

**Java Web开发技术**

**三 级 项 目 报 告**

**题 目： 模拟ATM网页版**

**学院（系）： 信息科学与工程学院**

**专业班级： 计算机22-2**

**姓 名： 蔡远**

**学 号： 202211040042**

# 摘要

本项目是一个模拟ATM功能的**SpringBoot**与**Vue**集成项目。后端基于Spring Boot框架，负责处理ATM的核心业务逻辑，包括存取款、转账和**查询**余额等核心功能。同时，为了满足系统管理的需求，后端还提供了管理员模块，允许管理员对用户和银行卡信息进行创建、编辑、删除等管理操作。

前端部分则采用了Vue.js框架，通过构建直观易用的用户界面，为用户提供便捷的操作体验。用户可以通过前端界面轻松完成**登录**、查询余额、存取款、转账等操作，同时管理员也可以通过前端界面对用户和银行卡进行管理。

整个项目采用了前后端不分离的开发模式，后端提供接口供前端调用，保证了系统的可维护性和可扩展性。通过本项目，成功地模拟了ATM的基本操作，并为实际开发提供了有价值的参考和借鉴。

**关键字：** **SpringBoot Vue 查询 登录**

目录

[摘要 2](#_Toc169189280)

[1项目概述 5](#_Toc169189281)

[1．1项目简介 5](#_Toc169189282)

[1．2主要功能 5](#_Toc169189283)

[1.2.1用户功能 5](#_Toc169189284)

[1.2.2管理员功能 5](#_Toc169189285)

[1.3技术栈 5](#_Toc169189286)

[1.4项目价值 6](#_Toc169189287)

[1.4.1 个人学习能力的提升 6](#_Toc169189288)

[1.4.2 后端学习能力的提升 6](#_Toc169189289)

[1.4.3前端学习能力的提升 6](#_Toc169189290)

[1.5未来展望 7](#_Toc169189291)

[2.数据库设计 7](#_Toc169189292)

[2.1系统涉及的关系实体和对应关系 7](#_Toc169189293)

[2.2数据库设计 7](#_Toc169189294)

[3.项目后端部分 8](#_Toc169189295)

[3.1项目后端介绍 8](#_Toc169189296)

[3.2后端功能实现 9](#_Toc169189297)

[3.2.1线程获取 9](#_Toc169189298)

[3.2.2异常处理 10](#_Toc169189299)

[3.2.3自动补全对应项 10](#_Toc169189300)

[3.2.4通用返回包装类 11](#_Toc169189301)

[3.2.5登录过滤功能 12](#_Toc169189302)

[3.2.6数据库的增删查改功能 13](#_Toc169189303)

[3.2.7存取款转账和记录功能 15](#_Toc169189304)

[3.2.8登录和登出功能 17](#_Toc169189305)

[3.2.9分页和搜索功能 18](#_Toc169189306)

[4.项目前端部分 19](#_Toc169189307)

[4.1项目前端介绍 19](#_Toc169189308)

[4.2前端功能实现 19](#_Toc169189309)

[4.2.1获取url上的参数 19](#_Toc169189310)

[4.2.2通过正则表达式判断数据是否合理 20](#_Toc169189311)

[4.2.3vue的基本操作 20](#_Toc169189312)

[5.项目效果 21](#_Toc169189313)

[5．1登录页面 21](#_Toc169189314)

[5.2管理员页面 21](#_Toc169189315)

[5.3用户页面 22](#_Toc169189316)

[5.4项目目录结构 22](#_Toc169189317)

[6.总结 23](#_Toc169189318)

# 1项目概述

## 1．1项目简介

本项目是一个基于Spring Boot的ATM模拟系统，旨在提供一个实现基本功能在线银行服务体验。系统后端采用Spring Boot框架，集成了数据库访问该核心功能，前端则采用Vue.js框架，结合HTML、CSS和JavaScript构建用户友好的界面。该系统模拟了真实ATM机的主要功能，包括存钱、取钱、转账等，并扩展了管理员功能，如添加管理员和用户、管理银行卡、查看交易记录等。

## 1．2主要功能

### 1.2.1用户功能

* 存钱：用户可以将资金存入其账户。
* 取钱：用户可以从其账户中提取现金。
* 转账：用户可以将资金从自己的账户转移到其他用户的账户。
* 余额查看：用户可以查看自己的银行卡余额。

### 1.2.2管理员功能

* 用户管理：管理员可以添加新用户、删除用户、修改用户信息。
* 银行卡管理：管理员可以为用户申请新的银行卡、修改银行卡信息（如额度、状态等）。
* 密码管理：管理员可以重置用户的登录密码或银行卡密码。
* 挂失管理：管理员可以处理用户的银行卡挂失请求，包括解除挂失状态。
* 交易记录查看：管理员可以查看所有用户的存取钱、转账等交易记录，以便进行监控和审计。
* 密码修改：用户可以修改自己的登录密码或银行卡密码。
* 挂失银行卡：用户可以对丢失或被盗的银行卡进行挂失操作。

## 1.3技术栈

**后端：**采用Spring Boot框架，使用MyBatisORM框架与数据库进行交互。

**前端：**使用Vue.js框架构建单页面应用（SPA），结合HTML、CSS和JavaScript实现用户交互和界面展示。使用Axios与后端进行HTTP通信。

**数据库：**使用MySQL关系型数据库存储用户信息、银行卡信息、交易记录等数据。

## 1.4项目价值

### 1.4.1 个人学习能力的提升

**深入实践Spring Boot框架：**通过实际开发ATM模拟系统，我更加深入的理解和掌握Spring Boot的核心组件和配置，如Spring MVC、Spring Data JPA等，提升了自己的Spring Boot开发能力。

**增强前后端交互能力：**本项目要求开发者同时掌握前端Vue.js和后端Spring Boot的开发技术，通过前后端交互的实践，提升开发者在前后端协同开发中的能力。

**提高问题解决能力**：在项目开发过程中，我遇到了各种问题和挑战，如如何将前端数据传给后端，如何对前端代码进行debug，如何通过报错信息找到自己的错误，如何再后端处理前端的请求，数据一致性问题等。通过解决这些问题，我发现我解决问题的能力得到了显著提升。不至于遇到问题就什么都不知道。

### 1.4.2 后端学习能力的提升

**熟悉数据库操作：**在ATM模拟系统中，需要熟悉关系型数据库（本项目使用的是MySQL）的操作，包括数据的增删改查，从而提升了自己的数据库操作能力。

**理解微服务架构**：虽然本项目是一个单体应用，但通过学习和了解微服务架构的思想和原理，我可以为未来开发微服务架构的系统打下坚实基础。

**JSON 序列化/反序列化：**使用 Jackson 来实现 Java 对象与 JSON 字符串之间的转换。

**日志管理：**使用 SLF4J来记录和管理系统日志。

### 1.4.3前端学习能力的提升

**学会写简单的Vue**：Vue.js 是一个构建用户界面的渐进式框架，它易于上手且功能强大。Vue.js 提供了响应式数据绑定、组件化开发、虚拟 DOM 等特性，可以更加高效地构建用户界面。在本项目中，使用了Vue 将用于构建前端页面，实现与用户的交互。

**Axios：**Axios 是一个基于 Promise 的 HTTP 客户端，用于浏览器和 node.js。它是 Vue.js 社区中广泛使用的 HTTP 通信库，提供了丰富的 API 和配置选项，可以更加便捷地发送 HTTP 请求和接收响应。在本项目中，使用了Axios 用于实现前端与后端之间的 HTTP 通信。

**HTML/CSS/JavaScript：**HTML 用于构建网页的结构，CSS 用于设置网页的样式，JavaScript 用于实现网页的交互功能。这三者是前端开发的基础技术，也是构建 Vue.js 应用的必备技能。在本项目中，HTML/CSS/JavaScript 将用于构建前端页面的基本结构和样式，以及实现一些简单的交互功能。

## 1.5未来展望

本项目只实现了一些简单的功能，由于个人学习的进度还不够好和开发能力薄弱，现在只实现了一些简单的功能，还有一些功能没有实现，但是现在已经上传到了github等有空的时候再进行完善，对订单的订单详细查看功能还没有实现，还有随机生成银行卡号功能想实现但是已经来不及写了，还有对前端界面的优化，由于时间关系前台和客户端用的一个登陆界面和同样的操作方式，这一点还不够完善，等有空继续改进。

# 2.数据库设计

## 2.1系统涉及的关系实体和对应关系

本项目有四个实体，分别是管理员、用户、银行卡和交易记录。

管理员的基本属性有管理员ID作为唯一标识符，名称，密码和状态。状态为1表示启用，为0则表示禁用，一个管理员可以管理多个用户和多个用户的银行卡，所以管理员和用户与银行卡是一对多的关系。

用户基本属性有用户ID作为唯一标识符，身份证号，姓名，电话，地址，创建时间，更新时间。由于一个用户可以申请多张银行卡，所以用户和银行卡是一对多关系。

银行卡实体的基本属性有银行卡ID作为唯一标识符，用户身份证号，用户姓名，开户时间，开户金额，卡号，货币类型，是否冻结，密码，最后一次操作时间和余额。一个银行卡可以有多条交易数据，银行卡和交易记录是一对多关系。

## 2.2数据库设计

**管理员表设计：**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 备注 |
| Id | Int | 主键自增 | 管理员ID |
| Name | Varchar（25） | 非空，唯一 | 名称 |
| Password | Varchar(64) | 非空 | 密码 |
| Status | Int | 非空 | 状态 |

**用户表设计：**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 备注 |
| Id | Int | 主键自增 | 用户ID |
| Pid | Char（25） | 非空，唯一 | 身份证 |
| Username | Varchar（50） | 非空 | 姓名 |
| Telephone | Varchar（50） | 非空 | 电话 |
| Address | Varchar（256） | 非空 | 住址 |
| Create\_time | Datetime |  | 创建时间 |
| Update\_time | Datetime |  | 更新时间 |

**银行卡表：**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 备注 |
| Id | Int | 主键自增 | 卡ID |
| User\_name | Varchar(20) | 非空 | 持有人姓名 |
| User\_pid | Varchar(20) | 非空 | 持有人身份证 |
| Money\_type | Varchar（20） | 非空 | 货币类型 |
| Number | Varchar（16） | 非空 | 卡号 |
| Password | Varchar（6） | 非空 | 密码 |
| Opne\_money | Decimal（20，2） | 非空 | 开户金额 |
| Balance | Decimal（20，2） | 非空 | 余额 |
| Is\_loss | Int | 非空 | 是否挂失 |
| Create\_time | Datetime |  | 创建时间 |
| Update\_time | Datetime |  | 最后一次更新 |

**交易记录表：**

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 约束 | 备注 |
| Id | Int | 主键自增 | 交易记录ID |
| Number | Varchar（20） | 非空 | 卡号 |
| Create\_ime | Datetime | 非空 | 交易时间 |
| Trade\_money | Decimal（20，2） | 非空 | 交易金额 |
| Trade\_type | Char（20） | 非空 | 交易类型 |
| remark | Text |  | 备注 |

# 3.项目后端部分

## 3.1项目后端介绍

项目后端主要分为一下几个部分，common包，mapper包，entity包，fliter包，config包，和service包，以及启动类。

**Common包：**该报下放的是整个项目通用的的类，其中包括异常处理，线程获取，自动填充对象所需要的时间信息，和请求的通用返回包装类。

**Config包：**用来配置Mybatis plus和springmvc的静态资源映射用来支持访问前端页面。是一些固定的配置。

**Controller包：**是请求处理类，用来处理前端发出的请求。并根据请求的内容、类型和参数等信息执行相应的操作。

**Entity包**：用于存放与数据库表结构相对应的实体类。这些实体类代表了数据库中的表，并且使用注解来映射表结构和字段。

**Mapper包：**继承Mybatis plus的增删改查功能。

**Fliter包：**使用servlet实现对web的拦截，在用户访问请求的时候，自动判断是否登录。

**Service包：**业务逻辑，并在需要时与数据访问层（Mapper层）进行交互。其中还有impl包，用来编写其的实现类。

## 3.2后端功能实现

### 3.2.1线程获取

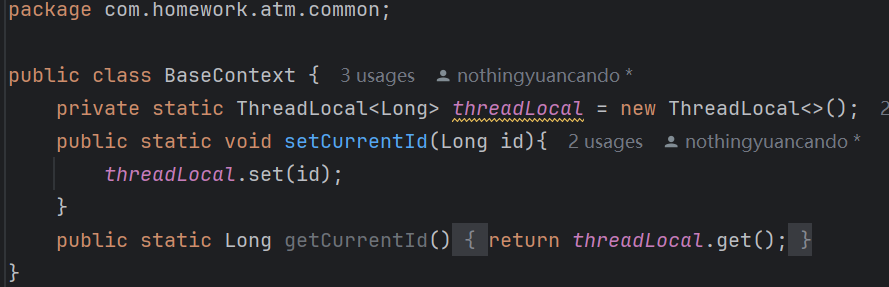
使用了threadLocal 来封装登录信息，当我们在单个线程内多次获取用户信息时，就不用反复从数据库中查询，直接从对应的内存变量中获取即可。

**threadLocal：**用于提供线程局部变量。这些变量与普通的变量不同，因为每一个访问这个变量的线程都有其自己的独立初始化的变量副本。ThreadLocal 实例通常用作私有静态字段，在类中被访问，它们在线程之间隔离存储数据。

**setCurrentId(Long id):** 这个方法将给定的id设置为当前线程的ThreadLocal变量值。这意味着，在调用这个方法的线程中，后续通过getCurrentId()方法获取到的值将是这id。

**getCurrentId():** 这个方法返回当前线程中 ThreadLocal 变量所存储的值。如果之前没有通过 setCurrentId 方法设置过值，那么将返回 null。

**代码：**



### 3.2.2异常处理

创建了一个自定义业务异常类和一个全局异常处理类可以更加规范的管理异常。

GlobalExceptionHandler类是一个全局异常处理器，用于在Spring MVC中集中处理特定的异常。

**@ControllerAdvice：**用于定义全局的异常处理逻辑，以及数据绑定、请求映射等。annotations = {RestController.class, Controller.class}表示这个@ControllerAdvice只针对被@RestController或@Controller注解的类起作用。

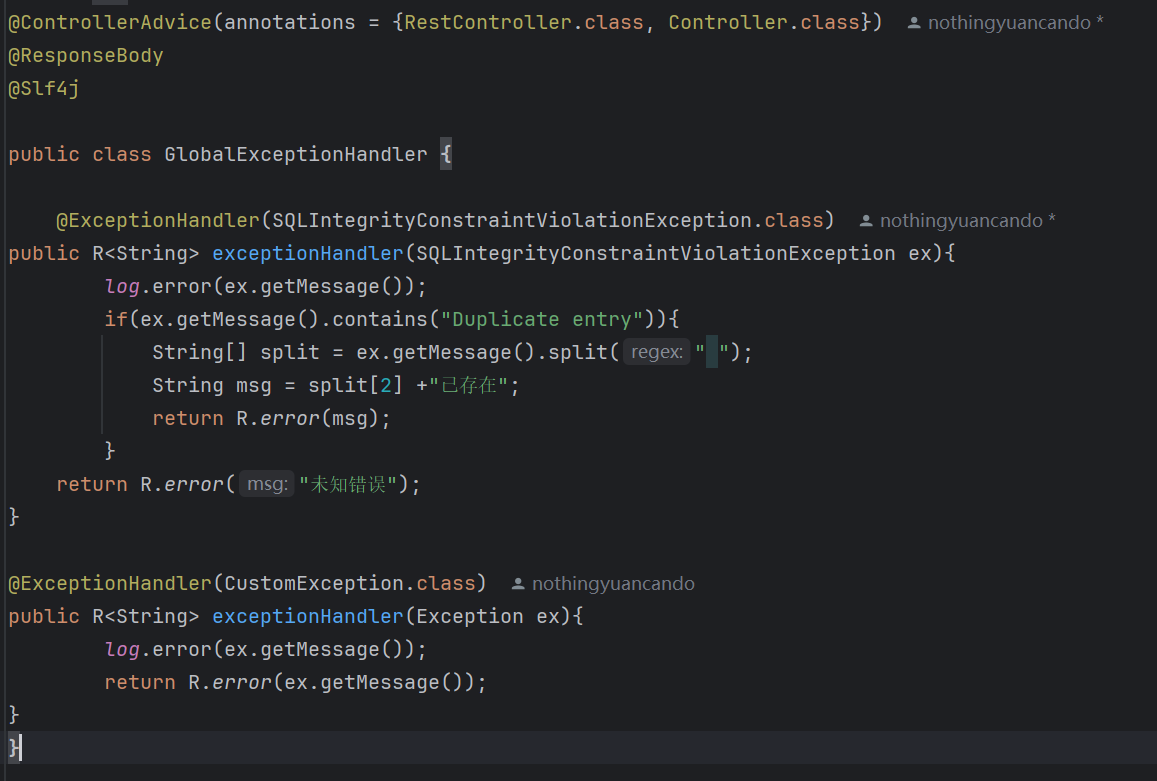
**@ResponseBody：**表示这个类中的方法返回的数据将直接写入HTTP响应体，因为它们直接返回JSON格式的数据。

类中有两个@ExceptionHandler注解的方法，用来处理SQLIntegrityConstraintViolationException异常和CustomException异常。

**SQLIntegrityConstraintViolationException异常**：当数据库操作违反了完整性约束（如主键冲突）时，会抛出该异常。 如果异常消息中包含"Duplicate entry"，则是主键冲突，并提取具体的冲突字段，返回一个包含相应的错误信息的响应。如果不是主键冲突，则返回一个通用的"未知错误"消息。

**CustomException异常：**捕获到所有类型的异常，并记录其消息。 返回一个包含异常的消息的响应。

代码：



### 3.2.3自动补全对应项

为了实现能够自动填充字段如创建时间和更新时间。创建了 MetaObjectHandler类，这个接口与 MyBatis-Plus 框架一起使用，用于在插入或更新数据库记录时自动填充这些字段。在这个类中，创建了两个方法：

**insertFill(MetaObject metaObject):** 当插入记录时，这个方法会被调用。它直接将当前时间设置为createTime字段的值。

**updateFill(MetaObject metaObject):** 当更新记录时，这个方法会被调用。它直接将当前时间设置为updateTime字段的值。

使用这个方法时我们需要使用@TableField(fill = FieldFill.INSERT) 和 @TableField(fill = FieldFill.UPDATE) 注解来指定应该被自动填充的字段。这些注解需要写在实体类中的相应字段上。

**代码：**

****

### 3.2.4通用返回包装类

我们每个请求不仅要返回本来的数据之外，还要额外返回状态码和一些错误信息，便于debug，这个通用返回类（R）的主要功能是封装HTTP响应的数据，包括响应状态码（code）、响应消息（msg）、实际数据（data）以及一些动态数据（map）。

**具体功能描述如下：**

**状态码 (**code**)**：表示操作执行结果，1表示操作成功，0表示操作失败。

**响应消息 (**msg**)**：当操作失败时，msg字段提供详细的错误信息。

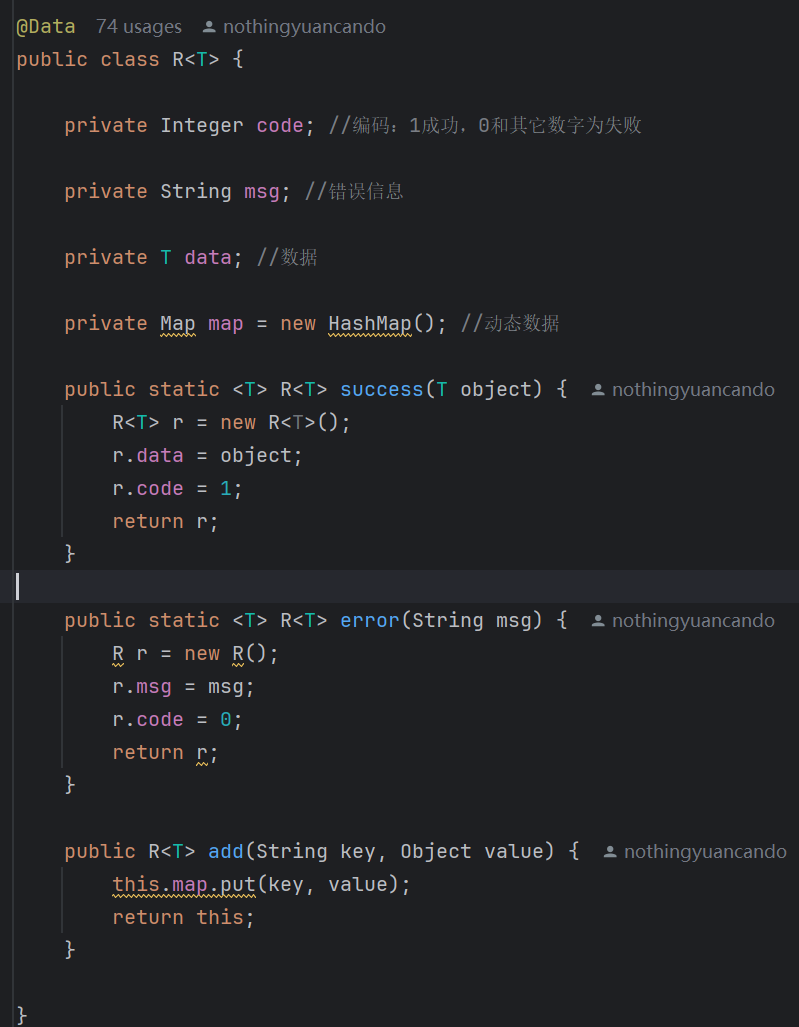
**实际数据 (**data**)**：包含了调用者所请求的具体数据。。

**动态数据 (**map**)**：允许在响应中包含额外的、不固定的数据。

**静态方法 (**success**和**error**)**：这两个静态方法提供了创建 R<T> 实例的便捷方式。success 方法用于创建表示操作成功的响应，它接受一个参数作为实际数据；error 方法用于创建表示操作失败的响应，它接受一个字符串参数作为错误信息。

**add方法**：这个方法允许在已经创建的 R<T> 实例上添加额外的动态数据。它接受一个键和一个值作为参数，并将它们添加到 map 字段中。

**代码：**

****

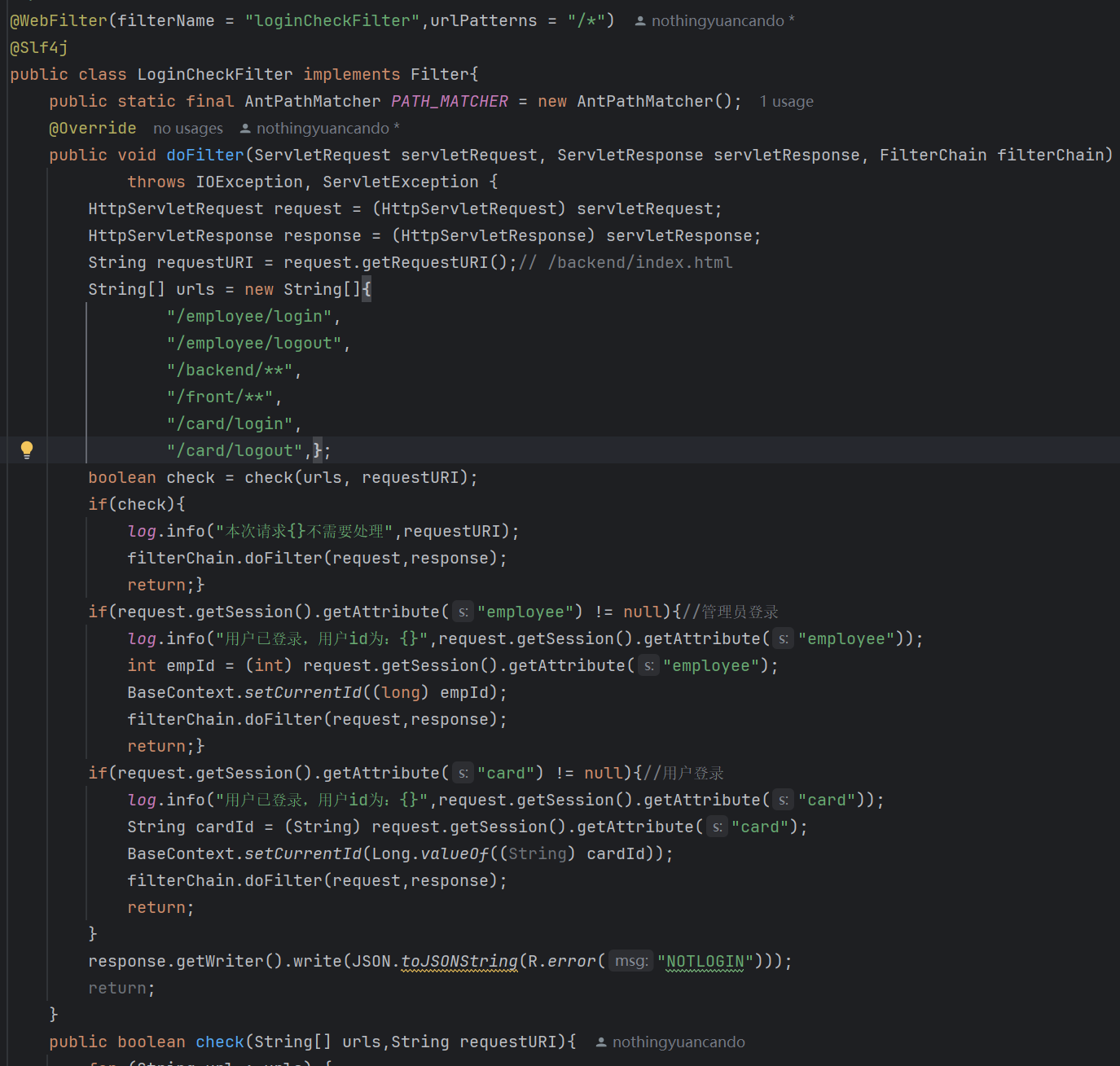
### 3.2.5登录过滤功能

LoginCheckFilter类是一个自定义的过滤器，用于检查用户的登录状态。当用户页面时，这个过滤器会拦截请求，并根据预设的规则来决定是否允许请求继续执行或重定向到登录页面。

**@WebFilter(filterName = "loginCheckFilter", urlPatterns = "/\*"):** 这是一个Servlet 3.0提供的注解，用于声明这个类是一个过滤器，并指定过滤器的名称和需要拦截的URL模式。urlPatterns = "/\*"表示该过滤器会拦截所有的请求。

**public static final AntPathMatcher PATH\_MATCHER = new AntPathMatcher();**: 这是一个路径匹配器，支持Ant风格的路径模式（如/backend/\*\*）。

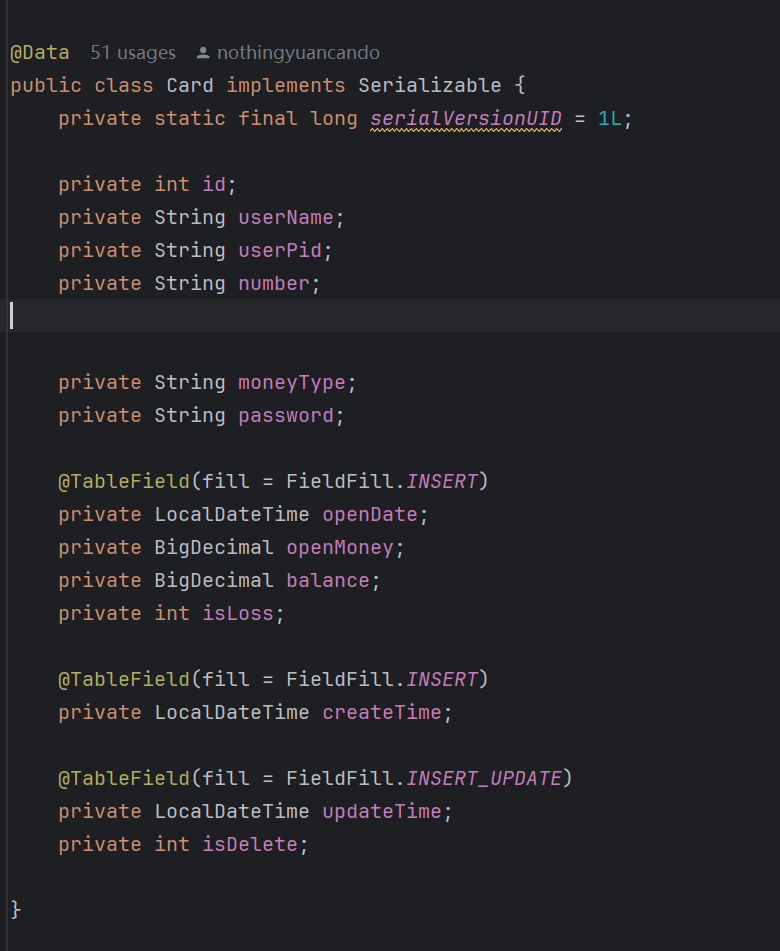
**doFilter方法：**这是Filter接口的核心方法，当请求被拦截时，该方法会被调用。方法内部首先获取请求的URI，然后检查这个URI是否匹配预设的免检路径。如果URI匹配免检路径，则直接放行请求。如果URI不匹配免检路径，则检查用户的登录状态。这里检查的是会话（session）中是否存储了用户信息（如管理员ID或用户ID）。如果用户已登录（会话中存在用户信息），则将用户ID设置到BaseContext（可能是一个自定义的上下文类，用于在应用中传递用户ID），然后放行请求。如果用户未登录，则通常应该重定向到登录页面。

**代码：**

### 3.2.6数据库的增删查改功能

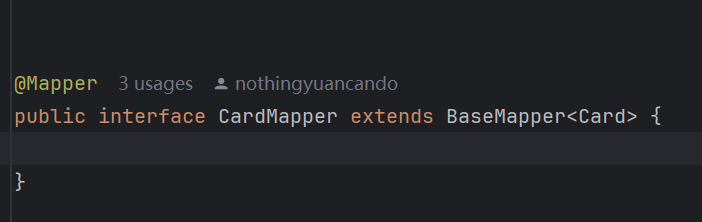
为了能够操作数据库，创建了entity层，其中的类用于对数据库的映射操作，和对前端传递数据的封装操作，并对其使用了@Data注解，这样可以对所有字段的getter方法，对于类的每个字段，Lombok都会生成一个标准的getter方法。从而获取其中对应变量的值。@TableiField用于对对应字段实现自动填充时间。

**例如entity中的Card类：**



在MyBatis-Plus中，@Mapper注解用于标记一个接口为MyBatis的Mapper接口，即数据访问层接口。当使用 MyBatisPlus 时，就不需要手动编写 SQL 语句，因为 MyBatis-Plus提供了操作的默认实现。CardMapper接口继承BaseMapper<Card>，其中 BaseMapper是 MyBatis-Plus 提供的一个泛型接口，它包含了大部分常用的数据库操作。泛型 <Card> 表示这个 Mapper 是用于操作Card实体类的。

对应代码：



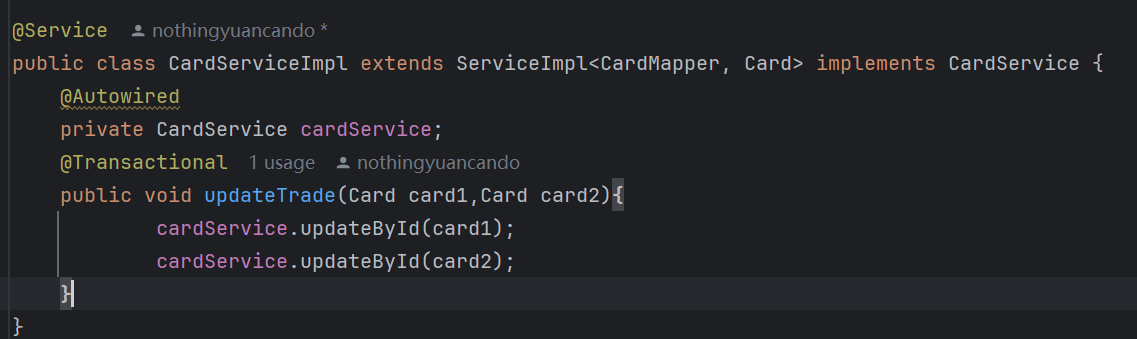
在CardService接口中，扩展了IService<Card>接口，这是MyBatis-Plus提供的一个通用服务接口，它包含了一些基本的CRUD方法。定义了方法updateTrade(Card card1, Card card2)用来进行转账操作。

**IService<Card> 继承：**这意味着您的CardService接口继承了MyBatis-Plus提供的所有基本服务方法，如selectById、save、updateById、removeById等。

**updateTrade方法**：updateTrade(Card card1, Card card2)将在CardServiceImpl类中完成，该类实现CardService接口。并实现转账方法。

在service层下还有impl层，用来编写service层的实现类，为了完成转账操作，使用了**@Transaction**注解，这样就保证了不会出现一方转账一方未收到的错误。@Service是Spring 框架中的一个注解，用于标注业务层（Service 层）的组件。当使用@Service注解标注一个类时，Spring 容器会将其识别为一个 bean，并自动将其纳入 Spring 的应用上下文中进行管理。这样，你就可以在需要的地方通过自动装配（使用@Autowired注解）来引用这个 bean，并执行相应的业务逻辑。

**对应代码：**

****

从而可以直接通过对应的service直接进行数据库的增删查改操作，不用再进行sql语句的编写。从而实现对数据库的操作，也实现了对对应实体的操作功能。对于剩下的三个数据库，用户，管理员，和交易记录表的操作实现代码和上述代码大同小异，这里不再过多叙述。

### 3.2.7存取款转账和记录功能

由于对speringboot的编写不是很熟练，我把很多方法都写到了控制层中，存取款和转账记录功能都在控制层中实现，对控制层的编写使用了很多注解，为了方便理解，这里将主要使用的一些注解进行解释：

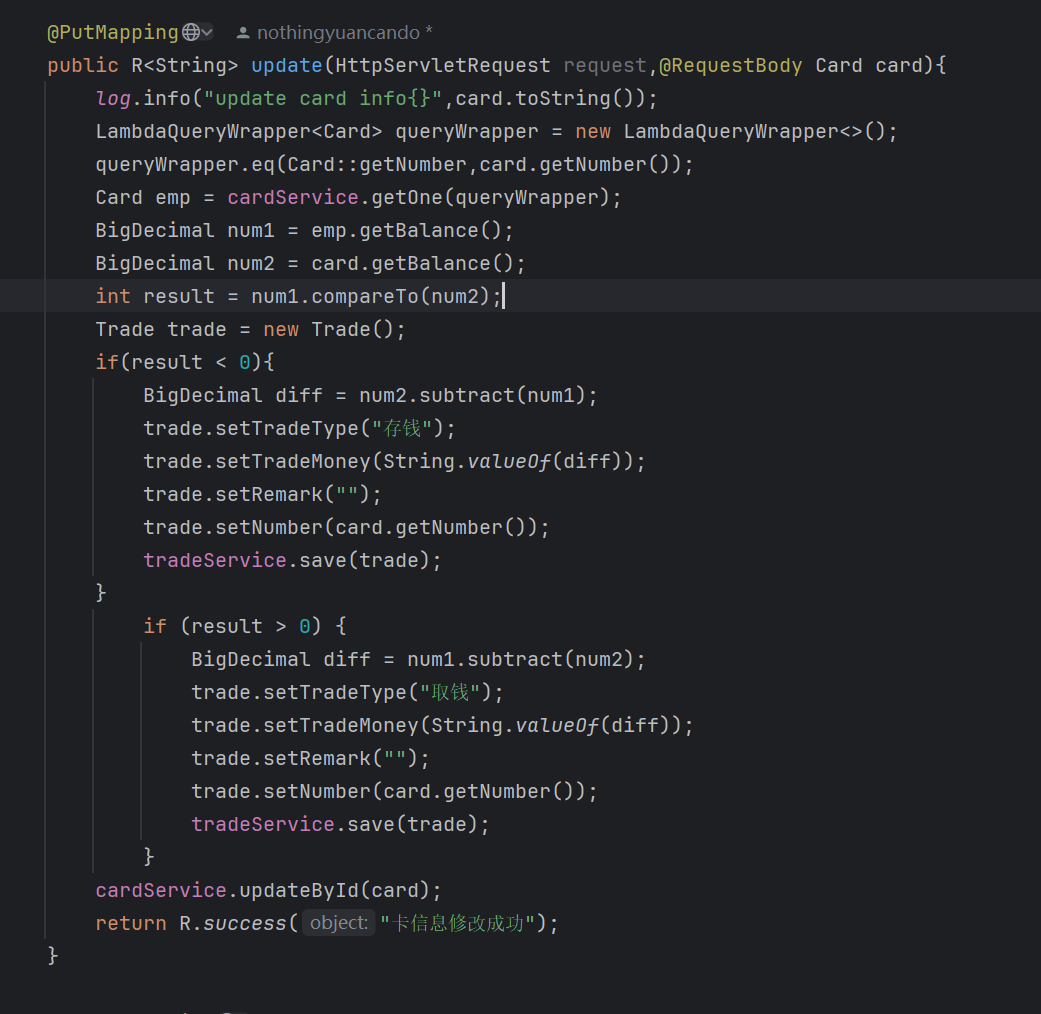
**@RestController:**Spring Framework 中的一个注解，用于标注一个类为控制器（Controller），同时该控制器返回的响应体中的数据会自动转换为 JSON 或 XML 等格式,@RestController注解实际上是@Controller和@ResponseBody的结合体，但使用起来更加简洁。当你一个类上使用@RestController注解，该类中的所有方法都会默认添加@ResponseBody注解，这意味着方法返回的对象将自动被转换为 JSON 或 XML 响应，而无需在每个方法上都添加 @ResponseBody。

**@RequestMapping**:Spring MVC 中的一个核心注解，用于将特定的请求映射到控制器类中的方法上。这个注解可以定义在类级别上，也可以定义在方法级别上。当定义在类级别上时，它指定了一个基本路径，该路径下的所有请求都会映射到该类中的方法上。

**@Post/Get/Put/DeleteMapping** 是 Spring Framework 4.3 引入的一个组合注解，它是@RequestMapping(method = RequestMethod.请求方式) 的简写。这个注解用于处理 HTTP请求，并将它们映射到控制器类中的方法上。使用@PostMapping可以使代码更加简洁和易于理解，因为它明确地指出了该方法只处理的**请求**和**路径**。

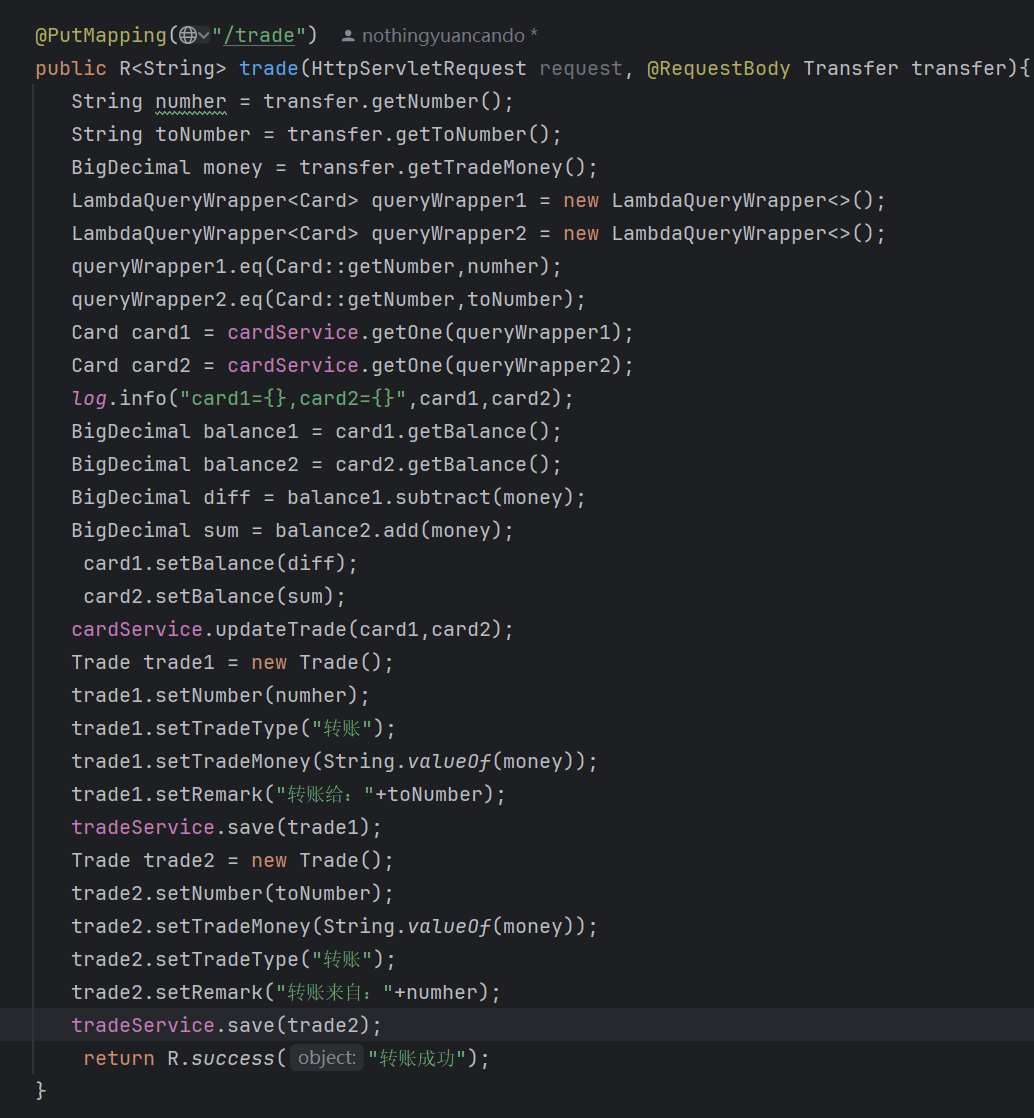
**LambdaQueryWrapper<T>：**是 MyBatis-Plus 中提供的一个查询包装器，用于构建 SQL 查询条件，简化了构建条件查询的过程，允许直接使用实体类中的字段名来指定查询条件，T是一个泛型参数，代表要查询的实体类的类型。

**存取款和更新银行卡信息代码并记录下存取款信息：**



**转账功能实现：**

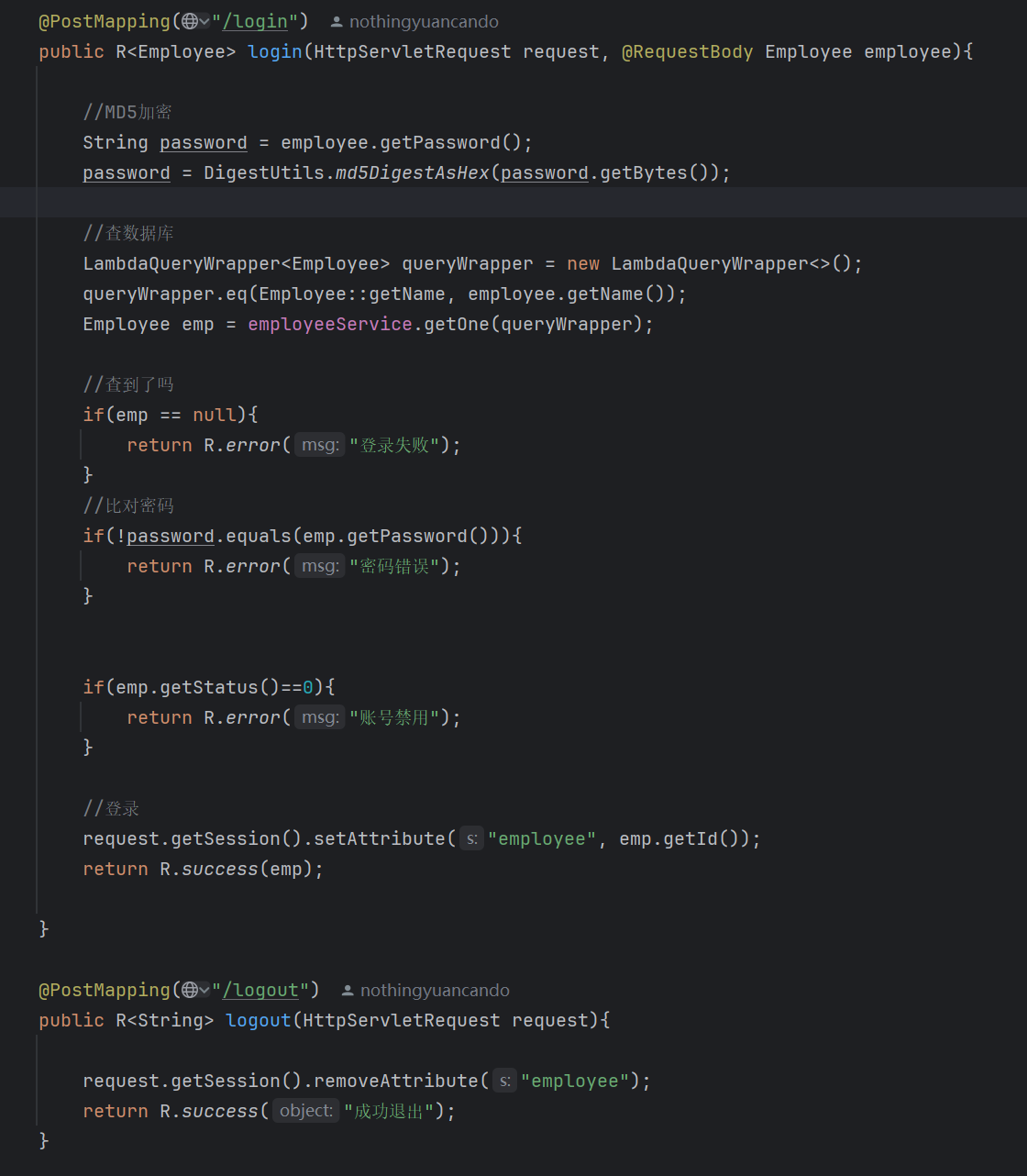
根据接收到的封装好的数据来进行查询对应卡和更新对应卡的余额并将信息记录在交易记录表中。



### 3.2.8登录和登出功能

当接受到前端传过来的数据后使用MD5算法对传入的密码进行加密，使用LambdaQueryWrapper 创建一个查询条件，用于从数据库中查找管理员信息，根据用户名查询使用服务层方法查询员工信息 如果找到了员工信息 ，并且加密后的密码与数据库中的密码匹配且检查员工状态未被禁用则将员工的信息存储在HTTP会话中，用于后续的身份验证并返回成功响应，并带上员工信息，如果上面有一条不符合，则返回登陆失败或密码错误或者被禁用。登出就比较简单 直接从HTTP会话中移除员工的信息，表示用户已注销 返回成功响应，提示用户已成功退出。对于银行卡操作时的登录操作也一样，这里不再叙述。

**登录登出代码：**



### 3.2.9分页和搜索功能

对于分页操作，使用了Mybatis-plus的Page分页插件，@GetMapping("/page")注解用于处理分页查询请求，以下是该方法的步骤：

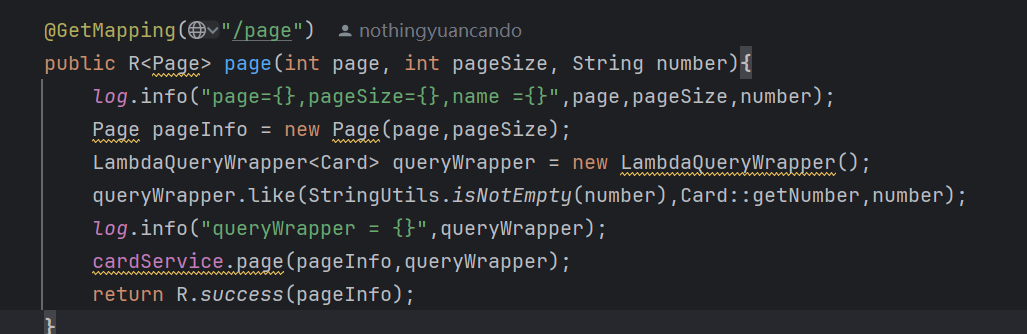
**创建分页对象**：使用传入的page和pageSize参数创建一个Page对象，这个对象将用于记录分页信息（当前页码、每页记录数）。

**构建查询条件实现搜索功能**：使用LambdaQueryWrapper来构建查询条件。这里只使用了一个模糊查询条件，即根据number字段查询卡号。如果number不为空，则添加这个查询条件。

**执行分页查询**：调用cardService.page(pageInfo,queryWrapper)方法执行分页查询。这个方法是CardService接口中的一个方法，根据提供的分页信息和查询条件从数据库中查询数据，并将查询结果设置到pageInfo对象中。

**返回结果**：使用R.success(pageInfo)返回一个该分页信息的成功响应。这里的R为通用返回包装类，用于封装HTTP响应的状态码、消息和数据。

**具体代码如下：**

****

# 4.项目前端部分

## 4.1项目前端介绍

由于我主要学习的是后端SSM框架，但该项目的实现也需要前端内容，所以又只能加急学了一点vue，但是前端还是写的不忍直视，我还需要继续学习，来完善这个项目，在完成这个项目的过程中，后端代码写起来倒不是很麻烦，主要是前端写起来很不容易，大部分时间都在处理简单的一两个小问题，但是想找到就很麻烦，对于前端内容的薄弱，可能是因为学习的侧重点不同，但好在还是勉勉强强实现了基础功能和一些附加功能，在写前端代码的时也参考借鉴了不少其他项目的方式，所以主页面会有一些和其他项目相似的地方，但是对应功能的实现都是我自己实现的，包括对数据的封装，利用axios来发送 HTTP 请求，对前端输入的检查判断等。

## 4.2前端功能实现

### 4.2.1获取url上的参数

为了获取url上的参数，我们可以通过以下几步：

**获取 URL：**var url = location.href 获取当前页面的完整 URL。

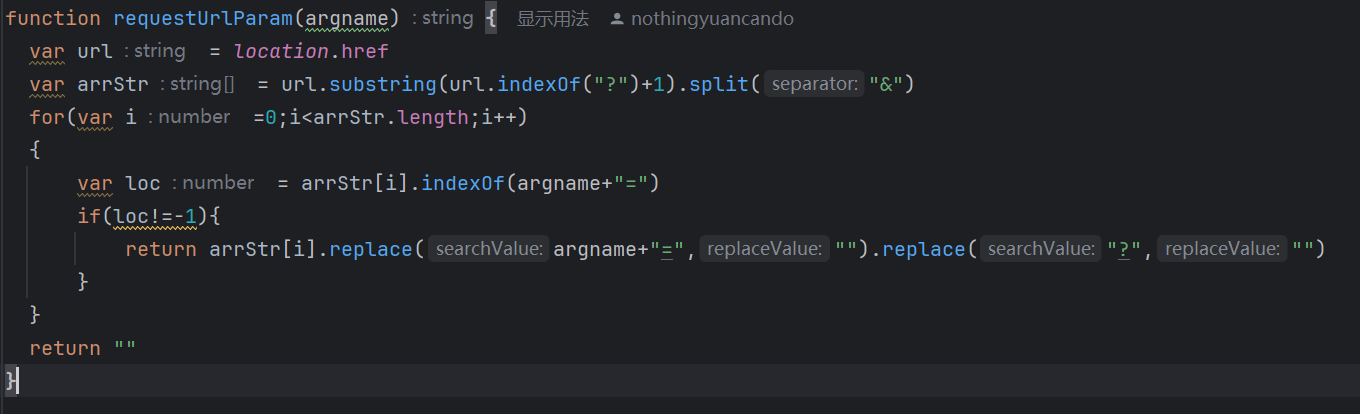
**分割查询字符串：**var arrStr = url.substring(url.indexOf("?")+1).split("&") 从 URL 中提取查询字符串（即 ? 后面的部分），并使用 & 符号将其分割成一个数组。

**遍历参数数组：**使用 for 循环遍历参数数组 arrStr。

**查找参数名：**var loc = arrStr[i].indexOf(argname+"=") 查找当前参数是否包含指定的参数名。

**返回参数值：**如果找到了参数名，使用 replace 方法去除参数名和等号，并尝试去除一个可能不存在的?，如果在所有参数中都没有找到指定的参数名，函数返回空字符串 ""。

**代码：**



### 4.2.2通过正则表达式判断数据是否合理

当在对应的输入框输入数字时，对应函数会判断输入的数字是否合理并给出提示信息，对于如何判断对应输入是否合理，使用了正则表达式来进行判断，对于其他参数如手机号也使用了对应的正则表达式，这里不再叙述，对应代码如下：



### 4.2.3vue的基本操作

**模板语法：**Vue.js使用双大括号{{}}来显示数据，使用v-指令（如v-bind、v-on、v-if等）来实现对DOM的操作。

**指令**：Vue.js提供了一系列内置的指令，如v-for用于遍历数组或对象，v-model用于实现表单元素的双向数据绑定。

* + v-for：可以基于一个数组或对象来渲染一个列表或表格。
  + v-model：使得表单输入元素（如<input>、<textarea>等）和Vue实例的数据保持双向同步。

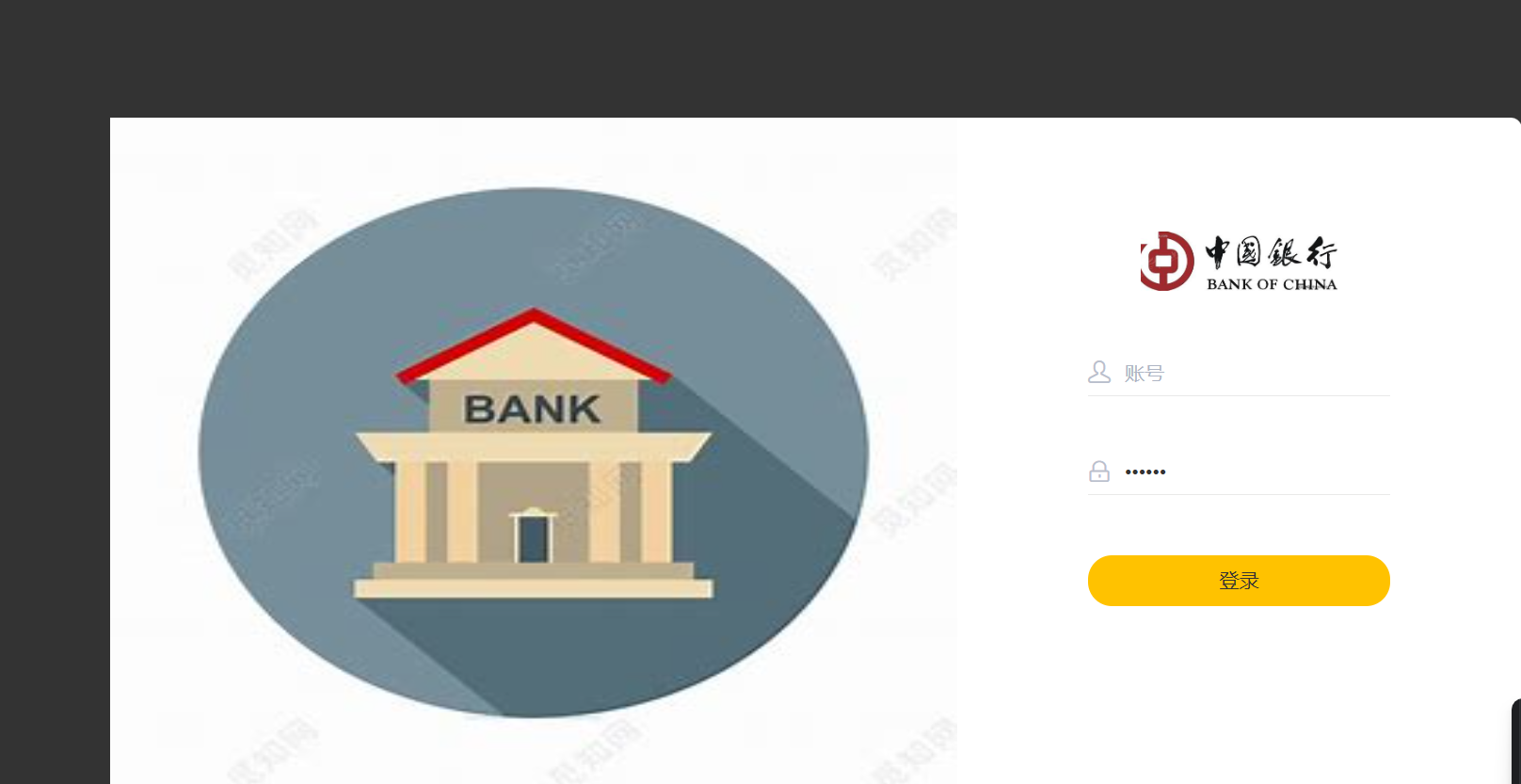
**组件：**Vue.js中的组件是可复用的Vue实例，它们具有自己的模板、数据和方法。全局组件：可以在任何Vue实例的模板中使用。局部组件：只能在定义它的Vue实例的模板中使用。组件之间可以通过props传递数据，通过自定义事件进行通信。

**生命周期钩子：**Vue.js实例在其生命周期中会经历多个阶段，每个阶段都对应一个生命周期钩子函数。这些钩子函数允许我们在特定的阶段执行特定的操作，如数据初始化、DOM挂载、数据更新等。

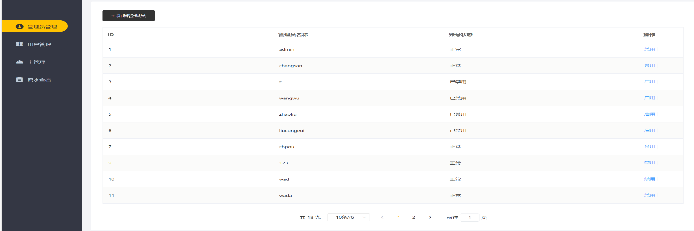
通过学习Vue的基础操作，我部分体会到了其简洁、高效和灵活的特点。Vue的响应式数据绑定和组件化开发模式使得前端开发变得更加轻松和高效。同时，Vue丰富的API和插件生态也提供了更多的可能性。未来继续深入学习Vue并将其应用到实际项目中。

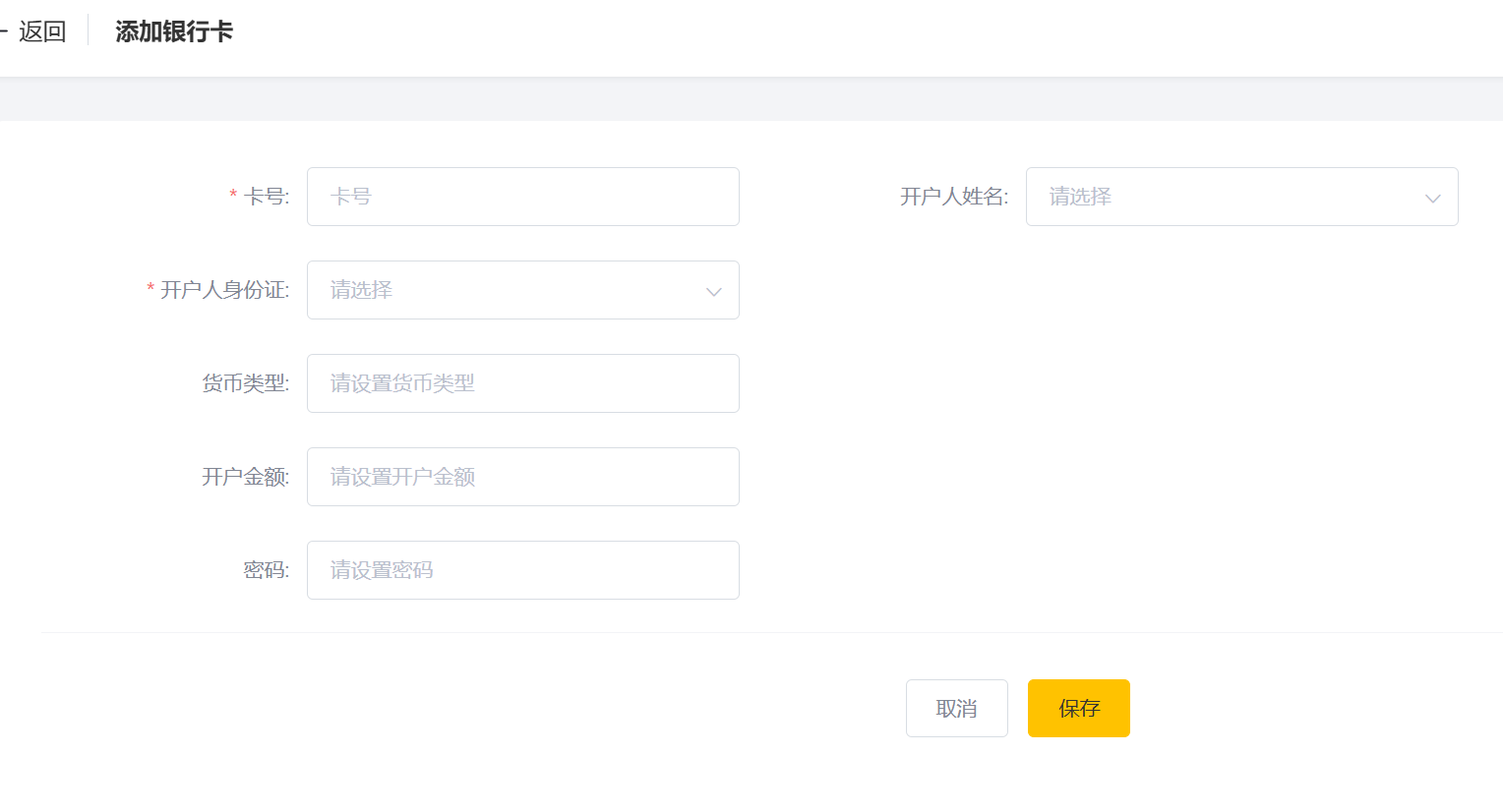
# 5.项目效果

## 5．1登录页面



## 5.2管理员页面

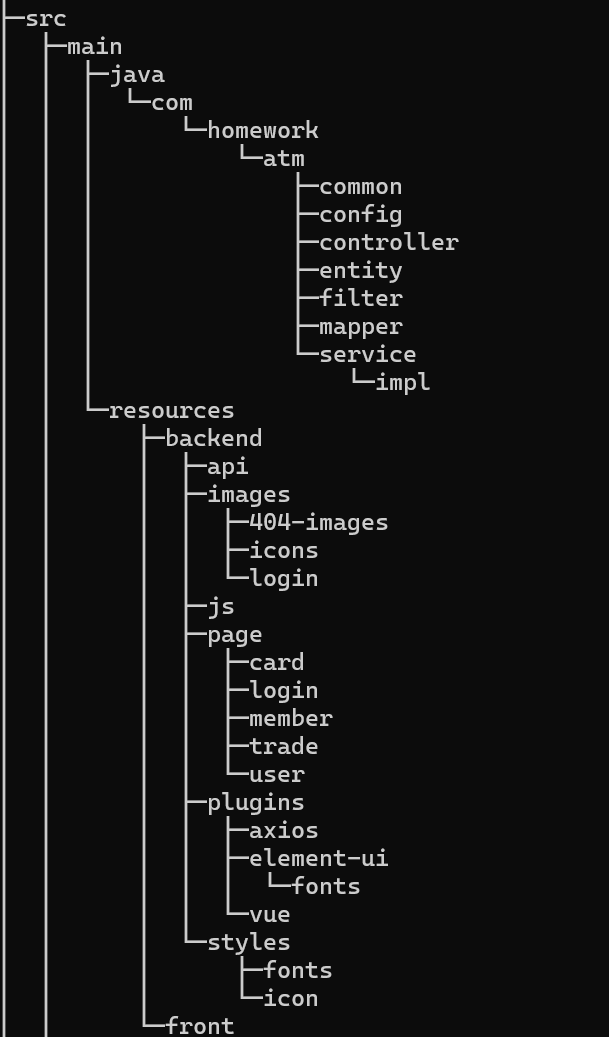




## 5.3用户页面



## 5.4项目目录结构



# 6.总结

本项目基本实现了要求的基本功能，对于用户的转账和存取款，查询功能已经实现，管理员可以增加新的管理员，对用户进行编辑和申请银行卡等操作，也可以查看交易流水，但其实还可以多加一些功能，比如随机生成银行卡，用户登录界面再改一下，改的和ATM一样，再多一些操作时的提示语句和页面跳转停留几秒的功能，或者增加语音播报的功能，这些都可以再以后加上。还有对于其他方面的考虑，比如数据的安全性，转账和存取款过程中发生错误的话，异常处理还不够好，前后端的数交互不是很安全，前端页面还要不够流畅完整，后面会继续添加功能，这也算是对自己网课学习的成果的一个检验，也是对自己能力的一个检验，总之，项目的实现不仅提高了我的实践能力，也为未来开发积累了宝贵经验。