第2组-Lab2实验报告

1 小组分工

1.1 人员安排

- 组织：林琰钧
- 数据库：林琰钧、马成
  - 使用Navicat进行Mysql数据库关系模型设计
  - 部署数据库
- 后端：林琰钧、马成
  - SpringBoot
  - 编写controller、service、mapper，以及必要的vo、dto、entity
- 前后端接口：林琰钧、马成、王兆瀚、李咸若
  - 使用APIFox进行前后端接口设计
  - 包括url、请求参数（xxxFVO）、响应参数（xxxVO）
- 前端：王兆瀚、李咸若
  - Vue
  - 根据前后端接口，设计页面
- 测试：马成、李咸若
  - 编写数据库数据脚本
  - 编写单元测试
  - 进行接口测试
- 部署：李咸若
  - 将项目前后端串联并部署

1.2 开发记录

- 3.20(Mon.)
  - 下午编译课后线下讨论，讲解项目、安排任务
- To 3.22(Wed.) 23:59.p.m.
  - 林琰钧、马成：利用Navicate设计lab2数据库关系模型、导出sql文件

    - 林琰钧、王兆瀚、李咸若：利用APIFox设计前后端接口与VO
    - 所有人：技术栈的学习
- 3.23(Thu.)
    - 上午编译实验课后线下讨论，确定数据库关系模型、前后端接口
- To 3.23(Thu.) 23:59.p.m.
    - 林琰钧、马成：检查Navicat的数据库设计，导出最终的sql文件，在 `lab2开发` 中填写 `数据库设计` 部分
    - 林琰钧、马成、王兆瀚：检查APIFox的前后端接口与VO，在 `lab2开发` 中填写 `前后端接口` 部分（可以等开发完后填，开发时直接看APIFox）
    - 所有人：继续技术栈的学习
    - 后端开发：
        - 林琰钧：user、shop
        - 马成：admin、myshop
    - 前端开发：
        - 王兆瀚：前端静态页面编写
- 3.27(Mon.)
    - 下午编译课后线下讨论，报告工作进展
- 3.30(Thu.)
    - 上午编译实验课后线下讨论，报告工作进展
- To 3.31(Fri.) 23:59.p.m
    - 前后端：开发完毕
    - 部署与测试：
        - 李咸若：在静态页面中加入动态元素，解决跨域问题，在本地部署前后端，运行并调试项目
    - 调试：
        - 前后端开发人员根据部署与测试中出现的问题进行debug
- To 4.2(Sun.) 23:59.p.m.
    - 完成测试、部署、调试等
- 4.3(Mon.)
    - 上午编译课后线下讨论，演示前后端整合成果
- To 4.3(Mon.) 23:59.p.m.
    - 测试：

        - 马成：写测试数据脚本
        - 李咸若：运行项目测试数据，修复前端bug，报告后端bug
    - 前端debug：王兆瀚、李咸若
        - 消除所有飘红
        - 消除分页等bug
    - 编写心得体会：所有人
    - 准备好项目提交文档

2 实验设计

2.1 开发约定

（仅在本报告中展示，后续报告延续该开发约定，不会再展示）

2.1.1 命名规范

- 搜索资料发现，数据库和前端喜欢用下划线，Java喜欢用驼峰，有2种方案（讨论一下orz）
    - 按照上述规范，不过下划线和驼峰的转换挺麻烦的
    - 都用驼峰，因为Java似乎没办法转下划线，但其他的可以苟全一下
- 讨论结果：全用java的驼峰！！！命名规范如下（教材中的内容）

表 4.1　标识符命名的形式

命名形式	形式特点	例　子	常用情况
大驼峰	大写字母开头；多个单词时，每个单词首字母大写，其他字母小写	CourseOffered	类名、接口名、注解、枚举类型
小驼峰	小写字母开头；多个单词时，除了第一个单词全小写，其他每个单词首字母大写、其余字母小写	selectedCourse	类的属性名、局部变量名、方法名、方法参数
全大写	所有字母大写	MANDATORY、IN_PROGRESS	枚举值、静态常量

- 项目特殊命名规范
    - controller、service、serviceImpl、mapper命名为xxxController、xxxService、xxxServiceImpl、xxxMapper
    - dto的类命名为xxxDTO
    - vo的类命名
        - 前端post请求传给后端（后端 from 前端的 vo）：xxxFVO

        - service传给controller：xxxVO（随后封装入cm传给前端）
    - entity单表的类与数据库表名称相同（除了大写首字母），多表类自行命名

2.1.2 Git规范

（参考书里的git内容）

- 流程

  1、开始时：直接拉取远程仓库（默认在master分支），然后新建一个分支（xxx），进行开发

  git fetch --all

  git checkout -b xxx

  2、如果开发一半有别人推送了master（即远程仓库更新）

  git commit -m "..."

  git switch master

  git pull

  git switch xxx

  git merge master

  3、如果开发完成：从分支xxx切换master分支，然后在master分支rebase分支xxx，然后推送到远程

  git checkout master

  git pull（in case别人已推送）

  git rebase xxx（远程能看到你改了什么）

  git push

  4、之后的开发：在master分支拉取远程master分支，然后新建一个分支（xxx），进行开发，然后重复2、3、4步骤

- commit -m格式：比如lyj在写第x轮迭代，就写"lyj-x: ....(做了啥改动)"

2.1.3 数据库规范

- 数据库用 `mysql`
- 辅助软件使用 `Navicat Premium 15` 软件，因为它可以画图建模，然后由画图一键生成数据库表，不需要自己用代码手动建表！
    - 破解版安装教程：https://www.exception.site/essay/how-to-free-use-navicat

- 我们的Spring Boot持久层可能会选择 `MyBatis` 而不是文档里推荐的 `JPA` 。简言之，虽然JPA的抽象层次更高，代码写起来也更简洁，但是JPA适合业务模型固定的场景，适合比较稳定的需求，但是国内这种朝三暮四的需求风格，产品经理这种传话筒式的设计模式，造成了需求的泛滥和不确定，JPA在这种模式下就是渣……
  - 详细的原因见：https://zhuanlan.zhihu.com/p/263043522
- 有了数据库表后，Spring Boot需要操作对应数据库表的entity类，对于单表查询来说，可以通过groovy脚本一键自动生成所有数据库表，开发时我会给出脚本；对于多表查询来说，请自行手动建entity类。
  - 使用脚本操作见：https://blog.csdn.net/qq_33591873/article/details/111692720（这里边给的脚本要稍微改改）

2.1.4 后端规范

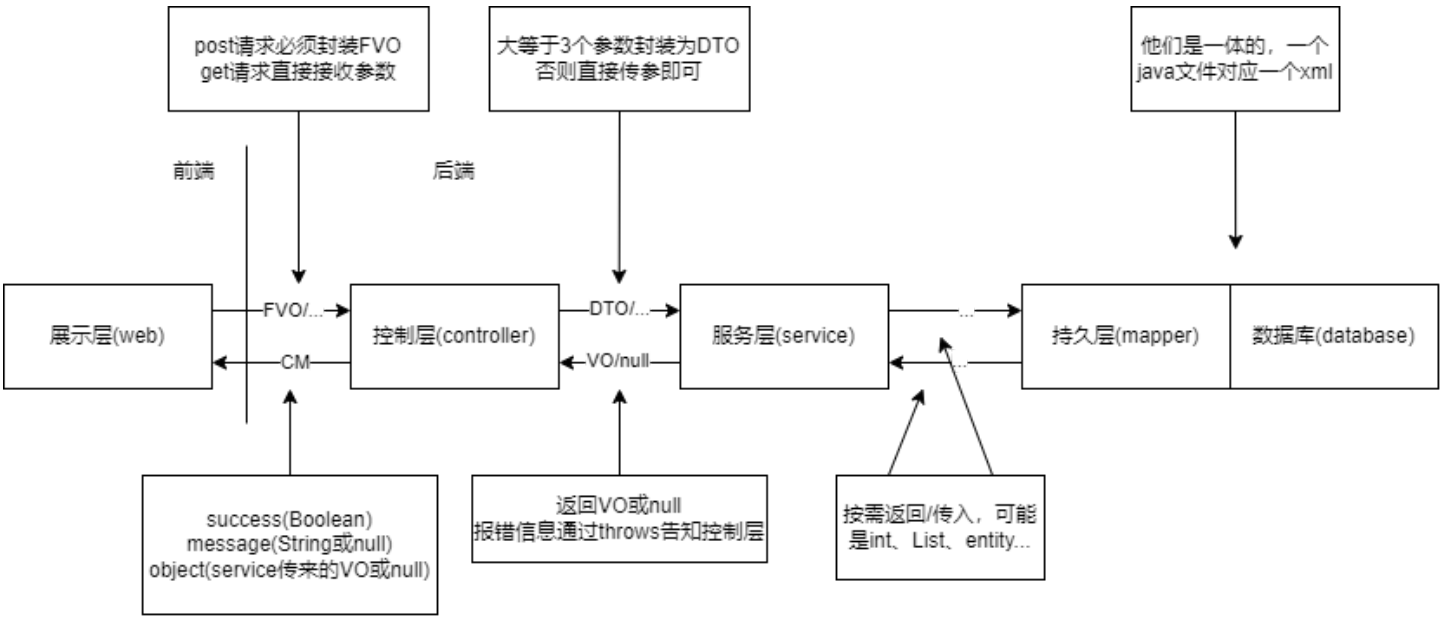
项目结构

```
|_ src/
   |_ main/
      |_ java/ 后端开发
         |_ com.onlineshopping/ 根包(root package)
            |_ controller/ 控制层，每个类调用相应service的方法，为展示层(web前端)提供
               xxxController.java
               ...
            |_ exception/ 异常类，用于service中的方法throws异常
               ServiceException.java
               ...
            |_ mapper/ 持久层，为service提供数据库查询接口
               xxxMapper.java
               xxxMapper.xml
               ...
            |_ model/
               |_ dto/ 数据传输对象(Data Transfer Object)
                  xxxDTO.java
                  ...
               |_ entity/ 实体，对接数据库的表，单表查询的类可以通过脚本自动生成，多表查
                  User.java
                  ...
               |_ vo/ 视图对象(View Object)
                  xxxVO.java
                  xxxFVO.java
                  ...
            |_ service/ 服务层接口
               |_ impl/ 服务层实现，编写核心的代码逻辑，对接mapper和controller!
                  xxxServiceImpl.java
```

```
                xxxService.java
                ...
            |_ .../ 其他必要的文件夹，比如util等工具文件夹
            OnlineShoppingApplication.java 应用启动类
        |_ resources/ 资源文件
            |_ static/ 用来存放静态资源
            |_ templates/ 用来存放默认的模板配置路径
            application.yml 配置文件
        |_ webapp/ 前端开发
            ...
    |_ test/ 测试
```

实体类

- 在model文件夹下有：vo、dto、entity



模块

- mapper将sql操作映射为方法，为service提供数据库查询接口
- service中编写核心的业务逻辑，承上启下的角色，代码往这里堆！异常要throw！
- controller的逻辑务必简单！一个controller对应一个URL，调用一个service的方法，要try catch处理异常！

注释

- 每个方法都要有如下注释

```
/**
 * @Description:
 * @Author:
```

```
4  */
```

- 方法的实现关键流程也需要有必要的注释

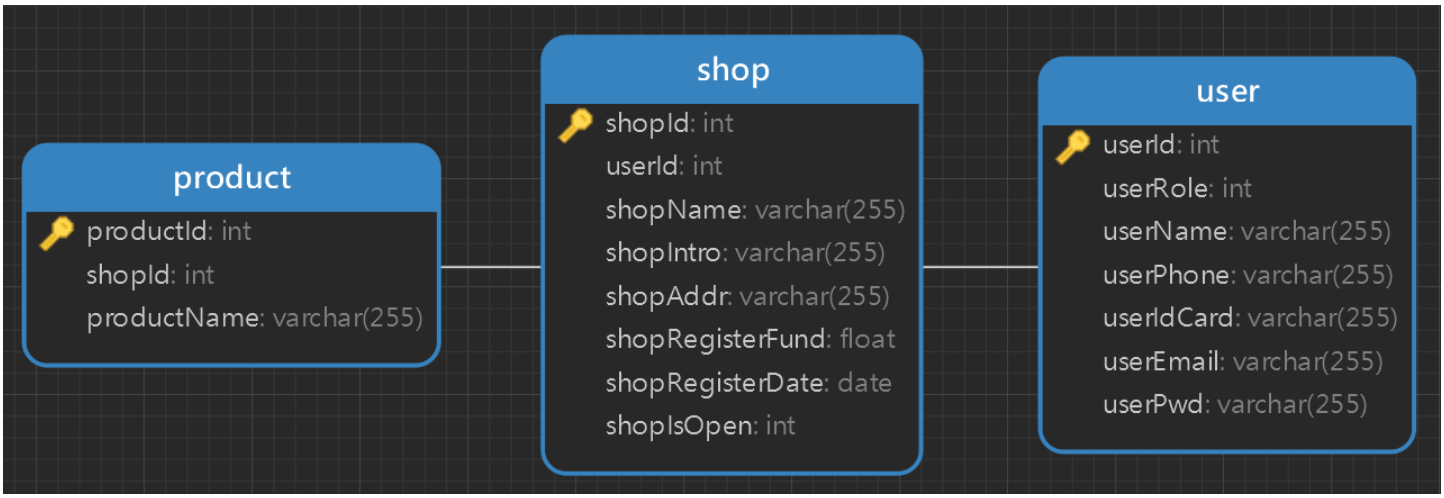
2.1.5 前后端接口规范

- 使用 `APIFox` 定义前后端接口
  - 【Apifox】Demon0511邀请你加入软工 https://www.apifox.cn/web/invite?token=swNKKDoq98saHwcGzt-Y-
  - APIFox可以将设计的接口导出为Markdown文件，因此在APIFox里务必详细说明各字段与接口的含义，这样一键导出即可，不需要再额外写文档
- 接口类型
  - 只使用GET或POST
  - 因为RESTful风格的API 固然很好很规范，但大多数互联网公司并没有按照或者完全按照其规则来设计，因为REST是一种风格，而不是一种约束或规则，过于理想的RESTful API 会付出太多的成本。比如RESTful API也有一些缺点
    - 操作方式繁琐，RESTful API通常根据GET、POST、PUT、DELETE 来区分操作资源的动作，而HTTP Method 本身不可直接见，是隐藏的，而如果将动作放到URL的path上反而清晰可见，更利于团队的理解和交流。
    - 有些浏览器对GET,POST之外的请求支持不太友好，还需要特殊额外的处理。
    - 过分强调资源，而实际业务API可能有各种需求比较复杂，单单使用资源的增删改查可能并不能有效满足使用需求，强行使用RESTful风格API只会增加开发难度和成本。
  - 关于RESTful的更多讨论参考：https://zhuanlan.zhihu.com/p/334809573
- 请求参数：在数据模型的 `FVO` 文件夹里定义一个 `xxxFVO` ，然后引用其中的字段
  - 注意：类型；是否必须；可否为空
  - 对相应字段写注释
- 响应参数：
  - 在数据模型的 `VO` 文件夹里定义一个 `CommonResult` ，其中参数列表需要填以下3个字段
    - success：boolean类型；必须；不可为空
    - message：string类型；必须；可以为空
    - object：暂时不填；必须；可以为空
  - 引用 `CommonResult` ，接触绑定，在数据模型的 `VO` 文件夹里定义一个 `xxxVO` ，然后object引用该VO
    - 注意：类型；是否必须；可否为空

- 对相应字段写注释

2.2 数据库设计

2.2.1 物理模型

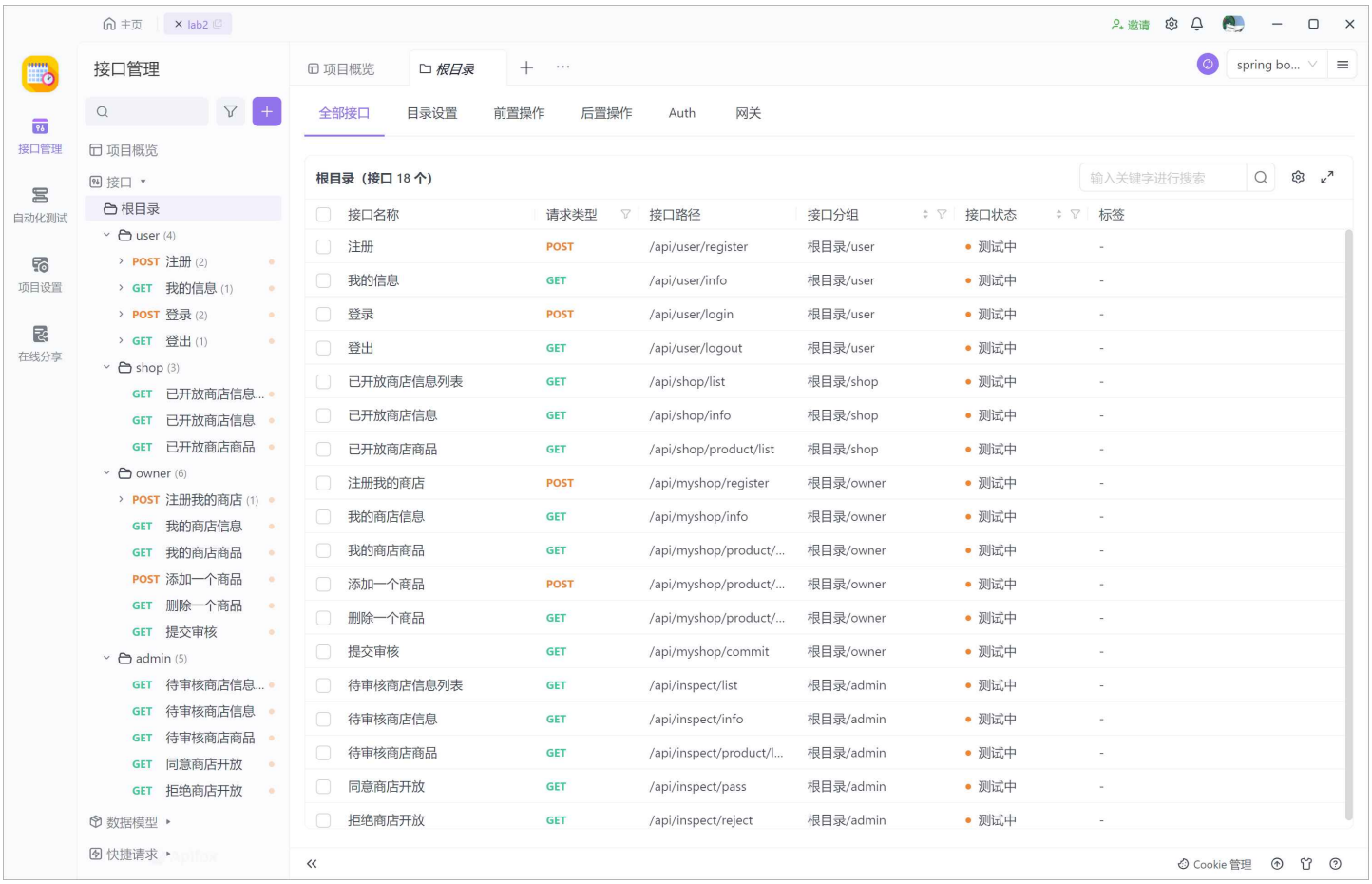


2.2.2 表的概述

- user
  - 存储用户、商户、管理员的信息
  - 主键是userId，自动递增
  - userRole标识用户、商户、管理员，分别是0、1、2
- shop
  - 存储商店信息
  - 主键是shopId，自动递增；外键是userId，级联修改删除
  - shopIsOpen标识商店未提交审核（未开放）、商店提交审核等待管理员处理（未开放）、审核未通过（未开放）、审核通过（开放），分别是0、1、2、3
- product
  - 存储所售卖的商品类别
  - 主键是productId，自动递增；外键是shopId，级联修改删除

2.3 前后端接口设计

2.3.1 设计概览



2.3.2 设计思路

- 注
  - 本次的用户信息、商店与商品信息没有修改，只有增加和删除
  - 涉及的商店等信息的分页查询统一为一页9个，方便后期带图片的九宫格展示
- 用户信息
  - 登录就可以看自己的
- 用户
  - 查看shopIsOpen为3的商店
- 商户
  - 开始时，商户没有商店，显示注册按钮；如果有商店，就显示商店信息与商店商品
  - 如果没有商店，点击注册并提交注册表单，生成shopIsOpen为0的商店，然后显示商店信息与商店商品
  - 如果有商店，显示商店信息与商店商品；如果添加或删除商品（商店信息暂时不允许修改），会导致shopIsOpen变为0
  - shopIsOpen为0时会有提交审核按钮，点击提交审核，shopIsOpen变为1
- 管理员
  - 审核shopIsOpen为1的商店

- 审核通过，则shopIsOpen变为3
- 审核不通过，则删除shop所有信息，需要重新注册（如果之后有修改功能，则保留信息，shopIsOpen变为2）

2.3.3 设计详情

- User
  - POST 注册
    - POST /api/user/register
    - 注册成功后跳转到 ”登录“ 界面
  - POST 登录
    - POST /api/user/login
    - 登录成功后跳转到"以开放商店"页面
  - GET 登出
    - GET /api/user/logout
  - GET 我的信息
    - GET /api/user/info
- shop
  - GET 已开放商店信息列表
    - GET /api/shop/list
    - 发回已开放（shopIsOpen为3）shop所有信息的list，前端选择数据进行展示
  - GET 已开放商店信息
    - GET /api/shop/info
    - 已开放商店的信息（不包括商品）
  - GET 已开放商店商品
    - GET /api/shop/product/list
    - 分页已开放商店的商品
    - 之所以商品和商店分开，是因为商品分页，每次只加载一部分商品，无需重新加载商店信息
- owner
  - POST 注册我的商店
    - POST /api/myshop/register
    - 填写商店信息（不包括商品；注册时间后台自动生成）

- GET 我的商店信息
  - GET /api/myshop/info
  - 获取商店信息（不包括商品）
- POST 添加一个商品
  - POST /api/myshop/product/add
  - 添加一个商品
- GET 删除一个商品
  - GET /api/myshop/product/delete
  - 删除一个商品
- GET 我的商店商品
  - GET /api/myshop/product/list
  - 分页获取商店商品，页号从1开始
- GET 提交审核
  - GET /api/myshop/commit
  - 修改shopIsOpen状态（0->1）

- admin
  - GET 待审核商店信息列表
    - GET /api/inspect/list
    - 发回待审核（shopIsOpen为1）shop所有信息的list，前端选择数据进行展示
  - GET 同意商店开放
    - GET /api/inspect/pass
    - 修改shopIsOpen（1->3）
  - GET 拒绝商店开放
    - GET /api/inspect/reject
    - 删除商店信息+商品
    - 如果之后有修改商店功能的话，无需删除，只需修改shopIsOpen（1->2）
  - GET 待审核商店商品
    - GET /api/inspect/product/list
    - 分页待审核商店的商品
  - GET 待审核商店信息
    - GET /api/inspect/info

- 待审核商店的信息（不包括商品）

2.4 后端特性

- 用户密码会以md5加密到数据库
- 使用token来保存用户登录信息：
  - 加密方式是采用Jwt的方式，即Json Web Tokens，密钥是 `SoftwareEngineering2023`
  - 在登录的时候生成一个token发给前端
  - 之后前端的每次请求都会在header里面携带Authorization字段把这个token给发回来，后端进行校验。
  - token类似session，内容一样，只不过session存在服务器，token存在客户端，而且token可以跨域（重点！详情可以参考：https://cloud.tencent.com/developer/article/1704064）

2.5 前端特性

- 用token而不用session和cookie,可以解决跨域问题
- 只有登录才能访问页面，除了注册页面，其余全部重定向到登录页面。通过token来实现。
- 登录后进入"home"界面在"home"界面，会需要根据用户权限（记录在token中）获得不同的菜单列表
- 登录后如果访问不存在页面，或者不在权限内的页面，会全部重定向到"/home"页面
- 退出后会清除token
- 商店与商品信息分离，按页码访问商品信息时只重新获取新一页商品信息

2.6 前后端拼接

  - 使用axios，封装前端接口
  - 前端本地存储token，(shopId,accessPath)等信息。
  - 跨域流程：vue.config.js 配置转发表，启用代理（https://www.jianshu.com/p/4e6dac726c54）
    - 前端解决跨域，后端不需要改变

2.7 本地部署

2.7.1 环境配置

1. 安装MySQL
   - https://blog.csdn.net/SoloVersion/article/details/123760428

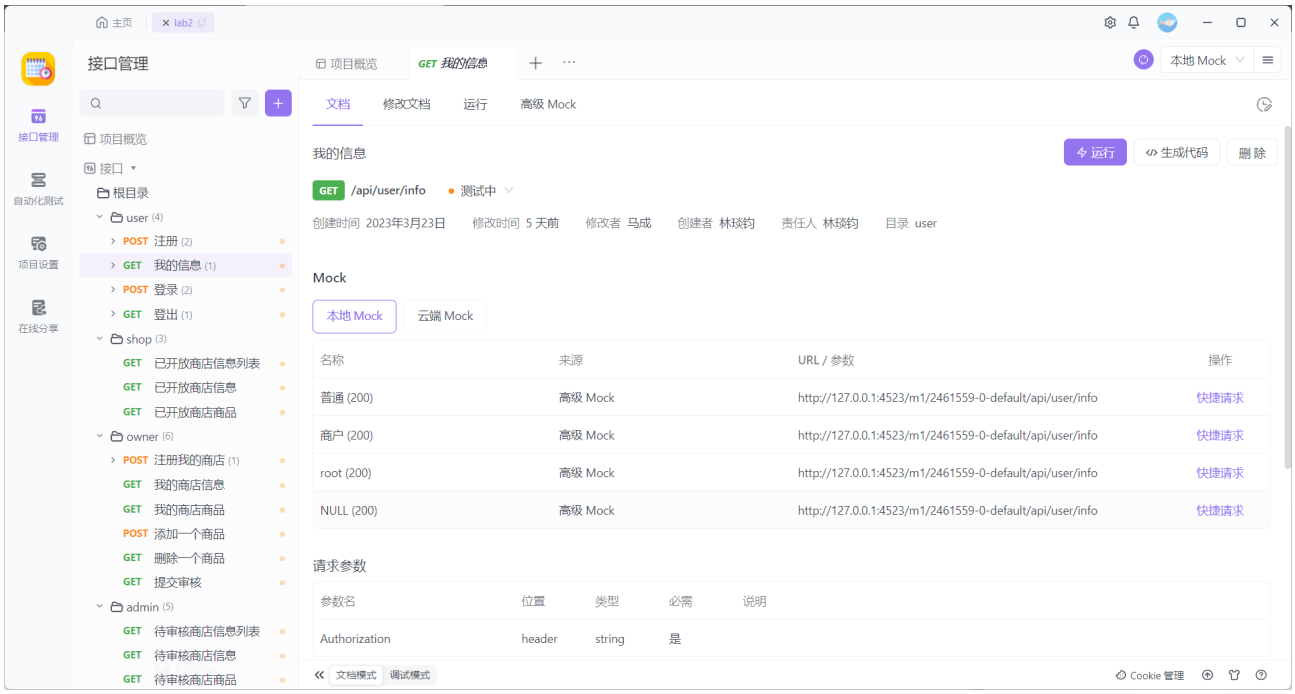
2. 安装Navicat，并运行sql脚本（lab2.sql、test.sql在后端resourses文件夹下的对应位置）
    - https://www.exception.site/essay/how-to-free-use-navicat
3. 安装Intellij
    - https://blog.csdn.net/weixin_44505194/article/details/104452880
4. 安装NodeJS（注意管理员权限运行cmd）
    - https://huangfuyk.blog.csdn.net/article/details/105028164
    - 注意按照教程配置环境变量
5. 安装VUE CLI（注意管理员权限运行cmd）
    - https://blog.csdn.net/leoxingc/article/details/127945708
    - 配置vue的环境变量：https://blog.csdn.net/m0_48276047/article/details/113926266

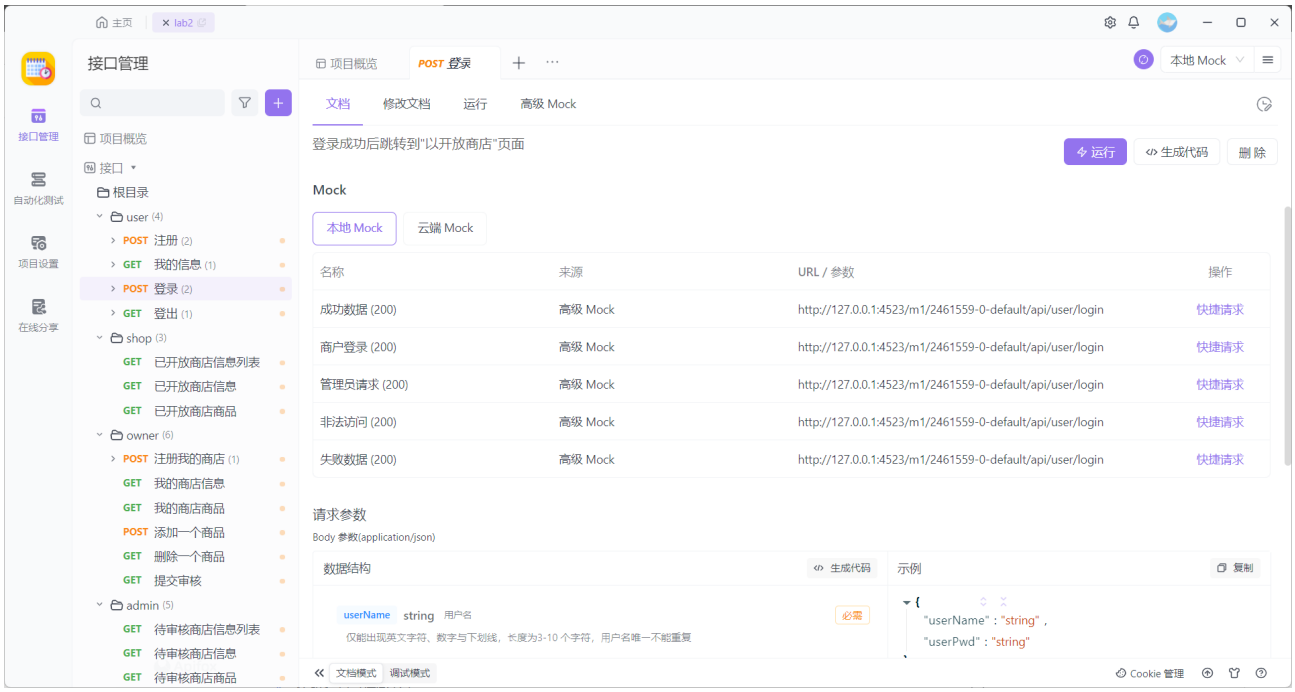
2.7.2 运行流程

1. 如果没有项目的话，先在某个文件夹git clone一下
2. 启动后端：IntelliJ打开项目，并启动，启动地址为localhost:8080
3. 启动前端：在Windows cmd中cd至前端文件目录(*/onlineshopping/webapp/shop)，运行 `npm install` ，检查文件夹下是否存在node_modules，然后输入vue ui（第一次运行时要导入项目），在项目栏中进入该项目。在“任务”边栏，点击“serve” -> "运行"。待编译完成后，点击启动app，此时前端端口为localhost:8081，且前后端能够交互
4. 项目操作：访问localhost:8081相应的url

2.8 测试

1. 单元测试
    - 后端单元测试
        i. TestProductMapper.java 测试了ProductMapper类的功能包括针对Product表的增删改查以及分页功能。
        ii. TestShopMapper.java 测试了ShopMapper类的功能包括针对Shop表的增删改查以及分页功能。
        iii. TestShopService.java 测试了ShopService的相关功能，测试了针对Shop这个模块的Service层的功能包括注册商店，查询已经开张的商店列表等功能。
    - 前端单元测试
        i. APIFOX测试了前端对后端发送请求，得到不同回复的反应。



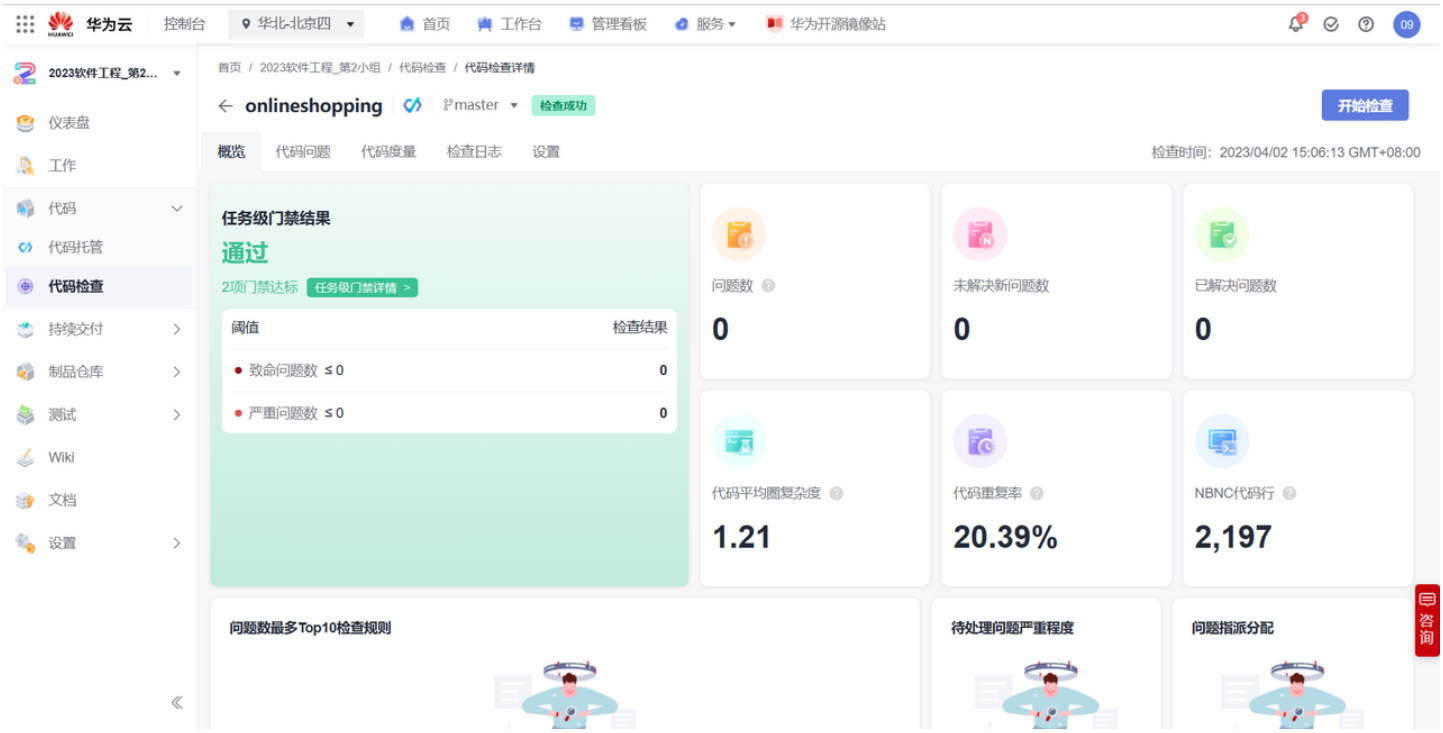


ii. 前端在静态页面的部分（注释掉的部分），模拟了在不导入axios接口的情况下，静态界面的展示逻辑。

2. 简单测试所有功能和页面逻辑（空数据库初始态）
   - 注册并登录了三种权限的账号
   - 在普通用户界面，测试了个人信息，越权访问和商店列表。
     - 测试了商店为空、商店内商品为空的特殊情况
   - 在商户界面，测试了注册商店，修改商品。
   - 在审核员界面，测试了审核商店。
3. 导入数据库测试所有功能和页面逻辑

- 导入了随机生成的数据库，规模为（管理员：1，商户：50-60，用户：50-60，商店：50-60，产品：3000）
- 重复了2中的所有测试
- 测试了分页展示的逻辑，以及边界情况

4. 代码检查（全默认）



3 问题及解决方案

3.1 技术栈的学习

（仅在本报告中列出，后续不会再体现，因为我们已经在本lab中学会了技术栈）

3.1.1 后端

- Java：https://www.bilibili.com/video/BV1Ca411w7HB/?spm_id_from=333.999.0.0
- Spring：https://www.bilibili.com/video/BV1KT4y1c7Tq/?spm_id_from=333.999.0.0
- Spring MVC：https://www.bilibili.com/video/BV11r4y1F7S4/?spm_id_from=333.999.0.0
- Mybatis：https://www.bilibili.com/video/BV1uy4y1k7LC/?spm_id_from=333.999.0.0
- Spring Boot：http://tengj.top/2017/04/24/springboot0/

3.1.2 前端

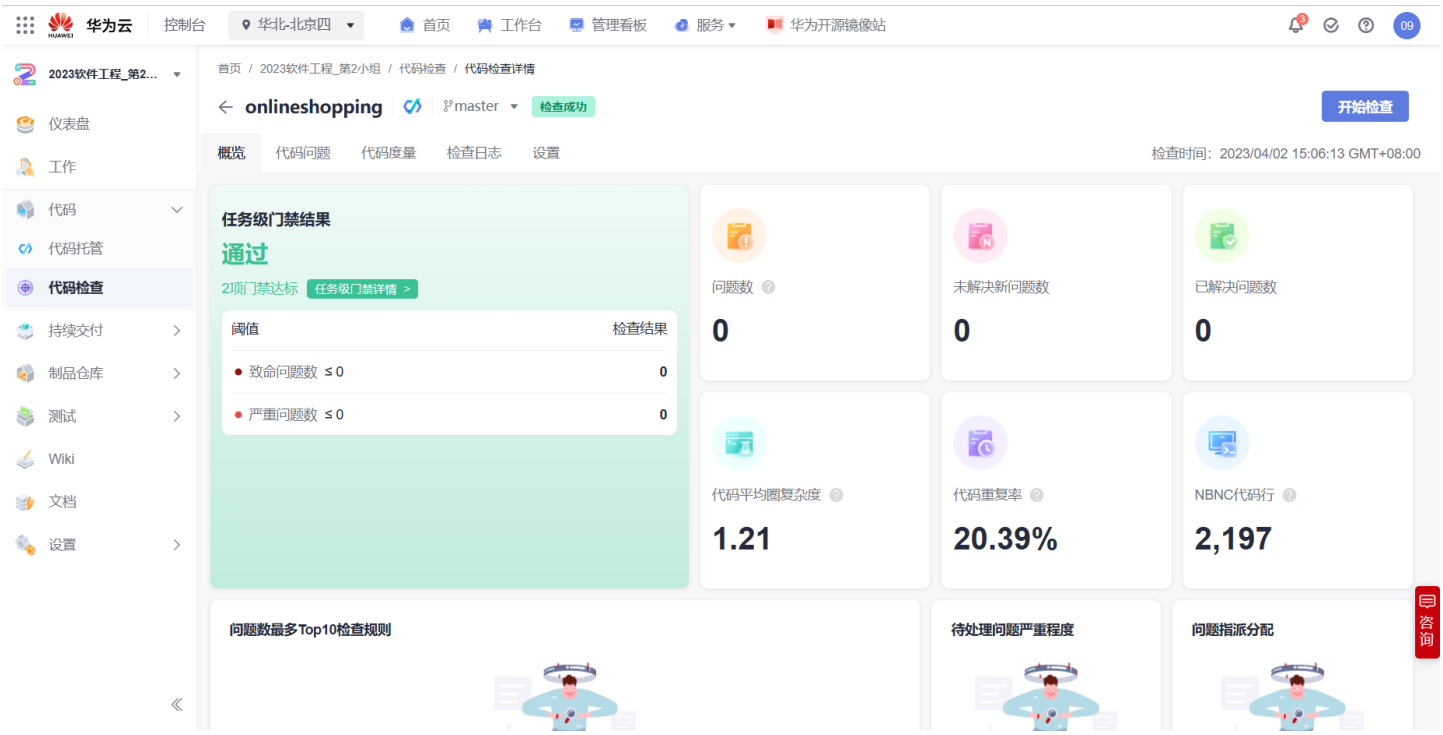
- html和CSS：简单了解语法即可，~~想要好看的页面，肯定是引用外部的CSS；~~还有使用element ui

- js：https://www.bilibili.com/video/BV1P741147CT/?spm_id_from=333.1007.top_right_bar_window_custom_collection.content.click&vd_source=f9f9b6f00b600c2e7217aa2684d20b37
- Ajax：https://www.bilibili.com/video/BV1WC4y1b78y/?spm_id_from=333.337.search-card.all.click&vd_source=f9f9b6f00b600c2e7217aa2684d20b37
- Promise：https://www.bilibili.com/video/BV1GA411x7z1?p=3&vd_source=f9f9b6f00b600c2e7217aa2684d20b37
- Axios：https://www.bilibili.com/video/BV1wr4y1K7tq/?spm_id_from=333.337.search-card.all.click&vd_source=f9f9b6f00b600c2e7217aa2684d20b37。我们主要使用Axios，axios的是基于promise，对ajax的一种封装，可以直接看axios
- Vue基础语法：https://www.bilibili.com/video/BV12J411m7MG/?spm_id_from=333.1007.top_right_bar_window_view_later.content.click
- Vue系统教程：https://www.bilibili.com/video/BV1Zy4y1K7SH?p=1&vd_source=f9f9b6f00b600c2e7217aa2684d20b37

3.2 开发过程

在 `实验设计` 的过程中遇到了诸多问题，并得到了相应的解决方案，详情见上

4 代码检查结果



5 心得体会

林琰钧

我负责的内容是：

- 团队组织：进行人员分工与任务安排、制定开发计划、定期召开线下会议讨论进度
- 文档记录：编写项目开发的约定与安排文档、lab开发进度文档
- 架构设计：设计数据库、前后端接口
- 代码编写：编写部分后端代码

我的心得体会是：

- 在团队开发前，需要有个明确的分工，大家学习各自的技术栈，做到前后端分离
- 在编写代码前，需要有个明确的开发约定与规范，这对项目后期的维护至关重要；同时，需要商量确定好数据库模型与前后端接口，以便进行前后端的整合
- 在团队开发时，需要贯彻敏捷式开发的思想，制定大致的开发计划，定期线上或线下会议讨论进度，并及时解决遇到的问题，让大家能够及时完成各阶段的编码任务

马成

我负责的内容是：

- 数据库模型设计
- 代码编写：编写所有有关商店和商品增删改的代码
- 测试：编写数据库测试文件以及完成单元测试

我的心得体会是：

- 软件开发是团队项目需要多和队友沟通协调才能提升效率。
- 在开始工作之前要先过一遍文档和需求，思考清楚将要实现的代码逻辑以及他们是否合理，都想清楚再写代码事半功倍。
- 在遇到有新的需求或者需求更改的时候要了解清楚需求，快速的定位需要修改的地方。同时注意对重复代码的提炼这样可以在修改的时候快速的修改。
- 一开始可以简要的写一个数据库测试文件插入一些随机数据用于单元测试等，在后续的测试中还需要一个严谨的数据库测试文件。

王兆瀚

我负责的内容是：

- 前后端接口设计
- 前端页面设计
- 代码编写：前端静态页面编写

我的心的体会是：

- 软件开发是一个需要不断迭代完善的过程，一些设计会在实现中发现不合理，或者因为需求改变而被改动
- 前后端分离+Apifox+git可以让团队并行开发，提高效率。
- 统一的接口非常重要，需要团队所有人参与，并从需求和前后端的实现难度等角度提出意见。
- 将任务划分成阶段性的小任务，并及时开会交流进度，可以使项目平滑高效地完成。
- Vue对新手非常友好，配合ELement-UI，可以快速上手前端。

李咸若

我负责的内容是：

- 前后端接口设计
- 前后端串通及测试
- 代码编写：前端界面动态组件编写

我的心得体会是：

- 前后端的第一级在逻辑上需要一定程度的分离，前端页面展示的逻辑和后端处理业务的逻辑拥有很大差距。
- 技术栈的细节在实践中能理解的更加清楚。特别是前端方面，一定要实际编译项目跑起来调试才能搞明白一些内置接口的作用。
- 现代的开发工具如API fox很厉害，能够提供高强度的Mock。
- 小组沟通很重要，要确保对同一接口功能和命名规范的理解上的统一。
- 测试需要考虑各种情况，尤其是边界情况和非法情况，都需要在必要的节点规制。