集美大学

JIMEI UNIVERSITY

学 士 学 位 论 文

BACHELOR DISSERTATION

论文题目 基于 Spring Cloud 微服务架构的程序员社交平台

学生姓名 姬龙龙

学　　号 201521122082

学　　院 计算机工程学院

专　　业 软件工程

指导教师 王俊玲

2019 年　5 月　30 日

集美大学本科学位论文诚信书

毕业论文（设计）题　目	基于 Spring Cloud 微服务架构的程序员社交平台				
学生姓名	姬龙龙	专　业	软件工程	学　号	201521122082
指导老师	王俊玲		职　称	讲师	
所在学院	计算机工程学院				

诚信书

本人慎重承诺和声明：

我承诺在毕业设计（论文）活动中遵守学校有关规定，恪守学术规范， 在本人的毕业设计（论文）中未剽窃、抄袭他人的学术观点、思想和成果， 未篡改研究数据，如有违规行为发生，我愿承担一切责任，接受学校的处理。

学生（签名）：姬龙龙

2019 年 5 月 30 日

备注：本表存入学生毕业设计（论文）资料袋。

基于 Spring Cloud 微服务架构的程序员社交平台

[摘要] 在信息繁杂的互联网时代中，程序员们在遇到工作中的难题时需要频繁地借助搜索引擎查询，但由于大量无效信息的干扰，时间成本消耗极大。本系统是一个为程序员服务的技术社区，致力于建立关于编程中遇到的各类问题的详细解答宝库。由于传统单体应用架构在系统开发中存在效率低，扩展困难等问题，所以本系统选择使用微服务架构作为技术研究方向。Spring Cloud 微服务架构是一套适用于现代分布式系统环境的代码开发体系，相比于传统的单体应用架构，它是由一系列独立运行的微服务(部署在 Docker 容器中)共同构建起来的，易于开发维护，适用于需求快速迭代的场景。本系统使用前后端分离模式进行开发，前端使用 Vue 框架，后端使用以 Spring Boot 为脚手架的 Spring Cloud 技术栈。

[关键词] 微服务； Spring Cloud； ELK； Docker

Programmer Social Platform Based on Spring Cloud Microservice Architecture

[Abstract] In the Internet age with multifarious information, this group of programmers often need to frequently make use of search engine to query when they encounter difficulties in work, but due to the interference of a large amount of invalid information, the cost of time is enormous. This system is a technical community for programmers, it working to build a library of detailed answers to every question about programming. In addition, the traditional single application architecture is inefficient and difficult to expand in system development, this system chooses microservice architecture as the research direction of technology. The Spring Cloud Microservice Architecture is a set of code development architecture that is suitable for modern distributed system environment. It is builted from a series of independently running Microservices (deployed in Docker containers) compared to the traditional single architecture. Easy to develop and maintain, it is very suitable for business needs rapid iteration scenario. The system uses the separation mode of the front-end and backend for developing. The front-end uses the Vue framework, and the backend uses the Spring Cloud technology stack with Spring Boot as the scaffolding.

[Keywords] Microservices; Spring Cloud; ELK; Docker

目录

第1章 引言

1.1 系统开发背景及意义

在当今互联网产品百花齐放的时代，程序员们获取信息的来源变得越来越广泛了，各类技术社区(CSDN、掘金)、贴吧、百度知道等平台给他们带来了许多的技术帮助，但往往在有效信息中也掺杂着大量垃圾信息(广告、水贴)，导致程序员在遇到工作中的难题时，不能及时找到解决方案而降低了工作效率。

在调研过程中，我发现大量优质教程资源与问题的解决方案集中在国外技术社区，最知名的网站是StackOverflow，它被设计成为用户提出问题、解决问题的平台，由世界各地开发者贡献出的多个领域，分门别类的提问与回答组成，在其中程序员们可以进行激烈的问题探讨，并提出多种方式的解决方案，它逐渐的成为程序员们寻找问题解决方案的首选渠道，帮助了成千上万的程序员们避免加班，但由于国内网络的问题，对其的访问及其困难，并且网站的语言文字是全英文的，国内程序员使用起来十分不方便。

本次研究的课题正是基于这样一个背景，旨在建立一个专业、功能完善的程序员技术社区，它能够让程序员在工作中遇到技术难题时能够快速得到解决方案，而无需去互联网搜索引擎上盲目搜寻，节省了大量的时间，另外一些扩展功能模块，如优质文章阅读、在线教育学习、吐槽交友也让用户在使用该系统时能得到更全面的体验。

1.2 本课题研究内容与目标

本系统是现代主流的前后端分离项目，后端的技术架构为本次课题主要研究方向，采取了Spring Cloud微服务架构[1]（一种适合业务需求快速迭代、产品快速部署的技术架构），在传统Web应用技术架构体系中，一个项目往往采取单体应用架构实现，这种架构利用水平切分的思想，将应用拆分为用户表现层、业务逻辑层与数据模型层，也就是常见的MVC架构，但随着互联网应用的高速发展和网民数量的不断增加，系统业务变得非常复杂且迭代周期很短，网站的用户访问量也变得十分巨大，采用单体架构会使得产品开发效率低、编译时间长、回归测试周期长，最主要的是由于服务器硬件资源的限

制，往往达到一定的吞吐量后，服务器系统会变得卡顿，甚至宕机，直接导致公司的经济损失。

微服务架构是把一个大型的系统按照功能进行服务拆分，让每个功能模块独立部署和管理，例如短信发送，文件上传就可以单独拆分出来部署，并且利用容器化技术 Docker 与 Kubernetes(生产级别的容器编排系统)[2]进行应用的全自动部署与管理，大大缩减了应用上线周期与运维成本[3]。本系统采用 Spring Cloud 来实现微服务架构是由于目前各大互联网公司已经在生产环境使用过该技术体系，稳定性和安全性都得到了保证，通过该技术架构的实施，能让本系统在线上稳定的运行，后期扩展业务也十分方便快捷。

第 2 章 相关技术介绍

2.1 相关理论

本系统采用现代化 Web 开发中主流的理论，争取用最先进的开发思想来设计本项目，这些优秀理论都是来自于互联网的高速发展中出现的各种问题而总结出来最合适的解决方案。下面对主要的几个理论做详细介绍。

2.1.1 前后端分离的模块化设计

传统的软件开发流程为：产品经理根据客户需求制作出原型设计，后端根据原型设计进行需求分析后进行服务接口开发，UI 设计人员根据原型图设计出具体的产品界面，前端工程师根据 UI 设计人员制作的设计图进行前端页面的开发，后端拿到前端人员的前端页面后与后端服务接口进行集成，测试人员进行测试，运维人员将项目进行部署上线。我们发现这个流程中存在几个严重的问题。首先，当项目上线后，若发现生产环境的展示效果与测试环境有差异，需要修改前端页面时，由于前端人员没有后端的代码运行环境，所以就不知道服务接口返回的数据格式有无问题，在调试起来十分困难，需要频繁和后端工程师交流联调，这样就导致了项目整体开发效率降低，还可能让开发人员在交流中产生矛盾。其次，后端在对接前端页面时，往往会往其中加入一些后台代码，比如 jsp、freemarker、thymeleaf 等静态页面模板引擎[4]需要的语法，这就导致前端人员在阅读代码时产生了障碍，增加了他们的学习成本。由于 RESTFUL[5]风格 API 的提出与 JSON[6]数据格式的发展使得前后端交互日益便捷，前后端分离开发模式[7]变得流行起来，这种模式只需要事先让前后端协作定义好 API 接口文档，前端工程师就可以专注于页面的开发，在后台接口没提供的情况下，前端可以使用数据构造工具(Mock.JS)进行接口模拟。得到后台提供的接口后通过 ajax 技术调用 API 接口获取所需数据进行展示，后端工程师可以采用任何服务端编程语言(Go、Python、Java、Php 等)按照 API 接口文档规定的格式进行业务接口开发，这样可以达到前后端开发人员并行开发的效果，团队整体的开发效率就大大提升了，整个前后端分离的技术架构图如下图 2-1 所示。这种开发模式也推动了前端领域 Web 框架的发展，出现了 Vue.JS、AngularJS、ReactJS 这些

优秀的前端 MVVM[8]框架。

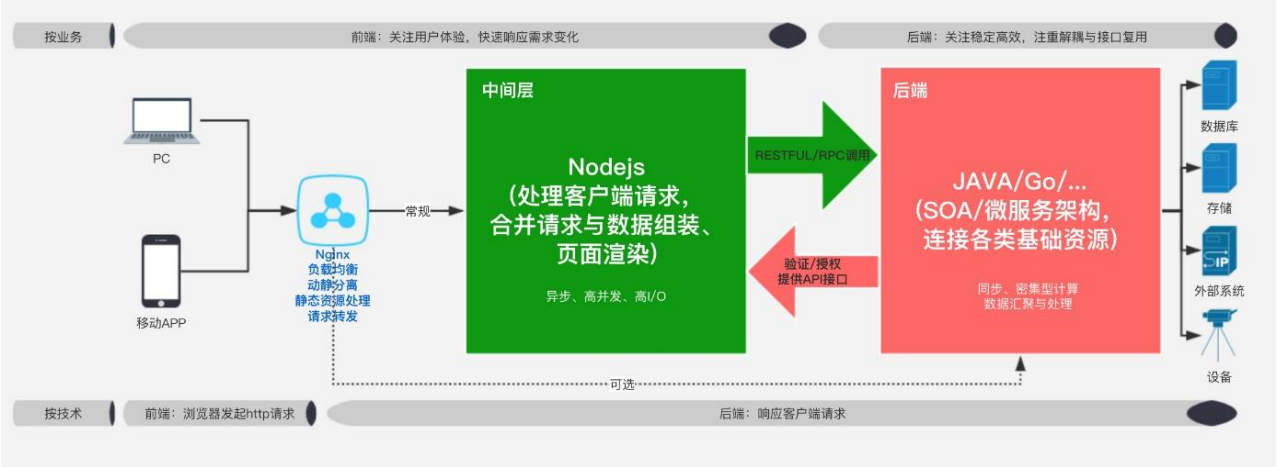

图 2-1 前后端分离企业架构

2.1.2 容器化

虚拟化技术[9]在 it 领域中使用十分普遍，知名的技术有 VMware，Xen，KVM 等，这些虚拟化技术都以操作系统底层为出发点，制作出一个和真实操作系统几乎一模一样的沙盒环境，但此类虚拟化技术的缺点也很明显，资源暂用多、冗余步骤多、系统启动慢，导致其不能满足现代化微服务架构快速部署的需求。正是由于需要快速部署的需求，新的虚拟化技术 Container 诞生了，它直接将代码和其所依赖的环境打包，方便开发人员将一个应用程序在不同的计算环境中进行可靠的迁移，它与传统虚拟化技术的设计原理对比区别如下图 2-2。容器不需要绑定一整套操作系统，只需要包含该项目运行需要的环境与配置，使得开发人员可以快速将项目运行环境迁移到测试人员的机器上，在运维人员正式上线应用时，也只需要简单几步即可完美可靠的运行项目。另外容器实际上是一个服务器进程，所以启动起来十分快速，达到了高效部署微服务架构产品应用的目的。本系统使用的 Docker[10]容器引擎，是由 Go 语言编写的一款基于容器化技术开源产品，它可以帮助我们快速搭建开发环境以及对微服务架构中的各个服务进行弹性伸缩控制。

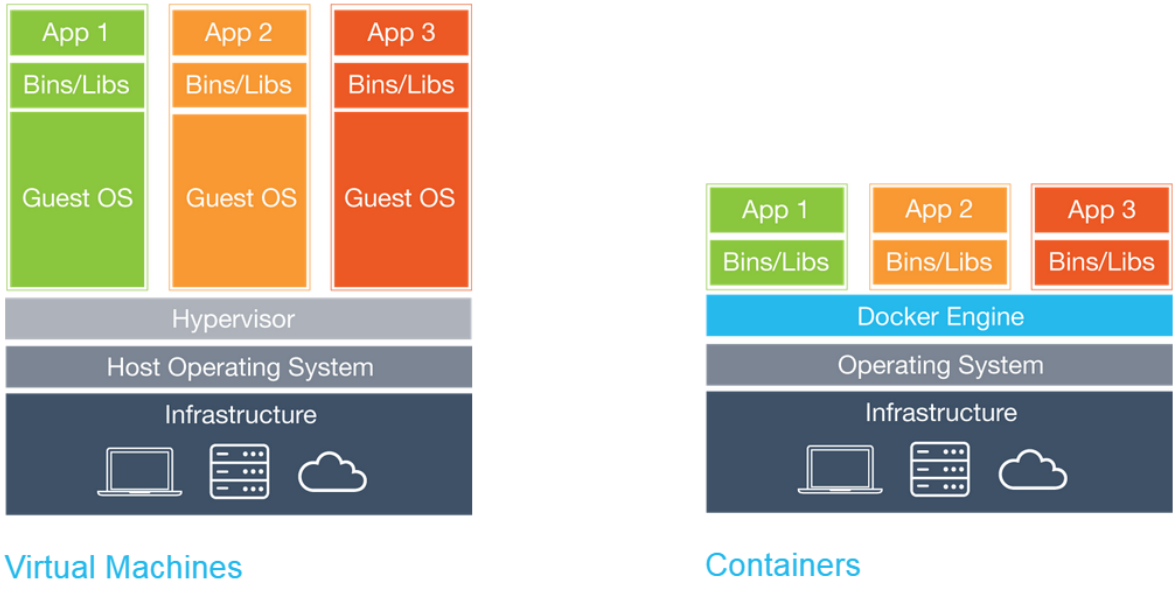

图 2-2 Virtual Machines 与容器（Container）的设计对比

2.1.3 微服务架构

在传统的系统架构中，项目往往都是单体应用，它的好处是开发起来比较方便，开发人员只需要编写一份代码，版本控制简单，部署起来也很容易。正如在传统的 MVC 技术体系中，项目开发完成后，只需要打包成 WAR 包部署到 Tomcat 中即可对外提供服务，早期在互联网流量小，业务需求的复杂度不高、技术人员人数不多的情况下，使用单体架构开发产品，公司在开发人员和运维成本上都是可控的。但随着业务规模随互联网技术的发展与网民的增加变得越来越庞大时，单体应用架构就逐渐开始出现了问题。首先，单体应用的代码变得越来越多，当编译打包时就会花费很长时间，部署效率低下，其次在公司中开发项目通常采用 Git 分布式版本控制工具，它使得一个团队中多个开发人员可以协作开发，但由于项目是单体应用，当测试阶段发现某处功能出现问题，就需要让所有开发人员都参与代码调试，调试完毕后，项目必须重新编译打包部署，然后重新预览测试，这就造成开发效率低下，时间成本极高。随着 Container 容器化技术的发展与普及，更重要是 Docker 这款划时代开源产品的诞生，让项目部署运行以及管理变得十分简单，微服务架构也就开始流行并逐渐成为应用架构的未来演进方向。微服务架构的设计思想是将庞大的单体应用按照功能模块进行细粒度的服务化拆分，并将各个拆分出来的服务进行独立打包部署到 Docker 容器中，各个服务之间通过 RPC 或 REST[11]接口互相调用，这样分别对各个小服务进行管理就很方便了，也可以轻松的对某个服务根据业

务场景进行弹性扩缩容。

2.1.4 DevOps

DevOps 是由 Development(开发)和 Operations(运维)两个单词缩写组成。当产品使用单体应用架构时，一个业务需求只需要修改一个应用的代码，并且针对这个单体应用进行测试，测试通过即可将单体应用的代码发布到线上，但当将技术架构切换到微服务架构时，一个大的系统被拆成多个小的系统，一个业务需求可能需要同时修改多个微服务模块的代码，这样就导致多个微服务都需要进行测试，测试全通过了才能将其发布到线上，显然工作量成倍的增加了，这时候就迫切需要一种新的开发、测试和运维模式来解决这个问题，这时候 DevOps 就孕育而出。在传统的开发模式下，开发人员、测试人员和运维人员的职责划分十分明确，他们往往分属于不同的职能部门，一次业务上线流程需要三者之间进行多次沟通交流，整个项目周期变得十分长。DevOps 就是基于此问题提出的解决方案，它将开发、测试和发布流程串联到一起，就像生产流水线那样，每个步骤完成后自动进行下一个步骤，无须过多的人为干预，业务迭代效率大大提升。在我看来，DevOps 是一种新型的业务研发流程，也是一种新的技术思维，它摒弃了传统的开发、测试和运维严格区分的观念，把三者的角色融为一体，让服务的开发者负责从开发、测试到发布的整个生命周期，真正的承担起服务负责人的角色，目前业界比较通用的实现 DevOps 的方案主要有两种，一种是使用 GitLab，一种是使用 Jenkins。本系统主要使用的是 Jenkins 来实现 DevOps。

2.2 关键技术

本系统使用了现代企业中主流的技术栈，在前端层、服务端层、数据库层都使用了较新的技术。

2.2.1 Spring Cloud

Spring Cloud 是 Spring 公司在微服务架构理念下创造的一整套优秀的解决方案，它不是一个单一的框架，而是一系列框架的有序集合，它的出现归功于 Spring 公司的另一款优秀的产品 Spring Boot 框架，Spring Boot 可以理解为 Spring Cloud 的脚手架，它具有约定大于配置的特性，使得之前使用 SSH(Struts，Spring，Hibernate)与

SSM(SpringMVC，Spring，MyBatis)的后端工程师们，抛弃了繁杂的 xml 配置文件，很方便就可以创建一个 Web 应用，正是这种开发便利性使得分布式系统基础设施的开发变得简化了。Spring Cloud 技术体系中包括分布式配置中心的管理、服务的发现、断路器、公共网关、消息总线、负载均衡[12]等功能组件，它们都是由互联网上优秀的框架通过 Spring Boot 的开发风格进行封装集成到 Spring Cloud 中，Spring Cloud 的设计思想没有浪费时间重新制作轮子，而是择其善者而用之，将成熟的、经过线上真实考验的框架组合起来，是其能够快速流行起来的重要原因之一。做到让开发者自由选择适合自己公司的框架集成到微服务架构中。

Spring Cloud 主要组件的框架如下所列：

1、服务发现--Netflix Eureka

2、服务调用--Netflix Feign/Ribbon+RestTemplate

3、熔断器--Netflix Hystrix

4、服务网关--Netflix Zuul

5、分布式配置--Spring Cloud Config

6、消息总线--Spring Cloud Bus

7、链路监控--Spring Cloud Sleuth

2.2.2 基于 JWT(JSON Web Token)的 Token 身份认证机制

在 Web 系统中，常见的身份认证机制有 HTTP Basic Auth(根据用户提供的账号密码进行认证)、Cookie 认证机制(用户发起请求后，服务端会在本地创建一个 Session 对象，并且使得用户本地浏览器中存储了一个 Cookie 对象，该对象中包含了刚才创建的 Session 对象的 id 号，当浏览器再次访问该服务器，服务端会将请求的 HTTP Header 中的 Cookie 值取出与存储在本地的 Session 对象进行比较来实现认证效果的)，上述两类认证方式在某些方面都有问题，前者存在账号密码这类隐私信息泄露的风险，而后者则存在浏览器跨域问题，更可能引起 CSRF(Cross-Site Request Forgery)攻击，而且服务端的存储压力也会随着用户访问量的增大而变大，网络响应速度也会被影响。Token

Auth 是本系统所使用的现代化身份认证机制，它解决了上述两种传统认证机制出现的问题，支持了跨域访问(由于不需要传递 Cookie)，具有无状态(服务端可扩展行)、去耦、对移动端 app 应用支持良好(同样是由于不需要使用 Cookie)等特性，它也解决了 CSRF 攻击的风险，性能极佳。

JSON Web Token[13]是用于为应用程序创建访问令牌的标准。它的工作方式是：服务端生成一个证明用户身份的令牌，并将其发送客户端。客户端存储服务端传过来的该令牌，并在之后的每个请求的请求头部(HTTP Header)中携带该令牌，服务器根据 JWT 的规定标准进行验证，由此得知该请求是否来自合法用户，这种体系结构在现代 Web 应用程序中被证明非常有效，在用户通过身份验证后，就可以通过调用 RESTFUL 的 API 接口进行功能调用，JWT 的身份认证机制原理如图 2-4 所示。

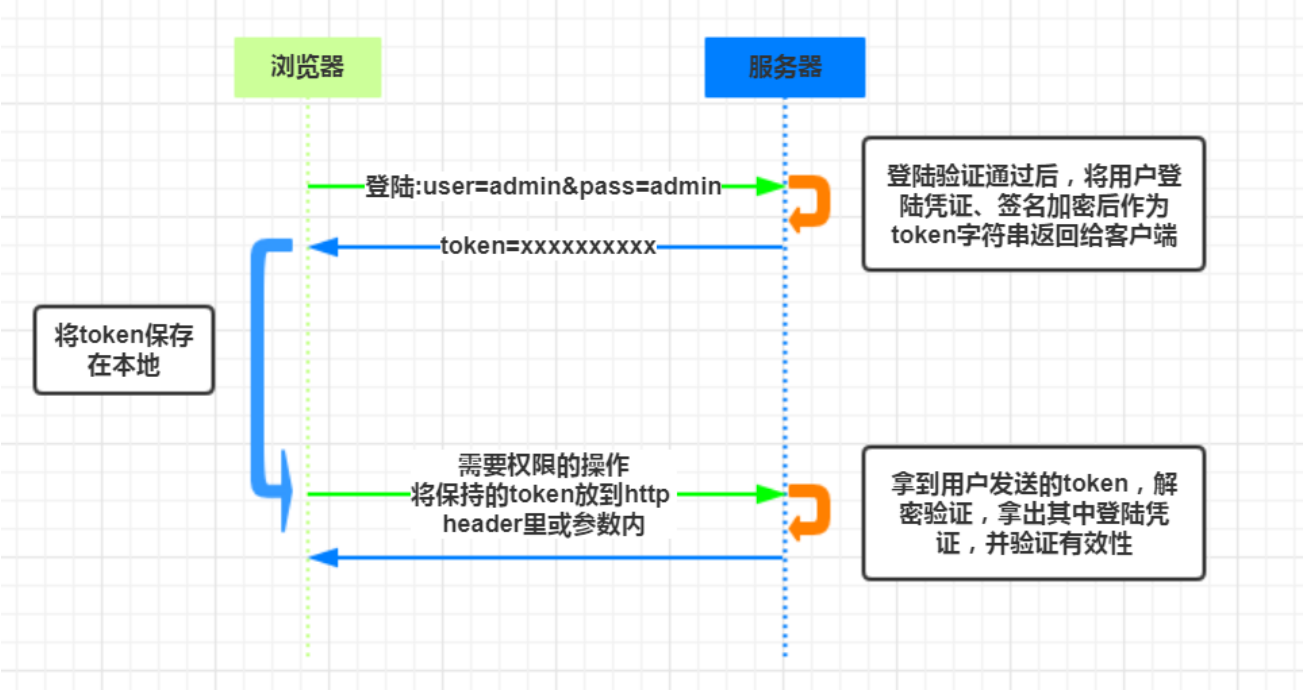

图 2-4 基于 JWT 的 token 认证机制原理

2.2.3 ELK Stack(Elasticsearch, Logstash, Kibana)技术

Elasticsearch、Logstash 和 Kibana 这三款开源的框架工具组合成了互联网上流行的 ELK[14]技术栈，此技术栈在企业中经常作为实时日志分析平台使用，它的技术原理图如下图 2-5 所示。由于传统的应用日志一般都是输出到服务器(apache，Nginx 等)的日志文件中，这些日志通常是开发人员在调试程序时必备的工具，但在线上部署的应用出

现问题需要排查时，由于开发人员没有登录线上服务器的权限，需要运维人员登录服务器取出日志交给开发，这样长时间下来，对运维人员来说是一个不小的工作量，大大影响了运维的工作效率，在当公司部署了 ELK 实时日志分析平台后，开发人员随时都可以登录到 Kibana 中进行日志的查看，大大缩短了问题的排查时间。

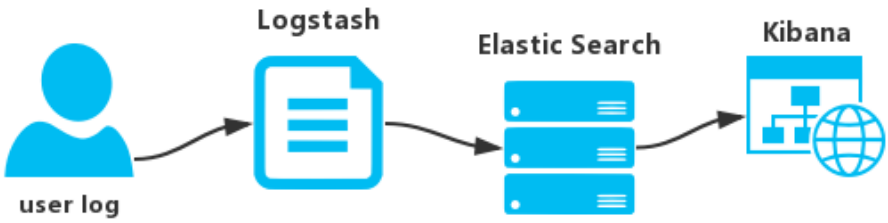

图 2-5 ELK Stack 原理图

2.2.4 Vue 框架

Vue 是一个采用 MVVM 架构，通过数据驱动视图的形式来构建用户界面的渐进式框架，其中"渐进式"的意思表示 Vue 的理念是从少到多，从弱到强，不强制开发人员一下使用其全部的功能，Vue 从设计角度上可以分为声明式渲染、组件系统、客户端路由、构建工具、大规模状态管理几大部分。Vue 使用分层思想对整体框架进行了合理的设计，使得你可以在 Vue 核心基础库上进行灵活更换其他层上的解决方案，不需要一定要使用 Vue 自己提供的，这样就给予开发者很高的自由度。Vue 的核心特性就是数据驱动视图，这也是现代前端框架比传统框架做出最卓越的进步，Vue 核心库只关注视图层，视图和数据状态是自动绑定的，实时保持同步。MVVM 是一种软件的架构模式，它是在 MVC 架构上进行扩展的。在 MVC 架构下，从项目结构角度上可以划分为用户展示 View 层、进行路由控制与业务逻辑操作的 Controller 层和数据库交互的模型层 Model。由用户在 View 上触发事件，Controller 根据不同事件，执行相应的业务逻辑，并修改 Model 的数据状态，最后 Model 中的新数据使得 View 进行视图更新，用户因此在页面上看到了操作反馈。而在 MVVM 架构中，Controller 部分被替换成了 ViewModel。它作为 View 层与 Model

层进行沟通的桥梁，负责监听 Model 和 View 的修改，由此实现了 View 和 Model 的”双向绑定“。即当 Model 层的数据被修改时，ViewModel 层检测到了变化后，主动通知 View 层进行相应的视图修改，反之，当 View 层做了修改后，Model 层的数据也会得到相应的修改，MVC 与 MVVM 两种模式的原理对比如下图 2-6 与图 2-7 所示。

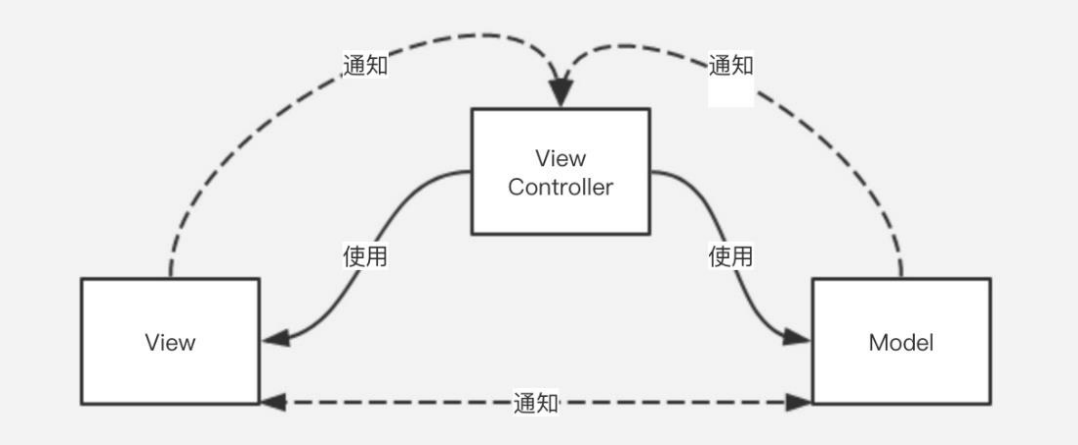

图 2-6 传统 MVC 架构图

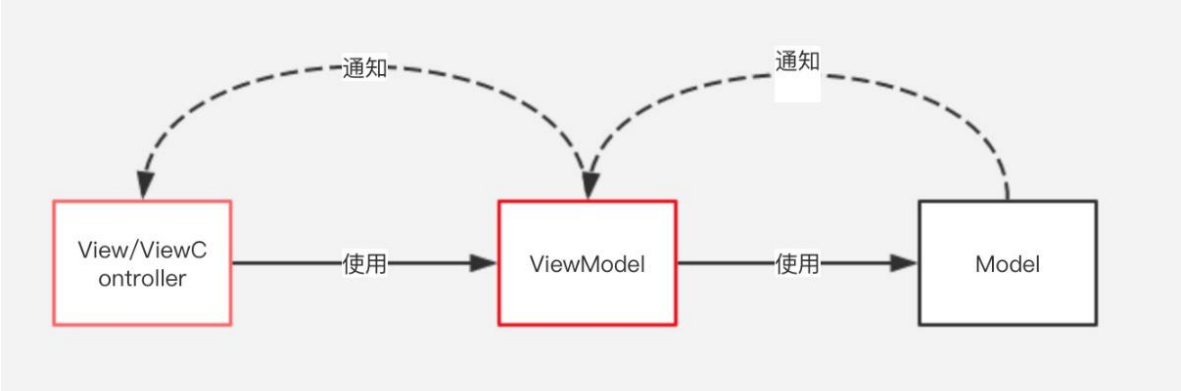

图 2-7 现代 MVVM 架构图

第 3 章 系统分析

3.1 应用场景

由于互联网上的信息资源太过繁杂，程序员想寻找问题的解决方案往往需要浏览很多平台，比如 CSDN、百度知道、Stackoverflow。本系统的设计目的就是实现一个方便程序员发起问题，解答别人的问题，并且提供优秀的技术文章浏览、还可以作为一个交友平台，让程序员的社交圈子得到扩展。由于之前使用过传统的单体应用架构进行过大型系统的开发，在开发过程中发现了一些问题，流行的微服务架构理念正好解决了这些问题，所以就采用了 Spring Cloud 微服务技术来实现本系统，借此学习了微服务架构的设计思想以及它与传统单体应用相比做出的卓越改进。

3.2 系统需求分析

本系统的使用角色分为非注册用户、注册用户、管理员，以下分别对三者进行描述：

- 非注册用户： 拥有对文章、问题、吐槽等信息的浏览权限。
- 注册用户： 拥有对文章、问题、吐槽等信息的浏览、评论、回答等权限，可以修改个人信息，可申请文章专栏以便对自己所有文章的分类分享。
- 管理员： 拥有该系统所有信息资源的最高操作权限，包括增删改查、封禁用户等权限。

3.2.1 系统用例

本系统用户分为前台用户与管理员，下面分别绘制其用例图。

前台用例图如 3-1 所示。

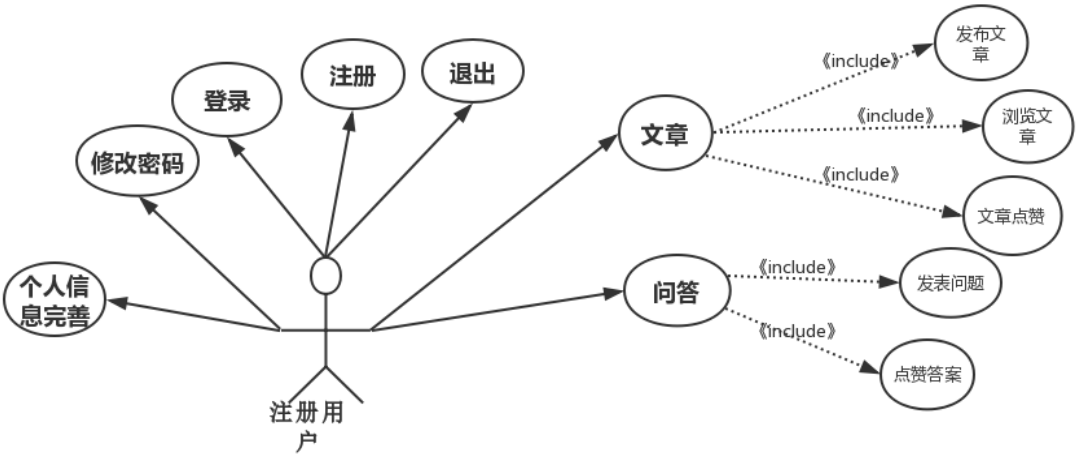

图 3-1 前台用例图

后台用例图如 3-2 所示。

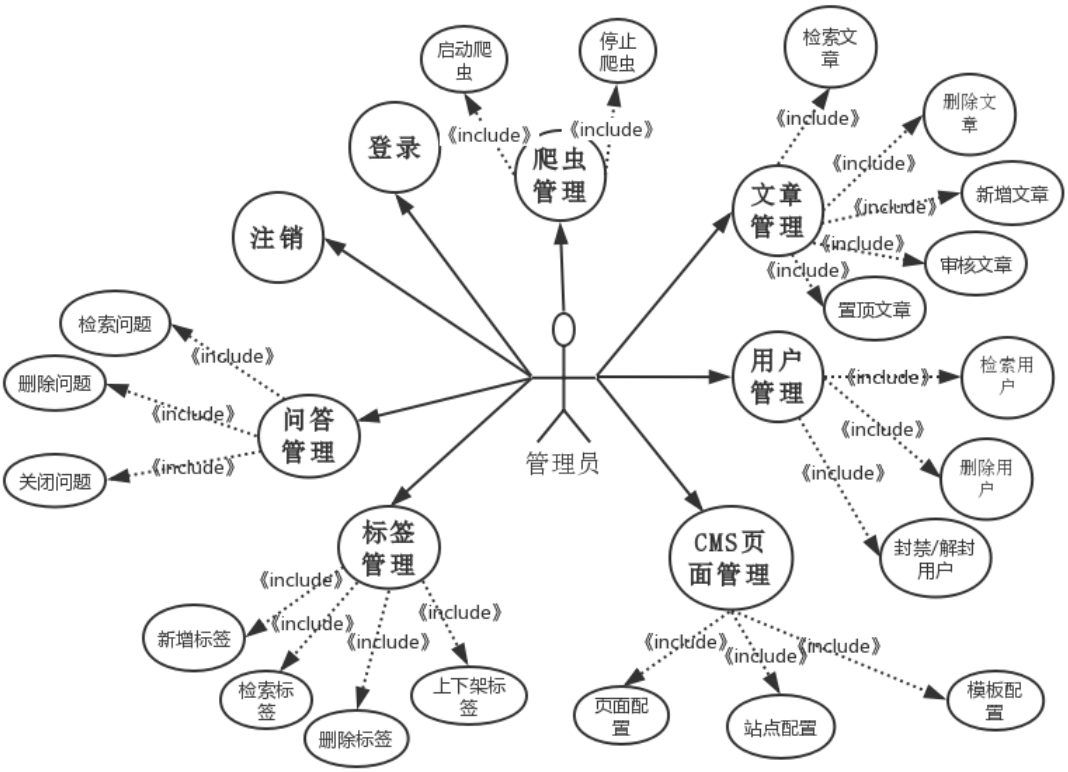

图 3-2 后台用例图

3.2.2 业务流程

主要介绍登录注册模块、文章模块、问答模块的业务说明。

(1) 登录注册

当用户点击首页上登录注册按钮时，会跳转到登录注册页，输入符合要求的登录注册信息后会触发登录注册流程，或者当用户在未登录状态下在文章、问答模块进行发布、点赞、回复等操作也会触发该流程，该流程的流程图如下图 3-3 所示。

说明：手机号通过正则表达式进行验证，可以验证包括大陆和香港的手机号类型。手机验证码为长度 6 位的数字字母组合序列，有效期为 5 分钟，在一分钟内禁止重复发送验证码，此操作限制由前端页面和后端接口同时把控。用户密码长度至少 6 位。

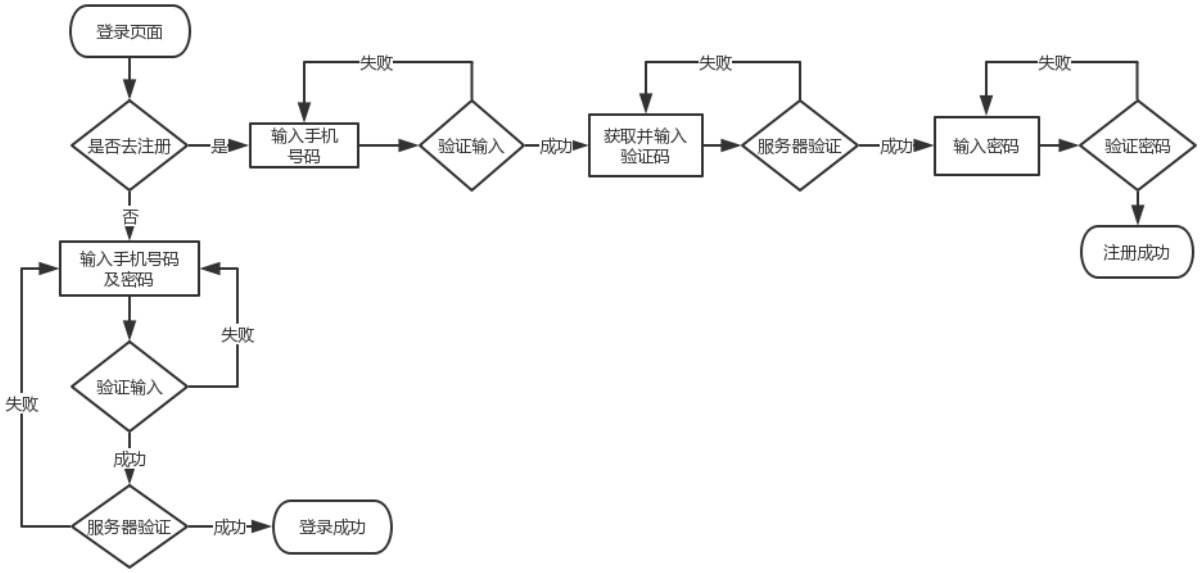

图 3-3 登录注册流程图

(2) 文章模块

未登录用户可以进行热门文章的浏览，也可以按照不同标签分类进行选择性阅读。当用户想要评论文章、发布文章、申请专栏时，需要用户登陆后进行操作，系统在这些操作的反馈中会自动判断用户当前登录状态，若发现用户未登录，会自动进入登录流程中，该流程的流程图如下图 3-4 所示。

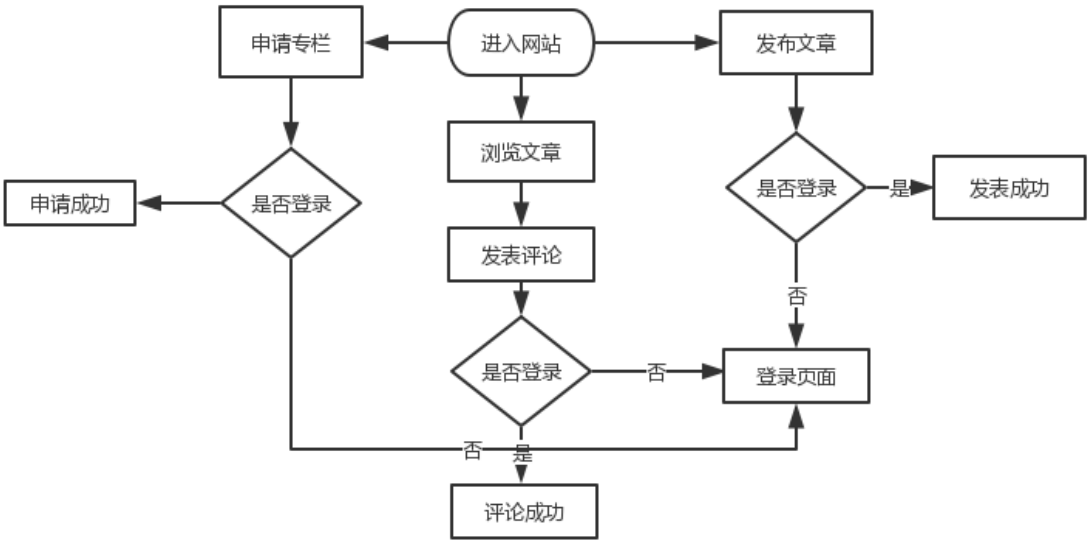

图 3-4 文章模块流程图

(3) 问答模块

用户进入问答模块后，可以通过页面上的各类标签来选择问题的技术方向，未选择标签时，该系统会将浏览量最高的前 5 个标签推荐给用户。用户点击相应技术标签后进入问题列表页面，用户可以翻页浏览也可通过页面上方的搜索框进行问题的查寻。进入问题详情页面查看解决方案，若解决方案不能解决当前遇到的问题，可以选择发布自己问题的帖子，也可以对相似问题帖子进行追加评论，询问更详细的解决方法。在本系统中用户不仅可以寻求帮助，也可以主动回复他人的问题帖子进行解答，优质答案会被其他用户进行点赞置顶，该流程的流程图如下图 3-5 所示。

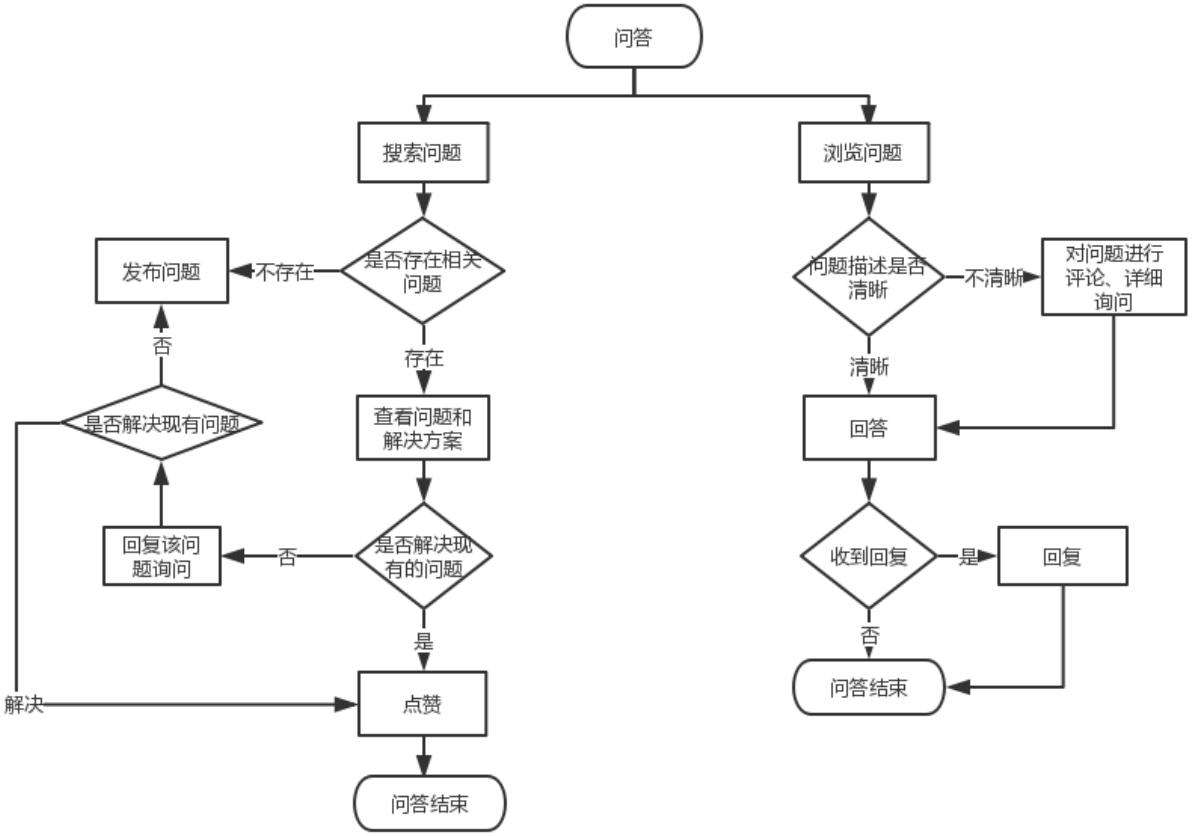
问答
搜索问题
浏览问题
是否存在相关问题
不存在
发布问题
存在
问题描述是否清晰
不清晰
对问题进行评论、详细询问
清晰
否
查看问题和解决方案
回答
是否解决现有问题
是否解决现有的问题
否
回复该问题询问
收到回复
是
回复
是
否
解决
点赞
问答结束
问答结束

图 3-5 问答模块流程图

第 4 章 系统设计

4.1 系统架构

本系统使用现代企业常用的技术架构体系进行开发，系统结构如下图 4-1 所示：

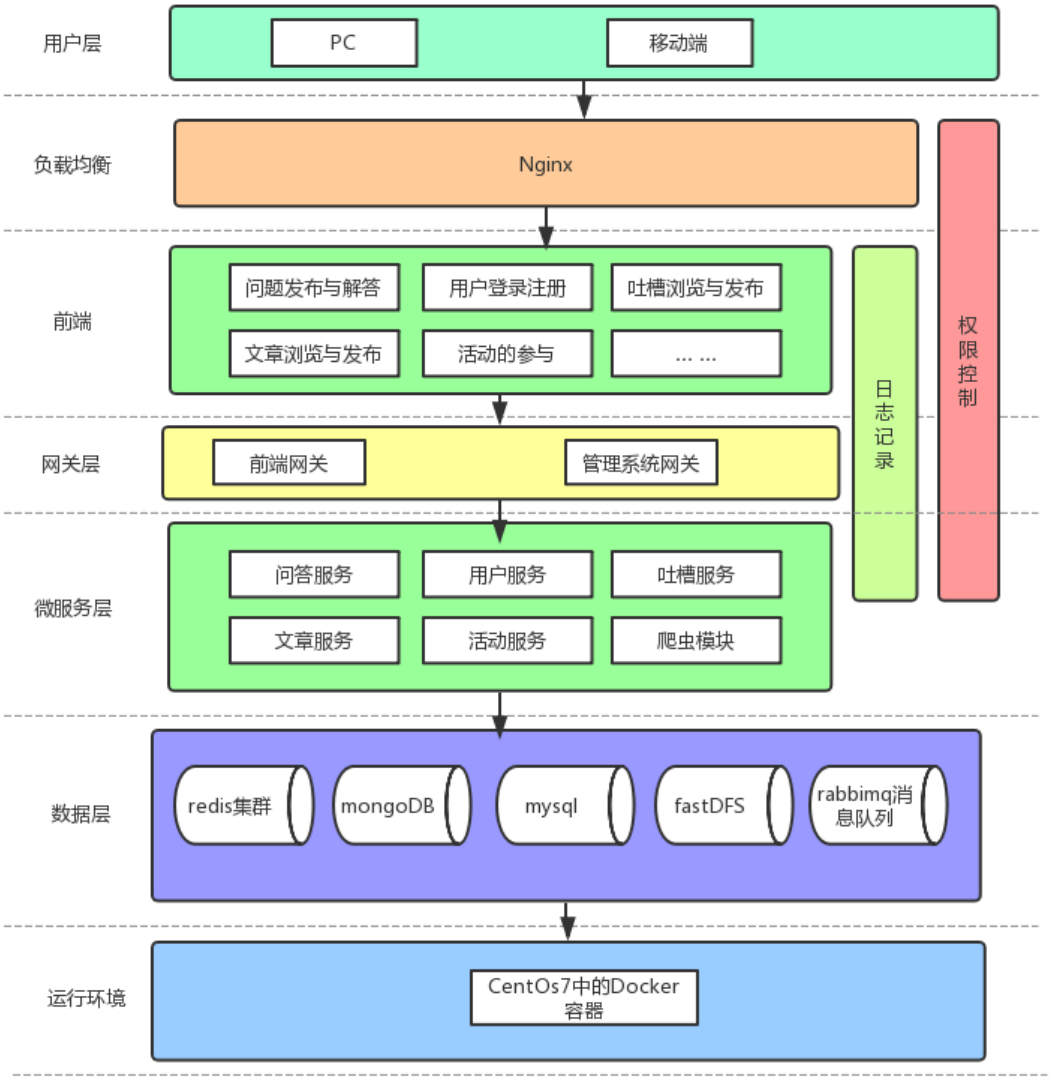

图 4-1 微服务架构图

对上述架构图进行如下解释：

1. 用户层：本层描述了系统所支持的客户端用户种类，包含了 PC 与移动端用户了。
2. 负载均衡：本系统中用户不是直接访问前端页面，而是需要经过 Nginx 服务器作为负载均衡中间件，这样设计可以让我们的应用部署到多台服务器上，然后通过负载均衡技术将用户的请求分发到不同的服务器以此来解决单点故障问题和提高网站的整体性能。
3. 前端层：向 PC 和移动端用户提供了操作体验友好的 Web 界面。
4. 网关层：本层的作用是将路由进行转发和过滤，解决了服务拆分很多时，各个服务的服务地址维护起来十分麻烦，使用网关进行统一管理并转发，更可以做到权限验证，请求拦截等高级功能，本系统使用 Spring Cloud 中的 Zuul 组件来实现网关功能，它默认集成了负载均衡的功能，十分强大。本项目将网关层拆分成了 2 个微服务的目的是用户前端层和管理系统前端层各自使用服务的目的是有差异的，管理员除了需要检索文章、吐槽等信息外还需要对这些资源进行增删改操作，所以在网关层进行角色权限划分后，在微服务层的接口调用中变得更加安全和减少了大量冗余代码。
5. 微服务层：本层提供了系统所有业务功能需要的服务，是以 SpringBoot 为脚手架创建的。它们通过 Spring Cloud 的 Eureka 组件进行服务注册，并通过 Feign 组件进行服务间的互相调用，返回的数据格式是 JSON，便于前端处理。
6. 数据层：本系统使用了 Redis 作为缓存数据库，并搭建了三主三从的高可用架构，用于文章信息的缓存与爬虫 Spider 模块去重处理。使用 MongoDB 作为吐槽模块和 CMS 模块数据的存储，并使用其集成的 GridFS 文件系统作为 CMS 页面模板文件存储的媒介。使用开源免费的 Mysql 数据库作为主要数据存储，它服务于文章、用户、交友、爬虫等数据相当重要的服务模块。在用户注册业务中使用了 Rabbitmq 消息队列作为异步操作处理的解决方案，避免了程序因为一个操作阻塞住而影响整个请求的响应时间。相对于其他文件系统，本系统使用的 FastDFS 分布式文件系统非常适合存储图片等小文件，因为 FastDFS 不对文件进行分块存储，少了最后取文件时分块合并的时间开销，性能很高，并且其网络通信采用 socket，通信速度很快。本层所使用的软件工具都是采用 Docker 拉取 DockerHub 上的镜像的方式进行部署，非常方便快捷，几分钟就能完成数据层的构建。

7.运行环境：选用 Linux 的 Centos 发行版作为服务器，并通过 Docker 容器进行各个微服务的部署与管理。

4.1.1 接口定义

整个系统使用前后端分离模式进行开发，所有微服务的 API 接口开发统一使用 Swagger 注解进行描述，Swagger 技术是 OpenAPI 规范中最成功的实现，受到了全世界 it 人士的广泛赞誉和使用。本系统的接口定义规范如下：

- POST 请求： 前端请求 Form 表单数据(application/x-www-form-urlencoded)和 json 数据(application/json)、多部件类型数据(multipart/form-data)，对于 json 数据，SpringMVC 框架采取@RequestBody 注解来接受。
- GET 请求：前端请求携带 key/value 结构参数，SpringMVC 框架采用基本数据类型或自定义类型接受。
- 响应使用统一的 json 返回格式:{flag:“success”,code:200,data:{},message:“操作成功”}

4.1.2 微服务注册中心

使用 Spring Cloud 中的 Eureka 组件进行服务的注册与发现，它的作用与阿里巴巴公司 Dubbo 微服务解决方案中的 Zookeeper 工具一样，都是用来协调多个微服务之间互相调用关系的,它的原理是:经过在 SpringBoot 项目的 yml 配置文件中进行配置 eureka 服务端的地址后，当服务启动会自动向 Eureka 服务端进行注册，Eureka 服务端会将每个服务的注册信息向其他 Eureka 服务端进行同步，当服务消费者需要使用相应的服务时，就会向服务注册中心获取服务提供者的地址，并且会将其缓存到本地，下次可直接从本地缓存中获取进行服务调用，本系统使用两台 Eureka 服务端进行互相注册，组成高可用的特性，服务注册中心的设计原理如下图 4-2 所示。微服务远程调用使用 Spring Cloud 的另一个组件 Feign，它是一种客户端负载均衡技术，与 Nginx 服务器负载均衡相辅相成，共同维持整个系统的高性能，大大提高了网站的吞吐量，服务间远程调用的设计原理如下图 4-3 所示。

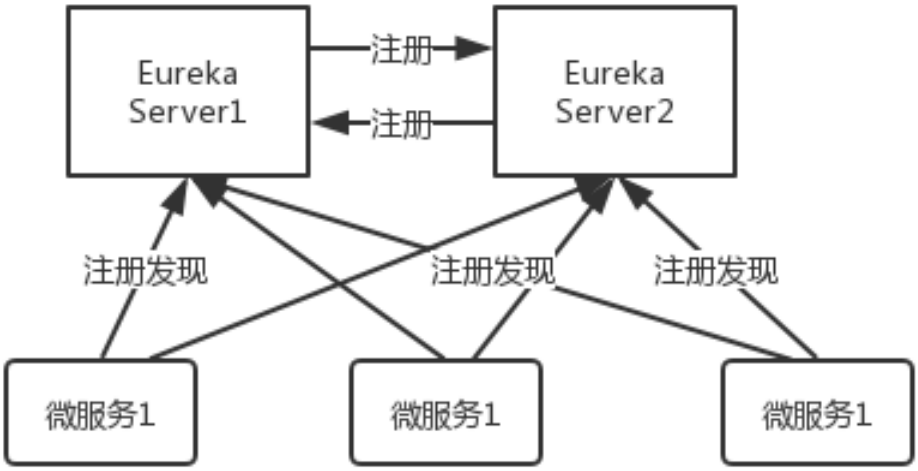

图 4-2 Eureka 服务注册中心原理图

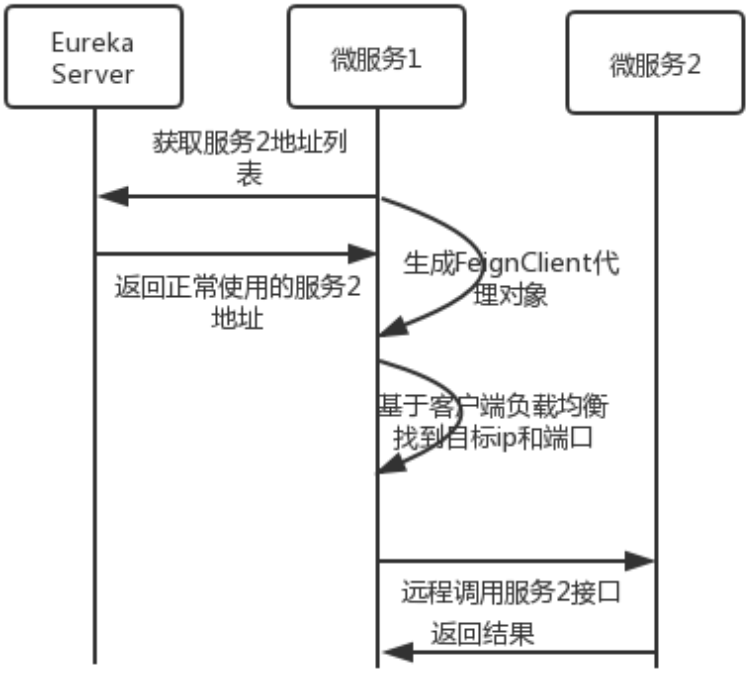

图 4-3 服务间远程调用原理图

4.1.3 网关管理

本系统使用 Zuul 组件作为微服务架构中网关的实现，它分为管理员 Manager 网关和用户 Web 网关，用以给不同角色系统提供 API 服务，Manager 网关提供给管理员后台用来对资源做高级权限的操作，Web 网关提供用户使用正常的功能，这种职责划分清晰的设计保证了该系统的安全性。Zuul 网关在微服务中起到负载均衡、路由转发、权限认证、请求拦截过滤等功能。本系统采取网关 Zuul 与 Nginx 服务器配合使用，设计图如下图 4-4 所示：

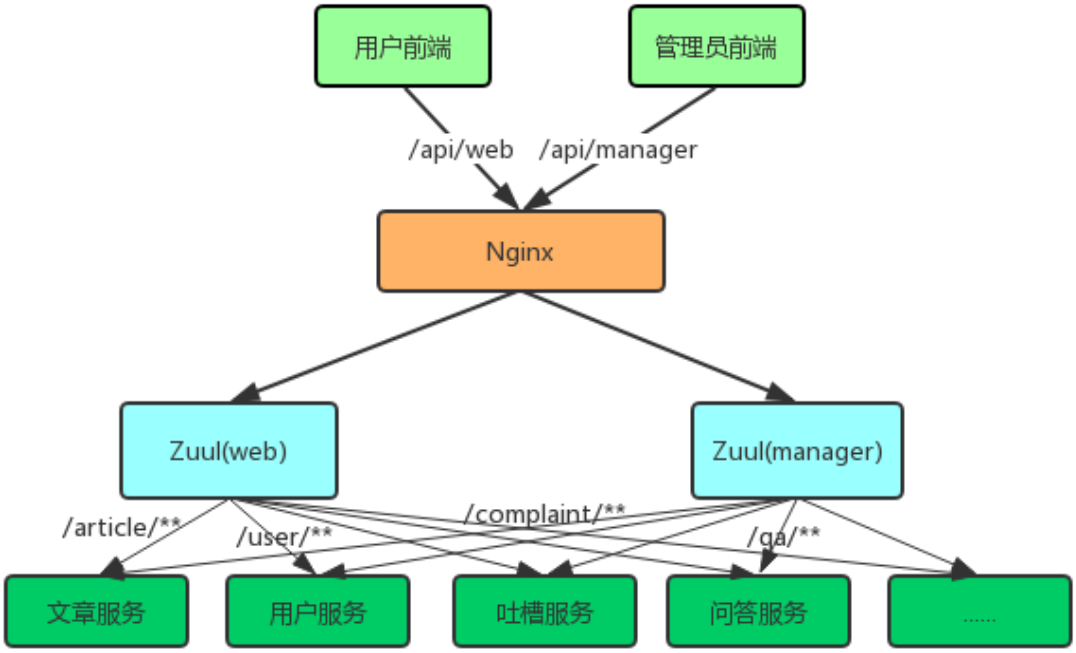

图 4-4 网关负载均衡

4.1.4 容错机制

由于在微服务架构中，某个业务功能往往需要多个服务模块相互调用来实现，这就可能出现某个调用过程出现错误，这时如果没有相应的容错保护，就容易导致雪崩效应[15]。微服务的雪崩效应体现在服务之间的调用，当一个服务因为某种原因提供不了服务时，就可能导致其它的服务也崩溃。比如在问答模块中问题资源需要调用 Base 基本模块中的标签资源进行相关联查询，如果 Base 模块服务器出现了宕机等不可预知问题，

导致服务接口长时间无法响应，从而使得问答模块中问题按标签分类列表功能无法使用，像这样由一个服务崩溃引起其他服务不能正常输出服务就是雪崩效应。本系统使用 Spring Cloud 中的 Hystrix 组件，用来解决上述出现的问题，实现了断路保护，其原理如下图 4-5 所展示。

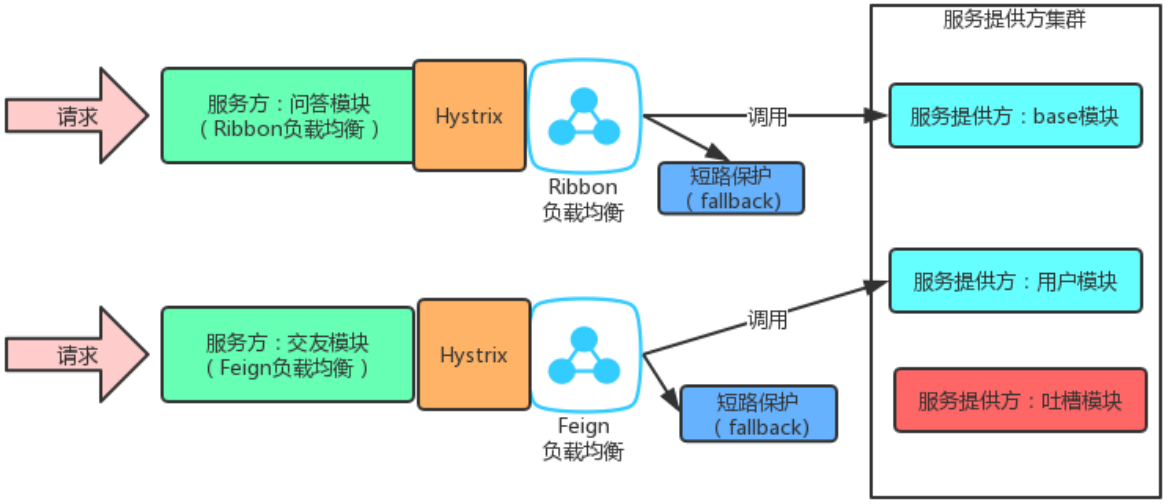

图 4-5 Hystrix 短路保护

4.1.5 认证机制

本系统使用了现代化的身份认证机制 Token Auth，当用户正确登录系统后，后续对各个资源微服务的访问以及对资源操作的授权都变得极为方便，同时使用了高度安全的 JWT 标准来生成令牌。整个认证流程为：当用户输入正确的账号密码后，其身份就被服务端验证，服务端根据预设的指定算法生成 JWT 令牌并返回给浏览器端，之后客户端发起的 HTTP 请求的头部携带上该 JWT，服务端就可以对其身份的合法性进行验证。本系统身份认证整体设计如下图 4-6 所示。

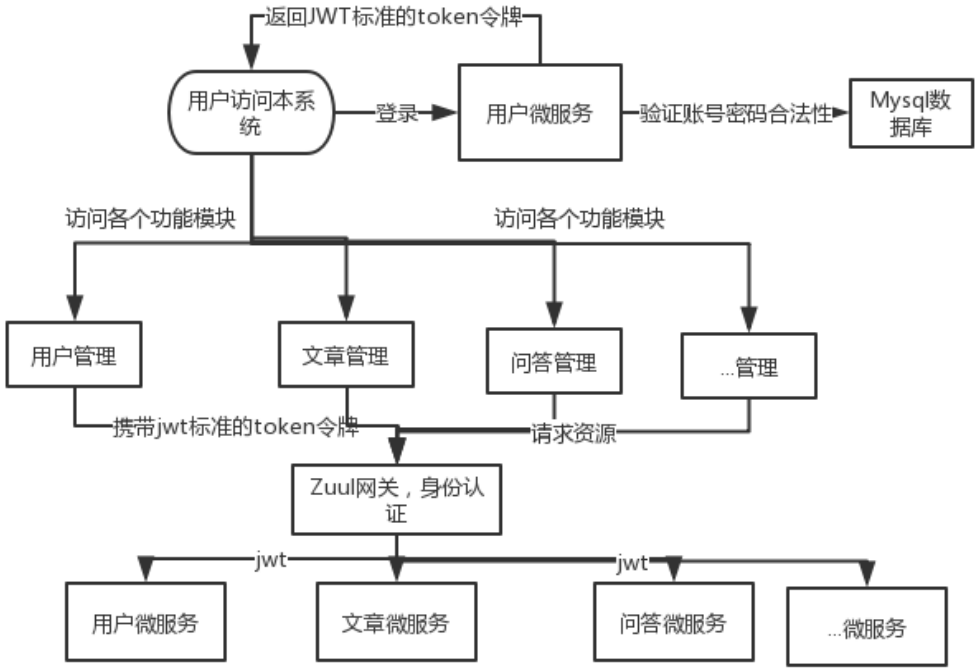

图 4-6 身份认证原理图

4.1.6 CMS 页面管理

本系统的在线课程学习门户页面属于高访问量、具体高时效性的信息发布特性。运营人员需要时常上传图片和文字信息更改门户页面前端展示，与京东商城首页类比，轮播图、特价商品随着时间的推移在不断的变化，所有就有必要开发一个 CMS 内容管理系统来对各个子站点的页面进行管理，从而实现随着运营需求的变更快速页面开发、上线的需求，CMS 页面的发布流程如下图 4-8 所示。主要采用 Nginx 服务器中的 SSI 服务端包含技术，本系统将门户首页不同区域的代码拆开编写，采用类似 JavaWeb 开发中的 Jsp 页面的 include 指令，来将刚才拆分的部分合并成一个页面，最终渲染返回给用户查看。拆分页面文件如下图 4-7 所示：

```
-rwxr-xr-x 1 laoji games 7.1K Mar 27  2018 course_detail_dynamic1.html
-rwxr-xr-x 1 laoji games  12K May 25  2018 course_detail_dynamic.html
-rwxr-xr-x 1 laoji games 2.3K Feb 24  2018 course_detail_side.html
-rwxr-xr-x 1 laoji games 1.5K May 31 01:44 footer.html
-rwxr-xr-x 1 laoji games 6.0K May 31 01:43 header.html
-rwxr-xr-x 1 laoji games 2.3K May 31 02:39 index_banner.html
-rwxr-xr-x 1 laoji games  14K Jan 22  2018 index_category.html
```

图 4-7 门户静态文件拆分图

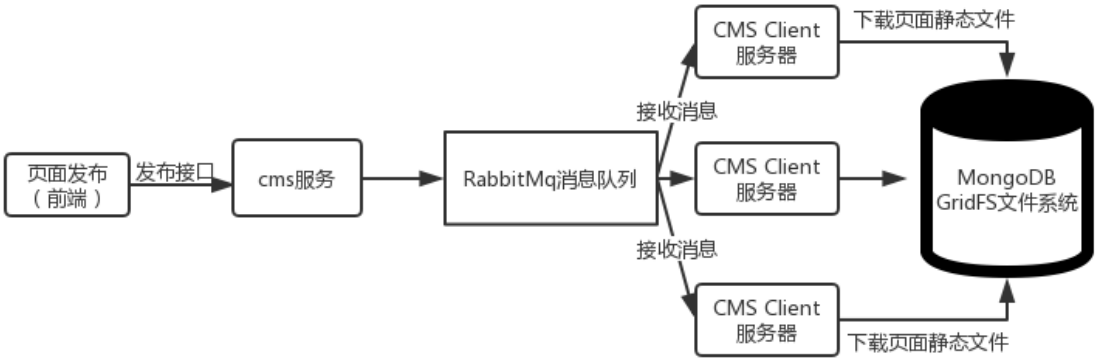

图 4-8 CMS 页面发布流程

4.1.7 持续集成流程

DevOps 中包含 CI(Continuous Integration)--持续集成、CD(Continuous Deploy)--持续部署，其中 CD 还有另外一个解释是持续交付(Continuous Delivery)。持续集成在企业项目开发中十分重要，在企业中各个小组分别负责不同功能模块的开发，即使单一模块独立测试没有问题，但当把所有模块合并到一起做集成测试时往往就会出现很多问题，这时需要将大量代码打回返工重写，经过几次往返，产品的交付日期就被拖延到很后面，所以频繁将所有功能模块集成到一起进行测试，这样可以让问题提早被发现与解决，这就是持续集成的思想，它关注的是需要尽早发现分模块开发的项目在整体部署运行时出现的问题，并尽早解决。避免到将问题遗留到项目后期，那时就很难排查和解决，下图 4-9 是本系统持续集成设计图。

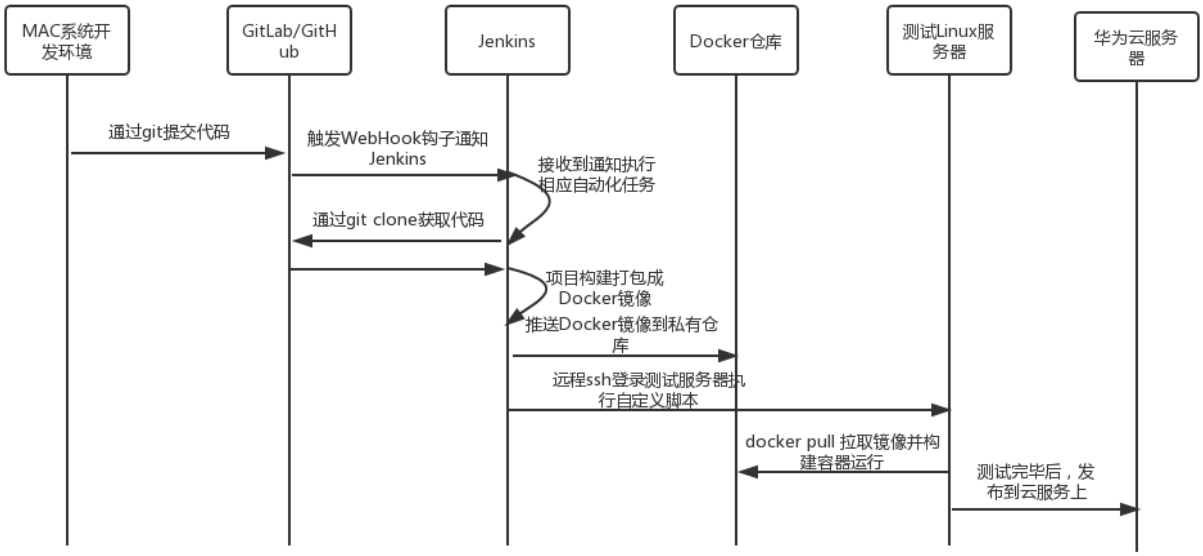

图 4-9 持续集成设计图

本系统采用 Maven 项目管理工具，用以提供管理项目中所有的 jar 包依赖以及项目工程打包服务。项目的手动部署方式： 使用 Maven 中 Docker 的插件 docker-maven-plugin 可以做到使用命令 mvn clean package docker:build -DpushImage 来自动编译打包项目并封装成 Docker 镜像，最后上传到用户配置的私有仓库中，最后通过 docker pull 命令拉取项目镜像并通过 docker run 来启动一个容器来运行该项目。Maven 的 Pom 文件中的关键配置如下图 4-10 所示：

```
<build>
    <finalName>app</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>0.4.13</version>
            <configuration>
                <imageName>172.16.103.130:5000/${project.artifactId}:${project.version}</imageName>
                <baseImage>jdk1.8</baseImage>
                <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>
                <resources>
                    <resource>
                        <targetPath>/</targetPath>
                        <directory>${project.build.directory}</directory>
                        <include>${project.build.finalName}.jar</include>
                    </resource>
                </resources>
                <dockerHost>http://172.16.103.130:2375</dockerHost>
            </configuration>
        </plugin>
    </plugins>
</build>
```

图 4-10 Pom 文件中持续集成相关配置

自动部署方式：使用 Git 分布式版本控制工具来提交代码，代码远程仓库使用开源免费的 Gitlab，在 Gitlab 中配置相应的 WebHook 钩子来根据用户的某种 git 行为响应操作，本系统使用 git push 操作钩子来触发 Jenkins 的自动构建任务，Jenkins 会将所有任务的构建成功与否结果显示在它提供的前端页面中，如下图 4-11 所示。在 Jenkins 中配置 Shell 脚本用来做一些前置和后置操作，如停止和删除之前正在运行的同一项目的容器，并且清理镜像，避免垃圾资源占用服务器磁盘。前置操作命令执行完后，就可以执行 docker 拉取镜像并运行容器的后置操作命令了，最后可以配置 Jenkins 邮件服务用以通知一个自动化构建任务成功与否结果。前置、后置操作 Shell 脚本命令如下：

前置操作 Shell 脚本：

```
#!/bin/bash
result=$(docker ps | grep "172.16.103.130：5000/workoverflow_base")
if [[ "$result" != "" ]]
then
echo "stop docker_workoverflow_base"
docker stop docker_workoverflow_base
fi
result1=$(docker ps -a | grep "172.16.103.130：5000/workoverflow_base")
```

```
if [[ "$result1" != "" ]]
then
echo "rm docker_workoverflow_base"
docker rm docker_workoverflow_base
fi

result2=$(docker images | grep "172.16.103.130:5000/workoverflow_base")
if [[ "$result2" != "" ]]
then
echo "172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT"
docker rmi 172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT
fi
```

后置操作 Shell 命令：

```
docker pull 172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT
docker run --rm -dt --name=docker_workoverflow_base -p 9001:9001
172.16.103.130:5000/
workoverflow_base:1.0-SNAPSHOT
```

图 4-11 Jenkins 自动化任务列表

4.1.8 前端技术栈

由于本系统是使用前后端分离模式进行架构，所以选用 Vue 作为前端框架，由于前

端工程 SPA 单页面应用,大量使用了 Javascript 技术,不利于各类搜索引擎对其的 SEO,比较适合对 SEO 没有要求的系统，所以本系统的管理员后台管理系统就使用的 Vue 的 ElementUI 框架进行开发，用户前端使用 Vue 与服务端渲染技术共同开发。服务端渲染(SSR Servder Side Render)是一种在服务端完成页面的内容，而不是在客户端通过 ajax 来获取数据拼装页面，它相对于单页面应用，拥有更好的 SEO，搜索引擎的爬虫能够获取完全渲染的页面，而不是一整段压缩后的 Javascript 代码。采取 Nuxt.js(基于 Vue 的服务端渲染框架)来进行用户页面的开发。采用 NodeJS 技术作为服务中间件，用来作为连接前端与后端微服务之间的桥梁，可以合并一些请求以减少请求的个数，加快响应速度，使用 Npm 包管理工具，类似于 JavaEE 中 Maven 工具，都是对项目中所依赖的第三方工具包进行下载和管理。使用 Webpack 模块打包工具，对前端的代码和资源进行打包优化，减少静态文件的大小，使得用户访问更快速，不会因为网络延迟而影响用户体验感。

4.2 数据库设计

由于本系统属于分布式系统架构，所以难免会出现某个瞬间高并发的情况，如果表的主键使用数据库自带的自增策略，会出现插入冲突的情况，所以本项目采用 twitter(国外的一个社交网络平台)开源的 snowflake 雪花算法进行生成每条记录唯一的主键值。为了模拟企业生产环境的开发方式，本系统对数据库进行了分库分表设计，使得每个微服务模块只使用与自己业务相关的数据库，本系统所使用的数据库如下图 4-12 所示：

图 4-12 系统所用数据库列表

4.2.1 数据字典

下面列举出本系统所使用到 Mysql 数据库中的表。

workoverflow_qa 问答模块数据库：

问题表（tb_problem）

表 4-1 问题表（tb_problem）

编号	列名	类型	字段含义	备注
1	id	varchar	问题主键	
2	title	varchar	问题标题	默认空字符串
3	content	text	问题内容	
4	create_time	datetime	创建日期	
5	update_time	datetime	更新日期	
6	user_id	varchar	问题发起人 id	
7	nick_name	varchar	用户昵称	默认空字符串
8	visits	bigint	浏览量	默认 0
9	thumb_up	bigint	点赞数	默认 0
10	reply_count	bigint	回复数	默认 0
11	solve	tinyint	解决标识	0 未解决 1 已解决
12	reply_name	varchar	最新回复昵称	
13	reply_time	datetime	最新回复时间	
14	accept_id	varchar	采纳的回答 id	

回答表（tb_reply）

表 4-2 回答表（tb_reply）

编号	列名	类型	字段含义	备注
1	id	varchar	回答主键	
2	problem_id	varchar	问题 id	与问题表关联
3	content	text	回答内容	
4	create_time	datetime	创建日期	
5	update_time	datetime	更新时间	

续表 4-2

编号	列名	类型	字段含义	备注
6	user_id	varchar	回答用户 id	
7	nick_name	varchar	回答用户昵称	
8	parent_id	varchar	父回答 id	
9	thumb_up	int	赞同数	

问答标签中间表（tb_pl）

表 4-3 回答标签中间表（tb_pl）

编号	列名	类型	字段含义	备注
1	problem_id	varchar	问题 id	联合主键
2	label_id	varchar	标签 id	联合主键

用户评价表（tb_thumb）

表 4-4 用户评价表（tb_thumb）

编号	列名	类型	字段含义	备注
1	id	varchar	主键	
2	user_id	varchar	用户 id	
3	message_id	varchar	信息 id	
4	type	tinyint	信息类型	1:问题 2:回答 3:文章 4:吐槽
5	create_time	datetime	评价时间	
6	thumb_type	tinyint	评价类型	1:点赞 2:踩

workoverflow_article 文章模块数据库

文章表（tb_article）

表 4-5 文章表（tb_article）

编号	列名	类型	字段含义	备注
1	id	varchar	文章主键	
2	column_id	varchar	专栏 id	
3	user_id	varchar	用户 id	
4	title	varchar	文章标题	
5	content	text	文章正文	
6	image	varchar	文章封面	
7	create_time	datetime	创建日期	
8	update_time	datetime	更新日期	
9	is_public	varchar	公开标识	0 不公开 1 公开
10	is_top	varchar	置顶标识	0 不置顶 1 置顶
11	visits	int	浏览量	默认 0
12	thumbup_count	int	点赞数	默认 0
13	comment_count	int	评论数	默认 0
14	collect_count	int	收藏数	默认 0
15	state	tinyint	审核状态	0 未审核 1 已审核
16	channel_id	varchar	所属频道 id	
17	url	varchar	文章地址链接	
18	type	tinyint	文件类型	0 原创 1 转载

专栏表（tb_column）

表 4-6 专栏表（tb_column）

编号	列名	类型	字段含义	备注
1	id	varchar	专栏主键	
2	name	varchar	专栏名称	
3	summary	varchar	专栏简介	
4	user_id	varchar	用户 id	
5	create_time	datetime	创建日期	

续表 4-6

编号	列名	类型	字段含义	备注
6	check_time	datetime	审核日期	
7	state	tinyint	审核状态	0 未审核 1 已审核
8	mark	varchar	唯一标识	专栏链接地址标识
9	url	varchar	专栏地址	
10	follow_count	int	关注数	

用户评论表（tb_comment）

表 4-7 用户评论表（tb_ comment）

编号	列名	类型	字段含义	备注
1	id	varchar	主键	
2	message_id	varchar	信息 id	
3	type	tinyint	信息类型	1:问题 2:回答 3:文章 4:吐槽
4	content	varchar	评论内容	
5	create_time	datetime	评论时间	
6	update_time	datetime	更新时间	
7	user_id	varchar	评论人 id	
8	nick_name	varchar	评论人昵称	
9	parent_id	varchar	父评论 id	
10	thumbup_count	int	点赞数	

workoverflow_base 基础模块数据库

标签表（tb_label）

表 4-8 标签表（tb_label）

编号	列名	类型	字段含义	备注
1	id	varchar	标签主键	
2	label_name	varchar	标签名称	

续表 4-8

编号	列名	类型	字段含义	备注
3	description	varchar	标签描述	
4	state	tinyint	状态	0 下架 1 上架
5	count	int	使用数量	默认 0
6	recommend	tinyint	推荐标识	0 不推荐 1 推荐
7	fans	int	关注数	
8	create_time	datetime	创建日期	
9	update_time	datetime	更新日期	

用户举报表（tb_report）

表 4-9 用户举报表（tb_ report）

编号	列名	类型	字段含义	备注
1	id	varchar	主键	
2	user_id	varchar	举报人 id	
3	message_id	varchar	被举报信 id	
4	type	tinyint	信息类型	1:问题 2:回答 3:文章 4:吐槽
5	report_level	tinyint	违规类型	1:推广 2:暴力敏感信息 3:人身攻击 4:其他问题
6	extra	varchar	额外说明	
7	state	tinyint	处理状态	0:未处理 1:已处理
8	create_time	datetime	评论时间	
9	update_time	datetime	更新时间	

workoverflow_user 用户模块数据库

用户表（tb_user）

表 4-10 用户表（tb_user）

编号	列名	类型	字段含义	备注
1	id	varchar	用户主键	
2	mobile	varchar	手机号码	
3	password	varchar	密码	
4	nick_name	varchar	昵称	
5	sex	bit	性别	
6	birthday	datetime	出生年月日	
7	avatar	varchar	头像	
8	email	varchar	邮箱	
9	reg_date	datetime	注册日期	
10	update_date	datetime	修改日期	
11	last_date	datetime	最后登录日期	
12	interest	varchar	兴趣爱好	
13	personality	varchar	个人简介	
14	fans_count	int	粉丝数	默认 0
15	follow_count	int	关注数	默认 0
16	is_delete	tinyint	删除标识	0 存在 1 删除
17	state	tinyint	用户状态	0 封禁 1 正常

管理员表（tb_admin）

表 4-11 管理员表（tb_admin）

编号	列名	类型	字段含义	备注
1	id	varchar	主键	
2	username	varchar	用户名	
3	password	varchar	密码	
4	state	tinyint	状态	
5	mobile	varchar	手机号	
6	reg_date	datetime	创建时间	6

续表 4-11

编号	列名	类型	字段含义	备注
7	last_date	datetime	最后登录时间	
8	sex	bit	性别	
9	avatar	varchar	头像	
10	last_position	varchar	上次登录地点	

workoverflow_gathering 活动数据库

活动表（tb_gathering）

表 4-12 活动表（tb_ gathering）

编号	列名	类型	字段含义	备注
1	id	varchar	主键	
2	name	varchar	活动标题	
3	summary	varchar	简介	
4	detail	text	详细说明	
5	sponsor	varchar	主办方	
6	image	varchar	宣传图片	
7	start_time	datetime	活动开始时间	
8	end_time	datetime	活动结束时间	
9	address	varchar	举办地点	
10	enroll_time	datetime	报名截止时间	
11	state	tinyint	用户可见标识	0:不可见 1:可见
12	city	varchar	举办城市	

4.3 动态模型

本系统使用动态模型中的活动图来进行一些功能的描述。

4.3.1 活动图

（1）用户登录注册活动图如图 4-13 所示。

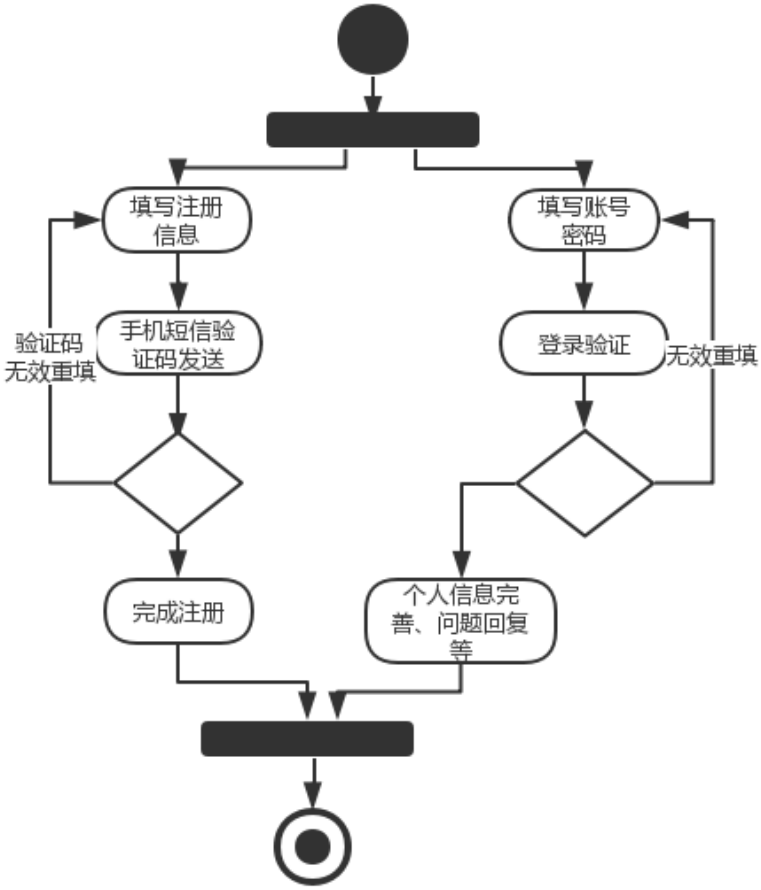

图 4-13 用户登录、注册活动图

（2）问题浏览活动图如图 4-14 所示。

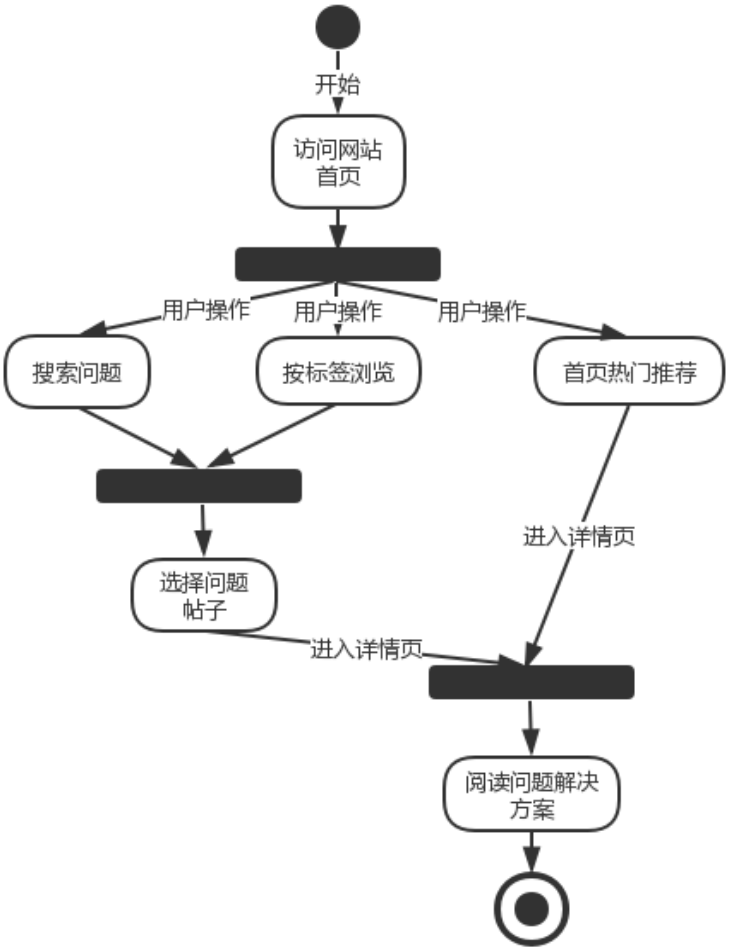

图 4-14 用户问题浏览活动图

第 5 章 系统实现

由于本系统采用前后端分离开发模式与微服务架构，所以系统的实现可以分为微服务的治理、用户前台、管理员后台，密码加密与微服务鉴权 JWT，容器部署与持续集成、容器管理与容器监控。

5.1 开发及运行环境

- 开发环境：mac 操作系统　　运行环境：Linux Centos7 操作系统
- 数据库：
  1、MySQL 5.7
  2、Redis 集群(三主三从)
  3、MongoDB
- 软件包：
  1、Java 8
  2、Spring Cloud Finchley.M9
  3、Spring Boot 2.0.1.RELEASE
- 测试包：Junit4
- 开发工具：
  1、Intellij IDEA
  2、Maven 包管理工具
  3、Git 版本控制工具
  4、Chrome 浏览器
  5、Vmware Fusion 虚拟机工具
  6、Docker 容器
- 第三方服务：
  1、 华为云服务器 ECS
  2、 阿里云容器服务

3、 阿里云短信服务

5.2 用户前台

用户登录注册页面如下图 5-1

图 5-1 用户登录注册页面

用户浏览问题模块首页如下图 5-2

图 5-2 问答首页

用户进入问题查看详情如下图 5-3

图 5-3 问答详情页面

5.3 管理员后台

管理员拥有对该系统所有资源操作的最高权限，下面对管理员的功能进行展示。管理员登陆功能如图 5-4 所示：

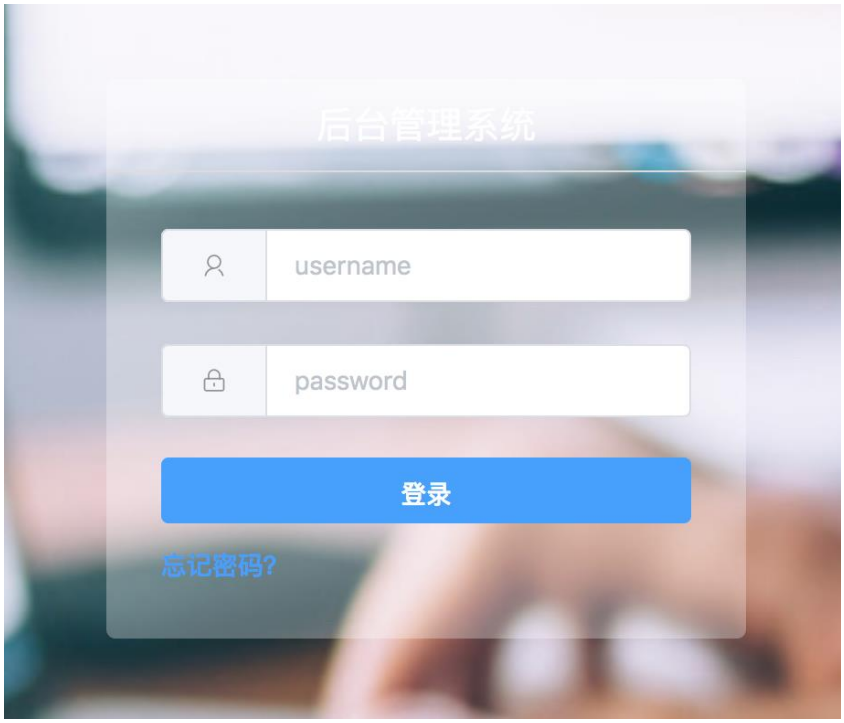

图 5-4 管理登录页面

管理界面首页如下图 5-5：

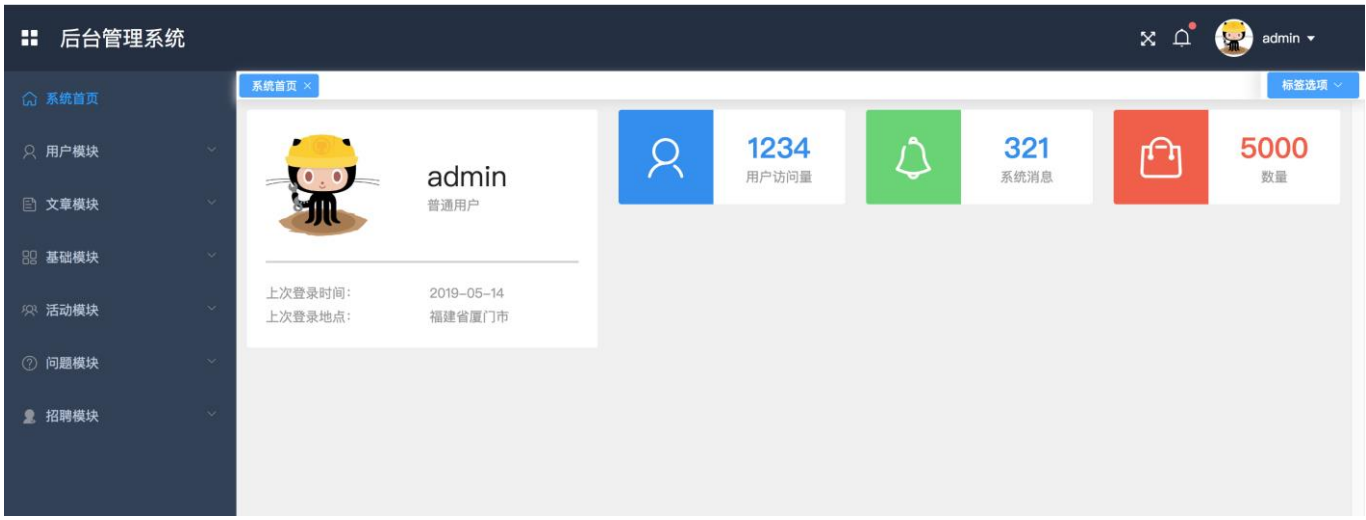

图 5-5 管理系统首页

管理员对用户的管理如下图 5-6：

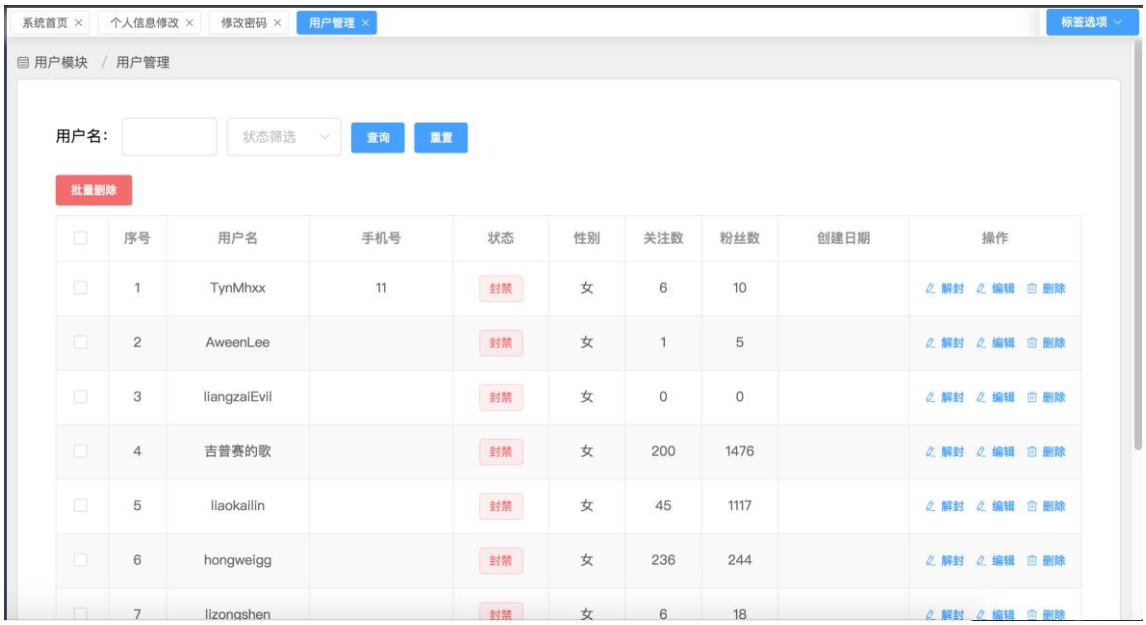

图 5-6 用户管理

管理员对文章的管理如下图 5-7：

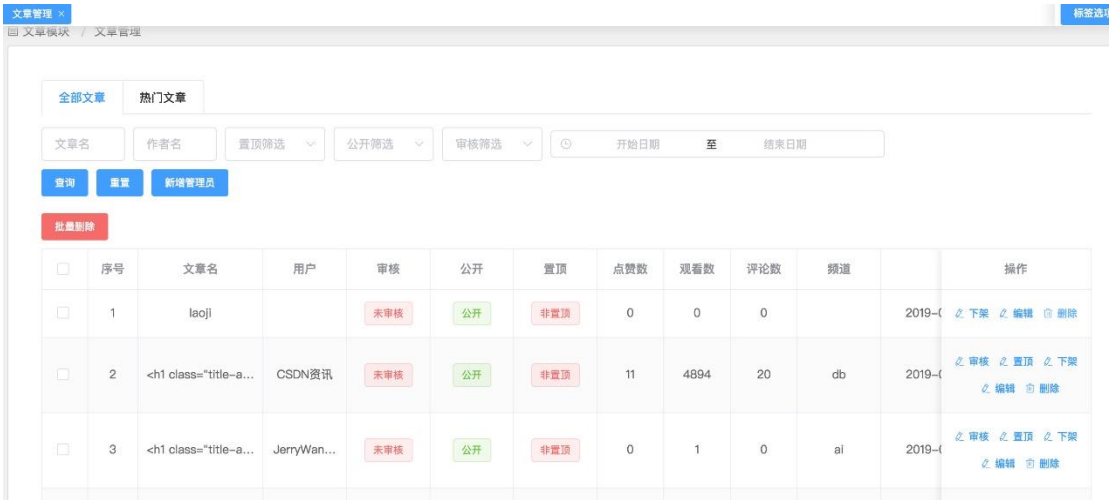

图 5-7 文章管理

管理员个人信息和密码修改如下图 5-8、5-9：

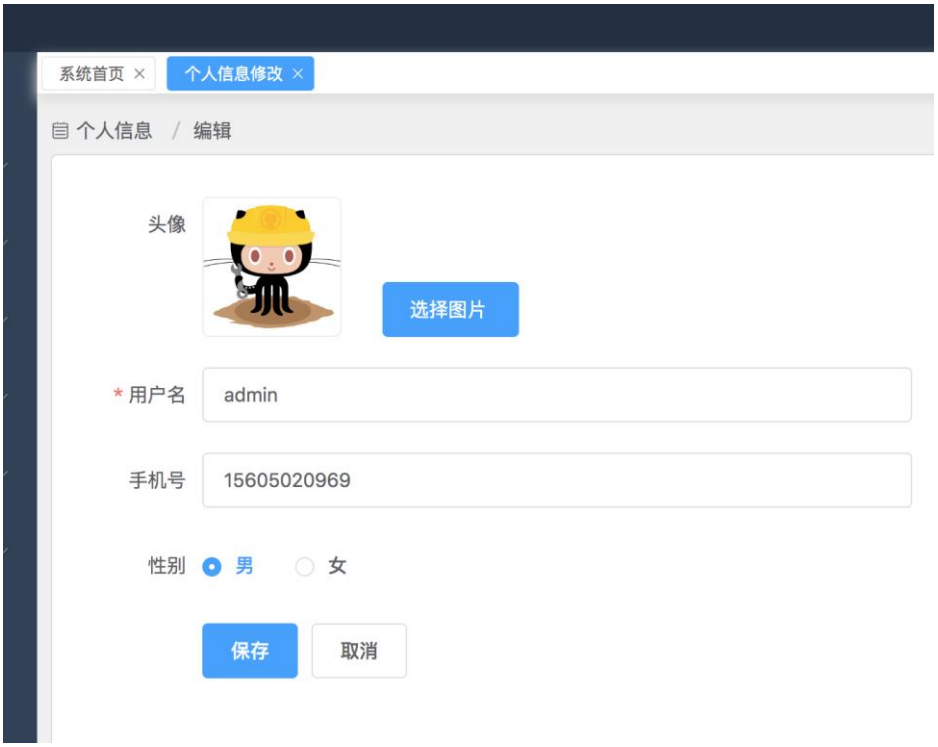

图 5-8 个人信息修改

图 5-9 修改密码

标签资源管理如下图 5-10：

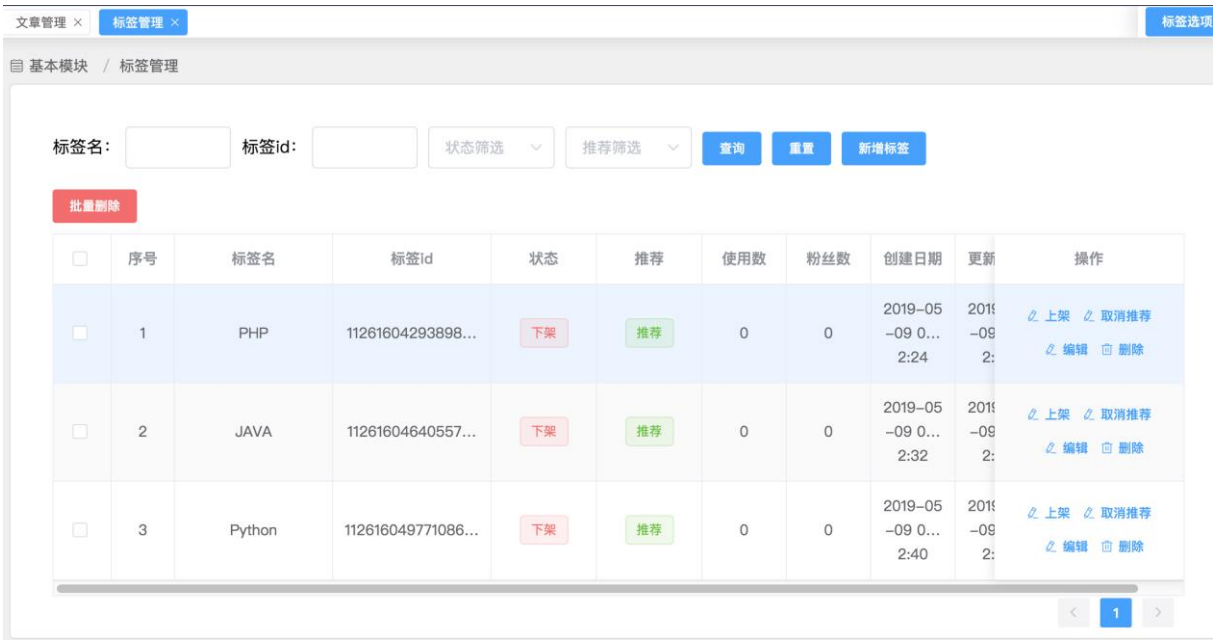

图 5-10 标签管理

5.4 密码加密与微服务鉴权 JWT

本系统考虑到了用户信息的安全性，所以对用户的密码进行哈希算法加密，使用 SHA 或者 MD5 这类哈希算法结合 salt(盐值)进行加密是一个不错的选择，但考虑到本系统使用了 Spring 系列的框架技术较多，所以在安全方面也采用 Spring Security 安全框架，它可以使用 BCrypt 强哈希方法来加密密码。BCrypt 强哈希方法使得每次加密的结果都不一样，非常安全，管理员密码加密后如下图 5-11 所示。

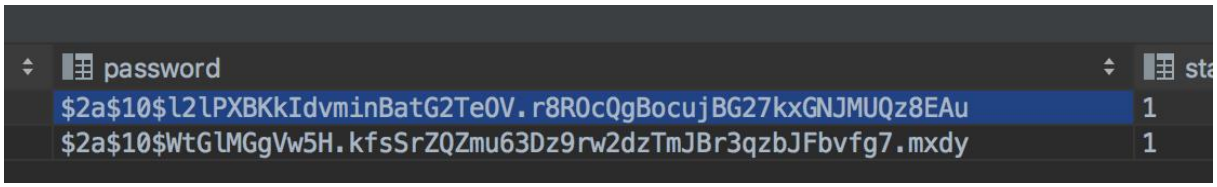

图 5-11 强哈希算法加密后的管理员密码

JWT 身份认证机制在用户登录成功后，服务端会返回一段 token 信息，将其保存在浏览器本地，下次请求时在 HTTP 头部带上这些信息即可正常进行身份的认证，服务端返回的 token 信息如下图 5-12 所示。

```
{
  "flag": true,
  "code": 20000,
  "message": "登陆成功",
  "data": {
    "token":
"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI5ODQzMjc1MDc4ODI5MzgzNjgiLCJzdWIiOiJ4aWF
vbWkiLCJpYXQiOjE1MjM1MjQxNTksInJvbGVzIjoiYWRtaW4iLCJleHAiOjE1MjM1MjQ1MTl9
._YF3oftRNTbq9WCD8Jg1tqcez3cSWoQiDIxMuPmp73o",
    "name":"admin"
  }
}
```


图 5-12 认证成功返回 JSON 结果信息

5.5 微服务治理

由于本系统拆分出来的服务较多，所以需要做好服务的统一管理与健康监控，这里我使用了 Spring Cloud 中的 Eureka 组件作为服务注册中心，通过创建了一个服务注册微服务，其他服务通过该服务进行注册，这样就可以在服务之间需要相互调用时，能够通过服务注册中心进行快速的服务发现和调用，其中的调用不是简单的 HTTP 远程调用，而是通过另一个 Spring Cloud 组件 Feign 通过注册时指定的微服务名进行服务调用，注册结果的 Eureka 前端界面如下图 5-13 所示。

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
WORKOVERFLOW-ACTIVITY	n/a (1)	(1)	UP (1) - bogon:workoverflow-activity:9005
WORKOVERFLOW-ARTICLE	n/a (1)	(1)	UP (1) - bogon:workoverflow-article:9004
WORKOVERFLOW-BASE	n/a (1)	(1)	UP (1) - bogon:workoverflow-base:9001
WORKOVERFLOW-COMPLAINT	n/a (1)	(1)	UP (1) - bogon:workoverflow-complaint:9006
WORKOVERFLOW-FRIEND	n/a (1)	(1)	UP (1) - bogon:workoverflow-friend:9010
WORKOVERFLOW-MANAGER	n/a (1)	(1)	UP (1) - ab9b2c5041c2:workoverflow-manager:9011
WORKOVERFLOW-MQ	n/a (1)	(1)	UP (1) - bogon:workoverflow-mq:9009
WORKOVERFLOW-QA	n/a (1)	(1)	UP (1) - bogon:workoverflow-qa:9003
WORKOVERFLOW-USER	n/a (1)	(1)	UP (1) - bogon:workoverflow-user:9008

图 5-13 服务注册中心 Eureka

5.6 容器部署与持续集成

本系统将所有的服务都部署到 Centos 系统里的 Docker 容器中，通过 docker ps 可

以查看到当前已经运行起来的项目如下图 5-14 所示。

```
[7:59:30] root:~ # docker ps
CONTAINER ID        IMAGE                                                  COMMAND                  CREATED             STATUS
 PORTS                    NAMES
156f708d5e6f        rancher/healthcheck:v0.3.8                             "/.r/r /rancher-en..."   38 hours ago        Up 38 hours
                         r-healthcheck-healthcheck-1-969a1a29
f2a73e7b135c        172.16.103.130:5000/workoverflow_manager:1.0-SNAPSHOT  "java -jar /app.jar"     39 hours ago        Up 39 hours
 0.0.0.0:9011->9011/tcp     docker_workoverflow_manager
8016c20a4d05        172.16.103.130:5000/workoverflow_qa:1.0-SNAPSHOT       "java -jar /workov..."   46 hours ago        Up 40 hours
 0.0.0.0:9003->9003/tcp     docker_workoverflow_qa
9f86fffc97c5        172.16.103.130:5000/workoverflow_article:1.0-SNAPSHOT  "java -jar /app.jar"     46 hours ago        Up 40 hours
 0.0.0.0:9004->9004/tcp     docker_workoverflow_article
32350f1851a0        172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT     "java -jar /app.jar"     46 hours ago        Up 40 hours
 0.0.0.0:9001->9001/tcp     docker_workoverflow_base
26e797f9e1c6        172.16.103.130:5000/workoverflow_spider:1.0-SNAPSHOT   "java -jar /app.jar"     46 hours ago        Up 40 hours
 0.0.0.0:9017->9017/tcp     docker_workoverflow_spider
9d7815eea971        172.16.103.130:5000/workoverflow_learn_cms:1.0-SNAPSHOT  "java -jar /app.jar"   46 hours ago        Up 40 hours
 0.0.0.0:31001->31001/tcp   docker_workoverflow_cms
aac3dab96b15        172.16.103.130:5000/workoverflow_web:1.0-SNAPSHOT      "java -jar /app.jar"     3 days ago          Up 40 hours
 0.0.0.0:9012->9012/tcp     r-workoverflow-workoverflow-web-1-b6a7be9d
5a97bd110515        rancher/net:v0.13.17                                   "/rancher-entrypoi..."   4 days ago          Up 40 hours
                         r-ipsec-ipsec-router-1-ba4de7e5
```

图 5-14 Docker 中运行容器列表

持续集成使用 Jenkins 工具进行实现，下图 5-15 为自动化任务构建项目并部署运行到测试服务器上的 Docker 容器中。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 20.368 s
[INFO] Finished at: 2019-04-22T03:45:04-07:00
[INFO] Final Memory: 59M/323M
[INFO] ------------------------------------------------------------------------
[JENKINS] Archiving /var/lib/jenkins/workspace/workoverflow_base/workoverflow_base/pom.xml to
com.workoverflow/workoverflow_base/1.0-SNAPSHOT/workoverflow_base-1.0-SNAPSHOT.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/workoverflow_base/workoverflow_base/target/app.jar to
com.workoverflow/workoverflow_base/1.0-SNAPSHOT/workoverflow_base-1.0-SNAPSHOT.jar
channel stopped
[workoverflow_base] $ /bin/sh -xe /tmp/jenkins7901571026239047899.sh
+ docker pull 172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT
1.0-SNAPSHOT: Pulling from workoverflow_base
8ba884070f61: Already exists
09a969e2c5ad: Already exists
e90644fb2ae8: Already exists
7f7da26ab605: Pulling fs layer
7f7da26ab605: Download complete
7f7da26ab605: Pull complete
Digest: sha256:00d70f720a959e2786e347408c17b16cf9b485eb58608693f997e88c1cd3118f
Status: Downloaded newer image for 172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT
+ docker run --rm -dt --name=docker_workoverflow_base -p 9001:9001 172.16.103.130:5000/workoverflow_base:1.0-SNAPSHOT
3cc14404c71537c4557c0736be962b2cf154ca95f979e867066aaa365c5f2c0f
Finished: SUCCESS
```

图 5-15 Jenkins 构建任务成功结果截图

5.7 容器管理

本系统使用开源的工具 racher 来对所有的服务进行管理，它内置了很多中容器编排方式，例如 Kubernetes、Docker Swarm，它能对主机和主机上的所有容器进行非常全面的管理，下图 5-16 与 5-17 为 rancher 对主机与服务容器的管理界面。

图 5-16 rancher 主机管理

图 5-17 容器的管理

5.8 ELK 实时日志系统

本系统使用 ELK 技术栈对爬虫模块的爬取记录进行实时的分析管理，能够直观的从

柱状图中看出服务的爬取速率，下图 5-18 为爬虫模块运行时的实时记录展示。

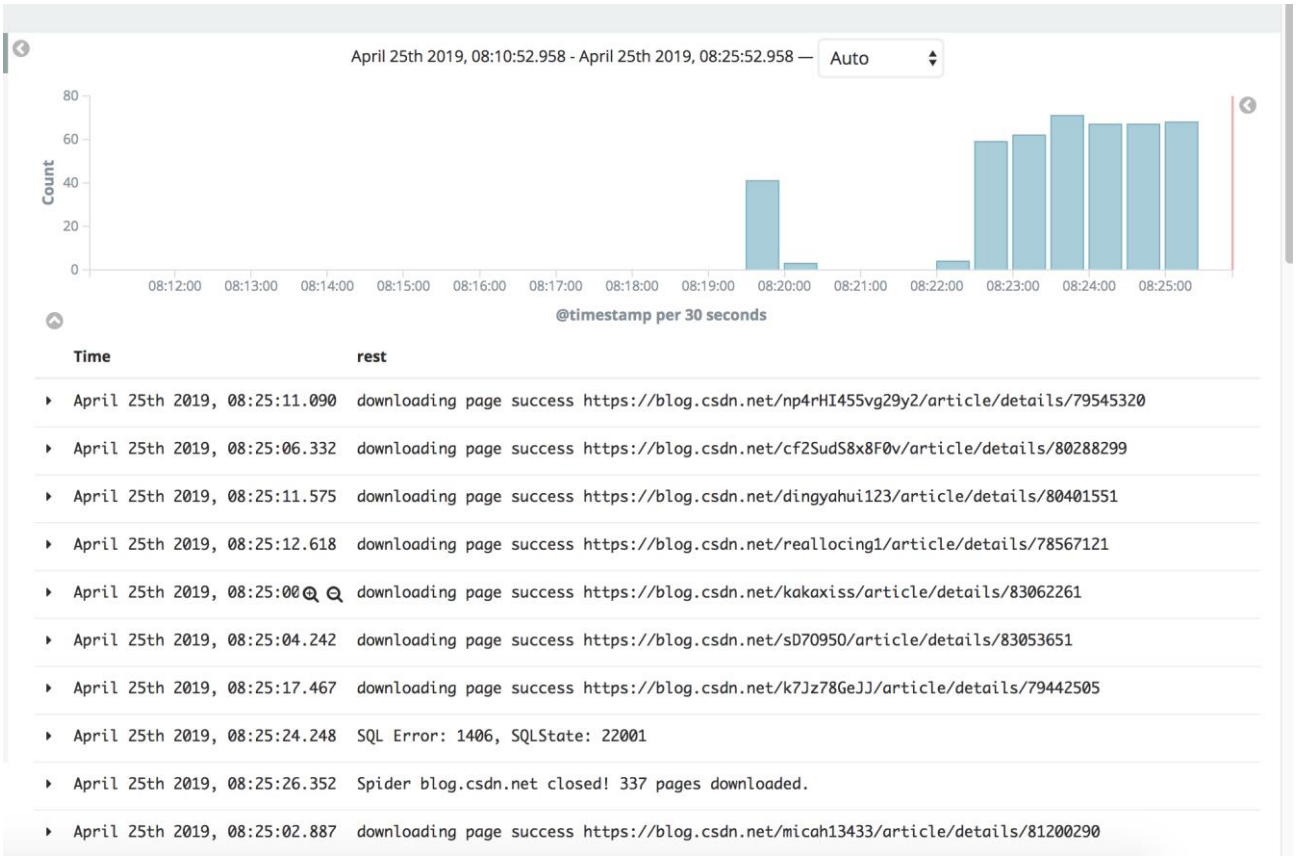

图 5-18 爬虫日志的分析

结 论

本系统使用了当前互联网企业应用开发中流行的微服务架构，并选择使用已经在各大互联网公司中落地成熟的技术体系 Spring Cloud。在传统的单体应用架构中，随着应用的业务变得复杂起来，项目的体积也跟着变大，导致了技术团队中的新人很难理解项目并加入开发流程中，在项目的打包编译上也浪费了很多时间。针对这些问题，国外技术专家提出了微服务的技术架构体系，经过查阅相关资料，我发现了微服务架构不仅解决了上述出现的问题，还针对现代分布式系统环境提出了许多优秀的解决方案，它把一个大型复杂的软件应用按照不同服务进行细粒度的划分，这样就可以分开部署在不同的服务器上，有效的利用了硬件资源，对单个小服务的维护难度和成本变得十分小，并且 Spring Cloud 中提供了很多丰富的组件供开发者自由选择，针对一种业务问题有多种不同的解决方案，这种“可插拔”的设计思想我认为是未来框架工具设计的标准。微服务架构的如此流行和容器化技术的成熟分不开关系，特别是 Docker 容器引擎，它使得微服务的打包、测试、以及部署都变得十分方便，直接推动了 DevOps 的发展，本系统也实现的一个小型的 DevOps，通过这次的实践使我对大型互联网企业中开发、测试、运维整个流程有了一定的认识。本系统根据当前互联网信息的繁杂性与程序员寻求技术问题困难的出发点，实现了一个供程序员发布问题、解决问题、查看优质文章的技术社交平台，由于该课题主要是对微服务架构以及互联网上流行的技术栈进行学习并使用，所以在网站的业务功能上实现的不是很严瑾，功能也不够完善，用户的体验感不是很好，所以希望在今后的工作中对其进行业务功能层面上的优化，并且能根据新出现的业务选用更适合其的技术解决方案，达到持续学习的目的。

致 谢

毕业设计和论文的顺利完成，首先我要感谢我的毕设指导老师王俊玲，正是因为王老师对毕业设计的耐心指导，让我在毕业设计实现的过程中，少走了很多的弯路。感谢微服务架构思想的提出者以及各种优秀开源框架工具的开发者，正是有了你们无私的开源贡献，让互联网技术变得百花齐放，使我学到了很多书本上学不到的知识，让我的开发能力有了明显的进步，在我的就业方面也提供了巨大的帮助，感谢你们。感谢所有期刊、文献、书籍的作者，你们提供的优秀资料让我在论文的写作中得心应手，感谢你们。我还要感谢大学四年所有的老师和同学，正是有了你们辛勤的教育和陪伴，让我的大学生活多姿多彩，个人综合能力得到了很大的提升，感谢你们。最后我要感谢集美大学给我提供了这么优质的学习平台，让我度过了快乐充实的四年。

参考文献

[1] 周立. Spring Cloud 与 Docker 微服务架构实战[M]. 电子工业出版社, 2017.

[2] 王骏翔,郭磊.基于 Kubernetes 和 Docker 技术的企业级容器云平台解决方案[J].上海船舶运输科学研究所学报,2018,41(03)：51-57.

[3] 徐江生. 容器云平台的设计与实现[D].北京邮电大学,2017.

[4] 梁兴波.FreeMarker 模板引擎在 Java 开发中的应用[J].硅谷,2013,6(21)：46+7.

[5] Adam Freeman. Using RESTful Web Services[M].Apress：2018-09-11.

[6] Peter A. Carter. Understanding JSON[M].Apress：2018-08-28.

[7] 杜艳美,黄晓芳.面向企业级 web 应用的前后端分离开发模式及实践[J].西南科技大学学报,2018,33(02)：83-87.

[8] 王鹤琴,朱珍元.基于 MVVM 模式的 Web 开发研究[J].菏泽学院学报,2019(02)：7-13.

[9] 王巍.云计算之虚拟化平台技术认知研究[J].通信管理与技术,2014(01)：17-18+23.

[10] 刘景云.浅析 Docker 虚拟化技术[J].网络安全和信息化,2019(01)：73-81.

[11] 高攀攀,王健,黄颖,何克清.互联网上基于 SOAP 和 REST 的 Web 服务的对比分析[J].小型微型计算机系统,2015,36(11)：2417-2421.

[12] 戴伟,马明栋,王得玉.基于 Nginx 的负载均衡技术研究与优化[J].计算机技术与发展,2019,29(03)：77-80.

[13] GOGOE VIDJINNAGNI PAUL. 基于 JSON Web 签名的 Web 安全研究[D].武汉理工大学,2015.

[14] 曾恒. 基于 ELK 的网络安全日志管理分析系统的设计与实现[D].北京邮电大学,2017.

[15] 潘向东,杨建梅,白桦.开源社区雪崩效应实证研究：以 Sourceforge 为例[J].复杂系统与复杂性科学,2015,12(04)：61-70.