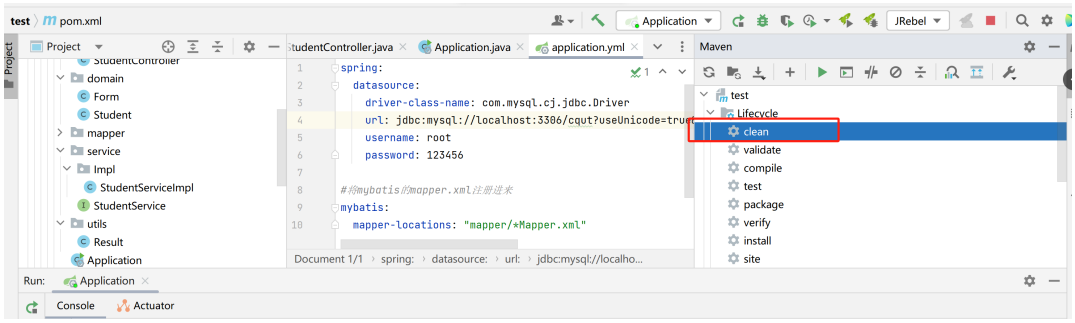
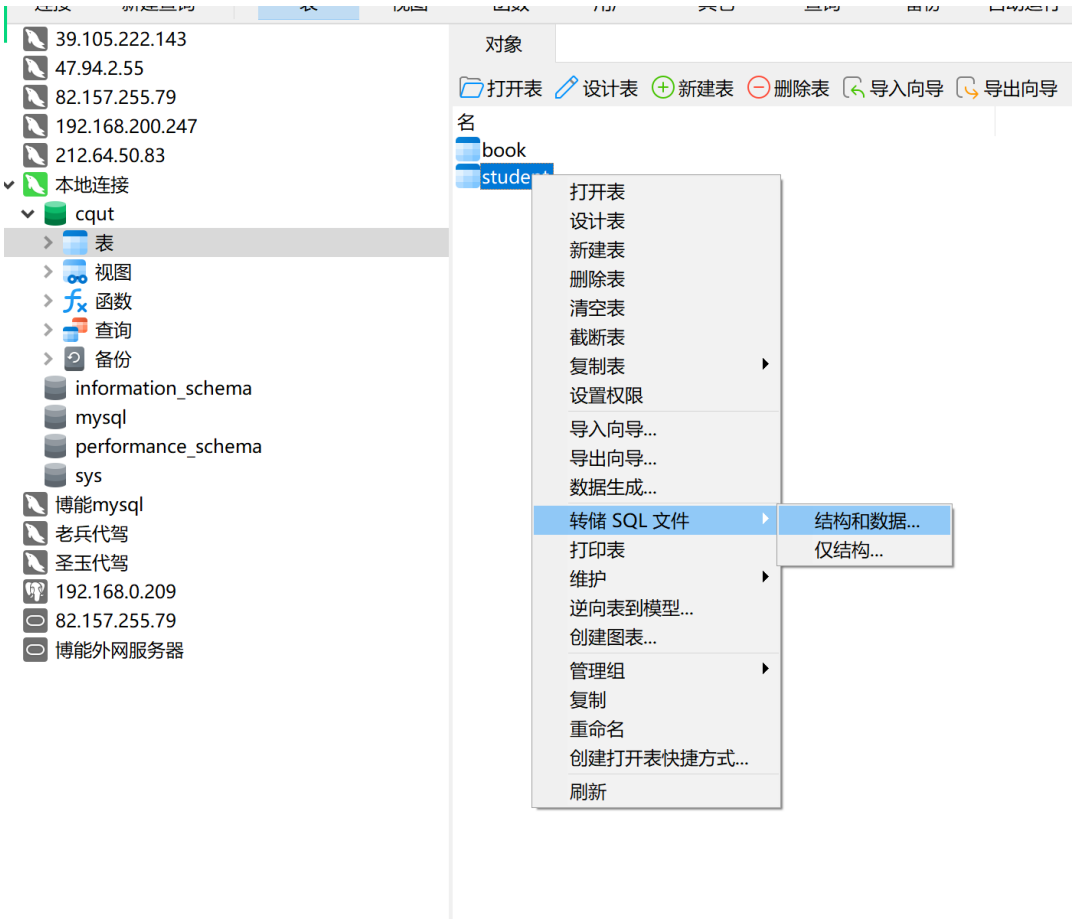
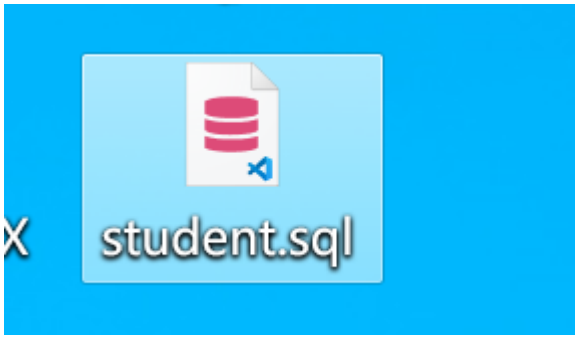
一、作业提交

- 前端代码
  - 删除node_modules
- 后端代码
  - 
  - clean后上交
- 数据库文件
  - 

- 

二、高级查询

1、分析

分析：

当我点击查询：

- 我要调用加载表格数据的方法loadStudents加载数据，并且把姓名和性别作为参数传到接口中
- 每次点击查询，都是从第一页开始显示

分析：

当我点击清空：

- 重置查询表单
- 重新执行查询 - 从第一页开始

2、代码实现

（1）查询条件的form表单


```vue
<template>
  <div>
    <!-- 查询条件的form表单 -->
    <el-form :inline="true" :model="searchForm" label-width="80px">
      <el-form-item label="姓名">
        <el-input v-model="searchForm.name" placeholder="请输入姓名"></el-input>
      </el-form-item>
      <el-form-item label="性别">
        <el-select v-model="searchForm.gender" clearable placeholder="请选择性别">
          <el-option
            v-for="item in genderOptions"
            :key="item.value"
            :label="item.label"
            :value="item.value">
          </el-option>
        </el-select>
      </el-form-item>
      <el-form-item label="出生日期">
        <el-date-picker
          v-model="searchForm.birthday"
          type="daterange"
          range-separator="至"
          start-placeholder="开始日期"
          end-placeholder="结束日期">
        </el-date-picker>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" @click="onSearch" icon="el-icon-search">查询</el-button>
      </el-form-item>
      <el-form-item>
        <el-button @click="onClear" icon="el-icon-refresh-right">清空</el-button>
      </el-form-item>
    </el-form>
  </div>
  <!--
    el-table 表格组件
```

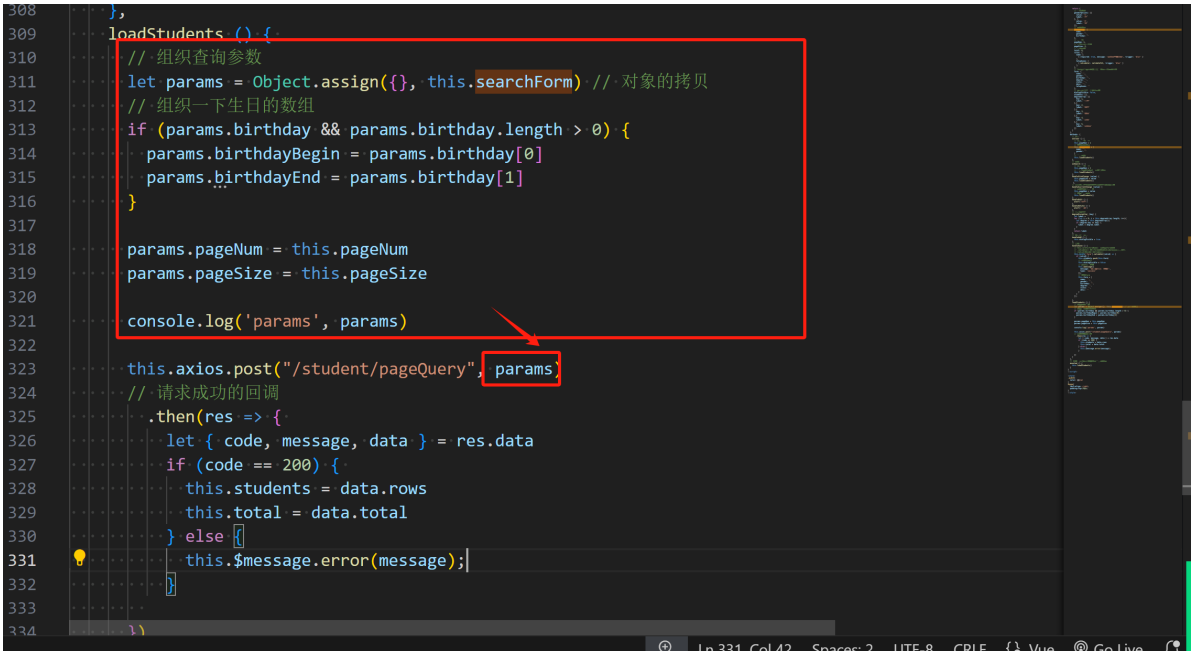
（2）data数据绑定

```
167
168
169     return {
170       // 性别数组
171       genderOptions: [{
172         value: '1',
173         label: '男'
174       }, {
175         value: '2',
176         label: '女'
177       }],
178       // 查询表单
179       searchForm: {
180         name: '',
181         gender: '',
182         birthday: ''
183       },
184       // 第几页
185       pageNum: 1,
186       // 每页显示多少条
187       pageSize: 2,
188       // 总条目数
189       total: 0,
190       rules: {
191         name: [
192           { required: true, message: '请输入学生姓名', trigger: 'blur' }
193         ],
194         telephone: [
```

（3）查询和清空的时间处理函数methods

```
153   data () {
227     }
228   },
229   methods: {
230     // 清空
231     onClear () {
232       // 从第一页开始
233       this.pageNum = 1
234       // 重置查询表单
235       this.searchForm = {
236         name: '',
237         gender: ''
238       }
239       // 执行查询
240       this.loadStudents()
241     },
242     // 查询
243     onSearch () {
244       // 从第一页开始
245       this.pageNum = 1
246       // 执行loadStudent加载表格数据
247       this.loadStudents()
248     },
249     handleSizeChange (value) {
250       this.pageSize = value
251       this.loadStudents()
252     },
253     // 当我点击分页的时候触发的时间的处理函数
254     handleCurrentChange (value) {
255       // 获得点的是第几页
256       this.pageNum = value
257       // 重新执行查询
258       this.loadStudents()
```

（4）修改loadStudents，添加查询参数处理的业务



```
},
loadStudents () {
  // 组织查询参数
  let params = Object.assign({}, this.searchForm) // 对象的拷贝
  // 组织一下生日的数组
  if (params.birthday && params.birthday.length > 0) {
    params.birthdayBegin = params.birthday[0]
    params.birthdayEnd = params.birthday[1]
  }

  params.pageNum = this.pageNum
  params.pageSize = this.pageSize

  console.log('params', params)

  this.axios.post("/student/pageQuery", params)
  // 请求成功的回调
    .then(res => {
      let { code, message, data } = res.data
      if (code == 200) {
        this.students = data.rows
        this.total = data.total
      } else {
        this.$message.error(message);
      }

  })
```

三、登录

1、数据库

2、后端接口

User.java 实体类

```
package cn.itsource.domain;

import lombok.Data;

@Data
public class User {

    private Long id;

    private String username;

    private String password;

}
```

UserCache.java 服务端缓存用户和token信息

```
package cn.itsource.config;
```

```
import cn.itsource.domain.User;

import java.util.HashMap;
import java.util.Map;

public class UserCache {

    private static Map<String, User> userMap = new
HashMap<>();

    public static void putUser(String token,User user){
        userMap.put(token,user);
    }

    public static User getUser(String token){
        return userMap.get(token);
    }

}
```

LoginController.java

```
package cn.itsource.controller;

import cn.itsource.domain.User;
import cn.itsource.service.IUserService;
import cn.itsource.util.AjaxResult;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class LoginController {

    @Autowired
```

```
    private IUserService userService;

    /**
     * 登录
     * @param user
     * @return
     */
    @PostMapping("/login")
    public AjaxResult login(@RequestBody User user){
        String token = userService.login(user);
        return
AjaxResult.success().setData(token).setMessage("登录成
功!");
    }

}
```

IUserService.java

```
package cn.itsource.service;

import cn.itsource.domain.User;

public interface IUserService {
    /**
     * 登录
     * @param user
     * @return
     */
    String login(User user);
}
```

UserServiceImpl.java

```
package cn.itsource.service.impl;

import cn.itsource.config.UserCache;
import cn.itsource.domain.User;
import cn.itsource.mapper.UserMapper;
import cn.itsource.service.IUserService;
```

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import java.util.UUID;

@Service
public class UserServiceImpl implements IUserService {

    @Resource
    private UserMapper userMapper;

    /**
     * 登录
     * @param user
     * @return
     */
    @Override
    public String login(User user) {
        // 1、根据用户名查询用户信息，看用户是否存在
        User resultUser =
userMapper.selectByUsername(user.getUsername());
        if(resultUser == null){
            throw new RuntimeException("用户名或密码错误!");
        }
        // 2、用户存在，把前端传过来的密码和数据库中的密码进行比较

 if(!user.getPassword().equals(resultUser.getPassword())){
            throw new RuntimeException("用户名或密码错误!");
        }
        // 登录成功!! 生成token
        // token和登录的用户信息之间要一一对应
        // 项目中一般存到redis中
        // JWT
        String token = UUID.randomUUID().toString();
        UserCache.putUser(token,resultUser);

        return token;
    }
}
```

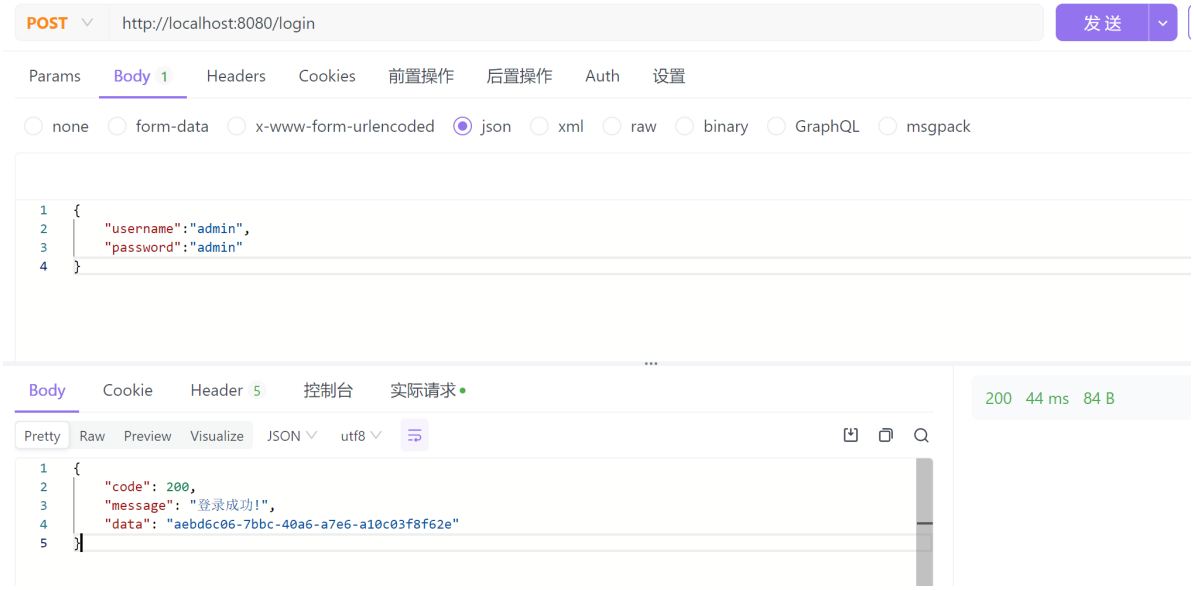
UserMapper.java

```
package cn.itsource.mapper;

import cn.itsource.domain.User;

public interface UserMapper {
    /**
     * 根据用户名查询用户信息
     * @param username
     * @return
     */
    User selectByUsername(String username);
}
```

UserMapper.xml

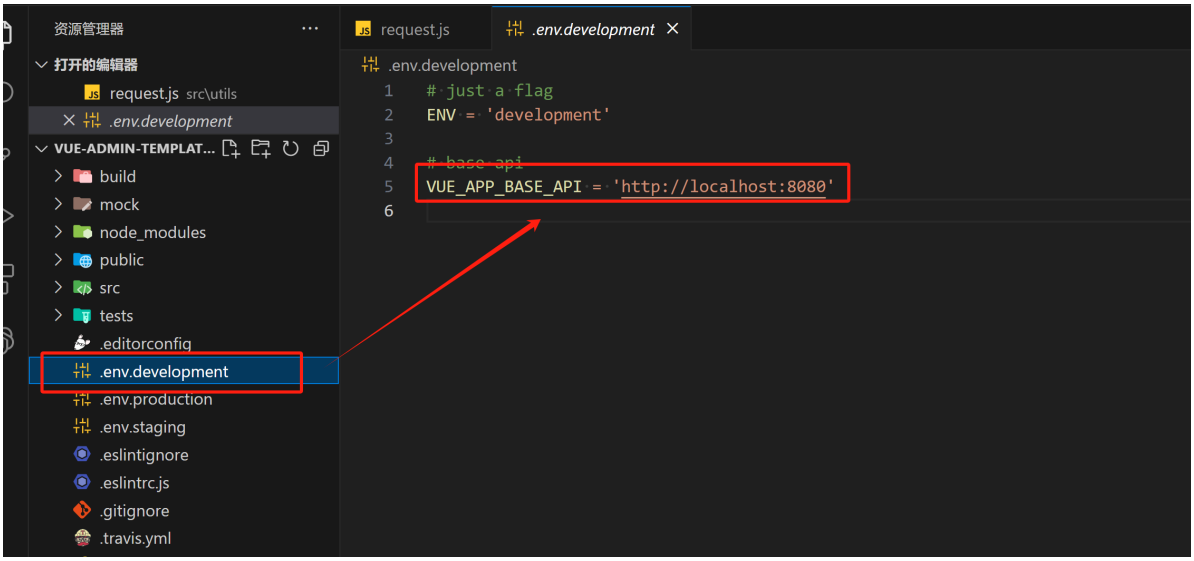
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper
3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="cn.itsource.mapper.UserMapper">
    <select id="selectByUsername"
resultType="cn.itsource.domain.User">
        select * from user where username = #{username}
    </select>
</mapper>
```

测试

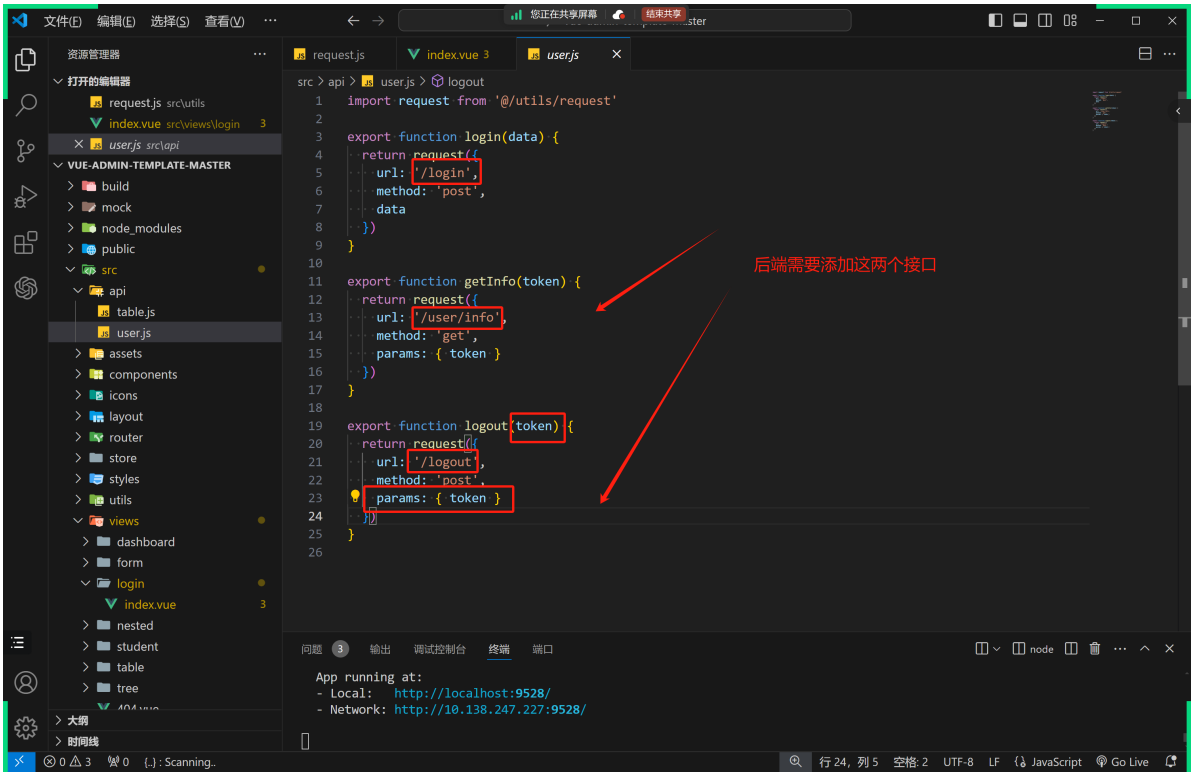


3、前端登录修改

更改development的环境文件，axios的baseUrl指向后端的服务器地址

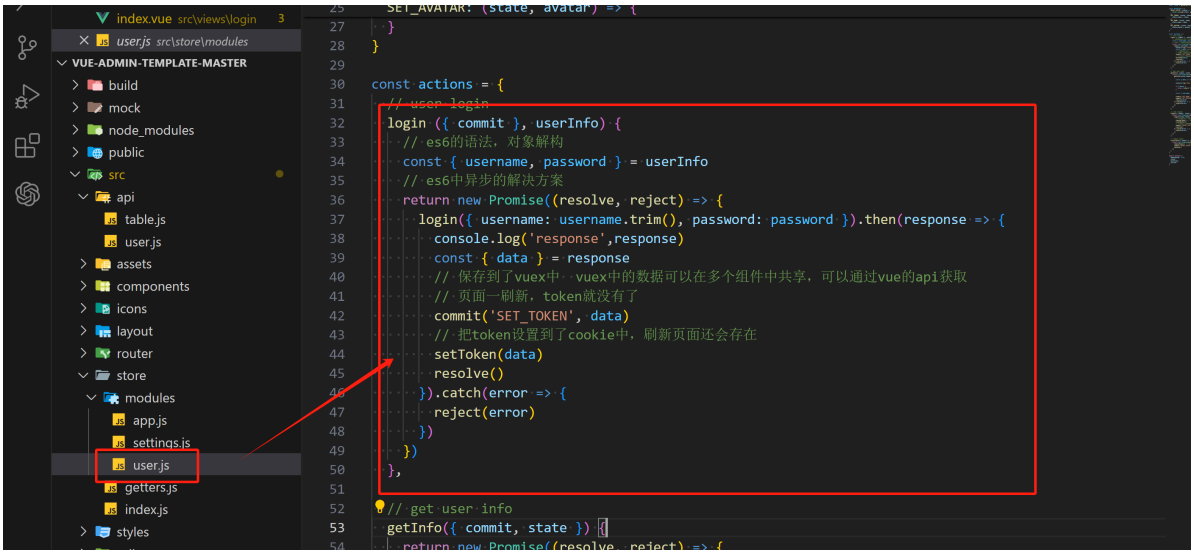


api/user.js




```
import request from '@/utils/request'

export function login(data) {
  return request({
    url: '/login',
    method: 'post',
    data
  })
}

export function getInfo(token) {
  return request({
    url: '/user/info',
    method: 'get',
    params: { token }
  })
}

export function logout(token) {
  return request({
    url: '/logout',
    method: 'post',
    params: { token }
  })
}
```

```
App running at:
- Local:   http://localhost:9528/
- Network: http://10.138.247.227:9528/
```



```
  SET_AVATAR: (state, avatar) => {
  }
}

const actions = {
  // user login
  login ({ commit }, userInfo) {
    // es6的语法，对象解构
    const { username, password } = userInfo
    // es6中异步的解决方案
    return new Promise((resolve, reject) => {
      login({ username: username.trim(), password: password }).then(response => {
        console.log('response',response)
        const { data } = response
        // 保存到了vuex中，vuex中的数据可以在多个组件中共享，可以通过vue的api获取
        // 页面一刷新，token就没有了
        commit('SET_TOKEN', data)
        // 把token设置到了cookie中，刷新页面还会存在
        setToken(data)
        resolve()
      }).catch(error => {
        reject(error)
      })
    })
  },

  // get user info
  getInfo({ commit, state }) {
    return new Promise((resolve, reject) => {
```

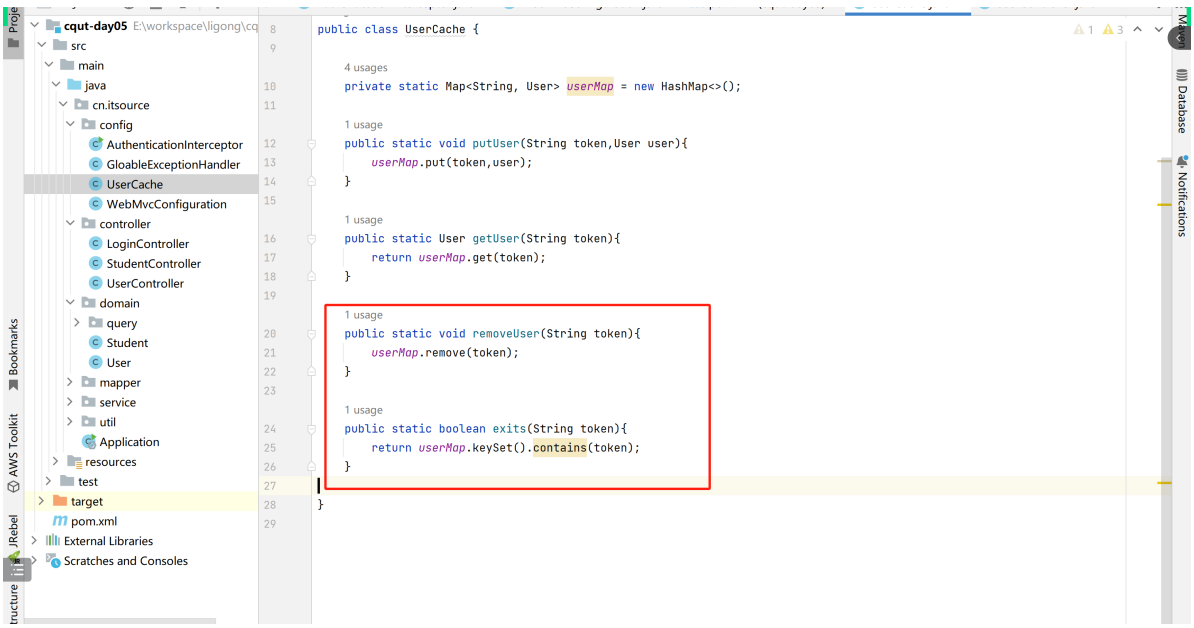
```
      }).catch(error => {
        reject(error)
      })
    })
  },

  // user logout
  logout({ commit, state }) {
    return new Promise((resolve, reject) => {
      logout(state.token).then(() => {
        // 从cookie中删除保存的token
        removeToken() // must remove  token  first
        resetRouter()
        // 把vuex中保存的数据也重置了
        commit('RESET_STATE')
        resolve()
      }).catch(error => {
        reject(error)
      })
    })
  },

  // remove token
  resetToken({ commit }) {
    return new Promise(resolve => {
      removeToken() // must remove  token  first
      commit('RESET_STATE')
```

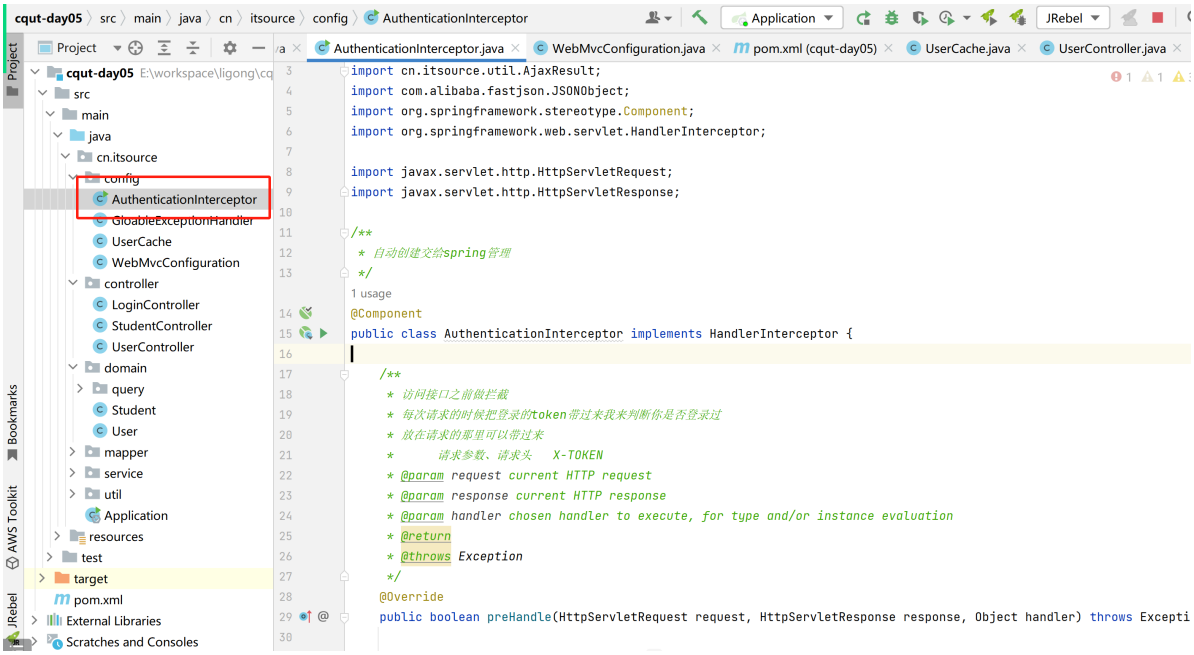
4、登录的拦截器

拦截器的目的：防止有些人绕过前端登录直接访问后端接口

解决方案：

- 编写拦截器拦截请求，判断请求头中的X-TOKEN是否是登录过的token
- 前端发送请求的时候携带X-TOKEN请求头

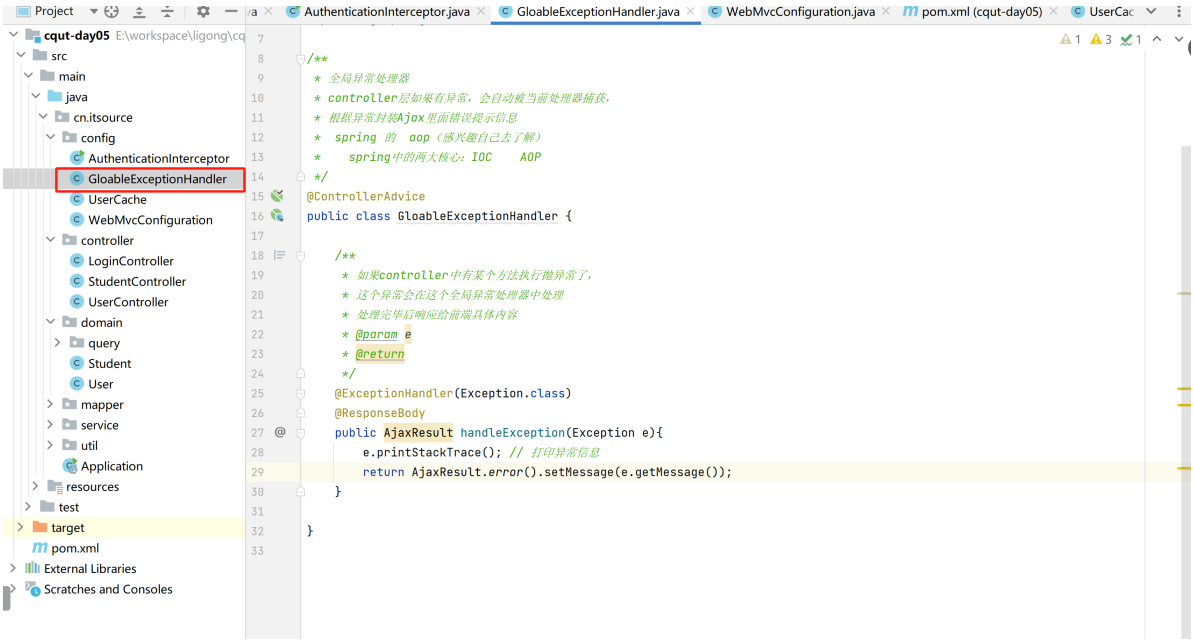





```
package cn.itsource.config;

import cn.itsource.util.AjaxResult;
import com.alibaba.fastjson.JSONObject;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 自动创建交给spring管理
 */
@Component
public class AuthenticationInterceptor implements
HandlerInterceptor {

    /**
     * 访问接口之前做拦截
     * 每次请求的时候把登录的token带过来我来判断你是否登录过
     * 放在请求的那里可以带过来
     *       请求参数、请求头    X-TOKEN
     * @param request current HTTP request
     * @param response current HTTP response
     * @param handler chosen handler to execute, for type
and/or instance evaluation
     * @return
```

```
     * @throws Exception
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws
Exception {

        String token = request.getHeader("X-TOKEN");
        if(UserCache.exits(token)){
            return true;
        }
        // 失败了，响应
        AjaxResult ajaxResult =
AjaxResult.error().setCode(402).setMessage("请先登录!");
        // java web的基础
        response.setHeader("content-
type","text/json;charset=utf-8");

 response.getWriter().println(JSONObject.toJSONString(ajax
Result));
        return false;
    }

    public static void main(String[] args) {
        AjaxResult ajaxResult =
AjaxResult.error().setCode(402).setMessage("请先登录!");

 System.out.println(JSONObject.toJSONString(ajaxResult));
    }
}
```

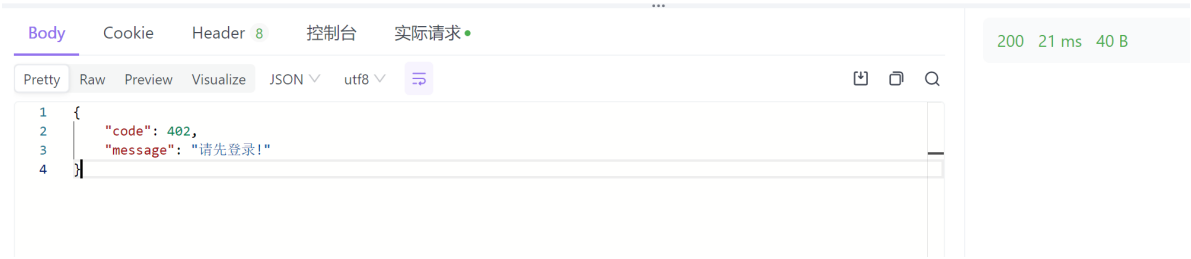
配置拦截器




```java
package cn.itsource.config;

import cn.itsource.util.AjaxResult;
import
org.springframework.web.bind.annotation.ControllerAdvice;
import
org.springframework.web.bind.annotation.ExceptionHandler;
import
org.springframework.web.bind.annotation.ResponseBody;

/**
 * 全局异常处理器
 * controller层如果有异常，会自动被当前处理器捕获，
 * 根据异常封装Ajax里面错误提示信息
 *  spring 的  aop（感兴趣自己去了解）
 *    spring中的两大核心：IOC    AOP
 */
@ControllerAdvice
public class GloableExceptionHandler {

    /**
     * 如果controller中有某个方法执行抛异常了，
     * 这个异常会在这个全局异常处理器中处理
     * 处理完毕后响应给前端具体内容
     * @param e
     * @return
     */
    @ExceptionHandler(Exception.class)
```

```
    @ResponseBody
    public AjaxResult handleException(Exception e){
        e.printStackTrace(); // 打印异常信息
        return
AjaxResult.error().setMessage(e.getMessage());
    }

}
```

测试：



四、作业

vue-admin-template配合后端接口实现登录和退出登录、认证拦截