基于微服务框架的众筹平台融资宝的设计与实现

摘要

21 世纪，作为一种新兴的网络金融模式——“众筹融资”进入了众人的视线，并且获取了各个阶层人群的喜爱，并且逐步成为了中小型企业活力和个人发展创业积极性的催化剂。众筹过程中，一个投资项目拥有多个的投资人，还有多个借贷项目的发起人，如何妥善的管理交易过程，并且保证资金流动的安全性和透明性，就变得尤为重要。这样众筹平台就担当起了管理的职责，一方面保证了资金流动和存储过程的安全性和透明性；另一方面保证了借款人的“还款计划”和投资人的“回款计划”计算的准确性。

论文设计的众筹平台——融资宝，按照众筹的类型，属于借贷模式。个人借钱给一个项目或企业，按照银行规定的借贷规则，如期得到本金并且获取一定的利息报酬；这个过程中，借款人是单体，出借人是群体或单体。这种模式也被称为 P2P 借贷。

论文首先对众筹平台融资宝进行了系统需求分析，为了更好的适配众筹过程中，借款人和出借人资金交易过程的需要，满足平台的性能需求，了解项目启动的硬件约束；对系统架构进行设计，包含前后端技术选型、第三方 API 使用、消息中间键以及数据库的确定。然后分析系统中各个技术栈特点、使用中比较适合的场景；通过这一系列初步的分析和设计，体现微服务框架的适配性。然后通过总体设计和核心功能的详细设计和实现，来体现微服务框架为系统带来的高可用。

在最后对本平台进行了全面的测试，一方面保证了平台各个服务功能达到需要的指标，另一方面了解平台的并发能力。通过测试结果的验证，确保了众筹平台的安全交付和稳定上线，确保了平台的高可用性。

关键字：众筹平台 微服务框架 高可用

DESIGN AND IMPLEMENTATION OF FINACING TREASURE OF CROWDFUNDING PLATFORM BASED ON MICROSERVICE FRAMEWORK

Abstract

In the 21st century, as a new online financial model, "crowdfunding" has entered the public's attention and won the love of people from all walks of life. It has gradually become a catalyst for the vitality of small and medium-sized enterprises and the initiative of individual development and entrepreneurship. In the process of crowdfunding, an investment project has multiple investors and sponsors of multiple lending projects. How to properly manage the transaction process and ensure the safety and transparency of capital flows has become particularly important. In this way, the crowdfunding platform has assumed the management responsibility, on the one hand, ensuring the safety and transparency of the capital flow and storage process; On the other hand, the calculation accuracy of the borrower's "repayment plan" and the investor's "repayment plan" is guaranteed.

The crowdfunding platform designed in this paper, Financing Treasure, is a lending model according to the types of crowdfunding. Individuals who lend money to a project or an enterprise, in accordance with the lending rules prescribed by the bank, receive the principal on schedule and receive a certain interest payment; In this process, the borrower is a single entity and the lender is a group or a single entity. This model is also known as P2P lending.

Firstly, the paper makes a systematic demand analysis of the crowdfunding platform, in order to better adapt to the needs of the borrower and the lender in the capital transaction process in the crowdfunding process, to meet the platform's performance requirements, and to understand the hardware constraints of project startup; The system architecture is designed, including front-end and back-end technology selection, third-party API use, message intermediate key and database determination. Then analyze the characteristics of each technology stack in the system and the more suitable scenes in use; Through this series of preliminary analysis and design, the adaptability of the microservice framework is reflected. Then through the overall design and the detailed design and implementation of the core functions, the high availability brought by the micro-service framework for the system is reflected.

At the end of the paper, the platform is tested comprehensively, which not only ensures that each service function of the platform reaches the required index, but also understands the concurrency of the platform. Through the verification of the test results, the safe delivery and stable online launch of the crowdfunding platform were ensured, and the high availability of the platform was ensured.

KEY WORDS: micro-service framework of crowdfunding platform is highly available

目录

摘要 1
Abstract 2
第一章 引言 5
1.1 课题研究背景 5
1.2 研究现状 5
1.3 论文的内容与结构 6
第二章 系统的技术栈介绍 7
2.1 Spring Boot 微框架介绍 7
2.2 Spring Cloud 微服务框架介绍 7
2.2.1 Nacos 介绍 7
2.2.2 Sentinel 介绍 8
2.2.3 Ribbon 介绍 8
2.2.4 RabbitMQ 介绍 8
2.2.5 Getway 介绍 8
2.3 Redis 介绍 9
2.4 Swagger 接口设计介绍 9
第三章 系统需求分析 10
3.1 系统用户角色分析 10
3.1.1 借款人和投资人 10
3.1.2 平台管理员 11
3.2 系统功能需求分析 12
3.2.1 登录注册服务 13
3.2.2 充值提现服务 14
3.2.3 会员管理服务 14
3.2.4 短信服务 14
3.2.5 数据字典和积分等级管理服务 14
3.2.6 借款管理服务 14
3.2.7 标的管理服务 15
3.3 系统非功能需求分析 15
第四章 系统的总体设计 16
4.1 微服务架构设计 16
4.2 业务流程设计 17
4.3 数据库设计 18
4.4.1 E-R 图设计 19
4.4.2 数据库表设计 20
第五章 系统搭建和核心业务功能实现 28
5.1 系统搭建 28
5.1.1 后台管理平台搭建 28
5.1.2 后端模块搭建 31
5.1.3 SpringCloud 基础设施 32
5.2 核心业务功能实现 36
5.2.1 Redis 缓存数据字典 36

5.2.2 短信微服务和用户注册 38
5.2.3 贷后 42
5.2.4 贷前 45
第六章 系统功能验证与性能测试 49
6.1 Swagger 系统功能验证 49
6.2 JMeter 系统性能测试 50
第七章 总结与展望 52
7.1 论文总结 52
7.2 项目展望 52
参考文献 53
致谢 54

第一章 引言

1.1 课题研究背景

21 世纪以来，一种新兴的网络金融模式——“众筹融资”进入了众人的视线，具备了互联网金融操作简便、高效、透明的优点，摆脱了传统金融市场的局限性，缩短了交易的距离，提升了金融交易的效率，改善了金融市场的发展局限性，促进了网络金融模式的发展。但是作为一种快速发展的网络金融模式，不仅给投资者带来的收益的机会，同时还带来了诸多问题和风险。

凭借在线化和技术化的优点，网络金融具备独具一格的竞争力和创造性。但是本质还是一种金融模式，不仅需要对借贷人信用评定、还要对金融产品可能存在的风险性进行评估。对比于传统金融模式，网络金融在互联网下进行操作。这是一把双刃剑，在提高了交易效率和金融产品数字计算准确性的同时，还夹带了这类互联网通信的通病，资金安全性、并发问题、系统宕机恢复等，都难以控制和避免。因此针对此类问题的防患和避免，需要一个完整的体系框架去专门的解决。

论文就此设计了一个基于微服务框架的众筹平台融资宝，处理这类网络金融模式可能出现的问题和风险。

1.2 研究现状

从世界范围来看，众筹融资在美国得到推广开始，快速得到了世界各地的效仿，在美国的相关机构承认了这一个网络金融模式的在金融行业的地位，并且也成为了世界最大的、体系最完备的众筹市场。而中国的众筹模式相对起步较晚，但在今日具有代表性的众筹行业——众筹网。据不完全统计，我国截至 2022 年共有 354 家众筹平台，而北京作为众筹行业的开拓地，平台聚集比较明显。

今日众筹行业发展前景仍然良好，2021 年国内行业总值超过了 1000 亿，且稳定持续上升发展。据公开显示，众筹网从上线至今共发起了 356 个众筹项目，累计参与人数 60423 人次，共计筹集资金超过 1.8 亿。国内对于众筹模式的探究整体也越来越清楚，我们的项目往往需要借款人的注入一定的情感或者实际价值，这样才能更容易得到投资人的认可和选择。

然而众筹平台为融资者和投资人建立起来的网络社交门户，在提供便利同时也带来了突出的问题，给金融发展和监管带来巨大挑战。高发的平台欺诈现象，2011 年的“贝尔投创”平台无法访问，涉嫌人卷钱消失，造成投资人损失。平台运营难度难以估计，2013 年的“众贷网”上线仅一个月宣布倒闭，管理团队缺乏众筹经验，未能把控好风险，给投资人造成无法挽回的损失。易卷入非法经营活动，众筹平台以 P2P 形式投资的名义在实际经营众将债券包装成理财产品，通过平台进行集资，演变为非法集资，突破了传统意义的 P2P 借贷模式。

因此众筹平台的设计也要妥善且合法的经营，将众筹的经验融合到平台管理中，简化人员的操作。同时使用了微服务框架技术，将我们的众筹的流程微粒化拆分为多个独立的微服务，这样类似于传统金融模式，每个业务流程可以各司其职，互不干扰，只用接收或传递相应的信息，通过微服务的组件专门的解决业务流程出现的棘手问题，以及平台运行

中出现的并发等问题。

1.3 论文的内容与结构

本平台是一个基于微服务框架的众筹平台融资宝的实现。主要分析众筹模式的利弊，可能出现的比较棘手的互联网金融的问题，如何使用微服务框架及相关中间键等技术栈去集中处理在众筹的业务流程中、平台上线后可能出现的问题。所以，本文会对每一个主要的技术栈介绍它的使用目的和为平台带来的收益，并且结合高可用的背景，让我们的平台使用的技术栈更加符合现实意义，然后介绍核心的业务流程的实现方法和配合到的技术栈，最后就是系统调试和测试阶段。为了平台更安全、可靠的运行上线。

具体章节安排如下：

第一章，引言。主要介绍论文课题研究的背景，研究现状，还有论文的内容和结构。

第二章，系统需求分析。

第三章，系统的技术栈介绍。

第四章，系统的总体设计。

第五章，系统搭建和核心业务功能实现。

第六章，系统功能验证和性能测试。

第七章，总结与展望。

第二章 系统的技术栈介绍

本章是对众筹平台的系统使用的技术栈进行介绍，通过微服务架构的设计，保证系统实现过程的技术支持。包括 Spring Boot 微框架、Spring Cloud 微服务框架、Redis 缓存、Swagger 接口设计。

2.1 Spring Boot 微框架介绍

说到 SpringBoot 就要先了解 Spring，Spring 作为一个“万能胶水”，把常用的技术进行合理的封装和设计，可以快速集成和开发，Spring 的核心是控制饭庄和依赖注入，在快速集成的同时也带来了依赖过多、配置太多、运行部署繁琐的问题，尽管可以通过注解缓解，但是本质问题没有解决。如何让开发者无需关注这些，专注于业务开发，Spring Boot 就诞生了。

Spring Boot 核心思想就是“约定优于配置”，最主要也就是 Starter 组件完成的自动装配，扫描约定目录下的文件进行扫描解析，把得到的配置类导入，完成 Bean 的自动装配。这种软件设计范式目的就是减少配置数量或者降低理解难度，从而提高开发效率。

而 Spring Cloud 生态中的组件，都是基于 Spring Boot 来实现的，也就是说 Spring Cloud 开发离不开 Spring Boot 的支持。

2.2 Spring Cloud 微服务框架介绍

了解 Spring Cloud 先了解 SOA（Service Oriented Architecture）面向服务的架构，是面向服务开发的雏形，可以理解为微服务的超集。SOA 注重的是可重用性（服务的复用）和信息孤岛问题，而微服务关注的是解耦，降低业务之间的耦合度。

微服务框架技术鲜明的特点就是技术选型灵活、可扩展性高、独立部署、容错性。一个优秀的微服务框架要致力于解决服务注册与发现、服务路由、服务调用、负载均衡、熔断器、分布式消息。Spring Cloud 只是一套规范，而 Spring Cloud Netflix、Spring Cloud Alibaba 才是实现。

而本次众筹平台的实现涉及的到的技术有 Nacos（分布式配置中心和服务注册与发现）、Sentinel（限流、熔断与降级）、Ribbon（负载均衡）、OSS（阿里云对象存储）、Getway（微服务网关）、RabbitMQ（分布式消息通信）、Skywalking（分布式链路追踪）。下面对一些比较重要的组件进行简单的介绍。

2.2.1 Nacos 介绍

Nacos 致力于解决微服务中统一配置、服务注册与发现等问题。微服务系统的每一个微服务都是一个单独的模块，随着微服务数量的增多，依赖关系选择和确定让系统更加复杂，把所有的服务注册到 Nacos 中，通过服务列表展示更加方便地配置和管理服务，同时 Nacos 还提供了实时健康检查可以阻止不健康服务发送请求，帮助系统发现、配置和管理服务。服务注册完成后，系统就可以更敏捷的找到对应的服务接口，而 Nacos 也可以对一些统一的配置文件进行维护推送到对应的服务节点实现动态更新。

2.2.2 Sentinel 介绍

互联网应用中，会有突发性的高并发访问场景，就像双十一或者大型秒杀活动，这些场景的特点就是超出系统处理的并发数。所有如果没有合理的保护机制，大量的流量都访问到我们的服务器就会造成宕机，造成不可避免地损失，而为了避免这些问题发生，服务限流、降级、熔断就可以很好的处理这些问题。

服务限流通过限制并发访问数或者限制一个时间窗口允许处理的请求数量来保护系统，一旦达到限制条件，就会对请求采取对应的拒绝策略，例如降级或跳转到错误界面。从本质上说就是损失一部分用户可用性，来保证大部分用户服务的可靠性。服务熔断为了解决服务堆积造成的雪崩效应，往往一个请求操作在微服务系统中需要多个微服务之间配合才能完成，但是在高并发中往往会出现一个服务链其中一个服务无响应，造成请求堆积，引起服务雪崩。服务熔断通常采用的策略有平均响应时间、异常比例、异常数。
通过 Sentinel 解决了高并发情况下服务访问的问题。

2.2.3 Ribbon 介绍

Ribbon 是一个客户端负载均衡器，把客户端服务请求合理的分配到一种服务的不同节点上，保证微服务架构的高可用性，合理的选择负载均衡算法可以减少请求访问系统的响应时间。Ribbon 提供常用的负载均衡实现有 RandomRule（随机法）、RoundRobinRule（轮询法）、WeightedResponseTimeRule（权重策略）、ZoneAvoidanceRule（默认策略）。通过 Ribbon 更好的处理微服务架构并发条件下多个请求访问服务分配策略，保证系统的高可用，降低系统响应时间。

2.2.4 RabbitMQ 介绍

微服务架构下，各个服务之间通过通信完成整体的功能。例如一个众筹交易过程，核心是交易，而交易成功的消息通知不是核心步骤，则可以剥离出来做异步处理。就可以使用 RabbitMQ，它的特点是低延迟、高可用、易使用的分布式消息队列，其中强大的顺序消息、消息重试、定时消息、追踪等功能，契合了众筹模式中对消息通知的需求。而 RabbitMQ 就是通过自己独特的内部机制保证了消息在传递过程中的可达性、准确性、可靠性。

2.2.5 Getway 介绍

网关类似于“门面”所有的外部请求都会经过网关这一层，不仅仅只是请求的转发和服务的整合，还提供了统一的鉴权、限流、熔断、日志，协议转换（后端对不同的协议，统一处理后以 HTTP 对外通过服务）等功能，同时网关的使用也屏蔽了非授权请求的入侵，是维护系统稳定性的第一层保护。Getway 作为一种网关技术，对请求进行路由转发，以及对请求进行前置或后置的过滤（鉴权和限流），保证系统的安全性和稳定性。

2.3 Redis 介绍

作为一种 NOSQL 数据库，最核心的特点就是数据间没有关系，使得系统的拓展性和读写能力大大提高。而他另一个特点是数据放在内存中，采用非阻塞 I/O，并且是单线程的，这样的设计，让数据的读写速度是高效的。通过 redis 把常用的数据存入缓存中，减少数据库和服务器内存的压力，提升服务的访问速度。另外 redis 提供了数据过期策略，对于系统中 token 或一些登录信息，尤其是短信验证码有时间限制，通过 redis 可以定时自动清理，如果过期，数据就不存在，刚好满足系统的功能需求，既不会影响系统数据库性能，还让系统功能更加方便、高效。

2.4 Swagger 接口设计介绍

无论是前端还是后端开发，对于程序员来说不是自己编写的功能或者自己的领域，想要正确的调用远程服务或者接口，往往需要一个规范的接口文档参考，才能去实现。Swagger 就是这样的一个工具对于 REST API 定义一个标准的接口，让人可以理解远程服务，并且更为方便的直接与远程服务进行交互，如果系统更新迭代，只需要更新 Swagger 重新生成文档。同时 Swagger 提供了 API 自动生成和在线测试，输入对应的参数即可测试接口，让系统开发过程更高效、简单。

第三章 系统需求分析

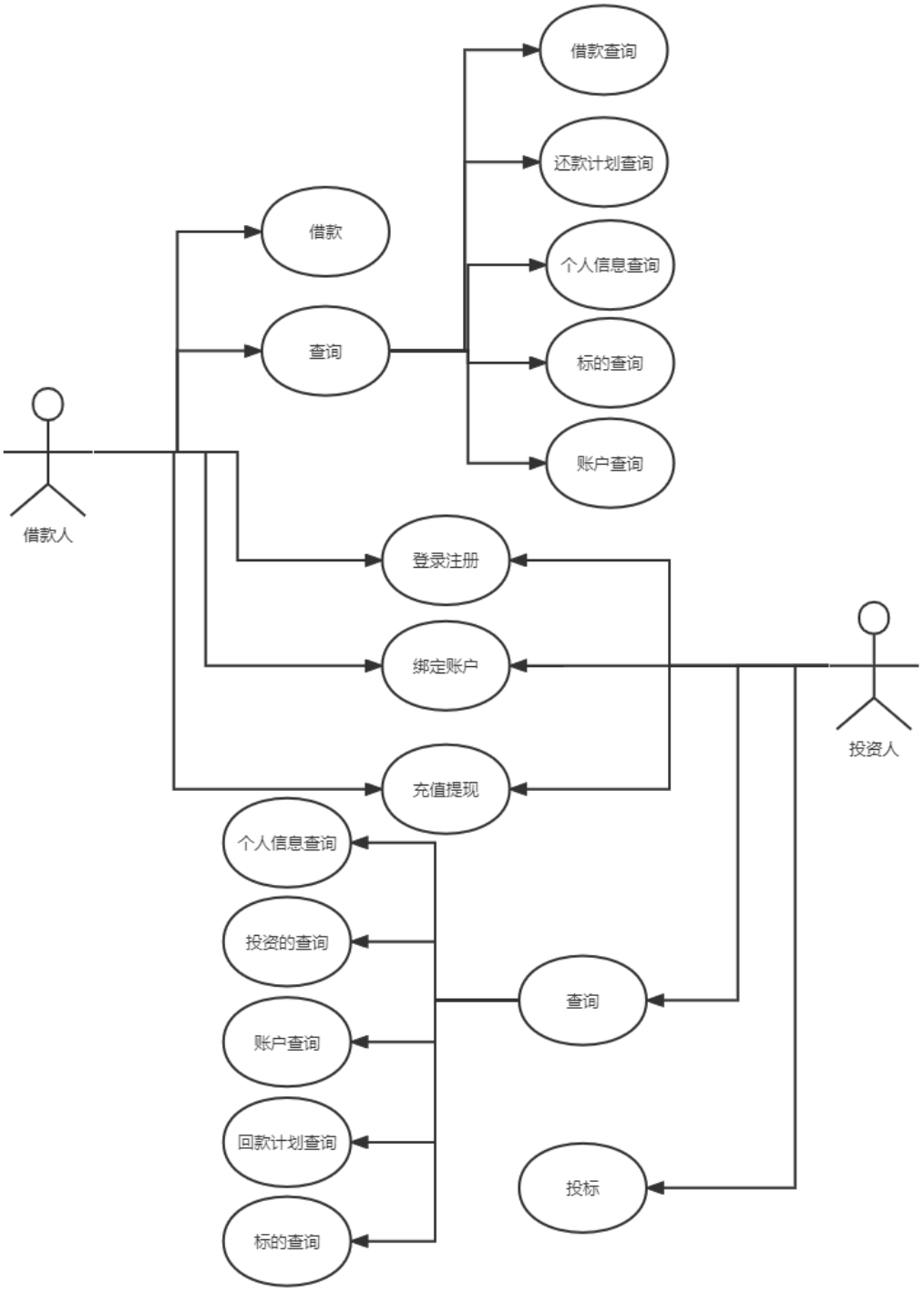
需求分析是系统开发过程的关键，主要工作就是将抽象化需求变成面向真实情况的分析过程。本章节从众筹模式的三个主体来进行分析，分别是借款人、投资人、平台管理员。通过三个主体要在众筹平台要满足的基本功能为起点，对系统需求进行详细的分析得到真实情况中各主体的真实需求。借款人的需求就是得到投资，定期还款；平台管理员的需求就是认真的审核系统信息，提高交易的可信度，降低存在的风险性；投资人的需求就是选择可信度高的项目投资获得收益，定期得到回款。所以系统只要满足三个主体的需求，协调三者功能就能基本满足一个众筹平台的需要。然后通过对平台上线指标分析了平台所需的非功能需求。

3.1 系统用户角色分析

3.1.1 借款人和投资人

借款人作为众筹平台核心的“生产者”，借款人的需要是进入平台发布自己的融资项目。第一步要先通过电话注册账户，正确填写基础信息，选择自己的身份借款人，得到短信验证码，输入正确完成注册；登录平台时选择借款人进入平台；绑定自己的卡号，完善自己信用的信息，通过审核后得到一定的信用额度，但是如果审核失败，则需要重新填写，然后借款人发布自己的项目，通过审核后，在平台主页即可浏览到自己的融资项目，否则需要重新填写发布融资项目。得到充足的投资后，管理放贷，钱款流入借款人账户；借款人选择提现，之后按照自己选择的还款方式定期还款，如果账户余额不足，可以选择充值，融资项目的全部还款计划归还完毕后整个流程结束。

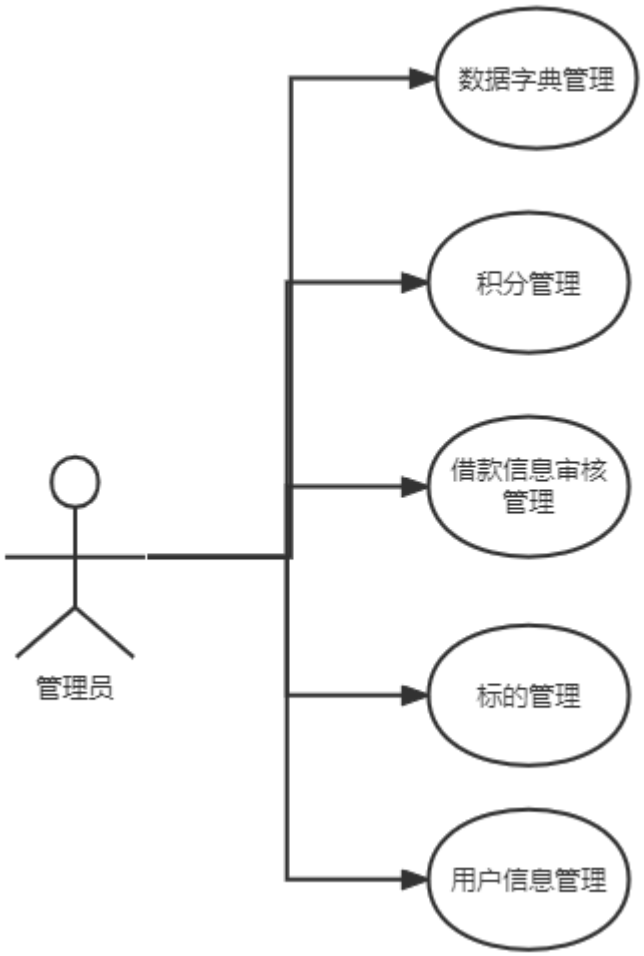
投资人作为众筹平台核心的“消费者”，投资人的需要是进入平台选择自己想要投资的项目。第一步先通过电话注册账户，正确填写投资人的基础信息，通过电话得到有效的验证码后完成注册；选择投资人登录平台，绑定自己有效的银行卡，充值账户后就可以选择融资项目进行投资；如果项目成功放款，即可查询自己的回款信息，定期得到回款，投资项目的全部回款计划结束后整个流程结束。



借款人和投资人的 UML 图

3.1.2 平台管理员

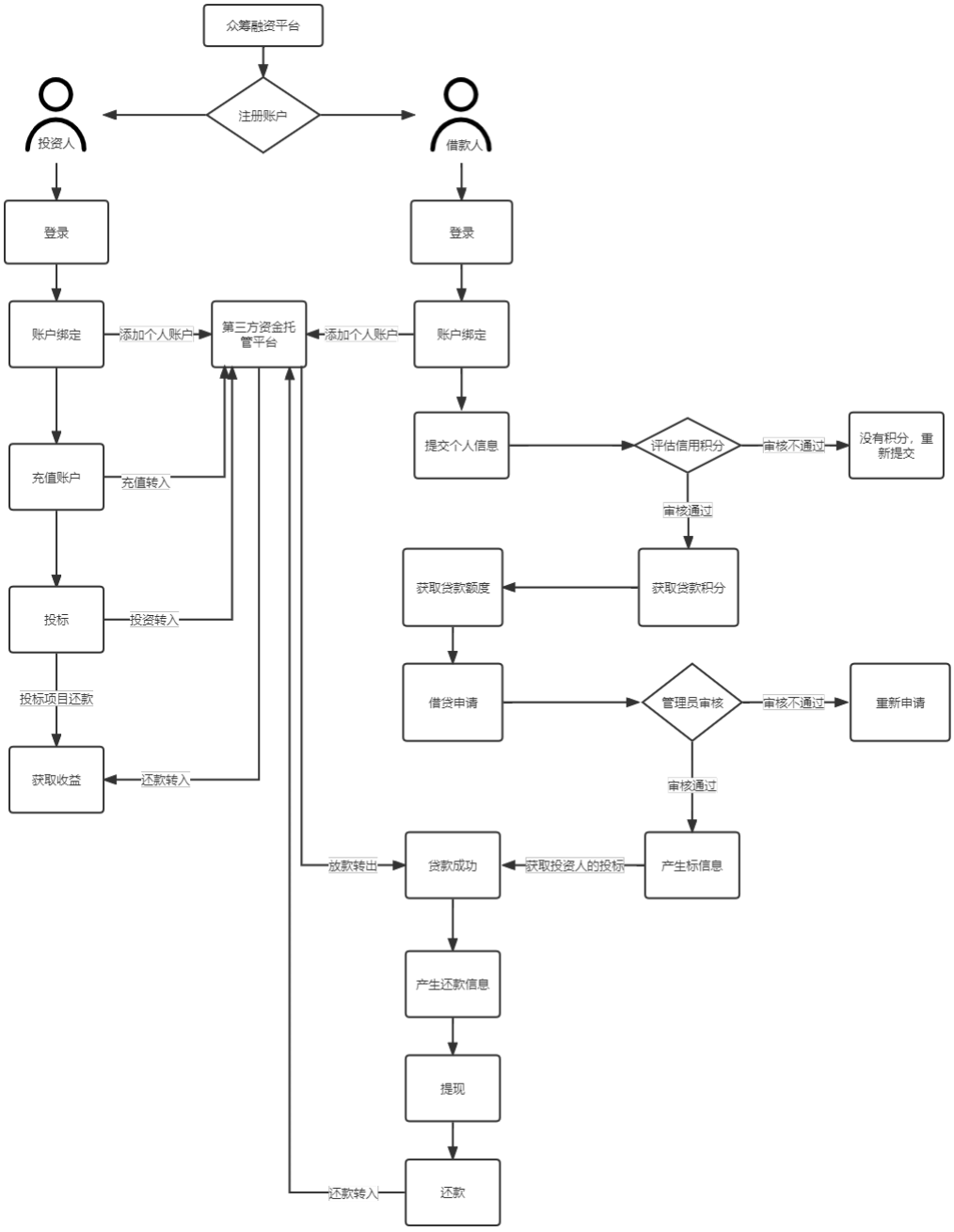
平台管理员作为众筹平台的“警察”，管理员的需要是维护信用积分的贷款力度；维护平台中的数据字典；及时处理投资人、借款人信息真伪；根据平台规则审核融资项目的可信度，保证投资人全款安全；融资项目信息查询，满标放款；查询用户登录信息；用户账户权限管理。



管理员的 UML 图

3.2 系统功能需求分析

对系统功能需求进行分析，还是要结合融资模式的三个主体，大部分的功能需求都是为了满足主体的需求。系统核心的功能分为登录注册服务、充值提现服务、会员管理服务、短信服务、数据字典和积分等级管理服务、标的管理服务、借款管理服务。系统功能和主体关系的 UML 图如下。



系统功能和主体关系的 UML 图

3.2.1 登录注册服务

登录注册服务包括管理员的后台登录、投资人和借款人的前台登录和注册。主要是前台服务，用户区分投资人和借款人，便于身份确定，注册统一采用电话短信验证，便于平台通过短信通知重要的消息，而注册要填写主要的信息，包括用户名、常用电话、密码。密码要加密，电话要接收注册的验证码，正确填写后跳转登陆界面，通过电话号码、用户身份、密码登录，通过后台的身份验证后成功访问平台主页。未完成身份验证没有权限访问其他核心业务界面。

3.2.2 充值提现服务

充值提现服务包括用户绑定银行卡、用户完善信息、用户充值和用户提现。本众筹平台采用了第三方资金托管方——汇付包，平台的只提供投资人、借款人交易的业务，但是资金交由第三方托管，避免平台管理问题，造成不必要的损失。用户需要提供银行卡号、预留电话、真实姓名、支付密码完成绑定。绑定成功才可以进行与资金有关的业务。未绑定成功则无法充值和提现。投资人无需完善详细信息，借款人只有完善信息填写性别、学历、身份证号、身份证正反面照片、车辆信息、房产信息等，来用于额度审核。充值和提现以银行卡交易为工具，需要跳转到第三方资金交易页面进行处理。

3.2.3 会员管理服务

会员管理服务包括会员列表展示、会员账户的登录日志、账户锁定与解锁、用户信息多条件查询。会员列表从数据库查询出平台所有会员的基本信息，用户注册成功后就可以查询到，包括用户类型、昵称、电话等；登录日志是用户每次登录的 IP 地址和时间显示；账户锁定后无法登录，联系管理员解锁方可登录；会员列表信息展示可以通过手机号、用户类型、账户状态多条件查询。

3.2.4 短信服务

短信服务包括注册验证短信、提现通知、充值通知等与资金和验证有关的短信信息。短信功能使用阿里云短信服务，通过提供的 API 快速实现发短信功能。不同的功能要有不同的提示模板，比如验证码由后台生成用于验证，资金有关的要有真实的金额显示。同时因为资金交易通知比较重要，要保证短信的发送可达。

3.2.5 数据字典和积分等级管理服务

数据字典和积分等级管理服务包括数据字典的展示、导入、导出，积分等级区间的展示、添加、删除的功能。数据字典用于保存平台中存在的常用分类或者一些固定数据，更好的帮助我们获取和适用这些数据。但是考虑到数据量比较大，关系比较复杂的情况，平台需要有 Excel 批量导入和导出的功能。积分等级管理是用于评估借款人的可信度，获取相应的贷款额度，也就是积分和额度的对应关系的管理，如果不满足现状可以新增、删除、修改。

3.2.6 借款管理服务

借款管理服务包括融资项目信息列表、借款人信息查询、借款人发布融资信息、管理员审核融资信息的功能。融资项目信息列表展示借款人名、手机、金额、借款时间、还款方式（包含等额本息、等额本金、每月还息一次还本、一次还本还息）、还款年利率、借款状态、借款时间；借款人列表展示姓名、电话、身份证号、性别、认证状态、申请时间、身份证正反面照片、房产信息、车辆信息、积分详情，还可以通过电话、手机、身份证号

多条件查询借款人信息；借款人则发布融资信息包含借款用途、还款方式、年利率、借款金额（不能超过用户的借款额度，否则需要完善积分信息重新评估借款额度），满足条件后提交有后台审核，审核通过回显在借款进度条上，如果不通过进度条显示不通过，重新提交融资信息；管理原审核融资信息就需要管理员查询信息是否合理，借款用途是否合乎情理，选择是否通过。

3.2.7 标的管理服务

标的管理服务包括投资人投资项目（标的生成）、标的列表、标的详情、满标放款的功能。投资人选择项目后填写投资金额（不能超过自己账户余额），可以直接根据项目的年利率和还款方式显示出投资人的预计收益和当前项目得到的投资，最后确定投资；投资成功后显示出借款项目的标的，包含标的金额、投资期数、年利化率、已投资金和人数、发布开始和结束时间、当前标的状态；标的详情就是标的基础信息和借款人的信息的拼接展示出来；满标放款就是融资项目得到预期的钱款审核无误后放款，投资的钱款会到借款人的账户中，然后平台就要计算收益和利息，定期发送还款和回款计划。

3.3 系统非功能需求分析

对于众筹平台来说，保证正常众筹模式业务进行的同时，在交易过程要和第三方资金托管平台频繁的资金流动要保证通知信息的及时和可达，另外还有系统安全性，系统响应时间，最大并发量，服务熔断处理，系统高可用度等非功能的需求。

交易资金的安全交由第三方资金托管平台，但是通知的信息通过异步通知、消息队列、失败重试机制的方法，一方面降低资金交易过程的系统响应时间，另一方面保证信息的可靠发送。

系统的安全性，众筹平台一些特定的接口比如个人中心（包含余额、还款计划、回款计划等），这些需要一定的权限验证，才能安全访问，用户登录通过了短信验证和单点登录设计，通过验证后得到一个 token 信息，使用 JWT（JWT 头、有效载荷、签名哈希，有效载荷就是 JWT 的主体内容部分）对 Token 信息进行防伪，这样通过算法加密保存入浏览器中，用户访问服务只需要带着 token 信息就可以通过单点登录的鉴权，访问其他服务无需再次鉴权。大大降低服务压力和节省响应时间，提高用户体验。

而最大并发和熔断处理，保证高可用性，则是通过微服务框架技术提供的 Getway、Nacos、RestTemplate+Ribbon 更好的处理这些问题。

第四章 系统的总体设计

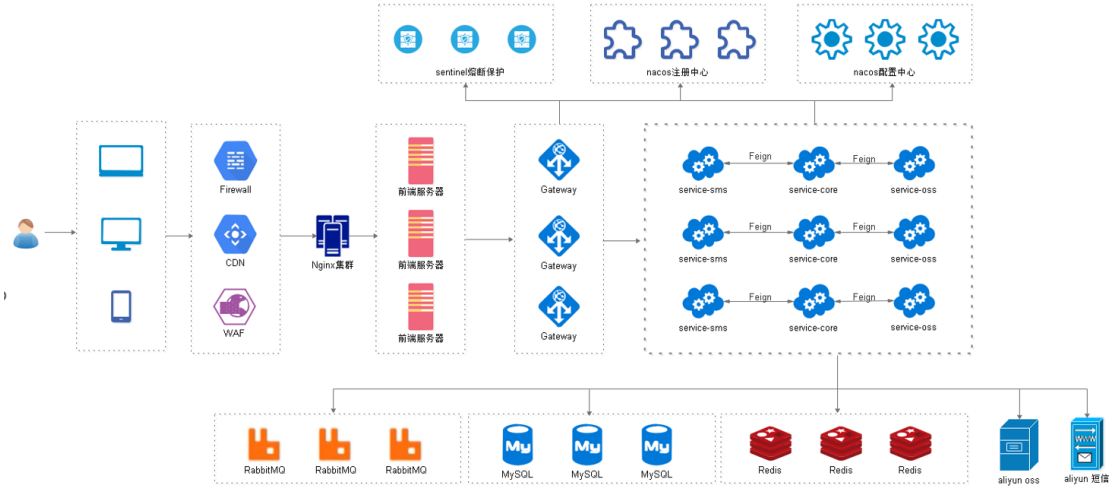
本章将根据系统需求分析，从微服务架构设计、业务流程设计、数据库设计三个方面进行系统的总体设计。

4.1 微服务架构设计

技术作为业务的支撑，技术的更新换代往往是跟随业务需求的改变而进步。传统的单体架构已无法满足高用户量和高访问量的需求，服务器的负载越来越大，业务也越来越复杂。所以本系统采用微服务架构设计，把原来复杂的业务拆分，使系统解耦。但是系统的业务比较于真实的众筹平台较为简单，分为了 service-sms（短信服务）、service-gatway（网关服务）、service-oss（对象存储服务）、service-core（融资宝核心服务）。Service-core 中就包含了登录注册服务、充值提现服务、会员管理服务、数据字典和积分等级管理服务、借款管理服务、标的管理服务。这些服务都注册再 Nacos 中，通过列表的形式展示。

主体在设备上通过前端服务器发起请求后，通过 Getway 网关将请求发送到对应的服务节点，完成相应的功能。服务的本身以及服务的核心配置都再 Nacos 中，被微服务的其他组件共同治理。服务间通过 OpenFeign 远程服务调用，OpenFeign 是再 Feign 基础上加了 Spring MVC 注解支持，例如@RequestMapping、@GetMapping 和 @PostMapping。系统使用 Sentinel 进行熔断保护，避免高并发情况下某些微服务宕机，造成大量的请求堆积引起服务雪崩，最终让系统崩溃。并且系统的服务都注册在 Nacos 中系统之间的调用关系层次递进，如果链路其中一个环节中断，导致请求失败，可以通过 SkyWalking 链路追踪到对应服务失败节点，跟利于系统排查维护的工作。

区分于不同的实体，大量的关系数据保存在 Mysql 关系型数据库中；一些系统常用的数据或时效性的数据则保存在 Redis 中，作为缓存数据。系统中的资金通知、登录验证码、注册验证码、提现通知、资金到账通知都是通过核心服务的远程调用顺序的进入消息队列，然后调用短信服务。而对象存储服务只是用来服务核心业务，用于系统的图片存储功能。整个微服务项目选择通过 Gitee 进行项目管理。

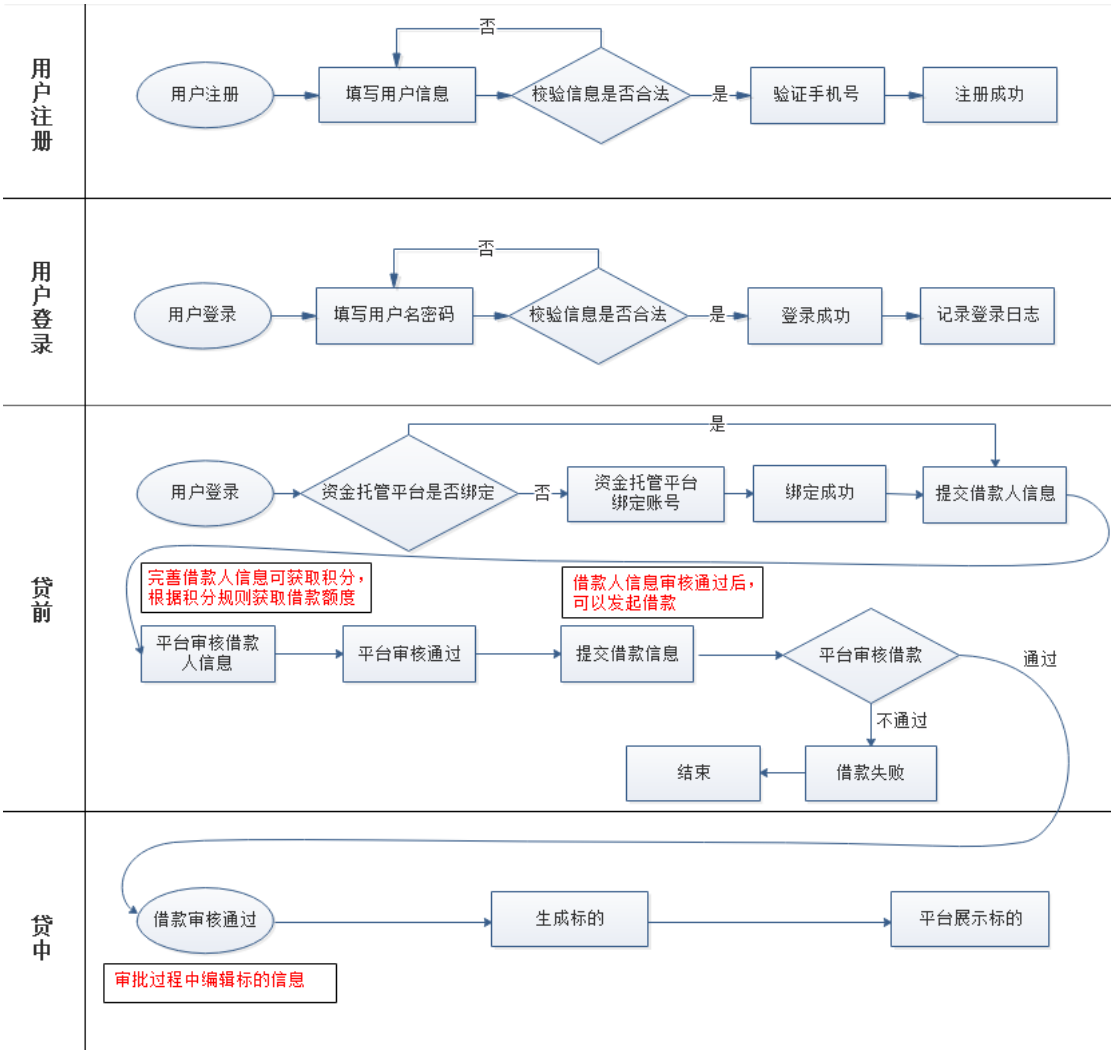


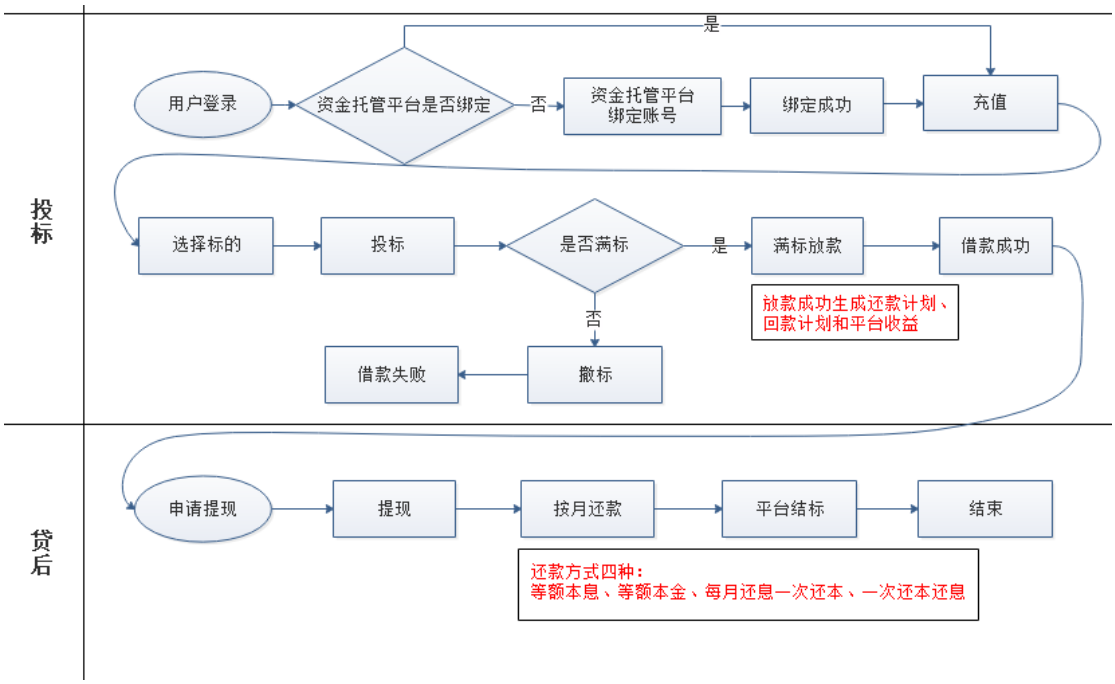
微服务架构图

4.2 业务流程设计

根据前面的系统功能需求分析，结合总体设计思路对系统的业务设计可以分为用户注册、用户登录、贷前、贷中、投标、贷后六个部分。

用户注册先填写用户信息，验证信息是否合法，不合法重新填写，合法验证手机号，然后注册成功。用户登录先填写用户名和密码，验证信息是否合法和正确，错误重新填写，正确就登录成功，并且记录登录日志。贷前先登录确定资金托管平台是否绑定，成功绑定则直接提交借款人信息，否则需要提供相应的银行卡信息绑定账户，绑定成功后提交借款人信息，交由平台审核借款人信息，获取积分兑换借款额度，审核通过后提交借款信息（只有借款人信息审核通过才能发起借款），再交由平台审核借款信息，不通过则借款失败，通过则进入贷中。贷中在借款信息通过下，编辑标的信息生成标的，在平台中展示标的。投标先登录确定资金托管平台是否绑定，未绑定则要绑定资金托管平台账户，绑定成功然后才能充值，进去平台选择标的，投标，如果满标就放款，放款成功系统生成还款计划、回款计划、平台收益，然后借款成功，反之撤标，借款失败。借款成功进入贷后，申请提现，平台提供四种还款方式等额本息、等额本金、每月还息一次还本、一次还本还息，还款结束后平台结标，最后整个业务流程结束。



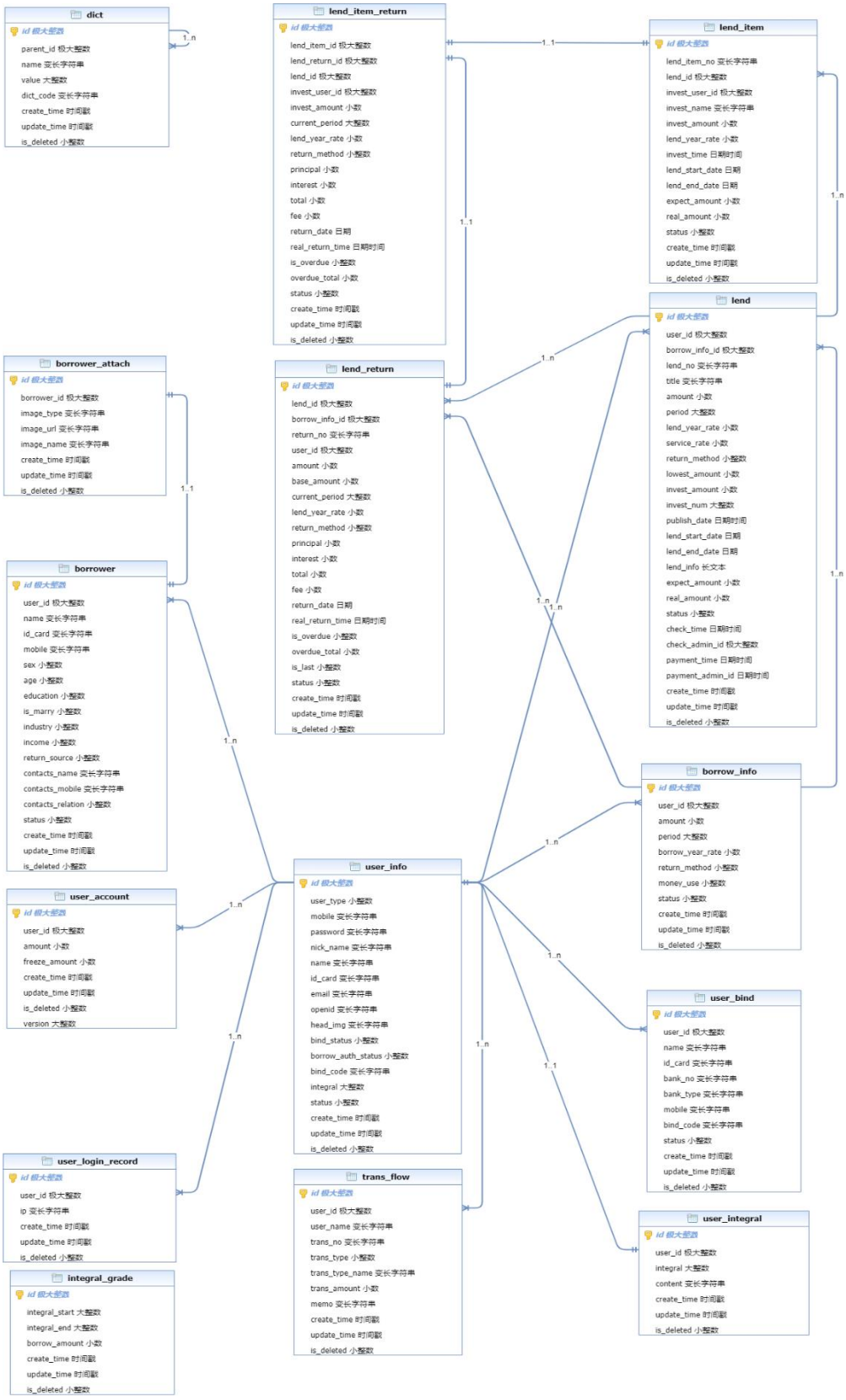


融资宝业务流程图

4.3 数据库设计

众筹平台融资宝系统采用关系型数据库 mysql 进行设计，一共设计了十五张表：dict（数据字典表）、user-info（用户信息表）、integral-grade（积分等级表）、user-login-record（用户登录日志表）、user-integral（用户积分表）、user-bind（用户绑定记录表）、user_account（用户账户表）、trans_flow（流水记录表）、borrower（借款人信息表）、borrower_attach（借款人附加信息表）、borrow_info（借款信息表）、lend（标的表）、lend_item（标的投资项表）、lend_return（还款计划表）、lend_item_return（回款计划表）。表间存在复杂的关系，每张表存储了系统中比较重要的的数据，通过 E-R 图和表设计进行说明。

4.4.1 E-R 图设计



融资宝 E-R 图

4.4.2 数据库表设计

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	借款用户 id	bigint	0		True
amount	借款金额	decimal	10		False
period	借款期限	int	0		False
borrow_year_rate	年化利率	decimal	10		False
return_method	还款方式 1-等额本息 2-等额本金 3-每月还息一次还本 4-一次还本	tinyint	0		False
money_use	资金用途	tinyint	0		False
status	状态（0：未提交，1：审核中， 2：审核通过， -1：审核不通过）	tinyint	0		True
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

borrow_info（借款信息表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	用户 id	bigint	0		True
Name	借款人名	varchar	30		False
id_card	身份证号	varchar	20		True
Mobile	电话	varchar	11		False
Sex	性别（1：男 0：女）	tinyint	0		False
Age	年龄	tinyint	0		False
Education	学历	tinyint	0		False
is_marry	是否结婚（1：是 0：否）	tinyint	1		False
Industry	行业	tinyint	0		False

Income	月收入	tinyint	0		False
return_source	还款来源	tinyint	0		False
contacts_name	预留人名	varchar	30		False
contacts_mobile	预留电话	varchar	11		False
contacts_relation	联系人关系	tinyint	0		False
Status	状态（0：未认证，1：认证中， 2：认证通过， -1：认证失败）	tinyint	0		True
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

Borrower（借款人信息表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
borrower_id	借款人 id	bigint	0		True
image_type	图片类型	varchar	10		False
image_url	图片网址	varchar	200		False
image_name	图片名	varchar	50		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

borrower_attach（借款人附加信息表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
parent_id	上级 id	bigint	0		True
Name	字段名	varchar	100		True
Value	值	int	0		False
dict_code	字段 code	varchar	20		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	删除标记（0:不可用 1:可用）	tinyint	1		True

Dict（数据字典表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
integral_start	积分区间开始	int	0		False
integral_end	积分区间结束	int	0		False
borrow_amount	借款额度	decimal	10		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

integral_grade（积分等级表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	借款用户 id	bigint	0		False
borrow_info_id	借款信息 id	bigint	0		False
lend_no	Lend 号	varchar	30		False
Title	标题	varchar	200		False
Amount	标的金额	decimal	10		False
Period	投资期数	int	0		False
lend_year_rate	年化利率	decimal	10		False
service_rate	平台服务费率	decimal	10		False
return_method	还款方式	tinyint	0		False
lowest_amount	最低投资金额	decimal	10		False
invest_amount	已投金额	decimal	10		False
invest_num	投资人数	int	0		False
publish_date	发布日期	datetime	0		False
lend_start_date	开始日期	date	0		False
lend_end_date	结束日期	date	0		False
lend_info	标备注	text	0		False
expect_amount	平台预期收益	decimal	10		False
real_amount	实际收益	decimal	10		False
Status	状态	tinyint	0		True
check_time	审核时间	datetime	0		False
check_admin_id	审核用户 id	bigint	0		False
payment_time	放款时间	datetime	0		False
payment_admin_id	放款用户 id	datetime	0		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删	tinyint	1		True

	除，0:未删除)				

Lend（标的表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
lend_item_no	标的项号	varchar	50		False
lend_id	标的 id	bigint	0		True
invest_user_id	投资用户 id	bigint	0		False
invest_name	投资人名	varchar	20		False
invest_amount	投资金额	decimal	10		False
lend_year_rate	年化利率	decimal	10		False
invest_time	投资时间	datetime	0		False
lend_start_date	开始日期	date	0		False
lend_end_date	结束日期	date	0		False
expect_amount	预期收益	decimal	10		False
real_amount	实际收益	decimal	10		False
Status	状态（0：默认 1：已支付 2：已还款）	tinyint	0		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

lend_item（标的投资项表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
lend_return_id	标的还款 id	bigint	0		False
lend_item_id	标的项 id	bigint	0		False
lend_id	标的 id	bigint	0		True
invest_user_id	出借用户 id	bigint	0		False
invest_amount	出借金额	decimal	10		False
current_period	当前的期数	int	0		False
lend_year_rate	年化利率	decimal	10		False
return_method	还款方式 1-等额本息 2-等额本金 3-每月还息一次还本 4-一次还本	tinyint	0		False
Principal	本金	decimal	10		False

Interest	利息	decimal	10		False
Total	本息	decimal	10		False
Fee	手续费	decimal	10		False
return_date	还款时指定的还款日期	date	0		False
real_return_time	真实归还时间	datetime	0		False
is_overdue	是否逾期	tinyint	1		False
overdue_total	逾期金额	decimal	10		False
Status	状态（0-未归还 1-已归还）	tinyint	0		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

lend_item_return（回款记录表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
lend_id	标的 id	bigint	0		False
borrow_info_id	借款信息 id	bigint	0		True
return_no	return_no	varchar	30		False
user_id	借款人用户 id	bigint	0		False
Amount	借款金额	decimal	10		False
base_amount	计息本金额	decimal	10		False
current_period	当前的期数	int	0		False
lend_year_rate	年化利率	decimal	10		False
return_method	还款方式 1-等额本息 2-等额本金 3-每月还息一次还本 4-一次还本	tinyint	0		False
Principal	本金	decimal	10		False
Interest	利息	decimal	10		False
Total	本息	decimal	10		False
Fee	手续费	decimal	10		False
return_date	还款时指定的还款日期	date	0		False
real_return_time	真实归还时间	datetime	0		False
is_overdue	是否逾期	tinyint	1		False
overdue_total	逾期金额	decimal	10		False

is_last	是否最后一次还款	tinyint	1		False
Status	状态（0-未归还 1-已归还）	tinyint	0		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

lend_return（还款计划表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为null
Id	编号	bigint	0	主键	True
user_id	用户 id	bigint	0		True
user_name	用户名	varchar	20		False
trans_no	流水号	varchar	30		True
trans_type	交易类型（1：充值 2：提现 3：投标 4：投资回款 ...）	tinyint	0		True
trans_type_name	流水类型	varchar	100		False
trans_amount	交易金额	decimal	10		False
Memo	流水描述	varchar	300		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

trans_flow（流水记录表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	用户 id	bigint	0		True
Amount	帐户可用余额	varchar	20		False
freeze_amount	冻结金额	varchar	30		True
create_time	创建时间	tinyint	0		True
update_time	更新时间	varchar	100		False
is_deleted	逻辑删除(1:已删除，0:未删除)	decimal	10		False
version	版本号	varchar	300		False

user_account（用户账户表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	用户 id	bigint	0		True
Name	用户名	varchar	20		True
id_card	身份证号	varchar	20		True
bank_no	银行卡号	varchar	50		False
bank_type	银行类型	varchar	20		False
Mobile	电话	varchar	11		True
bind_code	绑定码	varchar	50		False
Status	状态	tinyint	0		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

user_bind（用户绑定记录表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_type	1：出借人 2：借款人	tinyint	0		True
Mobile	电话	varchar	11		True
Password	密码	varchar	50		True
nick_name	别名	varchar	20		False
Name	用户名	varchar	20		False
id_card	身份证号	varchar	20		False
Email	邮箱	varchar	100		False
Openid	Openid	varchar	40		False
head_img	头像	varchar	200		False
bind_status	绑定状态（0：未绑定，1：绑定成功 -1：绑定失败）	tinyint	0		True
borrow_auth_status	借款人认证状态（0：未认证 1：认证中 2：认证通过 -1：认证失	tinyint	0		True

	败）				
bind_code	绑定码	varchar	50		False
Integral	用户积分	int	0		True
Status	状态（0：锁定 1：正常）	tinyint	0		True
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

user_info（用户信息表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	用户 id	bigint	0		False
Integral	积分	int	0		False
Content	评论	varchar	100		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

user_integral（用户积分表）

字段名称	解释说明	数据类型	长度	是否为主键	是否为 null
Id	编号	bigint	0	主键	True
user_id	用户 id	bigint	0		False
Ip	IP 地址	varchar	32		False
create_time	创建时间	timestamp	0		True
update_time	更新时间	timestamp	0		True
is_deleted	逻辑删除(1:已删除，0:未删除)	tinyint	1		True

user_login_record（用户登录日志表）

第五章 系统搭建和核心业务功能实现

本章根据总体设计内容，分析系统搭建和核心业务功能实现过程。系统搭建包括平台搭建、模块搭建、和 SpringCloud 基础设施搭建，核心业务功能介绍用户注册、贷前、贷后，以及 Redis 缓存和短信服务使用方法。

5.1 系统搭建

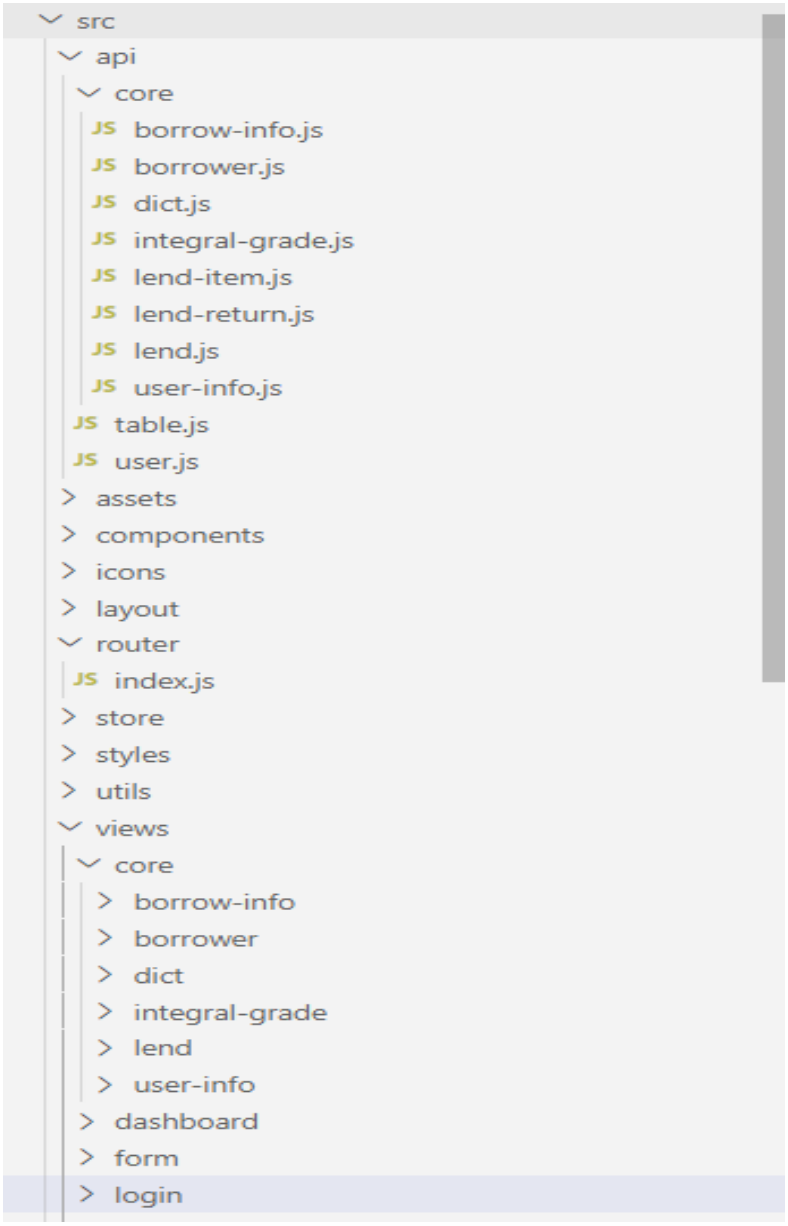
5.1.1 后台管理平台搭建

（1）后端项目选用 vue-element-admin 是基于 element-ui 的一套后台管理系统集成方案。在视图 views 文件夹添加需要的页面效果，然后在 router 文件夹配置路由规则，也就是通过锚点定义不同的 URL，达到访问不同的页面的目的。系统采用较为简单的基础模板（最少精简版）来开发。

基于 vue-element-admin 的融资宝后台管理图

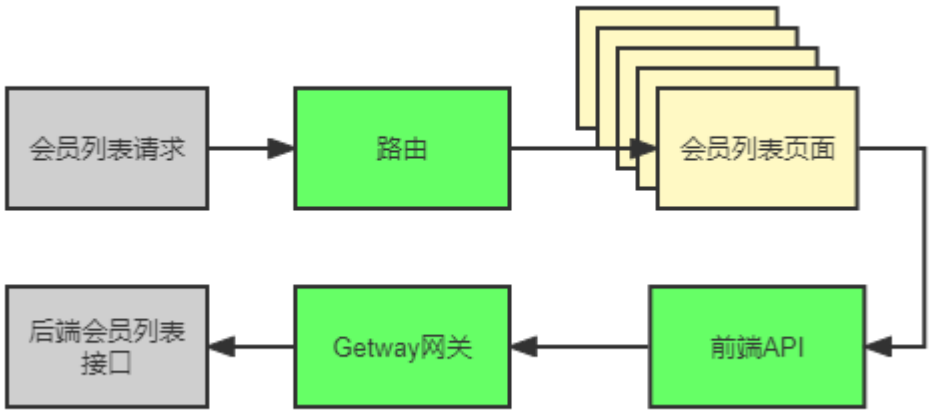
（2）配置路由规则，添加前端 API，添加后台分级的页面，borrow-info（借款列

表）、borrower（借款人列表）、dict（数据字典）、integral-grade（积分等级）、lend（标的管理）、user-info（会员管理）。



融资宝后台管理的前端目录功能图

（3）会员列表的业务功能展示前端请求调用后端服务的流程，点击会员列表的锚点，通过路由规则找到会员列表页面，页面加载中调用前端 API 中 getPageList 方法通过 axios 发送请求，通过 Getway 网关，调用后端会员列表接口。



前端调用后端 API 图

```
{
  path: '/core/user-info',
  component: Layout,
  redirect: '/core/user-info/list',
  name: 'coreUserInfo',
  meta: { title: '会员管理', icon: 'user' },
  alwaysShow: true,
  children: [
    {
      path: 'list',
      name: 'coreUserInfoList',
      component: () => import('@/views/core/user-info/list'),
      meta: { title: '会员列表' },
    },
  ],
},
```

会员列表的路由配置图

```
<script>
import userInfoApi from '@/api/core/user-info'
export default {
  data() {
    return {
      list: null, // 数据列表
      total: 0, // 数据库中的总记录数
      page: 1, // 默认页码
      limit: 10, // 每页记录数
      searchObj: {}, // 查询条件
      loginRecordList: [], //会员登录日志
      dialogTableVisible: false, //对话框是否显示
    }
  },
  created() {
    // 当页面加载时获取数据
    this.fetchData()
  },
  methods: {
    fetchData() {
      userInfoApi
        .getPageList(this.page, this.limit, this.searchObj)
        .then((response) => {
          this.list = response.data.pageModel.records
          this.total = response.data.pageModel.total
        })
    },
```

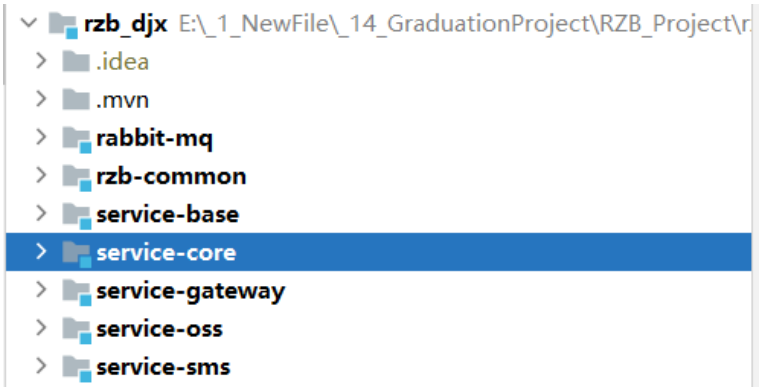
会员页面调用前端 API 图

```
import request from '@/utils/request'
export default {
  getPageList(page, limit, searchObj) {
    return request({
      url: `/admin/core/userInfo/list/${page}/${limit}`,
      method: 'get',
      params: searchObj,
    })
  },
```

会员前端 API 图

5.1.2 后端模块搭建

创建项目：rzb（父工程）、rzb-common、service-base、service-core、rabbit-mq、service-oss、service-sms 模块。系统核心业务 API 放在 service-core 中；工具类、异常类、结果集类放 rzb-common 中；系统配置类、数据传输对象放 service-base 中；发布消息到 MQ 的服务放 rabbit-mq 中；监听 MQ 中的消息，并发送短息的服务放 service-sms 中；系统中图片对象上传阿里云 OSS 中的服务放 service-oss 中。



融资宝后端工程目录图

5.1.3 SpringCloud 基础设施

（1）注册中心和服务发现。下载启动 nacos，在 service-base 中引入“spring-cloud-starter-alibaba-nacos-discovery”的依赖。在每一个微服务 application.yml 配置文件中添加配置。

```
spring:
  application:
    name: # service-core/service-sms/service-oss/..
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848 # nacos 服务地址
```

服务全部启动后查看已注册服务，在 Nacos 中查看服务列表。

Nacos 服务列表图

（2）OpenFeign 服务调用。在 service-base 中添加引入“spring-cloud-starter-openfeign”的依赖，在微服务启动类中添加“@EnableFeignClients”注解启用 Feign 客户端。然后再微服务 controller 层中添加服务调用注解“@RequestMapping、@GetMapping 和 @PostMapping”。

```
//短信服务接口
@Api(tags = "会员接口")
@RestController
@RequestMapping("/api/core/userInfo")
public class UserInfoController {
@ApiOperation("校验手机号是否注册")
@GetMapping("/checkMobile/{mobile}")
public boolean checkMobile(@PathVariable String mobile){
    return userInfoService.checkMobile(mobile);
}
```

在 service-sms 中添加远程调用接口，用到注解“@FeignClient”，发现调用的服务，然后通过“@GetMapping”配置完整接口地址。

```
//短信接口远程调用接口
@FeignClient(value = "service-core")
public interface CoreUserInfoClient {
    @GetMapping("/api/core/userInfo/checkMobile/{mobile}")
    boolean checkMobile(@PathVariable String mobile);
}
```

（3）OpenFeign 添加超时控制，一旦超时就出现远程调用错误，防止远程调用无响应一直等待。因此需要在服务消费者的 application.yml 中添加配置。

```
feign:
  client:
    config:
      default:
        connectTimeout: 10000 #连接超时配置
        readTimeout: 600000 #执行超时配置
```

（4）Sentinel 服务限流配置。在 service-base 中引入“spring-cloud-starter-alibaba-sentinel”的依赖，然后在 application.yml 中配置开启 Feign 支持 Sentinel 的支持。

```
#开启 Feign 对 Sentinel 的支持
#feign:
  sentinel:
    enabled: true
```

创建容错类，若服务调用失败，就降级调用容错类，也就是熔断。

```
//短信发送失败的容错类
@Service
@Slf4j
public class CoreUserInfoClientFallback implements CoreUserInfoClient {
    @Override
    public boolean checkMobile(String mobile) {
        log.error("远程调用失败，服务熔断");
        return false;
    }
}
```

在远程调用接口添加指定容错类。在“@FeignClient”中的 fallback 属性指定类。

```
@FeignClient(value = "service-core", fallback =
CoreUserInfoClientFallback.class)
public interface CoreUserInfoClient {
```

（5）添加网关微服务，同时也注册在 Nacos 注册中心里。创建 service-getway 模块，添加 Getway 网关依赖和 Nacos 服务发现依赖。

```
<dependencies>
    <dependency>                <!-- 网关 -->
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>                <!--服务注册-->
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>
</dependencies>
```

配置微服务端口、服务名、Nacos 服务地址、让 gateway 可以发现 nacos 中的微服务，并自动生成转发路由。

```
server:
  port: 80 # 服务端口
spring:
  profiles:
    active: dev # 环境设置
  application:
    name: service-gateway # 服务名
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848 # nacos 服务地址
```

```
    gateway:
      discovery:
        locator:
          enabled: true
# gateway 可以发现 nacos 中的微服务，并自动生成转发路由
```

在网关主启动类添加注解“@EnableDiscoveryClient”，让注册中心能够发现，扫描到该服务。然后配置网关路由规则，允许正确的路由地址访问微服务。

```
#spring:
# cloud:
#   gateway:
      routes:
      - id: service-core
        uri: lb://service-core
        predicates:
        - Path=/*/core/**
      - id: service-sms
        uri: lb://service-sms
        predicates:
        - Path=/*/sms/**
      - id: service-oss
        uri: lb://service-oss
        predicates:
        - Path=/*/oss/**
#放行 core、sms、oss
```

（6）添加跨域配置，解决同源策略限制。在 getway 微服务中添加配置类，运行携带 Cookie、接受任意域名、允许携带头、允许所有访问方式。

```
@Configuration
public class CorsConfig {
    @Bean
    public CorsWebFilter corsFilter() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true); //是否允许携带 cookie
        config.addAllowedOrigin("*"); //可接受的域，是一个具体域名或者*（代表任意域名）
        config.addAllowedHeader("*"); //允许携带的头
        config.addAllowedMethod("*"); //允许访问的方式
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return new CorsWebFilter(source);
    }
}
```

5.2 核心业务功能实现

5.2.1 Redis 缓存数据字典

在 Spring Boot 项目中中，默认集成 Spring Data Redis，Spring Data Redis 针对 Redis 提供了非常方便的操作模版 RedisTemplate，并且可以进行连接池自动管理。在 service-base 中添加 redis 依赖。然后再 service-core 中添加 redis 配置，包括主机、端口、密码、超时时间、连接池配置。

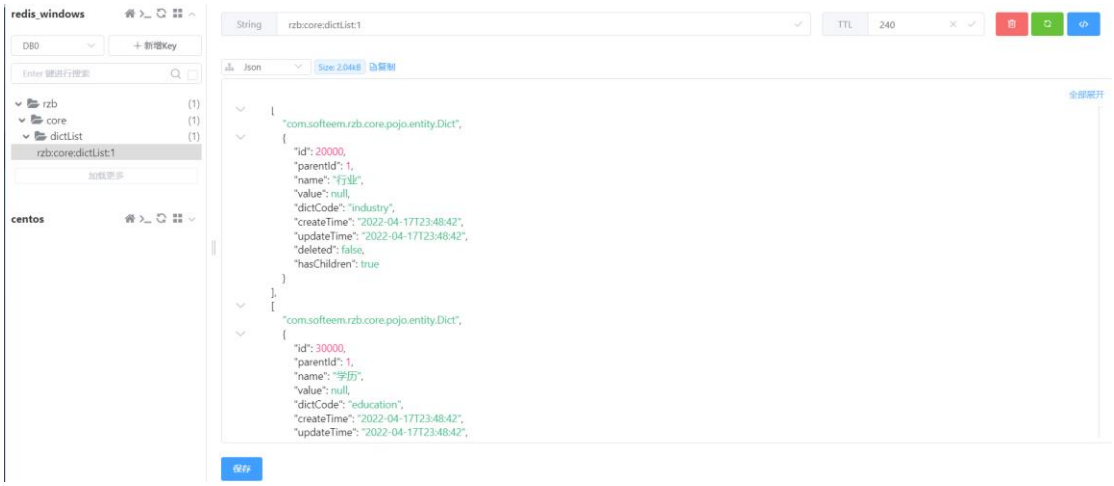
```
<!-- spring boot redis 缓存引入 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- 缓存连接池-->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
</dependency>
<!-- redis 存储 json 序列化 -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
</dependency>

#redis 配置文件
#spring:
  redis:
    host: 127.0.0.1
    port: 6379
    database: 0
    password: 123456 #默认为空
    timeout: 3000ms #最大等待时间，超时则抛出异常，否则请求一直等待
    lettuce:
      pool:
        max-active: 20  #最大连接数，负值表示没有限制，默认 8
        max-wait: -1    #最大阻塞等待时间，负值表示没限制，默认-1
        max-idle: 8     #最大空闲连接，默认 8
        min-idle: 0     #最小空闲连接，默认 0
```

启动 Redis 服务，当查询数据字典的时候，优先查询 Redis 中是否有 dict 列表，如果

有就取出并返回 dictlist，如果没有就从数据库中取新的 dictlist，然后存入 Redis 中，最后返回 dictlist。

```
@Resource
private RedisTemplate redisTemplate;
@Override
public List<Dict> listByParentId(Long parentId) {
    //先查询 redis 中是否存在数据列表
    List<Dict> dictList = null;
    try {
        dictList =
(List<Dict>)redisTemplate.opsForValue().get("rzb:core:dictList:" + parentId);
        if(dictList != null){
            return dictList;
        }
    } catch (Exception e) {
        log.error("redis 服务器异常：" + ExceptionUtils.getStackTrace(e));
    }
    dictList = baseMapper.selectList(new QueryWrapper<Dict>().eq("parent_id",
parentId));
    dictList.forEach(dict -> {    //如果有子节点，则是非叶子节点
        boolean hasChildren = this.hasChildren(dict.getId());
        dict.setHasChildren(hasChildren);
    });    //将数据存入 redis
    try {
        redisTemplate.opsForValue().set("rzb:core:dictList:" + parentId,
dictList, 5, TimeUnit.MINUTES);
    } catch (Exception e) {
        log.error("redis 服务器异常：" + ExceptionUtils.getStackTrace(e));
    }
    return dictList;
}
```

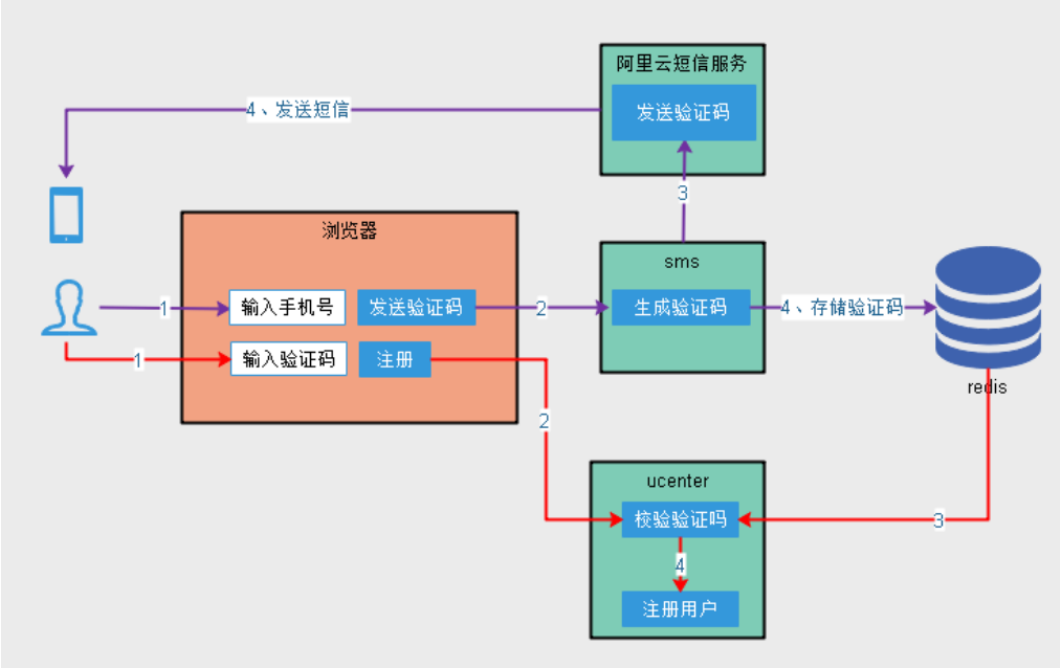


Redis 中数据字典数据图

名称	编码	值
行业	industry	
学历	education	
收入	income	
收入来源	returnSource	
关系	relation	
还款方式	returnMethod	
等额本息		1
等额本金		2
每月还息一次还本		3
一次还本还息		4
资金用途	moneyUse	
借款状态	borrowStatus	
学校性质	SchoolStatus	

系统数据字典查询结果图

5.2.2 短信微服务和用户注册



用户注册和发送验证码流程图

（1）在 service-sms 模块添加短信服务需要的依赖、配置 application.yml。

```
<!—购买的短信服务提供的依赖 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.15</version>
</dependency>
```

```
#配置文件
server:
  port: 8120 # 服务端口
spring:
  profiles:
    active: dev # 环境设置
  application:
    name: service-sms # 服务名
#短信配置
rzb:
  sms:
    host: https://gyytz.market.alicloudapi.com
    path: /sms/smsSend
    method: POST
    appcode: dee6d7be41154459a7b7f0da48cf39df  #写自己的 code
    sms-sign-id: 2e65b1bb3d054466b82f0c9d125465e2
    template-code: a09602b817fd47e59e7c6e603d3f088d  #模板 ID
```

（2）前端远程调用发送短信的接口。在短信服务的 controller 层，校验电话号码正确性，远程调用 core 服务检查电话是否被注册，系统生成验证码，调用 service 层发送短信功能，将验证码保存 redis 并设置五分钟有效时间、service 层，按要求短信服务厂商要求组装 host, path, method, headers, querys, bodys 参数利用 doPost 方式发送。

```
//远程调用发送短信的接口
this.$axios
     .$get('/api/sms/send/' + this.userInfo.mobile)
     .then((response) => {
        this.$message.success(response.message)
})



@RestController
@RequestMapping("/api/sms")
@Api(tags = "短信管理")
public class ApiSmsController {
   @Resource private SmsService smsService;
   @Resource private RedisTemplate redisTemplate;
   @ApiOperation("获取验证码")
   @GetMapping("/send/{mobile}")
   public R send(@ApiParam(value = "手机号", required = true) @PathVariable String
mobile) {
//检查 mobile 是否符合电话规则
……..
String code = RandomUtils.getFourBitRandom();    // 组装短信模板参数
```

```
Map<String, Object> param = new HashMap<>();
param.put("code", code);                                    // 发送短信
smsService.send(mobile, SmsProperties.TEMPLATE_CODE, param);
System.out.println("电话：" + mobile + "----验证码：" + code);// 将验证码存入 redis
redisTemplate.opsForValue().set("rzb:sms:code:" + mobile, code, 5, TimeUnit.MINUTES);
//短信验证码五分钟有效
```

（3）然后点击发送验证码，填写正确后点击注册。

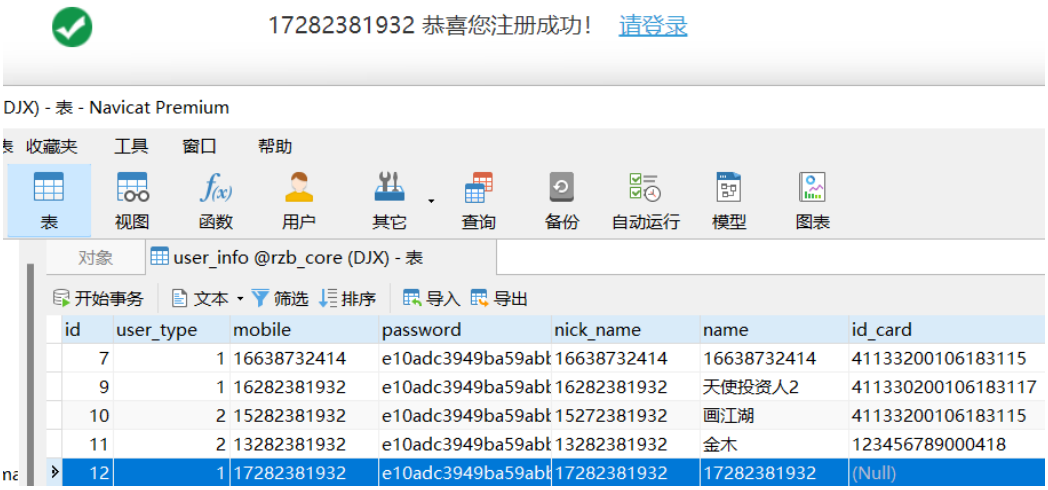
系统会员注册图

短信通知图

（4）点击注册远程调用会员注册接口。先验证电话是否符合规则，然后从 redis 中取

出验证码，和系统中生成的验证码进行比对，然后调用 service 层注册业务功能，返回成功。

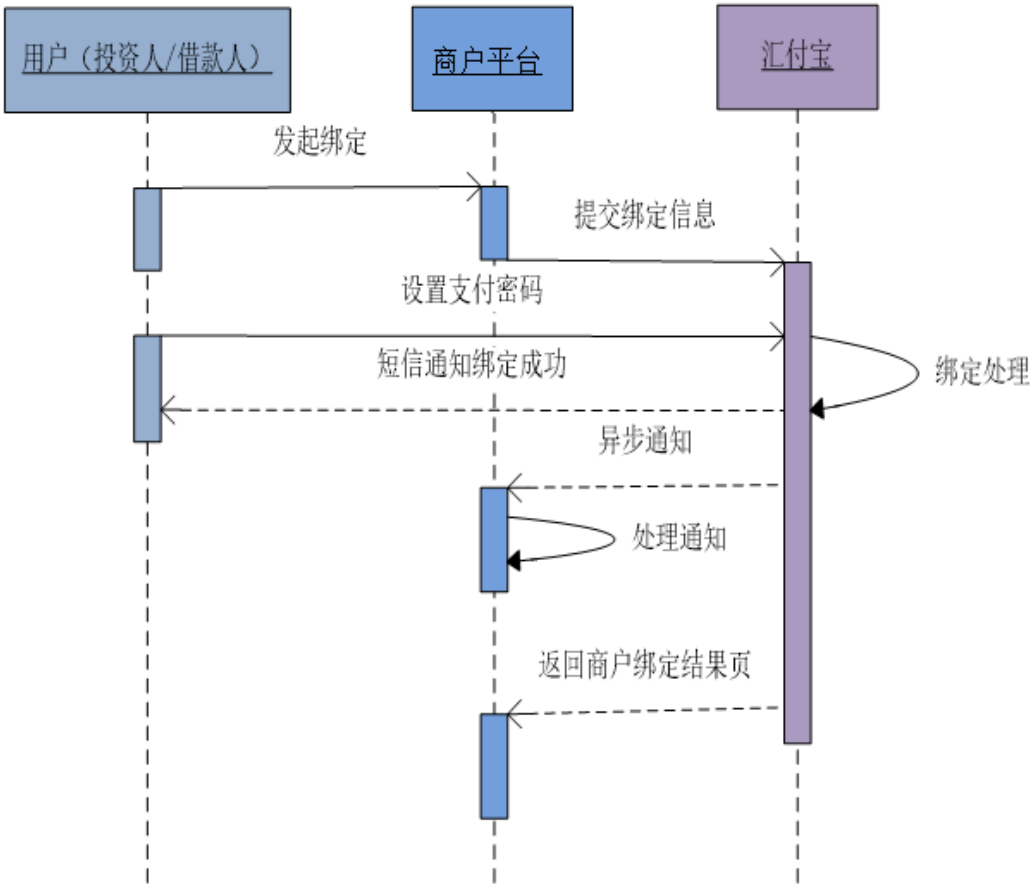
```
@Api(tags = "会员接口")
@RestController
@RequestMapping("/api/core/userInfo")
public class UserInfoController {
  @Resource private UserInfoService userInfoService;
  @Resource private RedisTemplate redisTemplate;
  @ApiOperation("会员注册")
  @PostMapping("/register")
  public R register(@RequestBody RegisterVO registerVO) {
    String mobile = registerVO.getMobile();
    String password = registerVO.getPassword();
    String code = registerVO.getCode();
//校验验证码、密码、电话不能为空，是否符合规则
.....
    String codeGen = (String) redisTemplate.opsForValue().get("rzb:sms:code:" +
mobile);    // CODE_ERROR(-206, "验证码不正确"),
    Assert.equals(code, codeGen, ResponseEnum.CODE_ERROR);
    // 注册
    userInfoService.register(registerVO);
    return R.ok().message("注册成功");
  }
```



会员注册成功图

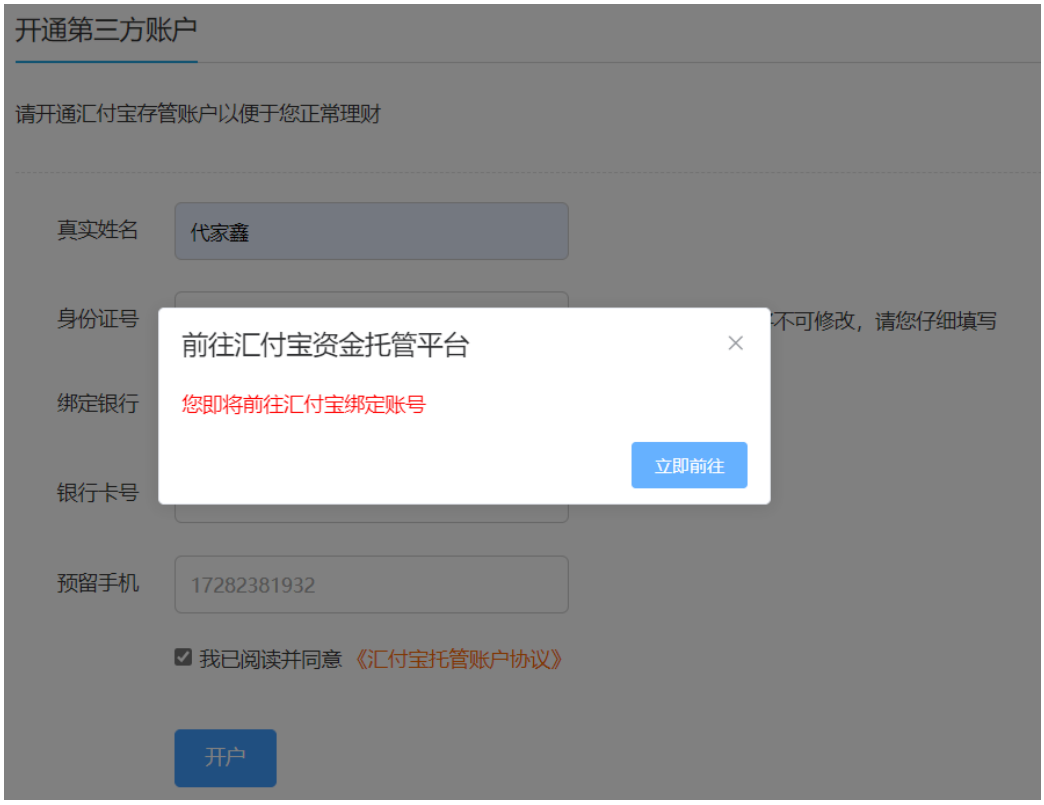
5.2.3 贷后

（1）使用注册的投资人登录成功。新注册的用户未绑定第三方支付账号，只有绑定后才能充值和投资。在绑定页面填写姓名、身份证号、银行、卡号、预留电话后发起绑定。后端绑定服务以 POST 方式提交数据并弹出/跳转汇付宝绑定账户页，用户完成绑定后同步返回到商户页，同时异步通知商户。



账户绑定图

（2）会员绑定接口。首先从请求中得到 token，通过 token 获取到 Userid（token 通过 JwtUtils 封装了会员的基本信息），然后调用绑定业务。绑定业务就是先根据身份证和用户 ID 判断是否绑定，然后查询是否有该会员的绑定记录，没有就插入绑定记录，设置用户绑定状态，反之更新绑定记录的信息。然后封装第三方绑定接口需要的参数 Map 类型。



会员绑定图

```
@Api(tags = "会员账号绑定")
@RestController
@RequestMapping("/api/core/userBind")
public class UserBindController {
   @Resource private UserBindService userBindService;
   @ApiOperation("账户绑定提交数据")
   @PostMapping("/auth/bind")
   public R bind(@RequestBody UserBindVO userBindVO, HttpServletRequest request) {
      String token = request.getHeader("token");
      Long userId = JwtUtils.getUserId(token);
      String formStr = userBindService.commitBindUser(userBindVO, userId);
      return R.ok().data("formStr", formStr);
   }

//formStr 封装的页面部分代码
String formStr = "<!DOCTYPE html>\n" +
            "<html lang=\"en\" xmlns:th=\"http://www.thymeleaf.org\">\n" +
            "<head>\n" +
            "</head>\n" +
            "<body>\n" +
            "<form name=\"form\" action=\""+url+"\" method=\"post\">\n"+
            inputStr +
            "</form>\n" +
```

```
            "<script>\n" +
            "\tdocument.form.submit();\n" +
            "</script>\n" +
            "</body>\n" +
            "</html>";
return formStr;
```

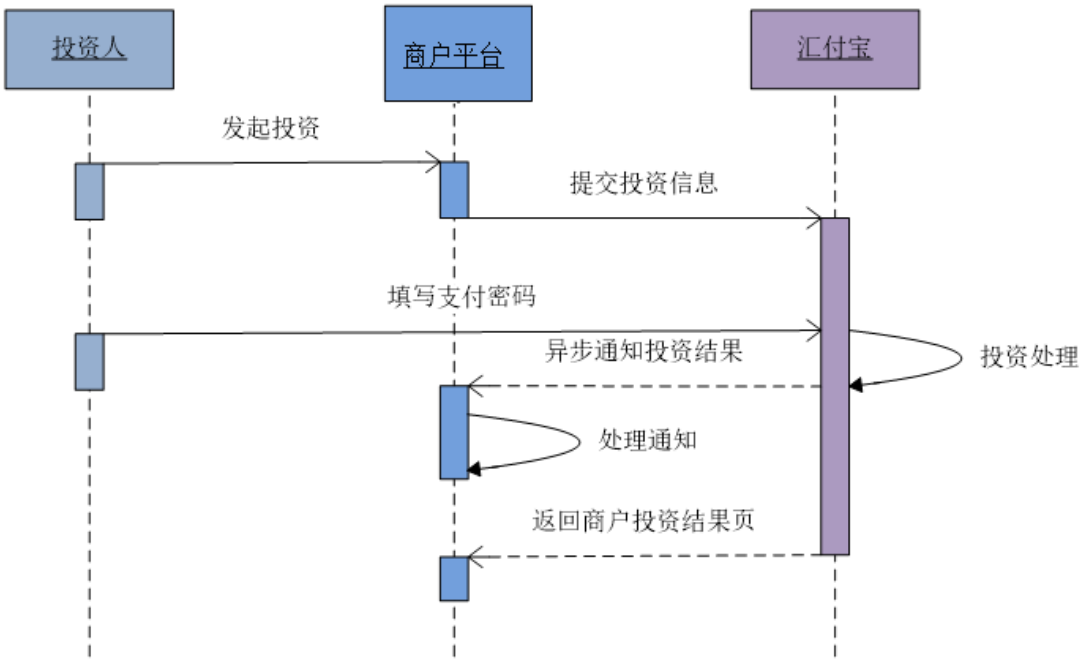
填写的的参数就封装在 formStr 中，这是一个汇付包绑定的 from 表单，以 String 类型形式返回前端渲染成页面。点击确定通过表单将参数（包含异步回调的地址）提交给汇付包绑定接口。绑定成功后，汇付包异步发送成功通知到回调地址。

（3）异步通知的参数有绑定 code 码、绑定结果、会员 ID、时间，以 Map 形式发送。系统接收参数，更新会员绑定 Code 码、绑定记录信息。然后会员就可以充值投资，选择合适的投资项目，选择要投资的金额，平台会自动根据该项目的利率计算收益。

项目投资收益图

点击立即投资，系统后台以 POST 方式提交数据并弹出/跳转到汇付宝投资页，投资人成功提交投资信息后，同步返回到商户页，同时汇付宝将投资处理结果异步通知到商户的 notify_url，更新对应信息。而后端标的投资接口，先验证 token 得到 UserId 和用户名，封装到投资对象中，仍然把投资信息封装到一个 fromStr 中返回。在前端渲染出汇付包投资

页面，投资成功后异步通知融资宝。通知的参数封装到 Map 中，包括投资描述、订单号、项目号、表类型、投资人绑定号、投资金额、手续费、通知时间，签名。通知的接口，需要更新会员账户资金，将投资金额改为冻结，修改投资记录的状态，更新标的信息，新增流水信息。



融资宝投资顺序图

5.2.4 贷前

（1）注册借款人登录。仍然是需要像借款人那样先绑定第三方支付账号，绑定成功后可以去借款。先绑定借款人信息，包括基础信息和图片信息，图片通过 Service-oss 服务异步上传，得到图片 url 即可，然后后端服务保存借款人信息。

```
@RestController
@Api(tags = "借款人")
@RequestMapping("/api/core/borrower")
public class BorrowerController {
    @Resource
    private BorrowerService borrowerService;
    @ApiOperation("保存借款人信息")
    @PostMapping("/auth/save")
    public R save(@RequestBody BorrowerVO borrowerVO, HttpServletRequest request) {
        String token = request.getHeader("token");
        Long userId = JwtUtils.getUserId(token);
        borrowerService.saveBorrowerVOByUserId(borrowerVO, userId);
        return R.ok().message("信息提交成功");
    }
```

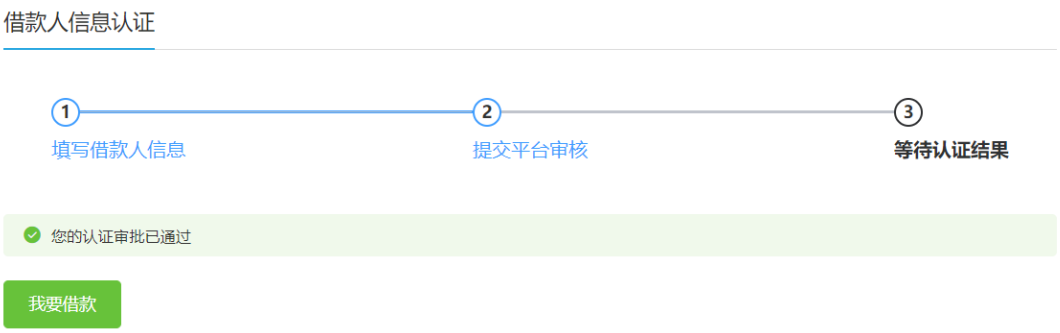
而借款人页面的状态显示根据状态来显示进度。对应填写借款人信息-提交审核-审核

结果三个阶段。

借款人信息认证状态图

（2）管理员审核借款人信息。管理平台查询出借款人信息并展示列表，点击审核，管理员根据信息内容判断是否准确得到相应的积分（用于获得借款额度）。审核通过，更新借款人认证状态变成认证通过，借款人页面会查询借款人认证状态重新更新进度。

管理审核积分图

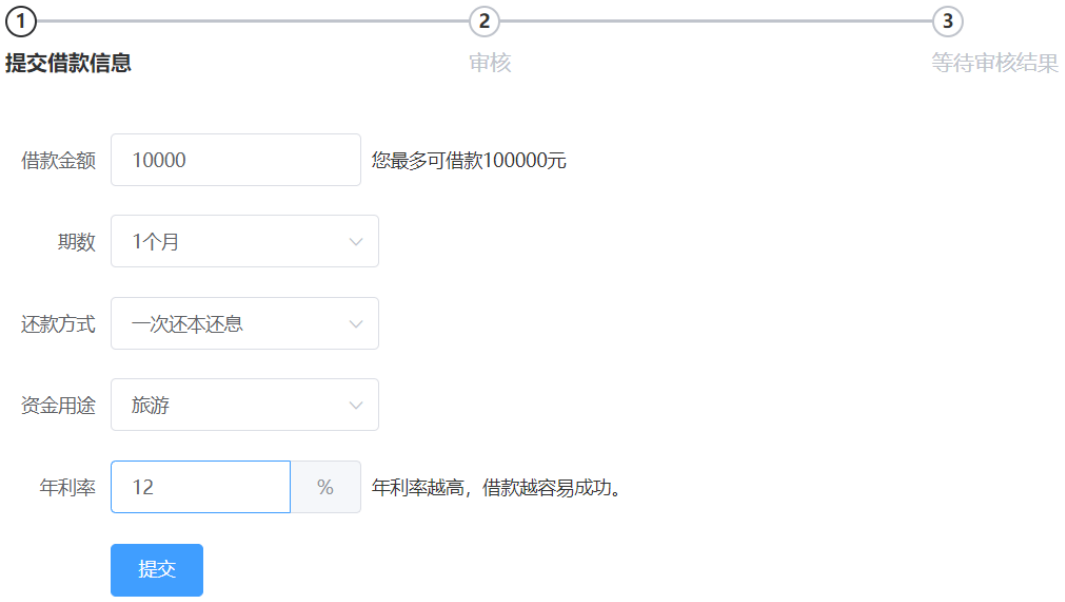


借款人信息认证成功图

（3）然后发起借款项目的申请，包括借款金额、目的、还款方式、年利率、还款期数。提交借款信息到后端，然后判断借款金额和借款额度是否匹配，满足就插入借款信息，同时更新状态为审核中。

```
@Api(tags = "借款信息")
@RestController
@RequestMapping("/api/core/borrowInfo")
```

```
public class BorrowInfoController {
    @Resource
    private BorrowInfoService borrowInfoService;
    @ApiOperation("提交借款申请")
    @PostMapping("/auth/save")
    public R save(@RequestBody BorrowInfo borrowInfo, HttpServletRequest request) {
        String token = request.getHeader("token");
        Long userId = JwtUtils.getUserId(token);
        borrowInfoService.saveBorrowInfo(borrowInfo, userId);
        return R.ok().message("提交成功");
    }
```



借款信息内容图

（4）管理员审核通过后，生成标的，修改借款信息状态。然后就可以在平台上看到我们标的信息。

管理员审核图


借款标题
借款金额
年利率
借款期限
还款方式
借款进度
操作
带女朋友旅游
99999元
12%
24个月
一次还本还息
98.80098800988O1%
募资中
治病
10000元
10%
24个月
等额本金
100%
还款中
旅游
10000元
12%
1个月
一次还本还息
0%
募资中

平台展示借款列表图

第六章 系统功能验证与性能测试

6.1 Swagger 系统功能验证

Service-base 中添加 swagger 依赖，这里采用私人分析的使用；添加 swagger 配置类，接口分为 adminApi（后台操作）、webApi（前台操作），然后添加 Api 基础信息，包括分组名，页面信息，路径。然后只需要在 controller 层添加注解“@Api(tags = "父接口") @ApiOperation("子接口")”。

```
public class Swagger2Config {
    //后台的操作
    @Bean
    public Docket adminApiConfig(){
        return new Docket(DocumentationType.SWAGGER_2)
                .groupName("adminApi")      //分组名
                .apiInfo(adminApiInfo())    //页面信息
                .select()
                //只显示admin路径下的页面,
                .paths(Predicates.and(PathSelectors.regex( pathRegex: "/admin/.*")))
                .build();
    }

    private ApiInfo adminApiInfo(){
        return new ApiInfoBuilder()
                .title("融资宝后台管理系统-API文档")
                .description("本文档描述了融资宝后台管理系统接口")
                .version("1.0")
                .contact(new Contact( name: "djx",  url: "http://www.softeem.com",  email: "1636814673@qq.com"
                .build();
    }
```

Swagger 配置类的后台操作 API 图

Swagger 生成 API 文档图

后端项目中所以的接口就通过 swagger 生成了可阅读的 API 文档，并且提供在线测试。选择会员接口调试会员登录功能。先添加全局参数 token（如果没有 token 无法通过

会员验证）和登录需要参数 loginVo。

swagger 会员登陆测试图

登录成功的响应内容图

6.2 JMeter 系统性能测试

对系统会员登录和资金记录进行性能测试，使用 JMeter 测试系统的性能。使用 500 歌线程，循环 5 次，启动时间设置为 5s。

性能测试计划图

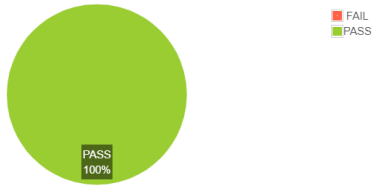
生成测试报告，仪表盘可以看到请求成功率是百分之百，在两个请求共计 5 千次请求量，系统在网络正常情况下仍然没有出现请求失败，请求成功率 100%，其中平均响应时间 2.6s，百分之九十五响应时间在 5.7s，最大和最小响应时间为 14.3s 和 3ms。

所以本次测试结果说明系统在 500 的线程数的并发环境下可以正常操作运行，基本满足预期需要，同时错误率 0。但是 Apdex 较低，也说明了服务器响应速度较慢，需要进一步优化。

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.174	**500 ms**	**1 sec 500 ms**	**Total**
0.171	500 ms	1 sec 500 ms	会员登录
0.176	500 ms	1 sec 500 ms	会员资金查询

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**5000**	**0**	**0.00%**	**2635.25**	**3**	**14367**	**2406.50**	**4823.00**	**5728.65**	**7898.88**	**150.25**	**239.24**	**51.13**
会员登录	2500	0	0.00%	2603.24	61	14367	2336.50	4791.60	5722.00	7898.88	75.13	49.96	19.88
会员资金查询	2500	0	0.00%	2667.27	3	12850	2599.00	4839.80	5748.60	7947.94	75.64	190.59	31.47

压力测试结果图

通过测试结果提供的可视化数据分析，大致清楚了系统性能的瓶颈，为后续的优化工作提供方向，并且得出，在高并发环境下，系统可以保证高可用性。

第七章 总结与展望

7.1 论文总结

论文到这里核心点就介绍完了，主要介绍了众筹的背景、融资宝平台需要的技术栈、以及三大主体借款人，投资人，管理员的业务需求分析以及实现。使用微服务架构创建平台，也是为了摆脱单体架构的缺点，增强系统的可扩展性和可维护性，并且从系统的实现过程中感受到微服务带给系统的便利和提升。虽然这个微服务项目在基础设施搭建上不是很完整，但是总体的实现和设计思路是遵循微服务架构原则。通过参考微服务项目设计过程的相关资料，选择了 Nacos 作为服务注册和发现，getway 作为网关，sentinel 作为服务限流，MQ 作为消息中间键，Redis 作为数据缓存，Swagger 作为后端 API 文档生成。最终确定了合适的系统实现架构和技术栈。通过 UML 图分析系统用户需求和功能需求，进行系统的总体设计，通过微服务架构设计、业务流程设计、数据库设计，完善了系统实现的蓝图，确保了整个系统大体的架构是正确可行的。然后就是系统实现过程，从前端平台搭建到后端项目搭建，并搭建 SpringCloud 基础设施，确认组件的正常使用。然后实现核心功能，并通过 swagger 进行在线测试，确认每一个功能可以正常使用，保证系统功能的完整性。最后通过 Jmeter 测试系统部分功能，确保系统的高性能。

通过本次项目实现和论文的编写，明确了对一个微服务项目整个设计思路和技术选择，合理的使用 SpringCloud 的组件可以更好的提升项目的可用性、可维护性、高性能。

7.2 项目展望

如今项目的架构比较流行微服务架构，但是本人对微服务的理解不是很深刻，缺乏开发经验。开发过程也遇到很多问题，在同学帮助和自己钻研下，最终完整的实现一个微服务架构的项目。但是本项目还要很多要加强和改进的地方。

（1） 项目没有真正部署在一个云服务器上，只是自己简单的实验和测试，真实上线肯定会有一些新的关于微服务项目上线的问题。

（2） 系统的登录方式比较单一，只能通过电话和密码登录，可以完善会员信息绑定微信或 QQ 登录服务，让用户可以灵活的登录平台。

（3） 系统的账户绑定途径比较单一，只能绑定银行卡，可以拓展支付宝沙箱功能，提供新的绑定、支付、充值手段。

参考文献

[1] 胡吉祥,吴颖萌.众筹融资的发展及监管[J].证券市场导报,2013(12):60-65.
[2] 李贞昊.微服务架构的发展与影响分析[J].信息系统工程,2017(01):154-155.
[3] 周永圣,侯峰裕,孙雯,杨磊,张小贝.基于 SpringCloud 微服务架构的进销存管理系统的设计与实现[J].工业控制计算机,2018,31(11):129-130+133.
[4] 马雄. 基于微服务架构的系统设计与开发[D].南京邮电大学,2017.
[5] 刘斌.基于 SpringCloud 的电信综合服务保障系统微服务改造之路[J].中小企业管理与科技(下旬刊),2019(11):90-91.
[6] 冯志勇,徐砚伟,薛霄,陈世展.微服务技术发展的现状与展望[J].计算机研究与发展,2020,57(05):1103-1122.
[7] 杨文豪. 基于微服务的网上商城系统的设计与实现[D].北京邮电大学,2021.DOI:10.26969/d.cnki.gbydu.2021.000275.
[8] 付博. 基于 SpringCloud 的绿植养护软件的设计与实现[D].北京邮电大学,2021.DOI:10.26969/d.cnki.gbydu.2021.001741.
[9] 张斌,任富彬,沈炜.基于 SpringCloud 的食品安全溯源系统的设计与实现[J].软件工程,2019,22(08):27-30.DOI:10.19644/j.cnki.issn2096-1472.2019.08.009.
[10] 张林.基于微服务架构的商对客模式电商网站的设计与实现[J].软件工程,2021,24(09):55-57.DOI:10.19644/j.cnki.issn2096-1472.2021.09.013.
[11] 俞林,康灿华,王龙.互联网金融监管博弈研究:以 P2P 网贷模式为例[J].南开经济研究,2015(05):126-139.DOI:10.14116/j.nkes.2015.05.008.
[12] 秦颐. 小额贷款公司融资方式研究[D].中国农业大学,2014.

致谢