武汉纺织大学

Web应用开发课程设计

**采用SSM框架实现简单的银行ATM系统**

**学 院： 数学与计算机学院**

**班 级： 物联网11804**

**姓 名： 廖文俊**

**学 号： 1804280428**

**指导老师： 聂刚**

**成 绩：**

**完成日期： 2020年12月19日**

目 录

[1 需求分析](#_Toc22456)

[1.1登录](#_Toc6247)

[1.2账户锁定](#_Toc15319)

[1.3存款](#_Toc14458)

[1.4 取款](#_Toc1897)

[1.5 转账](#_Toc7940)

[1.6 查询](#_Toc4651)

[1.7 改密](#_Toc30015)

[2 系统设计](#_Toc21868)

[2.1用户用例图](#_Toc16712)

[2.2数据库](#_Toc13261)

[2.3 UML类图](#_Toc12060)

[2.4 UML时序图](#_Toc9481)

[2.4.1 系统登录注册](#_Toc10858)

[2.5 UML活动图](#_Toc22054)

[2.5.1 存款](#_Toc12767)

[2.5.2 取款](#_Toc20020)

[3 系统实现](#_Toc20261)

[3.1 项目结构](#_Toc17930)

[3.2 配置文件](#_Toc14003)

[3.2.1 jdbc.properties文件](#_Toc32289)

[3.2.2 log4j.properties文件](#_Toc20299)

[3.2.3 mybatis-config.xml文件](#_Toc19038)

[3.3 示例代码](#_Toc7838)

[4 课程总结](#_Toc18618)

# 1 需求分析

设计一个银行ATM系统，实现“登录”、“账户锁定”、“存款”、“取款”、“转账”、“查询和改密”等功能，具体要求如下：

## 1.1登录

## 1.2账户锁定

### 1.3存款

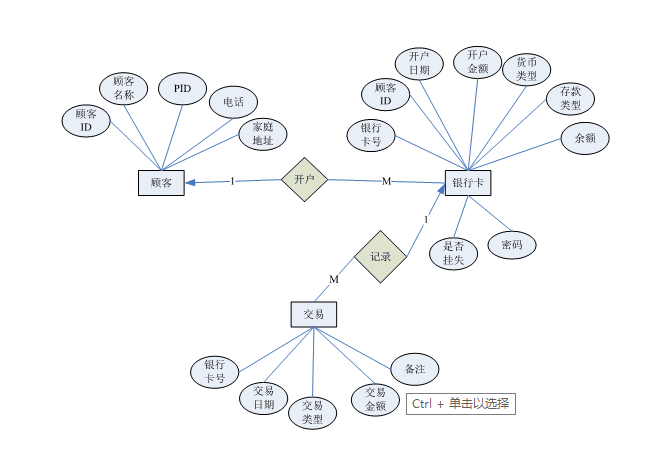
### 1.4取款

### 1.5转账

### 1.6查询

### 1.7改密

# 2 系统设计



## 2.1用户用例图

系统用户用例如下图2-1所示：

存款

取款

转账

查询

改密

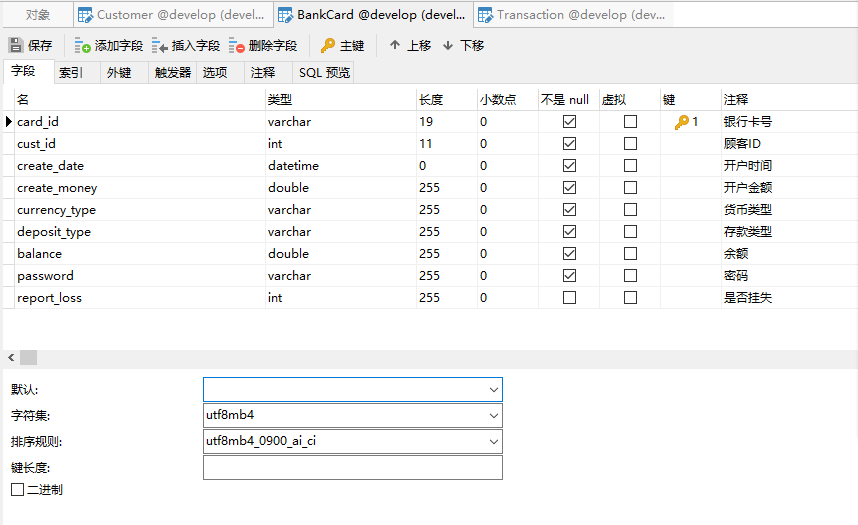
登录注册

图2-1 用户用例图

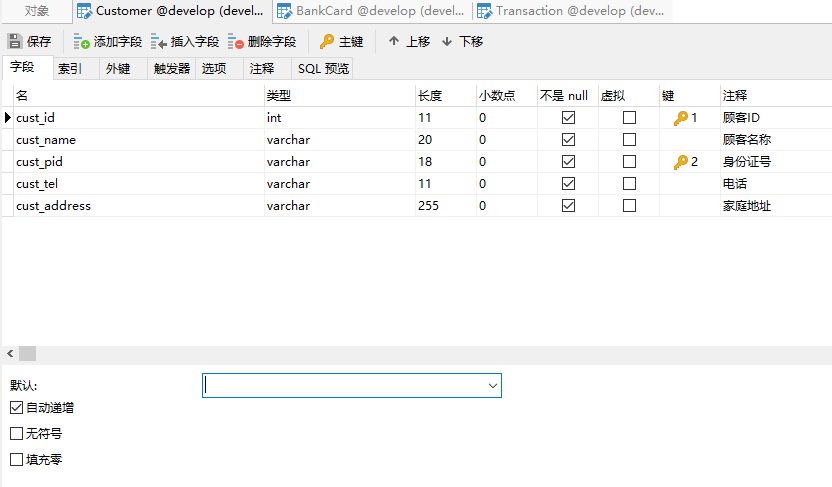
## 2.2 数据库

数据库设计3张表

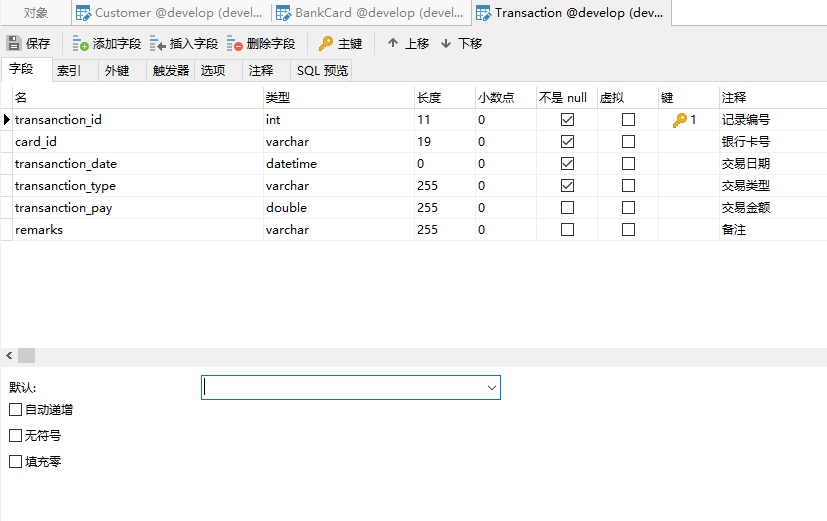
BankCard.sql



Customer.sql



Transaction.sql



## 2.3 UML类图（Class Diagram）

对于用户功能模块，共设计如下6个类。

* 数据库连接类DatabaseConnection：连接数据库并获取连接对象，关闭连接对象。
* VO实体类User：与数据库结构进行映射的类。主要由属性，setter, getter方法组成，VO类中的属性与表中的字段相对应，每一个VO类的对象都表示表中的每一条记录
* DAO接口IUserDAO：主要定义操作的接口，定义一系列数据库的原子性操作，例如增删改查（通常称为CRUD）等。
* DAO实现类UserDAOImpl：DAO接口的真实实现类，主要完成具体数据库操作，但不负责数据库的打开和关闭。
* DAO工厂类DAOFactory：通过工厂类取得一个DAO的实例化对象
* 业务逻辑实现类UserService：对于数据层的原子操作进行整合。还要负责数据库的打开与关闭（不管是否出异常，数据库都要关闭）
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*：

各类的结构及类之间的关系如图2-5所示：

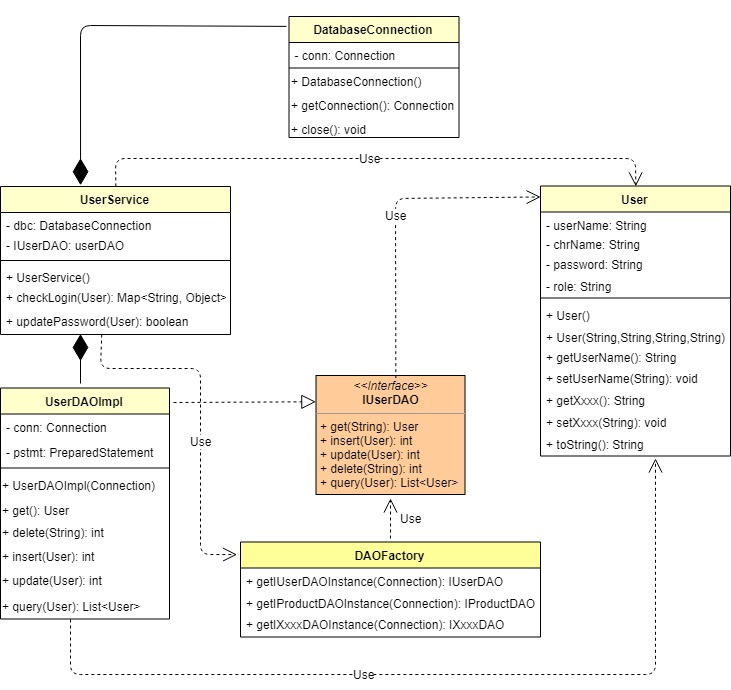
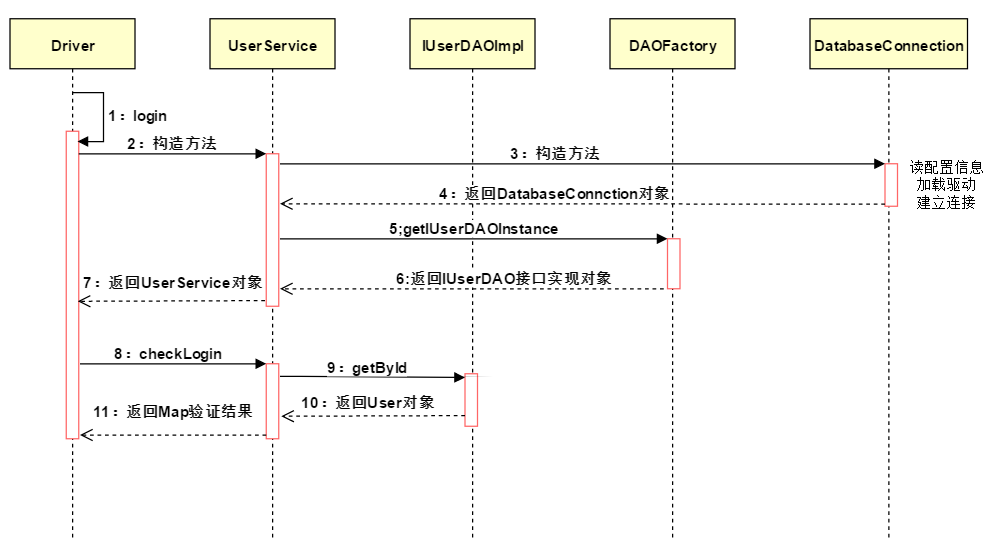


图2-5 用户功能类图

## 2.4 UML时序图（Sequence Diagram）

### 2.4.1 系统登录



## 2.5 UML活动图（Activity Diagram）

### 2.5.1 登录

登录操作需要验证用户名，密码，只有当用户名和密码同时存在即用户存在的情况下才能成功登录系统，否则会提示错误信息，具体流程如图2-6所示。

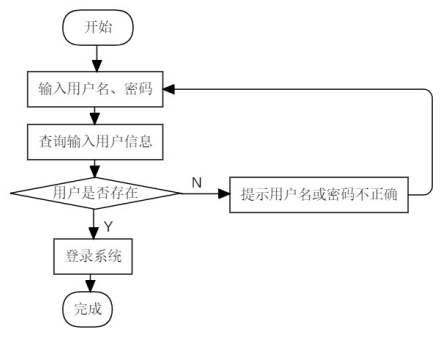


图2-2 登录流程图

### 2.5.2 存款

存款功能的活动图如图2-5所示。

存款时的账号

存款时输入的数值

查询存款余额

输入数值加入余额内

图2-5 功能活动图

### 2.5.3 取款

存款功能的活动图如图2-6所示。

取款时的账号

取款时输入的数值

### 

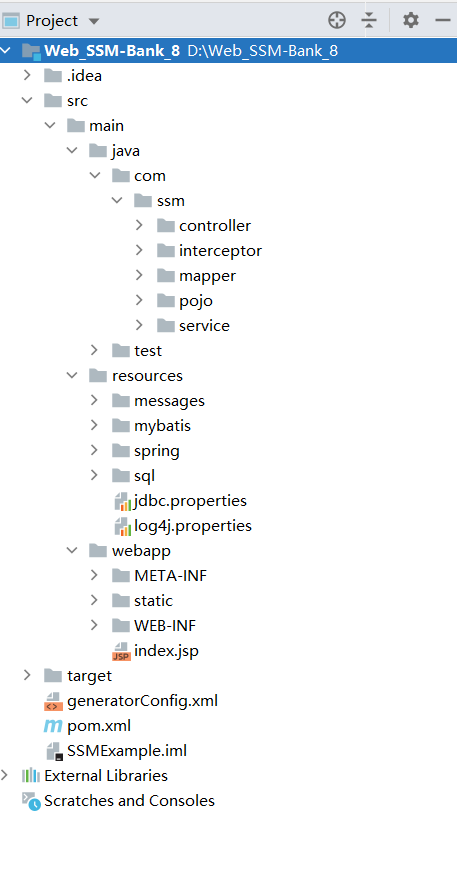
查询存款余额

存款减去输入的数值

图2-6 功能活动图

# 3 系统实现

## 3.1 项目结构



## 3.2 配置文件

### 3.2.1 jdbc.properties文件

该配置文件主要是以文件形式保存数据库的驱动类名称，连接数据库的URL地址，访问数据库的用户名及对应的密码，程序运行时会读取该文件相关信息，避免硬编码，当相关信息发生变化时，只需修改配置文件而不用修改源代码，增加程序的可扩展型。

**jdbc.driver**=**com.mysql.cj.jdbc.Driver  
jdbc.url**=**jdbc:mysql://127.0.0.1:3306/develop?characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai&rewriteBatchedStatements=true  
jdbc.username**=**root  
jdbc.password**=**root  
c3p0.minPoolSize**=**1  
c3p0.maxPoolSize**=**5  
c3p0.initialPoolSize**=**1  
c3p0.acquireIncrement**=**1**

### 3.2.2 log4j.properties文件

描述该文件作用，并对里面的配置进行解释

**log4j.rootLogger** = **info,stdout  
  
log4j.appender.stdout** = **org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target** = **System.out  
log4j.appender.stdout.layout** = **org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern** = **[%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n  
  
log4j.appender.D** = **org.apache.log4j.DailyRollingFileAppender  
log4j.appender.D.File** = **D://develop/SSMExample/logs/log.log  
log4j.appender.D.Append** = **true  
log4j.appender.D.Threshold** = **DEBUG  
log4j.appender.D.layout** = **org.apache.log4j.PatternLayout  
log4j.appender.D.layout.ConversionPattern** = **%-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n  
  
log4j.appender.E** = **org.apache.log4j.DailyRollingFileAppender  
log4j.appender.E.File** =**D://develop/SSMExample/logs/error.log  
log4j.appender.E.Append** = **true  
log4j.appender.E.Threshold** = **ERROR  
log4j.appender.E.layout** = **org.apache.log4j.PatternLayout  
log4j.appender.E.layout.ConversionPattern** = **%-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n  
  
log4j.appender.I** = **org.apache.log4j.DailyRollingFileAppender  
log4j.appender.I.File** =**D://develop/SSMExamples/logs/info.log  
log4j.appender.I.Append** = **true  
log4j.appender.I.Threshold** = **INFO  
log4j.appender.I.layout** = **org.apache.log4j.PatternLayout  
log4j.appender.I.layout.ConversionPattern** = **%-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n**

### 3.2.3 mybatis-config.xml文件

描述该文件作用，并对里面的配置进行解释

*<?***xml version="1.0" encoding="UTF-8"** *?>***<!DOCTYPE configuration  
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
 "http://mybatis.org/dtd/mybatis-3-config.dtd"*>***<**configuration**>  
  
 <**settings**>  
 *<!-- 指定 MyBatis 所用日志的具体实现 -->* <**setting name="logImpl" value="LOG4J"** />  
 *<!-- 是否开启自动驼峰命名规则 -->* <**setting name="mapUnderscoreToCamelCase" value="true"** />  
 *<!-- 延迟加载的全局开关 -->* <**setting name="lazyLoadingEnabled" value="true"**/>  
 *<!-- 按需加载对象 -->* <**setting name="aggressiveLazyLoading" value="false"**/>  
 </**settings**>  
  
 *<!-- 配置分页插件 -->  
<!-- <plugins>-->  
<!-- <plugin interceptor="com.github.pagehelper.PageHelper">-->  
<!-- &lt;!&ndash; 设置数据库类型 Oracle,Mysql,MariaDB,SQLite,Hsqldb,PostgreSQL六种数据库&ndash;&gt;-->  
<!-- <property name="dialect" value="mysql"/>-->  
<!-- &lt;!&ndash;<property name="dialect" value="oracle"/>&ndash;&gt;-->  
<!-- </plugin>-->  
<!-- </plugins>-->*</**configuration**>

## 示例代码Controller

BankcardController.java

**package** com.ssm.controller;  
  
**import** com.ssm.controller.converter.BankcrakDateConverter;  
**import** com.ssm.controller.validation.ValidGroupByBankcard;  
**import** com.ssm.pojo.Bankcard;  
**import** com.ssm.pojo.Transaction;  
**import** com.ssm.service.BankcardService;  
**import** com.ssm.service.TransactionService;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.ui.Model;  
**import** org.springframework.validation.BindingResult;  
**import** org.springframework.validation.ObjectError;  
**import** org.springframework.validation.annotation.Validated;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.ResponseBody;  
**import** org.springframework.web.bind.annotation.SessionAttributes;  
  
**import** java.util.Date;  
**import** java.util.HashMap;  
**import** java.util.List;  
**import** java.util.Map;  
  
@Controller  
@SessionAttributes({**"bankcard"**,**"list"**,**"Tlist"**,**"bankObjectErrors"**})  
**public class** BankcardController {  
  
 @Autowired  
 **private** BankcardService bankcardService;  
 @Autowired  
 **private** TransactionService transactionService;  
  
 @RequestMapping(**"insertBankcard"**)  
 **public** @ResponseBody  
 Bankcard insertBankcard(@Validated(ValidGroupByBankcard.**class**) Bankcard bankcard, BindingResult bindingResult, Model model){  
 *//进行校验* **if** (bindingResult.hasErrors()){  
 List<ObjectError> objectErrors = bindingResult.getAllErrors();  
 model.addAttribute(**"bankObjectErrors"**,objectErrors);  
 }  
 *//效验通过后进行主体操作* Bankcard newBankcard = **null**;  
 **try** {  
 bankcard.setCardId(bankcardService.getCardId());  
 newBankcard = bankcardService.insertBankCard(bankcard);  
 model.addAttribute(**"bankcard"**,newBankcard);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** newBankcard;  
 }  
  
 @RequestMapping(**"findBankcardAll"**)  
 **public** @ResponseBody  
 List<Bankcard> findBankcardAll(Bankcard bankcard,Model model){  
 List<Bankcard> list = **null**;  
 **try** {  
 list = bankcardService.findBankCardAll(bankcard);  
 model.addAttribute(**"list"**,list);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** list;  
 }  
  
 @RequestMapping(**"deleteBankcard"**)  
 **public** @ResponseBody Map<String,String> deleteBankcard(Bankcard bankcard,Model model){  
 Map<String,String> map = **new** HashMap<String, String>();  
 Bankcard newBankcard = **null**;  
 **try** {  
 newBankcard = bankcardService.deleteBankCard(bankcard);  
 findBankcardAll(bankcard,model);  
 **if** (newBankcard!=**null**)  
 map.put(**"msg"**,**"删除银行卡成功！"**);  
 **else** map.put(**"msg"**,**"删除银行卡失败！"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** map;  
 }  
  
 @RequestMapping(**"findBankcard"**)  
 **public** @ResponseBody Bankcard findBankcard(Bankcard bankcard ,Model model){  
 Bankcard newBankcard = **null**;  
 **try** {  
 newBankcard = bankcardService.findBankCardByCardId(bankcard);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** newBankcard;  
 }  
  
 @RequestMapping(**"reportLossBankcard"**)  
 **public** @ResponseBody Bankcard reportLossBankcard(Bankcard bankcard,Model model){  
 Bankcard newBankcard = **null**;  
 **try** {  
 newBankcard = bankcardService.findBankCardByCardId(bankcard);  
 **if** (newBankcard!=**null**);  
 newBankcard.setReportLoss(1);  
 newBankcard = bankcardService.updateBankCard(newBankcard);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** newBankcard;  
 }  
 @RequestMapping(**"updateAddBankcard"**)  
 **public** @ResponseBody Map<String,String> updateAddBankcard(Bankcard bankcard ,Model model){  
 Bankcard newBankcard = **null**;  
 Map<String,String> map = **new** HashMap<String, String>();  
 **try** {  
 newBankcard = bankcardService.findBankCardByCardId(bankcard);  
 **if** (bankcard.getPassword().equals(newBankcard.getPassword())) {  
 **if** (newBankcard.getReportLoss() != 1) {  
 newBankcard.setBalance(newBankcard.getBalance() + bankcard.getBalance());  
 newBankcard = bankcardService.updateBankCard(newBankcard);  
 Transaction transaction = **new** Transaction();  
 transaction.setCardId(newBankcard.getCardId());  
 transaction.setTransanctionDate(**new** Date());  
 transaction.setTransanctionType(**"存入"**);  
 transaction.setTransanctionPay(bankcard.getBalance());  
 transaction.setRemarks(**"银行卡为："** + newBankcard.getCardId() + **"的用户存入现金:"** + bankcard.getBalance() + newBankcard.getCurrencyType());  
 transactionService.insertTransaction(transaction);  
 map.put(**"msg"**,**"存款成功！"**);  
 } **else** {  
 map.put(**"msg"**,**"该用户为挂失状态，无法进行取款操作！"**);  
 }  
 } **else** {  
 map.put(**"msg"**,**"支付密码错误！"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** map;  
 }  
  
 @RequestMapping(**"updateRemBankcard"**)  
 **public** @ResponseBody Map<String,String> updateRemBankcard(Bankcard bankcard ,Model model){  
 Bankcard newBankcard = **null**;  
 Map<String,String> map = **new** HashMap<String, String>();  
 **try** {  
 newBankcard = bankcardService.findBankCardByCardId(bankcard);  
 **if** (bankcard.getPassword().equals(newBankcard.getPassword())) {  
 **if** (newBankcard.getReportLoss() != 1) {  
 newBankcard.setBalance(newBankcard.getBalance() - bankcard.getBalance());  
 newBankcard = bankcardService.updateBankCard(newBankcard);  
 Transaction transaction = **new** Transaction();  
 transaction.setCardId(newBankcard.getCardId());  
 transaction.setTransanctionDate(**new** Date());  
 transaction.setTransanctionType(**"取出"**);  
 transaction.setTransanctionPay(bankcard.getBalance());  
 transaction.setRemarks(**"银行卡为："** + newBankcard.getCardId() + **"的用户取出现金:"** + bankcard.getBalance() + newBankcard.getCurrencyType());  
 transactionService.insertTransaction(transaction);  
 map.put(**"msg"**,**"取款成功！"**);  
 } **else** {  
 map.put(**"msg"**,**"该用户为挂失状态，无法进行取款操作！"**);  
 }  
 } **else** {  
 map.put(**"msg"**,**"支付密码错误！"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** map;  
 }  
  
 @RequestMapping(**"Transfer"**)  
 **public** @ResponseBody Map<String,Boolean> Transfer(String cardId,Double Tmoney,String TcardId,Model model){  
 Map<String,Boolean> map = **new** HashMap<String, Boolean>();  
 Boolean flag = **false**;  
 **try** {  
 flag = bankcardService.Transfer(cardId,Tmoney,TcardId);  
 **if** (flag) {  
 Bankcard bankcard1 = **new** Bankcard();  
 bankcard1.setCardId(cardId);  
 Bankcard newBankcard1 = bankcardService.findBankCardByCardId(bankcard1);  
 Bankcard bankcard2 = **new** Bankcard();  
 bankcard2.setCardId(TcardId);  
 Bankcard newBankcard2 = bankcardService.findBankCardByCardId(bankcard2);  
 Transaction transaction = **new** Transaction();  
 transaction.setCardId(cardId);  
 transaction.setTransanctionDate(**new** Date());  
 transaction.setTransanctionType(**"取出"**);  
 transaction.setTransanctionPay(Tmoney);  
 **if** (newBankcard1.getCurrencyType().equals(**"人民币"**)&&newBankcard2.getCurrencyType().equals(**"人民币"**)) {  
 transaction.setRemarks(**"银行卡为："** +cardId+**"的用户使用"**+Tmoney+**"人民币"**+**"转给银行卡为:"**+TcardId+**"的用户"**+Tmoney+**"人民币"**);  
 }**else if** (newBankcard1.getCurrencyType().equals(**"美元"**)&&newBankcard2.getCurrencyType().equals(**"美元"**)) {  
 transaction.setRemarks(**"银行卡为："** + cardId + **"的用户使用"**+Tmoney+**"美元"** + **"转给银行卡为:"** + TcardId + **"的用户"** + Tmoney + **"美元"**);  
 }**else if** (newBankcard1.getCurrencyType().equals(**"美元"**)&&newBankcard2.getCurrencyType().equals(**"人民币"**)) {  
 transaction.setRemarks(**"银行卡为："** + cardId + **"的用户使用"**+Tmoney+**"美元"** + **"转给银行卡为:"** + TcardId + **"的用户"** + Tmoney \* 6 + **"人民币"**);  
 }**else if** (newBankcard1.getCurrencyType().equals(**"人民币"**)&&newBankcard2.getCurrencyType().equals(**"美元"**)) {  
 transaction.setRemarks(**"银行卡为："** + cardId + **"的用户使用"**+Tmoney+**"人民币"** + **"转给银行卡为:"** + TcardId + **"的用户"** + Tmoney / 6 + **"美元"**);  
 }  
 transactionService.insertTransaction(transaction);  
 transaction.setCardId(TcardId);  
 transaction.setTransanctionDate(**new** Date());  
 transaction.setTransanctionType(**"存入"**);  
 **if** (newBankcard1.getCurrencyType().equals(**"人民币"**)&&newBankcard2.getCurrencyType().equals(**"人民币"**)) {  
 transaction.setRemarks(**"银行卡为："** +cardId+**"的用户使用"**+Tmoney+**"人民币"**+**"转给银行卡为:"**+TcardId+**"的用户"**+Tmoney+**"人民币"**);  
 transaction.setTransanctionPay(Tmoney);  
 }**else if** (newBankcard1.getCurrencyType().equals(**"美元"**)&&newBankcard2.getCurrencyType().equals(**"美元"**)) {  
 transaction.setRemarks(**"银行卡为："** + cardId + **"的用户使用"**+Tmoney+**"美元"** + **"转给银行卡为:"** + TcardId + **"的用户"** + Tmoney + **"美元"**);  
 transaction.setTransanctionPay(Tmoney);  
 }**else if** (newBankcard1.getCurrencyType().equals(**"美元"**)&&newBankcard2.getCurrencyType().equals(**"人民币"**)) {  
 transaction.setRemarks(**"银行卡为："** + cardId + **"的用户使用"**+Tmoney+**"美元"** + **"转给银行卡为:"** + TcardId + **"的用户"** + Tmoney \* 6 + **"人民币"**);  
 transaction.setTransanctionPay(Tmoney \* 6);  
 }**else if** (newBankcard1.getCurrencyType().equals(**"人民币"**)&&newBankcard2.getCurrencyType().equals(**"美元"**)) {  
 transaction.setRemarks(**"银行卡为："** + cardId + **"的用户使用"**+Tmoney+**"人民币"** + **"转给银行卡为:"** + TcardId + **"的用户"** + Tmoney / 6 + **"美元"**);  
 transaction.setTransanctionPay(Tmoney / 6 );  
 }  
 transactionService.insertTransaction(transaction);  
 }  
 map.put(**"msg"**,flag);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** map;  
 }  
  
 @RequestMapping(**"findTransactionAll"**)  
 **public** String findTransactionAll(String cardId ,Model model){  
 List<Transaction> list = **null**;  
 **try** {  
 list = transactionService.findTransactionAll(cardId);  
 model.addAttribute(**"Tlist"**,list);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return "/home/page6"**;  
 }  
}

**更多示例见项目源代码及注释，限于页面不一一列举**

# 5 课程总结

**技术提炼总结**

### 5.1 Spring

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。 简单来说，Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

### 5.2 SpringMVC

SpringMVC：它用于web层，相当于controller(等价于传统的servlet和struts的action)，用来处理用户请求。举个例子，用户在地址栏输入https://网站域名/login，那么springmvc就会拦截到这个请求，并且调用controller层中相应的方法，(中间可能包含验证用户名和密码的业务逻辑，以及查询数据库  
操作，但这些都不是springmvc的职责)，最终把结果返回给用户，并且返回相应的页面(当然也可以只反馈josn/xml等格式数据)。springmvc就是做前面和后面过程的活，就是一个页面拦截器。

### 5.3 Mybatis

**MyBatis** 是支持普通 SQL查询，存储过程和和高级映射的优秀持久层框架。**MyBatis** 使用简单的 XML或注解用于配置和原始映射，将接口和 Java 的POJOs（Plain Ordinary Java Objects，普通的 Java对象）映射成数据库中的记录。

**课程学习完了，感谢聂刚老师的精彩授课，聂刚老师的讲课非常精彩，这门课在一些细节上比其他教材更为清晰更为实际、更为有用，具有很强的指导价值，非常感谢聂刚老师！**