

3022207128 杨宇鑫 虚拟钱包功能实现

一. 钱包功能需求分析:

1.对于交易的管理,不用直接交易(wx, 支付宝),通过虚拟钱包增加支付的多功能

2.核心功能:充值,提现,支付,查询余额,查询交易,流水(历史查询)

3.其他功能:冻结

4.业务分为两个部分:虚拟钱包系统,第三方支付系统

5.可扩展功能:

5.1. 钱包支持充值和提现功能。为鼓励用户在平台上沉淀资金,一般在充值时应有一定的奖励。相应的,提现时应该将充值时所得的奖励扣除。奖励规则并非固定,应有设置规则的功能。举例:充值时可以满 1000 元赠 100 元,提现时可以扣 10%的手续费,这样设置规则可以避免有人利用系统漏洞“薅羊毛”。

5.2. 钱包需要支持支付、查询余额、查询交易流水等基本功能。支付功能即从一个钱包向另一个钱包转账,注意支付过程要保证交易的原子性。

(商家钱包和用户钱包)

5.3. 根据系统功能需要,钱包也需要支持冻结和透支等功能。例如,买家向卖家完成支付后,货款虽已到卖家账户,但是是冻结状态,需要买家确认收货后方可解冻。对于VIP用户,可以设置一定的透支额度,透支金额如超过一定时间不归还的话,则需要支付一定利息。(类似小额贷款或者花呗、白条等功能)

6.交易流水+数据库的设计:



7. 提交方式:

仓库中需要提交代码和实验报告。代码放在/src 目录,代码应为完整的项目代码,包括饿了么项目的原有代码和新增代码。

实验报告放在/doc 目录,实验报告命名规则:“学号-姓名-程序设计中级实践-实验报告 1.pdf”,例如“3022244888-李四-程序设计中级实践-实验报告 1.pdf”。

实验报告应包含且仅包含本次作业所要求的功能实现的全部文档内容,即,不必包含饿了么项目原来的需求、设计等文档内容。实验报告的内容主要应包括:

(1)增加钱包功能的需求分析

(2)将钱包功能增加到原项目中的方案设计

(3)代码实现、单元测试和集成测试的过程

二. 实践过程:

1.代码重构与升级:

1.1 引入 Knife4j（使用 Swagger 的功能）

Swagger 主要用于 API 的文档生成和接口测试。它是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。Swagger 能够自动生成一个在线的接口文档，展示接口的功能、所需参数和返回数据，方便开发人员和测试人员理解和使用接口。Swagger 还提供了接口测试功能，允许用户直接在文档界面上进行接口测试，以验证接口的正确性。

Swagger 的测试功能主要包括：

接口文档的自动生成：Swagger 可以根据代码中的注解自动生成接口文档，这些文档包括接口的 URL、请求方法、参数、请求体和返回值等信息。

接口测试：Swagger 提供了一个用户界面，允许用户直接在浏览器中测试接口，输入参数并发送请求，然后查看接口的响应结果。

参数校验：Swagger 支持对接口参数进行校验，确保传入的参数符合接口定义的要求。

多环境配置：Swagger 允许为不同的环境（如开发、测试和生产环境）配置不同的 API 文档和测试接口。

Swagger 的使用不仅限于测试，它还有助于前后端分离的开发模式，通过提供清晰的接口文档，促进了前后端开发者之间的沟通和协作。此外，Swagger 还支持多种语言和框架，使其成为 API 开发和测试领域广泛使用的工具之一。

Knife4j是一个集Swagger2 和 OpenAPI3 为一体的增强解决方案

yml 文件配置:

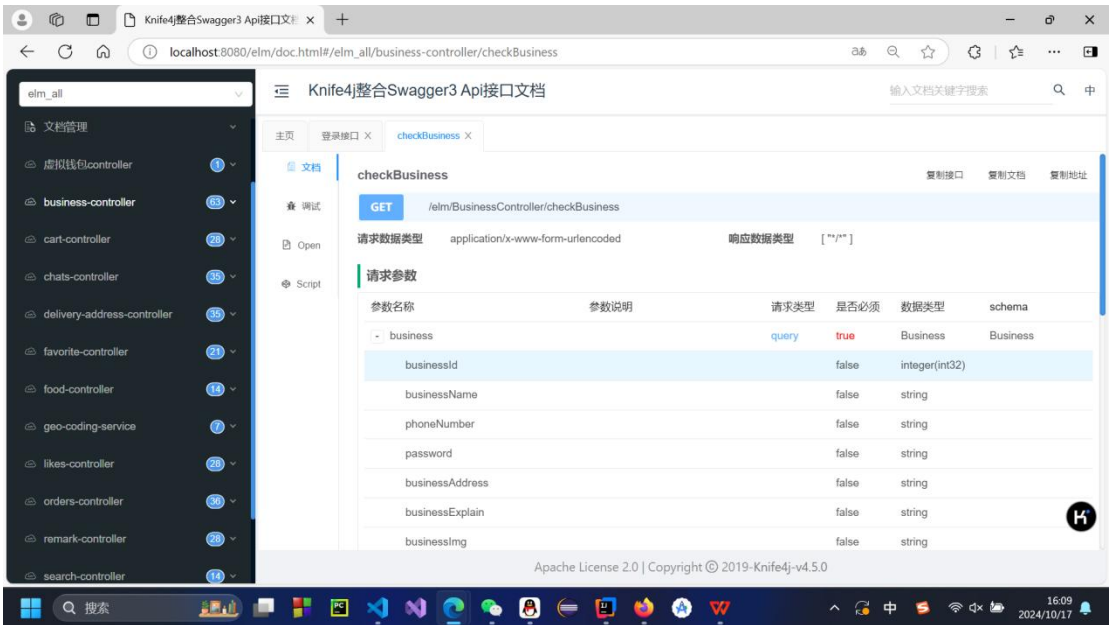
```
# springdoc-openapi项目配置
springdoc:
  swagger-ui:
    #自定义swagger前端请求路径,输入http://localhost:8080/swagger-ui会自动重定向到swagger页面
    path: /swagger-ui
    tags-sorter: alpha
    operations-sorter: alpha
  api-docs:
    path: /v3/api-docs #swagger后端请求地址
    enabled: true #是否开启文档功能
  group-configs: #分组配置,可配置多个分组
    - group: 'elm_plus'
      paths-to-match: '/plus/**'
      packages-to-scan: com.neusoft.elmboot
    - group: 'elm_all' #分组名称
      paths-to-match: '/**' #配置需要匹配的路径
      packages-to-scan: com.neusoft.elmboot #配置要扫描包的路径,一般配置到启动类所在的包名

knife4j:
  enable: true #开启knife4j,无需添加@EnableKnife4j注解
  setting:
    language: zh_cn #中文
    swagger-model-name: 实体列表
```

起步依赖引入:

```
<dependency>
  <groupId>com.github.xiaoymin</groupId>
  <artifactId>knife4j-openapi3-jakarta-spring-boot-starter</artifactId>
  <version>4.5.0</version>
</dependency>
```

完成效果:



2.2 引入 Lombok 减少样板代码，简化开发

Lombok 是一个 Java 库，它通过注解的方式简化了 Java 代码的编写。它自动地为 `getter`、`setter`、`toString`、`equals`、`hashCode` 等方法生成模板化的代码，从而减少了手动编写这些常见方法的工作量。Lombok 的核心目标是减少样板代码 (boilerplate code)，让开发者能够更专注于业务逻辑的实现。

以下是 Lombok 的一些主要功能和特点：

自动生成 `getter` 和 `setter` 方法：通过在字段上使用 `@Getter` 和 `@Setter` 注解，Lombok 会自动为你生成这些方法。

自动生成 `toString` 方法：使用 `@ToString` 注解，Lombok 会生成一个包含类中所有字段的 `toString` 方法。

自动生成 `equals` 和 `hashCode` 方法：通过 `@EqualsAndHashCode` 注解，Lombok 会为你生成这两个方法，通常用于比较对象的相等性和计算哈希值。

构造函数：使用 `@NoArgsConstructor`、`@AllArgsConstructor` 和 `@RequiredArgsConstructor` 注解，Lombok 可以生成不同参数的构造函数。

日志注解：Lombok 提供了 `@Slf4j` 和其他日志框架的注解，自动为你的类生成日志对象。

数据验证注解：例如 `@NonNull`、`@Size` 等，这些注解可以用于验证方法参数。

其他注解：如 `@Value`、`@Builder`、`@Data` 等，这些注解提供了更高级的功能，比如不可变对象、构建器模式等。

起步依赖引入:

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

完成效果:

```
@Data
@Schema(description = "用户钱包")
public class UserWallet {

    @Schema(description = "用户ID")
    private String userId;

    @Schema(description = "余额")
    private BigDecimal balance;

    @Schema(description = "是否冻结")
    private String freeze;

    @Schema(description = "上次更新时间")
    private LocalDateTime updateLast;
}
```

2.3 将 MyBatis 升级为 MyBatisPlus

MyBatis-Plus（简称 **MP**）是一个 **MyBatis** 的增强工具，在 **MyBatis** 的基础上只做增强不做改变，为简化开发、提高效率而生。它继承了 **MyBatis** 的所有特性，并且加入了强大的功能，比如自动填充、逻辑删除、性能分析等。以下是 **MyBatis-Plus** 的一些主要特点：
代码生成器：提供代码生成器，可以快速生成 **Entity**、**Mapper**、**Mapper XML**、**Service**、**Controller** 等代码，极大地减少了手动编写的重复代码。

内置功能：提供了内置的 **CRUD** 操作，开发者无需编写 **SQL** 语句，即可实现基本的增删改查功能。

自动填充字段：支持字段的自动填充功能，可以在插入或更新操作时自动填充创建时间、更新时间等字段。

逻辑删除：支持逻辑删除，通过一个字段标识记录是否被删除，而不是从数据库中真正删除记录。

性能分析：提供了 **SQL** 性能分析插件，可以输出每条 **SQL** 的执行时间，帮助开发者分析和优化 **SQL** 性能。

乐观锁与悲观锁：支持乐观锁和悲观锁，帮助处理并发更新的问题。

条件构造器：提供了强大的条件构造器 **Wrapper**，可以方便地构建各种复杂条件的查询。

分页插件：内置了分页插件，支持多种数据库的分页查询。

多租户插件：支持多租户场景，可以方便地在查询时添加租户 **ID** 作为条件。

事务支持：继承了 **MyBatis** 的事务管理功能，支持声明式事务。

MyBatis-Plus 通过减少模板代码，让开发者可以更专注于业务逻辑的实现，从而提高开发效率。同时，它也保持了 MyBatis 的灵活性，允许开发者在需要时编写自定义的 SQL 语句。

起步依赖引入：

```
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>2.2.0</version>
</dependency>
```

yaml 文件配置：

```
#配置mybatis-plus的一些文件
mybatis-plus:
  typeAliasesPackage: com.neusoft.elmboot.po
  mapperLocations: classpath:mapper/*.xml
# 配置sql注入关键字
sql:
  keywords: ;|'|\"|_|-|_|*|_|%|#|_|/|_|\\|_|=|_|or|_|and|_|like|_|select|_|insert|_|update|_|delete|_|alert|_|drop|_|truncate|_|declare|_|exec|_|execu
```

2.虚拟钱包-数据库设计：

2.1. 用户钱包表：

存储用户的钱包信息，包括用户 ID，可用余额，是否冻结（0：），上次更新时间。
表结构：

user_id	balance	freeze	update_last
---------	---------	--------	-------------

建表语句：

```
CREATE TABLE `user_wallet` (
  `user_id` char(32) NOT NULL COMMENT '用户user_id',
  `balance` decimal(10,2) unsigned DEFAULT '0.00' COMMENT '钱包总可用余额',
  `freeze` varchar(1) DEFAULT '0' COMMENT '0:未冻结 1:处于冻结状态',
  `update_last` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户钱包';
```

建表语句（可复制）：

```
CREATE TABLE `user_wallet` (
  `user_id` char(32) NOT NULL COMMENT '用户 user_id',
  `balance` decimal(10,2) unsigned DEFAULT '0.00' COMMENT '钱包总可用余额',
  `freeze` varchar(1) DEFAULT '0' COMMENT '0:未冻结 1:处于冻结状态',
  `update_last` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户钱包';
```

2.2. 交易流水记录表：

记录每一笔交易的详细信息，包括流水号、用户 ID、业务类型、对方 ID（可以没有，

根据业务类型判断)、变动金额(正负数), 创建时间
表结构:

id	user_id	target_type	target_id	fee	create_time
----	---------	-------------	-----------	-----	-------------

建表语句:

```
CREATE TABLE `wallet_log` (  
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'auto id 流水号',  
  `user_id` char(32) DEFAULT '' COMMENT '用户id',  
  `target_type` smallint(5) unsigned DEFAULT '0' COMMENT '业务类型, 1: 充值, 2: 提现 3: 支付',  
  `target_id` char(32) DEFAULT '' COMMENT '来源id(如果有)',  
  `fee` decimal(10,2) DEFAULT '0.00' COMMENT '变动的金额, 正负数。',  
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='钱包流水记录表';
```

建表语句(可复制):

```
CREATE TABLE `wallet_log` (  
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'auto id 流水号',  
  `user_id` char(32) DEFAULT '' COMMENT '用户id',  
  `target_type` smallint(5) unsigned DEFAULT '0' COMMENT '业务类型, 1: 充值, 2: 提现 3: 支付',  
  `target_id` char(32) DEFAULT '' COMMENT '来源id(如果有)',  
  `fee` decimal(10,2) DEFAULT '0.00' COMMENT '变动的金额, 正负数。',  
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='钱包流水记录表';
```

2.3. 查询优化: 对于数据库查询性能的优化, 可以采用索引、查询优化器提示(HINT)、并行查询、调整性能参数等方法。

3. 虚拟钱包-实体类 domain 设计:

3.1 PO:

3.1.1 UserWallet 用户钱包表

```
@Data no usages new *  
@Schema(description = "用户钱包")  
public class UserWallet {  
  
  @Schema(description = "用户ID")  
  private String userId;  
  
  @Schema(description = "余额")  
  private BigDecimal balance;  
  
  @Schema(description = "是否冻结")  
  private String freeze;  
  
  @Schema(description = "上次更新时间")  
  private LocalDateTime updateLast;  
}
```

3.1.2 WalletLogAll 钱包流水记录表

```
@Data new *
@Schema(description = "钱包流水记录")
public class WalletLog {
    @Schema(description = "流水号")
    private Integer id;
    @Schema(description = "用户ID")
    private String userId;
    @Schema(description = "业务类型")
    private Integer targetType;
    @Schema(description = "对方ID")
    private String targetId;
    @Schema(description = "变动金额")
    private BigDecimal fee;
    @Schema(description = "创建时间")
    private LocalDateTime createTime;
}
```

3.2 vo

3.2.1 WalletFlowVo 钱包流水

```
@Data 8 usages new *
@Schema(description = "钱包流水")
public class WalletFlowVo {
    @Schema(description = "流水号")
    private Integer id;
    @Schema(description = "用户ID")
    private String userId;
    @Schema(description = "业务类型")
    private Integer targetType;
    @Schema(description = "对方ID")
    private String targetId;
    @Schema(description = "变动金额")
    private BigDecimal fee;
    @Schema(description = "创建时间")
    private LocalDateTime createTime;
}
```

3.3 dto

3.3.1 WalletFlowDto 钱包流水

```
@Data 10 usages new *
@Schema(description = "钱包流水")
public class WalletFlowDto {
    @Schema(description = "用户ID")
    private String userId;
    @Schema(description = "金额")
    private BigDecimal amount;
    @Schema(description = "对方ID")
    private String targetId;
}
```

4.虚拟钱包-api 接口（后端代码实现）：

3.0 场景：用户点击申请**创建**自己的**虚拟钱包**

后端 api:

请求方式: **POST**

请求路径: /elm/plus/VirtualWalletController/addWalletById

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

userId=0

响应数据: 数据库中影响的行数

3.1 场景：用户**是否开启**了虚拟钱包功能

后端 api:

请求方式: **GET**

请求路径: /elm/plus/VirtualWalletController/getWhetherEnabledById

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

userId=0

响应数据: **0**: 未开启 **1**: 已经开启了

3.2 场景：**充值**功能

后端 api:

请求方式: **POST**

请求路径: /elm/plus/VirtualWalletController/rechargeById

请求类型: application/json

请求数据: userId amount

请求示例:

```
{
  "userId" :0,
  "amount" :100
}
```

响应数据: 用户当前的总积分

3.3 场景：**提现**功能

后端 api:

请求方式: **POST**

请求路径: /elm/plus/VirtualWalletController/withdrawById

请求类型: application/json

请求数据: userId amount

请求示例:

```
{
  "userId" :0,
  "amount" :100
}
```


}

响应数据：实际提现的金额（扣除了 10%的手续费）

3.4 场景：支付/转账功能

后端 api:

请求方式：POST

请求路径：/elm/plus/VirtualWalletController/pay

请求类型：application/json

请求数据：userId amount targetId

请求示例：

```
{
  "userId" :0,
  "amount" :100
  "targetId" :1
}
```

响应数据：0：未成功

其他值：当前用户的总积分

3.5 场景：查询余额功能

后端 api:

请求方式：GET

请求路径：/elm/plus/VirtualWalletController/getBalanceById

请求类型：application/x-www-form-urlencoded

请求数据：userId

请求示例：

userId=0

响应数据：balance

3.6 场景：查询历史交易（流水）功能

后端 api:

请求方式：GET

请求路径：/elm/plus/VirtualWalletController/listWalletLogById

请求类型：application/x-www-form-urlencoded

请求数据：userId

请求示例：

userId=0

响应数据：WalletLog 数组

3.7 场景：查询历史收入（流水）功能

后端 api:

请求方式：GET

请求路径：/elm/plus/VirtualWalletController/listWalletLogIncomeById

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

userId=0

响应数据: WalletLog 数组

3.7 场景: 查询历史支出 (流水) 功能

后端 api:

请求方式: GET

请求路径: /elm/plus/VirtualWalletController/listWalletLogOutcomeById

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

userId=0

响应数据: WalletLog 数组

3.8 场景: 查询用户的虚拟钱包是否冻结

后端 api:

请求方式: GET

请求路径: /elm/plus/VirtualWalletController/getFreeze

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

userId=0

响应数据: 0: 未冻结 1: 已经冻结

3.9 场景: 冻结用户的虚拟钱包

后端 api:

请求方式: POST

请求路径: /elm/plus/VirtualWalletController/freeze

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

userId=0

响应数据: 影响的行数

3.10 场景: 解冻用户的虚拟钱包

后端 api:

请求方式: POST

请求路径: /elm/plus/VirtualWalletController/unFreeze

请求类型: application/x-www-form-urlencoded

请求数据: userId

请求示例:

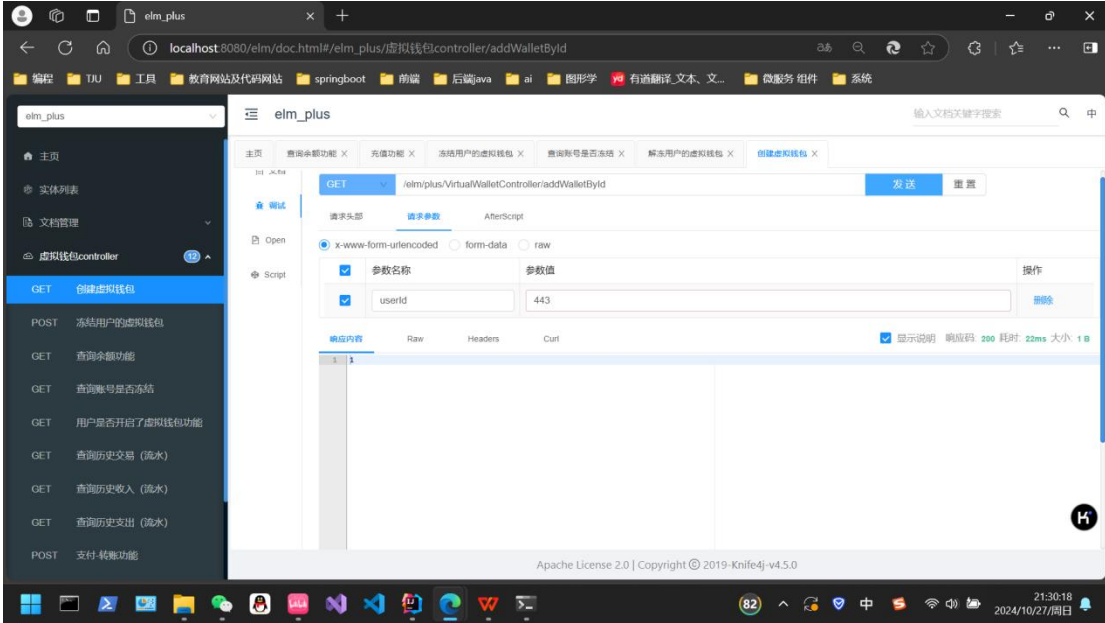
userId=0

响应数据：影响的行数

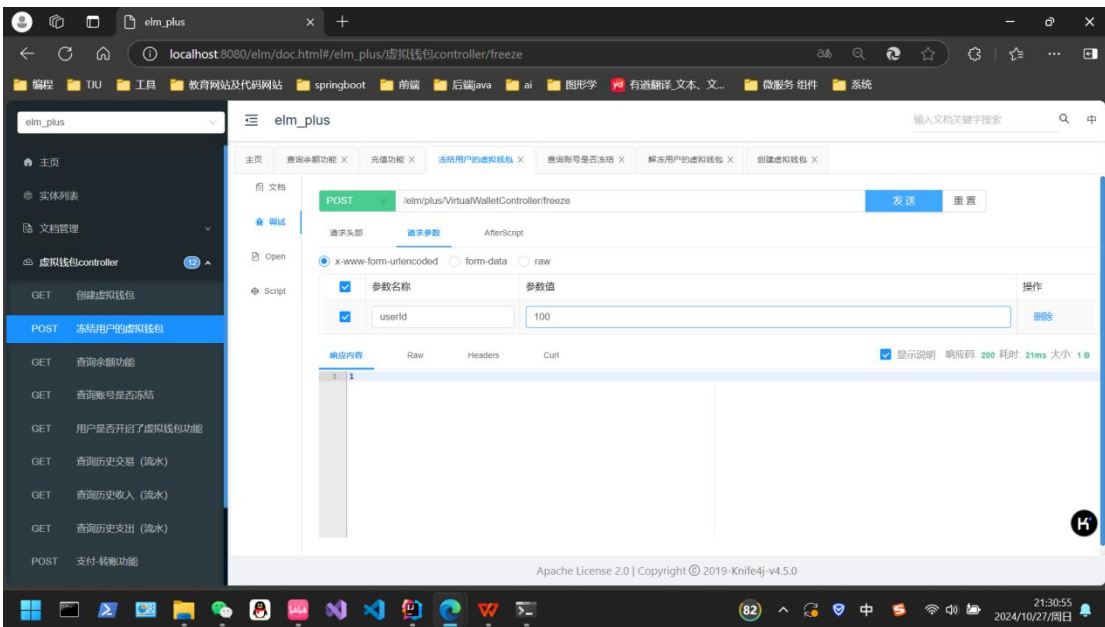
三. 测试：

通过 Knife4j（swagger）测试所有接口，圆满成功

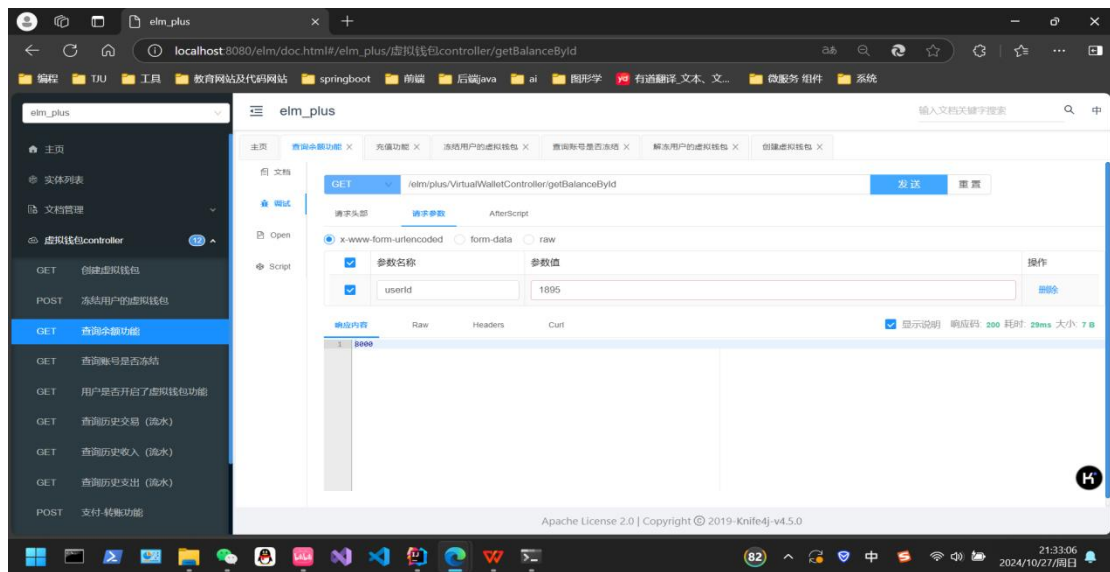
3.1 创建虚拟钱包接口（返回状态码 200 成功）：



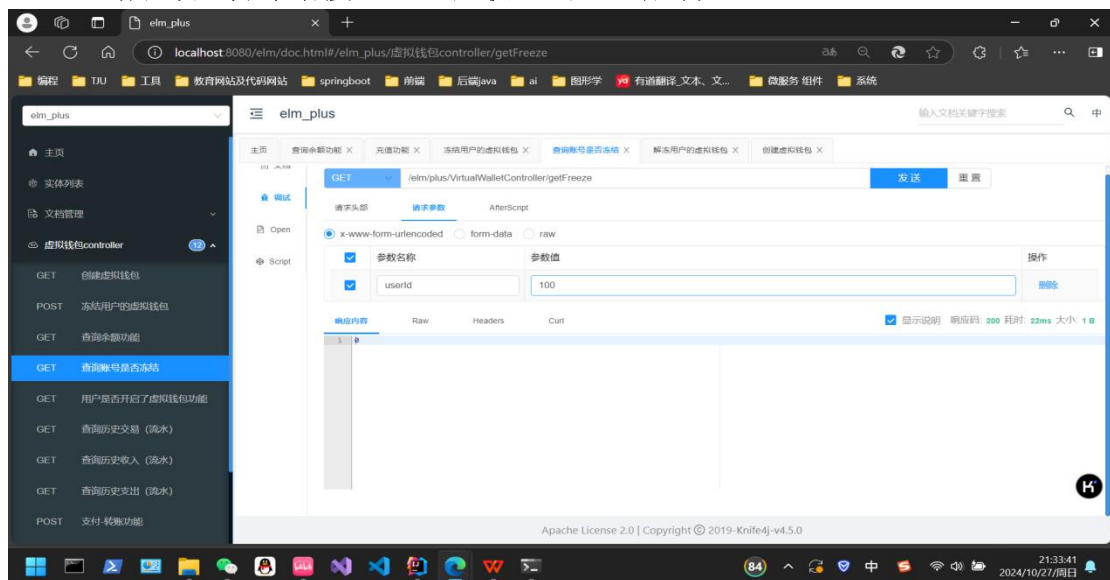
3.2 冻结用户钱包接口（返回状态码 200 成功）：



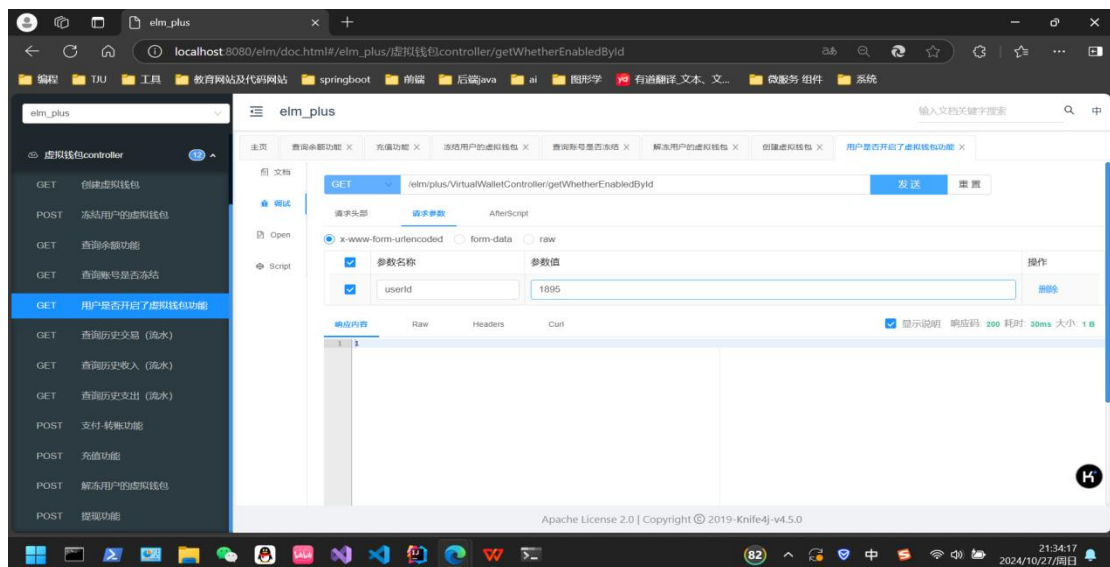
3.3 查询余额接口（返回状态码 200 成功）：



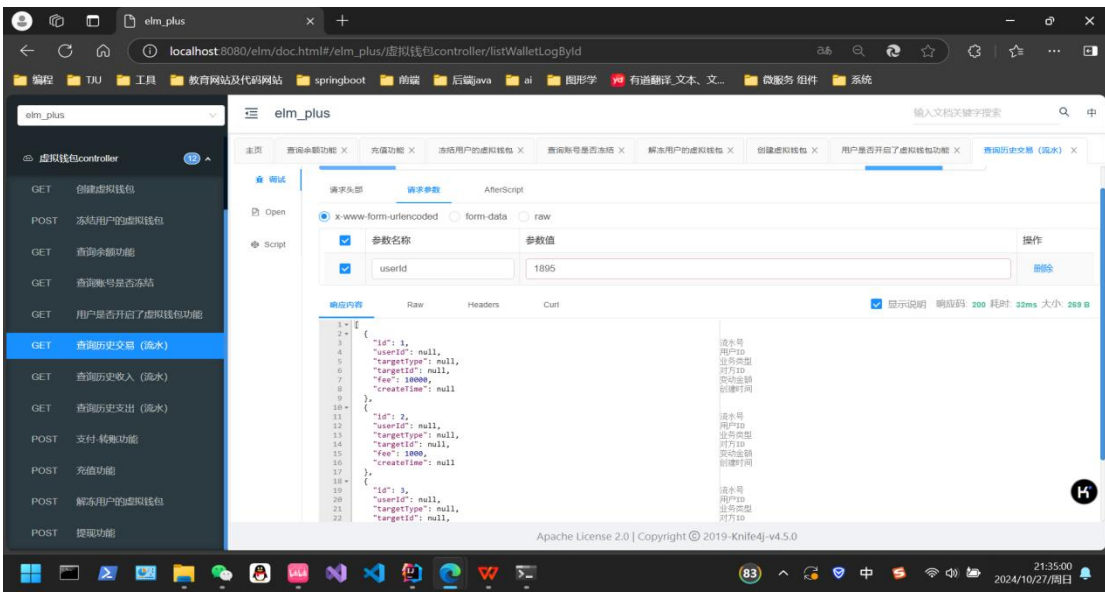
3.4 查询账号是否冻结接口（返回状态码 200 成功）：



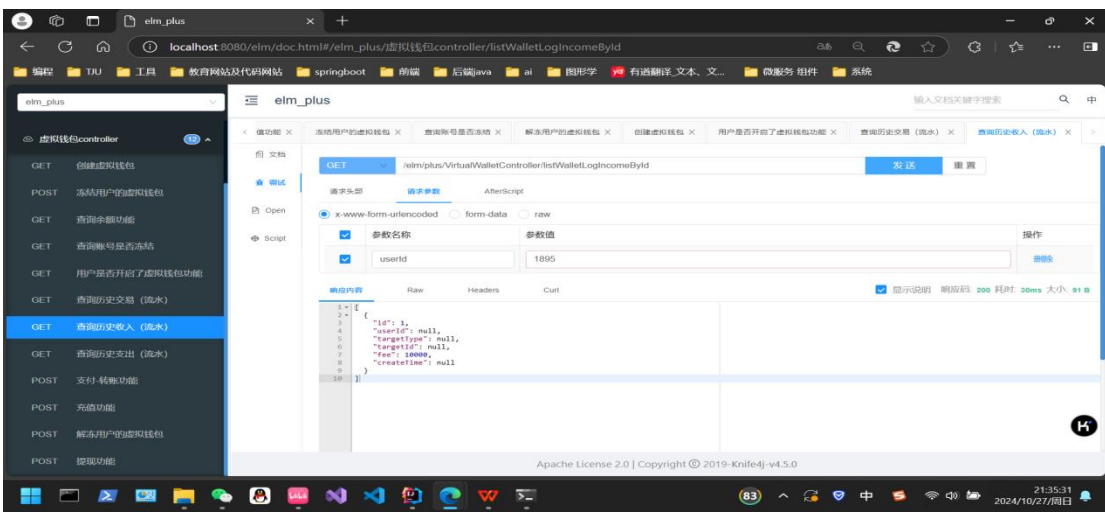
3.5 用户是否开启虚拟钱包功能（返回状态码 200 成功）：



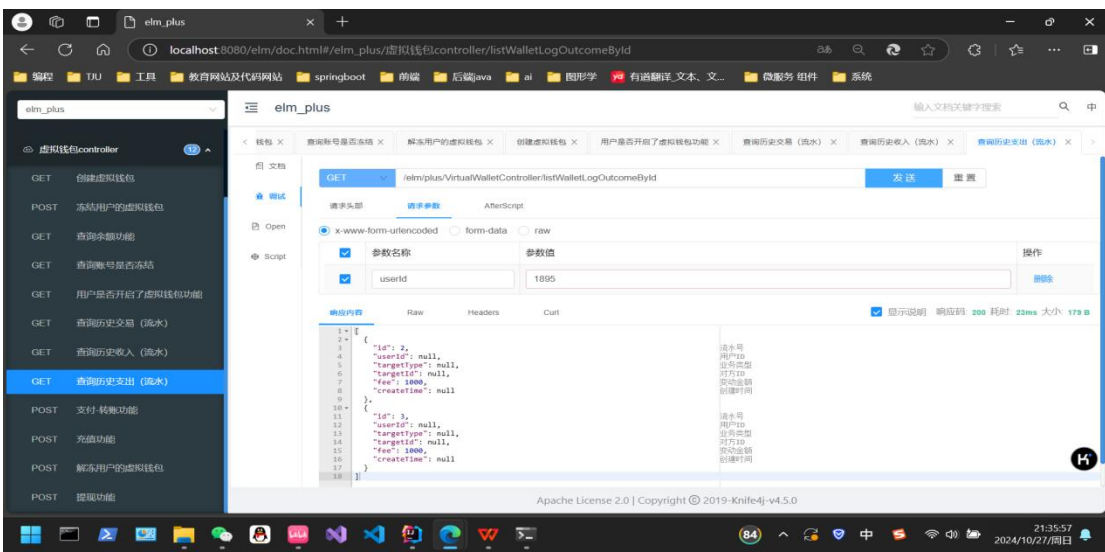
3.6 查看历史交易接口（返回状态码 200 成功）：



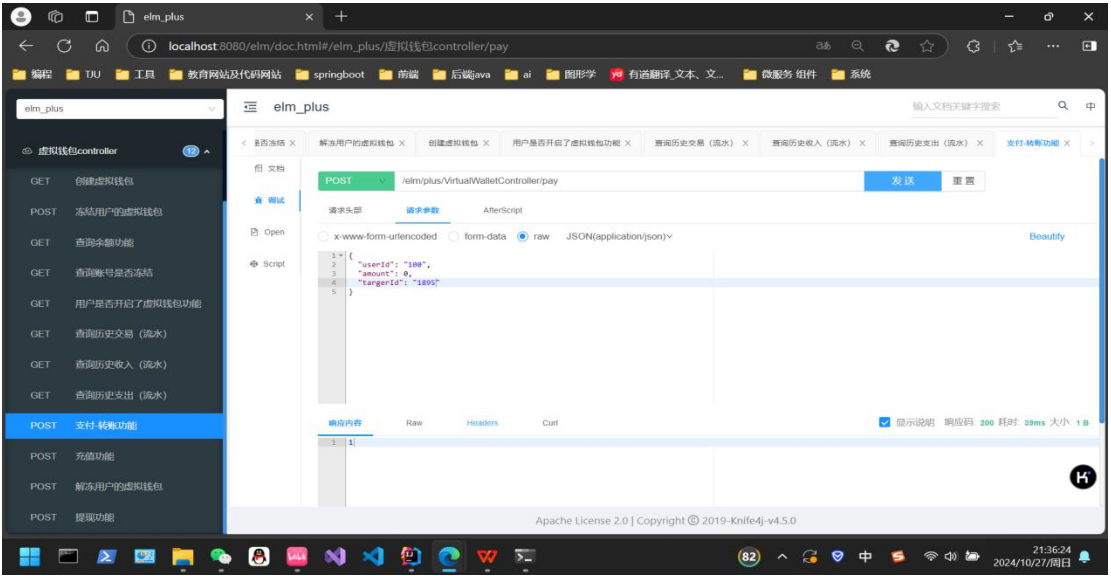
3.7 查看历史收入接口（返回状态码 200 成功）：



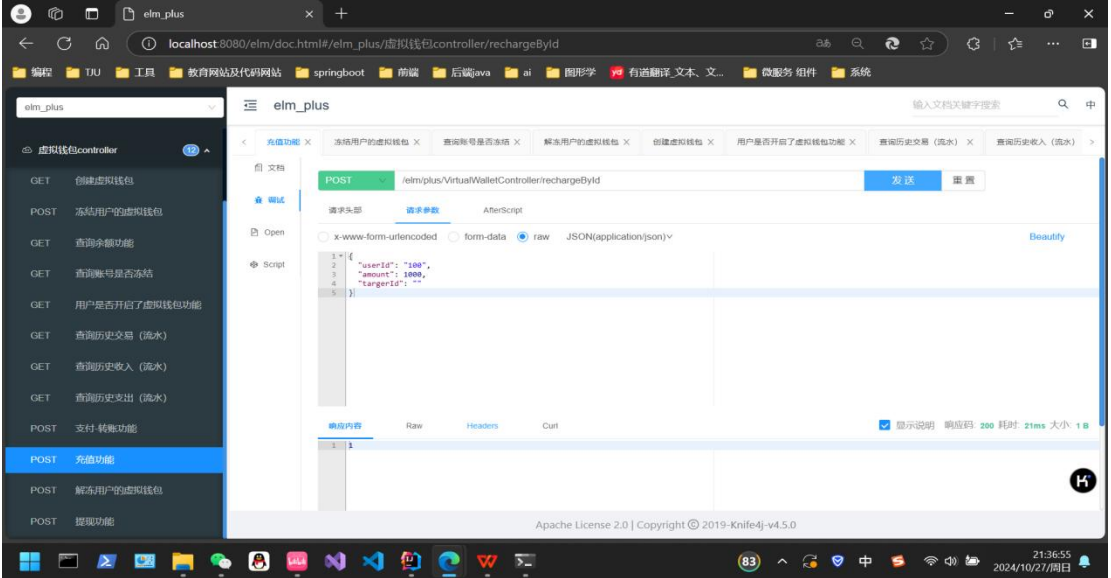
3.8 查看历史支出接口（返回状态码 200 成功）：



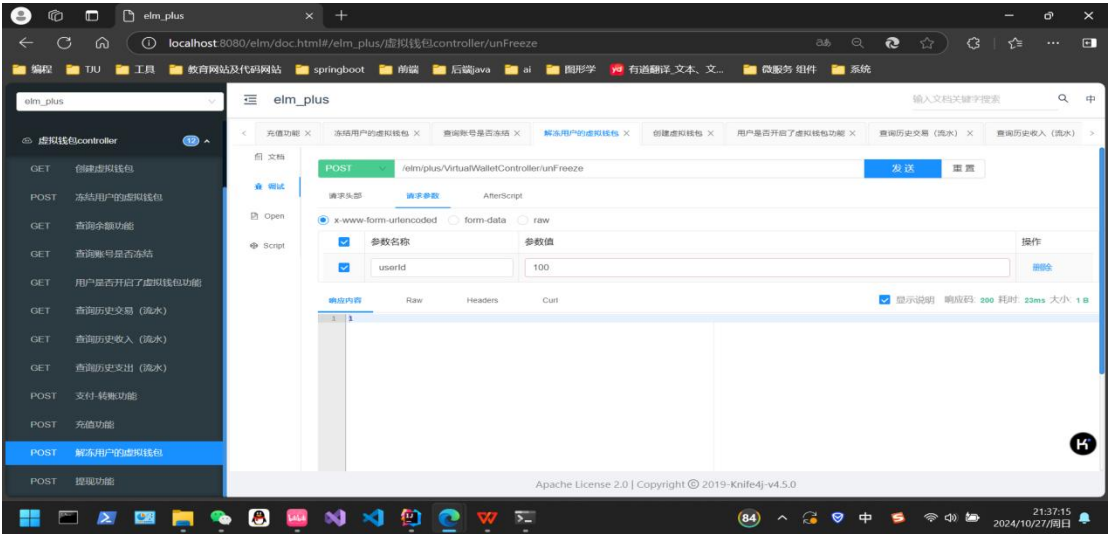
3.9 支付/转账功能（返回状态码 200 成功）：



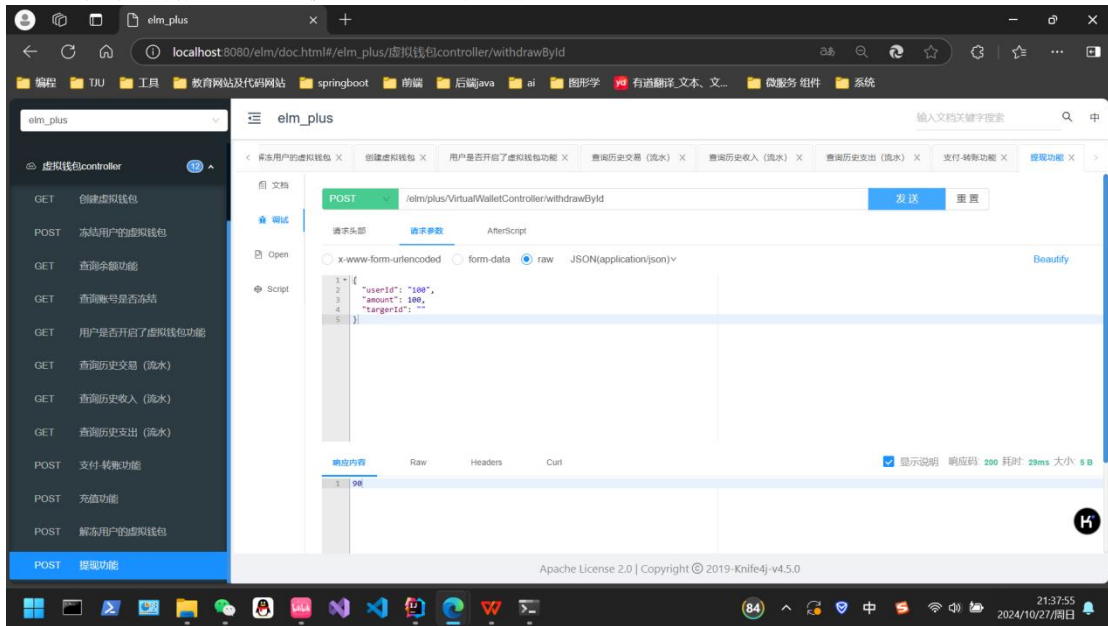
3.10 充值功能（返回状态码 200 成功）：



3.11 解冻用户的虚拟钱包（返回状态码 200 成功）：



3.12 提现功能（返回状态码 200 成功）：



四. 遇到的问题以及解决方案：

3.1.如何账号冻结了，再收到请求，怎么办呢？

约定大于配置，约定：冻结的钱包只能使用“查询是否冻结”和“解冻”api，其他情况要使用拦截器统一处理直接抛异常。

注：因为前端在调用api的时候首先应该“查询是否冻结”再进行操作，不会出现这种抛异常的情况，但是测试的时候可能会绕过查询直接对钱包进行操作，这时拦截器就发挥作用了，不能让已经冻结的钱包成功操作数据，直接报错返回。

拦截器：

```
@Component
public class WalletIsFrozenInterceptor implements HandlerInterceptor {

    @Autowired
    private VirtualWalletMapper virtualWalletMapper;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        // 从请求中获取需要的参数，比如用户ID
        String userId = request.getParameter("userId");

        if (!userId == null || userId.isEmpty()) {
            int freeze = virtualWalletMapper.getFreeze(userId);
            if(freeze==1){
                throw new Exception();
            }
            return true;
        }

        WalletFlowDto walletFlowDto = (WalletFlowDto) request.getAttribute("walletFlowDto");
        if (walletFlowDto != null) {
            userId = walletFlowDto.getUserId();
            int freeze = virtualWalletMapper.getFreeze(userId);
            if(freeze==1){
                throw new Exception();
            }
            return true;
        }
        return true;
    }
}
```

拦截器配置:

```
package com.neusoft.elmboot.config;

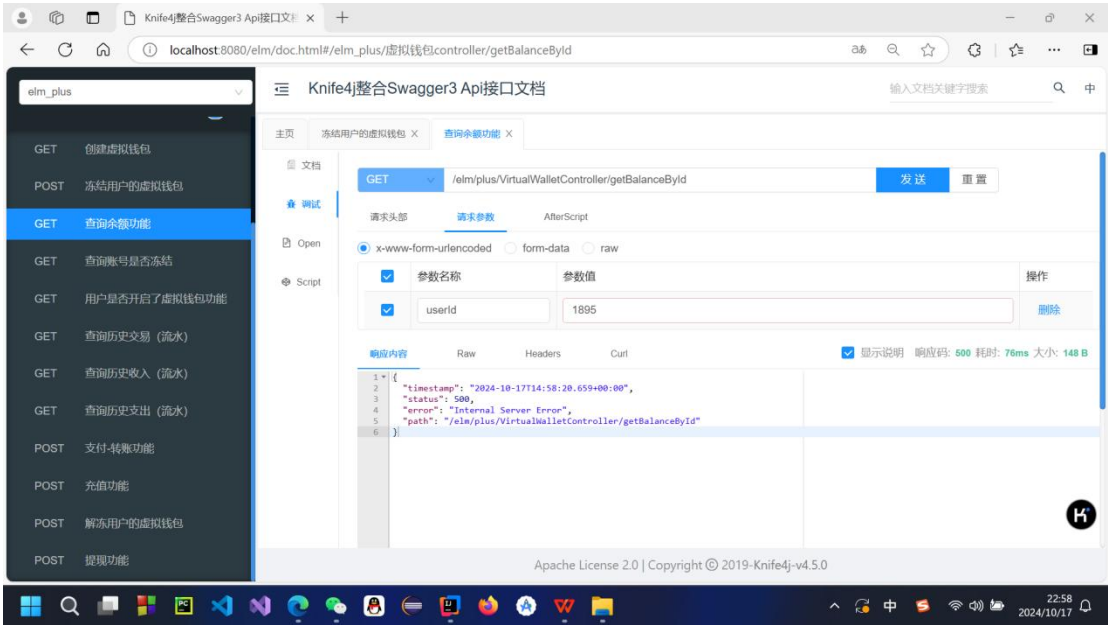
import com.neusoft.elmboot.intercept.WalletIsFrozenInterceptor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class InterceptorConfig implements WebMvcConfigurer {
    @Autowired
    private WalletIsFrozenInterceptor walletIsFrozenInterceptor;

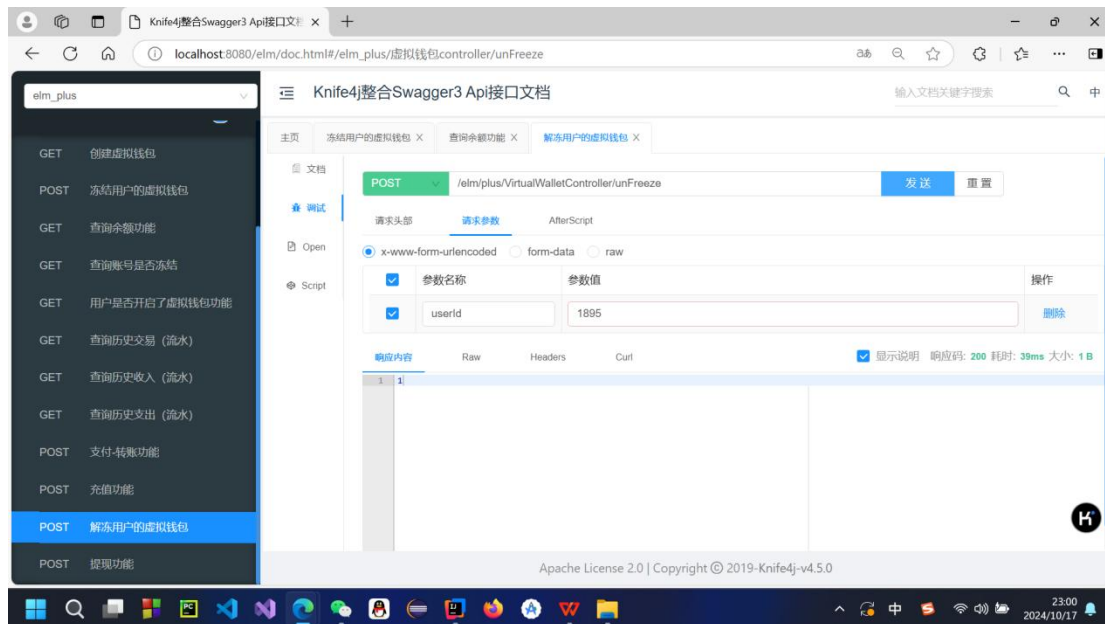
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(walletIsFrozenInterceptor)
            .addPathPatterns("/plus/**")
            .excludePathPatterns("/plus/VirtualWalletController/unFreeze")
            .excludePathPatterns("/plus/VirtualWalletController/getFreeze");
    }
}
```

成果:

①若钱包已冻结，则操作统一返回 500 错误



②. 若执行的是“查询是否冻结”和“解冻” api，则操作成功



3.2.如何用户没有创建钱包，怎么办呢？

约定大于配置，约定：如果用户没有创建钱包，则只能调用“创建钱包”功能，使用其他功能都直接报错，思路和 3.1 一样，采用 AOP 编程思想，使用拦截器。

配置拦截器：

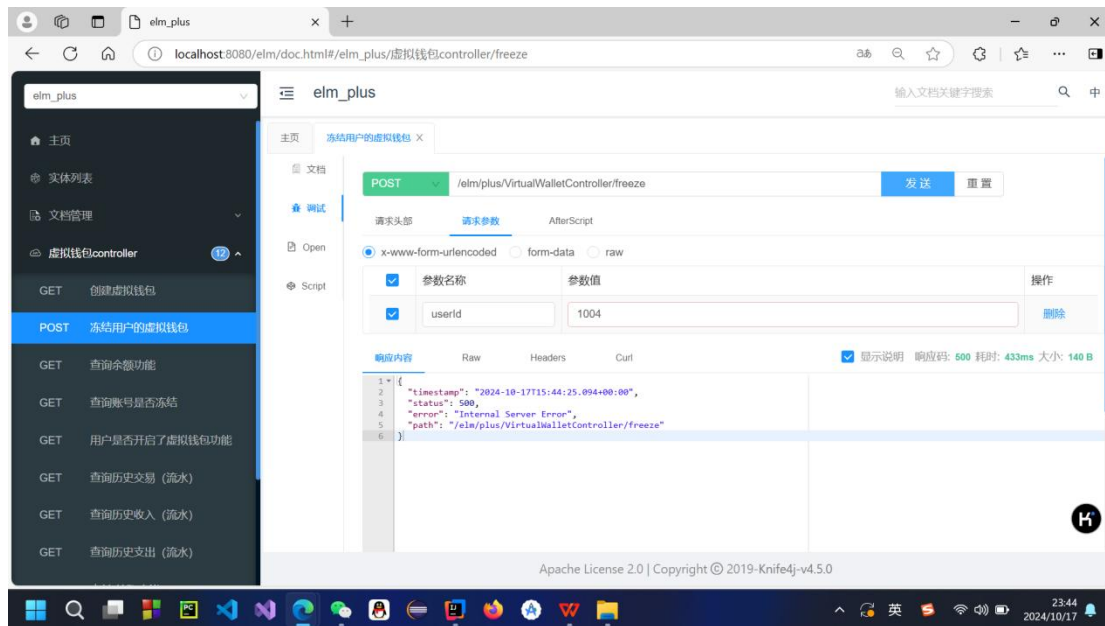
```
registry.addInterceptor(walletNotFoundInterceptor)
    .addPathPatterns("/plus/**")
    .excludePathPatterns("/plus/VirtualWalletController/addWalletById");
```

拦截器：

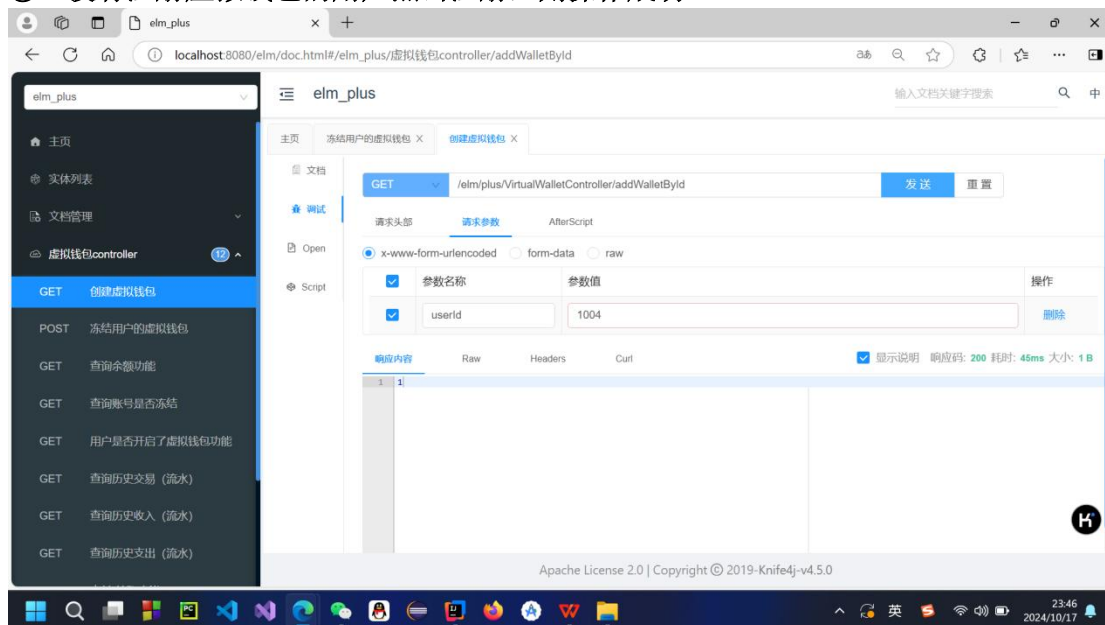
```
10
11 @Component new "
12 public class WalletNotFoundInterceptor implements HandlerInterceptor {
13     @Autowired
14     private VirtualWalletMapper virtualWalletMapper;
15     @Override no usages new "
16     public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
17
18         String userId = request.getParameter("userId");
19
20         if (!userId == null || userId.isEmpty()) {
21             int flag = virtualWalletMapper.getCountById(userId);
22             if(flag==0){
23                 throw new Exception();
24             }
25             return true;
26         }
27         WalletFlowDto walletFlowDto = (WalletFlowDto) request.getAttribute("walletFlowDto");
28         if (walletFlowDto != null) {
29             userId = walletFlowDto.getUserId();
30             int flag = virtualWalletMapper.getCountById(userId);
31             if(flag==0){
32                 throw new Exception();
33             }
34             return true;
35         }
36         return true;
37     }
38
39 }
```

成果：

①. 若用户没有注册虚拟钱包，则除了“注册虚拟钱包”的所有操作报错



②. 没有注册虚拟钱包的用户点击注册，则操作成功



3.3.关于转账的问题:

- ①支付/转账对方不存在问题
- ②不能转账给自己
- ③转账人钱不够的问题


```

public int pay(WalletFlowDto walletFlowDto) {
    //转账人钱不够
    if(virtualWalletMapper.getAmountById(walletFlowDto).compareTo(walletFlowDto.getAmount()) < 0){
        return 0;
    }
    //不能转账给自己
    if(walletFlowDto.getUserId()==walletFlowDto.getTargetId()){
        return 0;
    }
    //支付对方不存在的问题
    if(virtualWalletMapper.getCountById(walletFlowDto.getTargetId()) ==0 ){
        return 0;
    }
}

```

五. 扩展功能实现:

5.1 充值时可以满 1000 元赠 100 元

```

@Override 1 usage
@Transactional
public int rechargeById(WalletFlowDto walletFlowDto) {
    //充值时可以满 1000 元赠 100 元
    BigDecimal amount = walletFlowDto.getAmount();
    amount=amount.add(amount.divide(BigDecimal.valueOf(1000))
        .setScale( newScale: 0, RoundingMode.DOWN).multiply(BigDecimal.valueOf(100)));
    walletFlowDto.setAmount(amount);
    virtualWalletMapper.addRechargeFlow(walletFlowDto);
    return virtualWalletMapper.rechargeById(walletFlowDto);
}

```

5.2 提现时可以扣 10%的手续费

```

@Override 1 usage
@Transactional
public BigDecimal withdrawById(WalletFlowDto walletFlowDto) {
    if(virtualWalletMapper.getAmountById(walletFlowDto).compareTo(walletFlowDto.getAmount()) < 0){
        return BigDecimal.valueOf(0);
    }
    virtualWalletMapper.addWithdrawFlow(walletFlowDto);
    virtualWalletMapper.withdrawById(walletFlowDto);
    BigDecimal amount = walletFlowDto.getAmount();
    return amount.multiply(BigDecimal.valueOf(0.9));
}

```

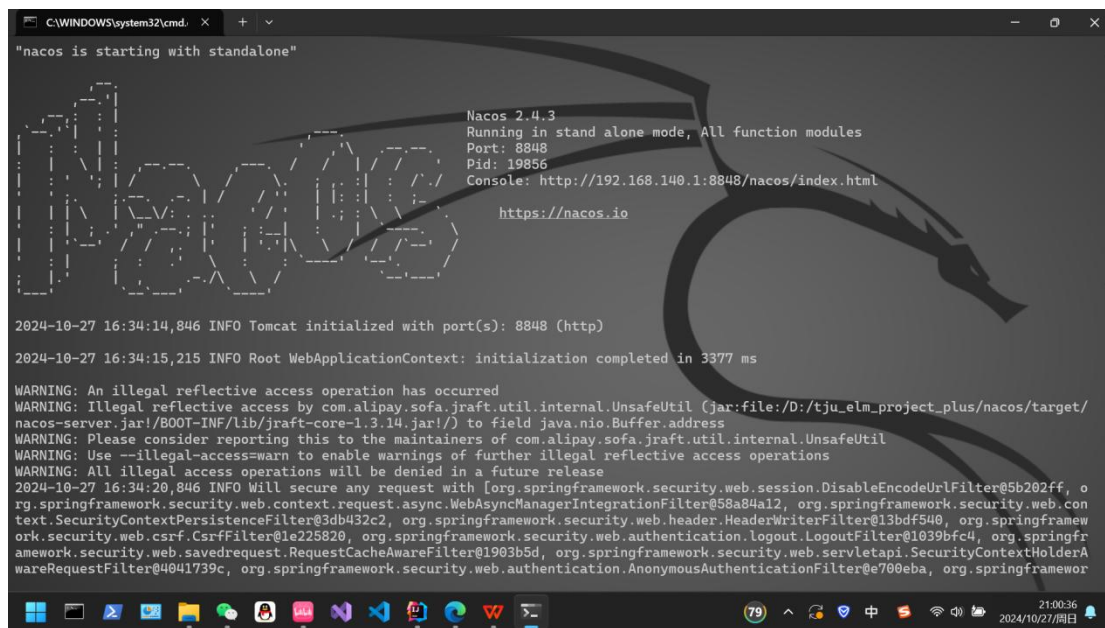
5.3 实现配置的热更新

方式: 使用 **Nacos** 实现配置的热更新

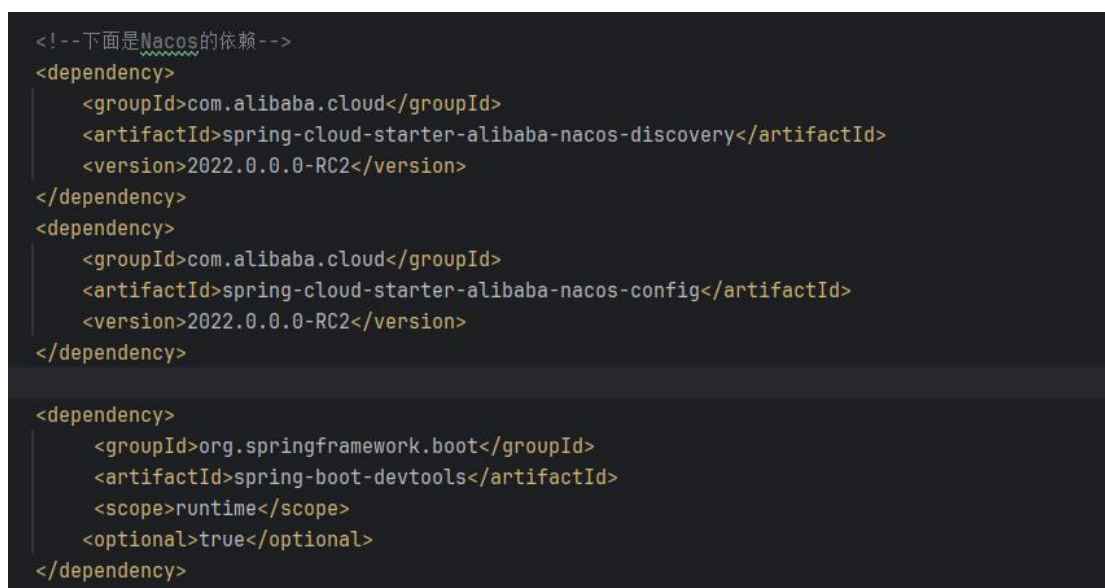
目的: 将满“1000”送“100”和扣减“10%”手续费中的常量配置到配置文件中, 避免硬编码问题, 并将配置文件放在 Nacos 中, 实现在 Nacos 前端控制台直接修改代码配置。

实现过程：

5.3.1： 下载 Nacos 组件并启动运行

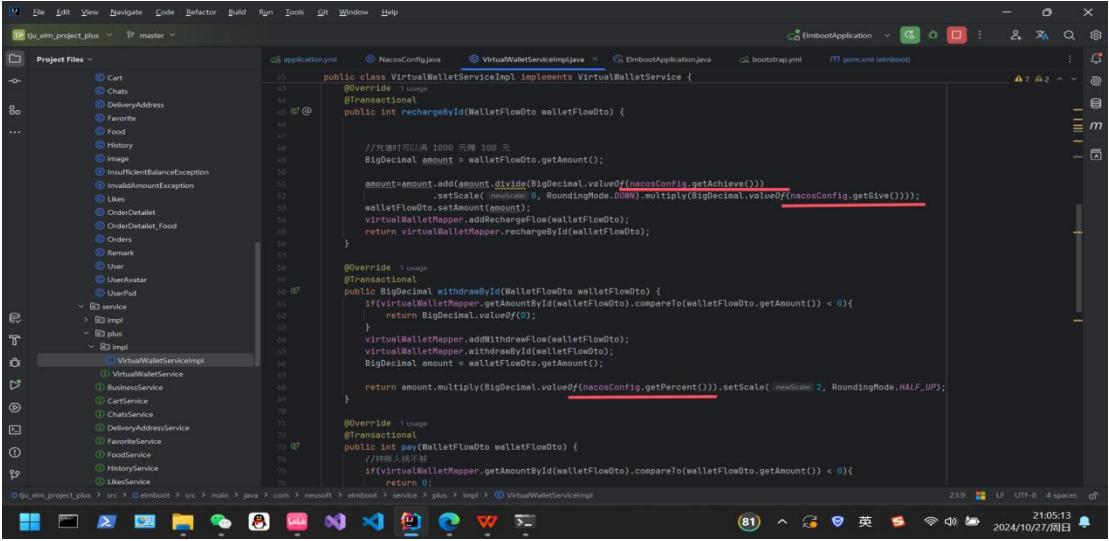


5.3.2 导入微服务 spring-cloud-alibaba 的组件 nacos 的起步依赖：



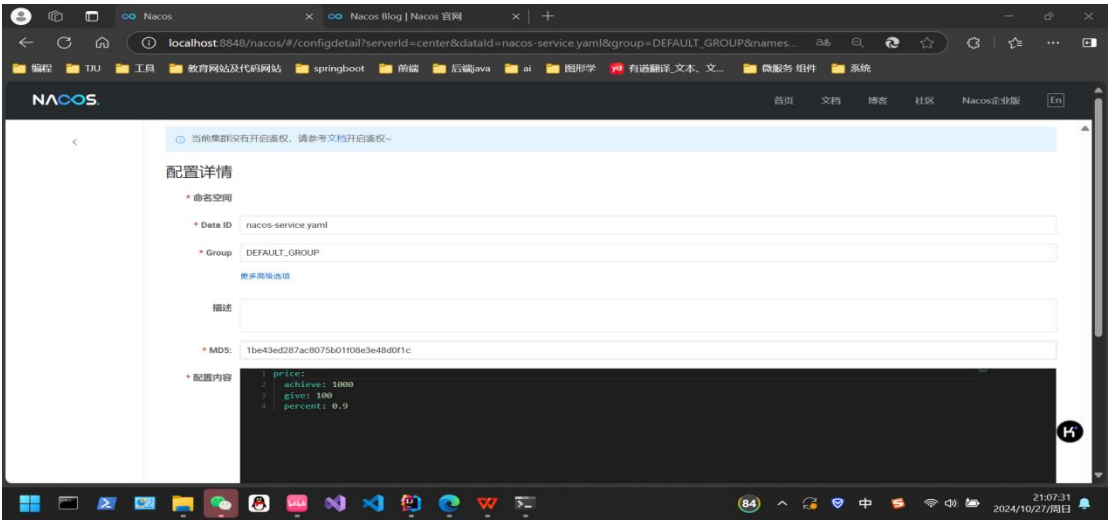
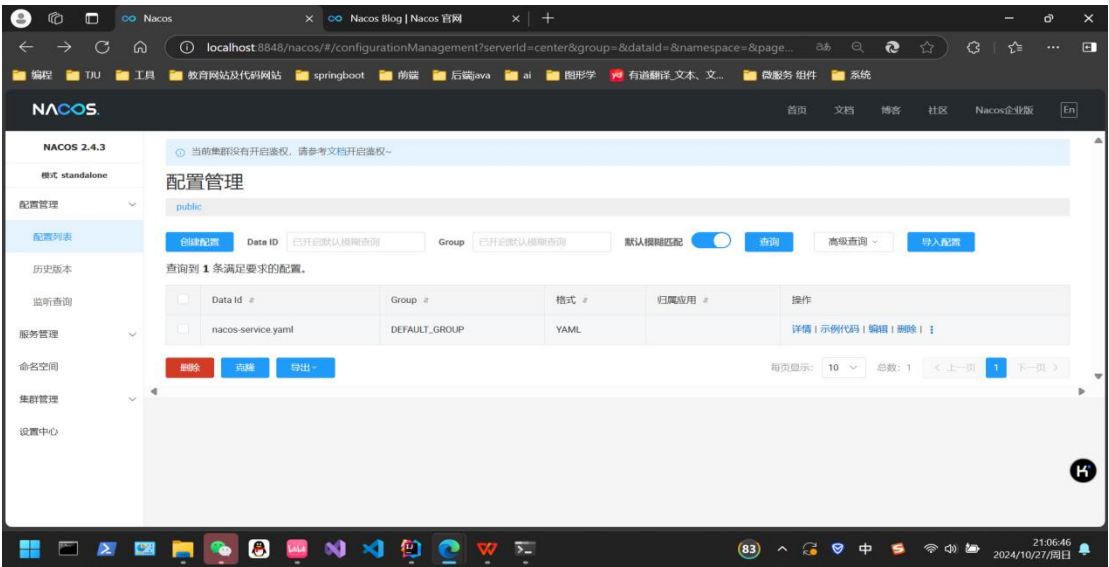
5.3.3： 配置 application.yml 文件

5.3.6: 注入并使用配置类的 Bean，在配置中读取常量，避免硬编码



```
public class VirtualWalletServiceImpl implements VirtualWalletService {  
    @Override  
    @Transactional  
    public int rechargeById(WalletFlowDto walletFlowDto) {  
        //充值时可以满 1000 元赠 100 元  
        BigDecimal amount = walletFlowDto.getAmount();  
        amount = amount.add(amount.divide(BigDecimal.valueOf(nacosConfig.getAchieve()))  
            .setScale(2, RoundingMode.DOWN).multiply(BigDecimal.valueOf(nacosConfig.getGive())));  
        walletFlowDto.setAmount(amount);  
        virtualWalletMapper.addRechargeFlow(walletFlowDto);  
        return virtualWalletMapper.rechargeById(walletFlowDto);  
    }  
    @Override  
    @Transactional  
    public BigDecimal withdrawById(WalletFlowDto walletFlowDto) {  
        if(virtualWalletMapper.getAmountById(walletFlowDto).compareTo(walletFlowDto.getAmount()) < 0){  
            return BigDecimal.valueOf(0);  
        }  
        virtualWalletMapper.addWithdrawFlow(walletFlowDto);  
        virtualWalletMapper.withdrawById(walletFlowDto);  
        BigDecimal amount = walletFlowDto.getAmount();  
        return amount.multiply(BigDecimal.valueOf(nacosConfig.getPercent())).setScale(2, RoundingMode.HALF_UP);  
    }  
    @Override  
    @Transactional  
    public int pay(WalletFlowDto walletFlowDto) {  
        //转账手续费  
        if(virtualWalletMapper.getAmountById(walletFlowDto).compareTo(walletFlowDto.getAmount()) < 0){  
            return 0;  
        }  
    }  
}
```

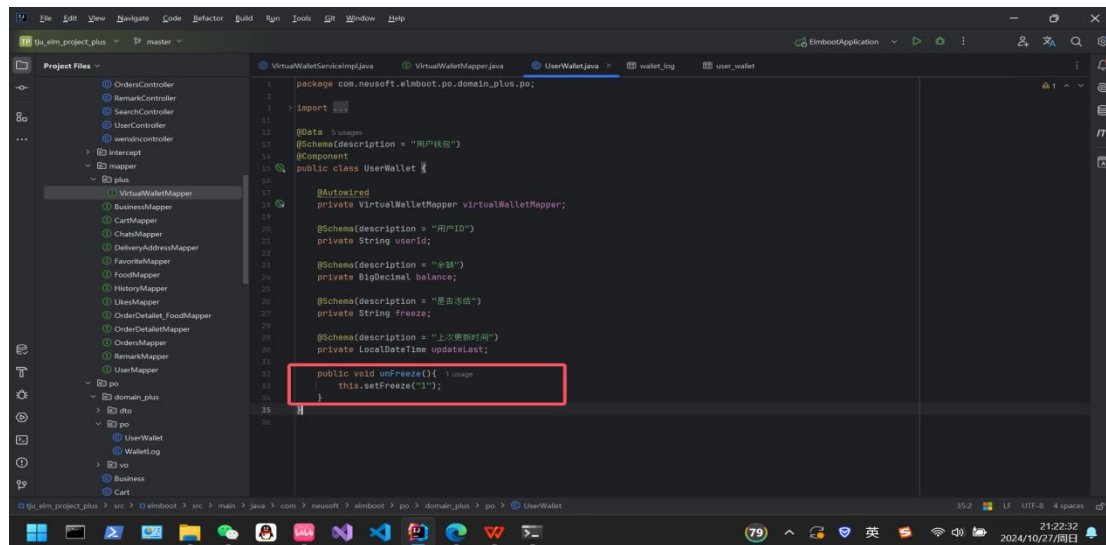
5.3.7: 在 Nacos 中配置常量



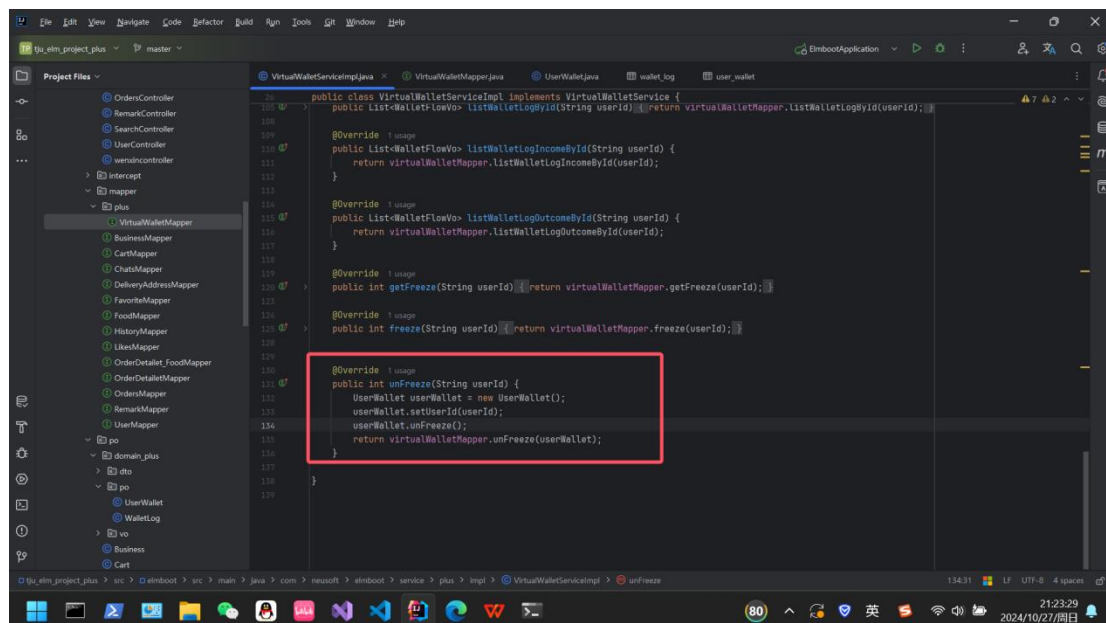
六. 将贫血模型改成充血模型

以解除用户冻结为例：

将用户解冻的方法放在 PO 层的 UserWallet 中定义实现，Service 层想实现用户解冻的功能，可以先调用 UserWallet 的方法 unfreeze 将解冻需要的数据封装在 UserWallet 中，然后拿着 UserWallet 封装好的数据交给 Mapper 层处理即可。即：将业务实现的一部分方法封装在了 Domain 中的 PO 层中，实现了充血模型。

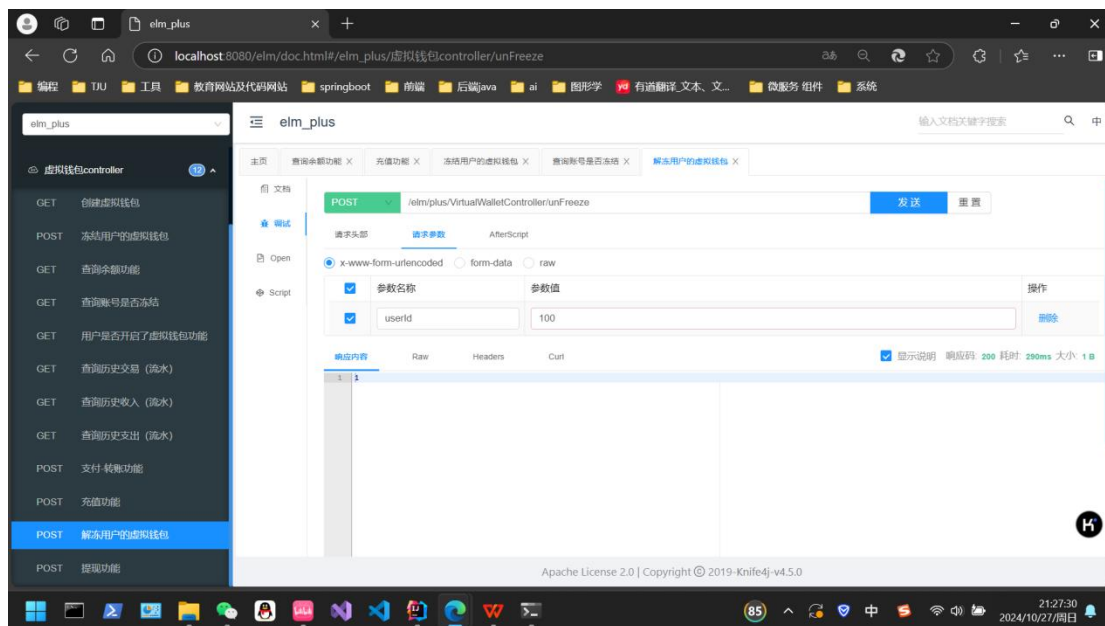


```
1 package com.neusoft.elmboot.po.domain_plus.po;
2
3 import jakarta.persistence.*;
4
5 @Data
6 @Schema(description = "用户钱包")
7 @Component
8 public class UserWallet {
9
10     @Autowired
11     private VirtualWalletMapper virtualWalletMapper;
12
13     @Schema(description = "用户ID")
14     private String userId;
15
16     @Schema(description = "余额")
17     private BigDecimal balance;
18
19     @Schema(description = "是否冻结")
20     private String freeze;
21
22     @Schema(description = "上次更新时间")
23     private LocalDateTime updateTime;
24
25     public void unfreeze() {
26         this.setFreeze("1");
27     }
28 }
```



```
1 public class VirtualWalletServiceImpl implements VirtualWalletService {
2     public List<WalletFlowVo> listWalletLogByUserId(String userId) {
3         return virtualWalletMapper.listWalletLogByUserId(userId);
4     }
5
6     @Override
7     public List<WalletFlowVo> listWalletLogIncomeByUserId(String userId) {
8         return virtualWalletMapper.listWalletLogIncomeByUserId(userId);
9     }
10
11     @Override
12     public List<WalletFlowVo> listWalletLogOutcomeByUserId(String userId) {
13         return virtualWalletMapper.listWalletLogOutcomeByUserId(userId);
14     }
15
16     @Override
17     public int getFreeze(String userId) {
18         return virtualWalletMapper.getFreeze(userId);
19     }
20
21     @Override
22     public int freeze(String userId) {
23         return virtualWalletMapper.freeze(userId);
24     }
25
26     @Override
27     public int unfreeze(String userId) {
28         UserWallet userWallet = new UserWallet();
29         userWallet.setUserId(userId);
30         userWallet.unfreeze();
31         return virtualWalletMapper.unfreeze(userWallet);
32     }
33 }
```

验证结果（成功）：



七. 完成本次阶段性实验