# 拿去花APP的系统实现

1. 系统功能结构

系统在设计之初就将扩展性作为一个重要因素考虑在内，在充分理解该金融交易平台的功能需求后，对功能进行了模块化的划分，如下图所示：



图2.3 模块功能图

核心交易模块主要提供了涉及交易的核心功能，其中包括：创建借款Agreement，对某一Agreement进行投资，绑卡，Account账户金额充值，Account账户金额提现，借款资质资料的提交（包括身份信息和证明材料）。

在客户管理模块中，提供了对平台用户的查看和管理的功能。

在社交关系模块中，提供了通讯录导入与识别功能、好友投资关系查看功能。通过Cordova调用智能手机的通讯录功能，将通讯录信息上传至服务器，通过了导入与识别模块将会检测和匹配上传的通讯录和系统中注册用户的电话号码进行匹配，一旦检测到A用户上传的通讯录中包含有平台注册用户B，那么即将B添加到A的好友列表中，使A可以查看到B的投资关系，包括可以查看到B发起的所有借款请求，并可以再详情中进行投资。

在客户通信模块中，提供了可与平台用户进行通信的功能。通信方式采用邮件的方式，在客户账户发生变动的时候，能够通过邮件的形式将账户变动信息告知给用户。

OPEN API模块对外提供各种服务，服务采用HTTP协议完成数据交换。设计优秀运行稳定的系统，需要对外提供服务，以此使服务利用率最大化。

1. 合同-账户-交易模型

金融交易平台系统具有逻辑复杂、可靠性要求高和时效性要求高等特点，为了保证业务的可用性，在核心交易逻辑的实现中需要遵循一定的模型规则，由此提出了拿去花APP基于合同-账户-交易的系统模型。

金融系统具有高度复杂性，需要有一个抽象易用的系统架构模型来指导技术实现，本文在广泛查阅和理解分析资料的基础上，提出了合同-账户-交易的金融交易系统架构模型。如下图所示：



图 1..1 合同-账户-交易模型

在合同-账户-交易模型中，每一个用户（User）即为平台上的一个注册用户，享有该金融交易平台提供的各种服务，例如投资提现功能。每一个User都有一个虚拟的账户（Account），这个Account是User在平台上资金流动的结果的汇总体现，里面包含User的账户余额和银行卡等信息。另一方面，用户可以在平台上发起借款请求，这里的每一个借款请求就对应图1.2中的一个Agreement，这个Agreement包含借款人的借款请求，针对该Agreement进行投资的所有投资操作的集合。而针对Agreement的每一笔操作，比如发起借款，进行投资都会生成一条AgreementRequest，在AgreementRequest中记录了对Agreement产生的变更的详细记录，比如Agreement的ID，变更类型（借款或投资），金额，时间，操作用户等信息。而该AgreementRequest将会驱动User的Account资金进行相应的增减。

该模型具有系统模块低耦合性和功能高度抽象性，能够较好地降低金融交易系统的业务逻辑复杂性，有效地降低了高效快速开发出稳定有效的系统难度。

1. 交易模块实现
2. 绑卡模块实现

绑卡模块提供用户绑卡功能，用户可以添加和删除银行卡的绑定。

1. 类与接口定义

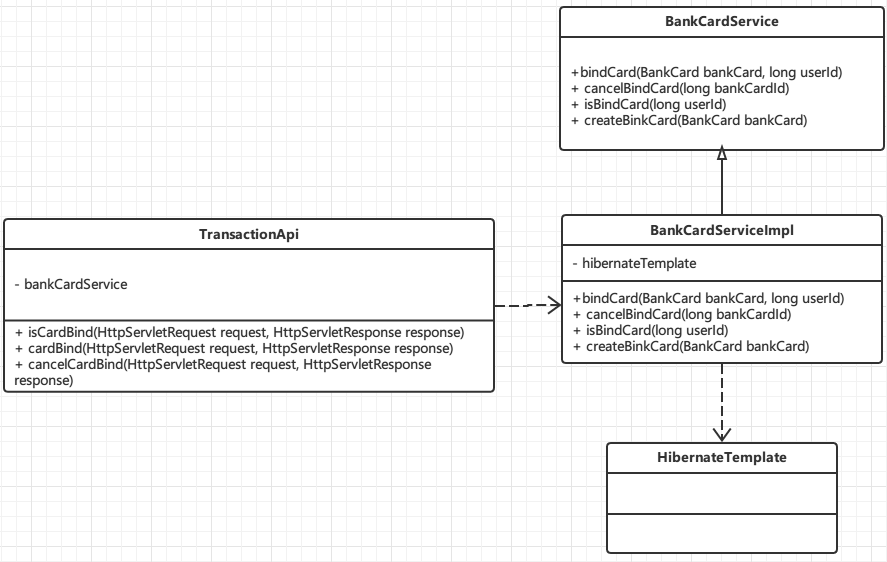
绑卡模块的类与接口设计如图1.2所示。

图1.一.2 绑卡模块类图

HibernateTemplate是Spring对Hibernate框架的一个模板类，对Hibernate框架对数据持久化的方法的封装，提供对象数据的增删改查功能。

TransactionApi是交易系统的开放接口服务API，主要接收HTTP请求，并对请求做出相应响应。绑卡模块需要调用其isCardBind、cardBind和cancelCardBind三个方法。

isCardBind提供一个查询方法，根据request请求中接收到的用户userId查询该用户是否已经绑卡，返回是否绑卡的状态信息。前台据此状态判断，是否允许其进行绑卡操作。

cardBind提供一个绑卡接口，该方法会接受和处理request请求中的绑卡信息：银行卡号，用户userId等，通过对这些信息的处理为该用户新增一张银行卡绑定。

cancelCardBind接收request请求信息，对该用户的银行卡进行解绑操作。

BankCardService是一个接口。

BankCardServiceImpl实现了BankCardService接口，该类主要负责对用户的银行卡进行一系列操作：查询是否绑卡、新建一张银行卡、将某银行卡和某用户绑定起来、将某银行卡和某用户解除绑定。

1. 部分实现代码

在绑卡模块中，需要对request请求中的有效信息进行一定的验证，比如绑卡用户是否存在，是否已经绑卡，该卡是否被其他用户绑定等限制条件做一定的验证。部分代码如下所示。

@Transactional  
 public void bindCard(BankCard bankCard, long userId) {  
 Assert.notNull(bankCard.getBankCardNo());  
 Assert.notNull(bankCard.getBankOwenerName());  
 hibernateTemplate.save(bankCard);  
 Account account = (Account) hibernateTemplate.find("from Account a where a.userId = ?", userId).get(0);  
 account.setBankCardId(bankCard.getId());  
 hibernateTemplate.update(account);  
 }  
 public boolean isBindCard(long userId) {  
 List<Account> list = (List<Account>) hibernateTemplate.find("from Account a where a.userId = ?", userId);  
 if (CollectionUtils.isEmpty(list)) {  
 throw new NQHException("用户不存在");  
 } else {  
 Long bankCardId = list.get(0).getBankCardId();  
 if (bankCardId != null && bankCardId != 0) {  
 return true;  
 }  
 }  
 return false;

}

}

1. 投资、借款和还款模块实现

借款资质审核通过的用户可发布借款请求，绑卡用户可在平台进行账户余额的充值，可在账户余额充足的情况下对平台上的借款请求进行投资。平台会定期清查到期借款请求，并通知借款人进行还款操作，借款人还款完成后，会自动分发本息金额到个投资人手中。

1. 类与接口定义

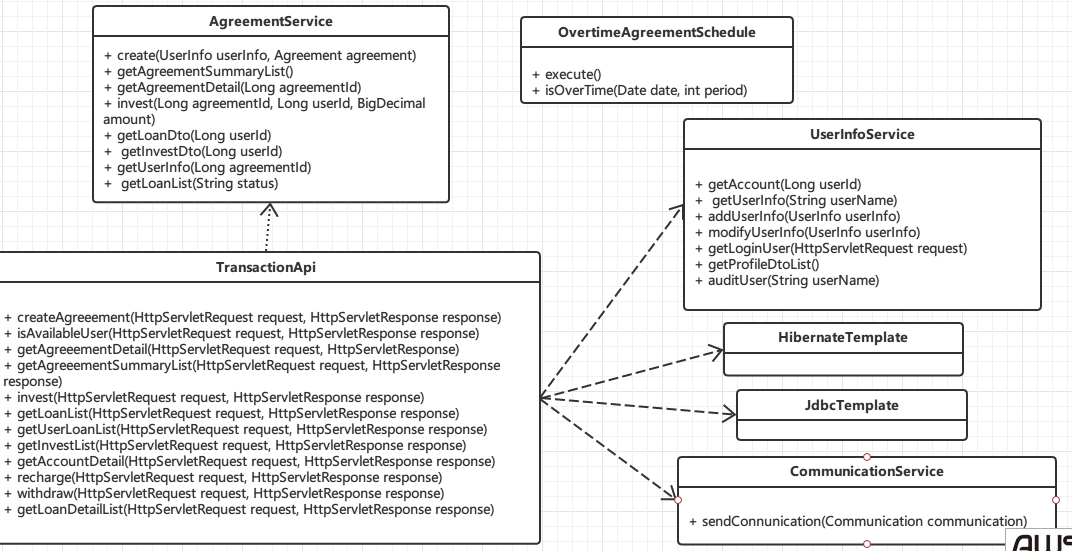
投资、借款和还款模块的类和接口定义如下图所示。

图 1.一.3 投资、借款和还款模块类图

TransactionApi：通过获取http请求中的信息，实现了对投资、借款和还款的功能。

createAgreeement：创建一个新的借款合同信息。

isAvailableUser：验证用户资质审核是否通过，是否有权限发布借款合同。

invest：投资，对某一特定的借款合同投资相应的金额。

recharge：账户余额充值。

withdraw：账户余额提现。

getAgreeementDetail：获取借款合同详情列表信息。

getAgreeementSummaryList：获取借款合同概要列表信息。

getUserLoanList：得到某用户的借款列表。

getAccountDetail：得到用户的账户详情，包括账户姓名、绑定的银行卡号、账户余额等。

AgreementService：合同交易管理服务类，主要提供对合同的一系列操作管理。

create：创建一个新的借款合同。

getAgreementSummaryList：得到所有借款合同摘要列表。

getAgreementDetail：得到所有借款合同详情列表。

invest：投资借款合同。

getLoanDto：根据条件得到借款列表。

OvertimeAgreementSchedule：定时任务类，主要任务为定时扫描所有借款列表是否满足借款到期条件。如果满足条件，则进行邮件催款操作。

JdbcTemplate：是Spring对JDBC的封装，提供了JDBC操作的一系列便捷方法。

CommunicationService：是客户通信服务类，该类提供了向特定客户发送邮件消息对的功能。

UserInfoService：是客户信息管理的服务类，提供了对客户基本信息和资质信息的管理。

1. 部分实现代码

部分实现代码如下所示。

系统实现过程中需要注意如下的功能逻辑：

* 投资借款的过程中，需要做如下验证：账户余额是否充足、投资的借款是否已经筹满等；
* 创建借款请求的时候，需要对用户的借款资质审核是否通过进行验证；
* 借款到期后需要根据相应合同的借款年利率进行自动催款和还款；
* 用户的每一笔资金流动操作都要通过Comunication模块告知用户资金表动情况。

//还款金额计算规则：(月利率/100+1)\*(借款天数/30）[向上取整]\*已筹集金额  
BigDecimal repaymentAmount = (new BigDecimal(1).add(agreement.getRateMonthly().divide(new BigDecimal(100)))).multiply(new BigDecimal(Math.ceil(agreement.getRepaymentLimit() / 30))).multiply(agreement.getRaisedAmount());  
BigDecimal newAmount = account.getAmount().subtract(repaymentAmount);  
  
AccountEntry accountEntry = new AccountEntry();  
accountEntry.setCreatedOn(new Date());  
accountEntry.setAmount(new BigDecimal(0).subtract(agreement.getAmount()));  
accountEntry.setAccountId(account.getId());  
accountEntry.setType(AccountEntryType.REPAYMENT.getValue());  
  
hibernateTemplate.save(accountEntry);  
  
account.setAmount(newAmount);  
agreement.setStatus(AgreementStatus.REPAYMENT\_COMPLETED.getValue());  
hibernateTemplate.update(account);  
  
logger.info("自动还款成功，正在通知。agreementId=" + agreement.getId());

1. 账户管理模块

账户管理模块提供用户的注册登陆、资料修改、资质的上传和修改等功能。

1. 类与接口定义

账户管理模块的类与接口定义如下图所示。

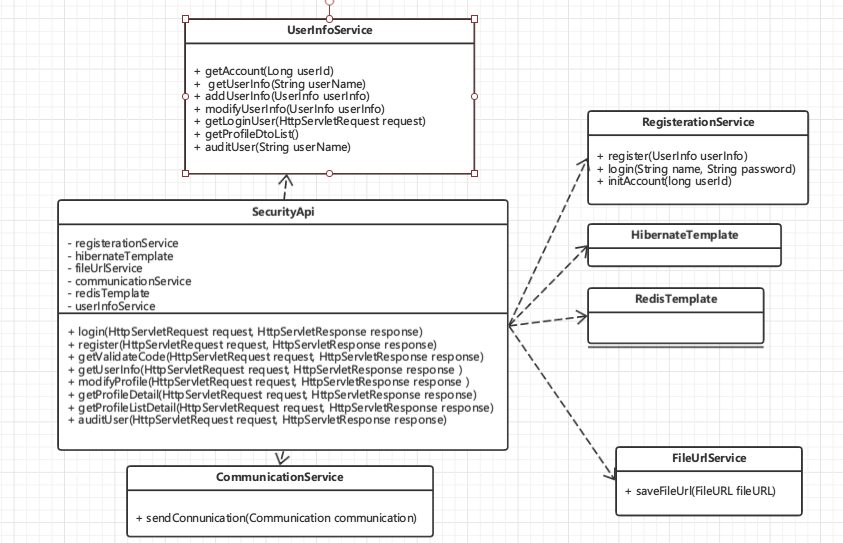
SecurityApi：提供用户账户管理相关的API接口调用。主要包括用户的登陆和注册、获取注册验证码、得到用户信息、提交和更改资质信息、审核用户资质功能。

图1..4 账户管理类图

login：验证用户的登陆请求是否合法。

register：注册一个新用户。

getValidateCode：注册时根据邮箱获取一个验证码，并将该验证码以一定的时效性存储在redis缓存中。

getUserInfo：获得用户信息。

modifyProfile：更改用户账户信息。

auditUser：审核用户的贷款资质。

RegisterationService：登陆注册服务类。

register：注册一个新用户，将新用户相关信息进行持久化。

login：验证用户登陆请求是否合法。

initAccount：新用户登陆成功，初始化一个Account账户实例。

FileUrlService：用户上传的资质文件服务类。文件化保存上传的图片，并对其基本信息，如文件类型、文件大小和URL等信息，进行持久化。

RedisTemplate：Spring对redis的操作封装类，可以方便地对redis数据库进行操作。

1. 部分实现代码

系统部分实现代码如下。

@RequestMapping(value = "/login", method = RequestMethod.GET)  
public void login(HttpServletRequest request, HttpServletResponse response) throws ServletRequestBindingException, IOException {  
  
 ResponseUtil responseUtil = new ResponseUtil();  
  
 try {  
 String userName = ServletRequestUtils.getStringParameter(request, "username");  
 String password = ServletRequestUtils.getStringParameter(request, "password");  
 boolean isLogin = registerationService.login(userName, password);  
 HttpSession session = request.getSession();  
 if (isLogin) {  
 logger.info("一个新用户登陆成功 = " + userName + ", password = " + password);  
  
 //1h登陆失效  
 redisTemplate.opsForValue().set(session.getId(), String.valueOf(userInfoService.getUserInfo(userName).getId()), 3600, TimeUnit.SECONDS);  
  
 responseUtil.setCode("OK");  
 responseUtil.setMsg(session.getId());  
 response.getWriter().print(responseUtil.toString());  
 return;  
 } else {  
 logger.info("A user login field, the username = " + userName + ", password = " + password);  
 responseUtil.setCode("FAILED");  
 responseUtil.setMsg("登陆失败");  
 response.getWriter().print(responseUtil.toString());  
 return;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 responseUtil.setCode("FAILED");  
 responseUtil.setMsg("登陆失败");  
 response.getWriter().print(responseUtil.toString());  
 }  
}

1. 好友模块实现

好友关系模块提供好友的添加和查看好友借款的功能。

1. 类与接口定义

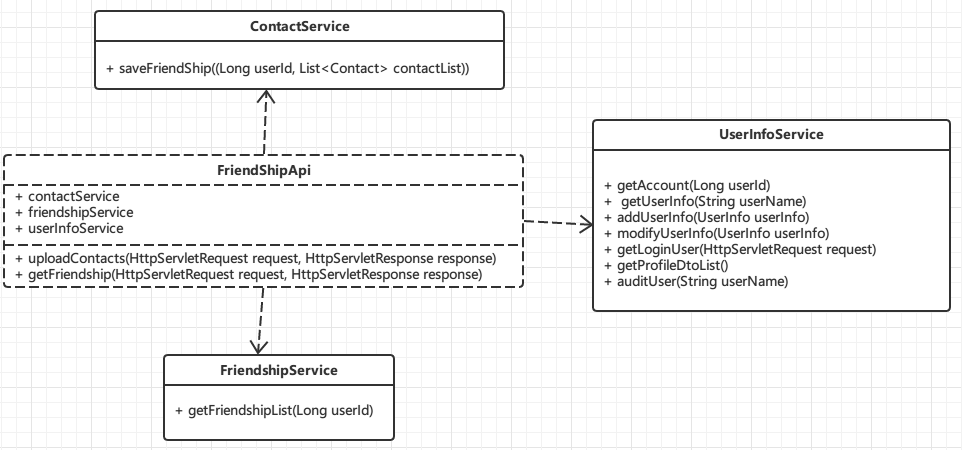
好友模块的类与接口定义图如下所示。

图1.5 好友管理类图

FriendShipApi：好友关系管理API，主要提供上传通讯录列表，实现自动匹配好友功能，和获得用户的好友列表功能。

uploadContacts：获得HTTP请求中的通讯录列表，通过逻辑解析，将通讯录电话号码和平台用户电话号码进行匹配，如有匹配的，则自动建立通讯录上传者和用户该电话用户的单向好友关系。建立好友关系后可以查看好友的借款情况。

getFriendship：获得用户的好友列表。

FriendshipService：好友关系服务类，该类提供获取好友列表的功能。

ContactService：通讯录功能服务类，可以对用户上传的通讯录信息进行解析，并通过电话号码匹配自动建立好友关系。

1. 部分实现代码

@Transactional  
public void saveFriendShip(Long userId, List<Contact> contactList) {  
 Iterator<Contact> it = contactList.iterator();  
 while (it.hasNext()) {  
 Contact next = it.next();  
 ArrayList<UserInfo> userInfoList = (ArrayList<UserInfo>) hibernateTemplate.find("from UserInfo u where u.phoneNo = ?", next.getPhoneNumber());  
 if (userInfoList.size() != 0) {  
 Friendship friendship = new Friendship();  
 friendship.setCreatedOn(new Date());  
 friendship.setUserId(userId);  
 friendship.setFriendId(userInfoList.get(0).getId());  
  
 if (!isExistedFriendShip(userId, userInfoList.get(0).getId())) {  
 hibernateTemplate.save(friendship);  
 }  
 }  
 }  
}

1. 系统安全设计
2. 用户访问系统权限管理

系统实行强制的权限管理，未登陆用户不能对平台上的任何资源进行查看或修改。对所有资源的访问都需要进行登陆权限认证，且每类资源都有相应的权限级别。系统划分两类权限级别：普通用户和管理员。普通用户可以对自己拥有的资源进行操作，如自己创建的借款合同和账户信息。管理员可以对平台的所有资源进行管理，比如借款资质的审核、用户的管理等。

通过实行强制性的登陆验证和权限划分，使系统的安全性得到一定的保障。

1. 密码单向加密存储（MD5加密）

MD5是一种单向加密的消息摘要算法，本系统的所有密码信息都是在MD5加密后再将密文持久化到数据库中的。这种方法很好地保护了用户的密码明文。

1. OPEN API权限认证

系统后台服务运行在Tomcat容器中，客户端可以通过HTTP服务调用的方式发起服务请求，并得到响应。后台服务会对客户端的每一次请求进行权限认证，具体认证流程为：

用户在登陆成功后，服务器会返回一串32为的随机字符串，我们称之为access\_token，access\_token和用户的userId一起作为一对键值对保存在redis缓存中，并且有一定的有效期，对应于账户登录的有效期概念。而后用户的每一次请求，对需要在请求参数中带上access\_token的值，服务器会根据access\_token的值查找登陆的userId，如果找不到则拒绝资源请求。通过这种方式实现了对OPEN API系统调用的权限认证。

Java: 是一种跨平台、面向对象的编程语言，目前已广泛运用于处理企业级的复杂业务场景中。在本文搭建的金融交易的复杂业务系统中，选用Java语言进行后台逻辑的编写，具有运行稳定高效、维护方便和跨平台的优势。

Spring：Spring是一个开源的Java应用框架，该框架拥有强大的控制反转和对象生命周期管理的功能，同时也提供面向切面的编程实现，也集成了如SpringMVC、SpringJDBC等丰富的功能。本文利用Spring实现了和Hibernate框架的继承以实现对对象的持久化操作，利用Spring实现控制反转使系统易于维护，利用SpringMVC实现了应用的MVC编程风格。

Hibernate：Hibernate是Java平台下的一种可靠高效的对象关系映射的解决方案，提供了一个将面向面向对象的领域模型到传统关系型数据库的转化实现方案。本文使用Hibernate框架实现了将系统对象和关系型数据库的映射功能，实现了对对象的持久化操作。

运行环境及条件：

后台服务：

JDK Version >= 1.7

Redis Vesion = 2.4.2

操作系统：Windows, Unix, OSX跨平台

APP应用：

Android Version >= 5.0

响应时间：

金融系统业务由于其高效性和及时性都有很高的要求，本文要求所有系统操作的响应时间< 1秒

存储速度：

本文系统采用MySql作为数据库，通过添加索引等操作，使系统的存储速度达到单机条件下500+次每秒。

安全性：

为了保证系统的安全性，要求所有资源访问都必须进行权限认证。所有密码信息都必须以密文的形式存储。

可靠性：

本文搭建的是一个高可用性的金融交易平台。要求系统的服务可用率达到90%以上。