子路由里面进行挂载

### 渲染分类参数页面的基本UI结构

面包屑导航

卡片视图区域

找到Notice中的Alert警告基本用法

在element.js中进行导入 注册

:closable=”false”

警告区域

 <el-alert *show-icon* *title*="注意：只允许为第三级分类设置相关参数！" *type*="warning" :*closable*="false"></el-alert>

选择商品分类区域

### 调用API获取商品分类列表数据

**接口文档1.6.1**

 // 获取所有的商品分类列表

    async getCateList() {

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

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取商品分类失败！')

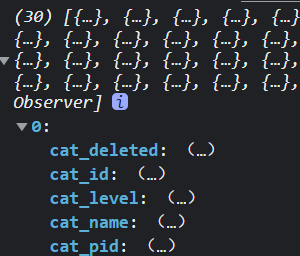
      }

*this*.catelist = res.data

      console.log(*this*.catelist)

    },

打印catelist



### 渲染商品分类的级联选择框

找到Form中的Cascader 级联选择器中hover代码

    // 级联选择框的配置对象

      cateProps: {

        value: 'cat\_id',

        label: 'cat\_name',

        children: 'children'

      },

### 控制级联选择框的选中范围

// 获取参数的列表数据

    async getParamsData() {

      // 证明选中的不是三级分类

      if (*this*.selectedCateKeys.length !== 3) {

*this*.selectedCateKeys = []

*this*.manyTableData = []

*this*.onlyTableData = []

        return

      }

      // 证明选中的是三级分类

      console.log(*this*.selectedCateKeys)

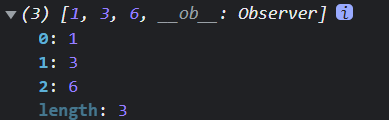
      // 根据所选分类的Id，和当前所处的面板，获取对应的参数

      const { data: res } = await *this*.$http.get(`categories/${*this*.cateId}/attributes`, {

        params: { sel: *this*.activeName }

      })

打印selectedCateKeys



      if (res.meta.status !== 200) {

        return *this*.$message.error('获取参数列表失败！')

      }

      res.data.forEach(*item* => {

*item*.attr\_vals = *item*.attr\_vals ? *item*.attr\_vals.split(' ') : []

        // 控制文本框的显示与隐藏

*item*.inputVisible = false

        // 文本框中输入的值

*item*.inputValue = ''

      })

      console.log(res.data)

      if (*this*.activeName === 'many') {

*this*.manyTableData = res.data

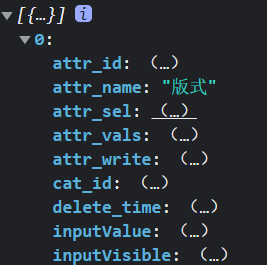
      } else {

*this*.onlyTableData = res.data

      }

    },

打印res.data



### 渲染分类参数的Tabs页签

找到navigation中的Tabs标签页

在element.js中导入 注册

 <!-- tab 页签区域 -->

      <el-tabs *v-model*="activeName" @*tab-click*="handleTabClick">

<!-- 添加动态参数的面板 -->

        <el-tab-pane *label*="动态参数" *name*="many">

   // 被激活的页签的名称

      activeName: 'many',

// tab 页签点击事件的处理函数

    handleTabClick() {

      console.log(*this*.activeName)

*this*.getParamsData()

    },

打印activeName



### 渲染添加参数按钮并控制按钮的禁用状态

 <!-- 添加参数的按钮 -->

          <el-button *type*="primary" *size*="mini" :*disabled*="isBtnDisabled" @*click*="addDialogVisible = true">添加参数</el-button>

在computed中

 // 如果按钮需要被禁用，则返回true，否则返回false

    isBtnDisabled() {

      if (*this*.selectedCateKeys.length !== 3) {

        return true

      }

      return false

    },

### 获取参数列表数据

接口文档1.7.1

 // 当前选中的三级分类的Id

    cateId() {

      if (*this*.selectedCateKeys.length === 3) {

        return *this*.selectedCateKeys[2]

      }

      return null

    },

// 动态计算标题的文本

    titleText() {

      if (*this*.activeName === 'many') {

        return '动态参数'

      }

      return '静态属性'

    }

### 切换Tabs面板后重新获取参数列表数据

分别在handleChange和handleTabClick中调用这个函数getParamsData

### 将获取到的参数数据挂载到不同的数据源上

### 渲染动态参数和静态属性的Table表格

找到Basic中的Button按钮

   <!-- 展开行 -->

            <el-table-column *type*="expand">

<!-- 索引列 -->

            <el-table-column *type*="index"></el-table-column>

            <el-table-column *label*="参数名称" *prop*="attr\_name"></el-table-column>

            <el-table-column *label*="操作">

              <template *slot-scope*="scope">

                <el-button *size*="mini" *type*="primary" *icon*="el-icon-edit" @*click*="showEditDialog(scope.row.attr\_id)">编辑</el-button>

                <el-button *size*="mini" *type*="danger" *icon*="el-icon-delete" @*click*="removeParams(scope.row.attr\_id)">删除</el-button>

              </template>

            </el-table-column>

### 渲染添加参数的对话框

找到Others中的Dialog对话框

// 动态计算标题的文本

    titleText() {

      if (*this*.activeName === 'many') {

        return '动态参数'

      }

      return '静态属性'

    }

 <!-- 添加参数的对话框 -->

    <el-dialog :*title*="'添加' + titleText" :*visible*.*sync*="addDialogVisible" *width*="50%" @*close*="addDialogClosed">

找到Form中的Form表单的表单验证

<!-- 添加参数的对话框 -->

      <el-form :*model*="addForm" :*rules*="addFormRules" *ref*="addFormRef" *label-width*="100px">

        <el-form-item :*label*="titleText" *prop*="attr\_name">

          <el-input *v-model*="addForm.attr\_name"></el-input>

        </el-form-item>

      </el-form>

 // 添加表单的验证规则对象

      addFormRules: {

        attr\_name: [{ required: true, message: '请输入参数名称', trigger: 'blur' }]

      },

 // 监听添加对话框的关闭事件

    addDialogClosed() {

*this*.$refs.addFormRef.resetFields()

    },

### 完成动态参数和静态属性的添加操作

**接口文档1.7.2**

// 点击按钮，添加参数

    addParams() {

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

        if (!*valid*) return

        const { data: res } = await *this*.$http.post(`categories/${*this*.cateId}/attributes`, {

          attr\_name: *this*.addForm.attr\_name,

          attr\_sel: *this*.activeName

        })

        if (res.meta.status !== 201) {

          return *this*.$message.error('添加参数失败！')

        }

*this*.$message.success('添加参数成功！')

*this*.addDialogVisible = false

*this*.getParamsData()

      })

    },

### 渲染修改参数的对话框

<!-- 修改参数的对话框 -->

    <el-dialog :*title*="'修改' + titleText" :*visible*.*sync*="editDialogVisible" *width*="50%" @*close*="editDialogClosed">

      <!-- 添加参数的对话框 -->

      <el-form :*model*="editForm" :*rules*="editFormRules" *ref*="editFormRef" *label-width*="100px">

        <el-form-item :*label*="titleText" *prop*="attr\_name">

          <el-input *v-model*="editForm.attr\_name"></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" @*click*="editParams">确 定</el-button>

      </span>

    </el-dialog>

### 完成修改参数的操作

**接口文档1.7.4**

// 点击按钮，展示修改的对话框

    async showEditDialog(*attrId*) {

      // 查询当前参数的信息

      const { data: res } = await *this*.$http.get(`categories/${*this*.cateId}/attributes/${*attrId*}`, {

        params: { attr\_sel: *this*.activeName }

      })

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取参数信息失败！')

      }

*this*.editForm = res.data

*this*.editDialogVisible = true

    },

 // 重置修改的表单

    editDialogClosed() {

*this*.$refs.editFormRef.resetFields()

    },

**接口文档1.7.5**

// 点击按钮，修改参数信息

    editParams() {

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

        if (!*valid*) return

        const { data: res } = await *this*.$http.put(`categories/${*this*.cateId}/attributes/${*this*.editForm.attr\_id}`, { attr\_name: *this*.editForm.attr\_name, attr\_sel: *this*.activeName })

        if (res.meta.status !== 200) {

          return *this*.$message.error('修改参数失败！')

        }

*this*.$message.success('修改参数成功！')

*this*.getParamsData()

*this*.editDialogVisible = false

      })

    },

### 完成删除参数的业务逻辑

**接口文档1.7.3**

   // 根据Id删除对应的参数项

    async removeParams(*attrId*) {

      const confirmResult = await *this*.$confirm('此操作将永久删除该参数, 是否继续?', '提示', {

        confirmButtonText: '确定',

        cancelButtonText: '取消',

        type: 'warning'

      }).catch(*err* => *err*)

      // 用户取消了删除的操作

      if (confirmResult !== 'confirm') {

        return *this*.$message.info('已取消删除！')

      }

      // 删除的业务逻辑

      const { data: res } = await *this*.$http.delete(`categories/${*this*.cateId}/attributes/${*attrId*}`)

      if (res.meta.status !== 200) {

        return *this*.$message.error('删除参数失败！')

      }

*this*.$message.success('删除参数成功！')

*this*.getParamsData()

    },

找到Notice中的MessageBox弹框

### 渲染参数下的可选项

    res.data.forEach(*item* => {

*item*.attr\_vals = *item*.attr\_vals ? *item*.attr\_vals.split(' ') : []

        // 控制文本框的显示与隐藏

*item*.inputVisible = false

        // 文本框中输入的值

*item*.inputValue = ''

      })

 <!-- 展开行 -->

            <el-table-column *type*="expand">

              <template *slot-scope*="scope">

                <!-- 循环渲染Tag标签 -->

                <el-tag *v-for*="(item, i) in scope.row.attr\_vals" :*key*="i" *closable* @*close*="handleClose(i, scope.row)">{{ item }}</el-tag>

                <!-- 输入的文本框 -->

                <el-input

*class*="input-new-tag"

*v-if*="scope.row.inputVisible"

*v-model*="scope.row.inputValue"

*ref*="saveTagInput"

*size*="small"

                  @*keyup*.enter.*native*="handleInputConfirm(scope.row)"

                  @*blur*="handleInputConfirm(scope.row)"

                >

                </el-input>

                <!-- 添加按钮 -->

                <el-button *v-else* *class*="button-new-tag" *size*="small" @*click*="showInput(scope.row)">+ New Tag</el-button>

              </template>

            </el-table-column>

### 解决attr\_vals为空字符串时候的小问题

### 控制按钮与文本框的切换显示

找到Data中的Tag标签的动态编辑标签

<!-- 输入的文本框 -->

                <el-input

*class*="input-new-tag"

*v-if*="scope.row.inputVisible"

*v-model*="scope.row.inputValue"

*ref*="saveTagInput"

*size*="small"

                  @*keyup*.enter.*native*="handleInputConfirm(scope.row)"

                  @*blur*="handleInputConfirm(scope.row)"

                >

                </el-input>

                <!-- 添加按钮 -->

                <el-button *v-else* *class*="button-new-tag" *size*="small" @*click*="showInput(scope.row)">+ New Tag</el-button>

### 让文本框自动获得焦点

   // 点击按钮，展示文本输入框

    showInput(*row*) {

*row*.inputVisible = true

      // 让文本框自动获得焦点

      // $nextTick 方法的作用，就是当页面上元素被重新渲染之后，才会指定回调函数中的代码

*this*.$nextTick(*\_* => {

*this*.$refs.saveTagInput.$refs.input.focus()

      })

    },

### 实现文本框与按钮的切换显示

  // 文本框失去焦点，或摁下了 Enter 都会触发

    async handleInputConfirm(*row*) {

      if (*row*.inputValue.trim().length === 0) {

*row*.inputValue = ''

*row*.inputVisible = false

        return

      }

      // 如果没有return，则证明输入的内容，需要做后续处理

*row*.attr\_vals.push(*row*.inputValue.trim())

*row*.inputValue = ''

*row*.inputVisible = false

      // 需要发起请求，保存这次操作

*this*.saveAttrVals(*row*)

    },

### 完成参数可选项的添加操作

**接口文档1.7.5**

// 将对 attr\_vals 的操作，保存到数据库

    async saveAttrVals(*row*) {

      // 需要发起请求，保存这次操作

      const { data: res } = await *this*.$http.put(`categories/${*this*.cateId}/attributes/${*row*.attr\_id}`, {

        attr\_name: *row*.attr\_name,

        attr\_sel: *row*.attr\_sel,

        attr\_vals: *row*.attr\_vals.join(' ')

      })

      if (res.meta.status !== 200) {

        return *this*.$message.error('修改参数项失败！')

      }

*this*.$message.success('修改参数项成功！')

    },

### 删除参数下的可选项

   // 删除对应的参数可选项

    handleClose(*i*, *row*) {

*row*.attr\_vals.splice(*i*, 1)

*this*.saveAttrVals(*row*)

    }

### 清空表格数据

async getParamsData() {

      // 证明选中的不是三级分类

      if (*this*.selectedCateKeys.length !== 3) {

*this*.selectedCateKeys = []

*this*.manyTableData = []

*this*.onlyTableData = []

        return

      }

}

### 完成静态属性表格中的展开行效果

### 将本地goods\_params分支提交并推送到码云

git branch

git status

git add .

git status

git commit -m “完成了分类参数的开发”

git status

git push

git checkout master

git branch

git merge goods\_params

git push

### 创建goods\_list子分支并推送到码云

git checkout -b goods\_list

git branch

git push -u origin goods\_list

## 商品列表

### 通过路由加载商品列表组件

在goods下创建List.vue

在router.js中导入

面包屑导航区域

卡片视图区域

在form中找到input输入框找到有搜索功能的输入框

### 获取商品列表数据

**接口文档1.8.1**

 // 根据分页获取对应的商品列表

    async getGoodsList() {

      const { data: res } = await *this*.$http.get('goods', {

        params: *this*.queryInfo

      })

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取商品列表失败！')

      }

*this*.$message.success('获取商品列表成功！')

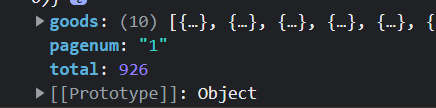
      // console.log(res.data)

*this*.goodslist = res.data.goods

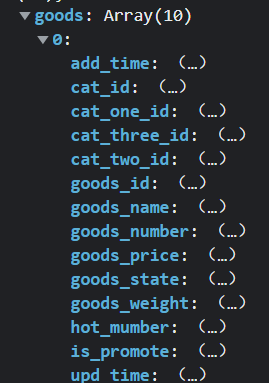
*this*.total = res.data.total

    },

打印res.data



打印res.data.goods



### 渲染商品表格数据

table表格区域

### 自定义格式化时间的全局过滤器

在main.js中

Vue.filter('dateFormat', function(*originVal*) {

  const dt = **new** *Date*(*originVal*)

  const y = dt.getFullYear()

  const m = (dt.getMonth() + 1 + '').padStart(2, '0')

  const d = (dt.getDate() + '').padStart(2, '0')

  const hh = (dt.getHours() + '').padStart(2, '0')

  const mm = (dt.getMinutes() + '').padStart(2, '0')

  const ss = (dt.getSeconds() + '').padStart(2, '0')

  return `${y}-${m}-${d} ${hh}:${mm}:${ss}`

})

 <el-table-column *label*="创建时间" *prop*="add\_time" *width*="140px">

          <template *slot-scope*="scope">

            {{ scope.row.add\_time | dateFormat }}

          </template>

        </el-table-column>

### 实现商品列表的分页功能

找到Data中的Pagination分页的完整功能

分页区域

### 实现搜索与清空的功能

卡片视图区域

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

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

          </el-input>

### 根据id删除商品数据

  async removeById(*id*) {

      const confirmResult = await *this*.$confirm('此操作将永久删除该商品, 是否继续?', '提示', {

        confirmButtonText: '确定',

        cancelButtonText: '取消',

        type: 'warning'

      }).catch(*err* => *err*)

      if (confirmResult !== 'confirm') {

        return *this*.$message.info('已经取消删除！')

      }

      const { data: res } = await *this*.$http.delete(`goods/${*id*}`)

      if (res.meta.status !== 200) {

        return *this*.$message.error('删除失败！')

      }

*this*.$message.success('删除成功！')

*this*.getGoodsList()

    },

找到Notice中的MessageBox弹框的确认消息

**接口文档1.8.5**

## 商品添加

### 通过编程式导航跳转到商品添加页面

 goAddpage() {

*this*.$router.push('/goods/add')

    }

在goods文件中添加Add.vue

初始化基本结构

在router.js中导入挂载

### 渲染添加页面的基本结构

面包屑导航区域

卡片视图

提示区域

找到Notice中的Alert警告的文字居中灰色的

<!-- 提示区域 -->

      <el-alert *title*="添加商品信息" *type*="info" *center* *show-icon* :*closable*="false">

      </el-alert>

找到Navigation中的Steps步骤条的含状态步骤条

在element.js中导入注册

   <!-- 步骤条区域 -->

      <el-steps :*space*="200" :*active*="activeIndex - 0" *finish-status*="success" *align-center*>

        <el-step *title*="基本信息"></el-step>

        <el-step *title*="商品参数"></el-step>

        <el-step *title*="商品属性"></el-step>

        <el-step *title*="商品图片"></el-step>

        <el-step *title*="商品内容"></el-step>

        <el-step *title*="完成"></el-step>

      </el-steps>

美化步骤条组件

*.el-steps* {

  margin: 15px 0;

}

*.el-step\_\_title* {

  font-size: 13px;

}

### 渲染tab栏区域

找到Navigation的Tabs标签页的left

<!-- tab栏区域 -->

### 实现步骤条和tab栏的数据联动效果

<el-tabs *v-model*="activeIndex" :*tab-position*="'left'" :*before-leave*="beforeTabLeave" @*tab-click*="tabClicked">

          <el-tab-pane *label*="基本信息" *name*="0">

            <el-form-item *label*="商品名称" *prop*="goods\_name">

              <el-input *v-model*="addForm.goods\_name"></el-input>

            </el-form-item>

### 分析表单的组成部分

找到Form中的Form表单的表单验证

label-position 表单域标签的位置，如果值为 left 或者 right 时，则需要设置 label-width string right/left/top 默认right

<el-form :*model*="addForm" :*rules*="addFormRules" *ref*="addFormRef" *label-width*="100px" *label-position*="top">

### 绘制基本信息面板的UI结构

**接口文档1.8.2**

### 获取商品分类数据

**接口文档1.6.1**

 // 获取所有商品分类数据

    async getCateList() {

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

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取商品分类数据失败！')

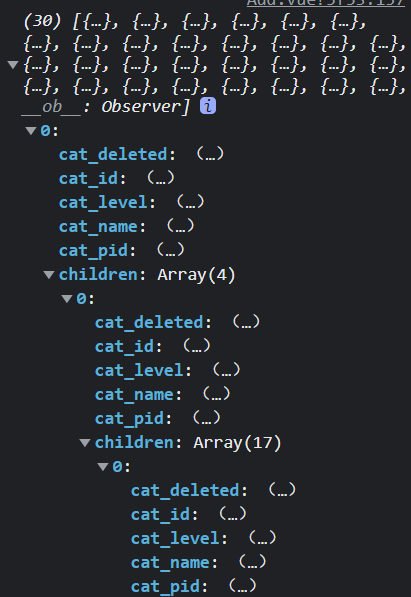
      }

*this*.catelist = res.data

      console.log(*this*.catelist)

    },

打印this.catelist



### 绘制商品分类的级联选择器

找到Form中的Cascader的hover

 <el-form-item *label*="商品分类" *prop*="goods\_cat">

              <el-cascader *expand-trigger*="hover" :*options*="catelist" :*props*="cateProps" *v-model*="addForm.goods\_cat" @*change*="handleChange"> </el-cascader>

            </el-form-item>

 cateProps: {

        label: 'cat\_name',

        value: 'cat\_id',

        children: 'children'

      },

**接口文档1.6.1**

**接口文档1.8.2**

// 级联选择器选中项变化，会触发这个函数

    handleChange() {

      console.log(*this*.addForm.goods\_cat)

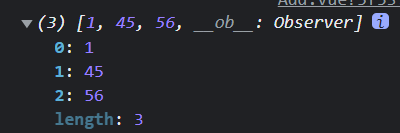
      if (*this*.addForm.goods\_cat.length !== 3) {

*this*.addForm.goods\_cat = []

      }

    },

打印this.addForm.goods\_cat



### 只允许选中三级商品分类

### 阻止页签切换

找到Navigation中的Tabs标签页

before-leave 切换标签之前的钩子，若返回 false 或者返回 Promise 且被 reject，则阻止切换。 Function(activeName, oldActiveName)

 beforeTabLeave(*activeName*, *oldActiveName*) {

      // console.log('即将离开的标签页名字是：' + oldActiveName)

      // console.log('即将进入的标签页名字是：' + activeName)

      // return false

      if (*oldActiveName* === '0' && *this*.addForm.goods\_cat.length !== 3) {

*this*.$message.error('请先选择商品分类！')

        return false

      }

    },

### 获取动态参数列表数据

**接口文档1.7.1**

找到Navigation中的Tabs标签页的Tabs Events

tab-click tab 被选中时触发 被选中的标签 tab 实例

  async tabClicked() {

      // console.log(this.activeIndex)

      // 证明访问的是动态参数面板

      if (*this*.activeIndex === '1') {

        const { data: res } = await *this*.$http.get(`categories/${*this*.cateId}/attributes`, {

          params: { sel: 'many' }

        })

        if (res.meta.status !== 200) {

          return *this*.$message.error('获取动态参数列表失败！')

        }

        console.log(res.data)

        res.data.forEach(*item* => {

*item*.attr\_vals = *item*.attr\_vals.length === 0 ? [] : *item*.attr\_vals.split(' ')

        })

*this*.manyTableData = res.data

      } else if (*this*.activeIndex === '2') {

        const { data: res } = await *this*.$http.get(`categories/${*this*.cateId}/attributes`, {

          params: { sel: 'only' }

        })

        if (res.meta.status !== 200) {

          return *this*.$message.error('获取静态属性失败！')

        }

        console.log(res.data)

*this*.onlyTableData = res.data

      }

    },

  computed: {

    cateId() {

      if (*this*.addForm.goods\_cat.length === 3) {

        return *this*.addForm.goods\_cat[2]

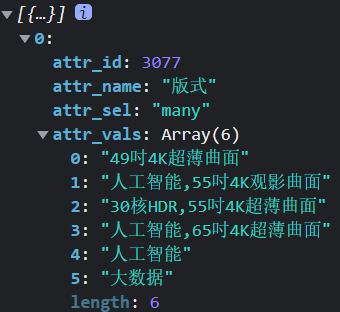
      }

      return null

    }

  }

打印res.data



### 绘制商品参数面板中的复选框组

找到Form中的Checkbox多选框的多选框组

在element.js中导入注册CheckboxGroup,Checkbox

 <!-- 复选框组 -->

              <el-checkbox-group *v-model*="item.attr\_vals">

                <el-checkbox :*label*="cb" *v-for*="(cb, i) in item.attr\_vals" :*key*="i" *border*></el-checkbox>

              </el-checkbox-group>

### 优化复选框的样式

*.el-checkbox* {

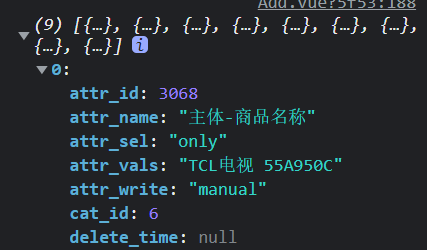
  margin: 0 10px 0 0 !important;

}

### 获取静态属性列表数据

**接口文档1.7.1**

打印res.data



### 渲染商品属性面板的UI结构

### 初步使用upload上传组件

找到Form中的Upload上传，图片列表缩略图

 <el-tab-pane *label*="商品图片" *name*="3">

            <!-- action 表示图片要上传到的后台API地址 -->

            <el-upload :*action*="uploadURL" :*on-preview*="handlePreview" :*on-remove*="handleRemove" *list-type*="picture" :*headers*="headerObj" :*on-success*="handleSuccess">

              <el-button *size*="small" *type*="primary">点击上传</el-button>

            </el-upload>

          </el-tab-pane>

**接口文档1.9图片上传**

 // 上传图片的URL地址

      uploadURL: 'http://127.0.0.1:8888/api/private/v1/upload',

  // 处理图片预览效果

    handlePreview(*file*) {

      console.log(*file*)

*this*.previewPath = *file*.response.data.url

*this*.previewVisible = true

    },

 // 处理移除图片的操作

    handleRemove(*file*) {

      // console.log(file)

      // 1. 获取将要删除的图片的临时路径

      const filePath = *file*.response.data.tmp\_path

      // 2. 从 pics 数组中，找到这个图片对应的索引值

      const i = *this*.addForm.pics.findIndex(*x* => *x*.pic === filePath)

      // 3. 调用数组的 splice 方法，把图片信息对象，从 pics 数组中移除

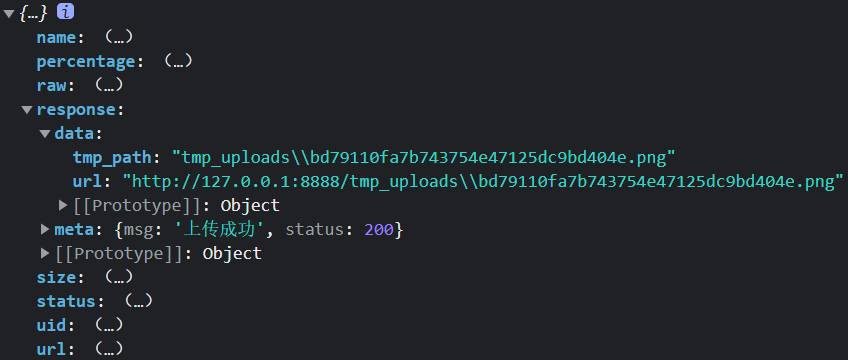
*this*.addForm.pics.splice(i, 1)

      console.log(*this*.addForm)

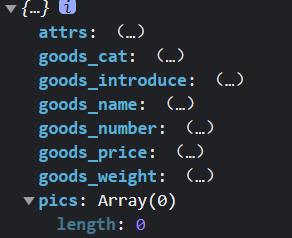
    },

在element.js中进行导入注册Upload

打印file



打印this.addForm



### 手动为upload组件绑定headers请求头

headers 设置上传的请求头部 object

 // 图片上传组件的headers请求头对象

      headerObj: {

        Authorization: window.sessionStorage.getItem('token')

      },

### 监听upload组件的on-success事件

**接口文档1.8.2**



on-success 文件上传成功时的钩子 function(response, file, fileList)

 // 监听图片上传成功的事件

    handleSuccess(*response*) {

      console.log(*response*)

      // 1. 拼接得到一个图片信息对象

      const picInfo = { pic: *response*.data.tmp\_path }

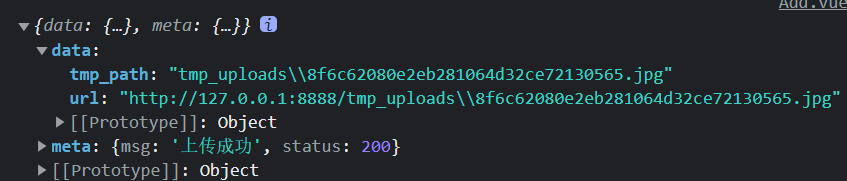
      // 2. 将图片信息对象，push 到pics数组中

*this*.addForm.pics.push(picInfo)

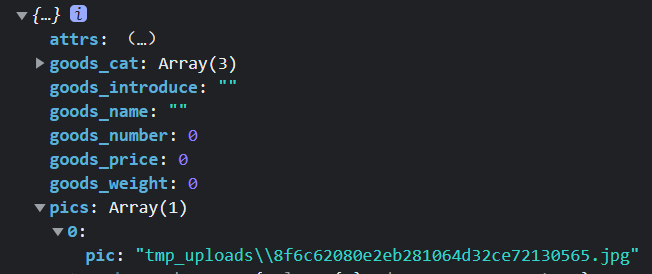
      console.log(*this*.addForm)

    },

打印response



打印this.addForm



### 实现图片的预览效果

找到Form中的Upload中上传的

on-preview 点击文件列表中已上传的文件时的钩子 function(file)

  // 处理图片预览效果

    handlePreview(*file*) {

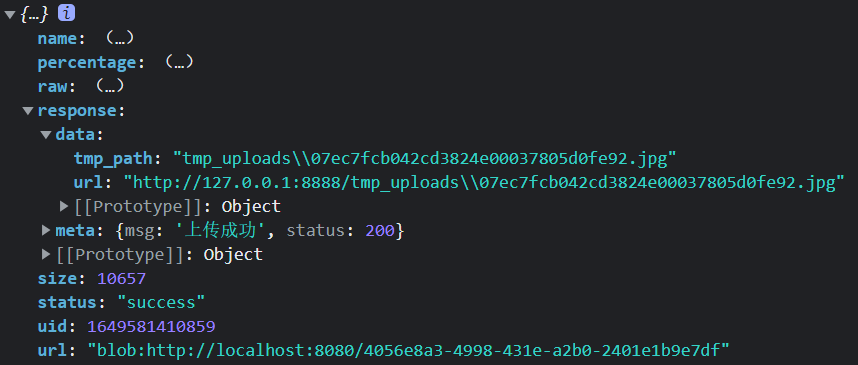
      console.log(*file*)

*this*.previewPath = *file*.response.data.url

*this*.previewVisible = true

    },

打印file



找到Others中的Dialog对话框

 <!-- 图片预览 -->

    <el-dialog *title*="图片预览" :*visible*.*sync*="previewVisible" *width*="50%">

      <img :*src*="previewPath" *alt*="" *class*="previewImg" />

    </el-dialog>

*.previewImg* {

  width: 100%;

}

### 安装并配置vue-quill-editor

打开可视化工具 依赖 右上角安装依赖按钮 运行依赖 vue-quill-editor

查看详情 Mount mount with global

找到main.js

// 导入富文本编辑器

import VueQuillEditor from 'vue-quill-editor'

// require styles 导入富文本编辑器对应的样式

import 'quill/dist/quill.core.css'

import 'quill/dist/quill.snow.css'

import 'quill/dist/quill.bubble.css'

// 将富文本编辑器，注册为全局可用的组件

Vue.use(VueQuillEditor)

在Add.vue中

**接口文档1.8.2**

   <el-tab-pane *label*="商品内容" *name*="4">

            <!-- 富文本编辑器组件 -->

            <quill-editor *v-model*="addForm.goods\_introduce"></quill-editor>

            <!-- 添加商品的按钮 -->

            <el-button *type*="primary" *class*="btnAdd" @*click*="add">添加商品</el-button>

          </el-tab-pane>

global.css中

*.ql-editor* {

  min-height: 300px;

}

*.btnAdd* {

  margin-top: 15px;

}

### 实现表单数据的预验证

    // 添加商品

    add() {

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

        if (!*valid*) {

          return *this*.$message.error('请填写必要的表单项！')

        }

        // 执行添加的业务逻辑

        // lodash   cloneDeep(obj)

        const form = \_.cloneDeep(*this*.addForm)

        form.goods\_cat = form.goods\_cat.join(',')

        // 处理动态参数

*this*.manyTableData.forEach(*item* => {

          const newInfo = {

            attr\_id: *item*.attr\_id,

            attr\_value: *item*.attr\_vals.join(' ')

          }

*this*.addForm.attrs.push(newInfo)

        })

        // 处理静态属性

*this*.onlyTableData.forEach(*item* => {

          const newInfo = { attr\_id: *item*.attr\_id, attr\_value: *item*.attr\_vals }

*this*.addForm.attrs.push(newInfo)

        })

        form.attrs = *this*.addForm.attrs

        console.log(form)

        // 发起请求添加商品

        // 商品的名称，必须是唯一的

        const { data: res } = await *this*.$http.post('goods', form)

        if (res.meta.status !== 201) {

          return *this*.$message.error('添加商品失败！')

        }

*this*.$message.success('添加商品成功！')

*this*.$router.push('/goods')

      })

    }

### 把goods\_cat从数组转换为字符串

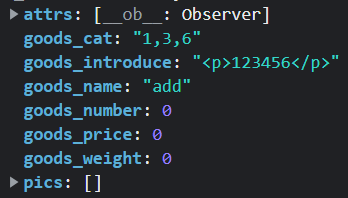
**接口文档1.8.2**

可视化工具 依赖 安装依赖 运行依赖 lodash4.17.1 安装

Add.vue中导入

import \_ from 'lodash'

打印form



### 添加attrs数组

goods\_cat就转换成字符串了

### 完成商品添加操作

**接口文档1.8.2**

### 将goods\_list分支的代码合并并且推送到码云中

git status

git branch

git add .

git commit -m “完成商品功能开发”

git status

git push

git checkout master

git branch

git merge goods\_list

git push

### 创建order子分支

git branch

git checkout -b order

git branch

git push -u origin order

# 订单管理

## 订单列表

### 通过路由渲染订单列表页面

创建order/Order.vue

在router.js中导入挂载组件

面包屑导航区域

卡片视图区域

找到input输入框的复合型输入框

### 根据分页获取订单数据列表

**接口文档1.10.1**

 async getOrderList() {

      const { data: res } = await *this*.$http.get('orders', {

        params: *this*.queryInfo

      })

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取订单列表失败！')

      }

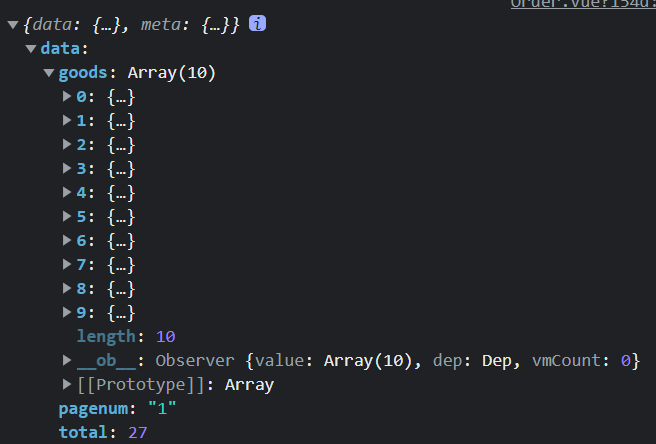
      console.log(res)

*this*.total = res.data.total

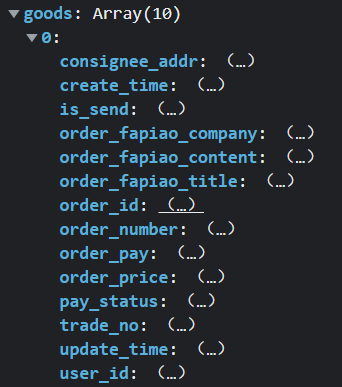
*this*.orderlist = res.data.goods

    },

打印res



打印orderlist



### 渲染订单table表格

找到Basic的Button按钮的图标按钮

    <el-table-column *label*="下单时间" *prop*="create\_time">

          <template *slot-scope*="scope">

            {{ scope.row.create\_time | dateFormat }}

          </template>

        </el-table-column>

 <el-table-column *label*="是否付款" *prop*="pay\_status">

          <template *slot-scope*="scope">

            <el-tag *type*="success" *v-if*="scope.row.pay\_status === '1'">已付款</el-tag>

            <el-tag *type*="danger" *v-else*>未付款</el-tag>

          </template>

        </el-table-column>

### 实现分页功能

找到Data中的Pagination分页中的完整功能

### 实现省市区县数据联动效果

找到Others中的Dialog对话框

找到D:\vs code html\project\电商项目实战\8.vue-项目实战day6\素材中的citydata.js复制到components下的order下，和order.vue平级。

在order.vue中导入citydata.js

级联选择器

 <el-form-item *label*="省市区/县" *prop*="address1">

          <el-cascader :*options*="cityData" *v-model*="addressForm.address1"></el-cascader>

        </el-form-item>

### 展示物流进度对话框并获取物流信息

找到Others中的Dialog对话框

**接口文档1.10.5**

    async showProgressBox() {

      const { data: res } = await *this*.$http.get('/kuaidi/804909574412544580')

      if (res.meta.status !== 200) {

        return *this*.$message.error('获取物流进度失败！')

      }

*this*.progressInfo = res.data

*this*.progressVisible = true

      console.log(*this*.progressInfo)

    }

### 手动导入并使用Timeline组件

在Others中找到Timeline时间线2.6.0 是2019三月发布的 而vue-cli-plugin-element是2019一月发布的，所以没有时间线。

D:\vs code html\project\电商项目实战\8.vue-项目实战day6\素材中找到timeline和timeline-item文件，复制到src中的plugins中导入两个文件相应的index.js

在element.js中导入

import Timeline from './timeline/index.js'

import TimelineItem from './timeline-item/index.js'

注册

<!-- 时间线 -->

      <el-timeline>

        <el-timeline-item *v-for*="(activity, index) in progressInfo" :*key*="index" :*timestamp*="activity.time">

          {{ activity.context }}

        </el-timeline-item>

      </el-timeline>

### 将本地order分支的代码合并并且推送到码云

git branch

git status

git add .

git status

git commit -m “完成订单功能的开发”

git push

git checkout master

git branch

git merge order

git push

### 创建report子分支

git checkout -b report

git branch

git push -u origin report

# 数据统计

## 数据报表

### 通过路由加载数据报表组件

在components创建report文件创建Report.vue

面包屑导航

卡片视图区域

### 安装Echarts并渲染Demo图表

找到可视化工具点击依赖-运行依赖搜索echarts查看详情，安装

// 1. 导入 echarts

import echarts from 'echarts'

  <!-- 卡片视图区域 -->

    <el-card>

      <!-- 2. 为ECharts准备一个具备大小（宽高）的Dom -->

      <div *id*="main" *style*="width: 750px;height:400px;"></div>

    </el-card>

 // 此时，页面上的元素，已经被渲染完毕了！

  async mounted() {

    // 3. 基于准备好的dom，初始化echarts实例

    var myChart = echarts.init(document.getElementById('main'))

// 4. 准备数据和配置项

    const result = \_.merge(res.data, *this*.options)

    // 5. 展示数据

    myChart.setOption(result)

### 获取折线图数据并渲染图标

  data() {

    return {

      // 需要合并的数据

      options: {

        title: {

          text: '用户来源'

        },

        tooltip: {

          trigger: 'axis',

          axisPointer: {

            type: 'cross',

            label: {

              backgroundColor: '#E9EEF3'

            }

          }

        },

        grid: {

          left: '3%',

          right: '4%',

          bottom: '3%',

          containLabel: true

        },

        xAxis: [

          {

            boundaryGap: false

          }

        ],

        yAxis: [

          {

            type: 'value'

          }

        ]

      }

    }

  },

import \_ from 'lodash'

### 将本地的report分支合并且推送到码云

git status

git branch

git add .

git status

git commit -m “完成了报表功能的开发”

git push

git branch

git checkout master

git merge report

git push

# 项目优化

## 优化

1. 生成打包报告
2. 第三方库启用CDN
3. ElementUI组件按需加载
4. 路由懒加载
5. 首页内容定制

### 通过nprogress添加进度条效果

可视化工具打开依赖-安装依赖-运行依赖nprogress0.2.0点击安装

找到main.js

// 导入 NProgress 包对应的JS和CSS

import NProgress from 'nprogress'

import 'nprogress/nprogress.css'

// 在 request 拦截器中，展示进度条 NProgress.start()

// 在 response 拦截器中，隐藏进度条 NProgress.done()

### 解决serve命令中提示的ESLint语法错误

点击任务-输出 运行serve 解决警告错误

在.prettierrc文件中

"printWidth": 200

设置一行有多少个字符，超过200就会强制换行，默认80，不会被格式化

### 在执行build命令期间移除所有的console

任务 build 解决警告错误

发布不允许出现console

babel-plugin-transform-remove-console

依赖中开发依赖，进行安装

### 只在发布阶段移除所有的console

在发布阶段没有问题 在测试开发阶段不想移除console怎么解决

只在发布阶段生效不在测试开发阶段生效

在.babel.config.js文件中

// 这是项目发布阶段需要用到的 babel 插件

const prodPlugins = []

if (process.env.NODE\_ENV === 'production') {

  prodPlugins.push('transform-remove-console')

}

 // 发布产品时候的插件数组

    ...prodPlugins,

展开运算符

### 生成打包报告

打包时，为了直观地发现项目中存在的问题，可以在打包时生成报告，生成报告的方式有两种：

1. 通过命令行参数的形式生成报告

通过vue-cli的命令选项可以生成打包报告

--report选项可以生成report.html以帮助分析包内容

vue-cli-service build --report

1. 通过可视化的UI面板直接查看报告(推荐)

在可视化的UI面板中，通过控制台和分析面板，可以方便地看到项目中所存在地问题。

任务-build-运行-分析 体积大小 控制台 资源 依赖项 进行优化

### 通过vue.config.js修改webpack的默认配置

通过vue-cli3.0工具生成的项目，默认隐藏了所有webpack的配置项，目的是为了屏蔽项目的配置过程，让程序员把工作的重心，放到具体功能和业务逻辑的实现上。

如果程序员有修改webpack默认配置的需求，可以在项目根目录中，按需创建vue.config.js这个配置文件，从而对项目的打包发布过程做自定义的配置。

创建vue.config.js文件

### 为开发模式与发布模式指定不同的打包入口

默认情况下，vue项目的开发模式与发布模式，共用一个打包的入口文件(即src/main.js)。为了将项目的开发过程与发布过程分离。我们可以为两种模式，各自指定打包的入口文件，即：

1. 开发模式的入口文件为src/main-dev.js
2. 发布模式的入口文件为src/main-prod.js

configureWebpack和chainWebpack

在vue.config.js导出的配置对象中，新增configureWebpack或chainWebpack节点，来自定义webpack的打包配置。

在这里，configureWebpack和chainWebpack的作用相同，唯一的区别就是它们修改webpack配置的方式不同：

1. chainWebpack通过链式编程的形式，来修改默认的webpack配置
2. configureWebpack通过操作对象的形式，来修改默认的webpack配置

### 通过chainWebpack自定义打包入口

将main.js重命名为main-dev.js,创建main-prod.js文件

*module*.*exports* = {

  chainWebpack: *config* => {

    // 发布模式

*config*.when(process.env.NODE\_ENV === 'production', *config* => {

*config*

        .entry('app')

        .clear()

        .add('./src/main-prod.js')

*config*.set('externals', {

        vue: 'Vue',

        'vue-router': 'VueRouter',

        axios: 'axios',

        lodash: '\_',

        echarts: 'echarts',

        nprogress: 'NProgress',

        'vue-quill-editor': 'VueQuillEditor'

      })

*config*.plugin('html').tap(*args* => {

*args*[0].isProd = true

        return *args*

      })

    })

    // 开发模式

*config*.when(process.env.NODE\_ENV === 'development', *config* => {

*config*

        .entry('app')

        .clear()

        .add('./src/main-dev.js')

*config*.plugin('html').tap(*args* => {

*args*[0].isProd = false

        return *args*

      })

    })

  }

}

### 通过externals加载外部CDN资源

默认情况下，通过import语法导入的第三方依赖包，最终会被打包合并到同一个文件中，从而导致打包成功后，单文件体积过大的问题。

可以通过webpack的externals节点，来配置并加载外部的CDN资源。凡是声明在externals中的第三方依赖包，都不会被打包。

第一步：

*config*.set('externals', {

        vue: 'Vue',

        'vue-router': 'VueRouter',

        axios: 'axios',

        lodash: '\_',

        echarts: 'echarts',

        nprogress: 'NProgress',

        'vue-quill-editor': 'VueQuillEditor'

      })

第二步：样式表

同时，需要在public/index.html文件的头部，添加如下的CDN资源引用：

   <!-- nprogress 的样式表文件 -->

    <link *rel*="stylesheet" *href*="https://cdn.staticfile.org/nprogress/0.2.0/nprogress.min.css" />

    <!-- 富文本编辑器 的样式表文件 -->

    <link *rel*="stylesheet" *href*="https://cdn.staticfile.org/quill/1.3.4/quill.core.min.css" />

    <link *rel*="stylesheet" *href*="https://cdn.staticfile.org/quill/1.3.4/quill.snow.min.css" />

    <link *rel*="stylesheet" *href*="https://cdn.staticfile.org/quill/1.3.4/quill.bubble.min.css" />

第三步：js文件

同时，需要在public/index.html文件的头部，添加如下的CDN资源引用：

   <script *src*="https://cdn.staticfile.org/vue/2.5.22/vue.min.js"></script>

    <script *src*="https://cdn.staticfile.org/vue-router/3.0.1/vue-router.min.js"></script>

    <script *src*="https://cdn.staticfile.org/axios/0.18.0/axios.min.js"></script>

    <script *src*="https://cdn.staticfile.org/lodash.js/4.17.11/lodash.min.js"></script>

    <script *src*="https://cdn.staticfile.org/echarts/4.1.0/echarts.min.js"></script>

    <script *src*="https://cdn.staticfile.org/nprogress/0.2.0/nprogress.min.js"></script>

    <!-- 富文本编辑器的 js 文件 -->

    <script *src*="https://cdn.staticfile.org/quill/1.3.4/quill.min.js"></script>

    <script *src*="https://cdn.jsdelivr.net/npm/vue-quill-editor@3.0.4/dist/vue-quill-editor.js"></script>

### 演示externals的效果

可视化-build-资源体积变小

### 通过CDN优化ElementUI的打包

虽然在开发阶段，我们启用了element-ui组件的按需加载，尽可能减少打包的体积，但是那些被按需加载的组件，还是占用了较大的文件体积。此时，我们可以将element-ui中的组件，也通过CDN的形式来加载，这样能够进一步减小打包后的文件体积。

具体操作流程如下：

1. 在main-prod.js中，注释掉element-ui按需加载的代码
2. 在index.html的头部区域中，通过CDN加载element-ui的js和css样式

 <!-- element-ui 的样式表文件 -->

    <link *rel*="stylesheet" *href*="https://cdn.staticfile.org/element-ui/2.8.2/theme-chalk/index.css" />

   <!-- element-ui 的 js 文件 -->

    <script *src*="https://cdn.staticfile.org/element-ui/2.8.2/index.js"></script>

在main-prod.js中

删除import ‘./plugins/element.js’

### 自定制首页内容

不同打包环境下，首页内容可能会有所不同。我们可以通过插件的方式进行定制，插件配置如下：

在vue.config.js

 // 发布模式

*config*.when(process.env.NODE\_ENV === 'production', *config* => {

*config*.plugin('html').tap(*args* => {

*args*[0].isProd = true

        return *args*

      })

    })

 // 开发模式

*config*.when(process.env.NODE\_ENV === 'development', *config* => {

*config*.plugin('html').tap(*args* => {

*args*[0].isProd = false

        return *args*

      })

    })

在public/index.html首页中，可以根据isProd的值，来决定如何渲染页面结构

按需渲染页面的标题

 <title>*<*%= htmlWebpackPlugin.options.isProd ? '' : 'dev - ' %>电商后台管理系统</title>

按需加载外部的CDN资源

通过externals加载的外部CDN资源

模板字符串语法

*<*% if(htmlWebpackPlugin.options.isProd){ %> *<*% } %>

可视化-serve 运行

开发中dist下index.html 标题为电商后台管理系统

发布中项目运行标题为dev-电商后台管理系统

### 实现路由懒加载

当打包构建项目时，JavaScript包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

具体需要3步：

1. 安装@babel/plugin-syntax-dynamic-import包
2. 在babel.config.js配置文件中声明该插件。
3. 将路由改为按需加载的形式，示例代码如下：

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

const Login = () => import(/\* webpackChunkName: "login\_home\_welcome" \*/ './components/Login.vue')

// import Home from './components/Home.vue'

const Home = () => import(/\* webpackChunkName: "login\_home\_welcome" \*/ './components/Home.vue')

// import Welcome from './components/Welcome.vue'

const Welcome = () => import(/\* webpackChunkName: "login\_home\_welcome" \*/ './components/Welcome.vue')

安装 npm install --save-dev @babel/plugin-syntax-dynamic-import

或者在可视化中依赖中开发依赖 @babel/plugin-syntax-dynamic-import安装

在babel.config.js中配置

 // 发布产品时候的插件数组

    ...prodPlugins,

    '@babel/plugin-syntax-dynamic-import'

可视化面板 任务-build-运行

## 上线

### 通过node创建web服务器

1. 通过node创建web服务器
2. 开始gzip配置
3. 配置https服务
4. 使用pm2管理应用

创建node项目，并安装express，通过express快速创建web服务器，将vue打包生成的dist文件夹，托管为静态资源即可，关键代码如下：

D:\vs code html\project\电商项目实战\9.vue-项目实战day7\code创建vue\_shop\_server右键用vscode打开

在终端中npm init -y 初始化项目

npm i express -S

复制vue\_shop目录下的dist到vue\_shop\_server

新建app.js文件

const express = require('express')

const compression = require('compression')

const *https* = require('https')

const *fs* = require('fs')

const app = express()

const options = {

  cert: *fs*.readFileSync('./full\_chain.pem'),

  key: *fs*.readFileSync('./private.key')

}

// 一定要把这一行代码，写到 静态资源托管之前

app.use(compression())

app.use(express.static('./dist'))

app.listen(80, () => {

  console.log('server running at http://127.0.0.1')

})

// https.createServer(options, app).listen(443)

终端中node .\app.js

### 开启文件的Gzip网络传输压缩

开启gzip配置

使用gzip可以减小文件体积，使传输速度更快。

可以通过服务器端使用Express做gzip压缩。其配置如下：

检查源代码中 network 文件大小和网络传输大小对比

安装相应包

npm install compression -D

导入包

const compression = require('compression')

使用中间件

app.use(compression())

### 配置HTTPS服务

采用HTTPS协议对传输的数据进行了加密处理，可以防止数据被中间人窃取，使用更加安全

申请SSL证书(https://freessl.org)

1. 进入https://freessl.cn/官网，输入要申请的域名并选择品牌
2. 输入自己的邮箱并选择相关选项
3. 验证DNS(在域名管理后台添加TXT记录)
4. 验证通过后，下载SSL证书(full\_chain.pem公钥；private.key私钥)

在后台项目中导入证书

const *https* = require('https')

const *fs* = require('fs')

const options = {

  cert: *fs*.readFileSync('./full\_chain.pem'),

  key: *fs*.readFileSync('./private.key')

}

// https.createServer(options, app).listen(443)

将素材中的full\_chain.pem和private.key复制到vue\_shop\_server

### 使用pm2管理应用

在vue\_shop\_server中powershell 终端中输入node .\app.js

关闭终端窗口还可以运行

1. 在服务器中安装pm2：npm i pm2 -g
2. Cmd中安装 启动项目：pm2 start脚本 --name自定义名称
3. 查看运行项目：pm2 ls
4. 重启项目：pm2 restart自定义名称
5. 停止项目：pm2 stop 自定义名称
6. 删除项目:pm2 delete 自定义名称

在powershell终端中输入pm2 start .\app.js --name web\_vueshop

关闭powershell也可以运行

在cmd终端中输入 pm2 ls

关闭pm2 stop 0

重启 pm2 restart 0

删除pm2 delete 0